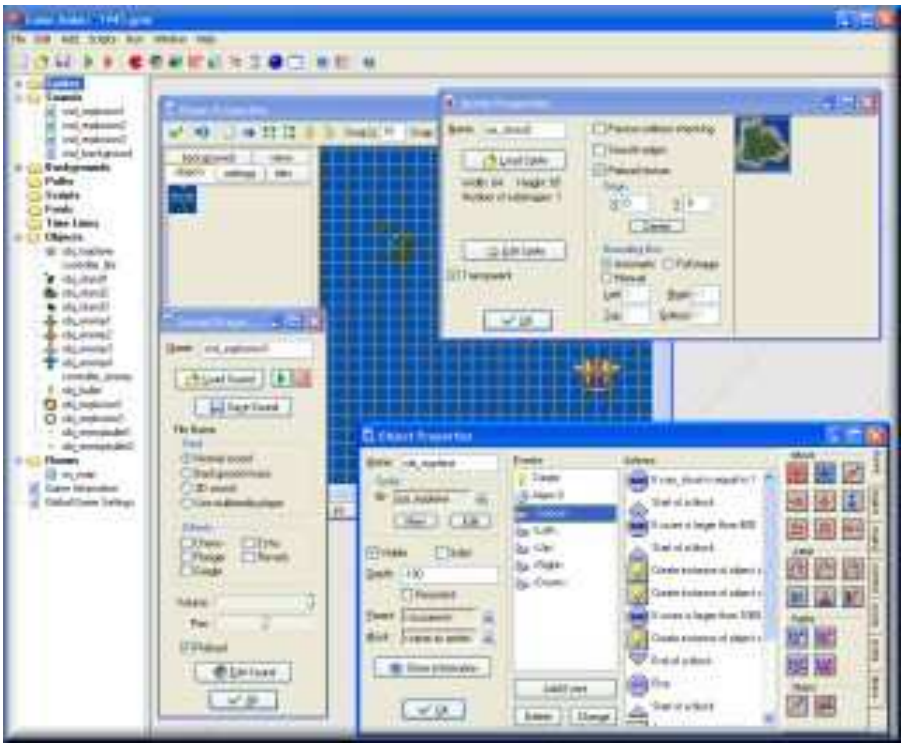
Création de jeux avec Game Maker

Version 6.1

Développé par Mark Overmars

Traduction française par Philippe Ragni (2005)



La documentation de *Game Maker* est composée de quatre parties:

Utilisation de Game Maker

Cette section décrit l'utilisation de base de *Game Maker*. Elle énonce l'idée générale du programme et indique comment ajouter des sprites, un arrière-plan (background), des sons et la manière de créer des objets avec événements et actions ainsi que la façon de les ajouter dans des salles (rooms).

Les thèmes suivants sont décrits dans cette section :

Utilisation avancée

Cette section décrit les aspects les plus avancés de *Game Maker*. Cela comprend les chemins (paths), les fontes de caractères (fonts), les lignes de temps, les scripts et les techniques pour créer des rooms avec tuiles (tiled rooms) et l'utilisation de vues (views) dans les rooms.

Les thèmes suivants sont décrits dans cette section :

La Finalisation de votre jeu

Cette section indique comment finaliser votre projet pour obtenir un produit fini. Elle décrit la manière d'ajouter de l'aide dans votre jeu, comment paramétrer les diverses options de ce dernier et la façon de créer des jeux autonomes pouvant être distribués à d'autres personnes ne possédant pas *Game Maker*.

Les thèmes suivants sont décrits dans cette section :

Le Langage de Game Maker

Game Maker comprend un langage de programmation intégré. Ce langage de programmation vous apporte beaucoup plus de flexibilité et de contrôle que le permettent les actions standards. Ce langage sera désigné dans cette documentation sous le nom de **GML** (***G**ame **M**aker **L**anguage*). Dans cette section, nous décrirons le langage GML et donnerons un aperçu de toutes les fonctions (1000 environ) et variables disponibles, destinées à contrôler tous les aspects de votre jeu.

Les thèmes suivants sont décrits dans cette section :

Quoi de Neuf

La version 6.1 de *Game Maker* est une mise à jour mineure de la version 6.0. Les changements suivants ont été réalisés :

Incompatibilités

Les principales incompatibilités concernaient le système de particules. Certaines fonctions ont été supprimées et la taille des formes de sprites intégrés présente un effet différent. De plus, quelques formes de sprites intégrés ont été modifiées. Voir ci-dessous pour plus d'informations.

Une petite incompatibilité concernait la façon dont une vue (view) suivait une instance. Comme le sprite n'est plus pris en compte dorénavant, une bordure légèrement plus large sera nécessaire.

Les fichiers sont compatibles de manière ascendante et descendante. Autrement dit, la version 6.1 peut lire les fichiers créés avec la version 6.0 et la version 6.0 peut lire les fichiers créés avec la version 6.1. Bien entendu, cela fonctionnera si aucune des fonctions de la nouvelle version 6.1 n'est utilisée.

Effets

Un mécanisme très simple à utiliser pour faire des effets et des explosions a été implémenté. C'est simplement une action que vous pouvez utiliser dans n'importe lequel événement. Il créera des effets comme une explosion, de la fumée, des feux d'artifice, de la pluie, ou encore de la neige. Il y a douze types d'effets différents possibles, disponibles en trois tailles et avec la couleur de votre choix. Vous pouvez également générer ces effets en utilisant un simple appel de fonction.

Système de particules

Le système de particules a été considérablement amélioré, mais aboutit aussi à certaines incompatibilités avec le système employé dans le passé. Le changement principal a porté sur le fait que maintenant les systèmes de particules sont automatiquement mis à jour et dessinés. Il n'est plus désormais nécessaire de créer un objet pour cela. Les précédentes fonctions ne sont plus disponibles et doivent être supprimées de votre code. De plus, l'action pour créer un type de particules a aussi changé.

- Les systèmes de particules ont maintenant une profondeur et une position qui indiquent à quelle profondeur et à quelle position ils doivent être dessinés.
- Il y également davantage de types de particules intégrés, notamment des cercles, de la fumée, des nuages et des explosions. Il y a plus de possibilités pour les couleurs, les particules peuvent avoir une orientation (changeante au besoin), leur ratio d'aspect peut être réglé et un mélange additif est possible.
- Le dosage aléatoire de la taille, de la vitesse et de la direction, ajoutée à chaque étape, a été remplacée par une quantité de 'wiggling', essentiellement pour des effets plus intéressants, comme par exemple des étoiles scintillantes.
- Les possibilités des systèmes de particules utilisant des actions 'drag-and-drop' ont été augmentées, en tenant compte des particules de sprites et en augmentant les possibilités des couleurs et des paramètres alpha.
- La démonstration sur les particules a été supprimée de la distribution mais une intéressante démonstration avec sources est disponible sur le site web.

Editeur d'images

Un certain nombre d'améliorations ont été apportées dans l'éditeur d'images. La possibilité de créer, de déplacer et de copier des sélections a été ajoutée. Le mécanisme d'ajout de textes a été amélioré, rendant possible le déplacement des textes. Vous pouvez plus facilement dessiner des lignes horizontales, verticales et diagonales, ainsi que des carrés et des cercles. Vous pouvez utiliser une touche pour choisir une couleur du dessin de l'image courante. Il y a aussi des commandes pour créer un contour d'image et pour inverser les couleurs. Egalement, un meilleur marqueur d'images a été incorporé.

Gestion des mouvements

Des fonctions de gestion de mouvements de champs peuvent maintenant éviter des instances d'un type particulier. En utilisant la notion de parents, cela peut rendre encore plus flexible les possibilités de gestion de mouvements.

Modèles graphiques 3D

Un nouveau jeu de fonctions a été ajouté afin de pouvoir créer, dessiner, sauvegarder et charger des modèles 3D. Ces fonctions s'avèrent plutôt limitées en possibilité mais peuvent accroître considérablement la vitesse des graphiques 3D.

Dessin de surfaces

Plutôt que de dessiner sur l'écran, il est désormais possible de définir des surfaces pour y dessiner dessus. Ces surfaces peuvent être placées à l'écran ou utilisées comme textures. Les surfaces peuvent être partiellement transparentes à l'aide des valeurs alpha. On peut les sauvegarder dans un fichier. Il est aussi possible de les transformer en sprites ou en décors (backgrounds).

Autres changements

Il y a un certain nombre d'autres changements et additifs. Voici les plus importants d'entre eux.

- Un installeur plus agréable est maintenant utilisé.
- Une action a été ajoutée pour empaqueter une instance autour de l'écran.
- Une action a été ajoutée pour créer une instance aléatoire en fonction de quatre choix possibles.
- Une nouvelle structure de données en grille a été incorporée.
- Le nombre d'événements de type alarme a été porté à 12 et le nombre d'événements définis par l'utilisateur a été fixé à 16.
- Ajout des fonctions **choose**(val1,val2,...) et **median**(val1,val2,...).
- Les blocs d'actions de l'objet ou de la ligne de temps sont désormais indentés.
- Il y a davantage d'options de placement lors de l'ajout d'images dans les sprites.
- Ajout des événements de gestion de la molette de la souris ('mouse wheel up and mouse wheel down events').
- Il est possible maintenant d'utiliser des nombres hexadécimaux commençant par un $, ex: $0000FF est rouge.
- Les réglages de synchronisation dans les options de jeu devraient mieux fonctionner désormais.
- Les moments des lignes de temps peuvent maintenant être dupliqués.
- Ajout d'une fonction **screen_wait_vsync()** qui attend un rafraîchissement vertical du moniteur informatique.
- Vous pouvez maintenir la touche <Shift> lors de la sélection de tuiles (tiles) pour en choisir plusieurs ou presser <Ctrl> pour en sélectionner plusieurs de la taille de la grille de la room.
- ...

Bogues corrigés

Les principaux bogues suivants ont été corrigés.

- Les fonctions de collision fonctionnent désormais correctement.
- Lors de la création de rooms (salles ou pièces) durant le jeu, la vitesse est maintenant par défaut réglée à 30.
- Le buffer est effacé lors du démarrage du jeu ou lorsque le mode graphique est changé.
- Bogue corrigé dans la fonction **d3d_vertex_normal_texture_color()**.
- Bogue corrigé lors de la création d'un sprite ou d'un arrière-plan (background) à l'écran.
- Les variables globales sont maintenant correctement affichées en mode déboguage.
- Bogue corrigé lors de l'ajout de moments dans une ligne de temps vide pendant le jeu.
- Bogue corrigé dans le réglage de l'origine dans la fonction **replace_sprite()**.
- Une vue (view) peut désormais suivre une instance ne possédant pas de sprite.
- Un certain nombre de problèmes concernant les sons ont été corrigés.
- Les valeurs réelles dans les fichiers INI sont désormais correctement lues avec les séparateurs ',' et '.'.
- Correction d'erreurs dans le fichier d'aide.
- L'action et la fonction pour sauter à une position aléatoire (Jump to a random position action and function) ne sautent plus en dehors de la room comme c'était parfois le cas auparavant.
- Les événements sur touches (key events) réagissent maintenant à des touches comme ~ . , [,etc.
- Bogue résolu concernant les destructeurs et commutateurs de particules de forme elliptique.
- Bogue corrigé qui parfois survenait quand l'on déplaçait (dragging) des actions dans la liste d'actions.
- Lors de la fusion de jeux (merging games), l'objet que doit suivre la vue, demeure désormais correct.
- Le volume par défaut du son est toujours fixé désormais au maximum.
- Ai changé la façon qu'une vue suit un objet (uniquement la position et non plus la prise en compte du sprite). Cela évite beaucoup de problèmes mais peut demander des réglages de bordure légèrement plus grosse.
- Une vitesse de chemin (path speed) peut maintenant être effectivement négative.
- Correction de certaines erreurs de la barre de scrolling dans l'éditeur de rooms (room editor).
- ...

Utilisation de Game Maker

Game Maker est un programme simple d'utilisation pour la création de vos propres jeux sur ordinateur. Cette section de ce fichier d'aide vous donne toutes les informations nécessaires pour créer vos premiers jeux. Les sections suivantes aborderont des thèmes plus avancés, comme par exemple la finalisation et la distribution de votre jeu et le langage intégré de programmation GML.

Des informations sur l'utilisation de base de *Game Maker* peuvent être trouvées dans les pages suivantes :

Ainsi vous souhaitez créer vos propres jeux sur ordinateur

Jouer à des jeux sur ordinateur est amusant. Mais ce qui est actuellement encore plus amusant est de pouvoir créer vos propres jeux sur ordinateur et laisser d'autres personnes y jouer. Malheureusement, la création de jeux sur ordinateur n'est pas si facile. Les jeux commerciaux sur ordinateur que l'on peut acheter aujourd'hui, présentent une durée de développement pouvant s'étendre sur une à trois années, avec des équipes composées de 10 à 50 personnes. Les budgets peuvent facilement atteindre des millions de dollars (donc des millions d'euros). Et toutes ces personnes ont beaucoup d'expérience : programmeurs, graphistes, techniciens du son, etc.

Cela signifierait donc qu'il est impossible de créer vos propres jeux sur ordinateur ? Heureusement non. Bien sûr, ne vous attendez pas à créer un jeu comme *Quake* ou *Age of Empires* en quelques semaines. Mais cela n'est pas nécessaire et même souhaitable. Des jeux plus simples, comme *Tetris*, *Pacman*, *Space Invaders*, etc. sont aussi amusants à y jouer et beaucoup plus faciles à créer. Malheureusement, cela nécessite un bon niveau de programmation pour concevoir les graphiques, les sons, l'interaction entre le joueur et bien sûr le jeu, etc.

Mais voici *Game Maker* qui a été écrit dans le but de rendre beaucoup plus facile la création de tels jeux. Il n'est pas nécessaire de savoir programmer. Une interface intuitive et employant le 'drag-and-drop' (sélectionner et déplacer) vous permet de créer vos propres jeux très rapidement. Vous pouvez importer et créer des images, des sprites (des images animées) ainsi que des sons pour les utiliser ensuite dans vos propres créations. Vous pourrez facilement définir les objets de votre jeu et indiquer leur comportement, vous pourrez aussi définir des rooms (salles) attrayantes avec des décors présentant des scrollings et dans lesquels le jeu prendra vie. Et si vous désirez un contrôle total, vous aurez la possibilité d'utiliser un langage de programmation très facile d'emploi , intégré à *Game Maker* et qui vous donnera le plein contrôle du déroulement de votre jeu.

Game Maker est spécialisé pour les jeux en deux dimensions. Cela ne signifie pas que l'on ne puisse pas créer de mondes en 3D comme *Quake*, bien que les fonctions graphiques 3D présentent certaines limitations. Mais que cela ne vous décourage pas. La plupart des grands jeux, comme *Age of Empires*, la série des *Command &ampConquer* et *Diablo* utilisent la technologie des sprites en deux dimensions, malgré qu'ils semblent apparaître en dimension 3D. Et le développement de jeux en deux dimensions est beaucoup plus rapide et facile à réaliser.

Game Maker existe à la fois en version gratuite et enregistrée. La version gratuite (tout comme la version enregistrée bien entendu) peut être utilisée libre de droits. Vous pourrez librement

distribuer les jeux que vous aurez créés avec et pourrez même les vendre si vous le souhaitez. Pour plus de détails, veuillez lire la licence contractuelle jointe à *Game Maker*.

Je vous encourage fortement à vous enregistrer pour obtenir une copie pleinement fonctionnelle de *Game Maker*. Cela débloquera un certain nombre de fonctions supplémentaires du logiciel et supprimera le logo lors de l'exécution des jeux. Mais surtout, vous contribuerez ainsi à soutenir les développements futurs de *Game Maker*.

Ce document vous enseignera tout ce que vous devez savoir sur *Game Maker* et la façon de réaliser vos propres jeux avec le logiciel. Cependant, veuillez comprendre que même à l'aide d'un programme comme *Game Maker*, le développement de jeux sur ordinateur vous demandera quelques efforts. Il y a beaucoup trop d'aspects importants : le gameplay, les graphiques, les sons, l'interaction entre le jeu et l'utilisateur, etc. Commencez avec des exemples simples et donc faciles à créer. Vous réaliserez ainsi que la création de jeux peut être un loisir très amusant. Pensez également à visiter le site web.

http://www.gamemaker.nl/

pour obtenir de nombreux exemples, tutoriaux, idées et des liens sur d'autres sites et forums. Et très bientôt, vous deviendrez vous aussi un expert dans la conception de jeux sur ordinateur. Alors, amusez-vous bien avec *Game Maker* .

Installation

Vous avez déjà probablement effectué l'installation du logiciel mais dans le cas contraire, voici comment installer *Game Maker*. Pour ce faire, lancez le programme "`gmaker.exe`". Suivez les instructions à l'écran. Vous pouvez installer le programme où vous le souhaitez mais il est préférable de conserver les options proposées par défaut. Une fois l'installation terminée, vous pourrez trouver dans le menu 'Démarrer' un nouveau groupe de programmes où vous pourrez lancer *Game Maker* et lire le fichier d'aide.

Il vous sera demandé la première fois que vous lancerez *Game Maker* si vous désirez lancer le programme en mode **Simple** ou **Avancé**. Si vous n'avez jamais auparavant utilisé un programme de création de jeux et que vous n'êtes pas un programmeur expérimenté, je vous conseille d'utiliser le mode simple (aussi choisissez **No**). Dans le mode simple, il y a moins d'options affichées. Vous pourrez ultérieurement basculer dans le mode avancé en utilisant l'article approprié dans le menu **File**.

Dans le répertoire d'installation (par défaut "`C:\Program Files\Game_Maker6\`"), il y a un certain nombre d'autres répertoires :

- **`examples`** : contient des exemples de jeux que vous pourrez examiner et/ou modifier.
- **`lib`** : contient des librairies d'actions. Si vous souhaitez installer des librairies d'actions supplémentaires, vous devrez les mettre dans ce répertoire.
- **`sprites`** : ce dossier est utilisé pour contenir les sprites à utiliser. L'installation par défaut installe juste quelques sprites, mais sur le site web de *Game Maker* (http://www.gamemaker.nl/), vous pourrez charger un certain nombre de ressources qui contiennent des sprites supplémentaires, des sons, des décors (backgrounds), etc.
- **`backgrounds, sounds`** : des répertoires similaires qui contiennent des images de fonds d'écran et des sons.

Système nécessaire pour utiliser Game Maker

Game Maker nécessite pour fonctionner un PC moderne de type Pentium tournant sous Windows 98SE, 2000, Me, XP, ou supérieur. Une carte graphique compatible DirectX 8 avec au moins 16 Mo de mémoire est nécessaire pour créer la plupart des jeux. Il faut également une résolution d'écran d'au moins 800x600 et 65000 couleurs (16-bit). Vous devrez également disposer d'une carte son compatible DirectX 8. DirectX version 8.0 ou supérieur doit être installé sur votre ordinateur (vous pouvez télécharger la dernière version de DirectX sur le site web de Microsoft à l'adresse : http://www.microsoft.com/windows/directx/). Lors de la conception et du test de jeux, le besoin en mémoire est assez élevé (au moins 64 Mo. Il est préférable d'en avoir davantage, ceci dépendant principalement du système d'exploitation utilisé). Quand vous lancez des jeux pour y jouer, les besoins en mémoire sont moins importants et dépendent beaucoup du type de jeu.

Enregistrement

Game Maker est gratuit et peut être utilisé libre de droits. La version non enregistrée, cependant, présente quelques limitations et affiche un petit logo lors de l'exécution des jeux. Pour obtenir les fonctionnalités supplémentaires, enlever le logo et apporter son soutien aux développements futurs du logiciel, il est fortement recommandé d'enregistrer votre copie de *Game Maker*.
L'enregistrement ajoutera les fonctionnalités suivantes :

- Plus d'affichage du logo *Game Maker* lors du lancement du jeu.
- Sprites tournants, avec changement de couleurs et transparents.
- Actions supplémentaires comme par exemple jouer de la musique d'un CD, textes en rotation et formes colorées.
- Effets sonores spéciaux et sons de type spatial.
- Un certain nombre de fonctions avancées de dessin, comme les polygones texturés.
- Un système de particules pour créer des feux d'artifice, des flammes, de la pluie et d'autres effets.
- Fonctions graphiques 3D.
- La possibilité de faire des jeux multi-joueurs jouables en réseau.
- Des fonctions pour créer et modifier des ressources (sprites, arrière-plans (backgrounds), etc.) durant le jeu.
- Une collection de fonctions pour créer, gérer et utiliser des structures de données.
- Des fonctions pour la gestion des déplacements.
- La possibilité d'étendre *Game Maker* en utilisant des librairies DLL.

Le prix d'enregistrement de *Game Maker* est seulement de 15 Euros ou l'équivalent en autres monnaies (ex: 20 $ US). Il y a plusieurs manières d'enregistrer votre copie du programme. La façon la plus simple est d'utiliser l'enregistrement en ligne qui utilise un système de paiement sécurisé par carte de crédit ou encore un compte PayPal. Vous pouvez aussi transférer de l'argent vers notre compte bancaire, nous envoyer de l'argent ou encore utiliser un mandat bancaire. Les détails sont sur le site web d'enregistrement de *Game Maker* :

http://www.gamemaker.nl/registration.html

Pour enregistrer votre copie de *Game Maker* utilisez le site web indiqué ci-dessus ou choisissez **Registration** (Enregistrement) à partir du **menu d'aide**. A la gauche du formulaire qui s'affiche, cliquez sur le bouton **Go to Registration Webpage** (Aller sur la page d'enregistrement).

Vous serez alors redirigé sur la page de notre site web où les différentes options d'enregistrement sont indiquées, y compris l'enregistrement en ligne.

Une fois que nous aurons reçu votre enregistrement, vous recevrez un message électronique (email) avec le nom et la clé à utiliser ainsi que la façon d'entrer la clé dans le programme. Pour entrer la clé, sélectionnez une nouvelle fois **Registration** (Enregistrement) du **menu d'aide**. A la gauche du formulaire, pressez le bouton **Enter a Registration Key** (Entrer la clé d'enregistrement). Tapez le nom et la clé puis pressez **OK**. Si vous n'avez pas fait d'erreur, le programme sera alors enregistré.

Si vous avez déjà une version enregistrée de la version 5 de *Game Maker* installée sur votre machine, vous pouvez échanger votre clé d'enregistrement de la version 5 contre une clé d'enregistrement de la version 6. A la fin, choisissez **Registration** (Enregistrement) à partir du **menu d'aide**. A la gauche du formulaire, un bouton **Convert a Version 5 Key** (convertir une clé de la version 5) devrait apparaître dans ce cas. (Sinon, cela signifie qu'il n'y a pas d'enregistrement valide de la version 5 que vous possédez ou que votre version 6 est déjà enregistrée). Il y a également un texte expliquant comment convertir les clés. Lisez tout ceci avec attention et suivez les instructions puis pressez le bouton.

L'idée générale du programme

Avant de décrire les possibilités de *Game Maker* il serait bon de parler en premier lieu de l'idée générale du fonctionnement du logiciel. Les jeux créés avec *Game Maker* se déroulent dans une ou plusieurs *rooms* (salles). Les rooms sont à 1 seule dimension mais peuvent contenir des graphiques avec un aspect 3D. Dans ces rooms, vous pouvez placer des *objects*, que vous définissez vous-même dans le programme. Par exemple, des murs (walls), des balles animées, le personnage principal, des monstres, etc. Certains objets, comme les murs (walls), sont juste placés là mais ne font rien d'autre dans le jeu. D'autres objets, tel le personnage principal, se déplacent et réagissent aux actions de l'utilisateur (clavier, souris et joystick) mais également entre eux. Par exemple, lorsque le personnage principal rencontrera un monstre, il pourra mourir. Les objets sont les ingrédients les plus importants dans les jeux développés avec *Game Maker*. Aussi laissez-moi vous en dire plus à leur sujet.

Première chose, la plupart des objets nécessitent des images afin de rendre ces derniers visibles à l'écran. On appelle ces images des *sprites*. La plupart du temps, un sprite n'est pas uniquement une simple image mais un ensemble d'images affichées l'une après l'autre pour ainsi créer une animation. De cette façon, le personnage peut marcher, une balle rouler, un vaisseau exploser, etc. Durant le jeu, le sprite d'un objet donné peut changer (le personnage peut ainsi paraître différent lorsqu'il marche vers la gauche ou vers la droite). Vous avez la possibilité de créer vos propres sprites dans *Game Maker* ou les charger à partir de fichiers (ex: images GIF animées).

Certaines choses peuvent survenir aux objets. C'est ce que l'on appelle des *événements* (events). Les objets peuvent exécuter certaines *actions* lorsque des événements surviennent. Il existe un grand nombre d'événements différents mais également un nombre important d'actions différentes que ces objets peuvent effectuer. Par exemple, Il y a un *événement création* (creation event) lorsque l'objet est créé (pour être plus précis, lorsqu'une instance de l'objet est créée car il peut y avoir de multiples instances d'un même objet). Par exemple : lorsqu'une balle est créée, vous pouvez lui affecter certaines actions de mouvement (motion action). Ainsi, elle commencera à bouger. Lorsque deux objets entrent en collision, un *événement de collision* (collision event) sera généré. Dans ce cas, vous pourrez décider de stopper ou d'inverser le sens de direction de la balle. Il vous est aussi possible de jouer des sons avec effets ou non. Dans ce but, *Game Maker* vous permet de définir des *sons*. Quand le joueur presse une touche du clavier, un *événement clavier* (keyboard event) est créé. L'objet peut ainsi effectuer une action appropriée, comme se déplacer dans la direction indiquée. Nous espérons que vous avez compris l'idée générale exprimée ci-dessus. Pour chaque objet créé, vous pouvez donc indiquer des actions correspondant à divers événements. De cette façon, vous définissez le comportement de l'objet.

Une fois vos objets définis, il sera alors temps de créer des *rooms* (salles) dans lesquelles vos objets vont vivre. On peut utiliser les rooms comme niveaux de jeu ou pour décrire différents décors de jeu. Il existe des actions pour aller d'une room à une autre. Premièrement, les rooms ont un arrière-plan ou décor *(background)*. Cela peut être une couleur unie ou encore une image. Des images du décor (background images) peuvent être créées avec *Game Maker* ou chargées à partir de fichiers [le décor (background) ne fait pas grand chose en lui-même mais pour l'instant, retenez que le décor est destiné à embellir les rooms]. Enfin, vous pouvez placer des objets dans la room. Vous pouvez avoir de multiples instances du même objet dans une room. Ainsi, par exemple, vous ne définirez qu'un seul objet **mur** et pourrez l'utiliser à différents endroits. De la même manière, vous aurez de multiples instances d'un même objet **monstre**, dans la mesure cependant où les monstres présentent le même comportement.

Nous sommes maintenant prêts à lancer le jeu. La première room sera affichée et des objets prendront vie à cause des actions définies dans leurs événements de création. Ils interagiront entre eux en raison des événements de collision et réagiront aux actions de l'utilisateur en employant des événements clavier et/ou souris.

En résumé, les choses suivantes [appelées souvent ressources (resources)] jouent un rôle très important :

- *les objets*: qui constituent les vraies entités du jeu
- *les rooms* (salles ou pièces) : ce sont les endroits (niveaux de jeu) où vos objets vont prendre vie
- *les sprites* : les images (animées) qui représentent les objets
- *les sons* : on peut les utiliser dans les jeux, soit comme musiques d'arrière-plan soit comme effets
- *les backgrounds* (décors ou arrière-plans): les images utilisées comme décors dans les rooms

Actuellement, il y a aussi d'autres types de ressources : les chemins (paths), les scripts, les fontes de caractères (fonts) et les lignes de temps (time lines). Elles sont utilisées dans les jeux plus sophistiqués. Vous ne pourrez les voir uniquement que lorsque vous exécuterez *Game Maker* en mode avancé. Elles seront abordées plus loin dans les chapitres plus avancés de ce document.

Etudions ensemble un exemple

Il est bon pour commencer d'étudier un exemple très simple. Nous supposerons que vous avez lancé *Game Maker* dans le mode simple. La première chose à faire est de décrire le jeu que nous souhaitons réaliser (vous devriez toujours commencer par ceci car cela vous économisera beaucoup de temps par la suite). Le jeu sera très simple : une balle rebondissant sur des murs. Le joueur devra essayer de cliquer sur la balle avec la souris. A chaque essai réussi, il gagnera un point.

Comme nous pouvons nous en rendre compte, nous avons besoin de deux objets différents : la **balle** et le **mur**. Nous aurons également besoin de deux sprites : un pour l'objet **mur** et un autre pour l'objet **balle**. Finalement, nous souhaiterons entendre un son quand nous réussirons à cliquer sur la balle avec la souris. Nous utiliserons juste une room (salle) où le jeu se déroulera (si vous ne souhaitez pas faire le jeu par vous-même, vous pouvez charger le fichier "`hit the ball.gm6`" à partir du répertoire "`Examples`").

Première chose à créer : les sprites. Sélectionnez **Add Sprite** dans le menu **Add** (vous pouvez aussi utiliser le bouton approprié dans la barre d'outils). Un écran s'affichera. Dans le champs **Name**, tapez “spr_wall”. Choisissez le bouton **Load Sprite** puis sélectionnez une image appropriée pour le mur. C'est tout : fermez maintenant la fenêtre. De la même manière, créez le sprite de la balle en donnant “spr_ball” comme nom de sprite.

Maintenant, nous allons créer le son. Dans le menu **Add**, sélectionnez **Add Sound**. Une fenêtre différente s'ouvre. Donnez un nom au son (ex: "son_hit_ball") puis choisissez **Load Sound**. Choisissez un son approprié puis écoutez-le en appuyant sur le bouton de lecture. Si vous êtes satisfait, vous pouvez fermer la fenêtre.

La prochaine étape consiste à créer les deux objets. Commençons par l'objet **mur**. De nouveau, choisissez **Add Object** dans le menu **Add**. Un écran un peu plus complexe que les précédents, s'affichera. A gauche, il y a les informations générales sur l'objet. Donnez un nom à l'objet (ex: **obj_mur**) puis sélectionnez un sprite approprié pour le mur dans le menu du bas ("spr_wall"). Un mur étant de type solide, cliquez sur la case **Solid**. Ce sera tout pour le moment. De même, créez un nouvel objet, appelez-le **obj_balle** et affectez-lui le sprite de la balle ("spr_ball"). La balle ne doit pas être de type solide. Pour la balle, nous devons définir un comportement. Au milieu, vous pouvez voir une liste vide d'événements. En dessous, il y a un bouton nommé **Add Event**. Pressez-le et vous verrez tous les événements possibles. Sélectionnez **creation event**.

Il est maintenant ajouté à la liste des événements. Plus loin à droite, vous verrez les actions possibles pour un certain nombre de groupes. A partir du groupe **move**, sélectionnez l'action avec l'icône représentant 8 flèches rouges puis tirez-la (drag and drop) dans la liste des actions du milieu. Cette action créera l'objet move avec une certaine direction. Ceci fait, une fenêtre de dialogue apparaîtra dans laquelle vous indiquerez la direction du mouvement. Sélectionnez toutes les flèches (8 en tout) pour indiquer une direction aléatoire. Vous pouvez laisser la vitesse réglée à 8. Fermez maintenant la fenêtre de dialogue. La balle se déplacera lorsqu'elle sera créée. En second lieu, nous devons définir ce qu'il doit se passer lors d'un événement de collision avec le mur. Une nouvelle fois, pressez **Add Event**. Cliquez sur le bouton des événements de collision puis déplacez l'objet **mur** dans le menu du bas. Nous avons aussi besoin d'une action pour gérer le rebond (bounce action). Vous pouvez voir ce que fait chaque action en plaçant pendant quelques instants le curseur de la souris près du nom de l'action. Enfin, nous avons besoin de définir ce que nous devons faire lorsque l'utilisateur presse le bouton gauche de la souris sur la balle. Ajoutez les événements correspondants puis sélectionnez le bouton gauche de la souris à partir du menu pop-up. Quelques actions sont aussi nécessaires pour cet événement : l'une pour jouer un son (peut être trouvé dans le groupe des actions **main1**) et l'autre pour changer le score (dans le groupe **score**)) et deux de plus pour déplacer la balle à une nouvelle position aléatoire et la placer dans une nouvelle direction (de la même façon que dans la création de l'événement). Concernant l'action sur le son, sélectionnez le son approprié ("son_hit_ball"). Pour l'action sur le score, tapez 1 comme valeur et vérifiez que la case **Relative** est cochée. Cela signifie que 1 est ajouté au score actuel. (Si vous faites une erreur, vous pouvez double-cliquer sur l'action pour changer ces paramètres.)

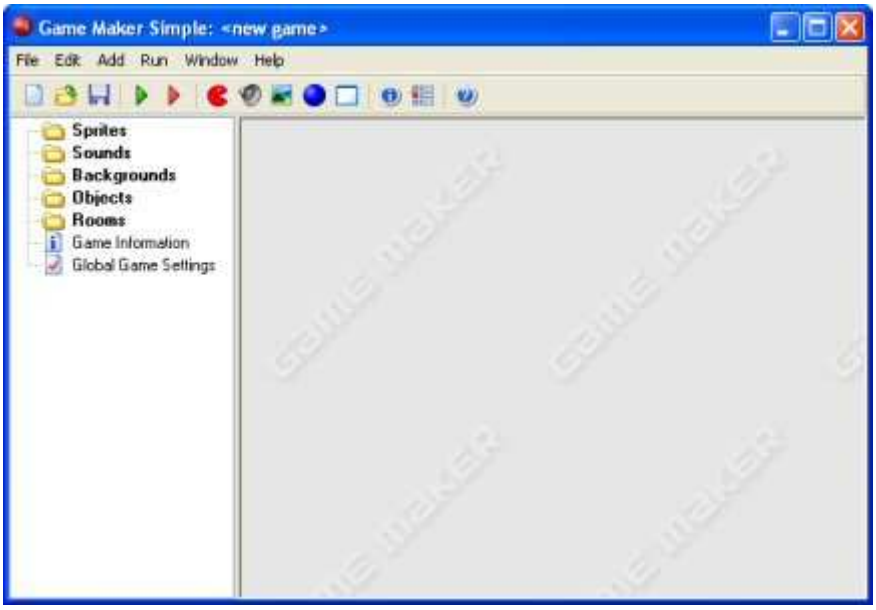
Nos objets sont maintenant prêts. Il reste à définir la room. Ajoutez une nouvelle **room** dans le jeu, toujours à partir du menu **Add**. A droite, vous voyez la room vide. A gauche, vous trouverez quelques onglets, un pour régler le décor (background) et un autre pour paramétrer les propriétés générales comme la largeur et la hauteur de la room, et un autre où vous pourrez ajouter des instances dans la room. En bas, vous pouvez sélectionner un objet dans le menu pop-up. En cliquant dans la room, vous pourrez placer des instances de cet objet à l'endroit souhaité. Vous pouvez enlever des instances en utilisant le bouton droit de la souris. Créez une jolie bordure autour de la room avec l'objet **obj_mur**. Enfin, placez 1 ou 2 objets **obj_balle** dans la room. Notre jeu est prêt.

Il est temps maintenant de tester notre jeu. Pressez le bouton **Run** (le triangle vert sur la barre de boutons en haut de la fenêtre) et regardez le déroulement du jeu. Si vous n'avez pas fait d'erreurs, la balle commence à se déplacer. Essayez de cliquer dessus avec la souris et regardez ce qui se passe. Vous pouvez arrêter le jeu en appuyant sur la touche <Esc>. Vous pourrez faire ultérieurement des modifications.

Félicitations. Vous venez de créer votre premier (petit) jeu. Mais il est temps maintenant d'en apprendre un peu plus sur *Game Maker*.

L'interface Utilisateur de base

Lors du lancement de *Game Maker*, l'écran suivant est affiché :



(Ceci est ce que vous pouvez voir quand vous exécutez *Game Maker* en mode simple. Dans le mode avancé, des options supplémentaires seront affichées). A gauche, il y a les différentes ressources mentionnées plus haut : les sprites, les sons (sounds), les décors (backgrounds), les objets, les salles (rooms) et deux ressources supplémentaires : 'Information sur le jeu' (Game Information) et 'Paramètres généraux de jeu' (Global Game Settings). En haut, nous trouvons le menu habituel ainsi que la barre d'outils. Dans ce chapitre, nous allons décrire brièvement les différents articles de menu, les boutons, etc. Dans les chapitres suivants, nous détaillerons un certain nombre d'entre eux. Veuillez noter qu'il y a plusieurs manières de faire certaines choses : en choisissant une commande à partir du menu, en cliquant sur un bouton ou en cliquant avec le bouton droit de la souris sur une ressource. Dans le menu **File** (Fichier), vous pourrez trouver les commandes habituelles comme charger et sauver les fichiers, plus celles-ci :

- **New**. (Nouveau) Choisissez cette commande pour commencer la création d'un nouveau jeu. Si le jeu actuel a été modifié, il vous sera demandé si vous souhaitez le sauvegarder. Il existe également un bouton de la barre d'outils pour réaliser cette action.
- **Open**. (Ouvrir) Ouvre un fichier jeu. Les fichiers *Game Maker* portent l'extension ".gm6" (vous pouvez aussi ouvrir d'anciens fichiers ".gmd" réalisés avec une version précédente de *Game Maker*. Notez que ceux-ci peuvent ne plus fonctionner correctement dans la nouvelle version). De même, il y a un bouton de la barre d'outils qui exécute cette commande.

- Il est également possible d'ouvrir un jeu en déplaçant à la souris (drag & drop) le fichier dans la fenêtre de *Game Maker* .
- **Recent Files**. (Fichiers récents) Utilisez ce sous-menu pour réouvrir les fichiers de jeu récemment ouverts.
- **Save**. (Sauver) Sauvegarde le jeu sous son nom actuel. Si aucun nom n'est précisé, un nouveau nom vous sera demandé. Notez que vous ne pouvez utiliser cette commande que si le fichier a déjà été modifié. Comme à l'accoutumée, il existe un bouton dans la barre d'outils pour effectuer la même action.
- **Save As**. (Sauver comme) Sauvegarde le jeu sous un nom différent. Un nouveau nom vous sera demandé.
- **Create Executable**. (Créer un exécutable) Une fois que votre jeu est prêt, vous souhaiterez probablement le donner à d'autres personnes pour qu'ils puissent y jouer. L'utilisation de cette commande vous permettra de créer une version autonome de votre jeu. C'est simplement un exécutable que vous pourrez donner à d'autres personnes afin qu'ils puissent y jouer (NDT : plus besoin alors de disposer de Game Maker).
- **Advanced Mode**. (Mode avancé) En cochant cette option, *Game Maker* basculera entre les modes simple et avancé. Dans le mode avancé, des commandes et des ressources supplémentaires sont disponibles.
- **Exit**. (Quitter) Probablement l'action la plus explicite. Sélectionnez cette option pour quitter *Game Maker*. Si le jeu actuel a été modifié, il vous sera demandé si vous souhaitez le sauver.

Menu Edition (Edit menu)

Le menu **Edit** (Edition) contient des commandes qui concernent les ressources actuellement sélectionnées (objet, sprite, son, etc.). Selon le type de ressource, certaines commandes peuvent ne pas être disponibles.

- **Insert resource**. (Insérer ressource) Insère une nouvelle instance du type de ressource actuellement sélectionné avant la ressource courante. Un écran s'ouvrira dans lequel vous pourrez modifier les propriétés de la ressource. Plus de détails dans les chapitres suivants.
- **Duplicate**. (Dupliquer) Réalise une copie de la ressource actuelle et l'ajoute dans les ressources disponibles. Un écran s'affiche dans lequel vous pourrez modifier la ressource.
- **Delete**. (Effacer) Supprime la ressource actuellement sélectionnée (ou groupe de ressources). Soyez prudent car il est impossible d'annuler cette opération. Cependant, un message de confirmation s'affichera.
- **Rename**. (Renommer) Donne un nouveau nom à la ressource. Ceci peut être également fait à partir de l'écran de propriétés de la ressource. Vous pouvez aussi choisir la ressource et simplement cliquer sur son nom.

- **Properties**. (Propriétés) Utilisez cette commande afin d'afficher la fenêtre de modification des propriétés. Veuillez noter que tous les écrans de propriétés sont accessibles dans la même fenêtre. Vous pourrez ainsi éditer plusieurs propriétés en même temps. Il est possible également d'éditer les propriétés en double-cliquant sur la ressource.

Note : Toutes ces commandes peuvent être accessibles de plusieurs façons. Faire un clic droit sur une ressource ou un groupe de ressources fera apparaître également le menu pop-up.

Menu Ajout (Add menu)

Dans ce menu, vous pourrez ajouter de nouvelles ressources de tout type. De même, il existe un bouton dans la barre d'outils ainsi qu'un raccourci-clavier qui effectuent la même action.

Menu Exécution (Run menu)

Ce menu est utilisé pour lancer le jeu. Il y a deux manières de lancer un jeu.

- **Run normally**. (Lancement normal) Lance le jeu en mode normal. Le jeu est exécuté de la façon la plus efficace possible et ressemble en tout point au jeu sous sa forme exécutable.
- **Run in Debug mode**. (Lancement en mode déboguage) Lance le jeu dans le mode déboguage. Dans ce mode, vous pourrez vérifier certains aspects du jeu, effectuer une pause et avancer pas à pas dans le jeu. Cela peut être utile lorsque quelque chose ne fonctionne pas comme prévu. Veuillez noter que ce mode est plutôt réservé à des utilisateurs avancés.

Une fois votre jeu terminé, vous pourrez créer une version exécutable autonome du jeu en utilisant la commande ad hoc dans le menu **File** (menu Fichier).

Menu Fenêtre (Window menu)

Dans ce menu, vous trouverez les commandes usuelles pour gérer la disposition et l'affichage des différentes propriétés des fenêtres :

- **Cascade**. Affiche en cascade toutes les fenêtres de manière à ce que chacune d'entre elles soit visible partiellement.
- **Arrange Icons**. (Ranger les fenêtres icônifiées) Aligne toutes les fenêtres icônifiées (ceci peut être utile notamment après avoir redimensionné l'écran principal).
- **Close All**. (Fermer tout) Ferme toutes les fenêtres, vous demandant si nécessaire de sauver ou non les changements effectués.

Menu d'Aide (Help menu)

Vous trouverez ici quelques commandes pour vous aider :

- **Contents**. (Sommaire) Utilisez cette commande pour afficher ce menu d'aide.
- **Registration**. (Enregistrement) Bien que la version de base de *Game Maker* puisse être utilisée gratuitement, je vous encourage à enregistrer le programme. Cela débloquera les fonctionnalités supplémentaires du logiciel et soutiendra les développements futurs du programme. Vous trouverez ici les informations sur la manière d'enregistrer le logiciel. Si vous venez de vous enregistrer, utilisez cette commande pour entrer la clé d'enregistrement que vous avez reçue.
- **Web site**. (Site Web) Vous connecte sur le site web de *Game Maker* où vous pourrez trouver des informations sur la plus récente version de *Game Maker* mais aussi un grand nombre de jeux et de ressources pour *Game Maker*. Nous vous recommandons d'aller sur ce site pour vérification au moins une fois par mois.
- **About *Game Maker***. (A propos de Game Maker) Donne quelques informations rapides sur cette version de *Game Maker*.

L'Explorateur de ressources (The resource explorer)

A gauche de l'écran principal, vous trouverez l'explorateur de ressources. Vous pouvez voir une vue arborescente de toutes les ressources composant votre jeu. Cela fonctionne de la même manière que dans l'explorateur de Windows et ainsi, vous devriez vous sentir familier avec l'explorateur. Si un article a un signe **+** en face de son nom, vous pourrez alors cliquer sur ce signe pour voir le contenu de la ressource. En cliquant sur le signe **-**, le contenu disparaîtra à nouveau. Vous pouvez changer le nom de la ressource (sauf pour les premiers niveaux du haut) en la sélectionnant (d'un simple clic) puis en cliquant sur le nom. Double-cliquez sur une ressource pour éditer ses propriétés. Utilisez le bouton droit de la souris pour accéder aux mêmes commandes du menu **Edit** (menu Edition).

Vous pouvez changer l'ordre des ressources en cliquant dessus avec la souris et en maintenant le bouton de celle-ci pressé. Maintenant, déplacez (drag) la ressource à l'endroit souhaité (bien sûr, l'emplacement choisi doit être correct --> vous ne pouvez pas déplacer un son dans la liste des sprites).

Définition des Sprites

Les sprites sont la représentation visible des objets d'un jeu. Un sprite est soit une simple image, dessinée avec un quelconque programme de dessin, soit une suite d'images qui, affichées l'une après l'autre, crée une animation. Par exemple, les quatre images suivantes forment le sprite d'un Pacman se déplaçant vers la droite.

Lors de la création d'un jeu, vous commencerez normalement à vous procurer une collection de jolis sprites pour les objets de votre jeu. Des collections de sprites intéressants peuvent être trouvées sur le site web de *Game Maker*. D'autres sprites peuvent être téléchargés sur le web, généralement sous la forme d'images GIF animées.

Pour ajouter un sprite, choisissez l'article **Add Sprite** du menu **Add**, ou bien utilisez le bouton correspondant dans la barre d'outils. L'écran suivant s'affichera alors.

En haut, vous pourrez indiquer un nom de sprite différent de celui proposé. Tous les sprites (ainsi que toutes les autres ressources) doivent avoir un nom. Il est préférable de donner à chacun des sprites un nom évocateur. Soyez certain que toutes les ressources ont un nom différent. Bien que non obligatoire, il est fortement conseillé d'utiliser uniquement des lettres, des chiffres et le sigle "underscore" (_) dans le nom de sprite (et des autres ressources également) et de commencer par une lettre. N'utiliser pas notamment le caractère **espace**. Ceci est très important si vous envisagez d'utiliser la programmation dans votre jeu.

Pour charger un sprite, cliquez sur le bouton **Load Sprite**. Une boîte de dialogue standard s'ouvrira dans laquelle vous choisirez un fichier sprite. *Game Maker* peut charger différents fichiers

graphiques. Quand vous chargez une animation GIF , les sous-images successives constituent les images du sprite. Un fois le sprite chargé, la première image est affichée à droite. Lorsque le sprite comporte plusieurs images, vous pouvez les afficher une par une en utilisant le bouton de type flèche.

La case à cocher nommée **Transparent** indique que le fond rectangulaire de l'image du sprite sera considéré comme transparent. La plupart des sprites sont transparents. La couleur du fond du sprite est déterminée par celle du pixel en bas le plus à gauche de l'image. Aussi, vérifiez qu'il n'existe pas de pixel de l'image actuelle qui ait cette couleur. (veuillez noter que les fichiers GIF définissent généralement leur propre couleur transparente. Cette couleur ne sera pas utilisée par *Game Maker*).

Avec le bouton **Edit Sprite**, vous pouvez éditer le sprite pour le modifier et même créer un nouveau sprite.

Sons et Musiques

La quasi-totalité des jeux présentent des effets sonores et une musique de fond. Les effets sonores les plus intéressants peuvent être trouvés sur le site web de *Game Maker*. D'autres sons sont également disponibles sur d'autres sites web.

Pour ajouter à votre jeu une ressource de type son, utilisez l'article **Add Sound** du menu **Add** ou utilisez le bouton correspondant de la barre d'outils. L'écran suivant s'affichera.

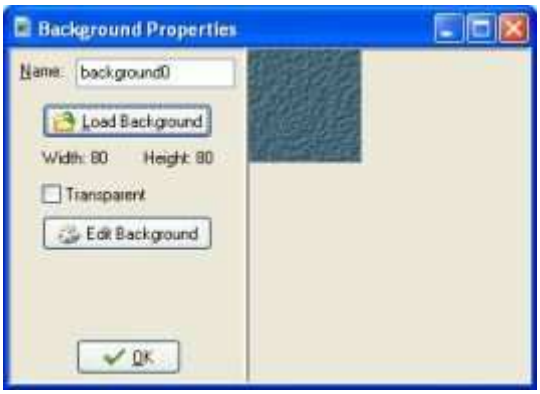
Pour charger un son, appuyez sur le bouton **Load Sound**. Un sélecteur de fichiers apparaîtra alors dans lequel vous pourrez sélectionner le fichier sonore. Il y a deux types de fichiers son, les fichiers **Wave** et les fichiers **Midi**. Les fichiers Wave sont utilisés pour les courts effets sonores. Ils utilisent beaucoup de mémoire mais sont joués instantanément. Les utiliser pour tous les effets sonores de votre jeu. Les fichiers Midi décrivent la musique d'une façon différente. Ils utilisent beaucoup moins de mémoire mais se limitent au fond sonore à base d'instruments de musique. De plus, un seul son midi peut être joué à la fois.

Une fois votre fichier musical chargé, vous pouvez l'écouter en utilisant le bouton de lecture. Il existe également le bouton **Save Sound** pour sauver le son actuel dans un fichier. Ce bouton n'est pas réellement indispensable mais vous pourriez en avoir besoin en cas de perte du son original.

Arrière-Plans (Décors)

Le troisième type de ressources de base sont les arrière-plans (ou décors). Les arrière-plans sont généralement de grandes images utilisées comme fonds de jeu (ou comme avant-plans) pour les rooms (salles) où le jeu se déroule. Très souvent, les images d'arrière-plan sont conçues de telle façon qu'elles couvrent plusieurs surfaces de l'écran de jeu sans décrochement visuel. Ainsi, vous pouvez remplir le décor avec quelques petits modèles graphiques appelés tuiles (tiles). Un certain nombre de tuiles de fonds d'écran sont téléchargeables sur le site web de *Game Maker*. On peut en trouver bien d'autres sur le web.

Pour ajouter à votre jeu une ressource de type arrière-plan (background resource), utilisez l'article **Add Background** du menu **Add** ou utilisez le bouton correspondant de la barre d'outils. La fenêtre suivante apparaîtra.



Appuyez sur le bouton **Load Background** pour charger une image de fonds. *Game Maker* supporte différents formats d'images. Les images d'arrière-plan ne peuvent être animées ! La case à cocher **Transparent** indique si **OUI** ou **NON** le fonds doit être partiellement transparent. La plupart des décors de jeu ne sont pas transparents, aussi la valeur est fixée par défaut à non transparent. La couleur du pixel situé le plus en bas à gauche du décor sera utilisée comme couleur de transparence.

Il vous est possible de changer l'arrière-plan ou d'en créer un nouveau en utilisant le bouton **Edit Background**.

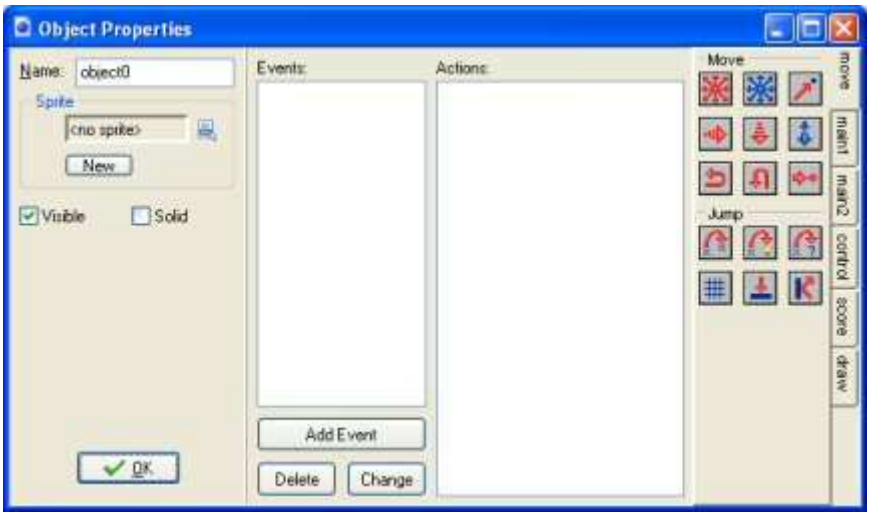
Définition des objets

Avec les ressources, vous avez vu jusqu'ici que vous pouvez ajouter de belles images et des sons dans le jeu mais que celles-ci ne font rien d'autres. Nous arrivons maintenant à la plus importante ressource de *Game Maker* : les **objets**. Les objets sont les entités de jeu qui accomplissent des actions. La plupart du temps, les sprites sont les objets visibles du jeu car ce sont des représentations graphiques de ce dernier. Ils présentent un comportement car ils peuvent réagir à certains événements. Toutes les choses que vous voyez dans un jeu (à l'exception des arrière-plans) sont des objets (ou pour être plus précis, des instances d'objets). Les personnages, les monstres, les balles, les murs, etc. sont tous des objets. Il existe aussi des objets invisibles et qui contrôlent certains aspects du gameplay.

Veuillez bien faire la différence ente les sprites et les objets. Les sprites ne sont que des images (animées) qui n'ont pas de comportement particulier. Habituellement, les objets possèdent un sprite les représentant mais les objets possèdent surtout des comportements. Sans objet, il n'y aurait donc pas de jeu !

Faites bien aussi la distinction entres les objets et les instances. Un objet décrit une certaine entité, par exemple un monstre. Il peut y avoir de multiples instances de cet objet dans le jeu. Quand nous parlons d'instance, nous désignons une instance bien particulière de cet objet. Lorsque nous parlons d'un objet, nous voulons parler de toutes les instances de ce même objet.

Pour ajouter un objet dans votre jeu, choisissez **Add Object** du menu **Add**. La fenêtre suivante apparaîtra :



Cet écran paraît plus complexe car il présente davantage de fonctionnalités. A gauche, nous trouvons les informations générales sur l'objet. Au milieu, il y a une liste d'événements pouvant interagir avec l'objet. A droite, on trouve les différentes actions que l'objet peut effectuer. Les événements et les actions seront abordés dans les prochains chapitres.

Comme d'habitude, vous pouvez (et même devriez) donner un nom à votre objet. Ensuite, vous pourrez choisir le sprite pour l'objet. Dans ce but, cliquez sur le champs du sprite à l'aide du bouton gauche de la souris ou encore sur le bouton de menu tout près de lui. Un menu s'affichera contenant tous les sprites disponibles. Sélectionnez celui que vous souhaitez utiliser pour votre objet. Si vous ne disposez pas de sprite déjà tout fait, cliquez sur le bouton **New** pour ajouter une nouvelle ressource sprite et la modifier. Quand vous sélectionnez une ressource, il y a également un bouton **Edit** que vous utiliserez pour changer le sprite. Il est plus rapide de procéder de la sorte plutôt que de rechercher la ressource dans la liste des ressources puis d'indiquer que vous voulez l'éditer.

En dessous, il existe deux cases à cocher : **Visible** indique si les instances de cet objet doivent être visibles. En clair, la plupart des objets sont visibles mais parfois, il est utile d'avoir des objets invisibles. Par exemple, vous pouvez les utiliser dans le but d'avoir un monstre en mouvement. Les objets invisibles réagiront aux événements et aux autres instances et pourront rentrer en collision avec eux. La case **Solid** (Solide) indique si un objet doit être de type solide (comme un mur). Les collisions entre objets solides sont traitées de façon différente de celles avec les objets non solides. Il est fortement conseillé d'utiliser l'option **Solid** uniquement pour les objets ne se déplaçant pas.

Les Evénements

Game Maker utilise une approche temps réel pour la gestion des événements. Cela fonctionne de la manière suivante. Lorsque quelque chose se produit dans le jeu, les instances des objets reçoivent des événements (sorte de messages indiquant que quelque chose s'est produit). Les instances peuvent alors réagir à ces messages en exécutant certaines actions. Pour chaque objet, vous devrez donc indiquer à quels événements ils doivent répondre et quelles actions ils doivent accomplir quand l'événement survient. Cela peut sembler compliqué mais c'est en fait très simple. Premièrement, pour la plupart des événements, les objets n'auront rien à faire de particulier. En ce qui concerne les événements où une action est nécessaire, vous utiliserez une approche de type 'drag-and-drop' (sélectionner & déposer) très simple pour spécifier les actions à accomplir.

Au milieu de la fenêtre des propriétés de l'objet, se trouve une liste des événements pour lesquels l'objet doit réagir. Au départ, cette liste est vide. Vous pourrez ajouter des événements en appuyant sur le bouton **Add Event**. Un écran apparaîtra avec tous les différents types d'événements possibles. Vous sélectionnerez ici l'événement que vous souhaitez ajouter. Parfois, un menu s'affichera avec des choix supplémentaires. Par exemple, pour les événements clavier, vous devrez choisir la touche concernée par l'événement. En dessous, vous trouverez une liste complète des différents événements avec leur description. Un seul événement de la liste peut être sélectionné à la fois. C'est l'événement sur lequel nous sommes en train de travailler. Vous pouvez changer l'événement sélectionné en cliquant dessus. A droite apparaissent toutes les actions symbolisées par de petites icônes. Elles sont regroupées sous forme d'onglets. Au prochain chapitre, nous verrons les descriptions de toutes ces actions et ce qu'elles réalisent. Entre les événements et les actions, il y a la liste d'actions. Cette liste contient les actions que l'événement courant peut accomplir. Pour ajouter des actions à la liste, déposez (drag) une action à l'aide de la souris dans la liste des actions apparaissant sur la droite. Elles seront placées chacune l'une en dessous de l'autre, accompagnées d'une courte description. Pour chaque action, il vous sera demandé de fournir quelques paramètres. Ils seront également abordés dans le chapitre suivant. Après avoir ajouté quelques actions, l'affichage ressemblera à ceci :

Vous pouvez maintenant ajouter des actions à d'autres événements. Cliquez sur l'événement concerné avec le bouton gauche de la souris pour le sélectionner puis déposer (drag) des actions dans la liste.

Vous pouvez modifier l'ordre des actions de la liste toujours en utilisant la fonction 'drag-and-drop' (sélectionner & déposer). Si vous maintenez la touche <Alt> lors d'un déplacement, vous créerez une copie de l'action. Vous pouvez même utiliser la fonction 'drag-and-drop' à l'intérieur des listes d'actions de différents objets. Lorsque vous cliquez avec le bouton droit de la souris sur une action, un menu apparaît dans lequel vous pourrez effacer l'action sélectionnée (peut être également réalisé en utilisant la touche <Del>) mais aussi copier ou coller des actions (vous pouvez aussi sélectionner plusieurs actions en même temps pour les couper, les copier ou les effacer en maintenant les touches <Shift> ou <Ctrl>. Pressez <Ctrl><A> pour sélectionner toutes les actions). Quand le curseur de la souris reste de manière prolongée sur une action, une plus longue description de l'action s'affiche. Voir le prochain chapitre pour plus d'informations sur les actions.

Pour effacer l'événement actuellement sélectionné ainsi que toutes ses actions, appuyez sur le bouton **Delete**. (les événements sans action seront automatiquement supprimés quand vous fermerez la fenêtre. Aussi, il n'est pas nécessaire de procéder manuellement à leur effacement). Si vous souhaitez assigner des actions à un événement différent (par exemple parce que vous souhaitez utiliser une touche différente pour celles-ci), appuyez sur le bouton **Change** puis sélectionnez le nouvel événement souhaité (l'événement ne doit pas être déjà défini !). En utilisant le menu apparaissant lors du clic droit de la souris sur la liste d'événements, vous pourrez ainsi dupliquer un événement, ce qui en final, ajoutera un nouvel événement avec les mêmes actions.

Comme indiqué ci-dessus, pour ajouter un événement, pressez le bouton **Add Event**. La fenêtre suivante s'affichera :

Vous pourrez ici sélectionner l'événement que vous souhaitez ajouter. Certaines fois, un menu s'affichera avec des choix complémentaires. Voici une description des différents événements. (rappelez-vous que vous ne devriez normalement n'en utiliser que très peu).

Create event (Evénement à la création)

Cet événement survient lorsque l'instance de l'objet est créée. Cela est utilisé habituellement pour paramétrer l'instance dans ses mouvements et/ou donner une valeur aux variables de l'instance.

Destroy event (Evénement à la destruction)

Cet événement arrive quand l'instance est détruite. Pour être plus précis, cela survient juste avant qu'elle ne soit détruite, ainsi l'instance existe encore lorsque l'événement surgit ! En principe, cet événement n'est pas utilisé mais vous pouvez néanmoins l'employer par exemple pour modifier le score ou encore pour créer un autre objet.

Alarm events (Evénements d'alarme)

Chaque instance a 12 alarmes chronomètres. Vous pouvez paramétrer les alarmes chronomètres en utilisant certaines actions (voir chapitre suivant). L'horloge effectue alors un compte à rebours et déclenche l'événement d'alarme quand le chronomètre arrive à 0. Pour mentionner les actions à effectuer pour une alarme donnée, vous devez en premier lieu sélectionner cette dernière dans le menu. Les alarmes chronomètres sont très utiles. Vous pouvez les utiliser afin que certaines choses surviennent de temps en temps. Par exemple, un monstre pourra changer de direction tous les 20 steps (20 unités de temps). Dans ce cas, une des actions de l'événement devra de nouveau fixer la valeur de l'alarme chronomètre.

Step events (Evénements d'étape)

L'événement step (étape) survient à chaque étape du jeu. Vous pouvez ici mettre des actions qui ont besoin de s'exécuter en permanence. Par exemple, dans le cas où un objet devrait en suivre un autre, vous pourrez ici adapter la direction du mouvement vers l'endroit où se trouve l'objet à suivre. Soyez néanmoins prudent avec ce type d'événement. Ne placez pas trop d'actions complexes dans l'événement step des objets où il existe plusieurs instances de ces derniers. Cela pourrait ralentir considérablement le jeu. Il existe trois événements step différents. En temps normal, vous ne devriez en avoir besoin que d'un seul. Mais, à l'aide du menu, vous pouvez aussi sélectionner l'événement début de step et l'événement fin de step. L'événement début de step est exécuté au début de chaque step, avant que tout autre événement ne soit pris en compte.

L'événement normal step est exécuté juste avant que les instances soient placées à leurs nouvelles positions. Enfin, l'événement fin de step est exécuté à la fin du step, juste avant l'étape d'affichage. Ceci est souvent utilisé pour changer un sprite en fonction de sa direction actuelle.

Collision events (Evénements de collision)

Lorsque deux instances entrent en collision (cela signifie que leurs sprites se chevauchent), un événement de collision est généré. En fait et pour être plus précis, deux événements de collision surviennent alors, un pour chacune des instances. L'instance peut réagir à l'événement de collision. A cet effet, à partir du menu, sélectionnez l'objet pour lequel vous souhaitez définir l'événement de collision. Puis placez vos actions à cet endroit.

Il existe une différence lorsqu'une instance entre en collision avec un objet solide ou non solide. Première chose, quand il n'y a aucune action de définie dans l'événement de collision, rien ne se passera. L'instance courante continue simplement son déplacement; même si l'autre objet est de type solide. Lorsque l'événement de collision contient des actions, il peut survenir les choses suivantes :

Si l'autre objet est de type solide, l'instance est replacée à son ancienne position (c'est à dire à la position avant la collision). Ensuite l'événement est exécuté. Finalement, l'instance est déplacée à sa nouvelle position. Si, par exemple, l'événement inverse la direction du déplacement, l'instance rebondit alors contre le mur sans s'arrêter. S'il y a toujours collision, l'instance est replacée à sa position précédente. Puis elle stoppe effectivement son déplacement.

Si l'autre objet n'est pas de type solide, l'instance n'est pas replacée à sa position précédente. L'événement est simplement exécuté avec l'instance à sa position courante. Ainsi, il n'y a donc pas de seconde vérification de collision. Si vous réfléchissez bien, c'est logiquement ce qui doit se passer. Parce que l'objet n'est pas de type solide, nous pouvons donc simplement le traverser. L'événement nous indique ce qui vient de se passer.

Les événements de collision peuvent être utilisés pour divers usages. Les instances peuvent les utiliser pour rebondir contre des murs. Vous pouvez les employer par exemple pour détruire des objets quand leurs instances sont touchées par une balle ou encore un obus.

Keyboard events (Evénements clavier)

Lorsque le joueur presse une touche, un événement clavier apparaît pour toutes les instances de tous les objets. Il existe un événement différent pour chaque touche. Dans le menu, vous pouvez choisir la touche pour laquelle vous souhaitez définir un événement clavier afin d'y déposer des actions. De manière générale, seuls quelques objets nécessitent des événements pour ne gérer finalement que quelques touches. Vous obtenez un événement à chaque step tant que le joueur appuie sur la touche. Il existe deux événements clavier spéciaux. L'un d'eux s'appelle <No key>.

Cet événement survient à chaque step lorsque aucune touche n'est pressée. L'autre se nomme <Any key> et arrive quand une touche est pressée. Lorsque le joueur presse sur plusieurs touches, les événements se produisent pour chacune des touches. Veuillez noter que les touches du clavier numérique renvoient des événements que lorsque <NumLock> est en fonction.

Mouse events (Evénements souris)

Un événement souris est visible par une instance lorsque le curseur de la souris survole le sprite représentant cette instance. Selon le bouton souris pressé, vous pouvez obtenir l'événement aucun bouton, bouton gauche, bouton droit ou bouton du milieu. Les événements souris sont créés à chaque step tant que le joueur maintient le bouton de la souris enfoncé. Les événements souris presse-bouton n'apparaissent que lorsqu'un bouton de la souris est pressé. Les événements relâche-bouton sont générés seulement quand le bouton est relâché. Veuillez noter que ces événements apparaissent uniquement quand la souris se trouve au-dessus d'une instance. Si vous souhaitez par contre effectuer une action lorsque l'on appuie ou relâche un bouton de la souris à un endroit arbitraire (NDT : donc indépendant d'une instance), utilisez les événement généraux de la souris que l'on peut trouver dans un des sous-menu. Il existe également deux événements spéciaux pour la souris. L'événement souris à l'entrée survient quand la souris pénètre dans l'instance. L'événement souris à la sortie apparaît lorsque la souris quitte l'instance. Ces événements sont habituellement utilisés pour changer d'image ou jouer certains sons. Les événements souris de mouvement de roulette vers le haut ou vers le bas surviennent lorsque l'utilisateur actionne la molette de la souris dans l'une des deux directions. Enfin, il existe un certain nombre d'événements concernant le joystick. Vous pouvez indiquer des actions pour les quatre directions principales du joystick (en diagonale, deux événements se produisent). Vous pouvez aussi définir des actions prenant en compte simultanément jusqu'à 8 boutons du joystick. Vous pouvez ainsi gérer à la fois le premier et un éventuel second joystick.

Other events (Autres événements)

Il y a d'autres événements pouvant être utiles dans certains jeux. Ils sont accessibles à partir de ce menu. Les événements disponibles sont les suivants :

- **Outside** (En dehors) : Cet événement survient lorsque l'instance se situe complètement hors de la room. C'est généralement à ce moment-là que l'on décide de détruire l'instance.
- **Boundary** (Au bord) : Cet événement apparaît quand l'instance touche les bords de la room.
- **Game start** (Au début du jeu) : Cet événement concerne toutes les instances de la première room au lancement du jeu. Cela survient avant que la room lance l'événement (voir ci-dessous) mais après les événements à la création (Creation events) des instances de la room. Cet événement est défini typiquement dans un seul objet "controller"

et s'utilise pour démarrer une musique de fond, pour initialiser des variables ou encore charger des données.

- **Game end** (A la fin du jeu) : Cet événement concerne toutes les instances lorsque se termine le jeu. Comme d'habitude, un seul objet définit cet événement. On l'utilise par exemple pour stocker des données dans un fichier.
- **Room start** (Au lancement de la room) : Cet événement concerne toutes les instances déjà présents dans la room quand celle-ci démarre. Ceci se produit après les événements à la création.
- **Room end** (A la fin de la room) : Cet événement concerne toutes les instance existantes lorsque la room s'achève.
- **No more lives** (Plus de vies) : *Game Maker* intègre un système de gestion des vies dans le jeu. Il existe une action pour fixer et changer le nombre de vies. Lorsque le nombre de vies devient inférieur ou égal à 0, cet événement apparaît. On l'utilise souvent pour finir ou relancer le jeu.
- **No more health** (Plus de santé) : *Game Maker* comprend un système de gestion de santé du joueur. Il y a une action pour régler et changer la santé du joueur. Quand la santé est inférieure ou égale à 0, cet événement survient. Habituellement utilisé pour réduire le nombre de vies ou pour relancer le jeu.
- **End of animation** (Fin d'animation) : Comme mentionné ci-dessus, une animation est constituée d'un certain nombre d'images affichées l'une après l'autre. Lorsque la dernière image est affichée, on boucle en réaffichant la première. Cet événement apparaît précisément à ce moment. Ceci peut être utilisé par exemple pour changer l'animation ou pour détruire l'instance.
- **End of path** (Fin de chemin) : Cet événement survient lorsque l'instance suit un chemin et que la fin de chemin est atteinte.
- **User defined** (Evénement défini par l'utilisateur) : 16 événements utilisateur peuvent être définis. Ils ne surviennent généralement jamais sauf si vous-même les invoqués dans un programme.

Drawing event (Evénement à l'affichage)

Les instances, lorsqu'elles sont visibles, dessinent leur sprite sur l'écran à chaque step. Quand vous spécifiez des actions dans l'événement d'affichage, le sprite n'est pas dessiné mais seules ces actions sont exécutées. Ceci peut être utilisé pour dessiner autre chose qu'un sprite ou encore effectuer des changements dans les paramètres du sprite. Un certain nombre d'actions d'affichage existent, destinées principalement aux événements à l'affichage. Veuillez noter que les événements à l'affichage sont uniquement exécutés lorsque les objets sont visibles. Notez également qu'indépendamment de ce que vous dessinez, les événements de collision ne concernent que le sprite associé à l'instance.

 **Key press events (Evénements si touche pressée)**

Cet événement est similaire à l'événement clavier mais survient une seule fois lorsqu'une touche est pressée, plutôt qu'en continu. Ceci est utile quand vous souhaitez une action ne devant survenir qu'une unique fois.

 **Key release events (Evénements si touche relâchée)**

Cet événement est semblable à l'événement clavier sauf qu'il se produit une unique fois lors du relâchement d'une touche et non de manière continue.

Dans certains cas, il est important de bien comprendre l'ordre dans lequel *Game Maker* traite les événements. L'ordre d'exécution est le suivant :

- Begin step events (Evénements de début de step)
- Alarm events (Evénements alarme chronomètre)
- Keyboard, Key press, and Key release events (Evénements clavier, si touche pressée et si touche relâchée)
- Mouse events (Evénements de la souris)
- Normal step events (Evénements step normaux)
- (now all instances are set to their new positions) (toutes les instances sont placées désormais à leurs nouvelles positions)
- Collision events (Evénements de collision)
- End step events (Evénements de fin de tep)
- Drawing events (Evénements à l'affichage)

La création, la destruction et les autres événements ne sont traités que lorsque des choses les concernant surviennent.

Les Actions

Les actions indiquent que quelque chose s'est produit dans le jeu créé avec *Game Maker*. Les actions sont placées dans les événements des objets. Lorsque l'événement survient, ces actions sont exécutées et influent sur le comportement des instances de l'objet. Il existe un grand nombre d'actions différentes disponibles et il est important de bien comprendre leur rôle respectif. Dans ce chapitre, nous allons décrire les actions accessibles dans le mode simple. Veuillez noter que certaines de ces actions ne sont disponibles que dans la version enregistrée de *Game Maker*. Ceci sera mentionné le cas échéant.

Toutes les actions sont disponibles dans des pages d'onglets situées à droite de la fenêtre de propriétés des objets. Il y a six jeux d'actions possibles. Vous sélectionnez le jeu souhaité en cliquant sur l'onglet approprié. Lorsque vous maintenez la souris appuyée au-dessus d'une action, une courte description s'affiche pour vous permettre de comprendre sa finalité.

Faisons un bref rappel : Pour placer une action dans un événement, il suffit de la déposer (drag) dans la liste d'actions à partir des pages d'onglets. Vous pouvez changer l'ordre dans la liste, en procédant de la même manière (dragging). Pour faire une copie de l'action, maintenez la touche <Alt> tout en la déplaçant (drag). Vous pouvez déposer (drag) et copier des actions à l'intérieur des listes des différents écrans de propriétés d'objets. Utilisez le bouton droit de la souris puis sélectionnez l'article du menu concerné pour supprimer des actions (ou encore utilisez la touche <Del> key), les copier ou les coller.

Lorsque vous lâcher (drop) une action dans la liste des actions, une fenêtre apparaîtra la plupart du temps, dans laquelle vous pourrez indiquer certains paramètres concernant cette action. Les paramètres seront décrits ci-dessous lors de la description des actions. Deux types de paramètres concernent la plupart des actions. Nous allons donc les décrire ici. En haut, vous pouvez mentionner sur quelle instance doit s'appliquer l'action. Par défaut, c'est `self` qui est l'instance pour laquelle l'action est exécutée. La plupart du temps, c'est ce que vous devrez utiliser. Dans le cas d'un événement de collision, vous pourrez préciser si l'action doit s'appliquer à une autre instance concernée par la collision. De cette manière, vous pourrez détruire une autre instance par exemple. Enfin, vous pourrez choisir d'appliquer l'action à toutes les instances d'un objet particulier. Ainsi, il vous sera possible de changer toutes les balles rouge en balles bleu. Le second type de paramètre est la case nommée **Relative**. En cochant cette case, les valeurs saisies seront relatives par rapport aux valeurs courantes. De cette façon, vous pourrez ajouter des points au score actuel, plutôt que d'affecter une nouvelle valeur absolue à ce score. Les autres paramètres

seront décrits plus bas. Vous pourrez ultérieurement modifier ces paramètres en double-cliquant sur l'action.

Des informations sur les différentes actions peuvent être trouvées dans les pages suivantes :

Les Actions de Mouvements

Le premier jeu d'actions concerne les déplacements d'objets. Les actions suivantes existent :

 **Start moving in a direction (Déplacement dans une direction)**

Utilisez cette action pour effectuer le déplacement d'une instance dans une certaine direction. Vous pouvez indiquer la direction de l'instance en utilisant les boutons fléchés. Utilisez le bouton du milieu pour stopper le déplacement. Vous devrez mentionner également la vitesse de déplacement. La vitesse est donnée en pixels par step. La valeur par défaut est de 8. Il est préférable de ne pas utiliser de valeurs négatives pour régler la vitesse. Vous pouvez indiquer plusieurs directions. Dans ce cas, un choix aléatoire sera réalisé. De cette manière, un monstre peut aller soit à gauche soit à droite.

 **Set direction and speed of motion (Régler la direction et la vitesse du déplacement)**

Ceci constitue la deuxième manière de réaliser un déplacement. Ici, vous indiquerez une direction précise. C'est un angle compris entre 0 et 360 degrés, 0 signifiant aller vers la droite. La direction suit le mouvement inverse des aiguilles d'une montre. Ainsi, par exemple, 90 indique un déplacement vers le haut. Si vous souhaitez une direction arbitraire, tapez alors `random(360)`. Comme nous le verrons ci-dessous, la fonction `random` fournit un nombre aléatoire inférieur à la valeur donnée en paramètre. Vous avez sans doute remarqué qu'il existe une case nommée **Relative**. Si vous la cochez, le prochain déplacement sera ajouté au précédent. Par exemple, si l'instance se déplace vers le haut et que vous ajoutez un déplacement vers la gauche, le nouveau déplacement se fera du haut vers la gauche.

 **Move towards a point(Déplacement vers un point donné)**

Cette action fournit une troisième façon de calculer un déplacement. Vous indiquerez ici une position et une vitesse et l'instance commencera à se déplacer vers le point désigné, à la vitesse indiquée (elle ne s'arrêtera pas à ce point !). Par exemple, si vous voulez qu'un boulet se dirige en direction d'un vaisseau spatial, vous pourrez utiliser comme position `spaceship.x`, `spaceship.y` (vous en apprendrez davantage sur l'emploi des variables plus bas dans ce document). Si vous cochez la case **Relative**, vous mentionnerez la position relative à celle actuelle de l'instance (la vitesse quant à elle, n'est pas prise en compte de manière relative !)

 **Set the horizontal speed (Régler la vitesse horizontale)**

La vitesse d'une instance se décompose horizontalement et verticalement. Avec cette action, vous pouvez changer la vitesse horizontale. Une vitesse horizontale positive signifie un déplacement vers la droite. Une valeur négative indique un déplacement vers la gauche.

La vitesse verticale suit le même principe. Utilisez la case relative pour augmenter la vitesse horizontale (une valeur négative la diminuera).

 **Set the vertical speed (Régler la vitesse verticale)**

De la même manière, vous pouvez changer avec cette action la vitesse verticale d'une instance.

 **Set the gravity (Paramétrer la gravité)**

A l'aide de cette action, vous allez créer la gravité d'un objet particulier. Vous indiquez une direction (un angle compris entre 0 et 360 degrés) et une vitesse. A chaque étape, la vitesse indiquée est ajoutée au déplacement actuel de l'instance de l'objet et ce dans la direction spécifiée. Vous devrez utiliser normalement un très petit incrément pour la vitesse (comme 0.01). Généralement, vous souhaiterez utiliser une direction vers le bas (270 degrés). Si vous cochez la case **Relative**, vous augmenterez la vitesse de la gravité ainsi que la direction. Veuillez noter que, contrairement à la réalité, des objets (différents) peuvent avoir des directions gravitationnelles différentes.

 **Reverse horizontal direction (Inverser la direction horizontale)**

Avec cette action, vous inverserez le déplacement horizontal de l'instance. Ceci peut être utilisé par exemple lorsqu'un objet entre en collision avec un mur vertical.

 **Reverse vertical direction (Inverser la direction verticale)**

Avec cette action, vous inverserez le déplacement vertical de l'instance. Ceci peut être utilisé par exemple quand un objet entre en collision avec un mur horizontal.

 **Set the friction (Paramétrer la friction)**

La friction ralentit le déplacement des instances. Vous indiquerez ici une valeur de friction. A chaque step, la quantité indiquée sera déduite de la vitesse jusqu'à ce que celle-ci arrive à 0. Habituellement, vous mentionnerez un très petit nombre (comme 0.01).

 **Jump to a given position (Aller à une position donnée)**

En utilisant cette action, vous pourrez placer l'instance à une position particulière. Vous indiquerez simplement les coordonnées x et y et l'instance sera placée à cette position avec son point de référence. Si vous cochez la case **Relative**, la position sera relative à l'actuelle position de l'instance. Cette action est souvent employée pour déplacer une instance en continu. A chaque step, la position est incrémentée d'une valeur très infime.

 **Jump to the start position (Aller à la position de départ)**

Cette action place l'instance de nouveau à la position qu'elle avait à sa création.

 **Jump to a random position (Sauter à une position aléatoire)**

Cette action déplace l'instance à une position aléatoire de la room. Seules les positions ne présentant pas de point d'intersection avec des instances d'objets solides, seront choisies au hasard. Vous pouvez indiquer l'alignement à utiliser. Si vous spécifiez des valeurs positives, les coordonnées seront déterminées en utilisant des entiers multiples des valeurs saisies. Cela peut être utilisé par exemple pour forcer une instance à s'aligner avec les cellules de votre jeu (s'il en existe). Vous pouvez indiquer séparément un alignement horizontal et vertical.

 **Snap to grid (Aligner avec la grille)**

Avec cette action, vous pouvez positionner précisément l'instance sur la grille. Vous indiquerez à la fois la valeur d'alignement horizontale et verticale (ceci correspond à la taille des cellules de la grille). Très utile pour être sûr que les instances restent sur la grille.

 **Wrap when moving outside (Permettre le déplacement de bord à bord)**

Avec cette action, une instance pourra se déplacer de bord à bord de la room (moving wrap). Cela se produit lorsque l'instance quitte un côté de la room pour réapparaître de l'autre côté. Cette action est habituellement employée dans l'événement **Outside**. Veuillez remarquer que l'instance doit mentionner une vitesse afin que le déplacement de bord à bord puisse fonctionner, ceci en raison de la direction de déplacement de bord à bord qui est basée sur la direction de mouvement. Vous pouvez indiquer si l'on doit permettre uniquement un déplacement horizontal, vertical ou encore dans les deux directions.

 **Move to contact position (Aller à une position de contact)**

Avec cette action, vous pourrez déplacer une instance dans une direction donnée jusqu'à ce que celle-ci entre en contact avec un objet. Si l'instance est déjà en collision avec un autre objet, elle ne sera pas déplacée. Sinon, l'instance sera positionnée juste avant que la collision ne survienne. Vous pouvez indiquer la direction mais aussi une distance maximale pour le déplacement. Par exemple, lorsque l'instance tombe, vous pouvez la faire se déplacer vers le bas d'une certaine distance jusqu'à ce qu'un objet soit rencontré. Vous pourrez également indiquer si l'on doit prendre en compte uniquement les objets solides ou encore tous les objets. Vous mettrez habituellement cette action dans un événement de collision afin de vous assurer que l'instance s'arrêtera au contact d'une autre instance impliquée dans la collision.

 **Bounce against objects (Rebonds contre des objets)**

Lorsque vous placez cette action dans l'événement de collision d'un objet, l'instance de l'objet rebondit de façon naturelle. Si vous régler le paramètre 'precise' à **Faux**, seules les murs horizontaux et verticaux seront traités correctement. Si 'precise' est paramétrée à **Vrai**,

les murs inclinés (et même ceux qui sont courbés) seront également pris en compte. Ceci est cependant assez lent. Aussi, vous devrez préciser si le rebond concerne uniquement les objets solides

ou encore tous les objets. Veuillez bien noter que la gestion du rebond n'est pas complètement fonctionnelle car cela dépend de nombreuses propriétés. Cependant, dans la plupart des situations, le résultat devrait généralement être assez satisfaisant.

Actions Principales, jeu n°1

La liste d'actions suivantes traite de la création, de la modification et de la destruction d'instances d'objets comportant des sons ainsi que les actions concernant les salles.

 **Create an instance of an object (Créer une instance d'un objet)**

Avec cette action, vous pourrez créer une instance d'un objet. Vous indiquerez quel est l'objet à créer ainsi que la position de la nouvelle instance. Si vous cochez la case **Relative**, la position sera relative à la position de l'instance courante. La création d'instances durant le jeu est extrêmement utile. Un vaisseau spatial peut créer des balles; une bombe peut créer une explosion, etc. Dans la plupart des jeux, vous aurez besoin d'un objet contrôleur qui créera de temps à autre des monstres ou bien encore d'autres objets. Pour les nouvelles instances créées, c'est l'événement à la création qui est exécuté.

 **Create an instance of an object with a speed and direction (Créer une instance d'un objet avec une vitesse et une direction)**

Cette action fonctionne de la même manière que celle décrite ci-dessus mais comprend deux champs supplémentaires. Vous pouvez ici préciser la vitesse et la direction de l'instance nouvellement créée. Veuillez noter que si la case **Relative** est cochée, seule la position sera relative et non la vitesse, ni la direction. Par exemple, pour faire en sorte qu'une balle aille dans la direction d'un personnage en train de tirer, vous devrez utiliser une petite astuce. Comme position, indiquez 0,0 et cochez la case **Relative**. Nous devrons également connaître la direction actuelle de l'instance. On peut l'obtenir en utilisant le mot `direction` (ceci est une variable qui indique toujours la direction actuelle dans laquelle se déplace une instance).

 **Create instance of random object (Créer une instance d'un objet aléatoire)**

Cette action vous permet de créer une instance avec plusieurs objets (un à quatre objets). Vous précisez les quatre objets ainsi que leur position. Une instance de l'un de ces quatre objets sera créée à la position indiquée. Si vous cochez la case **Relative**, la position sera relative à la position de l'instance courante. Si vous avez besoin de choisir moins de quatre objets, vous pourrez utiliser **No Object** pour certains d'entre eux. Ceci est utile par exemple pour générer et afficher un ennemi aléatoire.

 **Change the instance (Modification de l'instance)**

Avec cette action, vous pouvez modifier l'instance actuelle en une instance d'un autre objet. Ainsi, par exemple, vous pourrez changer l'instance d'une bombe en celle d'une explosion. Tous les paramètres, comme le mouvement et les valeurs des variables, resteront les mêmes.

Vous pourrez mentionner si l'on doit exécuter ou non l'événement à la destruction pour l'objet courant et l'événement à la création pour le nouvel objet.

 **Destroy the instance (Détruire l'instance)**

Avec cette action, vous détruisez l'instance courante. L'événement à la destruction de l'instance est exécuté.

 **Destroy instances at a position (Détruire des instances à une position donnée)**

Cette action vous permet de détruire toutes les instances se trouvant dans un certain périmètre. Ceci est utile par exemple lorsque vous faites exploser une bombe. Si vous cochez la case **Relative**, la position sera déterminée en fonction de la position de l'actuelle instance.

 **Change the sprite (Changer le sprite)**

Utilisez cette action pour changer le sprite de l'instance. Vous indiquerez avec quel sprite le changement doit avoir lieu. Vous pouvez également indiquer quelle sous-image doit être affichée. Vous utiliserez ordinairement 0 (la première sous-image) à moins que vous ne désiriez voir une sous-image particulière. Utilisez -1 si vous ne souhaitez pas changer la sous-image actuellement affichée. Enfin, vous pourrez changer la vitesse de l'animation des sous-images. Si vous voulez seulement voir une sous-image particulière, réglez la vitesse à 0. Si la vitesse est supérieure à 1, des sous-images seront occultées. Si elle est plus petite que 1 alors les sous-images seront affichées plusieurs fois. Ne pas utiliser de vitesse négative. Le changement des sprites est une possibilité intéressante et importante. Par exemple, vous souhaiterez souvent modifier le sprite d'un personnage en fonction de la direction de sa marche. Ceci peut être réalisé en créant différents sprites pour chacune (quatre) des directions. A l'aide des événements clavier et des touches fléchées, vous serez en mesure de diriger la direction du déplacement mais aussi le sprite.

 **Transform the sprite (Transformer le sprite)**

Utilisez cette action pour modifier la taille et l'orientation du sprite de l'instance. Utilisez un facteur d'échelle pour l'agrandir ou le rétrécir. L'angle indique l'orientation du sprite dans le sens contraire des aiguilles d'une montre. Afin que le sprite soit orienté dans la direction du mouvement, utilisez la variable `direction`. Par exemple, ceci est très utilisé pour une voiture. Vous pourrez aussi indiquer si le sprite doit avoir un effet miroir horizontal et/ou doit être retourné verticalement.
Cette action n'est disponible que dans la version enregistrée.

 **Set sprite blending (Paramétrer le mélange de couleur du sprite)**

D'ordinaire, le sprite est dessiné comme il a été défini. Si vous utilisez cette action, il vous sera possible de changer la couleur du sprite. Cette couleur sera mélangée avec celle du sprite, c'est à dire qu'elle sera combinée avec les autres couleurs du sprite. Si vous souhaitez dessiner un sprite avec des couleurs différentes, il est préférable de définir le sprite en noir et blanc puis d'utiliser

la couleur de mélange pour déterminer la nouvelle couleur. Vous pouvez aussi employer une transparence alpha. Une valeur fixée à 1 rendra le sprite opaque. Une valeur à 0 le rendra complètement transparent. Entre ces deux valeurs, vous apercevrez partiellement le décor sous le sprite. C'est vraiment super par exemple pour la création d'explosions. ***Cette action n'est disponible que dans la version enregistrée.***

 **Play a sound (Jouer un son)**

Cette action vous permet de jouer une ressource son que vous aurez incorporée auparavant dans votre jeu. Vous choisirez le son à jouer et déterminerez s'il doit être joué une seule fois (par défaut) ou en boucle. Plusieurs sons Wave peuvent être joués en même temps mais un unique son Midi peut être joué. Ainsi, si vous jouez un autre son Midi, l'actuel son Midi sera interrompu.

 **Stop a sound (Stopper un son)**

Cette action stoppe le son indiqué. Dans le cas où il y a plusieurs instances d'un même son, toutes seront arrêtées.

 **If a sound is playing (Teste si un son est actuellement joué)**

Dans le cas où le son indiqué est en cours de lecture, l'action suivante sera exécutée. Sinon, elle sera sautée. Vous pouvez sélectionner **Not** pour spécifier que l'action suivante sera prise en compte que si le son concerné n'est pas actuellement en cours de lecture. Par exemple, vous pourrez tester si la musique de fond est toujours en train d'être jouée et dans la négative, vous pourrez lancer la lecture d'une nouvelle musique en arrière-plan. Prenez note que cette action retourne la valeur **vrai** (true) lorsque le son sort actuellement des haut-parleurs. Après l'appel de cette action pour écouter un son, ce dernier n'a peut être pas encore atteint les haut-parleurs. Aussi, l'action pourra parfois retourner une valeur **faux** (false) pendant un certain laps de temps. Le même raisonnement s'applique quand le son est stoppé : il vous sera peut-être encore possible d'entendre le son (en raison de l'écho) alors que l'action retournera la valeur **vrai**.

 **Go to previous room (Aller à la room précédente)**

Va à la room précédente. Vous pouvez indiquer le type d'effet de transition entre les rooms. A expérimenter afin de constater par vous-même ce qui vous plaît le mieux. Si vous êtes dans la première room, vous obtiendrez une erreur.

 **Go to next room (Aller à la room suivante)**

Va à la room suivante. Là encore, vous pourrez indiquer l'effet de transition.

 **Restart the current room (Réafficher la room actuelle)**

La room courante est réaffichée. Vous pourrez mentionner un effet de transition.

 **Go to a different room (Aller à une room différente)**

Avec cette action, vous vous déplacerez à une room particulière. Vous indiquerez la room et l'effet de transition.

 **If previous room exists (Teste si la room précédente existe)**

Cette action teste s'il existe une room précédente. Dans l'affirmative, l'action suivante sera exécutée. Vous devriez effectuer ce test avant de vouloir vous déplacer à une room précédente.

 **If next room exists (Teste si la room suivante existe)**

Cette action teste s'il existe une room suivante. Si oui, la prochaine action sera réalisée. Vous devriez effectuer ce test avant de vouloir vous déplacer à la prochaine room.

Actions Principales, jeu n° 2

Vous trouverez ici la description des nouvelles actions principales ayant trait au timing, retournant des messages à l'utilisateur ou encore concernant les jeux dans leur ensemble.

 **Set an alarm clock (Paramétrer une alarme chronomètre)**

Cette action vous permet de régler l'une des douze alarmes chronomètres disponibles pour une instance. Vous devrez sélectionner le nombre de steps et le n° de l'alarme chronomètre. Après expiration du nombre de steps indiqués, l'instance recevra un signal de l'événement d'alarme correspondant. Vous pouvez aussi augmenter ou diminuer la valeur en cochant la case **Relative**. Si vous réglez l'alarme chronomètre à une valeur négative, vous suspendrez cette alarme. L'événement ne sera alors jamais généré.

 **Sleep for a while (Attendre un certain temps)**

Cette action vous permet de geler une scène pendant un certain temps. On l'utilise généralement au début ou à la fin d'un niveau ou lorsque l'on affiche un message au joueur. Vous mentionnerez le nombre de millisecondes à attendre. Vous pourrez aussi indiquer si l'écran doit être réaffiché afin de tenir compte des dernières modifications graphiques effectuées.

 **Display a message (Afficher un message)**

Avec cette action, vous pouvez afficher un message dans une boîte de dialogue. Vous devrez simplement taper le message. Si vous utilisez le caractère # dans le corps du message, il sera interprété comme un symbole de retour à la ligne (utilisez alors \# pour afficher le caractère # lui-même). Si le texte du message débute avec une simple ou double quotes, il sera interprété en tant qu'expression. Voir plus loin pour obtenir davantage d'informations sur les expressions.

 **Show the game information (Afficher les informations sur le jeu)**

Cette action fait apparaître la fenêtre d'informations concernant le jeu.

 **Restart the game (Relancer le jeu)**

Vous pouvez relancer le jeu depuis le début avec cette action.

 **End the game (Fin du jeu)**

Cette action vous permet de quitter le jeu.

 **Save the game (Sauver le jeu)**

Avec cette action, vous avez la possibilité de sauvegarder les principaux paramètres du jeu

en cours. Vous devrez mentionner le nom du fichier de sauvegarde (le fichier sera créé dans le répertoire de travail du jeu). Ultérieurement, le jeu pourra être rechargé en utilisant l'action

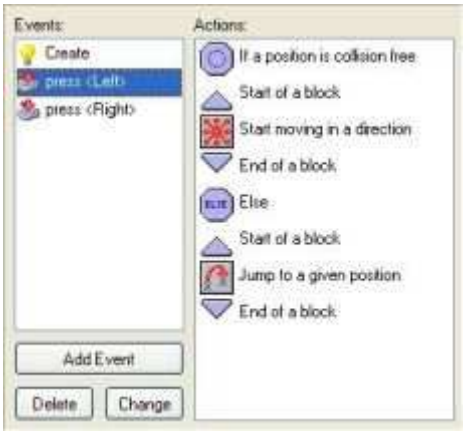
décrite ci-dessous (seul le statut de base du jeu est sauvé. Les choses qui par exemple ne sont pas sauvegardées, concernent les sons en cours et certains aspects avancés comme le contenu des structures de donnée, les particules, etc.).

 **Load the game (Charger le jeu)**

Cette action charge le statut du jeu à partir d'un fichier. Vous devrez indiquer le nom du fichier de sauvegarde. Soyez sûr que le jeu de sauvegarde indiqué correspond bien au jeu concerné et ait été généré avec la même version de *Game Maker*. Sinon, une erreur surviendra (pour être plus précis, le jeu est chargé à la fin du step courant. Aussi, certaines actions après celui-ci sont encore exécutées dans le jeu actuel, et non pas dans celui venant d'être chargé !).

Actions de contrôle

Il y a un certain nombre d'actions avec lesquelles vous pouvez contrôler quelles autres actions doivent s'exécuter. La plupart de ces actions posent une question, par exemple si une position de l'écran est libre. Si la réponse est **oui** (true), l'action suivante est exécutée, sinon, l'action est sautée. Si vous souhaitez avoir plusieurs actions exécutées ou sautées selon le résultat, vous pouvez les placer dans un bloc d'instructions délimité par des actions de début et de fin de blocs. Il peut y avoir également une instruction **else** qui sera exécutée lorsque la réponse à la question est **non**. Ceci est un exemple typique :



La question posée ici teste si la position de l'instance actuelle est libre de collision. Si **oui** alors l'instance se déplacera dans la direction indiquée. Si ce n'est pas le cas, l'instance sautera à une position indiquée dans le bloc **else**.

Pour toutes les questions, il existe un champs nommé **NOT**. Si vous cochez ce champs, le sens de la question est inversé. C'est à dire que si par exemple la réponse à la question était **vrai**, alors elle deviendra **faux** et vice-versa.. Cela vous permet d'exécuter certaines actions dans le cas où le résultat de la question ne serait **pas vrai**.

A la plupart des questions, vous pouvez indiquer ce qu'il devrait s'appliquer à toutes les instances d'un objet particulier. Dans ce cas, le résultat sera **vrai** si et seulement si le résultat est également **vrai** pour toutes les instances de l'objet. Par exemple, vous pouvez vérifier si la position légèrement à droite d'un ensemble de balles est libre de collision.

Les questions et actions suivantes sont disponibles (veuillez noter qu'elles ont toutes une représentation sous forme d'icône et une couleur de fond différentes ce qui permet de les distinguer plus facilement entre elles).

 **If a position is collision free (Test si une position n'est pas en collision)**

Cette question retourne **vrai** pour l'instance courante, placée à la position indiquée, si celle-ci ne génère pas de collision avec un objet. Vous pouvez indiquer la position sous forme absolue ou relative. Vous pouvez aussi mentionner si seuls les objets solides ou encore tous les objets doivent être pris en compte. Cette action est habituellement employée pour vérifier si l'instance peut se déplacer à un endroit précis.

 **If there is a collision at a position (Test s'il y a collision à cette position)**

C'est l'action inverse de la précédente. Elle retourne **vrai** si l'instance courante, placée à une position donnée, entre en collision avec une autre instance ou objet (toujours soit avec des objets solides uniquement ou encore avec tous les objets).

 **If there is an object at a position (Test si présence d'un objet à une position donnée)**

Cette question retourne **vrai** si l'instance placée à une position donnée, rencontre une instance de l'objet indiqué.

 **If the number of instances is a value (Test si le nombre d'instances est égal à une certaine valeur)**

Vous devez préciser un objet et un nombre. Si le nombre actuel d'instances de cet objet est égal au nombre donné alors la question retourne **vrai** sinon elle retournera **faux**. Vous pouvez également effectuer le test si le nombre d'instances est plus petit ou encore plus grand que la valeur fournie. On utilise ces tests afin de savoir si toutes les instances d'un type particulier ne sont plus présentes dans le jeu. Auquel cas, ce sera souvent le moment de terminer le niveau ou le jeu.

 **With a chance perform next action (Avec une certaine probabilité d'effectuer la prochaine action)**

Vous indiquez le nombre souhaité de côtés d'un dé qui sera ensuite lancé. Si le résultat du lancement du dé est 1 alors il sera retourné **vrai** (true) et la prochaine action sera exécutée. Ceci est utile pour ajouter des actions aléatoires dans votre jeu. Par exemple, à chaque step, il vous est possible de générer une bombe ou un changement de direction, le tout avec une certaine probabilité que l'événement se réalise. Plus le nombre de faces du dé est grand, moins il y aura de chance que cela se réalise. Actuellement, vous pouvez utiliser des nombres réels. Par exemple, si vous paramétrez le nombre de côtés à 1.5, la prochaine exécution sera exécutée deux fois sur trois. Utiliser un nombre inférieur à 1 n'a ici pas de sens (NDT : en effet, mathématiquement parlant, cela reviendrait à dire que l'événement aurait plus de chance de se réaliser que l'occurrence observée de l'événement ce qui n'est pas possible).

 **If the user answers yes to a question (Si l'utilisateur répond oui à une question)**

Vous posez une question. Une boîte de dialogue s'affiche avec un bouton **yes** (oui) et un bouton **no** (non). Le résultat sera **vrai** (true) si l'utilisateur a répondu **yes** (oui).

 **If an expression is true (Si une expression est vérifiée)**

C'est la condition la plus générale mais aussi la plus utilisée. Vous indiquerez ici une expression arbitraire. Si l'expression renvoie la valeur **vrai** (true) (par exemple si un nombre est supérieur ou égal à 0.5), l'action suivante sera réalisée. Voir ci-dessous pour obtenir plus d'informations sur les expressions.

 **If a mouse button is pressed (Si un bouton de la souris a été pressé)**

Retourne **vrai** (true) si le bouton de souris indiqué est pressé. On utilise fréquemment ce test dans l'événement step. Vous pouvez vérifier si un bouton de la souris a été pressé et dans l'affirmative, déplacer une instance à la position de la souris (utilisez l'action "Aller à une position donnée" (jump to a given position) avec les valeurs `mouse_x` et `mouse_y`).

 **If instance is aligned with grid (Test si une instance est alignée avec la grille)**

Retourne **vrai** (true) si la position de l'instance tient sur une grille. Vous indiquerez l'espacement horizontal et vertical de la grille. Cela peut être très utile lorsque certaines actions, comme faire un demi-tour, ne sont possibles que si l'instance se trouve dans une position de la grille.

 **Start of block (Début de bloc)**

Indique le début d'un bloc d'actions.

 **End of block (Fin de bloc)**

Indique la fin d'un bloc d'actions.

 **Else (Sinon)**

Après cette action, il y a les instructions suivant la clause **else**, qui seront exécutées quand le résultat à la question retourne **faux** (false).

 **Repeat next action (Répéter la prochaine action)**

Cette action est employée pour répéter l'action suivante (ou bloc d'actions) un certain nombre de fois. Vous indiquerez simplement ce nombre.

 **Exit the current event (Quitter l'événement courant)**

Quand cette action est utilisée, aucune autre action de cet événement ne sera désormais exécutée. On l'utilise généralement après une question. Par exemple, lorsqu'une position est libre, il n'est pas

nécessaire de faire quelque chose de particulier donc on quitte l'événement. Dans l'exemple donné, les actions suivantes ne seraient exécutées qu'en présence d'une collision.

Si vous souhaitez davantage de contrôle sur le fonctionnement de votre jeu, vous pourrez utiliser le langage de programmation intégré (GML), décrit dans la 4ième partie de cette documentation. Il vous apportera plus de flexibilité que l'emploi d'actions via l'interface graphique du logiciel. Il est d'utilisation facile mais surtout, il vous permettra l'usage de vos propres variables. Les actions suivantes ont trait à ce langage :

 **Execute a piece of code (Exécuter du code)**

Lorsque vous ajoutez cette action, une fenêtre apparaît à l'écran dans laquelle vous pourrez taper des lignes de code qui seront ultérieurement exécutées. Cela peut être de simples appels de fonction ou encore du code plus complexe. L'utilisation des actions sous forme de code n'est envisageable que pour les petits programmes. Pour les plus gros projets, il est fortement recommandé d'utiliser les scripts qui font l'objet d'une description dans la 2ième partie de ce document.

 **Comment (Commentaire)**

Utilisez cette action pour ajouter une ligne de commentaires à la liste d'actions. La ligne est affichée en ***italique***. L'ajout de commentaires vous aidera à vous souvenir de ce que doivent faire vos événements. Cette action ne fait rien de particulier mais sachez que c'est tout de même une action. Ainsi, si vous placez l'action après une action conditionnelle, ce sera l'action commentaire qui sera exécutée dans le cas où la condition serait **vraie** (true) (bien qu'elle ne fasse rien du tout).

 **Set the value of a variable (Fixer la valeur d'une variable)**

Il existe de nombreuses variables intégrées dans le jeu. Avec cette action, vous pourrez les modifier. Vous pourrez aussi créer vos propres variables et leur assigner des valeurs. Vous indiquerez le nom de la variable et la nouvelle valeur. Si vous cochez la case **Relative**, la valeur sera ajoutée à la valeur actuelle de la variable. Veuillez noter que ceci n'est possible que si la variable a déjà eu une valeur qui lui ait été affectée ! Voir ci-dessous pour en savoir davantage sur les variables.

 **If a variable has a value (Test si une variable a une certaine valeur)**

Cette action vous permet de tester la valeur d'une variable particulière. Si la valeur de la variable est égale au nombre indiqué, la question retournera **vrai** (true). Sinon, elle donnera **faux** (false). Vous pouvez également tester si la valeur d'une variable est inférieure ou supérieure à une valeur mentionnée. Voir plus bas pour obtenir plus d'informations au sujet des variables. Actuellement, vous pouvez également utiliser cette action pour comparer deux expressions.

Draw the value of a variable (Afficher la valeur d'une variable)

Avec cette action, vous pourrez afficher la valeur d'une variable à un endroit précis de l'écran.

Veuillez noter que cette action ne peut être utilisée que dans l'événement d'affichage d'un objet.

Actions de Gestion du Score

Dans la plupart des jeux, le joueur devra obtenir un certain score. Aussi, les jeux offrent généralement quelques vies. De plus, le joueur a souvent une santé donnée. Les actions suivantes rendent plus commode la gestion du score, des vies et de la santé du joueur.

 **Set the score (Donner une valeur au score)**

Game Maker comprend un mécanisme interne de gestion des scores. Le score est d'ordinaire affiché dans une fenêtre dédiée à cet usage. Vous pouvez utiliser cette action pour changer le score. Il vous suffira simplement de fournir la nouvelle valeur du score. Souvent, vous souhaiterez incrémenter le score. Dans ce cas, n'oubliez pas de cocher la case **Relative**.

 **If score has a value (Test si le score est égal à une certaine valeur)**

Avec cette action basée sur une question, vous pouvez vérifier si le score a atteint une valeur particulière. Vous indiquerez la valeur à comparer et si le score doit être égal, inférieur ou supérieur à cette valeur.

 **Draw the value of score (Afficher le score)**

Cette action vous permet d'afficher la valeur actuelle du score à un endroit précis de l'écran. Vous fournirez les positions et le motif devant être placés en face du score. Le score sera affiché dans la police de caractères courante. Cette action ne peut être utilisée que dans l'événement d'affichage d'un objet.

 **Display the highscore table (Afficher la liste des plus hauts-scores)**

Pour chaque jeu, les meilleurs scores sont conservés. Cette action affiche la liste des plus hauts-scores. Si le score actuel figure parmi les scores les plus élevés, le nouveau score sera inséré et le joueur pourra taper son nom. Vous pouvez préciser quelle image de fond doit être utilisée, si la fenêtre doit comporter une bordure, dans quelles couleurs s'afficheront la nouvelle entrée ainsi que les autres entrées du score et quelle police de caractères devra être utilisée.

 **Clear the highscore table (Effacer la liste des plus hauts-scores)**

Cette action efface la liste des meilleurs scores.

 **Set the number of lives (Paramétrer le nombre de vies)**

Game Maker comporte également un système intégré de gestion des vies. Cette action vous permet de modifier le nombre de vies disponibles au début du jeu. Habituellement, la valeur est fixée à 3 au lancement du jeu et augmente ou diminue selon la performance du joueur.

Cependant, n'oubliez pas de cocher la case **Relative** si vous souhaitez pouvoir augmenter ou diminuer le nombre de vies. Si le nombre de vies atteint (ou est plus petit que) 0, un événement "no more lives" (plus de vie) sera généré.

 **If lives is a value (Test si le nombre de vies a atteint une certaine valeur)**

Avec cette action sous forme de question, vous pouvez vérifier si le nombre de vies a atteint une certaine valeur. Vous indiquerez la valeur mais également si le nombre de vies doit être égal, inférieur ou supérieur à cette valeur.

 **Draw the number of lives (Afficher le nombre de vies)**

Cette action vous permet d'afficher le nombre de vies à un endroit précis de l'écran. Vous fournirez les positions et le motif à placer en face du nombre de vies. Le nombre de vies sera affiché dans la police de caractères courante. Cette action ne peut être utilisée que dans l'événement d'affichage d'un objet.

 **Draw the lives as image (Afficher les vies sous forme d'image)**

Plutôt que d'afficher le nombre de vies sous la forme d'un nombre, il peut être plus agréable graphiquement d'utiliser quelques petites images à cet effet. Cette action réalise précisément cela. Vous indiquerez la position et l'image souhaitée. Ainsi, le nombre de vies sera affiché avec des images. Cette action ne peut être utilisée que dans l'événement d'affichage d'un objet.

 **Set the health (Régler la santé du joueur)**

Game Maker comprend un mécanisme interne de gestion de la santé du joueur. Vous pourrez utiliser cette action pour modifier la santé. Une valeur de 100 indique une santé maximale alors que 0 signifie plus de santé du tout. Vous fournirez simplement la nouvelle valeur pour la santé. Souvent, vous désirerez diminuer la santé du joueur. Dans ce cas, n'oubliez pas de cocher la case **Relative**. Lorsque la santé sera inférieure ou égale à 0, un événement **plus de santé** apparaîtra.

 **If health is a value (Test si la santé a atteint une certaine valeur)**

Avec cette action sous forme de question, vous pourrez vérifier si la santé du joueur a atteint une valeur particulière. Vous mentionnerez la valeur et si la santé doit être égale, inférieure ou supérieure à cette valeur.

 **Draw the health bar (Afficher la barre de santé)**

Il vous sera possible avec cette action d'afficher le niveau de santé sous forme de barre de santé. Lorsque la santé est de 100, une barre de santé pleine sera affichée. Par contre, si la santé est à 0, la barre sera vide. Vous indiquerez la position et la taille de la barre de santé ainsi que les couleurs de la barre et du fond.

 **Set the window caption information (Paramétrer les informations de la fenêtre de jeu)**

D'ordinaire, le nom de la room ainsi que le score sont affichés dans le titre de la fenêtre de jeu. Cette action vous permettra de changer tout ceci. Vous indiquerez si oui ou non l'on doit afficher le score, les vies ou la santé et le motif pour chacun de ces éléments.

Actions d'Affichage

Habituellement, à chaque step du jeu et pour chaque instance, les sprites sont redessinés dans la room. Vous pouvez modifier ceci en plaçant les actions dans l'événement d'affichage (veuillez noter que ces actions seront exécutées uniquement lorsque l'instance est visible !). Les actions suivantes d'affichage sont disponibles. Ces actions ne peuvent être utilisées que dans les événements d'affichage. Elles seront ignorées dans les autres événements.

 **Draw a sprite image (Affichage d'une image sprite)**
Vous indiquerez le sprite, sa position (soit en absolu ou relative à la position de l'instance courante) et les sous-images du sprite (les sous-images sont numérotées à partir de 0). Si vous souhaitez dessiner l'actuelle sous-image, utilisez alors le numéro -1.

 **Draw a background image (Dessiner une image de fond)**
Vous indiquerez l'image de fond, sa position (absolu ou relative) et si l'image doit remplir ou non toute la room à l'aide de tuile (tile).

 **Draw a text (Afficher du texte)**
Vous mentionnerez le texte et sa position. Un symbole # dans le texte sera interprété comme un caractère de saut de ligne (utilisez alors \# pour pouvoir utiliser le symbole #). Vous pourrez ainsi créer du texte tenant sur plusieurs lignes. Si le texte débute par une simple ou double quotes, il sera interprété comme une expression. Par exemple, vous pouvez utiliser

```
'X: ' + string(x)
```

pour afficher la valeur de la coordonnée **x** de l'instance (la variable **x** mémorise la coordonnée actuelle de x. La fonction `string()` transforme ce nombre en une chaîne puis les deux chaînes sont concaténées).

 **Draw a text transformed (Affiche un texte avec effet de transformation)**
Cette action est similaire à la précédente action mais cette fois, vous pourrez indiquer également un facteur d'échelle horizontale et verticale pour changer la taille du texte ainsi que l'angle de rotation de celui-ci. ***Cette action n'est disponible que dans la version enregistrée.***

 **Draw a rectangle (Dessine un rectangle)**

Vous indiquerez les coordonnées des deux coins opposés du rectangle, soit en absolu ou relatif à la position de l'instance courante.

 **Draw a horizontal gradient (Dessine un rectangle avec couleurs gradient horizontales)**

Cette action dessine aussi un rectangle mais cette fois-ci en utilisant une couleur gradient changeante de la gauche vers la droite. Vous spécifierez le rectangle et les deux couleurs à utiliser. ***Cette action n'est disponible que dans la version enregistrée.***

 **Draw a vertical gradient (Dessine un rectangle avec couleurs gradient verticales)**

Cette action dessine également un rectangle mais utilise une couleur gradient changeante du haut vers le bas. Vous indiquerez le rectangle et les deux couleurs à employer. ***Cette action n'est disponible que dans la version enregistrée.***

 **Draw an ellipse (Dessine une ellipse)**

Cette action dessine une ellipse. Vous mentionnerez les cordonnées des deux coins opposés de la bordure du rectangle, soit en absolu ou relatif à la position de l'instance courante.

 **Draw a gradient ellipse (Dessine une ellipse gradient)**

De nouveau, une ellipse est affichée mais ici, vous indiquerez une couleur pour le centre et une autre pour le bord. ***Cette action n'est disponible que dans la version enregistrée.***

 **Draw a line (Dessine une ligne)**

Vous mentionnerez les coordonnées des deux points opposés de la ligne, soit en absolu ou relatif à la position de l'instance courante.

 **Draw an arrow (Dessine une flèche)**

Dessine une flèche. Vous indiquerez les coordonnées des deux points opposés de la ligne ainsi que la taille de la flèche.

 **Set the colors (Choisir les couleurs)**

Vous permet de fixer la couleur à utiliser pour dessiner des formes, des lignes et du texte (ceci n'influence pas la façon dont les sprites et les arrière-plans sont affichés).

 **Change fullscreen mode(Changer le mode d'affichage de l'écran)**

Avec cette action, vous pourrez permuter l'affichage soit en mode fenêtré ou plein écran et vice-versa. Vous indiquerez si vous voulez basculer directement en mode plein écran ou si vous souhaitez permuter entre le mode fenêtré ou plein écran.

 **Take a snapshot image of the game (Effectuer une copie d'écran du jeu)**

Cette action vous permet de faire une copie d'écran du jeu qui sera sauvegardée dans un fichier BMP. Vous indiquerez le nom de fichier de sauvegarde de la copie d'écran. ***Cette action n'est disponible que dans la version enregistrée.***

 **Create an effect (Créer un effet)**

Cette action vous permettra de créer très simplement toutes sortes d'effets. Vous spécifierez le type d'effet, par exemple une explosion ou de la fumée, sa position, sa taille, sa couleur et si l'effet doit être visible derrière ou devant les objets. Pour le reste, tout est géré automatiquement (pour la pluie et la neige, la position ne peut être indiquée car de toute façon, la neige ou la pluie tombe toujours du haut de l'écran. Pour obtenir un effet de pluie régulier, vous devrez créer cet effet à chaque step). ***Cette action n'est disponible que dans la version enregistrée.***

Utilisation d'expressions et de variables

Dans la plupart des actions, vous devrez fournir des valeurs comme paramètres. Plutôt que de taper juste un nombre, vous pouvez aussi taper une formule, par exemple : 32*12. Mais vous vous pouvez également taper des expressions beaucoup plus complexes. Par exemple, si vous désirez doubler la vitesse horizontale, vous pourrez écrire : `2*hspeed`. Ici, `hspeed` est une variable indiquant la vitesse horizontale actuelle de l'instance. Il existe un grand nombre d'autres variables que vous pouvez utiliser. Les plus importantes sont les suivantes :

x la coordonnée x de l'instance
y la coordonnée y de l'instance
hspeed la vitesse horizontale (en pixels par step)
vspeed la vitesse verticale (en pixels par step)
direction l'actuelle direction du déplacement en degrés (0-360)
speed la vitesse actuelle dans la direction courante
visible indique si un objet est visible (1) ou invisible (0)
image_index cette variable indique quelle sous-image du sprite courant est actuellement affichée. Si vous la changez et réglez la vitesse à 0 (voir plus bas), vous afficherez une sous-image fixe.
image_speed cette variable indique la vitesse à laquelle les sous-images sont affichées. La valeur par défaut est de 1. Si vous augmentez cette valeur de 1 alors certaines sous-images seront occultées afin de rendre l'animation plus rapide. Si vous la diminuez de 1 alors l'animation sera plus lente en répétant certaines sous-images.
score la valeur actuelle du score
lives le nombre actuel de vies restantes
health l'actuelle santé (0-100)
mouse_x la position en x de la souris
mouse_y la position en y de la souris

Vous pouvez modifier la plupart de ces variables en utilisant l'action **set variable** (affecter une valeur à une variable). Vous pouvez également définir vos propres variables en leur affectant une valeur (ne pas utiliser l'option relative car les variables n'existent pas encore !). Ensuite, vous pourrez utiliser ces variables dans des expressions. Les variables que vous créez sont locales à l'instance courante. C'est à dire que chaque objet possède sa propre copie de la variable créée. Pour créer une variable globale, placez le mot **global** et un **point** devant la variable.

Vous pouvez également vous référer aux valeurs des variables appartenant à d'autres objets, en mettant le nom de l'objet puis un point devant les variables. Ainsi par exemple, si vous souhaitez qu'une balle se déplace à l'endroit où se trouve l'objet **coin**, vous indiquerez (`coin.x` , `coin.y`). Dans le cas d'un événement de collision, vous pouvez mentionner la coordonnée x de l'autre objet en utilisant `other.x`. Dans les expressions conditionnelles, vous pourrez utiliser des opérateurs de comparaison comme < (**smaller than**), >, etc.

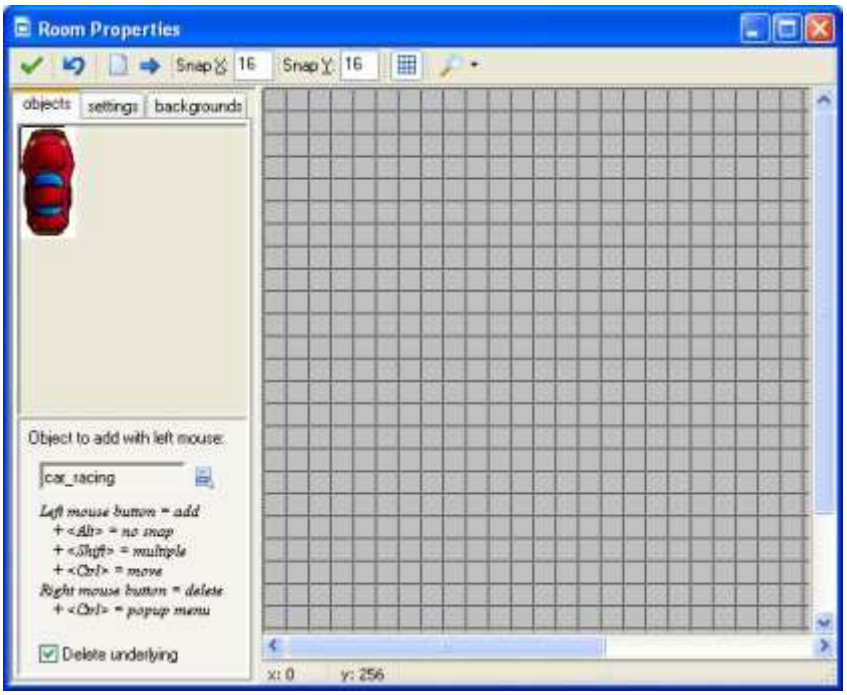
Dans vos expressions, il est également possible d'employer des fonctions. Par exemple, la fonction `random(10)` retourne un nombre réel aléatoire inférieur à 10. Ainsi, vous pourrez régler la vitesse ou la direction du mouvement à une valeur aléatoire. Beaucoup d'autres fonctions existent. Elles seront décrites dans la 4ième partie de cette documentation.

Création de rooms

Maintenant que vous avez défini les objets ainsi que leurs comportements dans la fenêtre d'événements et d'actions, il est temps de créer les rooms (ou niveaux) dans lesquelles le jeu va se dérouler. Tout jeu nécessite au moins une room. Dans ces rooms, nous placerons les instances des objets. Une fois le jeu lancé, la première room sera affichée et les instances appartenant à cette room s'animeront, ceci en raison des actions créés dans les événements lors de la création des instances.

Les possibilités sont très grandes en ce qui concerne la création de rooms. En plus de pouvoir paramétrer un certain nombre de propriétés et d'ajouter des instances pour les objets, vous pouvez aussi ajouter des décors (backgrounds), définir des vues et ajouter des tuiles (tiles). La plupart de ces options seront abordées ultérieurement. Dans ce chapitre, nous nous concentrerons sur les réglages de base, l'ajout d'instances pour les objets et le paramétrage des images pour agrémenter de décor.

Pour créer une room (salle), choisissez **Add Room** à partir du menu **Add**. L'écran suivant apparaîtra alors :



En haut de l'écran se trouve la barre d'outils. A l'aide de celle-ci, vous pourrez indiquer la taille des cellules utilisées dans la grille pour l'alignement des objets. Vous pourrez aussi mentionner si oui ou non l'on doit afficher les lignes de la grille et les décors (backgrounds), etc. Il est parfois utile

de cacher temporairement certains aspects de la room. Cependant, veuillez bien comprendre que les instances d'objets seront toujours affichées lorsque vous les ajouterez sur la room, et ceci indépendamment du paramétrage de la vue (view). Des boutons existent aussi pour effacer les instances de la room et pour décaler les instances d'un certain nombre de pixels. Utilisez des nombres négatifs pour le décalage des instances vers la gauche ou vers le haut. Cela peut être utile quand par exemple, vous décidez d'agrandir la room (vous pouvez également utiliser cette technique pour placer les instances en dehors de la room, ce qui peut parfois s'avérer très utile). Enfin, il y a le bouton **Undo** pour annuler la dernière opération effectuée sur la room et le bouton **OK** pour sauver les changements (cliquez sur la croix en haut à droite pour fermer la fenêtre sans sauvegarder les modifications réalisées).

A gauche vous verrez trois onglets (cinq dans le mode avancé). L'onglet **objects** sert lorsque vous ajoutez des instances d'objets dans la room. Dans l'onglet **settings**, vous indiquerez un certain nombre de réglages concernant la room. Dans l'onglet **backgrounds**, vous pourrez indiquer les images de fonds à afficher dans la room.

Ajout d'instances

A droite de l'écran de conception de la room, vous apercevrez le contenu de la room. Au départ, elle sera vide, avec un fonds gris.

Pour ajouter des instances dans la room, vous devrez en premier lieu sélectionner l'onglet **objects** si toutefois, le contenu de ce dernier n'était pas déjà visible. Ensuite, choisissez l'objet que vous souhaitez ajouter en cliquant sur le petit bouton représentant une icône d'un menu (ou en cliquant à gauche sur la zone de l'image représentant l'objet). L'image de l'objet apparaît à gauche (veuillez noter que si vous changez l'origine du sprite, une croix apparaîtra sur l'image.

Ceci indique de quelle manière les instances seront alignées sur la grille). Maintenant, cliquez avec le bouton gauche de la souris dans l'aire de la room visible sur la droite. Une instance de l'objet apparaît. Celle-ci sera positionnée dans l'alignement de la grille. Si vous maintenez pressée la touche <Alt> pendant que vous placez l'instance, cette dernière ne sera pas alignée dans la grille. Si vous maintenez appuyé le bouton de la souris tout en plaçant l'instance dans la room, vous déplacerez l'instance à l'endroit précis souhaité. De plus, si vous maintenez pressée la touche <Shift> tout en maintenant appuyé le bouton de la souris et en déplaçant cette dernière, plusieurs instances seront ajoutées. A l'aide du bouton droit de la souris, vous pourrez supprimer des instances. Ainsi, vous allez pouvoir définir précisément le contenu de la room.

Comme vous l'avez sans doute déjà remarqué, si vous placez une instance par-dessus une autre, l'instance originale disparaîtra. D'ordinaire, c'est ce que vous souhaiterez faire mais pas toujours. On peut contourner ceci en décochant la case **Delete underlying** (Effacer l'image en dessous) située en bas à gauche.

Si vous voulez changer la position d'une instance, maintenez pressée la touche <Ctrl> et cliquez avec le bouton gauche de la souris sur l'instance tout en maintenant appuyé ce bouton. Vous pouvez maintenant déplacer l'instance à sa nouvelle position (utilisez <Alt> pour un positionnement précis).

Si vous maintenez appuyée la touche <Ctrl> tout en cliquant sur une instance avec le bouton droit de la souris, un menu apparaîtra alors où vous pourrez effacer l'objet, indiquer une position précise pour cette instance ou encore placer l'instance d'arrière-plan en avant-plan et vice-versa.

Paramétrage de la room

Chaque room possède un certain nombre de paramètres que vous pourrez changer en cliquant sur l'onglet **settings**.

Chaque room possède un nom. L'idéal sera de donner un nom significatif et parlant pour vous. La room a aussi un titre. Ce titre est affiché dans l'entête de la fenêtre lorsque le jeu est lancé. Vous pouvez régler la largeur et la hauteur de la room (en pixels). Vous pourrez également fixer la vitesse du jeu. Celle-ci correspond au nombre de steps par seconde. Plus la vitesse sera grande et plus les déplacements seront fluides. Mais il vous sera alors nécessaire de disposer d'un ordinateur assez puissant pour lancer le jeu.

Paramétrer le décor (background)

Avec l'onglet **backgrounds**, vous pourrez donner une image de fonds pour la room. Actuellement, il vous est possible de spécifier plusieurs décors (backgrounds) pour le jeu. Le contenu de l'onglet se présente ainsi :

En haut, vous verrez la couleur du décor. Cliquez dessus pour la modifier. La couleur du fonds ne sera à utiliser que si vous n'employez pas d'image pour couvrir la totalité de la room. Sinon, il vaudra mieux décocher la case **Draw background color** afin de ne pas gaspiller de temps lors de l'exécution du jeu.

En haut, vous pourrez voir une liste de 8 décors possibles. Vous définirez chacun d'entre eux mais la plupart du temps, vous n'en aurez besoin que d'un seul ou deux tout au plus. Pour définir un décor, sélectionnez le tout d'abord dans la liste puis cochez la case **Visible when room starts** sinon vous ne pourrez pas le voir dans le jeu. Le nom du décor sera affiché en gras quand il aura toutefois été défini. Maintenant, indiquez une image de fonds à partir du menu. Certains paramètres pourront être changés. Premièrement, vous pourrez mentionner si l'image de fonds doit remplir la room dans le sens horizontal et/ou vertical. Vous pourrez aussi indiquer la position du décor dans la room (ceci a une influence sur la disposition des tuiles). Une autre option est d'élargir le décor. Le décor sera alors mis à l'échelle afin de remplir complètement la room. Le ratio d'aspect de l'image ne sera pas respecté. Enfin, vous pouvez indiquer un scrolling pour le décor en précisant une vitesse horizontale ou verticale (pixels par step). Il est préférable de ne pas utiliser le scrolling pour un décor comprenant une image élargie. Le résultat pourrait être quelque peu surprenant.

Il existe aussi une autre case à cocher nommée **Foreground image**. Lorsque vous cochez cette case, le décor (background) deviendra un avant-plan qui sera dessiné par-devant tous les objets plutôt que derrière. Il est évident qu'une telle image devra être partiellement transparente pour être d'une quelconque utilité.

Distribution de votre jeu

Les informations des précédents chapitres vous permettent de créer vos jeux. Lorsque votre jeu sera terminé, vous souhaiterez certainement que d'autres personnes puissent y jouer. Vous pourriez bien sûr donner le fichier '.gm6' que vous avez créé et laisser les gens y jouer à l'aide du logiciel *Game Maker* mais cela ne sera certainement pas ce que vous voudrez faire. Premièrement, vous ne désirerez peut-être pas que d'autres personnes puissent modifier votre jeu et deuxièmement, vous souhaiterez certainement que les gens puissent jouer à votre jeu sans disposer de *Game Maker*. Vous devrez alors créer un programme exécutable autonome de votre jeu.

La création d'exécutables autonomes est très facile avec *Game Maker*. Dans le menu **File**, vous choisirez l'option **Create Executable**. Il vous sera demandé un nom pour l'exécutable qui contiendra le jeu. Indiquez un nom, pressez **OK** et vous obtiendrez un jeu autonome que vous pourrez donner ensuite à qui vous voudrez. Vous pourrez aussi changer l'icône de ce jeu en allant dans **Global Game Settings**.

Une fois votre jeu autonome créé en utilisant la méthode décrite ci-dessus, vous pourrez donner ce fichier à d'autres personnes ou encore le mettre sur votre site web pour y être téléchargé. Vous êtes totalement libre de distribuer les jeux que vous aurez créés avec *Game Maker*. Vous avez même la possibilité de les vendre. Ceci suppose néanmoins que les sprites, les images et les sons utilisés peuvent être distribués ou vendus sans enfreindre la loi sur les droits d'auteurs. Veuillez consulter l'accord de licence pour plus d'informations à ce sujet.

Il est généralement utile de compacter votre exécutable, accompagné de sa documentation et de son fichier 'readme' (lisez-moi). Sous Windows XP, ceci peut directement être réalisé avec le menu apparaissant lors du clic du bouton droit de la souris. De plus, il existe de nombreux utilitaires de compactage (zip utilities) téléchargeables sur le web. En option, vous pouvez également créer un programme d'installation de votre jeu. De même, de nombreux programmes gratuits de création de packages d'installation peuvent être trouvés sur le web.

Mode avancé

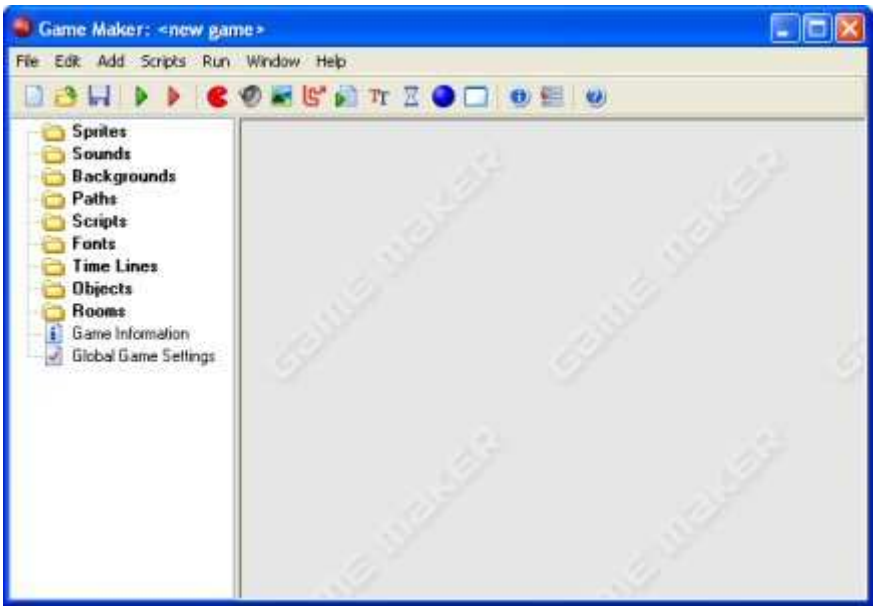
Cette section du fichier d'aide traite des informations concernant les aspects les plus avancés de *Game Maker*.

De l'information détaillée sur une utilisation plus avancée de *Game Maker* peut être trouvée dans les pages suivantes :

Mode avancé

Jusqu'à maintenant, nous n'avons fait qu'aborder les fonctions simples de *Game Maker*. Mais il existe beaucoup d'autres possibilités. Pour être pleinement en mesure de les utiliser, vous devrez lancer *Game Maker* dans le mode avancé. Pour se positionner dans ce mode, allez dans le menu **File** puis cliquez sur l'article de menu **Advanced mode** (il est nécessaire de relancer *Game Maker* ou tout du moins sauvegarder votre jeu puis le recharger, afin de pouvoir désormais utiliser les nouvelles fonctions du mode avancé).

Lorsque vous lancez *Game Maker* dans le mode avancé, l'écran suivant apparaît :



Ce mode présente, en plus des fonctions du mode simple, de nouvelles ressources, de nouveaux boutons et articles de menu. Comme nous le verrons dans les chapitres qui suivent, les ressources supplémentaires possèdent des options additionnelles. Nous détaillerons ici les nouveaux articles des menus.

Menu Fichier

Dans le menu fichier, vous trouverez les commandes supplémentaires suivantes :

- **Fusionner deux jeux.** Avec cette commande, vous pourrez fusionner toutes les ressources (sprites, sons, objets, rooms, etc.) d'un jeu dans l'actuel jeu en cours de développement. Ceci peut être très utile dans le cas où vous souhaiteriez réutiliser des parties de code ou de ressources dans votre jeu (ex: systèmes de menu).

Veuillez cependant noter que toutes les ressources, les instances et les tuiles (tiles) se verront affecter un nouvel ID, ce qui pourra poser des problèmes lors de l'utilisation de celles-ci dans des scripts. Il est de votre ressort de vérifier que les ressources des deux fichiers possèdent des noms différents. Dans le cas contraire, des problèmes surviendront.

- **Préférences.** Vous pourrez dans cette section paramétrer certaines préférences de *Game Maker*. Ces dernières seront automatiquement sauvegardées et mémorisées lors du prochain appel du logiciel. Vous trouverez ci-dessous la liste des possibilités.

Préférences

A partir du sous-menu **Preferences** du menu **File**, vous pourrez régler un certain nombre de préférences qui seront mémorisées de manière permanente. Les préférences suivantes peuvent être paramétrées :

- **Afficher les jeux récemment édités dans le menu fichier.** Si coché, les 8 jeux les plus récents seront affichés dans le sous-menu 'recent files' du menu fichier.
- **Charger le dernier fichier ouvert au démarrage.** Si coché, le dernier fichier ouvert sera automatiquement chargé lors du lancement de *Game Maker*.
- **Conserver des copies de sauvegarde des fichiers.** Si coché, le programme effectuera une copie de sauvegarde de votre jeu avec l'extension gb0-gb9. Vous pourrez également ouvrir ces jeux dans *Game Maker*. Il est fortement recommandé d'opter pour au moins une copie de sauvegarde automatique de votre travail !
- **Nombre maximal de sauvegardes.** Vous indiquerez ici le nombre de copies de sauvegardes différentes (1-9) qui doivent être faites par le logiciel.
- **Afficher la barre de progression lors du chargement et de la sauvegarde des fichiers.** Si coché, un indicateur de progression sera affiché lors du chargement et de la sauvegarde du jeu.
- **Au démarrage, rechercher et supprimer les anciens fichiers temporaires.** *Game Maker* et les jeux développés grâce à lui, créent des fichiers temporaires. Habituellement, ceux-ci sont automatiquement supprimés mais il arrive parfois qu'ils soient conservés comme par exemple lors du crash d'un jeu. Si cette option est cochée, *Game Maker* vérifiera si des fichiers de ce type existent et dans l'affirmative, les supprimera au démarrage du jeu.
- **Lancer les jeux en mode sécurisé.** Si coché, tout jeu créé avec *Game Maker* et s'exécutant sur votre machine, ne sera pas autorisé à exécuter des programmes externes ou à modifier / effacer des fichiers situés à un endroit différent de l'emplacement du jeu (ceci constitue une sécurité contre les chevaux de Troie bien que le résultat ne soit pas garanti). En cochant cette option, les jeux utilisant des fichiers externes ne fonctionneront

pas correctement. Ce paramètre ne fonctionne que lorsque *Game Maker* est lancé. Si vous lancez le jeu indépendamment de *Game Maker*, par exemple en tant qu'exécutable autonome, le jeu **NE SERA PAS** exécuté en mode sécurisé.

- **Afficher le point d'origine et de bordure des sprites.** Si coché, les points d'origine et de bordure du sprite seront visibles dans l'écran de propriétés du sprite.
- **Dans les propriétés de l'objet, afficher les informations des actions.** Si coché, lorsque la souris survole l'une des actions de l'écran des propriétés de l'objet, une description de l'action sera affichée.
- **A la fermeture, supprimer les instances se trouvant en dehors de la room.** Si coché, le programme vous indique que des instances ou des tuiles (tiles) se trouvent en dehors de la room et vous demande s'il faut les supprimer.
- **Mémoriser le paramétrage de la room lors de la fermeture de la fenêtre.** Si coché, un certain nombre de réglages de la room comme par exemple l'affichage de la grille, l'effacement des objets underlying, etc. sont mémorisés pour être réutilisés lorsque vous éditerez ultérieurement cette room.
- **Scripts, code et couleurs.** Se référer au chapitre sur les scripts pour obtenir davantage d'informations sur ces préférences.
- **Editeur d'image.** Par défaut, *Game Maker* utilise un éditeur intégré pour travailler sur les images. Si vous préférez utiliser un autre programme de traitement d'images, vous pourrez spécifier ici le nom de ce programme.
- **Editeurs externes de sons.** Il vous est possible d'indiquer ici avec quels éditeurs externes vous souhaitez travailler pour les différents types de sons (veuillez noter que *Game Maker* ne comprend pas en base d'éditeur de sons. Aussi, vous ne pourrez pas éditer les sons si vous n'indiquez pas dans cette option le nom de l'éditeur de sons à utiliser).

Menu Edition

Dans le menu Edition (Edit menu), vous trouverez les commandes supplémentaires suivantes :

- **Ajouter un groupe.** Les ressources peuvent être regroupées dans un ou plusieurs groupes. Ceci peut très utile lors de la conception de grands projets de jeux. Par exemple, vous pourrez placer dans un groupe tous les sons d'un objet particulier ou encore regrouper tous les objets utilisés dans un niveau de jeu spécifique. Cette commande crée un nouveau groupe dans le type de ressource actuellement sélectionnée. Il vous sera demandé un nom de groupe. Les groupes peuvent à leur tour contenir d'autres groupes, etc. Comme mentionné ci-dessous, vous pouvez déplacer à la souris (drag) des ressources dans les groupes.
- **Chercher une ressource.** Cette commande vous permet de taper le nom d'une ressource puis d'obtenir la fenêtre de propriétés correspondante à cette ressource.

- **Affichage de l'arborescence des ressources.** Développe dans son intégralité l'arborescence des ressources, affichant ainsi toutes les ressources de votre jeu.
- **Masquage de l'arborescence des ressources.** Masque complètement l'arborescence des ressources, cachant ainsi toutes les ressources.
- **Afficher les informations sur les objets.** Avec cette commande, vous pourrez obtenir un récapitulatif de tous les objets du jeu.

Ajouter un menu

A l'aide de ce menu, vous aurez la possibilité d'ajouter des ressources supplémentaires. Veuillez noter que pour chacune d'entre elles, il existe également un bouton dans la barre d'outils et un raccourci clavier.

Menu Scripts

Dans le menu Scripts, vous trouverez les commandes supplémentaires suivantes :

- **Importer des scripts.** Est utilisé pour importer des fichiers scripts que vous jugez utiles d'utiliser dans votre jeu.
- **Exporter des scripts.** Est utilisé pour sauvegarder vos scripts dans un fichier, qui pourront être employés dans d'autres programmes. Lorsque vous sélectionnez une ressource script, seul ce script sera sauvé. Quand vous sélectionnez un groupe, tous les scripts du groupe seront sauvegardés. Lorsque vous sélectionnez la ressource racine (root resource) ou un type différent de ressource, tous les scripts seront sauvés. Cette option du menu est uniquement disponible que lorsque vous cliquez avec le bouton droit de la souris sur un script ou un groupe de scripts.
- **Afficher les variables internes.** Affiche une liste triée des variables internes, comprenant les variables locales et globales de votre jeu.
- **Afficher les fonctions internes.** Affiche une liste triée des fonctions internes.
- **Afficher les constantes.** Affiche une liste triée des constantes internes et celles définies dans les options de jeu.
- **Afficher les noms de ressource.** Affiche une liste triée des noms de toutes les ressources. Vous pouvez cliquer sur un nom de ressource afin d'ouvrir cette dernière pour édition.
- **Rechercher dans les scripts.** Vous pouvez rechercher une chaîne de caractères dans tous les scripts. Vous pourrez ensuite cliquer sur l'un des éléments trouvés pour éditer le script où se trouve la chaîne.
- **Vérifier les noms des ressources.** Effectue une vérification des noms de toutes les ressources. Les noms incorrects seront indiqués s'il existe des doublons dans les noms de

ressources, ou encore si un nom de ressource possède le même nom qu'une variable, qu'une fonction ou qu'une constante. Vous pouvez cliquer sur un nom afin de pouvoir éditer une ressource particulière.

- **Vérifier tous les scripts.** Vérifie les erreurs de tous les scripts. Vous pouvez cliquer sur l'un des noms de script affiché afin de pouvoir l'éditer.

Précisions sur les sprites

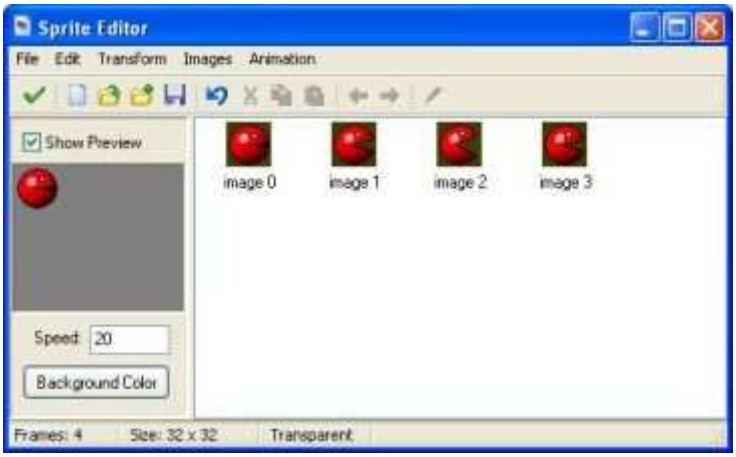
Un certain nombre de possibilités avancées existent pour vous permettre de créer vos propres sprites.

Des informations sur les différentes options avancées sur les sprites peuvent être trouvées dans les pages suivantes :

Edition de sprites

Jusqu'à maintenant, nous avons élaboré nos sprites à partir de fichiers. Il est également possible de les créer et de les modifier soi-même avec *Game Maker*. Pour effectuer cette tâche, ouvrez la fenêtre de propriétés des sprites en double cliquant sur l'un de vos sprites (ou encore en en créant un nouveau). Puis, pressez le bouton **Edit Sprite**. Un nouvel écran apparaîtra affichant toutes les sous-images composant votre sprite.

La fenêtre d'édition des sprites se présente ainsi :



A droite, vous apercevrez les différentes images que comporte le sprite. Veuillez noter que dans *Game Maker*, toutes les sous-images d'un sprite doivent avoir la même taille. A gauche, vous pourrez voir un aperçu de l'animation du sprite (si vous n'apercevez pas l'animation, cochez la case **Show Preview**). En dessous de cette animation, vous pourrez changer la vitesse de l'animation ainsi que la couleur de fond du sprite. De cette manière, vous pourrez vous rendre compte du rendu final qu'aura l'animation dans votre jeu (veuillez cependant noter que la vitesse d'animation du sprite n'est valable que pour la prévisualisation. La vitesse réelle de l'animation durant le jeu dépendra de la vitesse de la room).

L'éditeur de sprites comportent de nombreuses commandes pour créer et modifier un sprite. Elles sont toutes accessibles grâce aux menus (pour certaines d'entre elles, il existe cependant des boutons dans la barre d'outils). Quelques commandes sont destinées à travailler sur des images individuelles. Elles nécessitent que vous sélectionnez en premier lieu une sous-image à l'aide de la souris.

Menu Fichier (File menu)

Le menu Fichier (File menu) comporte des commandes permettant le chargement et la sauvegarde des sprites.

- **Nouveau (New).** Crée un nouveau sprite vierge. Vous devrez indiquer la taille du sprite (souvenez-vous que toutes les images d'un sprite doivent avoir la même taille).
- **Création d'un sprite à partir d'un fichier (Create from file).** Crée le sprite à partir d'un fichier. La plupart des types de fichiers graphiques existants peuvent être utilisés. Tous ces fichiers créeront un sprite comprenant une seule image, à l'exception des fichiers d'animation GIF qui peuvent comporter plusieurs images. Veuillez noter que la couleur de transparence du sprite est déterminée à partir du pixel se trouvant en bas le plus à gauche et non par la couleur de transparence du fichier GIF. Vous avez la possibilité de choisir plusieurs images en même temps qui seront chargées simultanément dans l'éditeur. Rappelez-vous que ces images doivent avoir toutes la même taille.
- **Ajout d'images à partir d'un fichier (Add from file).** Ajoute une image (ou des images) à partir d'un fichier graphique dans le sprite courant. Si les images ne présentent pas la même taille, vous pourrez choisir où les placer ou encore de les déformer. Vous pourrez aussi sélectionner plusieurs images en une seule opération qui seront ensuite chargées simultanément. Là encore, les images doivent avoir la même taille.
- **Sauvegarder comme image GIF (Save as GIF).** Sauve le sprite en tant qu'animation GIF.
- **Sauvegarder comme séquence d'images (Save as strip).** Sauvegarde le sprite dans le format bitmap, comportant toutes les images du sprite à la queue leu-leu.
- **Créer un sprite à partir de séquences d'images (Create from strip).** Vous permet de créer un sprite à partir d'une séquence d'images (strip). Référez-vous au chapitre suivant pour plus d'informations.
- **Ajouter des images à partir d'une séquence d'images (Add from strip).** A utiliser si vous souhaitez ajouter des images à partir d'une séquence d'images. Voir chapitre suivant.
- **Fermer la fenêtre de l'éditeur après sauvegarde (Close saving changes).** Ferme la fenêtre de l'éditeur de sprites, après avoir effectué une sauvegarde des changements effectués sur le sprite. Si vous ne souhaitez pas effectuer de sauvegarde des modifications, cliquez sur le bouton de fermeture en haut à droite de la fenêtre.

Menu Edition (Edit menu)

Le menu d'édition (Edit menu) comprend un certain nombre de commandes en rapport avec le sprite actuellement sélectionné. Vous avez la possibilité de couper le sprite dans le presse-papiers, de coller une image provenant du presse-papiers, de vider le sprite courant, de l'effacer ou encore

de déplacer les sprites de gauche à droite dans l'ordre de la séquence. Enfin, il existe une commande pour éditer une image particulière à l'aide d'un programme de dessin intégré (voir chapitre suivant).

Menu de transformations (Transform menu)

Dans ce menu, vous pourrez effectuer quelques transformations sur les images.

- **Miroir horizontal (Mirror horizontal).** Effectue un effet de miroir horizontal sur les images.
- **Retournement vertical (Flip vertical).** Effectue un retournement vertical des images.
- **Décalage (Shift).** Vous pourrez ici décaler les images en indiquant des valeurs horizontale et verticale.
- **Rotation (Rotate).** Vous pouvez faire tourner les images de 90 degrés, de 180 degrés ou encore d'un certain nombre de degrés souhaité. Dans ce dernier cas, vous pourrez également préciser la qualité désirée. A expérimenter par vous-même pour obtenir les meilleurs résultats.
- **Redimensionner le canevas (Resize Canvas).** Vous pourrez ici modifier la taille du canevas. Vous indiquerez également où devront être placées les anciennes images sur le nouveau canevas.
- **Redimensionner les images (Stretch).** Vous pourrez ici redimensionner les images. Vous indiquerez le facteur d'agrandissement et la qualité souhaités.
- **Echelle (Scale).** Cette commande effectue un agrandissement des images (sans en augmenter la taille réelle cependant !). Vous indiquerez le facteur d'agrandissement, la qualité voulue et la position des images courantes dans la nouvelle échelle.

Menu images (Images menu)

Dans le menu images, vous pourrez effectuer quelques opérations sur les images.

- **Décalage des images d'un cran vers la gauche (Cycle left).** Décale toutes les images d'un cran vers la gauche. Ceci fera débuter l'animation à un autre point de référence.
- **Décalage des images d'un cran vers la droite (Cycle right).** Décale toutes les images d'un cran vers la droite.
- **Noir et blanc (Black and white).** Colorise le sprite en noir et blanc (n'affecte pas la couleur de transparence !).
- **Colorisation (Colorize).** Vous pouvez ici changer la couleur ou la tonalité (hue) des images. Utilisez l'ascenseur pour choisir les différentes couleurs.
- **Colorisation partielle (Colorize Partial).** Vous permet de changer la couleur ou nuance (hue) d'une partie des images. Vous pourrez sélectionner l'ancienne couleur et une gamme

de couleurs autour d'elle puis indiquer la nouvelle couleur avec laquelle vous remplacerez cette gamme de couleurs. Ceci est utilisé par exemple pour changer uniquement la couleur des maillots (shirts) des joueurs.

- **Décalage de la tonalité de l'image (Shift Hue).** Ceci constitue une autre façon de modifier la couleur des images. Mais cette fois, les couleurs sont modifiées en utilisant une variation de couleurs ce qui peut donner des effets plutôt intéressants.
- **Intensité (Intensity).** A utiliser pour changer l'intensité en fournissant des valeurs pour la couleur de saturation et la luminosité des images.
- **Inversion (Invert).** Inverse les couleurs des images.
- **Fading.** Il vous sera demandé ici une couleur et une valeur allant de 0 à 256. Les couleurs des images seront ensuite interpolées vers cette couleur.
- **Transparence (Transparency).** Vous indiquerez ici un niveau de transparence (0 à 256). Ceci est réalisé en rendant un certain nombre de pixels transparents.
- **Flou (Blur).** En rendant plus flou les images, les couleurs sont mélangées légèrement, créant ainsi un effet de déformation de l'image. Plus la valeur sera grande et plus l'effet de déformation sera important.
- **Contour (Outline).** Crée un contour autour de l'image. Il vous sera demandé outre la couleur de contour si les images courantes doivent être supprimées ou non (ne conservant donc que le contour) ou encore si le contour doit être affiché sur l'image.
- **Bordure (Boundary).** Similaire à la fonction contour (Outline) mais cette fois-ci, l'affichage ne se fera pas en dehors de l'image mais sur les pixels en bordure de l'image.
- **Réduction (Crop).** Ceci réduit la taille (en octets) des images de la manière la plus compacte possible. Cela peut être très utile car plus les images sont grosses et plus *Game Maker* consommera de mémoire vidéo. Il vous sera possible de laisser une petite bordure autour des images afin d'éviter les problèmes de transparence.

Vous devrez expérimenter ces commandes par vous-même afin d'obtenir les sprites souhaités.

Menu animation (Animation menu)

A l'aide de ce menu, vous pourrez créer de nouvelles animations en utilisant l'animation courante. Il y a beaucoup d'options et il vous sera nécessaire d'expérimenter par vous-même afin d'être en mesure de créer les effets désirés. N'oubliez pas que vous pourrez toujours sauvegarder une animation et l'ajouter ultérieurement à l'animation courante. Selon le même principe, il vous sera également possible d'ajouter des images vierges et d'effacer celles que vous ne désirez plus utiliser. Nous allons énoncer ici brièvement les différentes possibilités.

- **Fixe le nombre d'images (Set Length).** Vous permet de changer la longueur de votre animation. L'animation est répétée autant de fois que nécessaire pour créer le nombre d'images que vous aurez indiqué (normalement, vous souhaiterez indiquer un multiple du nombre actuel d'images).
- **Stretch.** Cette commande modifie également la longueur de l'animation. Cette fois-ci, les images (frames) sont dupliquées ou supprimées afin d'obtenir un certain nombre d'images constituant l'animation. Ainsi, si vous augmentez le nombre d'images, l'animation sera plus lente. A contrario, si vous diminuez ce nombre, l'animation sera plus rapide.
- **Inverser l'animation (Reverse).** Comme son nom l'indique, cette fonction inverse le sens de l'animation. Ainsi, cette dernière sera jouée à l'envers.
- **Ajout d'images inversées (Add Reverse).** Une séquence d'inversion d'images est ajoutée, ce qui doublera le nombre total d'images de l'animation. Ceci peut être très utile afin qu'un objet aille de gauche à droite, change de couleur puis revienne à sa position, etc. Vous souhaiterez parfois supprimer la première image ainsi que celle du milieu qui seront en double dans l'animation créée.
- **Ordre de translation (Translation sequence).** Vous créerez ici une animation dans laquelle l'image sera légèrement dupliquée à chaque étape. Vous indiquerez le nombre d'images et la valeur de déplacement horizontal et vertical.
- **Ordre de rotation (Rotation sequence).** Vous permet de créer une animation dans laquelle les images subiront un effet de rotation. Vous pourrez choisir une rotation suivant le sens des aiguilles d'une montre ou l'ordre inverse. Indiquez le nombre d'images et l'angle total en degrés (360 correspond à un tour complet) (il vous sera peut être utile de redimensionner le canevas auparavant afin d'être certain que l'image entière restera visible pendant la rotation).
- **Colorisation (Colorize).** Crée une animation qui transforme l'image en utilisant une couleur particulière.
- **Fading vers une couleur (Fade to color).** Crée une animation qui applique un effet de fading à l'image en utilisant une certaine couleur.
- **Disparition (Disappear).** Fait disparaître l'image en utilisant la couleur de transparence du fond de l'image.
- **Réduction des images (Shrink).** Réduit la taille des images jusqu'à l'infini. Vous pourrez indiquer la direction.
- **Grossissement des images (Grow).** Agrandit la taille des images jusqu'à l'infini.
- **Réduction et écrasement (Flatten).** Réduit la taille des images de l'animation en appliquant un effet d'écrasement des images, le tout dans une certaine direction.
- **Agrandissement et étirement (Raise).** Agrandit les images dans une certaine direction avec un effet d'étirement.

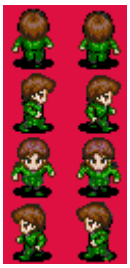
- **Superposition (Overlay).** Superpose à l'animation courante une autre animation ou une autre image provenant d'un fichier.
- **Transformation (Morph).** Transforme l'animation courante en une autre animation ou image provenant d'un fichier. Veuillez noter que le morphing donne de meilleurs résultats

si les deux animations couvrent la même zone de l'image. Sinon, certains pixels disparaîtront à mi-course alors que d'autres apparaîtront soudainement.

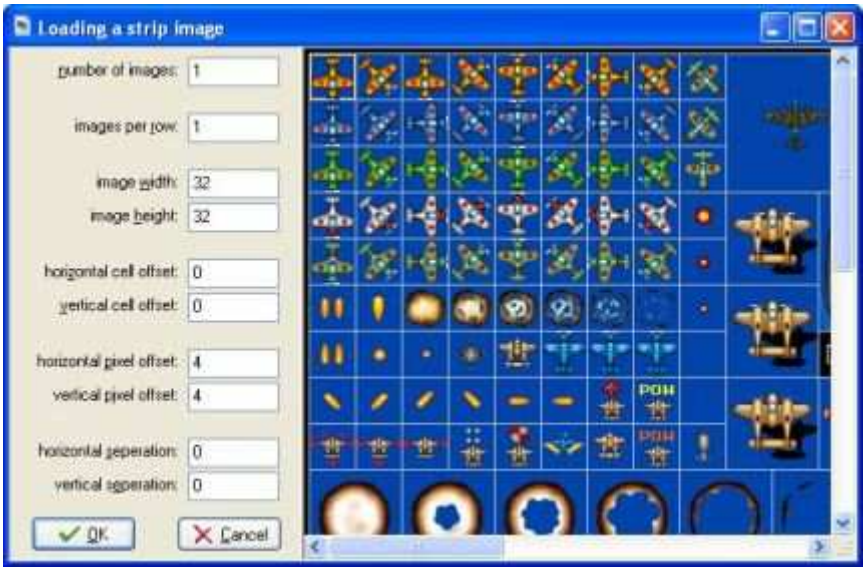
Les deux dernières commandes sont particulièrement puissantes. Par exemple, pour faire exploser un objet, ajoutez un certain nombre de copies de cet objet mais également une certaine quantité d'images vierges. Puis superposez-les avec une animation d'une explosion (soyez certain que le nombre d'images correspond). Vous pourrez aussi utiliser la fonction de morphing pour créer une explosion. Avec de l'entraînement, vous obtiendrez des sprites spectaculaires.

Les séquences animées

Comme indiqué précédemment, les sprites sont stockés habituellement dans des fichiers GIF ou encore dans des fichiers d'animations (strips). Ces derniers sont en quelque sorte une grosse image bitmap contenant toutes les images les unes à la suite des autres. Le problème vient que la taille de ces images n'est pas mémorisée dans le fichier bitmap. Aussi, de nombreux fichiers d'animations disponibles sur le web enregistrent plusieurs sprites dans un seul et unique fichier. Par exemple, le fichier d'animations ci-dessous comprend quatre animations différentes.



Pour extraire un ou plusieurs sprites de ces fichiers d'animations, vous utiliserez l'option **Create from Strip** ou **Add from Strip** du menu **File**. Après avoir indiqué le fichier image souhaité dans la séquence, la fenêtre suivante s'affichera :



A droite, vous pourrez voir les images (une partie seulement) de la séquence choisie. A gauche, vous pourrez entrer un nombre qui indiquera le nombre de sous-images que vous souhaitez récupérer. Veuillez noter qu'un rectangle (ou plus selon le nombre entré) apparaîtra dans l'image pour indiquer les images sélectionnées. Vous pourrez en outre préciser les paramètres suivants :

- **Nombre d'images (Number of images).** Ceci correspond au nombre d'images que vous souhaitez utiliser dans la séquence animée.
- **Images par colonne (Images per row).** Combien d'images souhaitez-vous afficher par colonne ? Par exemple, en affectant la valeur 1 à ce paramètre, vous sélectionnerez une séquence verticale d'images.
- **Largeur de l'image (Image width).** Largeur des images individuelles en pixels.
- **Hauteur des images (Image height).** Hauteur des images individuelles en pixels.
- **Déplacement horizontal dans les cellules (Horizontal cell offset).** Si vous ne désirez pas choisir les images situées en haut et à gauche, vous indiquerez ici le nombre d'images à sauter dans le sens horizontal.
- **Déplacement vertical dans les cellules (Vertical cell offset).** Vous mentionnerez ici le nombre d'images à sauter dans le sens vertical.
- **Déplacement horizontal d'un certain nombre de pixels (Horizontal pixel offset).** Parfois, il est nécessaire de laisser de l'espace entre les images et le bord supérieur gauche. Il vous suffira pour cela de mentionner la valeur (en pixels) dans cette zone.
- **Déplacement vertical d'un certain nombre de pixels (Vertical pixel offset).** Valeur de séparation entre le bord vertical et les images.
- **Séparation horizontale (Horizontal separation).** Dans certaines séquences d'images, il y a parfois des lignes ou des espaces entre les images. Vous pourrez indiquer ici la valeur horizontale d'espacement entre les images (en pixels).
- **Séparation verticale (Vertical separation).** Valeur de l'espacement vertical entre les images.

Une fois les images qui vous intéressent sélectionnées, appuyez sur **OK** pour créer votre sprite. Veuillez cependant ne pas oublier que vous ne serez autorisé à utiliser les images créées par d'autres personnes que si celles-ci vous ont donné leur autorisation écrite ou si les images sont libres de droits (freeware).

Edition des sous-images individuelles

Il vous est possible d'éditer individuellement chacune des sous-images. Pour ce faire, sélectionnez une sous-image puis choisissez **Edit Image** à partir du menu **Image**. Un petit programme de dessin intégré se lancera alors. Veuillez cependant bien comprendre qu'il s'agit ici d'un programme assez limité dans ses fonctions et qui est à utiliser pour réaliser de petites modifications sur des images **existantes** (et non pas pour en créer de nouvelles). Il existe de bons programmes de dessin plus complets pour créer des images que vous pourrez ensuite copier ou coller (via le presse-papiers) dans *Game Maker*. Il vous est possible de paramétrer l'usage d'un éditeur d'images externe à partir des préférences de *Game Maker*.

Cet écran affiche l'image au milieu et quelques boutons de dessin dans la partie gauche. Vous pourrez ici effectuer un zoom dans les deux sens, dessiner des pixels, des lignes, des rectangles, du texte, etc. Notez que la couleur dépend du bouton de la souris utilisé (droit ou gauche). Pour certains outils de dessin, vous pourrez régler les propriétés (comme la largeur de la ligne ou encore la visibilité de la bordure). Il existe un bouton spécial pour changer tous les pixels d'une certaine couleur en une autre teinte. Ceci est particulièrement utile pour modifier la couleur du décor qui est utilisée pour la transparence. Dans la barre d'outils, vous apercevrez certains boutons spéciaux pour déplacer les pixels de l'image à une position particulière. Vous aurez la possibilité également d'indiquer si l'on doit afficher la grille lors du zoom de l'image (fonctionne uniquement avec un facteur de zoom de 4 au minimum).

Vous pouvez choisir la zone de l'image de la manière habituelle, c'est à dire en pressant le bouton approprié puis en dessinant un rectangle. Ensuite, vous placerez la souris à l'intérieur de la zone

choisie pour la positionner autre part. Normalement, l'aire d'origine sera remplie avec la couleur sélectionnée grâce au bouton gauche de la souris. Si vous déplacez la sélection en maintenant appuyée la touche <Shift>, la zone d'origine ne sera pas affectée par cette couleur. Avec la touche <Shift>, vous pourrez aussi effectuer de multiples copies de l'aire sélectionnée. Si vous employez le bouton droit de la souris pour le déplacement (plutôt que le celui de gauche), la sélection sera interprétée comme une couleur transparente.

L'outil de manipulation de textes nécessite des explications supplémentaires. Pour ajouter du texte, pressez le bouton **texte** puis cliquez sur l'image. Une fenêtre apparaîtra ensuite dans laquelle vous entrerez le texte souhaité. Utilisez le symbole # pour insérer une nouvelle ligne. Une fois que vous avez appuyé sur **OK**, le texte sera affiché dans l'image, accompagné d'une boîte tout autour. Vous pourrez déplacer le texte en pressant un bouton de la souris dans la boîte puis en tirant le texte (drag). Vous pouvez modifier le texte en cliquant dans la boîte avec le bouton droit de la souris. L'utilisation du menu **Text** vous permettra de modifier l'alignement et la police de caractères utilisés.

A droite de l'écran, vous pourrez mentionner les couleurs que vous souhaitez utiliser (une couleur à l'aide du bouton gauche et une autre avec le bouton droit de la souris). Il existe quatre façons de modifier la couleur. En premier lieu, vous pourrez cliquer avec l'un des boutons de la souris (gauche ou droit) pour choisir l'une des 16 couleurs de base. Veuillez noter qu'il existe une boîte de couleur spéciale contenant la couleur du pixel le plus en bas à gauche de l'image qui sera utilisée comme couleur de transparence du sprite si toutefois celui-ci doit être transparent. Vous pouvez utiliser cette couleur pour rendre une partie de votre image transparente. En second lieu, vous pourrez cliquer dans l'image sur la couleur à changer. Vous aurez ici plus de couleurs de disponible. En maintenant pressé le bouton de la souris, vous pourrez voir la couleur que vous avez sélectionnée. En troisième lieu, vous pouvez cliquer à l'aide du bouton gauche de la souris dans les boîtes indiquant la couleur de gauche et droite. Une boîte de dialogue apparaîtra ensuite, vous permettant de choisir la couleur. Enfin, vous pourrez choisir l'outil de copie de couleur situé à gauche puis cliquer sur un endroit de l'image pour copier à cet emplacement précis la couleur actuellement sélectionnée.

Il y a deux fonctions spéciales. Lorsque vous pressez la touche <Ctrl>, vous pouvez sélectionner une couleur pour dessiner à partir de l'image courante. Lorsque vous maintenez la touche <Shift> tout en dessinant des lignes, vous obtiendrez uniquement des lignes horizontales, verticales ou diagonales. De la même façon, en pressant la touche <Shift> tout en dessinant des ellipses ou des rectangles, vous ne produirez comme figures que des cercles et des carrés.

Dans les menus, vous trouverez les mêmes commandes de modification et de transformation des images que celles proposées par l'éditeur de sprite. A la différence près que cette fois-ci,

ces commandes ne s'appliqueront uniquement qu'à l'image courante (si le sprite possède plusieurs images, les commandes modifiant la taille, comme stretch (étirement), ne seront pas disponibles). Il vous est également possible de sauvegarder l'image en tant que fichier bitmap. Il existe aussi deux commandes supplémentaires dans le menu **Image** :

- **Effacer (Clear).** Efface l'image avec la couleur de gauche (qui deviendra alors automatiquement la couleur de transparence).
- **Remplir de façon graduelle (Gradient fill).** Cette commande vous permettra de remplir l'image en utilisant une couleur changeante graduelle (pas très utile pour la réalisation de sprites mais le résultat peut être amusant pour d'autres usages. Utilisable également pour les décors qui utilise également le même programme de dessin).

Veuillez noter que cet éditeur ne comprend pas de routines de dessin très élaborées. Pour obtenir d'autres fonctions, il vous sera nécessaire d'utiliser un programme de dessin plus évolué (ou utiliser simplement le logiciel de dessin fourni avec Windows). Une manière très commode sera d'utiliser le bouton Copy (Copie) pour placer l'image dans le presse-papiers. Ensuite, dans votre logiciel de dessin, faites paste (coller) pour pouvoir le modifier. Modifiez le dessin à votre guise puis copiez-le de nouveau dans le presse-papiers. De retour sous *Game Maker*, vous pourrez alors recoller l'image que vous aurez modifiée avec le programme de dessin.

Paramétrage avancé des sprites

Dans le mode avancé, il existe dans l'écran de propriétés des sprites un certain nombre d'options avancées que nous allons détailler dans cette section.

En premier lieu, nous trouvons les options ayant trait à la vérification des collisions. Lorsque deux instances se touchent, un événement de collision se produit. Les collisions sont vérifiées de la façon suivante. Chaque sprite possède ce que l'on appelle une boîte de rebord (bounding box). Cette boîte contient la partie non transparente de toutes les sous-images. Lorsque les boîtes de rebord se recouvrent, on vérifie également si deux pixels des sous-images actuelles des deux sprites se recouvrent eux-aussi. Cette seconde opération consomme beaucoup de mémoire et de temps processeur. Aussi, si vous n'êtes pas particulièrement intéressé de vérifier précisément la collision d'un certain sprite, il sera préférable de décocher la case **Precise collision checking**. Dans ce cas, seule la boîte de rebord du sprite est prise en compte dans la collision. Vous avez la possibilité également de modifier la boîte de rebord du sprite. Ceci est très rarement nécessaire mais il peut vous arriver parfois de vouloir une boîte de rebord plus petite, ce qui aura comme conséquence de ne pas prendre en considération certaines parties du sprite.

Les bords des sprites peuvent paraître parfois quelque peu anguleux. Pour éviter ceci, cochez la case **Smooth edges** (bords atténués). Dans ce cas, les pixels aux bords du sprite (c'est à dire les pixels qui sont voisins des pixels transparents) seront affichés transparents en partie. Ceci peut les rendre plus jolis (à ne pas utiliser quand les sprites doivent s'assembler pour former de plus grandes formes car dans ce cas, une ligne partiellement transparente apparaîtra entre les sprites). L'effet de ce paramétrage n'est visible que dans le jeu, pas dans l'éditeur !

Lors du jeu, les sprites sont transformés en textures. Les textures sont copiées dans la mémoire vidéo (de la carte graphique) avant de pouvoir être utilisées. Si la case **Preload texture** est cochée, les textures seront affichées immédiatement lors du chargement du jeu. Ainsi, il n'y aura pas de délai d'attente pendant le jeu. Cependant, si vous avez beaucoup de grands sprites qui ne doivent pas être utilisés au début du jeu, il sera préférable de décocher cette option. *Game Maker* chargera les textures dans la mémoire vidéo et vice-versa lorsque cela sera nécessaire.

Enfin, vous pouvez indiquer l'origine du sprite. C'est le point du sprite qui indique sa position. Lorsque vous placez une instance à une position particulière, l'origine du sprite est définie à cet endroit. L'emplacement par défaut est le coin gauche en haut du sprite mais parfois, il est plus intéressant d'utiliser le centre ou encore une autre point. Vous pourrez même choisir une origine

se situant en dehors du sprite. Vous pouvez aussi fixer l'origine en cliquant dans l'image du sprite (lorsque l'origine est affichée dans l'image).

Précisions sur les Sons et la Musique

Dans le mode avancé, vous aurez plus de contrôle sur les sons et les morceaux de musique utilisés dans votre jeu. Lorsque vous ajouterez une ressource sonore, l'écran suivant apparaîtra :

A côté des boutons de chargement, de sauvegarde et d'écoute des sons, il existe de nombreux paramètres que nous allons décrire maintenant.

Premièrement, vous pourrez indiquer le genre de son souhaité. Quatre types de sons existent. Les sons normaux sont habituellement utilisés pour créer des effets sur les sons dans des fichiers WAVE (bien qu'ils puissent aussi être utilisés pour les fichiers MIDI). Plusieurs sons normaux peuvent être joués simultanément. Il vous est même possible d'écouter plusieurs copies du même son en simultané. La musique de fond (Background music) est comparable aux sons normaux mais une seule peut être jouée à un moment donné. Ainsi, si vous demandez à jouer un nouveau son comme musique de fond, le son actuellement en cours de lecture sera stoppé. Les fichiers MIDI sont utilisés par défaut comme musique de fond. Le son 3D est un son sur lequel vous pourrez appliquer des effets 3D grâce à des fonctions spéciales. Vous ne les utiliserez uniquement que pour créer des effets sophistiqués sur les sons.

Les fichiers sonores sont joués normalement à l'aide de DirectX. Ceci vous donne de nombreuses possibilités mais se limite aux fichiers WAVE et MIDI. Si vous souhaitez utiliser d'autres types de fichiers, comme les fichiers MP3, vous devrez alors sélectionner l'option pour utiliser Media Player.

Cela est encore plus limité que précédemment. Aucun réglage de volume ni effets ne pourront être utilisés et seul un morceau de musique pourra être entendu à la fois. Veuillez noter que les fichiers Midi, lorsqu'ils sont joués à l'aide de Media Player, peuvent paraître différents de ceux joués

en tant que musique de fond ou sons normaux. Ceci s'explique par le fait que Media Player utilise le synthétiseur hardware (qui peut varier d'une machine à une autre) alors que dans les autres cas, c'est un programme qui est utilisé pour jouer les sons (ainsi, les sons seront identiques sur toutes les machines). Il est conseillé d'éviter d'utiliser les fichiers MP3 dans vos jeux. Pour ce type de fichier, il est nécessaire de les décompresser ce qui consomme du temps de traitement et peut ralentir le jeu. Le fait que la taille de fichier soit plus petite, ne signifie pas que les sons consomment moins de mémoire. De plus, toutes les machines ne les supportent pas. Votre jeu pourra par conséquent ne pas fonctionner sur toutes les machines.

Deuxièmement, vous pourrez indiquer des effets sonores, comme **chorus** ou encore **echo** (uniquement dans la version enregistrée de *Game Maker* !). Vous pouvez sélectionner plusieurs combinaisons. Vous pourrez écouter immédiatement le résultat de vos effets (en programmant avec le langage GML, il vous sera même possible de modifier les paramètres de ces effets).

Vous indiquerez également le volume par défaut du son mais aussi si le son doit sortir du haut-parleur de gauche ou de celui de droite.

Pour tous les types de sons, vous pourrez mentionner si ces derniers doivent être préchargés ou pas. Lorsqu'un son est joué, il est nécessaire auparavant de le charger dans la mémoire audio. Si vous préchargez le son, ceci sera fait au début du jeu, rendant le son disponible immédiatement pour un usage ultérieur. Sinon, le son sera chargé la première fois qu'il sera utilisé. Cela pourra économiser de la mémoire mais occasionnera un petit délai d'attente lors de la première utilisation de ce son.

Game Maker ne comporte pas d'éditeur de sons intégré. Mais il est possible d'indiquer dans les préférences tous les éditeurs externes que vous souhaitez utiliser pour l'édition des sons. Si vous procédez ainsi, il vous suffira de presser sur le bouton **Edit Sound** pour éditer le son courant (la fenêtre de *Game Maker* sera alors cachée pendant l'édition du son puis réapparaîtra lorsque vous fermerez l'éditeur de sons).

Précisions sur les Arrière-plans (Backgrounds)

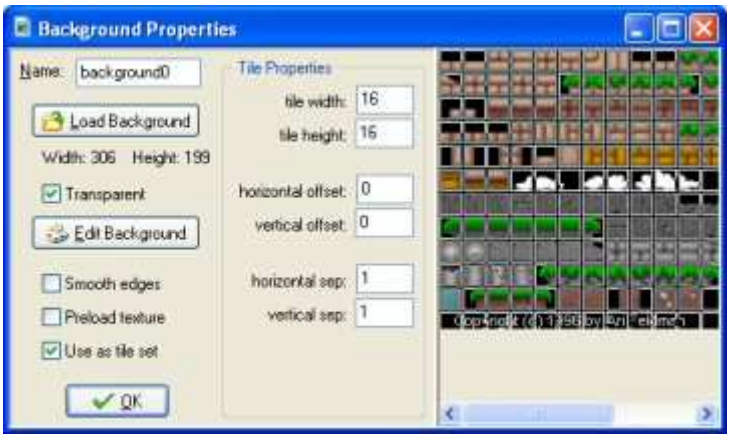
En plus de pouvoir les charger à partir de fichiers sur disque, vous pouvez aussi créer vos propres arrière-plans (backgrounds). Pour ce faire, cliquez sur le bouton **Edit Background**. Un petit programme de dessin intégré s'ouvrira avec lequel vous pourrez créer ou modifier votre arrière-plan. Ce programme n'est pas toutefois très élaboré. Pour obtenir davantage de fonctions d'édition, veuillez utiliser un programme de dessin du commerce (NDT : ou encore **Paint**, le logiciel fourni avec Windows). Il existe une option particulièrement utile. Dans le menu **Image**, vous trouverez la commande **Gradient Fill**. On l'utilise pour créer de jolis arrière-plans avec des couleurs dégradées.

Dans le mode avancé, l'écran de propriétés des arrière-plans propose un certain nombre d'options avancées.

Les bords des arrière-plans, plus particulièrement lorsque ceux-ci sont transparents, peuvent apparaître quelque peu anguleux. Vous pouvez éviter cela en cochant la case **Smooth edges**. Les pixels aux bords de l'arrière-plan (c'est à dire ceux voisins des pixels transparents) seront affichés en partie de manière transparente. Cela contribuera à les rendre plus jolis (à ne pas utiliser toutefois quand les sprites doivent s'assembler pour former de plus grandes formes car dans ce cas, une ligne partiellement transparente apparaîtra entre les sprites). L'effet de ce paramétrage n'est visible que dans le jeu, pas dans l'éditeur !

Lors du jeu, les sprites sont transformés en textures. Les textures sont copiées dans la mémoire vidéo (de la carte graphique) avant de pouvoir être utilisées. Si la case **Preload texture** est cochée, les textures seront affichées immédiatement lors du chargement du jeu. Ainsi, il n'y aura pas de délai d'attente pendant le jeu. Cependant, si vous avez beaucoup de grands sprites qui ne doivent pas être utilisés au début du jeu, il sera préférable de décocher cette option. *Game Maker* chargera les textures dans la mémoire vidéo et vice-versa lorsque cela sera nécessaire.

Parfois, vous souhaiterez utiliser un arrière-plan constitué d'un jeu de tuiles (tiles) qui sont en fait un ensemble de petites images encapsulées dans une plus grande image. Lors de la création de rooms, vous pourrez ajouter ces sous-images à différents endroits de la room. Ceci est très utile pour créer de ravissants niveaux de jeu. Afin de pouvoir utiliser un arrière-plan en tant que jeu de tuiles, cochez la case **Use as tile set**. L'écran ressemblera alors à ceci :



Il vous sera possible d'indiquer certains paramètres pour le jeu de tuiles. Notamment, vous pourrez mentionner la largeur et la hauteur de chaque tuile (une seule taille peut être donnée, aussi, vérifiez au préalable que toutes les tuiles présentent la même taille. En cas de tailles différentes, créez deux jeux de tuiles ou davantage). Il est possible également de fournir l'offset où débutera la tuile située la plus à gauche en haut. Enfin, une séparation entre les tuiles (normalement 0 ou 1) peut être mentionnée. Pour plus d'informations sur l'utilisation des tuiles, consultez le chapitre sur la création des rooms.

Veuillez prendre connaissance de l'avertissement suivant. Lorsque vous placez des bordures de séparation entre les sprites et utilisez une interpolation entre les pixels (consultez les réglages générales du jeu), des fissures peuvent apparaître entre les tuiles. Afin d'éviter cela, vérifiez que les pixels autour des tuiles correspondent aux pixels juste à l'intérieur des tuiles.

Précisions sur les Objets

Lorsque vous créez un objet dans le mode avancé, vous pouvez modifier certains réglages avancés.

Profondeur (Depth)

Premièrement, vous pourrez régler la **Profondeur** (Depth) des instances de l'objet. Lorsque les instances sont affichées à l'écran, elles sont dessinées selon leur ordre de profondeur. Les instances avec la plus grande profondeur seront affichées en premier alors que celles avec la plus petite profondeur seront dessinées en dernier. Quand les instances présentent la même profondeur, elles seront affichées dans l'ordre où elles ont été créées. Si vous voulez être certain qu'un objet recouvre tous les autres, donnez-lui une profondeur négative. Pour s'assurer que cet objet est en dessous des autres instances, indiquez alors une profondeur positive très grande. Il vous est également possible de changer la profondeur d'une instance durant le jeu en utilisant la variable dénommée **depth**.

Objets persistants

Deuxièmement, vous pouvez rendre un objet persistant. Un objet persistant continuera d'exister lorsque vous naviguerez d'une room à une autre. Il ne disparaîtra que quand vous le détruirez explicitement. Aussi, il vous suffira de placer une instance de l'objet dans la première room : cette instance demeurera également présente dans toutes les autres rooms. Ceci est une fonction très importante qui peut être utilisée par exemple lorsque vous avez un caractère principal qui se déplace de room en room. L'usage d'objets persistants est un mécanisme puissant mais qui peut aussi conduire à commettre certaines erreurs.

Parents

Chaque objet peut avoir un objet parent. Quand un objet possède un parent, il hérite du comportement de son parent. Autrement dit, l'objet est en quelque sorte une représentation particulière de l'objet parent. Par exemple, si vous avez 4 balles différentes, appelées ball1, ball2, ball3 et ball4, possédant toutes le même comportement mais ayant un sprite différent, vous pourrez faire en sorte que ball1 soit l'objet parent des trois autres. Ensuite, nous n'aurons besoin désormais que de définir les événements pour l'objet ball1. Tous les autres hériteront des événements et se comporteront de la même façon. Bien entendu, si vous appliquez des actions aux instances de l'objet parent, ces mêmes actions seront exécutées aux autres objets enfants.

Ainsi, par exemple, si vous détruisez toutes les instances de l'objet ball1, les instances de ball2, de ball3 et de ball4 seront également détruites. Ceci vous économisera beaucoup de travail dans vos jeux.

Souvent, les objets présentent un comportement presque identique et ont très peu de différences. Par exemple, un monstre doit pouvoir monter et descendre alors qu'un autre doit se déplacer de gauche à droite. Pour tout le reste, les monstres ont le même comportement. Dans ce cas, presque tous les événements devraient avoir les même actions sauf pour un ou deux événements qui présenteront des actions spécifiques. De nouveau, nous pourrons créer un objet parent d'un autre. Mais dans ce cas précis, nous définirons aussi certains événements pour l'objet enfant. Ces événements "écraseront" les événements de l'objet parent. Ainsi, lorsqu'un événement de l'objet enfant contient des actions, celles-ci seront exécutées à la place de celles de l'événement de l'objet parent. Si vous souhaitez cependant exécuter l'événement parent, vous pourrez le faire en appelant l'événement "inherited" (héritage) en utilisant l'action appropriée.

Une bonne habitude est dans ce cas de créer un objet de base. Cet objet contiendra tous les comportements par défaut mais il ne sera jamais utilisé dans le jeu. Tous les objets actuels auront cet objet de base comme parent. Les objets parents peuvent à leur tour avoir des parents et ainsi de suite (cependant, vous ne pourrez pas créer de cycles). De cette manière, vous pourrez créer une hiérarchie d'objets. Cela est extrêmement utile pour structurer votre jeu et il vous est fortement recommandé d'apprendre à maîtriser ce mécanisme.

Il existe également une seconde utilité à employer les objets parents. Ils héritent du comportement de collision pour les autres objets. Utilisons un exemple afin de mieux comprendre. Supposons que vous avez quatre objets différents pour le sol. Quand une balle heurte le sol, elle doit changer de direction. Il vous faut donc mentionner cette action dans l'événement de collision de la balle avec le sol. Parce qu'il y a quatre sols différents, vous devrez placer le code dans les quatre événements différents de collision de la balle. Par contre, si vous créer un objet de base pour le sol et rendez cet objet parent des quatre autres objets pour le sol, il ne vous sera ensuite nécessaire que de préciser l'événement de collision avec ce sol de base. Les autres collisions exécuteront le même événement. Encore une fois, cela vous économisera beaucoup de travail.

Comme déjà mentionné, lorsque vous utilisez un objet parent, celui-ci s'applique aussi aux descendants. Cela survient quand, dans une action, vous indiquez que cette action doit s'appliquer aux instances d'un certain objet. Cela arrive également quand vous employez l'instruction `with()` dans le code (voir plus loin). Cela est également vrai quand vous invoquez des fonctions comme `instance_position`, `instance_number`, etc. Enfin, cela est aussi valable lorsque vous faîtes référence à des variables d'autres objets. Dans l'exemple plus haut, lorsque vous fixez à 10 la variable `ball1`.speed, cela s'applique également à ball2, ball3 et ball4.

Masques (Masks)

Quand deux instances se chevauchent, un événement de collision apparaît. Afin de pouvoir décider à quel endroit les deux instances se sont touchées, on utilise les sprites. Cela est suffisant dans la plupart des cas mais parfois, vous souhaiterez que les collisions surviennent à un autre endroit. Par exemple, si vous créez un jeu isométrique, les objets auront en base une hauteur (afin de leur donner une vue 3D). Les seules collisions que vous désirez avoir, concerneront la partie basse du sprite. On peut réaliser ceci en créant un sprite séparé qui sera utilisé comme masque de collision pour l'objet.

Information

Le bouton **Show Information** (Afficher des informations) vous donne un aperçu de toutes les informations concernant un objet, aperçu qui pourra ensuite être imprimé. Ceci est particulièrement appréciable quand vous vous perdez quelque peu dans toutes les actions et événements de l'objet.

Précisions sur les Actions

Dans le mode avancé, il y a un certain nombre d'actions supplémentaires disponibles qui seront décrites ici.

Les informations sur les différentes actions supplémentaires peuvent être trouvées dans les pages suivantes :

Précisions sur les Actions de Déplacements

Quelques actions de déplacements supplémentaires sont disponibles dans le mode avancé. Les actions suivantes ont été ajoutées :

 **Définir un chemin pour toutes les instances (Set a path for the instance)**
Avec cette action, vous indiquez que l'instance doit suivre un chemin particulier. Vous déclarez le chemin qui doit être suivi et la vitesse en pixels par step. Lorsque la vitesse est positive, l'instance démarre au début du chemin. Si elle est négative, elle débutera à la fin du chemin. Ensuite, vous préciserez le comportement final, c'est à dire ce qui doit se passer quand la fin du chemin est atteinte. Vous pouvez choisir d'arrêter le déplacement, de redémarrer depuis le début, de relancer à partir de la position courante (qui correspond à la position de fin de chemin), ou encore d'inverser le mouvement. Enfin, vous pouvez indiquer si le chemin doit être **absolu**, c'est à dire que la position sera indiquée dans le chemin (ceci est utile lorsque vous avez désigné le chemin à un endroit particulier dans la room) ou en **relatif**, dans ce cas, le point de départ du chemin est placé à la position courante de l'instance (au point de fin quand la vitesse est négative). Consultez le chapitre sur les chemins pour avoir plus d'informations.

 **Stopper le chemin pour l'instance (End the path for the instance)**
Action à utiliser pour stopper le chemin de l'instance.

 **Définir la position du chemin (Set the position on the path)**
Avec cette action, vous pourrez changer la position courante de l'instance dans le chemin. Cela devra être une valeur comprise entre 0 et 1 (0=début, 1=fin).

 **Définir la vitesse du chemin (Set the speed for the path)**
Cette action vous permet de modifier la vitesse de l'instance dans le chemin. Une vitesse négative déplacera l'instance en sens inverse le long du chemin. Réglez cette valeur à 0 pour stopper temporairement le déplacement dans le chemin.

 **Exécuter une étape en direction d'un point (Perform a step towards a point)**
Cette action a normalement sa place dans l'événement step afin que l'instance effectue un step en direction d'une position particulière. Si l'instance est déjà à cette position, elle ne se déplacera pas. Vous indiquez la position à atteindre, la vitesse avec laquelle l'on doit se déplacer, ce qui représente la taille du step et ensuite si le déplacement doit s'arrêter en cas de collision avec une instance solide ou encore avec une instance quelconque.

 **Aller en direction d'un point tout en évitant les objets (Step towards a point avoiding objects)**

Ceci est une action de déplacement très puissante. Elle doit être placée dans l'événement step. Comme l'action précédente, elle ordonne à l'instance d'aller à un step situé à une position particulière. Mais cette fois-ci, elle essayera d'éviter les obstacles. Si l'instance rencontre une instance solide (ou toute autre instance), elle changera la direction de déplacement pour éviter de heurter l'instance et la contournera. Le résultat n'est cependant pas garantie mais dans la plupart des situations, cela devrait effectivement déplacer l'instance vers une direction donnée. Pour les cas plus compliqués, il existe des fonctions de gestion des déplacements. Vous entrez la position à atteindre, la vitesse de déplacement, c'est à dire la taille du step et indiquez si le déplacement doit éviter les instances de type solide ou encore toutes les instances.

Précisions sur les Actions Principales

Vous trouverez ici les actions principales supplémentaires disponibles dans le mode avancé. Les actions suivantes ont été ajoutées :

 **Définir une ligne de temps (Set a time line)**

(Uniquement disponible dans le mode avancé). Avec cette action, vous définirez la ligne de temps particulière d'une instance d'un objet. Vous mentionnerez la ligne de temps et la position de départ (0 correspond au début). Vous pourrez également utiliser cette action pour stopper une ligne de temps en indiquant **No Time Line** (aucune ligne de temps) comme valeur.

 **Définir la position de la ligne de temps (Set the time line position)**

(Uniquement disponible dans le mode avancé). Cette action vous permet de changer la position de la ligne de temps courante (en absolu ou en relatif). Ceci peut être utilisé pour sauter certaines parties de la ligne de temps ou encore répéter d'autres parties. Par exemple, si vous souhaitez créer une ligne de temps qui boucle, à la fin de la ligne de temps, ajoutez cette action pour régler de nouveau la position à 0. Vous pouvez aussi l'utiliser pour attendre que quelque chose de particulier se produise. Il suffit d'ajouter une action de test et, dans le cas où le résultat serait **faux**, affectez en relatif la valeur -1 à la position de la ligne de temps.

 **Afficher une vidéo (Show a video)**

Cette action vous permet de lire un fichier vidéo ou un film. Vous indiquez le nom du fichier et si la vidéo doit être affichée en plein écran ou en mode fenêtré. Vous devrez vérifier auparavant que le fichier vidéo existe. Vous pourrez soit distribuer la vidéo avec le jeu ou soit la mettre dans un fichier de données puis l'exporter.

 **Remplacer un sprite à partir d'un fichier (Replace a sprite from a file)**

On utilise cette action pour remplacer un sprite avec le contenu d'un fichier. Vous indiquerez le sprite à remplacer, le nom du fichier (.bmp, .jpg, or .gif) et le nombre de sous-images du sprite quand il sera chargé à partir du fichier BMP ou JPG. Pour un fichier GIF, le nombre de sous-images est calculé automatiquement en fonction du nombre de sous-images contenues dans le fichier GIF. Les autres paramètres du sprite, comme par exemple son éventuelle transparence, ne seront pas modifiés. Vous pouvez utiliser cette action pour éviter de stocker tous les sprites dans le programme. Par exemple, au début d'un niveau, vous pourrez remplacer les sprites par ceux de l'actuel personnage que vous souhaitez utiliser. NE PAS changer un sprite en cours d'utilisation

par une instance de la room. Cela pourrait donner des effets indésirables avec les collisions. ***Cette action est uniquement disponible dans la version enregistrée.***

 **Remplacer un son à partir d'un fichier (Replace a sound from a file)**

Avec cette action, il vous sera possible de remplacer un son par le contenu d'un fichier (.wav, .mid, or .mp3). Vous indiquerez le son et le nom du fichier. Cela vous permettra d'éviter de stocker tous les sons dans le jeu lui-même. Par exemple, vous pourrez utiliser différents morceaux de musique d'arrière-plans et choisir celui que vous désirez jouer. NE PAS changer un son lorsque celui-ci est en train d'être joué. ***Cette action est uniquement disponible dans la version enregistrée.***

 **Remplacer un décor à partir d'un fichier (Replace a background from a file)**

Cette action vous autorisera à remplacer un arrière-plan par celui contenu dans un fichier (.bmp, or .jpg). Vous indiquerez l'arrière-plan ainsi que le nom du fichier. De même, ceci vous évitera de stocker tous les décors dans le jeu lui-même. NE PAS changer un décor si ce dernier est encore visible. ***Cette action est uniquement disponible dans la version enregistrée.***

Précisions sur les Actions de Contrôles

Quelques actions de contrôles supplémentaires sont disponibles dans le mode avancé. Les actions suivantes ont été ajoutées :

 **Exécuter un script (Execute a script)**

Vous pouvez, grâce à cette action, exécuter un script que vous aurez ajouté dans votre jeu. Vous indiquerez le nom du script ainsi que 5 arguments au maximum à passer en paramètre au script.

Invoquer l'événement parent (Call the inherited event)

Cette action n'est à utiliser que si l'objet possède un objet parent. Elle invoque (=appelle) l'événement correspondant de l'objet parent.

Précisions sur les Actions d'Affichage

Les actions supplémentaires suivantes relatives à l'affichage sont disponibles dans le mode avancé :

 **Choisir une police pour l'affichage de texte (Set a font for drawing text)**

Vous pouvez choisir une police de caractères qui sera ensuite utilisée pour afficher le texte. Cela doit être l'une des fontes de caractères appartenant aux ressources que vous aurez définies au préalable. Si vous sélectionnez **No Font** (pas de police), la police Arial 12 points sera utilisée.

Les Actions concernant les Particules

Vous pouvez accéder à un ensemble d'actions concernant les particules dans l'onglet **Extra**. ***Ces actions ne sont disponibles que dans la version enregistrée de Game Maker***.

Les systèmes de particule sont utilisés pour créer des effets spéciaux. Les particules sont constituées de petits éléments (représentés par un pixel ou encore une petite forme). Ces particules se déplacent selon des règles prédéfinies et peuvent changer de couleur lors de leurs déplacements. De telles particules sont employées afin de créer des feux d'artifices, des flammes, de la pluie, de la neige, des champs d'étoiles, des débris en vol, etc.

Game Maker contient un système de particules complet auquel on accède grâce à des fonctions (NDT : via le langage GML). Un système de particules plus limité est néanmoins accessible avec les actions décrites ci-dessous.

Un système de particules peut traiter de particules de différents types. Après avoir créé le système de particules, la première chose à faire est d'indiquer les types de particule souhaités. Grâce aux actions ci-dessous, vous pourrez spécifier jusqu'à 16 types de particules. Chaque type possède une forme, une taille, une couleur de départ et d'arrivée. La couleur varie lentement à partir de la couleur de départ jusqu'à la couleur d'arrivée. Les particules se caractérisent par la notion de durée de vie limitée. Vous préciserez dans le type la durée de vie minimale et maximale des particules. Les particules ont aussi une vitesse et une direction. Enfin, la gravité et la friction s'appliquent aux particules.

Après avoir indiqué les types de particule, vous devrez les placer dans la room. Vous pourrez soit générer un certain nombre de particules d'un type particulier à un endroit ou encore créer un flux constant de particules. Les particules apparaissent sous la forme d'émetteurs.

Le système de particule peut disposer d'un maximum de 8 émetteurs fonctionnant simultanément. Aussi, après avoir créé les types de particule, vous devrez créer les émetteurs et leur indiquer de générer ou de créer des flux de particules.

Vous trouverez ici la liste complète des actions disponibles. Le mieux sera d'expérimenter par vous-même afin d'obtenir les effets désirés.

 **Créer le système de particule (Create the particle system)**

Cette action crée le système de particule. Ceci doit être effectué afin de permettre ensuite l'utilisation des autres actions. Il ne vous sera nécessaire de le faire qu'une seule fois. Vous pourrez indiquer la profondeur à laquelle les particules seront affichées ou dessinées. Si vous utilisez une grande profondeur positive, les particules apparaîtront derrière les instances. Si vous utilisez une profondeur négative, elles apparaîtront devant les instances.

 **Détruire le système de particule (Destroy the particle system)**

Cette action détruit le système de particule, libérant toute la mémoire utilisée. N'oubliez pas d'exécuter cette action (par exemple lorsque vous changez de room) car les systèmes de particule consomment beaucoup de mémoire.

 **Effacer toutes les particules du système (Clear all particles in the system)**

Cette action supprime toutes les particules actuellement visibles. Cela n'arrête pas toutefois les émetteurs, aussi vous pourrez créer de nouvelles particules si vous avez des émetteurs de flux (voir ci-dessous).

 **Créer un type de particule (Create a type of particle)**

Avec cette action, vous créerez un type de particule. Vous pourrez choisir l'un des 16 types disponibles. Pour le type de particule souhaité, vous indiquerez sa forme ou bien le sprite à utiliser. Si vous indiquez un **sprite**, c'est le sprite qui sera utilisé. Si vous paramétrez le sprite à la valeur **no sprite** (aucun sprite), la forme sera alors utilisée. Il existe un certain nombre de formes intéressantes déjà intégrées. Vous indiquerez également la taille minimale et maximale (lorsque la particule apparaîtra, une valeur aléatoire comprise entre ces valeurs sera utilisée). Enfin, vous préciserez l'incrément de taille à appliquer à chaque step. Pour diminuer la valeur, indiquez une valeur négative. Veuillez noter que seul un type de particule sera créé et non pas une particule actuelle. Pour cela, vous aurez besoin d'émetteurs (voir ci-dessous).

 **Choisir une couleur pour un type de particule (Set the color for a particle type)**

Une particule peut avoir une couleur (par défaut, la couleur est le blanc). Avec cette action, vous pourrez fixer la couleur à utiliser pour un type particulier. Vous devrez indiquer le type de particule pour lequel la couleur doit être définie. Ensuite, vous mentionnerez comment la couleur doit être

appliquée. Soit une couleur aléatoire sera choisie parmi deux couleurs données ou bien la couleur débutera avec la première couleur puis graduellement, et ce durant la durée de vie de la particule, changera vers la seconde couleur. Les deux couleurs doivent être données. Enfin, vous pourrez indiquer la transparence alpha. Vous spécifierez la transparence au moment de la création de la particule et lorsque celle-ci sera détruite. La transparence variera lentement entre ces valeurs. Il est conseillé habituellement de diminuer la valeur alpha à une valeur au-delà de la durée de vie de la particule.

 **Fixer la durée de vie pour un type de particule (Set the life time for a particle type)**
Une particule existe durant un nombre limité de steps. Elles disparaissent ensuite. Avec cette action, vous réglerez la durée de vie d'un type de particule. Vous donnerez deux valeurs limites et la durée de vie actuelle sera déterminée aléatoirement entre ces deux limites.

 **Régler le déplacement pour un type de particule (Set the motion for a particle type)**
Avec cette action, vous réglerez la vitesse et la direction du mouvement pour un type de particule. De nouveau, vous préciserez les deux valeurs limites. La valeur actuelle sera choisie aléatoirement entre ces deux valeurs. Par exemple, pour qu'une particule se déplace dans une direction aléatoire, indiquez 0 et 360 comme limites pour la direction. Vous avez aussi la possibilité de mentionner une friction. Cette valeur sera soustraite de la vitesse à chaque step jusqu'à ce que cette dernière atteigne 0 (vous pouvez accélérer la vitesse d'une particule en utilisant une friction négative).

 **Fixer la gravité d'un type de particule (Set the gravity of a particle type)**
Avec cette action, vous pourrez indiquer la valeur de la gravité ainsi que sa direction pour un type particulier de particule. 270 correspond vers le bas.

 **Créer des particules secondaires (Create secondary particles)**
Ceci est un peu plus compliqué. Les particules peuvent créer d'autres particules pendant leur durée de vie et lorsqu'elles meurent. Avec cette action, vous pourrez préciser tout cela. Vous pourrez définir le type et le nombre de particules à créer à chaque step durant la durée de vie et pourrez indiquer également le type et le nombre de particules à créer lors de la mort de cette dernière. Soyez cependant très prudent avec cette action. Vous pourriez facilement créer un trop grand nombre de particules de cette manière, ce qui conduirait à ralentir considérablement le système. Pour les nombres, vous pouvez utiliser une valeur négative. Une valeur **x** négative signifiera qu'à chaque step, une particule sera créée avec une chance de -1/x. Ainsi, par exemple, si vous souhaitez générer une particule secondaire tous les 4 steps, utilisez une valeur de -4. Les particules secondaires sont très utiles pour créer des effets comme des jets ou encore des explosions de particules.

 **Créer un émetteur de particule (Create a particle emitter)**

Cette action crée un émetteur de particule. Les particules sont générées par des émetteurs. Vous pouvez disposer de 8 émetteurs au maximum. Choisissez l'émetteur et indiquez la forme de ce dernier ainsi que sa taille et sa position (dans la boîte de dialogue).

 **Détruire un émetteur (Destroy an emitter)**

Cette action détruit l'émetteur indiqué. Veuillez noter que les particules existantes qui appartiennent à cet émetteur ne seront pas supprimées.

 **Générer un nombre de particules à partir d'un émetteur (Burst a number of particles from an emitter)**

Même si vous avez défini un type de particule et un émetteur, il n'y aura pas encore de particules. Vous devrez aussi indiquer à l'émetteur de générer les particules. Avec cette action, vous préciserez à un émetteur particulier de créer un nombre donné de particules d'un certain type. Toutes ces particules ne seront générées qu'une seule fois. Pour le nombre, vous pouvez également utiliser une valeur négative. Une valeur **x** négative indiquera que la particule sera créée avec une chance de -1/x. Ainsi, par exemple, si vous souhaitez générer une particule avec un pourcentage de chance de 25, utilisez une valeur de -4.

 **Créer des jets de particules à partir d'un émetteur (Stream particles from an emitter)**

Avec cette action, vous pourrez indiquer à un émetteur particulier de générer un nombre donné de particules d'un certain type. A chaque step, un certain nombre de particules sera généré, créant un jet continu de particules. L'émetteur continuera de créer des filets de particules jusqu'à ce que vous le détruisiez par vous-même ou que vous lui indiquiez de générer 0 particule. Pour le nombre, vous pouvez aussi utiliser une valeur négative. Une valeur **x** négative signifiera qu'à chaque step, une particule sera créée avec une chance de -1/x. Par exemple, si vous désirez générer une particule tous les 4 steps, utilisez une valeur de -4.

Les Actions Complémentaires

A partir de l'onglet **Extra**, vous pouvez accéder à des actions concernant la gestion de CD. ***Ces actions sont uniquement disponibles dans la version enregistrée de Game Maker***.

 **Lire un CD musical (Play a CD)**

Cette action vous permet de jouer les pistes d'un CD placé dans le lecteur utilisé par défaut. Vous indiquerez les pistes de début et de fin.

 **Arrêter la lecture du CD (Stop the CD)**

Stoppe la lecture du CD.

 **Mets en pause le CD (Pause the CD)**

Effectue une pause de la lecture du CD.

 **Reprendre la lecture du CD (Resume the CD)**

Reprend la lecture du CD actuellement en pause.

 **Tester si un CD est présent dans le lecteur (If a CD exists in the drive)**

En présence d'un CD dans le lecteur utilisé par défaut, l'action suivant ce test sera exécutée.

 **Tester si le CD est en cours de lecture (If the CD is playing)**

Si un CD est en cours de lecture dans le lecteur par défaut alors l'action qui suit ce test sera exécutée.

Enfin, il existe deux actions supplémentaires pouvant être utiles dans certains jeux.

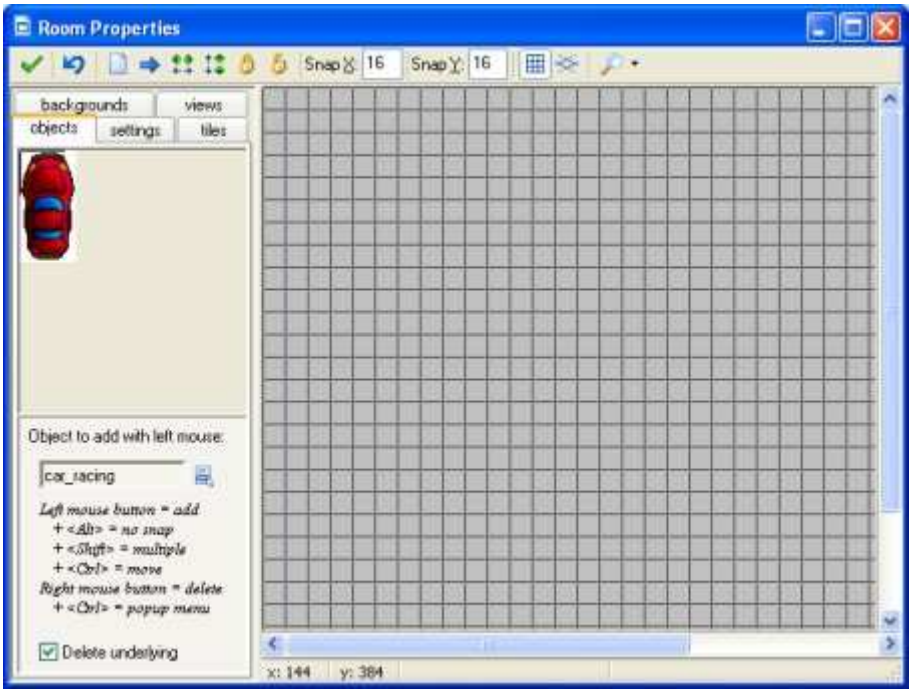
 **Remplacer le curseur de la souris (Set the mouse cursor)**

Vous utiliserez cette action pour remplacer le curseur de la souris de la fenêtre par un sprite. Vous indiquerez le sprite à utiliser et si le curseur de la souris de la fenêtre doit être affiché ou non. Le sprite peut être animé. Veuillez noter que le sprite ne sera affiché que dans la room et non pas en dehors de celle-ci.

 Ouvrir une page web dans un navigateur (Open a webpage in a browser) Vous pouvez indiquer une adresse web avec cette action. Cette page web sera alors ouverte à l'aide du navigateur utilisé par défaut sur la machine (on peut également utiliser cette action pour ouvrir d'autres documents). Cette action ne fonctionne pas dans le mode sécurisé.

Précisions sur les Rooms

Les rooms dans *Game Maker* possèdent beaucoup d'options. Auparavant, nous n'avons parlé que des plus importantes. Dans ce chapitre, nous aborderons les autres options. Lorsque vous ouvrez l'écran de la room dans le mode avancé, vous pourrez voir ceci :



Comme vous pouvez le voir, de nouveaux boutons apparaissent dans la barre d'outils. Il y a des boutons pour trier les instances horizontalement ou verticalement. Cela est utile quand les instances se chevauchent partiellement (lorsque vous ajoutez des tuiles, ces boutons et les autres agissent sur les tuiles plutôt que sur les instances). Il existe aussi des boutons pour verrouiller ou déverrouiller toutes les instances. Les instances verrouillées ne peuvent être déplacées ou effacées. Cela vous évitera de supprimer accidentellement des instances. A l'aide du menu accessible avec le bouton droit de la souris (hold <Ctrl> puis clic droit sur une instance), vous pourrez également verrouiller/déverrouiller des instances individuelles.

Enfin, vous pouvez indiquer si vous souhaitez utiliser une grille isométrique. Cela peut être très utile lors de la création de jeux utilisant des formes isométriques. Les lignes de la grille sont maintenant dessinées en diagonale. De plus, la rupture des instances est différente (cela fonctionne mieux quand l'origine de l'instance se situe en haut dans le coin gauche qui est l'option proposée par défaut).

Apparaissent également deux nouveaux onglets que nous détaillerons ultérieurement.

Des informations sur les différentes options avancées sur les rooms peuvent être trouvées dans les pages suivantes :

Paramétrage avancé
Ajout de tuiles
Vues

Paramétrage avancé

Il y a deux caractéristiques de l'onglet **settings** (paramétrage) que nous n'avons pas encore abordées. La première concerne la case à cocher nommée **Persistent**. Habituellement, lorsque vous quittez une room pour y retourner ultérieurement, les paramètres de la room sont à chaque fois réinitialisés à leurs valeurs initiales. C'est utile si vous avez plusieurs niveaux dans votre jeu mais cela ne sera certainement pas l'effet recherché par exemple dans un jeu de type RPG (NDT : jeu d'aventures). Dans ce cas, les paramètres de la room devront être précisément les mêmes que ceux de la room lorsque vous l'aviez quittée la dernière fois. En cochant la case **Persistent**, c'est ce qui se passera exactement. Le statut de la room sera mémorisé de telle manière que lorsque vous y retournerez plus tard, la room sera exactement identique. Seule une relance du jeu réinitialisera la room. Il existe cependant une exception à ce principe. Si vous marquez certains objets comme persistants, les instances de ces objets ne resteront pas dans la room mais seront placées dans la prochaine room.

La seconde caractéristique a trait au bouton **Creation code** (création de code). Vous pourrez ici taper du code en GML (**G**ame **M**aker **L**anguage -> voir plus loin) qui sera exécuté lors de la création de la room. Cela peut s'avérer utile pour affecter des valeurs à certaines variables de la room, pour créer de nouvelles instances, etc. Il est important de bien comprendre précisément ce qui se passe lorsque vous appelez une room dans le jeu.

- En premier lieu, dans la room courante (si toutefois il y en a plusieurs), toutes les instances reçoivent un signal d'événement de fin de room. Ensuite, les instances non-persistantes sont enlevées (aucun événement de destruction n'est généré ici !).
- Ensuite, dans la nouvelle room, les instances persistantes de la précédente room sont ajoutées.
- Toutes les nouvelles instances sont créées et leurs événements de création sont exécutés (si cependant la room est non persistante ou n'a pas été visitée auparavant).
- Si c'est la première room du jeu et pour toutes les instances, l'événement 'game-start' (début de jeu) est généré.
- Puis, le code de création de la room est exécuté.
- Enfin, toutes les instances reçoivent le signal d'événement 'room-start' (début de room).

Ainsi, par exemple, les événements 'room-start' peuvent utiliser des variables initialisées par le code de création de la room et dans ce code, vous pourrez référencer les instances (aussi bien les nouvelles que celles persistantes) de la room.

Il existe aussi une option plus évoluée. Dans le menu pop-up qui apparaît lors d'un clic droit sur une instance tout en pressant la touche <Ctrl> , vous pourrez également indiquer du code de création pour une instance spécifique. Ce code est exécuté lorsque la room est affichée, juste avant que l'événement de création de l'instance ne soit exécuté. Ceci est très utile par exemple pour régler certains paramètres propres à une instance.

Ajout de tuiles

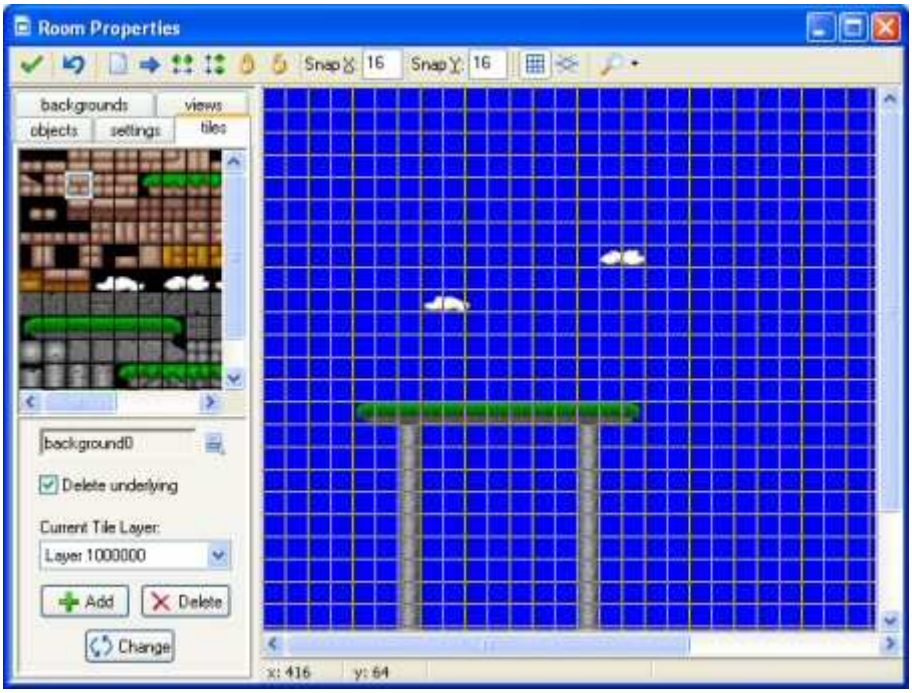
Il vous est possible de créer des arrière-plans comprenant des tuiles (tiled background). La justification en est la suivante : dans la plupart des jeux, vous souhaiterez disposer de jolis arrière-plans. Par exemple, dans un jeu de type labyrinthe, les murs du jeu devront s'emboîter harmonieusement et dans les jeux de plateformes, vous aimerez avoir d'agréables dessins de plateformes, de beaux arbres, etc. Vous pouvez réaliser tout ceci dans *Game Maker* en définissant plusieurs objets différents et en concevant vos rooms à partir de ces objets. Le problème est que cela demande beaucoup de travail, utilise une grande quantité de ressources et ralentit sérieusement les jeux en raison de la présence de nombreux objets différents. Par exemple, pour créer de jolis murs dans des jeux de type labyrinthe, vous aurez besoin au début de disposer de 15 objets murs différents.

Une façon fréquemment utilisée dans de nombreux jeux, est que les murs et les autres objets statiques sont actuellement dessinés sur le décor (ou arrière-plan). Mais vous vous demandez sans doute comment le jeu peut-il savoir si un objet frappe un mur s'il est lui-même dessiné sur l'arrière-plan ? L'astuce est la suivante : vous créerez uniquement un seul objet mur dans votre jeu. Il devra avoir une dimension correcte mais il ne devra pas forcément être joli. Lors de la création de la room, placez cet objet aux endroits où il doit y avoir un mur. Puis, intervient l'astuce qui consiste à rendre cet objet invisible. Ainsi, le joueur ne verra pas les objets de type mur lors du jeu. Bien au contraire, il apercevra un superbe arrière-plan. Mais les objets représentant les murs seront bien présents et les autres objets du jeu réagiront lors d'une collision avec ces objets.

Cette technique peut être utilisée pour les objets ne changeant pas de forme ou de position (vous ne pourrez pas employer cette astuce si l'objet doit être animé). Pour les jeux de plateformes, vous aurez certainement juste besoin d'un objet pour le sol et d'un autre pour le mur, mais vous devrez sans doute réaliser des arrière-plans plus travaillés graphiquement si vous voulez que votre jeu puisse permettre au joueur de marcher sur l'herbe, grimper à des branches d'arbres, etc.

Pour ajouter des tuiles dans votre room, vous aurez besoin d'ajouter une ressource **background** dans votre jeu qui contiendra toutes les tuiles (tiles). Si vous souhaitez avoir des tuiles partiellement transparentes, soyez certain que l'image d'arrière-plan est transparente. Quand vous ajoutez la ressource background, indiquez qu'elle doit être utilisée comme jeu de tuiles (set of tiles). Ensuite, précisez la taille de chacune des tuiles et s'il doit y avoir une room entre les tuiles, comme mentionné dans le chapitre sur les ressources de type **background**.

Maintenant, lors de la création de votre room, cliquez sur l'onglet **tiles**. L'écran suivant apparaîtra (vous pourrez constater que nous avons déjà ajouté quelques tuiles dans la room).



En haut à gauche, nous apercevons le jeu de tuiles actuellement utilisé. Pour sélectionner le jeu complet, cliquez sur le bouton du menu en dessous puis choisissez l'image d'arrière-plan souhaitée.

Vous pouvez désormais ajouter des tuiles en sélectionnant la tuile désirée en haut à gauche, puis en cliquant à l'endroit voulu dans la room située à droite. C'est le même principe utilisé pour ajouter des instances. Les tuiles 'Underlying' sont supprimées sauf si vous avez décoché la case **Delete underlying**. Vous pouvez utiliser le bouton droit pour effacer des tuiles. Maintenir la touche <Shift> pour ajouter plusieurs tuiles en même temps. De même, pressez la touche <Ctrl> pour placer les tuiles à un nouvel endroit de la room. La touche <Alt> évitera de créer une rupture avec la grille. Egalement, il y a un menu pop-up si vous maintenez pressée la touche <Ctrl> puis si vous cliquez sur une tuile avec le bouton droit de la souris. Les boutons de la barre d'outils effaceront ou décaleront toutes les tuiles, trieront les tuiles ou encore les verrouilleront / déverrouilleront (actuellement, cela ne fonctionne que sur la couche (layer) courante -> voir plus bas).

Dans certaines situations, vous désirerez mettre une partie de l'arrière-plan dans la room qui ne correspondra pas exactement à la taille d'une tuile ou comportera plusieurs tuiles. On peut faire cela de la manière suivante. Dans l'image du haut à gauche, pressez le bouton gauche de la souris tout en maintenant appuyée la touche <Alt>. Maintenant, vous pourrez déplacer avec la souris une aire que vous placerez dans la room de la même façon que pour les tuiles. Pour sélectionner plusieurs tuiles, maintenir la touche <Shift>. Veuillez prendre note que cela ne fonctionnera

correctement que s'il n'y a pas de séparation entre les tuiles. Si vous souhaitez choisir une aire qui est un multiple de la taille de la grille de la room, maintenez pressée la touche <Ctrl> plutôt que la touche <Shift> (notez que vous pouvez actuellement changer la touche pressée durant le déplacement à la souris. Cela pourra parfois vous être utile).

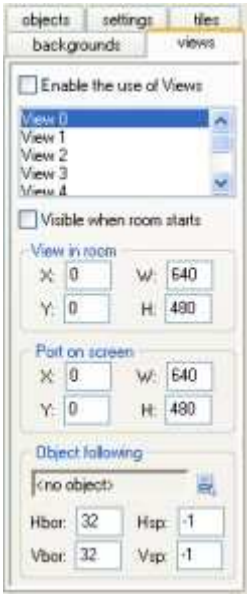
Les tuiles peuvent être placées dans des couches (layers) à différentes profondeurs. En bas, vous verrez la profondeur courante. Par défaut, elle est de 1000000 ce qui est normalement derrière toutes les instances. Autrement dit, les instances se déplaceront devant les tuiles. Vous pouvez employer le bouton **Add** pour ajouter de nouvelles couches de tuiles, avec une profondeur différente pour chacune d'entre elles. Les profondeurs négatives sont utilisées pour mettre des tuiles devant les instances. Si vous donnez également aux objets différentes profondeurs, vous pourrez ainsi les placer entre différentes couches de tuiles. Si vous pressez **Delete**, vous effacerez une couche de tuiles en plus de toutes ses tuiles (il doit toujours exister au moins une couche). Si vous pressez **Change**, vous pourrez modifier la profondeur d'une couche de tuiles. Si vous donnez à la couche la même profondeur qu'une autre couche, les couches seront fusionnées.

L'utilisation de tuiles est une fonctionnalité puissante qui devrait être employée le plus souvent possible. C'est beaucoup plus rapide que d'utiliser des objets et de plus, les images ne sont stockées en mémoire qu'une seule et unique fois. De cette manière, vous pourrez utiliser de grandes rooms avec tuiles tout en consommant très peu de mémoire.

Vues

Il existe également un onglet **views** (vues). Cet onglet vous permettra de disposer d'un mécanisme pour dessiner les différentes parties de votre room et ce à différents endroits de l'écran. Les vues sont utiles à plusieurs titres. Tout d'abord, dans de nombreux jeux, vous souhaiterez afficher exclusivement une partie bien précise de la room à un moment donné. Par exemple, dans la plupart des jeux de plateformes, la vue suit le personnage principal. Dans les jeux à deux joueurs, vous voudrez certainement diviser l'écran en deux parties, une pour chacun des joueurs. Une troisième utilisation possible des vues se rencontre dans les jeux où une partie de la room doit effectuer un scrolling (ex: avec le personnage principal) alors que l'autre plan de la room doit rester fixe (par exemple un panneau affichant le statut du jeu). Tout cela peut facilement être réalisé avec *Game Maker*.

Si vous cliquez sur l'onglet **views**, les informations suivantes seront affichées :



En haut, vous apercevrez une case nommée **Enable the use of Views** (Permettre l'usage des vues). Vous devrez cocher cette case afin de pouvoir utiliser les vues. En dessous, vous verrez la liste des huit vues qu'il est possible de définir. En dessous de cette liste se trouvent les informations sur les vues. En premier lieu, vous indiquerez si la vue doit être visible au lancement de la room. Soyez sûr qu'au moins une vue est visible. Les vues visibles apparaissent en **gras**.

Une vue est définie par une zone rectangulaire dans la room. C'est l'aire de jeu qui doit être affichée dans la vue. Vous préciserez la position du coin supérieur gauche et la largeur ainsi que la hauteur de cette zone. Puis, vous indiquerez à quel endroit l'aire doit être affichée dans la fenêtre

à l'écran. C'est ce que l'on appelle le (view)port. De nouveau, vous mentionnerez la position du coin supérieur gauche ainsi que la taille. Si vous avez une vue simple, la position sera certainement (0,0). Veuillez noter que la taille du port peut très bien être différente de la taille de la vue. Dans ce cas, la vue sera mise à l'échelle pour tenir dans le port (dans un programme GML, il est aussi possible de faire pivoter une vue). Les ports peuvent se chevaucher. Dans ce cas, ils seront affichés dans l'ordre indiqué en haut de chacun d'entre eux.

Comme mentionné plus haut, vous souhaiterez très souvent que la vue suive un objet bien précis. Vous pourrez mentionner cet objet en bas. S'il existe plusieurs instances de l'objet, seul la première sera suivie par la vue (dans un programme en GML, il vous sera possible de demander de suivre une instance particulière). Normalement, le personnage devrait pouvoir marcher un peu sans que la vue ne change. La vue ne changera que lorsque le personnage atteindra les bords de la vue. Vous pourrez indiquer la taille de la bordure devant rester visible autour de l'objet. Enfin, vous pourrez restreindre la vitesse à laquelle la vue doit changer. Cela signifie effectivement que le personnage pourra sortir de l'écran mais cette manière de faire pourra contribuer à améliorer le gameplay du jeu. Utilisez la valeur -1 si vous souhaitez que la vue change instantanément.

Les fontes ou polices de caractères

Lorsque vous souhaitez dessiner du texte dans votre jeu, le texte est affiché par défaut dans la police Arial 12 points. Afin de rendre les textes encore plus spectaculaires, vous souhaiterez certainement employer d'autres fontes de caractères. Pour utiliser des polices différentes, vous devrez créer des ressources de type **fonts**. Dans chacune de ces ressources fonts, vous mentionnerez un type particulier de fontes qui pourra ensuite être utilisé dans le jeu à l'aide d'actions pour paramétrer la police de caractères.

Pour ajouter une ressource **font** dans votre jeu, utilisez l'option **Add Font** du menu **Add** ou encore utilisez le bouton correspondant dans la barre d'outils. L'écran suivant apparaîtra alors.

Comme d'habitude, vous devrez mentionner un nom pour la ressource de fontes. Ensuite, vous pourrez choisir un nom pour la police de caractères. De plus, vous pourrez indiquer sa taille ainsi que si elle doit être affichée en gras et/ou en italique. Soyez conscient que les grandes polices de caractères consomment beaucoup de mémoire. Aussi, il est recommandé de ne pas utiliser de polices avec un pas supérieur à 32 (il est néanmoins possible d'appliquer une échelle sur les fontes de caractères pendant le fonctionnement du jeu). Un exemple de la fonte indiquée est affiché en bas.

Une fonte comprend généralement 256 caractères, numérotés de 0 à 255. Mais en général, vous utiliserez uniquement une infime partie de ceux-ci. Par défaut, dans une fonte, seuls les caractères allant de 32 à 127 sont sauvegardés dans la fonte. Plus il y aura de caractères et plus vous consommerez de mémoire. Vous pourrez modifier la plage de caractères utilisés. Pour voir l'index

de chacun des caractères, vous utiliserez la table des caractères que l'on trouve dans le menu **Démarrer** de Windows, rubrique **Accessoires**/**outils système**. Des plages standards peuvent être indiquées à l'aide des boutons : la plage **Normale** allant de 32 à 127 tandis que la plage **All** s'étend de 0 à 255, la plage **Digits** ne contient que des chiffres, et la plage **Letters** qui comprend quant à elle les lettres majuscules et minuscules. D'autres plages peuvent être utilisées en tapant les premiers et derniers caractères de l'index. Si un caractère manque dans la plage, il sera remplacé par un espace.

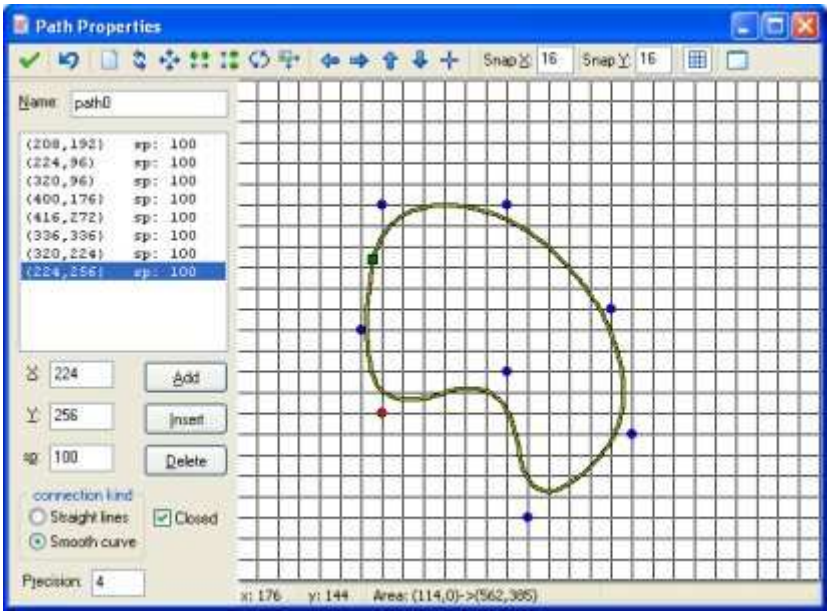
Habituellement, vous devriez disposer d'une grande quantité de fontes différentes déjà installées sur votre ordinateur et de plus, il existe des sites web où vous pourrez télécharger des centaines de polices supplémentaires. Le problème est que si vous utilisez ces polices dans votre jeu et que vous donniez ce dernier à une autre personne, il y aura de forte chance que cette autre personne ne possède pas les mêmes fontes installées sur son ordinateur. Afin d'éviter cette situation, *Game Maker* intègre toutes les fontes que vous utilisez dans le fichier contenant le jeu mais cela est vrai seulement que dans la version exécutable autonome du jeu. Aussi, si vous donnez à une personne la version exécutable de votre jeu, il ne sera donc pas nécessaire de fournir également les fichiers contenant les polices de caractères. Par contre, si vous donnez la version source éditable de votre jeu et que vous utilisez des fontes non standard, il sera préférable de donner à cette personne les fichiers des polices de caractères utilisées.

Les Chemins

Dans les jeux plus avancés, vous désirerez souvent que les instances suivent des chemins particuliers. Bien que cela soit parfaitement réalisable en utilisant des événements de temps ou encore du code GML, ceci reste plutôt compliqué. Les ressources Path (chemins) fournissent un mécanisme plus commode pour réaliser ceci. L'idée est assez simple. Vous définissez un chemin (path) tout simplement en le dessinant. Ensuite, vous placerez une action dans l'événement de création de l'objet pour indiquer à cet objet de suivre le chemin souhaité. Ce chapitre explique en détail ce mode de fonctionnement.

Définition de chemins (Defining paths)

Pour ajouter un chemin dans votre jeu, choisissez **Add Path** du menu **Add menu**. L'écran suivant apparaîtra alors (dans cet exemple, nous avons déjà ajouté un petit chemin).



En haut à gauche de l'écran, vous pourrez donner un nom au chemin comme à l'accoutumée. En dessous, vous trouverez les points qui définissent le chemin. Chaque point possède à la fois une position et une vitesse (indiquées par **sp**). Selon l'utilisation du chemin, la position sera soit absolue, c'est à dire que l'instance utilisée plus tard suivra le chemin à cet endroit bien précis, soit relative; l'instance débutera toujours à la première position du chemin puis suivra le chemin jusqu'à la fin de celui-ci. La vitesse est à interpréter de la façon suivante. Une valeur de 100 signifie que le chemin assigné à l'instance aura une vitesse normale. Une valeur plus faible réduira

la vitesse, une valeur plus grande l'augmentera (cela indique donc le pourcentage de la vitesse actuelle). La vitesse sera interpolée entre les points. Ainsi, la vitesse changera progressivement.

Pour ajouter un point, pressez le bouton **Add**. Une copie de l'actuel point sélectionné sera faite alors. Maintenant, vous pouvez changer l'actuelle position et la vitesse en modifiant les valeurs dans la boîte d'édition. Si vous sélectionnez un point dans la liste, vous pourrez également changer ses valeurs. Pressez **Insert** pour insérer un nouveau point avant le point courant et **Delete** pour effacer le point actuel.

A droite de l'écran, vous apercevrez le chemin actuel. Le point rouge indique le point de contrôle actuellement sélectionné. Les points bleus concernent les autres points de contrôle. Le carré vert signale la position où débute le chemin. Vous pouvez aussi changer le chemin en utilisant la souris. Cliquez quelque part dans l'image pour ajouter un point. Cliquez sur un point existant et déplacez-le à la souris pour modifier sa position. Quand vous maintenez pressée la touche <Shift> tout en cliquant sur un point, vous insérerez un nouveau point. Enfin, vous pouvez utiliser le bouton droit de la souris pour supprimer des points (veuillez noter que vous ne pouvez changer la vitesse de cette façon). Habituellement, les points seront alignés avec la grille. Vous pouvez modifier les paramètres de la grille avec la barre d'outils située en haut. A cet endroit également, vous pourrez indiquer si la grille doit être visible ou non. Si vous voulez positionner un point avec précision, maintenez la touche <Alt> lors des opérations d'ajout ou de suppression.

Vous pouvez influencer la forme du chemin de deux manières. Premièrement, vous pouvez utiliser le type de connexion. Vous pourrez soit choisir une ligne droite ou encore un chemin tout en courbe. Deuxièmement, vous pourrez indiquer si le chemin doit être fermé ou pas.

Dans la barre d'outils, il existe de nombreux contrôles importants. De la gauche vers la droite, les contrôles sont les suivants. Le premier bouton indique que vous êtes prêt et que vous voulez fermer l'écran, tout en conservant les modifications effectuées (si vous ne souhaitez pas les conserver, pressez la croix pour fermer la fenêtre et indiquer que vous ne souhaitez pas effectuer la sauvegarde des modifications). Ensuite, il y a un bouton pour défaire la dernière modification effectuée.

Le jeu de boutons suivants de la barre d'outils vous permet d'effacer le chemin, d'inverser l'ordre dans lequel le chemin sera parcouru, de décaler le chemin, de le copier horizontalement avec un effet de miroir, de le retourner verticalement, de le faire pivoter ou encore de modifier son échelle. Puis, vous trouverez les boutons pour décaler la vue (et non le chemin en lui-même; la zone de la vue courante est indiquée dans la barre de statut en bas) et pour centrer cette dernière.

Comme déjà mentionné plus haut, vous pouvez ensuite paramétrer les valeurs de rupture et indiquer si elles doivent être affichées sur la grille. Enfin, il existe un bouton pour signaler que

vous souhaitez voir une room particulière en guise d'arrière-plan pour ce chemin (vous pourrez ainsi mettre facilement le chemin à un endroit précis de la room, par exemple une piste de course,

de cette façon les instances suivront la bonne route (cela a un sens que lorsque vous utilisez des chemins absolus; voir plus loin)).

Assigner des chemins à des objets (Assigning paths to objects)

Pour assigner un chemin à une instance d'un objet, vous pourrez placer l'action du chemin dans un événement, par exemple dans l'événement de création. Dans cette action, vous devrez indiquer le chemin à partir du menu "drop down". Il y a quelques valeurs que vous pourrez fournir.

Vous devrez mentionner le chemin à suivre et la vitesse en pixels par step. Si la vitesse est positive, l'instance commencera au début du chemin. Si elle est négative, l'instance débutera à la fin. Souvenez-vous bien de cela quand vous souhaiterez définir pour le chemin la vitesse actuelle qui sera relative à la vitesse indiquée. Il y a aussi une action pour changer la vitesse d'exécution du chemin. Vous pourrez, par exemple, utiliser cette action pour ralentir ou accélérer une instance le long du chemin. Veuillez noter que la vitesse normale de l'instance sera ignorée dans ce cas précis (actuellement réglée à 0) quand le chemin sera exécuté. Egalement, certaines choses comme la gravité et la friction n'auront pas d'influence sur le mouvement le long du chemin.

Ensuite, vous indiquerez le comportement de fin, c'est à dire ce qui doit se passer lorsque la fin de chemin est atteinte. Vous pouvez choisir d'arrêter le mouvement et d'aller en fin de chemin. Vous pourrez aussi redémarrer au début du chemin, ce qui forcera donc l'instance à revenir à la position de départ du chemin pour exécuter de nouveau ce chemin. Une troisième option est de relancer à partir de la position courante, ce qui signifie que l'instance suit de nouveau le chemin mais maintenant, sa nouvelle position de départ sera à cet endroit (ce sera la même chose lorsque le chemin sera fermé). Finalement, vous pouvez choisir d'inverser le mouvement, forçant l'instance à faire marche arrière le long du chemin. Notez qu'un événement survient à la fin du chemin; voir plus loin.

Vous pourrez aussi indiquer si le chemin doit être absolu ou relatif. Un chemin absolu est exécuté à l'endroit où il a été défini. L'instance est placée à la position de début et se déplace à partir de cette position (la position de fin quand la vitesse est négative). C'est utile par exemple quand vous avez une piste de course de voiture pour laquelle vous avez défini le chemin. Si vous choisissez relatif, les instances commenceront d'exécuter le chemin à la position actuelle. Utile quand une instance doit faire un déplacement local. Par exemple, les vaisseaux dans un jeu de type **Invader** peuvent effectuer un tour à partir de leur position courante.

Quand vous placez l'instance à un point différent le long du chemin, vous pouvez utiliser l'action pour fixer la position du chemin. Une position de chemin possède toujours une valeur entre 0 et 1, 0 indiquant la position de départ et 1 la position de fin du chemin. Veuillez noter qu'à chaque step, la variable de direction sera automatiquement paramétrée à la direction correcte le long du chemin. Vous pouvez utiliser cette variable afin de choisir la bonne orientation pour les sprites.

Si vous utilisez des scripts ou du code de programmation GML, vous disposerez davantage de contrôle sur l'exécution du chemin. Il existe une fonction pour fixer le début de chemin d'une instance. La variable `path_position` indique la position actuelle du chemin (entre 0 et 1 comme indiqué ci-dessus). La variable `path_speed` indique la vitesse à suivre dans le chemin. Une variable `path_scale` peut être utilisée pour mettre le chemin à l'échelle. Une valeur de 1 correspond à la taille d'origine. Une valeur plus grande signale un chemin plus grand; une valeur plus petite un chemin plus petit. La variable `path_orientation` indique l'orientation vers laquelle le chemin est exécuté (en degrés et dans le sens contraire des aiguilles d'une montre). Cela vous permet d'exécuter le chemin dans des orientations différentes (ex: déplacement en haut et en bas plutôt que de gauche à droite). Il y a également une variable pour contrôler le comportement de fin. Enfin, il existe de nombreuses fonctions pour renseigner les propriétés des chemins (ex: les coordonnées x et y à certaines positions) et l'on trouve des fonctions pour créer des chemins. Il y a même des fonctions qui créent des chemins de collision afin qu'une instance puisse atteindre un but précis. Se reporter aux sections suivantes sur le GML pour obtenir plus de détails à ce sujet.

Vous vous êtes certainement demandé ce qui se passe quand l'instance entre en collision avec une autre instance alors que celle-ci suivait un chemin. C'est le même processus qui survient lorsque l'instance se déplace à une certaine vitesse. Lorsqu'il y a une instance solide, l'instance est placée à son ancienne position. Lorsque les instances ne sont pas de type solide, elles seront placées à leur nouvelle position respective. Ensuite, le ou les événements de collision sont exécutés et une vérification a lieu pour vérifier que la collision a bien été résolue. Dans la négative et si l'autre instance était solide, l'instance s'arrêtera, comme on pouvait s'y attendre (cela suppose toutefois qu'un évènement de collision ait été défini). De plus, la variable `path_position` ne sera pas incrémentée. Lorsque l'instance suspendue disparaît, l'instance continuera à suivre le chemin. Pour gérer les collisions par vous-même, la variable `path_positionprevious` pourra être très utile. Elle contient la précédente position du chemin et vous pouvez lui affecter la valeur de la position du chemin afin d'éviter d'avancer le long du chemin.

L'événement de chemin (The path event)

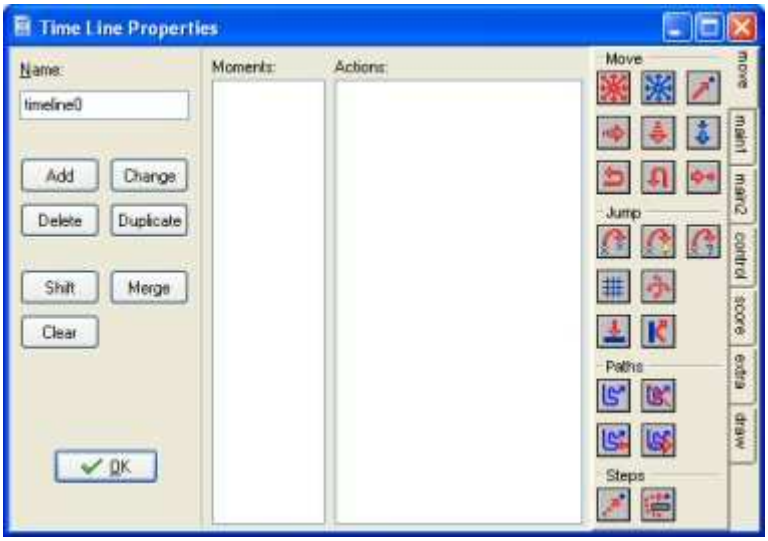
Comme mentionné plus haut, vous pouvez indiquer ce qui doit se passer lorsque l'instance atteint la fin du chemin. A ce moment également, un événement **End of Path** survient. Vous pourrez le

trouver dans les événements **Other**. Vous y placerez des actions. Par exemple, vous pourriez souhaiter détruire l'instance ou encore lui affecter un nouveau (et différent) début de chemin.

Les Lignes de Temps

Dans la plupart des jeux, certaines choses doivent survenir à des moments précis. Pour ce faire, vous pouvez utiliser des événements d'alarmes mais lorsque les choses deviendront trop compliquées, cela ne fonctionnera plus. La ressource de ligne de temps est alors utilisée. Dans une ligne de temps, vous indiquez quelles actions doivent s'effectuer à certains moments dans le temps. Vous pourrez utiliser toutes les actions disponibles pour les différents événements. Une fois votre ligne de temps créée, vous l'affecterez à une instance d'un objet. Cette instance exécutera alors les actions aux différents moments de temps indiqués. Voici un exemple pour mieux comprendre. Supposons que vous vouliez créer un garde. Ce garde doit bouger à gauche pendant 20 steps, puis aller en haut durant 10 steps, à droite pendant 20 steps, puis 10 steps en bas pour enfin s'arrêter. Pour réaliser ceci, vous créerez une ligne de temps où vous indiquerez au début de celle-ci un déplacement vers la gauche. Au moment 20, vous définirez un mouvement vers le haut, au moment 30 un déplacement à droite, au moment 50 un déplacement vers le bas et enfin au moment 60, vous arrêterez le déplacement. Maintenant, vous pouvez assigner cette ligne de temps au garde et ce dernier fera exactement ce que vous aurez prévu. Vous pouvez également utiliser une ligne de temps pour contrôler davantage votre jeu. Créer un objet contrôleur invisible, créer une ligne de temps qui à certains moments, génèrent des ennemis et les affectent à l'objet contrôleur. Dès que vous commencerez à utiliser les lignes de temps, vous trouverez ce concept très puissant.

Pour créer une ligne de temps, choisissez **Add Time Line** à partir du menu **Add**. L'écran suivant apparaîtra.



Cet écran ressemble quelque peu à celui des propriétés d'objets. A gauche, vous pourrez donner un nom et il existe aussi des boutons pour ajouter et modifier les moments de la ligne de temps. Ensuite, vous trouverez la liste des moments. Cette liste indique les moments de temps en steps avec les actions devant survenir. Puis, il y a la liste familière des actions pour les moments sélectionnés et enfin, vous trouverez le jeu complet d'actions disponibles.

Pour ajouter un moment précis, pressez le bouton **Add**. Indiquez le temps du moment (c'est le nombre de steps depuis le démarrage de la ligne de temps). Maintenant, vous pouvez déplacer à la souris les actions dans la liste comme pour les événements concernant les objets. Il existe aussi des boutons pour effacer le moment sélectionné, pour changer le temps du moment choisi, pour dupliquer un moment et pour effacer la ligne de temps.

Enfin , il y a deux boutons spéciaux. Avec le bouton **Merge**, vous pourrez ajouter tous les moments d'un intervalle de temps dans un autre. Avec le bouton **Shift**, vous pouvez déclarer tous les moments d'un intervalle de temps soit plus en avant ou plus en arrière en indiquant une valeur de temps précise. Soyez certain que vous ne créez pas de moments avec des valeurs négatives car ils ne seraient bien entendu jamais exécutés.

Il existe deux actions concernant les lignes de temps.

 **Paramétrer une ligne de temps (Set a time line)**
Cette action vous permettra de paramétrer une ligne de temps particulière pour une instance d'un objet. Vous indiquerez la ligne de temps et la position de départ de la ligne de temps (0 correspond au début). Vous pourrez aussi utiliser cette action pour terminer une ligne de temps en choisissant la valeur **No Time Line**.

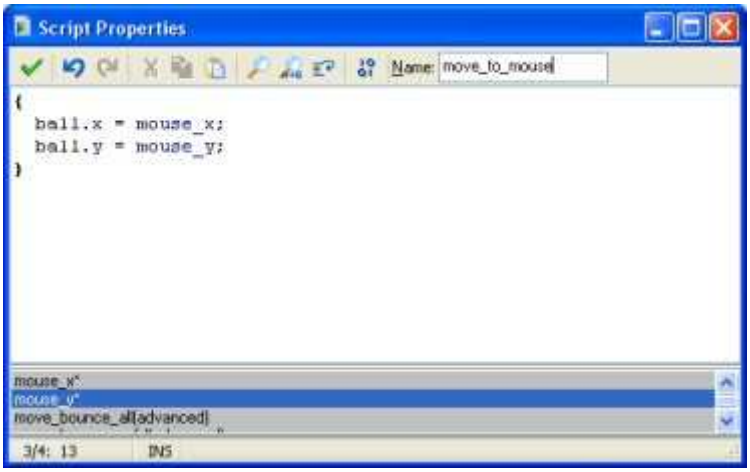
 **Fixer la valeur de la position de la ligne de temps (Set the time line position)**
Avec cette action, vous pouvez changer la position de la ligne de temps courante (soit en absolu soit en relatif). Ceci peut être utilisé afin de sauter certaines parties de la ligne de temps ou encore répéter certaines parties d'entre elles. Par exemple, si vous souhaitez faire une ligne de temps qui boucle indéfiniment , au dernier moment, ajoutez cette action pour fixer la position de nouveau à 0. Vous pourrez également utiliser cette action pour attendre que quelque chose de particulier se produise. Il suffira juste d'ajouter l'action de test et, si cette dernière renvoie FAUX (FALSE), vous réglerez la position de la ligne de temps à une valeur relative de -1.

Scripts

Game Maker comprend un langage intégré de programmation. Une fois que vous connaîtrez suffisamment *Game Maker* et que vous souhaiterez l'utiliser encore plus à fond, vous devrez apprendre l'utilisation de ce langage. Il y a deux façons d'employer le langage. La première manière sera de créer des scripts. Ce sont des morceaux de code auxquels vous donnerez un nom. Ils sont affichés dans l'arbre des ressources et peuvent être sauvegardés en tant que fichier pour bien entendu être rechargés ultérieurement. Ils peuvent aussi être employés pour constituer une librairie qui étendra les possibilités de *Game Maker*. Alternativement, vous pouvez ajouter une action comprenant du code à certains événements en tapant du code à cet endroit. L'ajout d'actions contenant du code fonctionne exactement de la même manière que l'ajout de scripts à deux différences près. Les actions de code ne possèdent pas de nom et ne peuvent utiliser d'arguments. De plus, elles ont un champ particulier qui indique sur quels objets l'action doit s'appliquer. Pour tout le reste, vous entrez le code exactement de la même façon que pour les scripts. Aussi, nous nous concentrerons plus sur les scripts dans ce chapitre.

Comme mentionné auparavant, un script est écrit en code GML (le langage de programmation intégré) et est destiné à effectuer une tâche particulière. Les scripts peuvent avoir des variables en entrée appelées arguments (parfois, on les appelle également paramètres). Pour exécuter un script à partir d'un événement, vous pourrez utiliser soit une action contenant du script ou employer du code. Dans le premier cas, vous indiquerez le nom du script à exécuter, en plus des cinq arguments maximum optionnels. Dans le second cas, ce sera la même chose mais vous appellerez une fonction en GML. Dans ce cas précis, il vous sera possible d'utiliser jusqu'à 16 arguments. Les scripts peuvent retourner une valeur. Cela est souvent utile pour élaborer des méthodes de calculs (mathematical methods). Le mot-clé **return** est utilisé à cet effet. Le code placé après ce mot-clé ne sera jamais exécuté ! Lorsqu'un script retourne une valeur, vous pourrez aussi utiliser cette valeur comme fonction en fournissant des valeurs pour d'autres actions.

Pour ajouter un script dans votre jeu, choisissez **Add Script** à partir du menu **Add menu**. L'écran suivant apparaîtra alors (dans l'exemple, nous avons déjà ajouté un petit script qui initialise deux variables).



(Actuellement, il existe un éditeur de scripts intégré. Dans les préférences, vous pouvez aussi indiquer que vous souhaitez utiliser un éditeur externe). En haut à droite, vous mentionnerez le nom du script. Vous aurez à disposition un petit éditeur dans lequel vous pourrez taper le script. Veuillez noter qu'une liste de toutes les fonctions, des variables internes et des constantes, est affichée tout en bas. Ceci constitue une aide qui vous permettra de trouver l'instruction souhaitée. Vous pouvez double-cliquer sur une instruction pour l'ajouter dans la liste (ou encore utiliser la touche <Ctrl>P). L'affichage de cette liste peut être paramétrée pour s'afficher ou non toujours à partir des préférences. L'éditeur présente de nombreuses propriétés utiles, la plupart étant accessibles à partir de boutons (pressez le bouton droit de la souris pour obtenir des commandes supplémentaires) :

- Plusieurs **Undo** et **Redo** en pressant une touche ou par groupes (peut être modifié dans les préférences)
- Indentation automatique intelligente qui aligne avec la ligne précédente (peut être modifié dans les préférences)
- Gestion intelligente des tabulations qui se positionnent au prochain caractère (différent du caractère **espace**) des lignes précédentes (peut être modifié dans les préférences)
- Utilisation de la touche <Ctrl>I pour indenter les lignes sélectionnées et des touches <Shift> <Ctrl>I pour les désindenter
- Couper et coller
- Recherche et remplacement
- Utilisation des touches <Ctrl> + up, down, page-up, ou page-down pour scroller sans modifier la position du curseur
- Utilisation de la touche de fonction **F4** pour ouvrir le script ou la ressource dont le nom se trouve à la position du curseur (ne fonctionne pas pour les actions comprenant du code; marche uniquement dans les scripts)
- Sauvegarde et chargement de scripts comme fichiers textes

Il existe aussi un bouton avec lequel vous pourrez tester si la syntaxe du script est correcte. Tous les aspects, cependant, ne peuvent être testés à ce stade mais la syntaxe de votre script sera vérifiée, de même que l'existence des fonctions utilisées.

Comme vous l'avez sans doute remarqué, certaines parties du texte du script sont en couleur. L'éditeur sait reconnaître les objets existants, les variables et les fonctions intégrées du langage, etc. La colorisation du code aide beaucoup dans la recherche des erreurs. En particulier, vous pourrez voir immédiatement si vous avez mal orthographié certains noms de commandes ou utilisé un mot-clé en tant que variable. Cependant, la colorisation du code est assez lente. Dans les préférences du menu **Fichier** (file menu), il est possible d'autoriser / d'interdire la colorisation du code. Vous pourrez également ici modifier la couleur des différents composants des programmes (si quelque chose ne devait pas bien fonctionner avec cette fonctionnalité, appuyez deux fois sur **F12**, pour permuter entre les deux états de cette fonction). Vous pouvez aussi changer la police de caractères utilisée dans les scripts et le code.

Les scripts sont extrêmement utiles afin d'étendre les possibilités de *Game Maker*. Cela suppose donc que vous conceviez vos scripts avec précaution. Les scripts peuvent être stockés dans des librairies qui seront ajoutées dans votre jeu. Pour importer une librairie, utilisez l'option **Import scripts** du menu Fichier (file menu). Pour sauver vos scripts sous forme de librairie, utilisez **Export scripts**. Les librairies de scripts sont de simples fichiers textes (malgré qu'elles portent l'extension **.gml**). Il est cependant préférable de ne pas les éditer directement avec un éditeur externe car elles ont une structure spéciale. Certaines librairies avec de puissants scripts, sont déjà incluses avec *Game Maker* (afin d'éviter du travail inutile lors du chargement du jeu, après l'importation d'une librairie, il est très conseillé de supprimer les scripts que vous n'utilisez pas).

Quand vous créez des scripts, vous pouvez très facilement commettre des erreurs. Il faut toujours tester les scripts en utilisant le bouton ad hoc. Lorsqu'une erreur survient pendant l'exécution d'un script, cela est mentionné, avec une indication du type d'erreur commise et à quel endroit celle-ci se trouve. Il est très rare que vous puissiez voir une fenêtre avec le texte "Unexpected error occurred during the game" ("erreur inattendue rencontrée pendant le jeu"). Ce message d'erreur indique qu'un problème est apparue dans Windows ou bien dans le matériel. La raison est due à une boucle récursive infinie, un manque de mémoire ou à cause d'un matériel inadapté pour effectuer les opérations demandées, un problème de pilotes (drivers) ou encore de logiciel (firmware). Pour être plus clair, ces erreurs n'ont pas de liens directs avec des problèmes dus à l'environnement de GM. Si vous avez besoin de vérifier ces choses plus en détails, vous devrez lancer le jeu dans le mode de déboguage (debug mode). Un écran apparaîtra alors avec lequel vous pourrez obtenir beaucoup d'informations sur votre jeu.

A partir du menu **Run**, vous pouvez suspendre le jeu, l'exécuter étape par étape et même relancer son exécution. Dans le menu **Watch**, vous pourrez surveiller la valeur de certaines expressions. Utilisez **Add** pour entrer une expression dont la valeur sera affichée à chaque étape du jeu. De cette manière, vous pouvez voir si le jeu réalise les choses correctement. Vous pouvez surveiller plusieurs expressions. Vous pouvez les sauvegarder pour un usage ultérieur (ex: après que vous ayez fait une correction du jeu). Dans le menu **Tools**, vous trouverez des options pour avoir encore plus d'informations. Vous pourrez ainsi examiner la liste de toutes les instances du jeu, surveiller toutes les variables globales (ce sont sans doute les plus importantes) et les variables locales d'une instance (vous pouvez utiliser soit le nom de l'objet ou l'ID de l'instance). Vous pourrez également voir les messages que votre code transmet en utilisant la fonction `show_debug_message(str)`. Enfin, vous pourrez donner des commandes au jeu et modifier la vitesse de celui-ci. Si vous réalisez des jeux complexes, vous devriez vraiment apprendre à utiliser les options de déboguage (debug options).

La distribution de votre jeu

Quand vous serez prêt pour distribuer votre jeu, vous devrez vous assurer qu'il contient tout ce qui fait de lui un grand jeu. Hormis le jeu en lui-même, cela signifie que vous devrez fournir un fichier d'informations, avoir bien globalement paramétré le jeu et avoir pris soin de bien régler sa vitesse. Cette section vous aidera concernant ces aspects.

De l'information sur la manière de parfaire votre jeu peut être trouvée dans les pages suivantes :

Informations sur le Jeu

TOUT BON JEU doit fournir au joueur quelques informations utiles notamment sur la manière de jouer au jeu. Ces informations seront affichées lorsque le joueur appuiera sur la touche <F1> pendant le jeu. Pour créer une page d'informations sur le jeu, double cliquez sur la ressource **Game Information** sur la partie gauche de l'écran. Un petit éditeur de textes intégré s'ouvrira dans lequel vous pourrez éditer l'information concernant le jeu. Il vous est possible d'utiliser différentes polices de caractères, couleurs et styles. Vous pourrez aussi régler la couleur de l'arrière-plan.

Dans le menu **File**, vous pourrez également paramétrer un certain nombre d'**options**. Vous indiquerez ici le titre de la fenêtre d'information du jeu. Vous pourrez aussi indiquer la position (utilisez le nombre **-1** pour un affichage centré) et la taille de la fenêtre d'information du jeu ainsi que si cette dernière doit avoir une bordure, redimensionnable ou pas par le joueur. Vous pouvez forcer la fenêtre d'informations à rester en haut de l'écran et indiquer si le jeu doit poursuivre son exécution pendant l'affichage de la fenêtre d'informations.

Une option intéressante est de reproduire la fenêtre principale du jeu. Si vous cochez cette option, la fenêtre d'aide sera affichée exactement à la position et à la taille de la fenêtre de jeu. Cela donnera l'illusion que le texte apparaît dans la fenêtre de jeu. Si la couleur d'arrière-plan est bien choisie, cela produira un joli effet visuel (il serait également intéressant que vous indiquiez au joueur de presser la touche **Escape** (ou Esc) pour qu'il puisse retourner au jeu).

Il est important que l'information affichée soit brève et concise. Bien entendu, vous ajouterez votre nom parce que c'est vous qui êtes le créateur du jeu. Tous les exemples de jeu fournis possèdent un fichier d'informations sur le jeu et comment ce dernier a été créé.

Si vous souhaitez créer une aide encore plus agréable, vous pouvez utiliser un programme de traitement de textes comme Word. Sélectionnez ensuite la partie souhaitée puis faites un copier/coller du logiciel Word dans l'éditeur d'information de jeu. Pour les jeux plus avancés, vous n'utiliserez certainement pas ce mécanisme mais emploierez plutôt des rooms spécialement conçues pour afficher de l'information sur le jeu.

Paramétrages généraux du Jeu

Vous pouvez modifier un certain nombre de paramètres de votre jeu. Ces modifications portent sur la présentation de la fenêtre principale, sur certaines options graphiques, ont trait aux réglages d'interactivité entre le jeu et le joueur, au chargement d'images, aux constantes et sur l'information concernant le créateur du jeu. Vous pourrez également indiquer quels doivent être les fichiers à inclure dans vos jeux exécutables autonomes et quelles erreurs doivent être gérées.

Les paramètres peuvent être changés en double cliquant sur **Global Game Settings** dans l'arbre des ressources situé à gauche de l'écran. Ils sont regroupés dans plusieurs onglets (certaines options ne sont disponibles que dans le mode avancé).

De l'information sur les différents réglages peut être trouvée dans les pages suivantes :

Options sur les Graphiques et sur la Fenêtre

A partir de cet onglet, vous pourrez régler certaines options concernant l'apparence graphique de votre jeu. Il est nécessaire de vérifier ces options car elles peuvent avoir un effet significatif sur la présentation du jeu. Souvenez-vous que les joueurs peuvent avoir des machines différentes. Aussi, il est préférable de vérifier que les réglages marcheront également sur les machines de ces personnes.

Lancer le jeu en mode plein écran (Start in fullscreen mode)
Si cette option est validée, le jeu se lancera en plein écran; sinon il fonctionnera en mode fenêtré.

Mise à l'échelle du jeu (Scaling)
Ici, vous indiquerez ce qui doit se passer lorsque la fenêtre est plus grande que la room ou encore lorsque le jeu fonctionne en mode plein écran. Trois choix sont possibles. Vous pourrez indiquer une échelle arbitraire. La room sera alors affichée dans cette échelle en utilisant la valeur fournie et sera centrée dans la fenêtre ou dans l'écran. 100 indique aucune échelle. Vous utiliserez habituellement une échelle arbitraire quand vos sprites et les rooms sont très petits. La seconde option permet de mettre à l'échelle la room de telle manière que celle-ci remplisse entièrement la fenêtre ou l'écran tout en conservant le même ratio d'aspect (rapport entre la largeur et la hauteur). La troisième option fera que la fenêtre ou l'écran sera totalement rempli. Cela pourra créer des déformations de l'image (en particulier dans le mode fenêtré lorsque le joueur voudra redimensionner la fenêtre).

Interpolation des couleurs entre pixels (Interpolate colors between pixels)
Si cette option est cochée, les couleurs des pixels des sprites, des arrière-plans et des tuiles qui ne sont pas alignés avec les pixels de l'écran, seront interpolées. Ceci se produira en particulier en cas de mise à l'échelle, de rotation ou de placement à des positions non entières (non-integer positions). L'interpolation contribuera à rendre les mouvements plus doux mais en contre-partie créera un effet de flou (ceci est également valable pour les tuiles qui présenteront des fissures entre elles, si ces dernières n'ont pas été prévues et conçues pour cet usage).

Couleur en dehors de l'aire de la room (Color outside the room region)
Si la room ne remplit pas complètement la fenêtre ou l'écran, il apparaîtra des zones inutilisées autour de la room. Vous indiquerez ici la couleur de la zone en question.

Autoriser le joueur à redimensionner la fenêtre (Allow the player to resize the game window)
Si cette option est cochée et si nous nous trouvons dans le mode fenêtré, le joueur pourra modifier la taille de la fenêtre de jeu en déplaçant ses coins à l'aide de la souris.

Afficher la fenêtre de jeu toujours au-dessus des autres fenêtres (Let the game window always stay on top)
Si cette option est cochée et si nous nous trouvons dans le mode fenêtré, la fenêtre de jeu apparaîtra toujours par-dessus toutes les autres fenêtres.

Ne pas afficher une bordure dans le mode fenêtré (Don't draw a border in windowed mode)
Si cette option est cochée et si nous nous trouvons dans le mode fenêtré, la fenêtre de jeu ne possédera pas de bordure ni de barre de titre.

Ne pas afficher les boutons dans le titre de la fenêtre (Don't show the buttons in the window caption)
Si cette option est cochée et si nous nous trouvons dans le mode fenêtré, le titre de la fenêtre n'affichera pas les boutons de fermeture et de minimisation/maximisation de la fenêtre.

Afficher le curseur (Display the cursor)
Indique si vous souhaitez rendre visible le pointeur de souris. Le rendre invisible devrait normalement accélérer les déplacements de la souris. (vous avez la possibilité avec *Game Maker* de créer facilement votre propre objet pointeur de souris).

Geler le jeu lorsque l'écran de jeu n'est plus affiché au premier plan (Freeze the game when the form looses focus)
Si cette option est cochée et lorsque le joueur ouvre un autre écran (ex: une autre application), le jeu sera suspendu jusqu'à ce que la fenêtre du jeu réapparaisse au premier plan.

La Résolution d'Ecran

Cet onglet vous permet de régler la résolution d'écran dans laquelle votre jeu sera exécuté. Par défaut, la résolution n'est pas modifiée. Mais parfois, vous souhaiterez lancer le jeu dans une résolution inférieure ou désirerez paramétrer la fréquence du moniteur pour vous assurer que le timing du jeu fonctionne correctement. Si vous voulez changer la résolution, vous devrez cocher la case nommée **Set the resolution of the screen**.

Il y a trois choses que vous pouvez modifier. En premier lieu, la profondeur de couleur. Ceci indique le nombre de bits utilisés pour représenter la couleur d'un pixel. La plupart des machines n'autorisent que 16-bit (High Color) ou 32-bit (Full Color) mais les machines plus anciennes ne permettent que le 8-bit et même parfois uniquement le mode couleur 24-bit. *Game Maker* ne fonctionne que dans les modes de couleur 16-bit et 32-bit. Le mode couleur 32-bit donne les plus belles images mais consomme plus de mémoire et de temps processeur. Si vous désirez que votre jeu puisse tourner sur les machines les plus anciennes, réglez la profondeur de couleur sur 16-bit. Sinon, utilisez le mode couleur 32-bit ou encore ne modifiez rien du tout.

En second lieu, il y a la notion de résolution d'écran, c'est à dire le nombre de pixels (horizontaux et verticaux) de l'écran. Changer la résolution est utile quand par exemple vos rooms sont très petites. Dans ce cas, il peut être utile de réduire la résolution d'écran. Ayez cependant bien en tête que cela aura des effets également sur les autres applications actuellement en fonctionnement. Cela peut en particulier créer des problèmes dans les modes basse résolution. En général, il est préférable de toucher à la résolution d'écran que si le jeu tourne en mode plein écran. *Game Maker* repositionnera automatiquement la résolution à sa valeur d'origine une fois le jeu terminé.

Enfin, vous pourrez modifier la fréquence de rafraîchissement. Ce paramètre indique combien de fois par seconde l'image doit être rafraîchie à l'écran. Si la vitesse de votre room est plus grande que la fréquence de l'écran, tous les steps ne seront pas actuellement visibles. Cela fonctionne mieux si la fréquence est un multiple de la vitesse de la room (si vous indiquez une fréquence trop importante ou non disponible, la fréquence ne sera pas modifiée).

Il existe aussi un réglage nommé **Use synchronization to avoid tearing** (utiliser la synchronisation afin d'éviter le scintillement). Ceci nécessite des explications. L'affichage est réalisé un certain nombre de fois par seconde, selon la fréquence de rafraîchissement actuellement sélectionnée. Si l'affichage d'une room est affichée par moitié, le haut de l'affichage contiendra encore l'ancienne image alors qu'en bas, la partie de la nouvelle image sera déjà affichée. Ce phénomène est appelé le scintillement (tearing). Afin d'éviter cela, vous pouvez cocher

cette option. Dans ce cas, la nouvelle image de la room sera seulement copiée à l'écran lorsque le rafraîchissement n'est pas en dehors de la fenêtre de jeu évitant ainsi le scintillement (tearing)

la plupart du temps. L'inconvénient est que normalement, il faudra attendre jusqu'au prochain rafraîchissement. Cela veut dire que le nombre maximal d'images est limité par la fréquence du moniteur et que, lorsque le traitement de ce processus n'est pas assez rapide, le taux d'images tombe immédiatement à la moitié de ce nombre. Il peut aussi y avoir un conflit entre le timing interne du jeu et la synchronisation. Si vous souhaitez obtenir le meilleur réglage, réglez la fréquence du moniteur par exemple à 60 Hz et modifiez si besoin la vitesse de la room à 30 ou encore à 60.

Autres Options Diverses

Vous trouverez ici des options de réglages supplémentaires. En premier lieu, vous pourrez paramétrer les valeurs par défaut de certaines touches :

Paramétrer la touche <Esc> Fin de jeu (End the game)
Si cochée, l'appui sur la touche **escape** (Esc) terminera le jeu. Pour la plupart des jeux, cependant, cela ne sera pas souhaitable car certains jeux veulent effectuer des traitements (comme la sauvegarde) avant de fermer le jeu. Dans ce cas, décochez cette case et fournissez vos propres actions lors de l'emploi de la touche **escape** (cliquer sur la croix de la fenêtre génère également un événement touche escape).

Paramétrer la touche <F1> Afficher l'information sur le jeu (show the game information)
Si cochée, cette option affichera des informations sur le jeu lors de la pression de la touche **F1**.

Paramétrer la touche <F4> Basculer entre les modes d'écran (switch between screen modes)
Si cochée, la touche **F4** basculera entre les modes plein écran et fenêtré.

Paramétrer la touche <F5> Sauvegarder le jeu (save the game) et <F6> Charger un jeu (load a game)
Si cochée, le joueur pourra utiliser la touche <**F5**> pour sauvegarder la situation courante du jeu et la touche <**F6**> pour charger le dernier jeu sauvegardé (veuillez noter que seules les données de base du jeu seront sauvées. Lorsque vous utiliserez des fonctionnalités avancées comme les particules ou encore les structures de données, ces paramètres ne seront pas sauvés et vous devrez par vous-même créer un mécanisme de sauvegarde.

Vous pourrez aussi changer la priorité du processus du jeu. Cette priorité indique combien de temps le processeur doit consacrer au jeu. En fonctionnement normal, le système d'exploitation essaie de donner à chacun des processus un temps processeur raisonnable et réparti. Plus la priorité sera grande et davantage de temps processeur sera affecté à votre jeu, permettant à ce dernier de fonctionner plus rapidement et de manière plus régulière. Mais les autres processus disposeront par conséquent de moins de temps pour leur exécution (ainsi, les processus Windows comme ceux de la souris ralentiront leur exécution). A utiliser avec précaution.

Options de Chargement

Vous indiquerez ici ce qui doit se passer lors du chargement du jeu. Premièrement, vous pourrez mentionner votre propre image de chargement. Deuxièmement, vous pourrez indiquez si l'on doit afficher une barre de progression avec une image en bas. Vous avez trois options. Soit ne pas afficher de barre de chargement ou d'afficher la barre par défaut ou encore de préciser deux images : celle d'arrière-plan et celle d'avant-plan lors du chargement. Vous pourrez également préciser si la barre de chargement en avant-plan doit être mise à l'échelle (par défaut) ou insérée ave un effet d'élargissement. Dans le deuxième cas, soyez sûr que l'image proposée soit assez grande pour remplir la barre (notez que les deux images devront absolument être mentionnées dans ce cas).

Il est possible de préciser que l'image chargée doit être transparente. Dans ce cas, le pixel du bas à gauche de l'arrière-plan de l'image sera utilisé comme couleur transparente. La valeur alpha de transparence peut aussi être indiquée. Une valeur de 0 signifie une transparence totale. Une valeur de 255 indique une opacité totale (ces deux paramètres ne fonctionnent que sous Windows 2000, XP, ou supérieur).

En second lieu, vous pourrez indiquer l'icône à utiliser pour les programmes de jeu exécutables. Vous pouvez utiliser uniquement des icônes de taille 32x32. Si vous essayez de choisir un autre type d'icône, vous obtiendrez un message d'erreur.

Enfin, vous pourrez modifier l'ID unique du jeu. Cet ID est utilisé pour sauvegarder la liste des plus hauts scores et sauver les fichiers de jeu. Si vous distribuez une nouvelle version de votre jeu et ne souhaitez pas écraser l'ancienne liste des highscores, vous devrez alors changer ce nombre.

Constantes

A partir de cet onglet, vous pourrez définir les constantes globales qui pourront être utilisées dans tous les scripts et le code ou encore comme valeurs pour les actions. Chaque constante possède un nom et une valeur. Les noms doivent suivre les mêmes règles que les variables, à savoir qu'elles doivent débuter avec une lettre ou un symbole **underscore** (**_**) puis être complétées avec des lettres, des chiffres ou encore des symboles underscore. Il est fortement recommandé de rendre vos constantes facilement différentiables. Il existe une convention qui n'utilise que les lettres capitales et les underscores.

La valeur d'une constante devrait être une expression constante. C'est à dire soit un nombre constant ou une chaîne de caractères (entourée de quotes) ou encore une expression. L'expression est évaluée avant toute chose dans le jeu. Ainsi, par exemple, la constante ne peut référencer l'actuelle room, d'autres instances ou des scripts. Mais elle peut contenir des références à des constantes intégrées et à des noms de ressources.

Vous pouvez ajouter une constante en utilisant le bouton **Add** et les effacer avec le bouton **Delete**. Vous pouvez modifier le nom d'une constante ou sa valeur en cliquant sur la constante. Il existe aussi des boutons pour effacer toutes les constantes ou pour trier leur nom.

Fichiers à inclure dans les Jeux Autonomes

Comme indiqué plus haut, vous pouvez créer des versions autonomes de votre jeu. Parfois, votre jeu utilisera des fichiers supplémentaires. Par exemple, vous pourrez souhaiter inclure des fichiers vidéo ou des fichiers textes qui seront utilisés dans le jeu. Dans certains cas, vous désirerez ajouter des fichiers DLL ou des images, des sons qui seront chargés lors du jeu. Il est bien sûr possible de fournir ces fichiers intégrés dans le fichier exécutable mais il est souvent plus commode de les joindre sous forme de fichiers séparés. De cette façon, seuls les fichiers nécessaires seront distribués.

Vous pouvez mentionner ici les fichiers que vous voulez joindre dans l'exécutable. En haut de l'écran, se trouve une liste de tous les fichiers à inclure. Utilisez **Add** pour sélectionner les fichiers à ajouter à la liste (vous pouvez choisir plusieurs fichiers). Utilisez **Delete** ou **Clear** pour supprimer les fichiers de la liste (pensez cependant que les fichiers ne seront pas inclus dans le fichier .gm6, mais seulement leur nom. Aussi, lorsque vous enverrez à une autre personne le fichier .gm6 source, vous devrez également donner les fichiers à inclure).

Les fichiers de la liste sont compactés dans l'exécutable. Quand le jeu est lancé, ces fichiers sont décompactés et sont ainsi accessibles par le jeu. Il est important de bien comprendre ce qu'il se passe alors. Il y a deux possibilités pour lesquelles vous devrez faire un choix. Dans la situation par défaut, les fichiers sont décompactés dans le répertoire où se trouve le jeu exécutable. C'est ce que l'on appelle le répertoire courant (ou de travail) du jeu. De cette manière, le jeu peut utiliser uniquement les noms de fichiers pour y accéder (aucun chemin n'est donc nécessaire). Cela fonctionne bien si l'exécutable du jeu est sauvé sur un disque dur mais cela ne marche pas si le jeu est stocké sur un périphérique en lecture seule comme par exemple un CD Rom.

La deuxième façon de procéder est d'indiquer que les fichiers doivent être décompactés dans un répertoire temporaire qui sera créé à l'occasion juste pour le fonctionnement du jeu. Si vous choisissez cette option, il vous sera nécessaire de fournir le chemin de ce répertoire temporaire à l'intérieur même de votre jeu. Ce chemin peut être obtenu en utilisant la variable interne `temp_directory`. N'oubliez pas d'ajouter le caractère **backslash** (\ ) dans ce cas. Dans le cas d'un fichier vidéo, vous pourriez taper par exemple le code programme suivant :

```
{
  show_video(temp_directory+'\movie.avi',true,true);
}
```

Comprenez bien que le répertoire temporaire sera effacé à la fin du jeu. Aussi, vous ne pourrez pas sauvegarder les jeux ou encore des informations spéciales dans ce cas de figure. Ne choisir cette option que si vous souhaitez que le jeu fonctionne à partir d'un CD Rom ou si votre jeu n'écrit pas dans des fichiers.

Si un fichier décompacté existe déjà, il ne sera pas normalement enregistré sur le disque. Vous pouvez changer ce comportement en cochant **Overwrite existing files**. Ainsi, quand le jeu sera terminé, les fichiers ne seront normalement pas effacés (sauf s'ils étaient dans le répertoire temporaire qui lui est complètement supprimé). Vous pourrez modifier cela en cochant **Remove at game end**.

Juste un dernier mot pour vous prévenir de ne pas commettre une certaine erreur. Si vous testez votre jeu, le répertoire courant du jeu est le répertoire où le fichier .gm6 est sauvegardé. Si vos fichiers sont également présents dans ce répertoire, et que vous ayez choisi de les supprimer à la fin du jeu, vous pourriez ainsi tous les perdre ! Il est donc préférable de ne pas sauvegarder ces fichiers avec le fichier .gm6 !

Options sur les Erreurs

Vous pourrez ici paramétrer un certain nombre d'options ayant traits à la manière dont les erreurs seront signalées.

Afficher les messages d'erreurs (Display error messages)
Si cette option est cochée, les messages d'erreurs seront affichés au joueur. Dans la version finale du jeu, vous souhaiterez certainement décocher cette option.

Enregistrer les messages d'erreurs dans le fichier `game_errors.log` (Write error messages to file `game_errors.log`)
Si cochée, tous les messages d'erreurs seront écrits dans le fichier `game_errors.log` situé dans le répertoire du jeu.

Arrêter le jeu en présence d'un message d'erreur (Abort on all error messages)
Normalement, certaines erreurs seront fatales alors que d'autres pourront être ignorées. Si cette option est cochée, toutes les erreurs seront considérées comme fatales et conduiront à suspendre le jeu. Dans la version finale du jeu que vous distribuerez, vous souhaiterez certainement cocher cette option.

Initialiser à 0 les variables non déclarées (Treat uninitialized variables as 0)
Une erreur courante est d'utiliser une variable avant qu'une valeur ne lui soit assignée. Parfois, c'est un problème qui sera difficile à éviter. En cochant cette option, les variables non initialisées ne généreront plus d'erreur mais seront traitées comme des variables initialisées à 0. Soyez cependant très prudent. Cela ne signifiera pas pour autant que vous n'ayez pas commis d'erreur de saisie.

Informations concernant le Jeu

Vous pourrez indiquer ici des renseignements sur le jeu comme le nom de l'auteur, la version du jeu et certaines informations générales. La date de dernière modification du jeu est également gérée. Cela peut être utile si vous développez des jeux à plusieurs ou encore si vous réalisez une nouvelle version ou une mise à jour. Les informations ne sont pas accessibles lors de l'exécution du jeu.

Considérations sur la vitesse de jeu

Si vous projetez de réaliser des jeux complexes, vous voudrez probablement les rendre aussi rapides que possible. Bien que *Game Maker* fera le maximum pour que les jeux soient rapides, cela dépendra beaucoup de la manière à laquelle vous avez conçu votre jeu. Bien entendu, il est vrai qu'il est plus facile de créer un jeu qui consomme trop de mémoire que l'inverse. Dans ce chapitre, je vous indiquerai certaines astuces qui pourront contribuer à rendre vos jeux plus rapides et plus petits en taille.

Premièrement, soyez attentif en ce qui concerne les sprites et les arrière-plans (backgrounds) utilisés. Les sprites animés prennent beaucoup de mémoire et dessiner de nombreux sprites prend beaucoup de temps. Aussi, créez vos sprites aussi petits que possible. Supprimez toutes les zones invisibles autour de ces derniers (la commande **crop** de l'éditeur de sprites le fait automatiquement). C'est la même chose en ce qui concerne les images d'arrière-plan. Si vous avez un arrière-plan plein, vérifiez que vous avez désactivé l'utilisation de la couleur d'arrière-plan.

Si vous êtes en mode plein écran, soyez sûr que la taille de la room (ou la fenêtre) ne soit jamais plus large que la taille de l'écran. La plupart des cartes graphiques savent efficacement mettre à l'échelle les images mais la mise à l'échelle de ces images ralentira cependant l'affichage ! Autant que possible, désactivez l'affichage du curseur. Lui aussi ralentit l'affichage des graphiques.

Soyez également prudent lors de l'utilisation des vues (views). Pour chaque vue, la room sera redessinée.

En plus des graphiques, il y a aussi d'autres aspects qui peuvent influencer la vitesse du jeu. Veillez pour qu'il y ait le moins possible d'instances. En particulier, détruisez les instances qui ne sont plus nécessaires (ex: lorsque vous quittez la room). Evitez de placer beaucoup d'actions dans l'événement step ou dans l'événement d'affichage (drawing event) des instances. Souvent, il n'est pas indispensable de les placer à chaque step. L'interprétation du code est assez rapide mais c'est tout de même du code interprété. De plus, certaines fonctions et actions prennent beaucoup de temps; en particulier celles qui doivent vérifier toutes les instances (comme par exemple l'action de rebondissement (bounce action)).

Réfléchissez sur la façon de gérer les événements de collision. Vous devriez avoir normalement deux options. Les objets qui n'ont pas d'événement de collision seront exécutés plus rapidement, aussi il est préférable de traiter les collisions avec les objets qui possèdent peu d'instances.

Soyez également attentif lors de l'utilisation de gros fichiers sons. Ils consomment beaucoup de mémoire et se compressent assez mal. Vous devriez analyser davantage vos sons afin de voir s'ils ne peuvent pas être mieux échantillonnés.

Enfin, si vous voulez réaliser un jeu avec lequel la plupart des gens puissent jouer, testez-le sur les plus vieilles machines.

Le Langage de Game Maker (GML)

Game Maker comprend un langage de programmation intégré. Ce langage de programmation vous apporte beaucoup plus de flexibilité et de contrôle que le permettent les actions standards. Ce langage sera désigné dans cette documentation sous le nom de GML (the ***G****ame* ***M****aker* ***L****anguage*). Dans cette section, nous décrirons le langage GML et donnerons un aperçu de toutes les fonctions (1000 environ) et variables disponibles, destinées à contrôler tous les aspects de votre jeu.

Des informations sur le GML peuvent être trouvées dans les pages suivantes :

Aperçu du langage GML

Game Maker comprend un langage de programmation intégré. Ce langage de programmation vous apporte beaucoup plus de flexibilité et de contrôle que le permettent les actions standards. Ce langage sera désigné dans cette documentation sous le nom de GML (***G**ame **M**aker **L**anguage*). Il existe plusieurs façons de taper des programmes dans ce langage. Premièrement, quand vous élaborerez des scripts. Un script est un programme en GML. Deuxièmement, lorsque vous ajouterez des actions sous forme de code dans un événement. Dans une action, il vous sera possible d'insérer un programme en GML. Troisièmement, dans le code de création d'une room. Et enfin, quand vous aurez besoin de préciser une valeur dans une action, vous pourrez également utiliser une expression en GML. Une expression, comme nous le verrons plus loin, n'est pas vraiment un programme en soi mais plutôt le résultat de l'exécution de code affecté à une valeur.

Dans ce chapitre, nous décrirons la structure de base des programmes en GML. Lorsque vous souhaiterez utiliser des programmes en GML, il y aura certaines choses pour lesquelles vous devrez faire attention. En premier lieu, vous devrez nommer toutes vos ressources (sprites, objets, sons, etc.) par un nom commençant par une lettre et ne comprenant que des lettres, nombres ou le symbole underscore '_' . Sinon, vous ne pourrez pas les référencer dans votre programme. Soyez sûr que toutes les ressources disposent chacune d'un nom différent. Ainsi, n'utilisez pas par erreur des noms de ressources comme **self**, **other**, **global** ou tout autre chose ayant une signification particulière pour le langage. Bien entendu, vous ne devrez pas utiliser l'un des mots-clés énumérés ci-dessous.

Des informations sur le langage GML peuvent être trouvées dans les pages suivantes :

Un Programme

Un programme est constitué d'un certain nombre d'instructions. Un programme doit débuter avec le symbole '**{**' et se terminer avec le symbole '**}**'. Entre ces symboles se trouvent les instructions. Les instructions doivent être séparées par le symbole '**;**'. Ainsi, la structure générale d'un programme est la suivante :

```
{
  <instruction>;
  <instruction>;
  ...
}
```

Il existe différents types d'instructions qui seront détaillées plus loin.

Les variables

A l'instar de tout langage de programmation, le GML comporte des variables. Les variables sont des emplacements mémoires destinés à sauvegarder de l'information. Ces emplacements se distinguent par leur nom respectif que vous pourrez utiliser dans un programme. Une variable en GML peut stocker un nombre réel ou encore une chaîne de caractères. Les variables peuvent ne pas être déclarées comme dans certains autres langages. Il existe une grande quantité de variables internes. Certaines sont de portée générale comme **mouse_x** et **mouse_y** qui indiquent la position actuelle de la souris, alors que d'autres sont locales aux instances d'objets pour lesquelles nous exécutons le programme, comme **x** et **y** qui indique la position courante de l'instance. Une variable possède un nom qui débute par une lettre et ne peut contenir que des lettres, des nombres ou le symbole underscore **'_'** (la longueur maximale est de 64 caractères). Lorsque vous utilisez une nouvelle variable, celle-ci sera de portée locale par rapport à l'instance courante et demeurera inconnue des programmes des autres instances (même s'il s'agit du même objet). Vous pourrez faire référence aux variables dans les autres instances; voir plus loin.

Les Affectations (Assignments)

Une affectation (assignment) stocke une valeur dans une variable. Une affectation a la forme suivante :

```
<variable> = <expression>;
```

Une expression peut être une simple valeur mais peut aussi être beaucoup plus compliquée. Plutôt que d'affecter une valeur à une variable, on peut également ajouter une valeur à la valeur actuelle d'une variable en utilisant **+=**. De façon similaire, il vous est possible de soustraire une valeur en employant **-=**, de la multiplier avec ***=**, de la diviser grâce à **/=** ou d'utiliser les opérateurs fonctionnant bit à bit comme **|=**, **&**, ou **^=**.

Les Expressions

Les expressions peuvent être des nombres réels (ex: **3.4**), des valeurs hexadécimales, débutant par un signe **$** (ex: **$00FFAA**), des chaînes de caractères entourées de simple ou de double quotes (ex: **'hello'** ou **"hello"**) ou encore des expressions plus compliquées. En tant qu'expressions, les opérateurs binaires suivants existent (par ordre de priorité) :

- **&& || ^^**: traite des valeurs Booléennes **(&&**; = **et**, **||** = **ou**, **^^** = **xor**)
- **< <= == != > >=**: comparaisons, résultat VRAI (1) ou FAUX (0) (True ou False)
- **| & ^**: opérateurs fonctionnant bit à bit (**|** = **ou** bit à bit, **&** = **et** bit à bit, **^**= **xor** bit à bit)
- **<< >>**: opérateurs bit à bit (**<<** = **décalage à gauche**, > > = **décalage à droite**)
- **+ -**: addition, soustraction
- *** / div mod**: multiplication, division, division entière et modulo

Veuillez noter que la valeur de **`x div y`** est la valeur de **x/y** arrondie à l'entier inférieur le plus proche. L'opérateur **`mod`** retourne le reste obtenu en divisant les opérandes. Autrement dit, **`x mod y = x - (x div y) * y`**. Il existe également les opérateurs unaires suivants :

- !: **non**, transforme VRAI (true) en FAUX (false) et FAUX en VRAI
- -: valeur opposée de la valeur fournie (inversion)
- ~: inverse la valeur bit à bit

Comme valeurs, vous pouvez utiliser des nombres, des variables ou encore des fonctions retournant une valeur. Des sous-expressions peuvent être placées entre crochets. Tous les opérateurs travaillent avec des valeurs réelles. Les comparaisons fonctionnent aussi avec les chaînes et le signe **+** concatène des chaînes de caractères (veuillez noter que, contrairement à certains langages, les deux arguments d'une opération Booléenne seront toujours calculés, même si le premier argument détermine le résultat produit).

Exemple

Voici un exemple présentant quelques affectations.

```
{
  x = 23;
  color = $FFAA00;
  str = 'hello world';
```

```
    y += 5;
    x *= y;
    x = y << 2;
    x = 23*((2+4) / sin(y));
    str = 'hello' + " world";
    b = (x < 5) && !(x==2 || x==4);
}
```

Les autres Variables (Extra variables)

Vous pouvez créer de nouvelles variables en leur assignant simplement une valeur (il n'est pas nécessaire de les déclarer au préalable). Si vous utilisez simplement un nom de variable, la variable ne sera connue que dans l'instance courante de l'objet. Aussi, il ne sera pas possible d'utiliser cette variable ultérieurement avec un autre objet (ou une autre instance du même objet). Vous pouvez également initialiser ou lire les variables des autres objets en mentionnant le nom de l'objet suivi d'un point puis du nom de la variable.

Pour créer des variables globales, qui seront visibles par toutes les instances d'objets, précédez les variables avec le mot `global` suivi d'un point. Par exemple, vous pourrez écrire :

```
{
  if (global.doit)
  {
    // Effectuer quelque chose...
    global.doit = false;
  }
}
```

Vous souhaiterez disposer parfois de variables uniquement dans le code ou le script courant. De cette façon, vous éviterez de gaspiller de la mémoire et serez sûr de ne pas avoir de conflit avec les noms des autres variables. En outre, c'est plus rapide que l'utilisation de variables globales. Pour finir, vous devrez déclarer les variables au début de votre code en utilisant le mot-clé **var**. La déclaration pourra ressembler à ceci.

```
var <nom1 variable>,<nom2 variable>,<nom3 variable>, ...
```

Par exemple, vous pourrez écrire :

```
{
  var xx,yy;
  xx = x+10;
  yy = y+10;
  instance_create(xx,yy,ball);
}
```

Adressage de variables dans les autres instances

Comme décrit plus haut, vous pouvez donner une valeur à une variable de l'instance courante en employant des instructions comme

```
x = 3;
```

Mais dans de nombreux cas, vous voudrez adresser les variables des autres instances. Par exemple, vous souhaiterez arrêter le déplacement de toutes les balles, ou encore déplacer le personnage principal à une position particulière, ou, dans le cas d'une collision, utiliser le sprite de l'autre instance concernée. Ceci peut être réalisé en précédant le nom de la variable avec le nom d'un objet suivi d'un point. Par exemple, vous pourrez écrire

```
ball.speed = 0;
```

Cela modifiera la vitesse de toutes les instances de l'objet **ball**. Il existe certains ***objets spéciaux*** :

- `self`: L'instance actuelle pour laquelle nous exécutons l'action
- `other`: L'autre instance concernée dans un événement de collision
- `all`: Toutes les instances
- `noone`: Aucune instance (cela peut paraître étrange mais cela a son utilité comme nous le verrons plus tard)
- `global`: Pas une instance mais un container qui stocke les variables globales

Ainsi, par exemple, vous pourrez utiliser les instructions suivantes :

```
other.sprite_index = sprite5;
all.speed = 0;
global.message = 'A good result';
global.x = ball.x;
```

Vous vous demandez certainement ce que fera la dernière affectation en présence de plusieurs balles. Et bien, la première sera prise en compte et sa valeur **x** sera affectée à la variable globale.

Mais alors, comment faire pour régler la vitesse d'une balle particulière ? Cela est un peu plus difficile. Chaque instance possède un ID unique. Lorsque vous placez des instances dans une room

à l'aide de l'éditeur de room, l'ID de cette instance est affiché quand vous gardez le pointeur de la souris sur l'instance. Ce sont des nombres supérieurs ou égaux à 100000. Vous pouvez également utiliser ce nombre en le saisissant à gauche du point. Mais soyez prudent ! Le point sera interprété comme le point décimal du nombre. Afin d'éviter ce problème, mettez des parenthèses autour de l'ID. Par exemple, supposons que l'ID de la balle soit 100032, vous pourrez alors écrire :

```
(100032).speed = 0;
```

Lorsque vous créez une instance dans un programme, la valeur de retour sera égal à l'ID. Il est donc possible d'écrire ceci dans un programme :

```
{
  nnn = instance_create(100,100,ball);
  nnn.speed = 8;
}
```

Ceci aura pour effet de créer une balle et de régler sa vitesse. Veuillez noter que nous avons affecté l'ID de l'instance à une variable puis utilisé cette variable juste avant le point. Cela est parfaitement valide. Soyons plus précis. Un point est un opérateur. Il admet une valeur comme opérande de gauche et une variable (adresse) comme opérande de droite puis retourne l'adresse de cette variable particulière dans l'objet ou l'instance indiqué. Tous les noms d'objet, ainsi que les objets spéciaux mentionnés ci-dessus, représentent simplement des valeurs et peuvent donc être utilisés comme n'importe quelle valeur. Par exemple, le programme ci-dessous est tout à fait valide :

```
{
  obj[0] = ball;
  obj[1] = flag;
  obj[0].alarm[4] = 12;
  obj[1].id.x = 12;
}
```

La dernière instruction est à lire de la façon suivante. Nous prenons l'ID du premier élément. Pour toutes les instances concernées par cet ID, nous affectons 12 à la coordonnée **x**.

Les noms d'objet, les objets spéciaux et les ID d'instances peuvent également être utilisés dans de nombreuses fonctions. Ils seront considérés comme des constantes dans les programmes.

Les tableaux

Vous pouvez utiliser des tableaux à 1 ou 2 dimensions avec le langage GML. Il vous suffit de mettre l'indice entre crochets pour un tableau à 1 dimension et les deux index séparés par une virgule pour les tableaux à 2 dimensions. Le fait d'utiliser un indice crée automatiquement le tableau. Un tableau débute à l'index 0. Aussi, soyez prudent en utilisant des index élevés car une grande quantité de mémoire sera alors réservée. Ne jamais employer d'indices négatifs. Le système limite la taille de chacun des index à 32.000 et à 1.000.000 pour l'ensemble des indices. Par exemple, il vous sera possible d'écrire :

```
{
  a[0] = 1;
  i = 1;
  while (i < 10) { a[i] = 2*a[i-1]; i += 1;}
  b[4,6] = 32;
}
```

L'instruction IF

Une instruction **IF** a la forme suivante :

```
if (<expression>) <instruction>
```

ou

```
if (<expression>) <instruction> else <instruction>
```

L'instruction peut aussi se présenter sous la forme d'un bloc. L'expression sera évaluée. Si la valeur (arrondie) est <=0 (**false (FAUX)**) alors les instructions après le **else** seront exécutées, sinon (le résultat est donc **true (VRAI)**), les autres instructions seront exécutées. C'est une bonne habitude de toujours placer des accolades pour entourer les instructions. Il est donc préférable d'écrire les instructions comme ci-dessous.

```
if (<expression>)
{
  <instruction 1>
  <instruction 2>
  <instruction ...>
}
else
{
  <instruction 1>
  <instruction 2>
  <instruction ...>
}
```

Exemple

Le programme suivant déplace un objet en avant au milieu de l'écran.

```

{
  if (x<200) {x += 4} else {x -= 4};
}
```

L'instruction Repeat

Une instruction **Repeat** a la forme suivante :

```
repeat (<expression>) <instruction>
```

L'instruction est répétée un certain nombre de fois indiqué par la valeur (arrondie) de l'expression.

Exemple

Le programme suivant crée cinq balles (**ball** étant l'instance de l'objet) à des positions aléatoires.

```
{
  repeat (5) instance_create(random(400),random(400),ball);
}
```

L'instruction While

Une instruction **While** se présente sous la forme suivante :

```
while (<expression>) <instruction>
```

Tant que l'expression est **vraie**, l'instruction (qui peut être également un bloc) est exécutée. Soyez prudent avec vos boucles **While**. Il est facile de créer des boucles infinies, dans lesquelles votre jeu se perdra et ne réagira plus à aucune action de l'utilisateur.

Exemple

Le programme suivant essaie de déterminer une position libre (c'est à dire non utilisée par un autre objet) pour l'objet courant. C'est exactement la même chose que l'action qui déplace un objet à une position aléatoire.

```
{
  while (!place_free(x,y))
  {
    x = random(room_width);
    y = random(room_height);
  }
}
```

L'instruction Do

Une instruction **Do** a la forme suivante :

```
do <instruction> until(<expression>)
```

L'instruction (qui peut être un bloc) est exécutée jusqu'à ce que l'expression soit **vraie**. L'instruction est exécutée au moins une fois. Soyez prudent avec vos boucles **Do**. Vous pourrez facilement créer des boucles infinies, pour lesquelles votre jeu se perdra et ne réagira plus aux actions du joueur.

Exemple

Le programme suivant essaie de placer l'objet courant à un emplacement libre (c'est identique à l'action qui consiste à déplacer un objet à une position aléatoire).

```
{
  do
  {
    x = random(room_width);
    y = random(room_height);
  }
  until (place_free(x,y))
}
```

L'instruction For

Une instruction **For** se présente ainsi :

```
for (<instruction 1> ; <expression> ;<instruction 2>)
<instruction 3>
```

Cela fonctionne de la façon suivante. Premièrement, l'instruction 1 est exécutée. Puis l'expression est évaluée. Si celle-ci est **vraie**, l'instruction 3 sera exécutée. L'instruction 2 puis l'expression sont réévaluées de nouveau. Ce processus se poursuit jusqu'à ce que l'expression soit **fausse**.

Cela peut vous paraître compliqué mais vous devez interpréter ceci de la manière suivante. La première instruction initialise la boucle **For**. L'expression teste ensuite si la boucle doit se terminer. L'instruction 2 est l'instruction de saut qui poursuit à la prochaine évaluation de la boucle.

On utilise principalement la boucle **For** pour créer un compteur dans une certaine plage.

Exemple

Le programme suivant initialise un tableau de longueur 10 avec des valeurs allant de 1 à 10.

```
{
  for (i=0; i<=9; i+=1) list[i] = i+1;
}
```

L'instruction Switch

Dans un certain nombre de situations, vous souhaiterez effectuer une action en fonction d'une valeur particulière. Vous pouvez faire cela en utilisant quelques instructions **IF** mais il est plus facile d'utiliser l'instruction **Switch**. Une instruction **Switch** se présente sous la forme suivante :

```
switch (<expression>)
{
  case <expression1>: <instruction 1>; ... ; break;
  case <expression2>: <instruction 2>; ... ; break;
  ...
  default: <instruction>; ...
}
```

Voici comment cela fonctionne. En premier lieu, l'expression est exécutée. Ensuite, elle est comparée avec les résultats des différentes expressions situées après les instructions **Case**. L'exécution se poursuit après la première instruction **Case** en cas d'inégalité, jusqu'à ce que l'instruction **Break** soit rencontrée. Si aucune instruction **Case** ne possède la bonne valeur, l'exécution continue après l'instruction **default** (il n'est pas obligatoire d'avoir une instruction **default**). Veuillez noter que plusieurs instructions **Case** peuvent être placées dans le même bloc d'instructions. Aussi, l'instruction **Break** n'est pas nécessaire. S'il n'existe pas d'instruction **Break**, l'exécution se poursuivra simplement à la prochaine instruction **Case**.

Exemple

Le programme suivant effectue une action selon la touche pressée au clavier.

```
switch (keyboard_key)
{
  case vk_left:
  case vk_numpad4:
    x -= 4; break;
  case vk_right:
  case vk_numpad6:
    x += 4; break;
}
```

L'instruction Break

L'instruction **Break** se présente sous la forme suivante :

```
break
```

Lorsqu'elle est utilisée à l'intérieur des boucles suivantes **For**, **While** et **Repeat** ainsi que dans les instructions **Switch** ou **With**, l'instruction **Break** suspend la boucle ou l'instruction concernée. Si cette instruction est utilisée en dehors de ces dernières, **Break** arrête le programme (mais pas le jeu).

L'instruction Continue

L'instruction **Continue** a la forme suivante :

```
continue
```

Si elle est utilisée à l'intérieur des boucles **For**, **While**, **Repeat** ou encore une instruction **With**, cette instruction continue l'exécution avec la valeur suivante de la boucle ou de l'instruction **With**.

L'instruction Exit

L'instruction **Exit** se présente ainsi :

```
exit
```

Cette instruction termine l'exécution du script ou du code programme actuellement en cours d'exécution (cela n'arrête pas l'exécution du jeu ! A cet effet, il existe la fonction **game_end()**; voir plus loin).

Les Fonctions

Une fonction se présente sous la forme d'un nom de fonction, suivi de zéro ou plusieurs arguments encadrés de parenthèses et séparés par des virgules.

```
<function>(<arg1>,<arg2>,...)
```

Il existe deux types de fonctions. En premier lieu, nous trouvons un grand nombre de fonctions intégrées, destinées à contrôler tous les aspects de votre jeu. En deuxième lieu, tout script que vous aurez défini dans votre jeu, pourra être utilisé comme fonction.

Veuillez noter que pour une fonction sans argument, il est nécessaire d'utiliser des parenthèses. Certaines fonctions retournent des valeurs qui peuvent être employées dans des expressions. Les autres fonctions exécutent tout simplement des commandes.

Il est impossible d'utiliser une fonction à la gauche d'une affectation. Par exemple, vous ne pouvez pas écrire : `instance_nearest(x,y,obj).speed = 0`. Par contre, vous pourrez écrire ceci : `(instance_nearest(x,y,obj)).speed = 0`.

Les Scripts

Lorsque vous créerez un script, vous souhaiterez accéder aux arguments qui lui auront été transmis (soit en utilisant une action script, soit en appelant le script comme fonction à partir d'un programme donné (ou à partir d'un autre, voire même à partir du même script). Ces arguments sont stockés dans les variables `argument0`, `argument1`, ..., `argument15`. Un maximum de 16 arguments peut ainsi être utilisé (lors de l'appel d'un script à partir d'une action, seuls les 5 premiers arguments peuvent être mentionnés). Vous pouvez également utiliser `argument[0]` etc.

Les scripts peuvent aussi retourner une valeur. De cette manière, ils peuvent être utilisés dans des expressions. A cet effet, vous utiliserez l'instruction **return** :

```
return <expression>
```

L'exécution du script s'arrêtera à l'instruction **return** !

Exemple

Vous trouverez ci-dessous un petit script qui calcule le carré de l'argument transmis :

```
{
  return (argument0*argument0);
}
```

Pour appeler un script dans du code, il suffit de procéder de la même façon que pour l'appel de fonctions. C'est à dire, écrire le nom du script complété des valeurs d'arguments entre parenthèses.

Constructeur With

Comme indiqué précédemment, il est possible de lire et de changer la valeur des variables appartenant à d'autres instances. Mais dans de nombreux cas, vous souhaiterez également effectuer davantage de choses avec ces instances. Par exemple, imaginez que vous vouliez déplacer toutes les balles de 8 pixels vers le bas. Vous pourriez penser que ceci peut simplement être réalisé avec le programme suivant :

```
ball.y = ball.y + 8;
```

Mais ceci n'est pas correct. La partie droite de l'affectation reçoit la valeur d'ordonnée **y** de la première balle puis y ajoute 8. Ensuite, cette nouvelle valeur est assignée à l'ordonnée **y** de toutes les balles. Ce qui entraîne que toutes les balles possèdent désormais la même ordonnée **y**. L'instruction

```
ball.y += 8;
```

produira exactement le même résultat car elle ne constitue simplement qu'une abréviation de la première instruction. Comment donc résoudre ce problème ? En utilisant l'instruction **with** spécialement conçue pour cela. La forme globale de l'instruction est la suivante :

```
with (<expression>) <instruction>
```

<expression> indique une ou plusieurs instances. Vous pouvez utiliser l'**ID** d'une instance, le nom d'un objet (pour mentionner toutes les instances de cet objet) ou l'un des objets spéciaux (**all**, **self**, **other**, **noone**). <instruction> sera maintenant exécutée pour chacune des instances indiquées comme si cette instance était l'instance courante (**self**). Ainsi, pour déplacer toutes les balles de 8 pixels vers le bas, vous devrez par exemple taper ceci :

```
with (ball) y += 8;
```

Si vous voulez exécuter plusieurs instructions, entourez-les avec des accolades. Par exemple, pour déplacer toutes les balles vers une position aléatoire, vous pouvez écrire :

```
with (ball)
```

```
{
  x = random(room_width);
  y = random(room_height);
}
```

Veuillez noter que, pendant ces instructions, l'instance mentionnée devient l'instance **self**. De la même manière, l'instance originale **self** devient l'instance **other** (l'autre instance). Par exemple, pour déplacer toutes les balles vers une position de l'instance courante, tapez ceci :

```
with (ball)
{
  x = other.x;
  y = other.y;
}
```

L'utilisation de l'instruction **With** est extrêmement puissante. Vous trouverez ci-dessous quelques exemples. Pour détruire toutes les balles, vous taperez :

```
with (ball) instance_destroy();
```

Si une bombe explose et que vous souhaitez détruire toutes les instances touchées, vous pourrez utiliser :

```
with (all)
{
  if (distance_to_object(other) < 50) instance_destroy();
}
```

Commentaires

Vous pouvez ajouter des commentaires dans vos programmes. Tous les caractères de la ligne placés après **//** ne seront pas interprétés. Vous pourrez également créer un commentaire sur plusieurs lignes en plaçant le texte entre **/*** et ***/** (la colorisation du code peut dans ce cas ne pas bien fonctionner ! Aussi, pressez **F12** pour recolorer le texte du code si une erreur survient).

Fonctions et Variables en GML

Le GML comprend un grand nombre de fonctions et de variables internes. Avec elles, vous pourrez contrôler l'ensemble du jeu. A chaque action correspond une fonction équivalente, ainsi vous n'êtes pas obligé d'utiliser les actions si vous préférez employer du code. Mais il existe davantage de fonctions et de variables pour contrôler le fonctionnement du jeu qui ne peuvent pas quant à elles être utilisées avec les actions. Ainsi, si vous désirez développer des jeux plus évolués, il est fortement recommandé de lire les chapitres suivants afin d'avoir un aperçu de toutes les possibilités. Veuillez noter que ces variables et ces fonctions peuvent également être employées lorsque vous fournissez des valeurs aux actions. Même si vous n'avez prévu d'utiliser du code programme ou encore d'écrire des scripts, vous tirerez cependant tout bénéfice de ces informations.

La convention suivante sera utilisée ici. Les noms de variables signalés avec le caractère ***** sont en lecture seule (leur valeur ne peut être modifiée). Les noms de variables suivis de **[0..n]** sont des tableaux. La plage admise des indices sera donnée.

Calculs et traitements

Game Maker comporte un grand nombre de fonctions utilisées pour effectuer des calculs et des traitements divers. En voici la liste complète.

Des informations sur le langage GML peuvent être trouvées dans les pages suivantes :

Les Constantes

Les constantes mathématiques suivantes existent :

true est égal à 1.

false est égal à 0.

pi est égal à 3.1415...

Fonctions sur les réels

Les fonctions suivantes existent et concernent les nombres réels.

random(x) Retourne un nombre réel aléatoire entre 0 et x. Le nombre sera toujours plus petit que x.

choose(val1,val2,val3,...) Retourne l'un des arguments choisi de manière aléatoire. La fonction accepte jusqu'à 16 arguments.

abs(x) Retourne la valeur absolue de x.

sign(x) Retourne le signe de x (-1, 0 ou 1).

round(x) Retourne x arrondi à l'entier le plus proche.

floor(x) Retourne la valeur plancher de x, qui correspond à x arrondi à l'entier juste inférieur.

ceil(x) Retourne la valeur plafond de x, qui correspond à x arrondi à l'entier juste supérieur.

frac(x) Retourne la partie décimale de x, qui correspond à celle juste après le point décimal.

sqrt(x) Retourne la racine carrée de x. x doit être un nombre positif.

sqr(x) Retourne x*x.

power(x,n) Retourne x à la puissance n.

exp(x) Retourne e à la puissance x.

ln(x) Retourne le logarithme naturel de x.

log2(x) Retourne log base 2 de x.

log10(x) Retourne log base 10 de x.

logn(n,x) Retourne log base n de x.

sin(x) Retourne le sinus de x (x exprimé en radians).

cos(x) Retourne le cosinus de x (x exprimé en radians).

tan(x) Retourne la tangente de x (x exprimé en radians).

arcsin(x) Retourne le sinus inverse de x.

arccos(x) Retourne le cosinus inverse de x.

arctan(x) Retourne la tangente inverse de x.

arctan2(y,x) Calcule arctan(Y/X) et retourne un angle dans un quart de cercle.

degtorad(x) Convertit les degrés en radians.

radtodeg(x) Convertit les radians en degrés.

min(val1,val2,val3,...) Retourne le minimum des valeurs. La fonction accepte jusqu'à 16 arguments. Ceux-ci doivent tous être des réels ou des chaîne de caractères.

max(val1,val2,val3,...) Retourne le maximum des valeurs. La fonction accepte jusqu'à 16 arguments. Ceux-ci doivent tous être des réels ou des chaînes

de caractères.

mean(val1,val2,val3,...) Retourne la moyenne des valeurs. La fonction accepte jusqu'à 16 arguments. Ceux-ci doivent tous être des réels.

median(val1,val2,val3,...) Retourne la valeur médiane des valeurs, qui correspond au milieu de la valeur (lorsque le nombre d'arguments est pair, la valeur la plus petite entre les deux valeurs du milieu est retournée). La fonction accepte jusqu'à 16 arguments. Ceux-ci doivent tous être des réels.

point_distance(x1,y1,x2,y2) Retourne la distance entre le point (x1,y1) et le point (x2,y2).

point_direction(x1,y1,x2,y2) Retourne la direction du point (x1,y1) vers le point (x2,y2) en degrés.

lengthdir_x(len,dir) Retourne le composant x horizontal du vecteur, déterminé par la longueur indiquée et la direction.

lengthdir_y(len,dir) Retourne le composant y vertical du vecteur, déterminé par la longueur indiquée et la direction.

is_real(x) Retourne si x est un nombre réel (en opposition à une chaîne).

is_string(x) Retourne si x est une chaîne (en opposition à un réel).

Fonctions opérant sur des chaînes de caractères

Les fonctions suivantes traitent des caractères et des chaînes.

chr(val) Retourne une chaîne contenant le caractère de code ascii **val**.

ord(str) Retourne le code ascii du premier caractère de **str**.

real(str) Transforme **str** en un nombre réel. **str** peut contenir un signe moins, un point décimal et même une partie exponentielle.

string(val) Transforme la valeur réelle en une chaîne de caractères en utilisant un format standard (aucune position décimale lorsqu'il s'agit d'un entier et deux positions décimales dans les autres cas).

string_format(val,tot,dec) Transforme **val** en une chaîne en utilisant votre propre format: **tot** indique le nombre total de positions et **dec** le nombre de positions décimales.

string_length(str) Retourne le nombre de caractères de la chaîne.

string_pos(substr,str) Retourne la position de **substr** dans **str** (0=pas d'occurrence).

string_copy(str,index,count) Retourne une sous-chaîne de **str**, débutant à la position **index** et de longueur **count**.

string_char_at(str,index) Retourne le caractère de **str** à la position **index**.

string_delete(str,index,count) Retourne une copie de **str** amputée d'une partie commençant à la position **index** et de longueur **count**.

string_insert(substr,str,index) Retourne une copie de **str** avec ajout de **substr** à la position **index**.

string_replace(str,substr,newstr) Retourne une copie de **str** avec la première occurrence de **substr** remplacée par **newstr**.

string_replace_all(str,substr,newstr) Retourne une copie de **str** avec toutes les occurrences de **substr** remplacées par **newstr**.

string_count(substr,str) Retourne le nombre d'occurrences de **substr** dans **str**.

string_lower(str) Retourne une copie en minuscule de **str**.

string_upper(str) Retourne une copie en majuscule de **str**.

string_repeat(str,count) Retourne une chaîne comprenant **count** copies de **str**.

string_letters(str) Retourne une chaîne qui ne contient que les lettres de **str**.

string_digits(str) Retourne une chaîne qui ne contient que les chiffres de **str**.

string_lettersdigits(str) Retourne une chaîne qui ne contient que les lettres et les chiffres de **str**.

Les fonctions suivantes traitent du presse-papiers pour stocker du texte.

clipboard_has_text() Retourne si présence de texte dans le presse-papiers.

clipboard_get_text() Retourne le texte courant du presse-papiers.

clipboard_set_text(str) Mets la chaîne **str** dans le presse-papiers.

Fonctions sur les dates et le temps

Dans *Game Maker*, il existe une grande quantité de fonctions pour manipuler les dates et l'heure. Une combinaison date-heure est sauvegardée dans un nombre réel. La partie complète de la valeur date-heure est le nombre de jours écoulés depuis le 30-12-1899 (12/30/1899). La partie fractionnelle de la valeur date-heure est la fraction du jour en heures déjà écoulée (ex: 18). Les fonctions suivantes existent :

date_current_datetime() Retourne la valeur date-heure correspondant à l'instant présent.
date_current_date() Retourne la valeur date-heure correspondant à la date courante uniquement (on ignore l'heure).
date_current_time() Retourne la valeur date-heure correspondant à l'heure courante uniquement (on ignore la date).
date_create_datetime(year,month,day,hour,minute,second) Crée une valeur date-heure correspondant à la date et l'heure indiquées.
date_create_date(year,month,day) Crée une valeur date-heure correspondant à la date indiquée.
date_create_time(hour,minute,second) Crée une valeur date-heure correspondant à l'heure indiquée.
date_valid_datetime(year,month,day,hour,minute,second) Retourne si la date et l'heure indiquées sont valides.
date_valid_date(year,month,day) Retourne si la date indiquée est valide.
date_valid_time(hour,minute,second) Retourne si l'heure indiquée est valide.
date_inc_year(date,amount) Retourne une nouvelle date qui est le nombre d'années après la date indiquée. **amount** doit être un nombre entier.
date_inc_month(date,amount) Retourne une nouvelle date qui est le nombre de mois après la date indiquée. **amount** doit être un nombre entier.
date_inc_week(date,amount) Retourne une nouvelle date qui est le nombre de semaines après la date indiquée. **amount** doit être un nombre entier.
date_inc_day(date,amount) Retourne une nouvelle date qui est le nombre de jours après la date indiquée. **amount** doit être un nombre entier.
date_inc_hour(date,amount) Retourne une nouvelle date qui est le nombre d'heures après la date indiquée. **amount** doit être un nombre entier.
date_inc_minute(date,amount) Retourne une nouvelle date qui est le nombre de minutes après la date indiquée. **amount** doit être un nombre entier.

date_inc_second(date,amount) Retourne une nouvelle date qui est le nombre de secondes après la date indiquée. **amount** doit être un nombre entier.
date_get_year(date) Retourne l'année correspondant à la date.
date_get_month(date) Retourne le mois correspondant à la date.
date_get_week(date) Retourne la semaine de l'année correspondant à la date.
date_get_day(date) Retourne le jour du mois correspondant à la date.
date_get_hour(date) Retourne l'heure correspondant à la date.
date_get_minute(date) Retourne la minute correspondant à la date.
date_get_second(date) Retourne la seconde correspondant à la date.
date_get_weekday(date) Retourne le jour de la semaine correspondant à la date.
date_get_day_of_year(date) Retourne le jour de l'année correspondant à la date.
date_get_hour_of_year(date) Retourne l'heure de l'année correspondant à la date.
date_get_minute_of_year(date) Retourne la minute de l'année correspondant à la date.
date_get_second_of_year(date) Retourne la seconde de l'année correspondant à la date.
date_year_span(date1,date2) Retourne le nombre d'années entre deux dates. Les années incomplètes seront exprimées sous forme de fraction.
date_month_span(date1,date2) Retourne le nombre de mois entre deux dates. Les mois incomplets seront exprimés sous forme de fraction.
date_week_span(date1,date2) Retourne le nombre de semaines entre deux dates. Les semaines incomplètes seront exprimées sous forme de fraction.
date_day_span(date1,date2) Retourne le nombre de jours entre deux dates. Les jours incomplets seront exprimés sous forme de fraction.
date_hour_span(date1,date2) Retourne le nombre d'heures entre deux dates. Les heures incomplètes seront exprimées sous forme de fraction.
date_minute_span(date1,date2) Retourne le nombre de minutes entre deux dates. Les minutes incomplètes seront exprimées sous forme de fraction.
date_second_span(date1,date2) Retourne le nombre de secondes entre deux dates. Les secondes incomplètes seront exprimées sous forme de fraction.
date_compare_datetime(date1,date2) Compare les deux valeurs date-heure. Retourne -1, 0, ou 1 selon que la première valeur soit plus petite, égale ou plus grande que la seconde valeur.
date_compare_date(date1,date2) Compare les deux valeurs date-heure en ne prenant en compte que l'aspect date. Retourne -1, 0, ou 1 selon que la première

valeur soit plus petite, égale ou plus grande que la seconde valeur.

date_compare_time(date1,date2) Compare les deux valeurs date-heure en ne prenant en compte que l'aspect heure. Retourne -1, 0, ou 1 selon que la première valeur soit plus petite, égale ou plus grande que la seconde valeur.

date_date_of(date) Retourne la partie de la date de la valeur date-heure indiquée, en initialisant à 0 la partie heure.

date_time_of(date) Retourne la partie heure de la valeur date-heure indiquée, en initialisant à 0 la partie date.

date_datetime_string(date) Retourne une chaîne indiquant la date et l'heure fournies dans le format par défaut du système.

date_date_string(date) Retourne une chaîne indiquant la date fournie dans le format par défaut du système.

date_time_string(date) Retourne une chaîne indiquant l'heure fournie dans le format par défaut du système.

date_days_in_month(date) Retourne le nombre de jours du mois indiqué dans la valeur date-heure.

date_days_in_year(date) Retourne le nombre de jours de l'année indiquée dans la valeur date-heure.

date_leap_year(date) Retourne si l'année indiquée dans la valeur date-heure est une année bissextile.

date_is_today(date) Retourne si la valeur date-heure indiquée représente aujourd'hui.

Gameplay

Il existe un grand nombre de variables et de fonctions pouvant être utilisées pour définir le gameplay. Cela a une influence particulière sur les mouvements et la création des instances, le timing, la room et la gestion des événements.

Des informations sur le gameplay peuvent être trouvées dans les pages suivantes :

Les Déplacements

Un aspect important des jeux est certainement le déplacement des instances d'objets. Chaque instance possède deux variables internes **x** and **y** qui indiquent la position de l'instance (pour être précis, elles indiquent la position où l'origine du sprite est placée). Position (0,0) correspond au coin supérieur gauche de la room. Vous pouvez changer la position de l'instance en changeant ses variables **x** et **y**. Si vous souhaitez que l'objet effectue des mouvements sophistiqués, ce sera la bonne méthode à suivre. Vous placerez habituellement ce code dans l'événement **step** de l'objet.

Si l'objet doit se déplacer avec une vitesse et une direction constantes, il y a une façon plus simple de réaliser tout cela. Chaque instance d'objet possède une vitesse horizontale (**hspeed**) et verticale (**vspeed**). Ces deux paramètres sont exprimés en pixels par step. Une vitesse positive horizontale indique un mouvement vers la droite, une vitesse négative horizontale signifie un déplacement vers la gauche. Une vitesse positive verticale mentionne un mouvement vers le bas tandis qu'une vitesse négative verticale un déplacement vers le haut. Vous ne devrez initialiser ces variables qu'une seule fois (par exemple dans l'événement de création) afin de donner à l'instance d'objet un mouvement constant.

Il y a plusieurs façons d'indiquer un mouvement, en utilisant une direction (en degrés 0-359) et une vitesse (ne doit pas être négative). Vous pouvez fixer et lire ces variables pour spécifier un déplacement arbitraire (en interne, cela concerne les valeurs **hspeed** et **vspeed**.) Il existe également la friction, la gravité et la direction de la gravité. Enfin, il y a la fonction **motion_add(dir,speed)** qui ajoute un déplacement à l'actuel mouvement.

Pour être complet, chaque instance possède les variables et les fonctions suivantes ayant traits à leur position et leur déplacement :

x sa position en **x**.
y sa position en **y**.
xprevious sa position précédente en **x**.
yprevious sa position précédente en **y**.
xstart sa position de départ en **x** dans la room.
ystart sa position de départ en **y** dans la room.
hspeed composant horizontal de la vitesse.
vspeed composant vertical de la vitesse.
direction sa direction courante (0-360, sens contraire des aiguilles d'une montre, 0 = vers la droite).

speed sa vitesse actuelle (pixels par step).
friction friction actuelle (pixels par step).
gravity valeur actuelle de la gravité (pixels par step).
gravity_direction direction de la gravité (270 indique vers le bas).
motion_set(dir,speed) paramètre le déplacement avec la vitesse indiquée vers la direction **dir**.
motion_add(dir,speed) additionne le mouvement à l'actuel mouvement (addition de vecteurs).

Il existe une grande quantité de fonctions disponibles qui vous aideront à définir vos mouvements :

place_free(x,y) Retourne si l'instance placée à la position **(x,y)** est hors collision. On l'utilise habituellement comme vérification avant de déplacer l'instance à sa nouvelle position.
place_empty(x,y) Retourne si l'instance placée à la position **(x,y)** rencontre quelque chose. Cette fonction prend également en compte les instances non solides.
place_meeting(x,y,obj) Retourne si l'instance placée à la position **(x,y)** rencontre **obj**. **obj** peut être un objet auquel cas la fonction retourne **true** (VRAI) si une instance de cet objet est rencontrée. Cela peut aussi être un **ID** d'instance, le mot spécial **all** indiquant une instance d'un objet quelconque ou le mot spécial **other**.
place_snapped(hsnap,vsnap) Retourne si l'instance est alignée avec les valeurs de rupture.
move_random(hsnap,vsnap) Déplace l'instance à une position libre, à une position de rupture, comme l'action correspondante.
move_snap(hsnap,vsnap) Snaps the instance, like the corresponding action. (NDT : **quelle est la traduction de cette phrase ?**)
move_wrap(hor,vert,margin) Place l'instance de l'autre côté lorsque celle-ci a quitté la room. **hor** indique si l'on doit la placer horizontalement et **vert** verticalement. **margin** indique à quelle distance l'origine de l'instance doit être en dehors de la room avant que le placement n'intervienne. C'est donc une marge par rapport aux bords de la room. Vous utiliserez habituellement cette fonction dans l'événement Outside.
move_towards_point(x,y,sp) Déplace les instances avec la vitesse **sp** vers la position **(x,y)**.
move_bounce_solid(adv) Rebondit contre les instances solides, comme l'action correspondante. **adv** indique si l'on doit employer le rebond anticipé qui prend également en compte les murs inclinés.

move_bounce_all(adv) Rebondit contre toutes les instances, au lieu de ne prendre en compte que les instances solides.

move_contact_solid(dir,maxdist) Déplace les instances dans la direction donnée jusqu'à ce qu'elles entrent en contact avec un objet solide. Si aucune collision ne se produit à la position courante, l'instance sera placée juste avant qu'une collision ne survienne. S'il y a déjà une collision, l'instance ne sera pas déplacée. Vous pouvez indiquer la distance maximale pour le déplacement (utilisez un nombre négatif pour avoir une distance arbitraire).

move_contact_all(dir,maxdist) Mêmes effets que la fonction précédente mais cette fois, l'instance sera stoppée si elle entre en contact avec un objet quelconque et non pas uniquement avec un objet solide.

move_outside_solid(dir,maxdist) Déplace l'instance dans la direction jusqu'à ce qu'elle ne se trouve plus en contact avec un objet solide. S'il n'y pas de collision à la position courante, l'instance ne sera pas déplacée. Vous pouvez indiquer la distance maximale pour le déplacement (utilisez un nombre négatif pour avoir une distance arbitraire).

move_outside_all(dir,maxdist) Mêmes effets que la fonction précédente mais cette fois-ci, vous déplacerez l'instance jusqu'à ce qu'elle ne se trouve plus en contact avec un objet quelconque et non pas uniquement avec un objet solide.

distance_to_point(x,y) Retourne la distance de la boîte de rebond de l'instance courante par rapport à **(x,y)**.

distance_to_object(obj) Retourne la distance de l'instance par rapport à la plus proche instance de l'objet **obj**.

position_empty(x,y) Retourne s'il n'y a rien à la position **(x,y)**.

position_meeting(x,y,obj) Retourne si à la position **(x,y)**, il y a une instance **obj**. **obj** peut être un objet, un **ID** d'une instance ou les mots-clés **self**, **other** ou **all**.

Les chemins

Avec *Game Maker*, vous pouvez définir des chemins et obliger des instances à suivre ces derniers. Bien que vous puissiez utiliser des actions pour effectuer cela, il existe des fonctions et des variables qui vous apporteront davantage de flexibilité :

path_start(path,speed,endaction,absolute) Débute l'exécution du chemin pour l'instance courante. La valeur **path** correspond au nom du chemin à suivre par l'instance. La valeur **speed** est la vitesse à laquelle le chemin doit être parcouru. Une vitesse négative signifie que l'instance se déplace en sens inverse le long du chemin. La valeur **endaction** indique ce qui doit se passer lorsque la fin du chemin est atteinte. Les valeurs suivantes peuvent être utilisées :

0 : stoppe le chemin
1: continue à partir de la position de départ du chemin (si le chemin n'était pas totalement parcouru, il y aura saut à la position de départ)
2: continue à partir de la position courante
3: inverse le chemin, c'est à dire change le signe de la vitesse

L'argument **absolute** devra être soit **true** soit **false**. Lorsqu'il est à **true**, les coordonnées absolues du chemin sont utilisées. Quand il est à **false** le chemin est relatif à la position courante de l'instance. Pour être plus précis, si la vitesse est positive, le point de départ du chemin sera placé à la position courante et le chemin sera suivi à partir de cette position. Si la vitesse est négative, le point final du chemin sera placé à la position courante et le chemin sera suivi en sens inverse à partir de cette position.
path_end() Suspend le parcours du chemin pour l'instance courante.
path_index* Index du chemin actuel suivi par l'instance. Il ne vous est pas possible de le modifier directement mais est utilisé par la fonction ci-dessus.
path_position Position du chemin actuel. **0** est le début du chemin tandis que **1** correspond à la fin. La valeur doit être comprise entre 0 et 1.
path_positionprevious Position précédente du chemin actuel. On l'utilise par exemple dans les événements de collision pour paramétrer la position du chemin à sa position précédente.
path_speed Vitesse (en pixels par step) à laquelle le chemin doit être parcouru. Utilisez une vitesse négative pour inverser le sens du déplacement.

path_orientation Orientation (sens contraire des aiguilles d'une montre) vers laquelle le chemin est dirigé. **0** correspond à l'orientation normale du chemin.
path_scale Mise à l'échelle du chemin. Augmentez cette valeur pour créer un chemin plus long. **1** est la valeur par défaut.
path_endaction L'action à exécuter en fin de chemin. Vous pouvez utiliser les valeurs indiquées ci-dessus.

La Planification des Mouvements

La planification des mouvements vous aidera à déplacer certaines instances d'une position à une autre tout en évitant les collisions avec d'autres instances (par exemple des murs). La planification des mouvements est assez difficile à mettre en oeuvre. Il est impossible de donner des fonctions générales qui fonctionneront correctement dans toutes les situations. De plus, le calcul des déplacements à des positions exemptes de collisions est une opération qui consomme beaucoup de temps. Aussi, vous devrez être prudent sur la manière et le moment où vous l'appliquerez. Gardez bien ces remarques à l'esprit quand vous utiliserez les fonctions suivantes.

Différentes formes de planification de mouvements sont déjà fournies avec *Game Maker*. La forme la plus simple force une instance à se diriger vers une position cible particulière, en essayant d'aller en ligne droite si possible mais en prenant une direction différente si c'est nécessaire. Ces fonctions doivent être utilisées en général dans l'événement step d'une instance. Elles sont similaires à la planification de mouvements proposée par les actions :

`mp_linear_step(x,y,stepsize,checkall)` Cette fonction déplace d'un step l'instance directement vers la position (x,y) indiquée. La taille du step est indiquée grâce au paramètre **`stepsize`**. Si l'instance est déjà à cette position, elle ne sera pas déplacée. Si **`checkall`** est à **true**, l'instance s'arrêtera lorsqu'elle entrera en contact avec une instance d'un objet quelconque. Si ce paramètre est à **false**, l'instance s'arrêtera uniquement quand elle heurtera une instance solide. Veuillez noter que cette fonction ne tentera pas de faire des détours si l'instance rencontre un obstacle. Elle échouera simplement dans ce cas. La fonction retourne si **oui** ou **non** la position cible a été atteinte.

`mp_linear_step_object(x,y,stepsize,obj)` Mêmes effets que la fonction ci-dessus mais cette fois, seules les instances de **`obj`** seront considérées comme des obstacles. **`obj`** peut être un objet ou encore un **ID** d'une instance.

`mp_potential_step(x,y,stepsize,checkall)` De la même façon que les fonctions précédentes, cette fonction déplace d'un step l'instance en direction d'une position particulière. Mais dans ce cas précis, elle essayera d'éviter les obstacles. Quand l'instance se dirigera vers une instance solide (ou n'importe laquelle instance si le paramètre **checkall** vaut **true**), elle changera la direction du mouvement afin d'éviter l'instance et la contournera. Le fonctionnement n'est pas garantie mais dans la plupart des cas, cette fonction déplacera l'instance vers la cible. La fonction retourne si la cible a été atteinte ou pas.

mp_potential_step_object(x,y,stepsize,obj) Identique à la fonction ci-dessus mais cette fois, seules les instances **obj** seront considérées comme des obstacles. **obj** peut être soit un objet soit un **ID** d'une instance.

mp_potential_settings(maxrot,rotstep,ahead,onspot) La fonction précédente effectue son travail en utilisant un certain nombre de paramètres pouvant être modifiés en utilisant la fonction. Globalement, cette méthode fonctionne de la manière suivante. Elle tente en premier lieu de déplacer directement l'instance vers la cible. Elle regarde un certain nombre de steps en avant, ce nombre pouvant être réglé avec le paramètre **ahead** (3 par défaut). Réduire cette valeur signifie que l'instance commencera à changer de direction plus tardivement. L'augmenter indiquera que l'instance débutera son changement de direction plus tôt. Si ce contrôle mène à une collision, elle commencera à regarder dans les directions les plus à gauche et les plus à droite afin de déterminer la meilleure direction. Elle effectue cela en utilisant des steps de taille **rotstep** (10 par défaut). En réduisant ce paramètre, cela donnera à l'instance plus de mouvements possibles mais cela ralentira le processus. Le paramètre **maxrot** est un peu plus difficile à expliquer. L'instance possède une direction courante. **maxrot** (30 par défaut) indique combien de changements de direction l'instance est autorisée à changer sa direction actuelle dans un step. Aussi, même si elle peut se déplacer vers la cible par exemple en ligne droite, elle le fera uniquement si elle ne transgresse pas le nombre maximal de changements de direction. Si vous donnez à **maxrot** une valeur plus grande, l'instance pourra changer davantage de directions à chaque step. Cela rendra plus facile la recherche du chemin le plus court mais le chemin obtenu sera plus tortueux. Si vous diminuez cette valeur, le chemin sera plus coulé mais vous obtiendrez davantage de détours (et parfois, la recherche de la cible échouera). Si aucun step ne peut être effectué, le comportement dépendra de la valeur du paramètre **onspot**. Si la valeur de **onspot** est **true** (la valeur par défaut), l'instance effectuera une rotation sur son spot selon la valeur indiquée par **maxrot**. Si la valeur est **false**, l'instance ne bougera pas du tout. Paramétrer cette valeur à **false** sera utile par exemple pour des voitures mais réduira la probabilité de trouver un chemin.

Veuillez noter que l'approche potentielle utilise uniquement des information locales. Aussi, elle ne pourra trouver un chemin que si ces informations locales sont suffisantes pour déterminer la bonne direction de mouvement. Par exemple, la recherche du chemin hors d'un labyrinthe échouera la plupart du temps.

Le deuxième type de fonctions détermine pour l'instance un chemin exempt de collision. Une fois ce chemin déterminé, vous pourrez l'assigner à l'instance afin de la diriger vers la cible. Le calcul du chemin prend un certain temps mais ensuite, l'exécution du chemin sera rapide. Bien entendu, ce chemin sera valide si seulement la situation n'a pas changé entre temps. Par exemple, si des obstacles se sont déplacés, vous devrez certainement recalculer le chemin. Veuillez remarquer que ces fonctions peuvent également échouer. ***Ces fonctions sont uniquement disponibles dans la version enregistrée de Game Maker.***

Les deux premières fonctions utilisent le déplacement linéaire et une approche de champs potentiel utilisés également dans les fonctions step.

mp_linear_path(path,xg,yg,stepsize,checkall) Cette fonction calcule un chemin en ligne droite pour l'instance de sa position courante vers la position **(xg,yg)** en utilisant la taille step indiquée. Elle utilise les steps comme dans la fonction **mp_linear_step()**. Le chemin mentionné doit déjà exister et sera écrasé par le nouveau chemin (se reporter au chapitre ultérieur pour savoir comment créer et détruire les chemins). La fonction retournera si un chemin a pu être trouvé. La fonction s'arrêtera et signalera une erreur si aucun chemin direct n'existe entre le début et la cible. En cas d'échec, un chemin sera toujours créé qui s'exécutera à partir de la position où l'instance a été bloquée.
mp_linear_path_object(path,xg,yg,stepsize,obj) Identique à la fonction ci-dessus mais cette fois, seules les instances **obj** seront considérées comme des obstacles. **obj** peut être soit un objet soit un **ID** d'une instance.
mp_potential_path(path,xg,yg,stepsize,factor,checkall) Cette fonction calcule un chemin pour l'instance de sa position courante et en orientation vers la position **(xg,yg)** en utilisant la taille step indiquée, tout en tentant d'éviter les collisions avec les obstacles. Elle utilise des steps de champs potentiel, comme dans la fonction **mp_potential_step()** mais aussi des paramètres que l'on peut régler avec **mp_potential_settings()**. Le chemin indiqué doit déjà exister et sera écrasé par le nouveau chemin (se reporter au chapitre ultérieur pour savoir comment créer et détruire les chemins). La fonction retournera si un chemin a pu être trouvé. Afin d'éviter que la fonction continue son calcul indéfiniment, vous devrez fournir un facteur de temps supérieur à 1. La fonction s'arrêtera et signalera une erreur si elle n'a pu déterminer un chemin plus court durant ce laps de temps entre le début et la cible. Un facteur de 4 devrait normalement être suffisant mais si vous souhaitez faire de plus longs détours, il sera peut-être souhaitable de choisir une valeur encore plus grande. En cas d'échec, un chemin sera toujours créé en direction de la cible qui cependant ne sera jamais atteinte.

mp_potential_path_object(path,xg,yg,stepsize,factor,obj) Idem que la fonction précédente sauf que seules les instances **obj** seront considérées comme des obstacles. **obj** peut être soit un objet soit un **ID** d'une instance.

Les autres fonctions utilisent un mécanisme beaucoup plus complexe et sophistiqué via une approche basée sur les grilles (appelée parfois algorithme A*). Cette approche est plus efficace en terme de recherche de chemins (bien qu'elle puisse aussi échouer !) et détermine des chemins généralement plus courts mais nécessite davantage de travail de votre part. L'idée générale est la suivante. En premier lieu, nous plaçons une grille à l'endroit le plus approprié de la room. Vous pouvez choisir d'utiliser une grille avec un maillage fin (ce qui sera plus lent) ou une grille brute. Ensuite, pour tous les objets concernés, nous déterminons les cellules de la grille qui recouvrent les objets (en utilisant soit des boîtes de rebonds ou soit un contrôle précis) et marquons ces cellules comme étant interdites. Ainsi, une cellule sera marquée comme **totalement** interdite, même si elle recouvre partiellement un obstacle. Enfin, nous indiquerons une position de départ et une position cible (qui ne pourra être que parmi les cellules libres) puis la fonction déterminera par elle-même le chemin le plus court (le plus proche théoriquement du plus court) entre ces positions. Le chemin s'exécutera entre les centres des cellules libres. Ainsi, si les cellules sont suffisamment grandes de façon à ce que l'instance placée en son centre soit complètement à l'intérieur, le résultat sera concluant. Vous pourrez ensuite donner ce chemin à une instance pour qu'elle le suive.

L'approche basée sur les grilles est très puissante (elle est d'ailleurs utilisée dans de nombreux jeux professionnels) mais demande de votre part une plus grande réflexion. Vous devrez déterminer quelle est la zone et la taille de cellule les mieux adaptées pour résoudre le problème. Vous devrez aussi déterminer quels objets doivent être évités et si une vérification précise est nécessaire pour effectuer ce travail. Tous ces paramètres auront une grande influence sur l'efficacité de cette approche.

En particulier, la taille des cellules est **cruciale**. Souvenez-vous que les cellules doivent être assez grandes de manière à ce que l'objet en déplacement et placé en son origine au centre d'une cellule, demeure complètement à l'intérieur de la cellule (soyez prudent en ce qui concerne la position de l'origine de l'objet. Pensez à décaler le chemin si l'origine de l'objet n'est pas bien au centre !) D'un autre côté, plus les cellules seront petites et plus il existera de chemins possibles. Si vous créez des cellules trop grandes, les ouvertures entre les obstacles pourront être fermées car toutes les cellules rentreront en intersection avec un obstacle.

Les fonctions actuelles concernant l'approche basée sur les grilles, sont les suivantes :

mp_grid_create(left,top,hcells,vcells,cellwidth,cellheight)
Cette fonction crée la grille. Elle retourne un index qui devra être utilisé dans tous les autres appels. Vous pouvez créer et gérer plusieurs structures de grilles en même temps. **left** (gauche) et **top** (haut) indiquent la position du coin supérieur gauche de la grille. **hcells** et **vcells** indiquent le nombre de cellules horizontales

et verticales. Enfin, **cellwidth** et **cellheight** indiquent la taille des cellules.
mp_grid_destroy(id) Détruit la structure de la grille indiquée et libère la mémoire occupée. N'oubliez pas d'appeler cette fonction si vous n'avez plus besoin de la structure.
mp_grid_clear_all(id) Marque toutes les cellules de la grille comme libres.
mp_grid_clear_cell(id,h,v) Efface la cellule indiquée. La cellule 0,0 correspond à la cellule en haut à gauche.
mp_grid_clear_rectangle(id,left,top,right,bottom) Efface toutes les cellules qui sont en intersection avec le rectangle indiqué (dans les coordonnées de la room).
mp_grid_add_cell(id,h,v) Marque les cellules indiquées comme étant interdites. La cellule 0,0 correspond à la cellule en haut à gauche.
mp_grid_add_rectangle(id,left,top,right,bottom) Marque toutes les cellules qui sont en intersection avec le rectangle indiqué comme étant interdites.
mp_grid_add_instances(id,obj,prec) Marque toutes les cellules qui sont en intersection avec une instance d'un objet donné comme étant interdites. Vous pouvez également utiliser une instance individuelle en donnant à **obj** l'**ID** de l'instance. Vous pouvez aussi employer le mot-clé **all** pour indiquer toutes les instances de tous les objets. **prec** indique si l'on doit utiliser une vérification précise dans les collisions (ne fonctionnera que si le paramètre **contrôle précis** a été coché pour le sprite utilisé par l'instance).
mp_grid_path(id,path,xstart,ystart,xgoal,ygoal,allowdiag)
Détermine un chemin à l'aide de la grille. **path** doit mentionner un chemin existant qui sera remplacé par le chemin calculé par l'ordinateur. **xstart** et **ystart** indiquent le début du chemin et **xgoal** et **ygoal** la cible. **allowdiag** indique si les mouvements en diagonal sont autorisés au lieu de ne considérer que ceux horizontaux ou verticaux. La fonction retourne si elle a réussi à déterminer un chemin (veuillez noter que le chemin est indépendant de l'instance courante; c'est un chemin calculé dans la grille et non pas un chemin pour une instance particulière).

mp_grid_draw(id) Cette fonction dessine la grille avec les cellules libres de couleur verte et les cellules interdites en rouge. Cette fonction est assez lente et n'est généralement utilisée que dans les outils de déboguage.

La Détection des Collisions

Lors de la planification des mouvements ou lorsque vous décidez d'effectuer certaines actions, il est souvent important de vérifier s'il existe des collisions entre des objets placés à différents endroits. Les routines suivantes peuvent vous être utiles pour effectuer ce contrôle. Ces routines possèdent toutes en commun trois arguments. L'argument **obj** peut être un objet, le mot-clé **all** ou encore l'**ID** d'une instance. L'argument **prec** indique si la vérification doit s'effectuer de manière précise ou seulement être basée sur la boîte de rebond de l'instance. Un contrôle précis n'est fait uniquement que lorsque le sprite de l'instance est paramétré avec l'option de contrôle précis de collision. L'argument **notme** peut être positionné à **true** pour indiquer que l'instance courante ne doit pas être vérifiée. Toutes ces fonctions retournent soit l'**ID** de l'une des instances en collision ou renvoient une valeur négative en cas de non-collision.

collision_point(x,y,obj,prec,notme) Cette fonction teste si une collision se produit au point **(x,y)** avec les entités de l'objet **obj**.

collision_rectangle(x1,y1,x2,y2,obj,prec,notme) Cette fonction teste s'il existe une collision entre le rectangle (plein) de coins opposés indiqués et les entités de l'objet **obj**. Par exemple, vous pouvez utiliser cette fonction pour tester si une zone est sans obstacles.

collision_circle(xc,yc,radius,obj,prec,notme) Cette fonction teste s'il y a une collision entre le cercle (plein) centré à la position (xc,yc) d'un rayon donné et les entités de l'objet **obj**. Par exemple, vous pouvez l'utiliser pour tester si un objet est proche d'un emplacement particulier.

collision_ellipse(x1,y1,x2,y2,obj,prec,notme) Cette fonction teste s'il y collision entre l'ellipse (pleine) de coins opposés indiqués et les entités de l'objet **obj**.

collision_line(x1,y1,x2,y2,obj,prec,notme) Cette fonction teste s'il se produit une collision entre la ligne droite positionnée de (x1,y1) à (x2,y2) et les entités de l'objet **obj**. Ceci est une fonction très puissante. Vous pouvez ainsi l'utiliser pour tester si une instance peut en apercevoir une autre en vérifiant si la ligne droite entre les deux instances ne rentre pas par exemple en intersection avec un mur.

Les Instances

Dans un jeu, les unités de base sont les instances des différents objets. Pendant le jeu, il vous est possible de modifier certains aspects de ces instances. Ainsi, vous pouvez créer de nouvelles instances ou en détruire d'autres. En plus des variables propres aux mouvements, abordées plus en avant et de celles relatives à l'affichage décrites plus loin, chaque instance possède les variables suivantes :

object_index* L'index de l'objet d'une instance. Cette variable ne peut être modifiée.
id* L'unique identificateur de l'instance (>= 100 000) (veuillez noter que lors de la définition de rooms, l'ID de l'instance est toujours signalé en plaçant la souris au-dessus de cette instance.)
mask_index L'index du sprite utilisé comme masque de collisions. Donnez à cet index la valeur -1 pour lui affecter la même valeur que sprite_index.
solid Indique si l'instance est de type solide. Ceci peut être modifié durant le jeu.
persistent Indique si l'instance est persistante et doit réapparaître lorsque vous changez de room. Vous souhaiterez souvent mettre la persistance à **off** à certains moments (par exemple si vous retournez à la première room.)

Il subsiste un problème avec les instances. Il n'est pas très facile d'identifier les instances de manière individuelle. Elles ne possèdent pas de nom. En présence d'une unique instance d'un objet particulier, il vous est possible d'utiliser le nom de l'objet mais dans les autres cas, il est nécessaire d'employer l'**ID** de l'instance. C'est l'unique identifiant de l'instance. Vous pouvez l'utiliser dans les instructions **with** en tant qu'identificateur d'objet. Heureusement, il existe un certain nombre de variables et de routines qui vous aideront à localiser les ID des instances.

instance_count* Nombre d'instances qui existent actuellement dans la room.
instance_id[0..n-1]* L'ID d'une instance particulière. **n** représente ici le numéro de l'instance.

Veuillez prendre note que l'affectation des ID aux instances varie à chaque step. Aussi, vous ne pouvez pas utiliser les valeurs issues des steps précédents. Voici un exemple afin de mieux comprendre. Supposons que chaque unité de votre jeu possède une puissance particulière et que vous souhaitiez localiser la plus résistante, vous pourriez écrire alors le code suivant :

```
{
  maxid = -1;
  maxpower = 0;
  for (i=0; i<instance_count; i+=1)
  {
    iii = instance_id[i];
    if (iii.object_index == unit)
    {
      if (iii.power > maxpower)
        {maxid = iii; maxpower = iii.power;}
    }
  }
}
```

Après exécution de la boucle, la variable **maxid** contiendra l'**ID** de l'unité la plus puissante (ne supprimez jamais les instances lors de l'exécution de boucles car elles seraient automatiquement supprimées du tableau et vous obtiendriez comme résultat des instances manquantes).

instance_find(obj,n) Retourne l'ID de l'instance (n+1) de type **obj**. **obj** peut être un objet ou le mot-clé **all**. S'il n'existe pas, l'objet spécial **noone** sera retourné. Notez que l'affectation des ID aux instances varie à chaque step. Aussi, vous ne pouvez pas utiliser les valeurs issues des steps précédents.
instance_exists(obj) Retourne s'il existe une instance de type **obj**. **obj** peut être un objet, un ID d'instance ou le mot-clé **all**.
instance_number(obj) Retourne le nombre d'instances de type **obj**. **obj** peut être un objet ou le mot-clé **all**.
instance_position(x,y,obj) Retourne l'ID de l'instance de type **obj** à la position **(x,y)**. Lorsque plusieurs instances sont à la même position, seule la première est retournée. **obj** peut être un objet ou le mot-clé **all**. Si celui n'existe pas, l'objet spécial **noone** sera retourné.
instance_nearest(x,y,obj) Retourne l'ID de l'instance de type **obj** la plus proche de **(x,y)**. **obj** peut être un objet ou le mot-clé **all**.
instance_furthest(x,y,obj) Retourne l'ID de l'instance de type **obj** la plus éloignée de **(x,y)**. **obj** peut être un objet ou le mot-clé **all**.
instance_place(x,y,obj) Retourne l'ID de l'instance de type **obj** rencontré

quand l'instance courante est placée à la position **(x,y)**. **obj** peut être un objet ou le mot-clé **all**. S'il n'existe pas, l'objet spécial **noone** sera retourné.

Les fonctions suivantes peuvent être utilisées pour la création et la destruction d'instances.

`instance_create(x,y,obj)` Crée une instance de **obj** à la position **(x,y)**. La fonction retourne l'**ID** de la nouvelle instance.
`instance_copy(performevent)` Crée une copie de l'instance courante. L'argument indique si l'événement de création doit être exécuté pour la copie. La fonction retourne l'**ID** de la nouvelle copie.
`instance_destroy()` Détruit l'instance courante.
`instance_change(obj,perf)` Change l'instance en **obj**. **perf** indique si l'on doit effectuer les événements de destruction et de création.
`position_destroy(x,y)` Détruit toutes les instances pour lesquelles le sprite se situe à la position **(x,y)**.
`position_change(x,y,obj,perf)` Change toutes les instances à la position **(x,y)** en **obj**. **perf** signifie que l'on doit effectuer les événements de destruction et de création.

La Désactivation des Instances

Lors de la création de rooms très vastes comme par exemple dans les jeux de plateformes et dans le cas de l'utilisation d'une vue réduite, de nombreuses instances se situent en dehors de la vue. De telles instances restent cependant toujours actives et exécutent leurs événements. Aussi, ces instances sont toujours prises en compte lors des contrôles de collision. Ceci consomme beaucoup de temps ce qui n'est pas souvent nécessaire (par exemple, il n'est pas très important de gérer les instances qui se déplacent en dehors de la vue). Pour résoudre ce problème, *Game Maker* fournit quelques fonctions permettant de désactiver et de réactiver des instances. Avant de pouvoir les utiliser, il est néanmoins nécessaire que vous puissiez bien comprendre leur fonctionnement.

Lorsque vous désactivez des instances, celles-ci sont en quelque sorte retirées du jeu. Elles ne sont donc plus visibles nulle part et leurs événements ne sont plus exécutés. Ainsi, pour toutes les actions et fonctions, elles n'existent plus. Cela économise beaucoup de temps mais vous devrez cependant être prudent. Par exemple, si vous effacez toutes les instances d'un type particulier, les instances désactivées ne seront pas effacées (car elles n'existent plus). Aussi, ne pensez pas qu'une clé en possession d'un joueur peut ouvrir une porte désactivée si cette instance n'existe plus.

L'erreur la plus importante que vous pouvez commettre est de désactiver l'instance qui est responsable elle-même de l'activation. Pour éviter cette situation, certaines routines ci-dessous vous autorisent à préciser quelles doivent être les instances appelantes ne pouvant se désactiver par elles-mêmes.

Voici les routines disponibles :

`instance_deactivate_all(notme)` Désactive toutes les instances de la room. Si **notme** est positionné à **true**, l'instance appelante n'est pas désactivée (ce qui sera habituellement le résultat souhaité).
`instance_deactivate_object(obj)` Désactive toutes les instances de la room d'un objet donné. Vous pouvez également utiliser le mot **all** pour indiquer que toutes les instances doivent être désactivées ou l'**ID** d'une instance pour désactiver une instance particulière.
`instance_deactivate_region(left,top,width,height,inside,notme)` Désactive toutes les instances de la région indiquée (c'est à dire celles dont la boîte de rebond est en partie à l'intérieur de la région). Si **inside** vaut **false** , les instances totalement en dehors de la région seront désactivées. Si **notme** est

à **true**, l'instance appelante ne sera pas désactivée (c'est ce que généralement vous désirerez).

instance_activate_all() Active toutes les instances de la room.

instance_activate_object(obj) Active toutes les instances de la room d'un objet donné. Vous pouvez employer le mot **all** pour indiquer que toutes les instances doivent être activées ou encore l'**ID** d'une instance pour activer cette instance individuellement.

instance_activate_region(left,top,width,height,inside) Active toutes les instances de la région indiquée. Si **inside** vaut **false**, les instance totalement en dehors de la région seront activées.

Par exemple, pour désactiver toutes les instances hors de la vue et activer celles à l'intérieur de cette vue, vous pourriez employer le code suivant dans l'événement **step** du personnage en déplacement :

```
{
  instance_activate_all();
  instance_deactivate_region(view_xview[0],view_yview[0],

view_wview[0],view_hview[0],false,true);
}
```

En pratique, il est préférable d'utiliser une région légèrement plus grande que la vue.

Le Timing

Les jeux de qualité exigent une gestion précise des événements pouvant survenir lors de leur déroulement. Fort heureusement, *Game Maker* effectue pour vous la quasi-totalité de cette tâche. *Game Maker* garantie que les choses surviendront à intervalles réguliers (les steps). Ces intervalles de temps sont définis lors de la conception des rooms. Mais vous pouvez les modifier en utilisant la variable globale **room_speed**. Ainsi, par exemple, vous pourrez progressivement augmenter la vitesse de jeu, rendant ce dernier plus difficile, en ajoutant un petit incrément (comme 0.001) à la variable **room_speed** et ce à chaque **step**. Si votre machine est trop lente, la vitesse de jeu pourra ne jamais être respectée correctement. On peut le vérifier en utilisant la variable **fps** qui en permanence mesure le nombre actuel d'images (frames) par seconde. Enfin, pour une gestion du temps encore plus avancée, vous pouvez utiliser la variable **current_time** qui vous donne le nombre de millisecondes écoulées depuis le lancement de votre ordinateur. Vous trouverez ci-dessous la liste complète des variables disponibles (seule la première pourra être modifiée) :

room_speed Vitesse de jeu de la room courante (en steps par seconde).
fps* Nombre d'images (frames) actuellement affichées par seconde.
current_time* Nombre de millisecondes écoulées depuis le démarrage du système.
current_year* L'année en cours.
current_month* Le mois en cours.
current_day* Le jour actuel.
current_weekday* L'actuel jour de la semaine (1=dimanche, ..., 7=samedi).
current_hour* L'heure courante.
current_minute* La minute courante.
current_second* La seconde courante.

Parfois, vous souhaiterez suspendre le jeu pendant un court instant. A cet effet, vous utiliserez la fonction **sleep**.

sleep(numb) Suspend le jeu pendant **numb** millisecondes.

Comme vous le savez déjà, chaque instance possède 12 alarmes d'horloge différentes que vous pouvez paramétrer. Pour modifier (ou lire) les valeurs des différentes alarmes, utilisez la variable suivante :

alarm[0..11] Valeur de l'alarme d'horloge indiquée (noter que les alarmes d'horloges ne sont mises à jour que lorsque l'événement d'alarme de l'objet contient des actions !)

Nous avons déjà vu que pour assurer une gestion complexe du timing, nous pouvions utiliser la ressource appelée ligne de temps. On peut associer à chaque instance une ressource ligne de temps. Les variables suivantes concernent cet aspect :

timeline_index Index de la ligne de temps associée à l'instance. Vous pouvez modifier cet index pour utiliser une position particulière de la ligne de temps. Réglez cet index à -1 pour arrêter l'utilisation de la ligne de temps affectée à l'instance.
timeline_position Actuelle position de la ligne de temps. Vous pouvez modifier la position afin de sauter ou répéter certaines parties de la ligne de temps.
timeline_speed Habituellement, la position de la ligne de temps est incrémentée de 1à chaque step. Vous pouvez changer cette valeur en affectant un autre montant à cette variable. Vous pouvez utiliser des nombres réels comme 0.5. Si la valeur est plus grande que un, plusieurs moments pourront survenir pendant le même step. Ils seront tous traités dans le bon ordre et aucune action ne sera sautée.

Les Rooms

Les jeux fonctionnent et se déroulent dans des rooms (salles ou pièces). Chaque room possède un index représenté par le nom de la room. La room courante est stockée dans la variable **room**. Vous ne pourrez jamais être certain que les rooms sont numérotées dans un ordre consécutif. Aussi, ne jamais ajouter ou soustraire un nombre de la variable **room**. Utilisez pour cela les fonctions et les variables mentionnées plus bas. Veuillez trouver ci-dessous un modèle de programme illustrant ces propos :

```
{
  if (room != room_last)
  {
    room_goto_next();
  }
  else
  {
    game_end();
  }
}
```

Les variables et les fonctions suivantes concernent les rooms.

room Index de la room courante. Peut être modifié afin de se déplacer d'une room à une autre mais il est recommandé d'utiliser les routines suivantes.
room_first* Index de la première room du jeu.
room_last* Index de la dernière room du jeu.
room_goto(numb) Se rendre à la room d'index **numb**.
room_goto_previous() Aller à la room précédente.
room_goto_next() Aller à la room suivante.
room_restart() Relancer la room actuelle.
room_previous(numb) Retourne l'index de la room juste avant **numb** (-1 = aucune room) mais ne s'y rend pas.
room_next(numb) Retourne l'index de la room juste après **numb** (-1 = aucune room).
game_end() Termine le jeu.

game_restart() Relance le jeu à son point de départ.

Lors de l'appel de l'une des fonctions ci-dessus afin de changer de room ou encore de finir ou de relancer le jeu, veuillez bien noter que le changement n'intervient pas précisément lors de l'appel de la fonction mais uniquement après que l'action courante soit totalement exécutée. Les instructions de fin de script pourront donc être encore en cours d'exécution. La même règle s'appliquera également aux autres scripts en cours d'appel.

Les rooms présentent les propriétés supplémentaires suivantes :

room_width* Largeur de la room en pixels.
room_height* Hauteur de la room en pixels.
room_caption Chaîne contenant le titre de la room. Cette chaîne est affichée dans la zone de titre de la fenêtre de la room.
room_persistent Indique si la room actuelle est de type persistante.

La plupart des jeux offrent au joueur la possibilité de sauvegarder le jeu et de le recharger ultérieurement. Dans *Game Maker*, cela se produit automatiquement quand le joueur appuie sur <**F5**> pour sauver et <**F6**> pour charger le jeu. Vous pouvez également sauvegarder et recharger un jeu à partir d'un programme (**N.B** : le chargement ne prendra effet qu'à la fin du step courant).

game_save(string) Sauvegarde le jeu dans un fichier avec le nom **string**.
game_load(string) Charge un jeu à partir du fichier de nom **string**.

Seules les données de base du jeu seront sauvegardées. Si par exemple vous jouez un morceau de musique, la position précise de lecture du morceau musical ne sera pas sauvegardée. Les ressources modifiées ne seront pas sauvées également. De la même façon, il existe certaines autres choses qui ne sont pas sauvegardées comme par exemple le contenu des structures de donnée, les particules et les réglages du mode multi-joueurs.

Le Score

D'autres aspects importants que l'on retrouve dans la plupart des jeux sont le score, la santé et le nombre de vies. *Game Maker* garde trace du `score` dans la variable globale **score** et du nombre de `vies` dans la variable globale **lives** (vies). Vous pouvez modifier simplement le score en changeant la valeur de la variable concernée. Le même raisonnement s'applique en ce qui concerne la santé (health) et les vies (lives). Si le nombre de vies (au départ supérieur à 0) devient inférieur ou égal à 0, l'événement **no-more-lives** (plus de vies) est exécuté pour toutes les instances. Si vous ne souhaitez pas afficher le score et les vies dans le titre de la fenêtre, réglez la variable **show_score**, **show_lives** à la valeur **false**. De même, vous pouvez changer le titre de la fenêtre. Pour les jeux plus complexes, il sera préférable d'afficher le score par vous-même en utilisant par exemple vos propres routines.

score Le score actuel.
lives Le nombre de vies.
health La santé actuelle (0 à 100).
show_score Indique si l'on doit afficher le score dans le titre de la fenêtre.
show_lives Indique si l'on doit afficher le nombre de vies restantes dans le titre de la fenêtre.
show_health Indique si l'on doit afficher la santé dans le titre de la fenêtre.
caption_score Le titre à utiliser pour le score.
caption_lives Le titre à utiliser pour le nombre de vies.
caption_health Le titre à utiliser pour la santé.

La Génération des Evénements

Comme vous le savez, *Game Maker* est totalement orienté événement. Toutes les actions surviennent en tant que résultats d'événements. Il existe un certain nombre d'événements différents. Les événements de création et de destruction apparaissent respectivement lorsqu'une instance est créée ou détruite. A chaque step, le système vérifie en premier lieu les événements des alarmes. Ensuite, il examine les événements clavier et souris puis l'événement **step**. Puis, les instances sont placées à leurs nouvelles positions après quoi l'événement de collision est inspecté. Enfin, l'événement d'affichage est utilisé pour dessiner les instances (veuillez noter qu'en présence de vues multiples, l'événement d'affichage est appelé plusieurs fois à chaque step). Vous pouvez aussi appliquer un événement à l'instance courante à partir d'un programme. Les fonctions suivantes existent :

event_perform(type,numb) Exécute l'événement **numb** du type indiqué sur l'instance courante. Les types d'événements suivants peuvent être mentionnés :

ev_create
ev_destroy
ev_step
ev_alarm
ev_keyboard
ev_mouse
ev_collision
ev_other
ev_draw
ev_keypress
ev_keyrelease

En présence de plusieurs événements d'un type donné, **numb** peut être utilisé pour spécifier un événement précis. Pour l'événement alarme, **numb** peut varier de 0 à 11. En ce qui concerne l'événement clavier, vous devrez utiliser le code renvoyé par la touche pressée. Pour les événements souris, vous utiliserez les constantes suivantes :

ev_left_button
ev_right_button
ev_middle_button

ev_no_button

ev_left_press

ev_right_press

ev_middle_press

ev_left_release

ev_right_release

ev_middle_release

ev_mouse_enter

ev_mouse_leave

ev_mouse_wheel_up

ev_mouse_wheel_down

ev_global_left_button

ev_global_right_button

ev_global_middle_button

ev_global_left_press

ev_global_right_press

ev_global_middle_press

ev_global_left_release

ev_global_right_release

ev_global_middle_release

ev_joystick1_left

ev_joystick1_right

ev_joystick1_up

ev_joystick1_down

ev_joystick1_button1

ev_joystick1_button2

ev_joystick1_button3

ev_joystick1_button4

ev_joystick1_button5

ev_joystick1_button6

ev_joystick1_button7

ev_joystick1_button8

ev_joystick2_left

ev_joystick2_right

ev_joystick2_up

ev_joystick2_down

ev_joystick2_button1

ev_joystick2_button2

ev_joystick2_button3
ev_joystick2_button4
ev_joystick2_button5
ev_joystick2_button6
ev_joystick2_button7
ev_joystick2_button8

Pour l'événement de collision, vous donnerez l'index de l'objet **other**. Enfin, pour l'événement **other**, vous pourrez utiliser les constantes suivantes :

ev_outside
ev_boundary
ev_game_start
ev_game_end
ev_room_start
ev_room_end
ev_no_more_lives
ev_no_more_health
ev_animation_end
ev_end_of_path
ev_user0
ev_user1
ev_user2
ev_user3
ev_user4
ev_user5
ev_user6
ev_user7
ev_user8
ev_user9
ev_user10
ev_user11
ev_user12
ev_user13
ev_user14
ev_user15

Pour l'événement step, vous donnerez l'index qui peut utiliser les constantes suivantes :

ev_step_normal
ev_step_begin
ev_step_end

event_perform_object(obj,type,numb) Ces fonctions fonctionnent de la même manière que la fonction ci-dessus hormis que cette fois, vous pourrez indiquer des événements d'un autre objet. Veuillez noter que les actions de ces événements s'appliquent à l'instance courante et non pas aux instances de l'objet indiqué..
event_user(numb) Dans les événements **other**, il vous est possible de définir 16 événements utilisateur. Ils seront exécutés uniquement si vous appelez cette fonction. **numb** représente une valeur allant de 0 à 15.
event_inherited() Exécute un événement issu de l'héritage. Cela ne fonctionne que si l'instance possède un objet parent.

Vous pouvez obtenir des informations sur l'événement courant en cours d'exécution en utilisant les variables en lecture seule suivantes :

event_type* Type de l'événement courant en cours d'exécution.
event_number* Numéro de l'événement courant en cours d'exécution.
event_object* L'index de l'objet pour lequel l'événement courant est en cours d'exécution.
event_action* L'index de l'action en cours d'exécution (0 correspond au premier événement, etc.).

Variables et Fonctions Diverses

Vous trouverez ici les variables et fonctions ayant traits aux erreurs.

error_occurred Indique qu'une erreur est apparue
error_last Chaîne de caractères contenant le message de la dernière erreur survenue
show_debug_message(str) Affiche la chaîne en mode déboguage (debug mode)

Les fonctions suivantes existent et vous permettent de vérifier si certaines variables existent; variables dont vous pourrez initialiser et lire le contenu. Dans toutes ces fonctions, le nom de la variable est passé en paramètre à l'aide d'une chaîne de caractères !

variable_global_exists(name) Retourne si une variable globale de nom **name** (une chaîne) existe.
variable_local_exists(name) Retourne si une variable locale de nom **name** (une chaîne) existe pour l'instance courante.
variable_global_get(name) Retourne la valeur de la variable globale de nom **name** (une chaîne).
variable_global_array_get(name,ind) Retourne la valeur de l'index **ind** de la variable globale tableau de nom **name** (une chaîne).
variable_global_array2_get(name,ind1,ind2) Retourne la valeur des index **ind1,ind2** de la variable globale du tableau à 2 dimensions de nom **name** (une chaîne).
variable_local_get(name) Retourne la valeur de la variable locale de nom **name** (une chaîne).
variable_local_array_get(name,ind) Retourne la valeur de l'index **ind** de la variable locale du tableau de nom **name** (une chaîne).
variable_local_array2_get(name,ind1,ind2) Retourne la valeur des index **ind1,ind2** de la variable locale du tableau à 2 dimensions de nom **name** (une chaîne).
variable_global_set(name,value) Affecte la valeur **value** à la variable globale de nom **name** (une chaîne).
variable_global_array_set(name,ind,value) Affecte la valeur **value** à l'index **ind** de la variable globale tableau de nom **name** (une chaîne).
variable_global_array2_set(name,ind1,ind2,value) Affecte la valeur **value** aux index **ind1,ind2** de la variable globale tableau à 2 dimensions de

nom **name** (une chaîne).
variable_local_set(name,value) Affecte la valeur **value** à la variable locale de nom **name** (une chaîne).
variable_local_array_set(name,ind,value) Affecte la valeur **value** à l'index **ind** de la variable locale du tableau de nom **name** (une chaîne).
variable_local_array2_set(name,ind1,ind2,value) Affecte la valeur **value** aux index **ind1,ind2** de la variable locale du tableau à 2 dimensions de nom **name** (une chaîne).

Par exemple, vous pourrez écrire :

```
{
  if variable_global_exists('munition')
    global.munition += 1
  else
    global.munition = 0
}
```

Vous pouvez aussi utiliser ces fonctions pour passer des variables à un script **par référence**, en donnant le nom des variables sous forme de chaînes et en utilisant les fonctions pour modifier le contenu de ces variables.

Vous pouvez modifier la priorité d'exécution du programme à l'aide des fonctions suivantes :

set_program_priority(priority) Fixe la priorité du programme. Vous pouvez indiquer une valeur allant de -3 à +3. Une valeur de -3 signifie que le programme s'exécutera si aucun autre processus sur l'ordinateur ne demande du temps processeur, soit autrement dit, lorsque tous les autres processus sont à l'état prêt. Des valeurs de -2 et -1 sont en dessous de la normale, ainsi les autres processus auront la priorité. 0 est la valeur normale. +1 et +2 accordent une plus haute priorité au programme de jeu, permettant d'obtenir une plus grande vitesse d'exécution du jeu ainsi qu'un affichage plus coulé. Mais les autres processus bénéficieront de moins de temps processeur. +3 indique un mode en temps réel. Dans ce dernier mode, tout le temps processeur sera théoriquement attribué au jeu. Cela peut conduire à de sérieux problèmes en ce qui concerne les autres applications tournant actuellement sur l'ordinateur. Même les événements clavier ou souris (comme par exemple le clic sur le bouton de fermeture de la fenêtre), pourront ne plus être gérés correctement par Windows. Aussi, cette fonction sera

à utiliser uniquement si vous souhaitez véritablement réserver tout le temps processeur pour le jeu. Il est conseillé d'utiliser cette fonction avec précaution et de sauvegarder le jeu avant de l'employer.

Interactions avec l'Utilisateur

Il n'existe pas de jeu ne demandant pas d'interactivité de la part de l'utilisateur. La méthode standard pour créer de l'interactivité dans *Game Maker* est de placer des actions dans les événements souris ou clavier. Mais parfois, vous aurez besoin davantage de contrôle. A partir d'un code programme, vous pourrez vérifier quelles sont les touches du clavier qui ont été pressées et contrôler la position de la souris et savoir si on a cliqué sur l'un de ses boutons. Normalement, vous devrez vérifier ces aspects dans l'événement **step** de certains objets contrôleurs et effectuer les actions en conséquence.

Des informations concernant l'interactivité avec l'utilisateur peuvent être trouvées dans les pages suivantes :

Le Clavier

Les variables et fonctions clavier suivantes existent afin de permettre une interactivité entre le jeu et l'utilisateur :

keyboard_lastkey Code de la dernière touche pressée. Voir ci-dessous pour connaître la liste complète des codes du clavier. Vous pouvez modifier le code retourné. Par exemple, vous pouvez lui affecter la valeur 0 si vous souhaitez le gérer par vous-même.
keyboard_key Code de l'actuelle touche pressée (voir plus bas; 0 si aucune).
keyboard_lastchar Dernier caractère pressé (retourné dans une chaîne de caractères).
keyboard_string La chaîne contient au plus les 1024 derniers caractères frappés au clavier. Cette chaîne ne contiendra que les caractères tapés imprimables. La chaîne assure une gestion correcte lors de la pression de la touche **retour arrière** (backspace) en effaçant le dernier caractère à chaque appui sur cette dernière.

Il est parfois utile de simuler l'appui d'une touche par une autre. C'est par exemple le cas lorsque vous souhaitez que le joueur utilise à la fois les touches fléchées et les touches du pavé numérique. Plutôt que de dupliquer les actions, vous pourrez faire en sorte que les touches du clavier numérique correspondent à celles des touches fléchées. Vous voudrez peut-être également implémenter un mécanisme dans lequel le joueur pourra choisir lui-même les touches à utiliser dans le jeu. Les fonctions suivantes sont dédiées à cet usage :

keyboard_set_map(key1,key2) Simule l'emploi de la touche de code **key1** avec la touche de code **key2**.
keyboard_get_map(key) Retourne l'actuel mappage de la touche **key**.
keyboard_unset_map() Réinitialise toutes les touches à leur valeur d'origine (plus de mappage).

Pour contrôler qu'une touche particulière du clavier ou qu'un bouton de la souris aient été pressés, vous pouvez employer les fonctions ci-après. Cela sera particulièrement utile lorsque plusieurs touches seront pressées simultanément.

keyboard_check(key) Retourne si la touche de code **key**, est actuellement pressée.
keyboard_check_pressed(key) Retourne si la touche de code **key** a été pressée depuis le dernier **step**.

keyboard_check_released(key) Retourne si la touche de code **key** a été relâchée depuis le dernier **step**.

keyboard_check_direct(key) Retourne si la touche de code **key** a été pressée en effectuant directement un contrôle hardware. Le résultat est indépendant de l'application en cours. Cela permettra d'effectuer davantage de vérifications. En particulier, vous pouvez utiliser les code de touches **vk_lshift**, **vk_lcontrol**, **vk_lalt**, **vk_rshift**, **vk_rcontrol** et **vk_ralt** pour vérifier si les touches **shift gauche**, **shift droit**, **contrôle** (**Ctrl**) ou encore **alt** ont été pressées.

Les routines suivantes peuvent être utilisées pour vérifier l'état du clavier :

keyboard_get_numlock() Retourne si la touche de verrouillage numérique (**Verr num**) est actuellement en fonction.

keyboard_set_numlock(on) Mets en service (**true**) ou hors service (**false**) la touche de verrouillage numérique (**Verr num**).

keyboard_key_press(key) Simule la pression d'une touche avec le code touche **key** indiqué.

keyboard_key_release(key) Simule le relâchement d'une touche avec le code touche **key** indiqué.

Les constantes suivantes existent et concernent les codes touches virtuelles :

vk_nokey code touche indiquant qu'aucune touche n'a été pressée

vk_anykey code touche indiquant qu'une touche quelconque a été pressée

vk_left code touche correspondant à la touche flèche de gauche

vk_right code touche correspondant à la touche flèche de droite

vk_up code touche correspondant à la touche flèche du haut

vk_down code touche correspondant à la touche flèche du bas

vk_enter touche **Entrée** (enter)

vk_escape touche **Esc** (escape)

vk_space touche **espace**

vk_shift touche **shift**

vk_control touche **contrôle** (**Ctrl**)

vk_alt touche **alt**

vk_backspace touche **retour arrière** (backspace)

vk_tab touche tabulation (**tab**)

vk_home touche **home** (place le curseur en haut et à gauche de l'écran)

vk_end touche **Fin** (end)

vk_delete touche **Suppr** (delete)

vk_insert touche **Inser** (insert)
vk_pageup touche **page précédente** (pageup)
vk_pagedown touche **page suivante** (pagedown)
vk_pause touche **pause/break**
vk_printscreen touche **printscreen/sysrq**
vk_f1 ... vk_f12 codes touches pour les touches de fonction **F1** à **F12**
vk_numpad0 ... vk_numpad9 touches correspondant aux chiffres du pavé numérique
vk_multiply touche multiplication **(*)** du pavé numérique
vk_divide touche division **(/)** du pavé numérique
vk_add touche addition **(+)** du pavé numérique
vk_subtract touche soustraction **(-)** du pavé numérique
vk_decimal touche point décimal **(.)** du pavé numérique

Pour les touches correspondant aux lettres, utilisez par exemple `ord('A')` (lettres capitales). Pour les touches correspondant aux chiffres, utilisez par exemple `ord('5')` pour obtenir la touche <5>. Les constantes suivantes peuvent être seulement utilisées avec **keyboard_check_direct** :

vk_lshift touche **shift gauche**
vk_lcontrol touche **contrôle** (Ctrl) **gauche**
vk_lalt touche **alt gauche**
vk_rshift touche **shift droite**
vk_rcontrol touche **contrôle** (Ctrl) **droite**
vk_ralt touche **alt droite**

Par exemple, supposons que vous ayez un objet que le joueur doit contrôler avec les touches fléchées, vous pourrez alors insérer le code programme suivant dans l'événement **step** de l'objet :

```
{
  if (keyboard_check(vk_left))  x -= 4;
  if (keyboard_check(vk_right)) x += 4;
  if (keyboard_check(vk_up))    y -= 4;
  if (keyboard_check(vk_down))  y += 4;
}
```

Bien entendu, il eût été beaucoup plus simple de mettre le code ci-dessus dans les événements clavier de l'objet.

Il existe quelques fonctions supplémentaires concernant l'interactivité clavier avec l'utilisateur.

`keyboard_clear(key)` Efface l'état de la touche **key** mentionnée. Cela signifie que celle-ci ne générera plus aucun événement clavier jusqu'au prochain appui sur la touche **key**.

`io_clear()` Efface tous les états du clavier et de la souris.

`io_handle()` Gestion des entrées-sorties **(I/O)** utilisateur, avec mise à jour du statut du clavier et de la souris.

`keyboard_wait()` Attends que l'utilisateur presse une touche au clavier.

La Souris

Concernant les interactions avec la souris, les variables et les fonctions suivantes existent :

mouse_x* Coordonnée en X de la souris dans la room. Ne peut être changée.
mouse_y* Coordonnée en Y de la souris dans la room. Ne peut être modifiée.
mouse_button Bouton de la souris actuellement pressé. Retourne comme valeur **mb_none**, **mb_any**, **mb_left**, **mb_middle**, ou **mb_right**.
mouse_lastbutton Dernier bouton pressé de la souris.

Pour déterminer le bouton de la souris venant d'être pressé, vous pouvez utiliser les fonctions suivantes. Cela sera particulièrement utile lorsque plusieurs touches seront pressées en même temps.

mouse_check_button(numb) Retourne si le bouton de la souris **numb** est actuellement pressé (utilisez comme valeurs **mb_none**, **mb_left**, **mb_middle**, ou **mb_right**).
mouse_check_button_pressed(numb) Retourne si le bouton de la souris **numb** a été pressé depuis le dernier **step**.
mouse_check_button_released(numb) Retourne si le bouton de la souris **numb** a été relâché depuis le dernier **step**.

Il existe quelques fonctions supplémentaires ayant traits aux interactions avec la souris.

mouse_clear(button) Efface l'état du bouton **button** de la souris. Cela signifie que ce bouton ne générera plus aucun événement souris jusqu'à ce que le joueur relâche ce bouton puis le presse de nouveau.
io_clear() Efface tous les états du clavier et de la souris.
io_handle() Gestion des entrées-sorties **(I/O)** utilisateur, avec mise à jour des statuts du clavier et de la souris.
mouse_wait() Attends que l'utilisateur presse un bouton de la souris
.

Les Joysticks

Certains événements sont associés aux joysticks. Il existe tout un ensemble de fonctions permettant un contrôle total sur les joysticks. *Game Maker* supporte jusqu'à deux joysticks. Aussi, toutes ces fonctions prennent comme argument l'**ID** du joystick.

joystick_exists(id) Retourne si l'**ID** du joystick (1 ou 2) existe.
joystick_name(id) Retourne le nom du joystick.
joystick_axes(id) Retourne le nombre d'axes du joystick.
joystick_buttons(id) Retourne le nombre de boutons du joystick.
joystick_has_pov(id) Retourne si le joystick possède des fonctionnalités de point de vue.
joystick_direction(id) Retourne le code touche (**vk_numpad1** à **vk_numpad9**) correspondant à la direction de l'**ID** du joystick (1 ou 2).
joystick_check_button(id,numb) Retourne si le bouton **numb** du joystick **ID** est pressé (numb allant de 1 à 32).
joystick_xpos(id) Retourne la position (-1 à 1) de l'axe en **X** de l'ID du joystick.
joystick_ypos(id) Retourne la position en **Y** du joystick.
joystick_zpos(id) Retourne la position en **Z** du joystick (si celui possède un axe Z).
joystick_rpos(id) Retourne la position du manche (ou quatrième axe).
joystick_upos(id) Retourne la position en **U** du joystick (ou cinquième axe).
joystick_vpos(id) Retourne la position en **V** du joystick (ou sixième axe).
joystick_pov(id) Retourne la position du point de vue du joystick d'id **ID**. Cela correspond à un angle entre **0** et **360** degrés. **0** correspond vers l'avant, **90** vers la droite, **180** vers le bas et **270** vers la gauche. Lorsque aucune direction de point de vue n'est sélectionnée par le joueur, la valeur **-1** est retournée.

Les graphiques du jeu

Les graphiques sont l'un des éléments essentiels des jeux. *Game Maker* prend en charge la plupart d'entre eux et cela suffira pour la création de jeux simples. Mais parfois, il sera nécessaire d'avoir davantage de contrôle. Pour certains aspects, vous pourrez utiliser les actions mais en programmant vous-même, vous pourrez contrôler beaucoup plus d'aspects de votre jeu. Ce chapitre décrit toutes les variables et fonctions disponibles et fournit une information plus détaillée sur leur mode d'emploi.

Des informations sur les graphiques du jeu peuvent être consultées dans les pages suivantes :

Sprites et Images

Chaque objet dispose d'un sprite qui lui est associé. Celui-ci peut être une simple image ou constitué de plusieurs images. Pour chacune des instances de l'objet, le programme dessine l'image correspondante sur l'écran, à son origine (comme définie dans les propriétés du sprite) et à la position **(x,y)** de l'instance. En présence d'images multiples, le programme effectue un affichage en boucle de ces images afin de créer un effet d'animation. Il existe quelques variables qui influencent la façon dont l'image est dessinée. Cela pourra être utilisé pour modifier les effets. Chaque instance possède les variables suivantes :

visible Si **visible** a la valeur **true** (1), l'image sera affichée sinon elle ne le sera pas. Les instances invisibles restent cependant actives et créent toujours des événements de collision mais vous ne les verrez pas. Paramétrer la visibilité à **false** peut être utile par exemple pour les objets contrôleurs (donnez leur le type **non-solid** pour éviter l'apparition d'événements de collision) ou encore pour des interrupteurs cachés dans votre jeu.
sprite_index C'est l'index du sprite courant de l'instance. Vous pouvez le changer en donnant un sprite différent à l'instance. Vous pouvez utiliser comme valeur le nom des sprites que vous aurez définis. Changer le sprite ne changera pas l'index de la sous-image actuellement visible.
sprite_width* Indique la largeur du sprite. Cette valeur ne peut être modifiée mais il vous est possible de l'utiliser.
sprite_height* Indique la hauteur du sprite. Cette valeur ne peut être modifiée mais il vous est possible de l'utiliser.
sprite_xoffset* Indique l'offset horizontal du sprite comme défini dans ses propriétés. Cette valeur ne peut être modifiée mais il vous est possible de l'utiliser.
sprite_yoffset* Indique l'offset vertical du sprite comme défini dans ses propriétés. Cette valeur ne peut être modifiée mais il vous est possible de l'utiliser.
image_number* Le nombre de sous-images de l'actuel sprite de l'instance (ne peut être changé).
image_index Lorsque l'image possède plusieurs sous-images, le programme effectue un affichage en boucle de ces images. Cette variable indique la sous-image actuellement affichée (elles sont numérotées en partant de 0). Vous pouvez modifier l'image courante en changeant cette variable. Le programme poursuivra l'affichage en boucle des images mais en commençant à ce nouvel index (la valeur peut comprendre une partie fractionnaire. Dans ce cas, l'index sera toujours

arrondi à la valeur entière inférieure afin d'obtenir la sous-image pouvant être correctement affichée).

image_speed La vitesse à laquelle le programme affiche les sous-images. Une valeur de 1 indique que lors de chaque step, le programme n'affichera qu'une seule image. De plus petites valeurs rendront plus lent l'affichage des sous-images, en affichant plusieurs fois chaque même sous-image à chaque step. Une valeur plus grande sautera des sous-images rendant l'animation plus rapide. Parfois, vous souhaiterez rendre visible une sous-image particulière et ne voudrez pas que le programme effectue un affichage en boucle des autres images. On peut réaliser cela en paramétrant la vitesse à 0 et en choisissant la sous-image souhaitée. Par exemple, supposons que vous ayez un objet en rotation et que vous vouliez créer un sprite qui dispose de sous-images pour effectuer un certain nombre d'orientations (dans le sens contraire des aiguilles d'une montre). Aussi, dans l'événement step de l'objet, vous pourrez écrire ceci :

```
{
  image_index = direction * image_number/360;
  image_speed = 0;
}
```

depth Habituellement, les images sont affichées selon l'ordre de création des instances Vous pouvez modifier cela en réglant la profondeur de l'image (image **depth**). Par défaut, cette valeur est de 0 à moins que vous n'ayez indiqué une autre valeur dans les propriétés de l'objet. Plus cette valeur sera grande et plus l'instance sera éloignée (il est possible d'utiliser des valeurs négatives). Les instances de profondeur très grande seront situées derrière les instances de profondeur plus faible. Le paramétrage de la profondeur vous garantira que les instances seront affichées dans l'ordre souhaité (ex: un avion devant des nuages). Les instances d'arrière-plan devront avoir une très grande profondeur (donc positive) alors que celles de premier plan auront une profondeur très faible (soit négative).

image_xscale Ceci est un facteur d'échelle pour rendre les images plus grandes ou plus petites. Une valeur de 1 indique une taille normale. Vous devrez paramétrer séparément l'échelle horizontale **xscale** et celle verticale **yscale**. La modification de l'échelle aura une incidence sur les valeurs de largeur et de hauteur de l'image et influencera aussi les événements de collision comme on pouvait s'y attendre. On peut utiliser le changement d'échelle pour obtenir un effet 3D.

Une valeur de -1 donnera au sprite un effet de miroir horizontal.

image_yscale Correspond à l'échelle verticale **yscale**. La valeur 1 indique aucune mise à l'échelle. Une valeur de -1 effectuera un retournement du sprite dans le sens vertical.

image_angle Correspond à l'angle de rotation du sprite. Vous devrez spécifier cette valeur en degrés et en suivant le sens contraire des aiguilles d'une montre. Une valeur de 0 signifie aucune rotation. ***Cette variable n'est disponible que dans la version enregistrée !***

image_alpha La valeur de transparence (alpha) à utiliser lors de l'affichage de l'image. Une valeur de 1 correspond à un réglage normal de l'opacité alors qu'une valeur de 0 signifie une transparence totale.

image_blend La couleur de mélange de couleur (Blending color) utilisée lors de l'affichage du sprite. La valeur **c_white** est celle utilisée par défaut. Si vous indiquez une autre valeur, l'image sera mélangée avec cette couleur. Cela peut être utilisé pour colorier le sprite à la volée. ***Cette variable n'est disponible que dans la version enregistrée !***

bbox_left* Côté gauche de la boîte des bords utilisée par l'image de l'instance (tient compte de l'échelle).

bbox_right* Côté droit de la boîte des bords de l'image de l'instance.

bbox_top* Côté supérieur de la boîte des bords de l'image de l'instance.

bbox_bottom* Côté inférieur de la boîte des bords de l'image de l'instance.

Les Arrière-Plans (Backgrounds)

Chaque room peut posséder jusqu'à 8 arrière-plans (backgrounds). Il existe également une couleur d'arrière-plan. Vous pouvez modifier tous les aspects de ces arrière-plans en ayant recours à la programmation et à l'aide des variables suivantes (veuillez noter que certaines de ces variables se rapportent à des tableaux dont les indices vont de 0 à 7, ces derniers correspondant à la numérotation des différents arrière-plans) :

background_color Couleur d'arrière-plan de la room.
background_showcolor Indique si l'on doit effacer la fenêtre avec la couleur d'arrière-plan.
background_visible[0..7] Indique si l'image d'un arrière-plan particulier doit être visible.
background_foreground[0..7] Indique si l'arrière-plan est actuellement un avant-plan.
background_index[0..7] Index de l'image d'arrière-plan pour l'arrière-plan.
background_x[0..7] position en **X** de l'image d'arrière-plan.
background_y[0...7] Position en **Y** de l'image d'arrière-plan.
background_width[0...7]* Largeur de l'image d'arrière-plan.
background_height[0...7]* Hauteur de l'image d'arrière-plan.
background_htiled[0..7] Indique si l'arrière-plan utilise des tuiles dans le sens horizontal.
background_vtiled[0..7] Indique si l'arrière-plan utilise des tuiles dans le sens vertical.
background_xscale[0..7] Facteur d'échelle horizontal pour l'arrière-plan (cette variable doit être positive : vous n'êtes pas autorisé à employer une valeur négative pour donner un effet miroir à l'arrière-plan).
background_yscale[0..7] Facteur d'échelle vertical pour l'arrière-plan (cette variable doit être positive : vous ne pouvez pas utiliser une valeur négative pour créer un effet de retournement de l'arrière-plan).
background_hspeed[0..7] Vitesse de scrolling horizontal de l'arrière-plan (pixels par step).
background_vspeed[0..7] Vitesse de scrolling vertical de l'arrière-plan (pixels par step).
background_blend[0..7] Couleur de mélange à utiliser lors de l'affichage de l'arrière-plan. La valeur **c_white** est celle utilisée par défaut. ***Uniquement disponible dans la version enregistrée !***

background_alpha[0..7] Valeur de transparence (alpha) utilisée lors de l'affichage de l'arrière-plan. Une valeur de **1** correspond à la valeur normale alors qu'une valeur de **0** indique une transparence totale.

Affichage des Sprites et des Arrière-plans (Backgrounds)

Les objets possèdent habituellement un sprite qui leur est associé et qui s'affiche dans la room. Mais il vous est possible d'utiliser également l'événement d'affichage (**draw event**) pour dessiner d'autres choses. Cette section et les deux suivantes présentent ce qu'il est possible de faire dans ce domaine. En premier lieu, il existe des fonctions pour dessiner des sprites et des arrière-plans et ce de différentes façons. Cela vous donnera davantage de contrôle sur l'apparence du sprite. Vous pouvez également dessiner des arrière-plans (ou seulement une partie d'entre eux).

draw_sprite(sprite,subimg,x,y) Dessine la sous-image **subimg** (-1 = image courante) du sprite d'index **sprite** avec son origine en position **(x,y)** (sans couleur de mélange et transparence alpha).
draw_sprite_stretched(sprite,subimg,x,y,w,h) Dessine le sprite en l'étirant de telle manière qu'il remplisse la zone située au coin supérieur gauche **(x,y)**, de largeur **w** et de hauteur **h**.
draw_sprite_tiled(sprite,subimg,x,y) Dessine le sprite sous forme de tuiles de façon qu'il remplisse entièrement la room. **(x,y)** correspond aux coordonnées où le sprite de départ est affiché.
draw_sprite_part(sprite,subimg,left,top,width,height,x,y) Dessine la sous-image indiquée **subimg** (-1 = image courante) du **sprite** en plaçant le coin supérieur gauche de cette sous-image à la position **(x,y)**.

draw_background(back,x,y) Dessine l'arrière-plan à la position **(x,y)** (sans couleur de mélange et transparence alpha).
draw_background_stretched(back,x,y,w,h) Dessine l'arrière-plan en l'étirant de telle manière qu'il occupe toute la région indiquée en paramètre.
draw_background_tiled(back,x,y) Dessine l'arrière-plan sous forme de tuiles de façon à ce qu'il remplisse entièrement la room.
draw_background_part(back,left,top,width,height,x,y) Dessine la zone indiquée de l'arrière-plan en plaçant le coin supérieur gauche de cette zone à la position **(x,y)**.

Les fonctions suivantes sont des fonctions étendues de celles vues précédemment. ***Elles ne peuvent être utilisées que dans la version enregistrée !***

draw_sprite_ext(sprite,subimg,x,y,xscale,yscale,rot,color,alpha) Dessine le sprite avec un facteur d'échelle **xscale** et **yscale**, en appliquant une rotation dans le sens contraire des aiguilles d'une montre de **rot** degrés. La couleur est celle utilisée pour le mélange de couleur (utilisez **c_white** pour ne pas utiliser de couleur de mélange) et **alpha** indique le niveau de transparence avec lequel les images seront fusionnées avec le fond. Une valeur de **0** créera un sprite totalement transparent. Une valeur de **1** rendra le sprite entièrement solide. Cette fonction est capable de créer de très grands effets (par exemple, des explosions partiellement transparentes).

draw_sprite_stretched_ext(sprite,subimg,x,y,w,h,color,alpha) Dessine le sprite en l'étirant de telle manière qu'il remplisse la région indiquée en paramètre, avec le coin supérieur gauche situé à la position **(x,y)**, de largeur **w** et de hauteur **h**. La couleur sera la couleur de mélange et **alpha** correspond au paramétrage de la transparence.

draw_sprite_tiled_ext(sprite,subimg,x,y,xscale,yscale,color,alpha) Dessine le sprite sous forme de tuiles de façon à remplir complètement la room mais cette fois-ci en utilisant une mise à l'échelle, une couleur et un paramétrage de la transparence.

draw_sprite_part_ext(sprite,subimg,left,top,width,height,x,y,xscale,yscale,color,alpha) Dessine la sous-image indiquée **subimg** (-1 = image courante) du sprite en plaçant le coin supérieur gauche de cette image à la position **(x,y)** mais en utilisant une mise à l'échelle, une couleur et un paramétrage de la transparence.

draw_sprite_general(sprite,subimg,left,top,width,height,x,y,xscale,yscale,rot,c1,c2,c3,c4,alpha) La fonction d'affichage la plus générale pour les sprites. Elle dessine la sous-image indiquée **subimg** (-1 = image courante) du sprite en plaçant le coin supérieur gauche de cette image à la position **(x,y)** mais en utilisant aussi une mise à l'échelle, un angle de rotation, une couleur pour chacun des quatre vertices (haut-gauche, haut-droit, bas-droit et bas-gauche) et une valeur de transparence alpha. Veuillez noter que la rotation s'effectue à proximité immédiate du coin supérieur gauche de la sous-image.

draw_background_ext(back,x,y,xscale,yscale,rot,color,alpha) Dessine l'arrière-plan avec une mise à l'échelle et une rotation en utilisant une couleur de mélange (utilisez **c_white** pour ne pas avoir de couleur de mélange) et une transparence **alpha** (**0-1**).

draw_background_stretched_ext(back,x,y,w,h,color,alpha) Dessine l'arrière-plan en l'étirant selon les coordonnées indiquées. La couleur est celle de la

couleur de mélange et la valeur **alpha** indique le paramétrage de la transparence.

draw_background_tiled_ext(back,x,y,xscale,yscale,color,alpha) Dessine l'arrière-plan sous forme de tuiles de manière à remplir totalement la room mais en utilisant cette fois-ci une mise à l'échelle, une couleur et un paramétrage de la transparence.

draw_background_part_ext(back,left,top,width,height,x,y,xscale, yscale,color,alpha) Dessine la zone indiquée de l'arrière-plan en plaçant le coin supérieur gauche de cette zone à la position **(x,y)** mais en utilisant également une mise à l'échelle, une couleur et un paramétrage de la transparence.

draw_background_general(back,left,top,width,height,x,y,xscale,y scale,rot,c1,c2,c3,c4,alpha) La fonction d'affichage la plus générale pour les arrière-plans. Elle dessine la région indiquée de l'arrière-plan en plaçant le coin supérieur gauche de cette zone à la position **(x,y)** mais en employant aussi une mise à l'échelle, un angle de rotation, une couleur pour chacun des quatre vertices (haut-gauche, haut-droit, bas-droit et bas-gauche) et une valeur de transparence **alpha**. Veuillez noter que la rotation s'effectue à proximité du coin supérieur gauche de la région indiquée.

Affichage de Formes (Shapes)

Il existe une collection complète de fonctions destinées à l'affichage des différentes formes. Il existe également des fonctions pour afficher du texte (se reporter à la section suivante). Vous ne pouvez utiliser ces fonctions que dans les événements d'affichage (draw event) d'un objet. Ces fonctions en général ne présentent d'intérêt que si on ne les utilise que dans du code GML. Veuillez bien comprendre que les collisions entre instances sont déterminées par leurs sprites (ou leurs masques) et non pas par ce que vous êtes en train d'afficher. Les fonctions d'affichages suivantes existent et peuvent être utilisées pour afficher des formes de base (basic shapes).

draw_clear(col) Efface complètement la room en utilisant la couleur indiquée (pas de couleur de mélange alpha).
draw_clear_alpha(col,alpha) Efface complètement la room avec la couleur indiquée et en utilisant une valeur **alpha** (utile en particulier pour les surfaces).
draw_point(x,y) Affiche un point à la position **(x,y)** avec la couleur courante.
draw_line(x1,y1,x2,y2) Affiche une ligne à partir de **(x1,y1)** jusqu'à **(x2,y2)**.
draw_rectangle(x1,y1,x2,y2,outline) Affiche un rectangle. **outline** indique si seule la bordure doit être affichée (valeur **true**) ou si le rectangle doit être plein (valeur **false**).
draw_roundrect(x1,y1,x2,y2,outline) Affiche un rectangle avec bords arrondis. **outline** signale si seule la bordure doit être affichée (**true**) ou si le rectangle doit être plein (**false**).
draw_triangle(x1,y1,x2,y2,x3,y3,outline) Affiche un triangle. **outline** précise si seule la bordure doit être affichée (**true**) ou si le triangle doit être plein (**false**).
draw_circle(x,y,r,outline) Affiche un cercle à la position **(x,y)** et de rayon **r**. **outline** indique si seule la bordure doit être affichée (**true**) ou si le cercle doit être plein (**false**).
draw_ellipse(x1,y1,x2,y2,outline) Dessine une ellipse. **outline** précise si seule la bordure doit être affichée (**true**) ou si l'ellipse doit être pleine (**false**).
draw_arrow(x1,y1,x2,y2,size) Affiche une flèche de la position **(x1,y1)** à la position **(x2,y2)**. **size** fournit la taille de la flèche exprimée en pixels.
draw_button(x1,y1,x2,y2,up) Affiche un bouton. **up** indique si le bouton est non pressé **(1)** ou pressé **(0)**.
draw_path(path,x,y,absolute) Vous pourrez afficher avec cette fonction

le chemin indiqué dans la room avec comme position de départ la position **(x,y)**. Si **absolute** est à **true**, le chemin sera affiché à la position où il a été défini et les valeurs **x** et **y** seront purement et simplement ignorées.

draw_healthbar(x1,y1,x2,y2,amount,backcol,mincol,maxcol,direction,showback,showborder) A l'aide de cette fonction, vous pourrez afficher une barre de santé (ou n'importe quelle barre représentant une valeur comme par exemple les dommages subis). Les arguments **x1**, **y1**, **x2** et **y2** indiquent la largeur totale (100%) de la barre. **amount** correspond au pourcentage de la barre qui doit être remplie (valeur entre 0 et 100). **backcol** représente la couleur de l'arrière-plan pour cette barre. **mincol** et **maxcol** indiquent la couleur que doit avoir la barre lorsque respectivement le pourcentage est de **0** ou de **100**. Pour les valeurs intermédiaires, la couleur de la barre sera interpolée. Ainsi, vous pourrez facilement créer une barre qui passe progressivement du vert au rouge. La direction correspond à celle de la barre affichée. **0** signale que la barre est ancrée à gauche, **1** à droite, **2** en haut et **3** en bas. Enfin, **showback** mentionne si un fond doit être affiché et **showborder** précise si la boîte et la barre doivent posséder une ligne noire pour la bordure.

La plupart des fonctions suivantes utilisent le paramétrage de couleur et la valeur alpha que l'on peut modifier avec les fonctions ci-dessous.

draw_set_color(col) Règle la couleur d'affichage à utiliser désormais pour les primitives de dessin.

draw_set_alpha(alpha) Règle la valeur de transparence **alpha** à utiliser à partir de maintenant pour les primitives de dessin. La valeur doit être comprise entre **0** et **1**. Une valeur de **0** indique une transparence totale alors que la valeur **1** correspond à une opacité complète.

draw_get_color() Retourne la couleur d'affichage pour les primitives de dessin.

draw_get_alpha() Retourne la valeur **alpha** utilisée par les primitives de dessin.

Les couleurs prédéfinies ci-dessous peuvent être utilisées :

c_aqua
c_black
c_blue
c_dkgray
c_fuchsia
c_gray
c_green

c_lime
c_ltgray
c_maroon
c_navy
c_olive
c_purple
c_red
c_silver
c_teal
c_white
c_yellow

Les fonctions suivantes vous aideront à créer les couleurs que vous souhaitez.

make_color_rgb(red,green,blue) Retourne une couleur avec les composants indiqués **rouge** (red), **vert** (green) et **bleu** (blue) pour lesquels *red*, *green* et *blue* doivent être des valeurs comprises entre **0** et **255**.
make_color_hsv(hue,saturation,value) Retourne une couleur avec les composants de **tonalité**, de **saturation** et de **valeur** indiqués (chaque valeur devant être comprise entre **0** et **255**).
color_get_red(col) Retourne le composant **rouge** (red) de la couleur mentionnée.
color_get_green(col) Retourne le composant **vert** (green) de la couleur indiquée.
color_get_blue(col) Retourne le composant **bleu** (blue) de la couleur précisée.
color_get_hue(col) Retourne le composant de **tonalité** de la couleur indiquée.
color_get_saturation(col) Retourne le composant de **saturation** de la couleur mentionnée.
color_get_value(col) Retourne le composant de **valeur** de la couleur précisée.
merge_color(col1,col2,amount) Retourne une couleur mélangée avec les couleurs *col1* et *col2*. Le mélange est déterminé en utilisant une valeur donnée. Une valeur de **0** correspond à la couleur **col1**, une valeur de **1** à la couleur **col2** et des valeurs intermédiaires à des couleurs fusionnées avec **col1** et **col2**.

Les fonctions diverses suivantes peuvent être également utilisées.

`draw_getpixel(x,y)` Retourne la couleur du pixel à la position **(x,y)** de la room. Cette fonction n'est pas très rapide : il convient donc de l'utiliser en connaissance de cause.

`screen_save(fname)` Sauvegarde l'écran dans une image de type **BMP** dans le fichier de nom **fname**. Très utile pour effectuer une copie d'écran de votre jeu.

`screen_save_part(fname,x,y,w,h)` Sauvegarde une partie de l'écran dans le fichier de nom **fname**.

Polices de caractères et Texte

Vous aurez parfois besoin dans vos jeux d'afficher des textes. Pour ce faire, vous indiquerez la police (ou fonte) de caractères à utiliser. Les polices ou fontes peuvent être définies en créant des ressources de polices de caractères (font resources) (en utilisant soit la programmation dans *Game Maker*, soit en employant des fonctions pour créer ces ressources). Plusieurs fonctions existent et permettent d'afficher des textes de différentes façons. Pour chacune des fonctions employées, vous spécifierez la position du texte à l'écran. Deux fonctions seront à utiliser pour régler l'alignement horizontal et vertical du texte en accord avec la position indiquée.

Les fonctions suivantes peuvent être employées pour afficher du texte :

draw_set_font(font) Indique la police de caractères à utiliser pour afficher du texte. La valeur **-1** sera utilisée pour employer la fonte par défaut (**Arial 12**).
draw_set_halign(halign) Règle l'alignement horizontal à utiliser lors de l'affichage de texte. Choisissez comme paramètre **halign** l'une des constantes ci-dessous :

fa_left
fa_center
fa_right

draw_set_valign(valign) Détermine l'alignement vertical à utiliser pour afficher du texte. Les constantes suivantes peuvent être utilisées :

fa_top
fa_middle
fa_bottom

draw_text(x,y,string) Affiche la chaîne à la position **(x,y)**, à l'aide de l'actuelle couleur du crayon et la valeur alpha. Un symbole # ou un retour chariot **chr(13)** ou encore un saut à la ligne suivante **chr(10)** seront interprétés comme des caractères de nouvelle ligne (newline). Ainsi, vous serez en mesure d'afficher du texte sur plusieurs lignes (utilisez **\#** si vous souhaitez utiliser le symbole **#** lui-même).
draw_text_ext(x,y,string,sep,w) Similaire à la fonction précédente mais cette fois-ci, vous pourrez indiquer deux valeurs supplémentaires. En premier lieu, le paramètre **sep** indiquera la distance de séparation entre deux lignes pour tout

texte s'affichant sur plusieurs lignes. Utilisez la valeur **-1** afin d'obtenir la distance par défaut. Employez le paramètre **w** pour préciser la largeur du texte en pixels. Les lignes dépassant cette longueur seront sectionnées avec un espace ou un trait d'union. Utilisez la valeur **-1** pour interdire le sectionnement des lignes.

string_width(string) Affiche la largeur de la chaîne de la police courante telle qu'elle serait affichée avec la fonction **draw_text()**. A utiliser pour positionner des graphiques de manière précise.

string_height(string) Hauteur de la chaîne de la police courante telle qu'elle serait affichée avec la fonction **draw_text()**.

string_width_ext(string,sep,w) Largeur de la chaîne de la police courante telle quelle serait affichée avec la fonction **draw_text_ext()**. On l'utilise pour positionner précisément des graphiques.

string_height_ext(string,sep,w) Hauteur de la chaîne de la police courante telle qu'elle serait affichée avec la fonction **draw_text_ext()**.

Les routines suivantes vous permettront d'afficher du texte selon une certaine échelle et orientation mais également vous autoriseront l'emploi de couleurs progressives dans vos textes.
Ces fonctions ne sont disponibles que dans la version enregistrées !

draw_text_transformed(x,y,string,xscale,yscale,angle) Affiche la chaîne à la position **(x,y)** de la même manière que les fonctions vues plus haut mais en effectuant une mise à l'échelle horizontale et verticale à l'aide des facteurs indiqués et en inclinant le texte de **angle** degrés dans le sens contraire des aiguilles d'une montre.

draw_text_ext_transformed(x,y,string,sep,w,xscale,yscale,angle) Combine les fonctions **draw_text_ext** et **draw_text_transformed**. Cela vous permettra d'afficher du texte sur plusieurs lignes tout en appliquant une orientation et une mise à l'échelle de ce texte.

draw_text_color(x,y,string,c1,c2,c3,c4,alpha) Affiche la chaîne à la position **(x,y)** comme précédemment. Les quatre couleurs indiquent respectivement les couleurs du coin supérieur gauche, supérieur droit, inférieur droit et inférieur gauche du texte. **alpha** correspond à la transparence alpha à utiliser (**0-1**).

draw_text_ext_color(x,y,string,sep,w,c1,c2,c3,c4,alpha) Similaire à la fonction *draw_text_ext()* mais avec des vertices de couleurs.

draw_text_transformed_color(x,y,string,xscale,yscale,angle,c1,c2,c3,c4,alpha) Similaire à la fonction *draw_text_transformed()* mais avec des vertices de couleurs.

draw_text_ext_transformed_color(x,y,string,sep,w,xscale,yscale,angle,c1,c2,c3,c4,alpha) Similaire à la fonction *draw_text_ext_transformed()* mais avec des vertices de couleurs.

Fonctions de Dessin Avancées

Ces fonctionnalités ne sont disponibles que dans la version enregistrée de Game Maker.

Nous avons vu et décrit précédemment un certain nombre de fonctions de base concernant l'affichage. Vous trouverez ici des fonctions supplémentaires qui vous offriront encore davantage de possibilités. En premier lieu, nous trouvons les fonctions pour afficher des formes (shapes) avec des couleurs en arc-en-ciel. En second lieu, nous détaillerons les fonctions permettant d'afficher des polygones plus généraux puis enfin, nous parlerons de la possibilité d'afficher des textures projetées sur des polygones.

Il existe les versions évoluées suivantes venant compléter les fonctions de base pour l'affichage. Chacune d'entre elles présente des paramètres supplémentaires concernant la couleur utilisée afin de déterminer la couleur des différents vertices. La couleur standard d'affichage n'est pas utilisée par ces fonctions.

`draw_point_color(x,y,col1)` Affiche un point à la position **(x,y)** dans la couleur indiquée **col1**.

`draw_line_color(x1,y1,x2,y2,col1,col2)` Affiche une ligne de la position **(x1,y1)** à la position **(x2,y2)**, avec interpolation de la couleur de **col1** à **col2**.

`draw_rectangle_color(x1,y1,x2,y2,col1,col2,col3,col4,outline)` Affiche un rectangle. Les quatre couleurs représentent les couleurs des vertex supérieur gauche, supérieur droit, inférieur droit et inférieur gauche. **outline** précise si seule la bordure doit être dessinée (**true**) ou si le rectangle doit être plein (**false**).

`draw_roundrect_color(x1,y1,x2,y2,col1,col2,outline)` Affiche un rectangle à bords arrondis. **col1** est la couleur du centre et **col2** celle du bord. **outline** indique si seule la bordure doit être affichée (**true**) ou si le rectangle doit être plein (**false**).

`draw_triangle_color(x1,y1,x2,y2,x3,y3,col1,col2,col3,outline)` Affiche un triangle. Les trois couleurs correspondent aux couleurs des trois vertices, couleurs qui seront interpolées dans le triangle. **outline** indique si seule la bordure doit être affichée (**true**) ou si le triangle doit être plein (**false**).

`draw_circle_color(x,y,r,col1,col2,outline)` Dessine un cercle à la position **(x,y)** et de rayon **r**. **col1** représente la couleur du centre et **col2** celle des bords. **outline** indique si seule la bordure doit être affichée (**true**) ou si le cercle doit être plein (**false**).

draw_ellipse_color(x1,y1,x2,y2,col1,col2,outline) Dessine une ellipse. **col1** est la couleur au centre et **col2** la couleur du bord. **outline** indique si seule la bordure doit être affichée (**true**) ou si l'ellipse doit être pleine (**false**).

Vous avez également la possibilité de dessiner de plus grandes primitives. Le fonctionnement est quelques peu différent. Vous commencerez à spécifier la primitive que vous souhaitez afficher. Ensuite, vous indiquerez les vertices (sommets) de la primitive puis enfin, vous terminerez la primitive qui sera ensuite affichée. Six types de primitives existent :

pr_pointlist Les vertices sont formés d'un jeu de points.
pr_linelist Les vertices sont un jeu de segments de ligne. Chaque paire de vertices forme un segment de ligne. Ainsi, il doit y avoir un jeu pair de vertices.
pr_linestrip Les vertices forment un ensemble de lignes dont la première est reliée à la seconde, la deuxième reliée à la troisième, etc. La dernière ligne ne sera pas reliée à la première. Vous devrez indiquer une copie supplémentaire du premier vertex.
pr_trianglelist Les vertices sont un ensemble de triangles. Chaque triplet de sommets forme un triangle. Ainsi, le nombre de vertices doit donc être un multiple de **3**.
pr_trianglestrip Les vertices forment à nouveau des triangles mais cette fois, cela fonctionne un peu différemment. Les trois premiers sommets forment un triangle. Les deux derniers vertices, en même temps que le prochain sommet, forme le second triangle, etc. Aussi, chaque nouveau vertex indique un nouveau triangle, relié au précédent.
pr_trianglefan Similaire à une liste de triangle mais cette fois, le premier vertex fait partie de tous les triangles. A nouveau, chaque nouveau vertex indique un nouveau triangle, relié au précédent vertex et au premier vertex.

Les fonctions suivantes existent pour les primitives de dessin.

draw_primitive_begin(kind) Débute une primitive du type indiqué.
draw_vertex(x,y) Ajoute le vertex **(x,y)** à la primitive, en utilisant la couleur et la valeur alpha définies précédemment.
draw_vertex_color(x,y,col,alpha) Ajoute le vertex **(x,y)** à la primitive, avec sa propre couleur et valeur alpha. Cela vous permet de créer des primitives avec des changements progressifs des valeurs de couleur et alpha.
draw_primitive_end() Termine la description de la primitive. Cette fonction dessine la primitive.

Il est enfin possible de dessiner des primitives en utilisant des sprites ou des arrière-plans comme textures. Lors de l'utilisation de texture, l'image est placée sur la primitive avec redimensionnement afin que cette dernière remplisse entièrement la primitive. Les textures sont utilisées pour ajouter des détails aux primitives, ex: un mur de briques. Pour utiliser des textures, vous devrez en premier lieu obtenir l'**ID** de la texture que vous souhaitez utiliser. Pour réaliser cela, utilisez les fonctions suivantes :

`sprite_get_texture(spr,subimg)` Retourne l'**ID** de la texture correspondant à la sous-image **subimg** du sprite mentionné.
`background_get_texture(back)` Retourne l'**ID** de la texture correspondant à l'arrière-plan indiqué.

La texture choisie peut ne pas être actuellement en mémoire vidéo. Le système la chargera alors pour vous en cas de besoin mais parfois, vous souhaiterez le faire par vous-même. Pour cela, il existe les fonctions suivantes :

`texture_preload(texid)` Charge immédiatement la texture en mémoire vidéo.
`texture_set_priority(texid,prio)` S'il y a trop peu de mémoire vidéo, certaines textures seront temporairement retirées de la mémoire afin de libérer de la place pour permettre le chargement des textures nécessaires. Les textures de priorité la plus basse seront supprimées en premier lieu. Par défaut, toutes les textures possèdent la priorité **0** mais il vous est possible de changer cette priorité à l'aide de cette commande (n'utilisez que des valeurs positives !)

Pour ajouter des textures à des primitives, vous devrez indiquer quelles parties des textures et à quel endroit ces dernières doivent être placées sur la primitive. Les positions de la texture sont indiquées avec des valeurs entre **0** et **1** mais cela peut poser problème. En effet, la taille des textures doit être une puissance de **2** (soit par exemple 32x32 ou 64x64). Si vous souhaitez utiliser des sprites ou des arrière-plans en guise de textures, il sera préférable de vérifier auparavant qu'ils possèdent cette taille. Sinon, cela ne marchera pas. Pour déterminer quelle partie de la texture est actuellement utilisée, vous pourrez utiliser les deux fonctions suivantes. Elles retournent une valeur entre **0** et **1** qui indique la largeur ou la hauteur de l'actuelle partie de la texture en cours d'utilisation. En précisant cette valeur comme coordonnée de texture, cela indiquera le côté droit ou inférieur de la texture.

`texture_get_width(texid)` Retourne la largeur de la texture correspondant à l'**ID** indiqué. La largeur doit être comprise entre **0** et **1**.
`texture_get_height(texid)` Retourne la hauteur de la texture correspondant à l'**ID** mentionné. La hauteur doit être comprise entre **0** et **1**.

Vous utiliserez les fonctions suivantes pour dessiner des primitives texturées :

draw_primitive_begin_texture(kind,texid) Débute une primitive de type indiqué et avec la texture mentionnée.
draw_vertex_texture(x,y,xtex,ytex) Ajoute un vertex **(x,y)** à la primitive et à la position **(xtex,ytex)** de la texture, avec mélange de couleur en utilisant la couleur et la valeur alpha choisies auparavant. **xtex** et **ytex** doivent normalement être compris entre **0** et **1** mais de plus grandes valeurs peuvent être employées, permettant ainsi une répétition de la texture (voir ci-dessous).
draw_vertex_texture_color(x,y,xtex,ytex,col,alpha) Ajoute un vertex **(x,y)** à la primitive à la position **(xtex,ytex)** de la texture, avec mélange de couleur en utilisant sa propre couleur et valeur **alpha**.
draw_primitive_end() Termine la description de la primitive. Cette fonction affichera la primitive.

Il y a trois fonctions influençant la manière dont les textures sont affichées :

texture_set_interpolation(linear) Indique si l'on doit utiliser une interpolation linéaire (**true**) ou au contraire utiliser le point le plus proche (**false**). Une interpolation linéaire donnera des textures plus douces mais pourra aussi donner des résultats légèrement plus troubles. De plus, cela prendra parfois plus de temps pour les afficher. Ce réglage influencera également l'affichage des sprites et des arrière-plans. La valeur par défaut est **false** (cette valeur peut être changée via les paramètres généraux de jeu).
texture_set_blending(blend) Indique si l'on doit utiliser la fonction de mélange de couleurs avec les valeurs de couleurs et alpha. En positionnant ce switch à **0** (*OFF*), cela peut rendre plus rapide l'exécution du jeu sur de vieilles machines dont le hardware est un peu dépassé. Ce réglage influence aussi l'affichage des sprites et des arrière-plans. La valeur par défaut est **true**.
texture_set_repeat(repeat) Indique si l'on doit répéter la texture. Ceci fonctionne de la manière suivante. Comme indiqué précédemment, les coordonnées de texture doivent être comprises entre **0** et **1**. Si une plus grande valeur que **1** est mentionnée, le reste ne sera pas par défaut dessiné. En réglant la valeur **repeat** à **true**, la texture sera répétée. Notez que les sprites et les arrière-plans sont toujours affichés sans possibilité de répétition. Aussi, une fois que vous avez affiché un sprite d'un arrière-plan donné, cette valeur sera remise à **false**.

Il existe deux autres fonctions qui seront utiles pas uniquement pour dessiner des textures. Habituellement, les primitives sont mélangées à l'aide d'un fondu avec l'arrière-plan en utilisant

la valeur **alpha**. Il vous est possible de préciser la manière dont cela doit se produire. En plus du mode normal, il est possible d'indiquer que la nouvelle couleur doit être ajoutée à celle existante ou encore soustraite de la couleur existante. Cela peut être utilisé pour créer par exemple des spots de lumière ou encore des ombres. Il est également possible de faire en sorte de prendre en compte au maximum la nouvelle couleur ou celle déjà existante. Cela permet d'éviter le phénomène d'effets de saturation qui survient lors de l'ajout de couleurs. Veuillez noter qu'a la fois les fonctions de soustraction et de maximalisation ne prennent pas pleinement en compte la valeur **alpha** (DirectX ne le permettant pas). Il est donc préférable de s'assurer que la zone en dehors soit noire. Il y a pour cela deux fonctions. La première vous donnera seulement quatre options qui ont déjà été décrites. La seconde fonction vous proposera davantage de possibilités. Il vous sera nécessaire d'expérimenter par vous-même en utilisant les différents paramètres. Si vous parvenez à bien utiliser ces fonctions, vous arriverez à créer par exemple des explosions intéressantes ou encore des effets de halo spectaculaires.

draw_set_blend_mode(mode) Indique si le mode de mélange de couleurs doit être utilisé. Les valeurs suivantes sont disponibles : `bm_normal`, `bm_add`, `bm_subtract` et `bm_max`. N'oubliez pas de remettre le mode à la valeur **normal** après utilisation de cette fonction car sinon, les autres sprites et même les arrière-plans seront affichés avec le nouveau mode de mélange de couleurs.
draw_set_blend_mode_ext(src,dest) Indique quel mode de mélange de couleurs est à utiliser pour à la fois la couleur source et celle de destination. La nouvelle couleur est pendant quelques temps fonction de la source et à d'autres moments fonction de la destination. Les facteurs de temps pourront être paramétrés avec cette fonction. Pour bien comprendre comment cela marche, la source et la destination ont à la fois un composant rouge, vert, bleu et alpha. Ainsi, la source est (Rs, Gs, Bs, As) et la destination est (Rd, Gd, Bd, Ad). Tous devront être dans la fourchette allant de **0** à **1**. Les facteurs de mélange de couleurs que vous pouvez choisir pour la source et la destination sont les suivants :

- bm_zero: Le facteur de mélange sera de (0, 0, 0, 0).
- bm_one: Le facteur de mélange sera de (1, 1, 1, 1).
- bm_src_color: Le facteur de mélange sera de (Rs, Gs, Bs, As).
- bm_inv_src_color: Le facteur de mélange sera de (1–Rs, 1–Gs, 1–Bs, 1–As).
- bm_src_alpha: Le facteur de mélange sera de (As, As, As, As).
- bm_inv_src_alpha: Le facteur de mélange sera de (1–As, 1–As, 1–As, 1–As).
- bm_dest_alpha: Le facteur de mélange sera de (Ad, Ad, Ad, Ad).

- bm_inv_dest_alpha: Le facteur de mélange sera de (1–Ad, 1–Ad, 1–Ad, 1–Ad).
- bm_dest_color: Le facteur de mélange sera de (Rd, Gd, Bd, Ad).
- bm_inv_dest_color: Le facteur de mélange sera de (1–Rd, 1–Gd, 1–Bd, 1–Ad).
- bm_src_alpha_sat: Le facteur de mélange sera de (f, f, f, 1); f = min(As, 1–Ad).

Par exemple, le mode normal de mélange de couleurs règle le mélange de couleurs de la source à **bm_src_alpha** et celui de la destination à **bm_inv_src_alpha**. N'oubliez pas de remettre le mode à la valeur **normal** après l'avoir utilisé car sinon, les autres sprites et même les arrière-plans seront affichés avec le nouveau mode de mélange de couleurs.

L'affichage de primitives à base de textures demande un peu de travail de votre part mais offre en contre-partie de grands résultats. Vous pourrez même les utiliser pour faire des jeux en 3D.

Dessin de Surfaces

Ces fonctionnalités ne sont disponibles que dans la version enregistrée de Game Maker.

Dans certains cas, vous souhaiterez ne pas dessiner directement à l'écran mais plutôt sur un canevas qui pourra ultérieurement être réutilisé pour dessiner d'autres choses sur l'écran. Ce type de canevas est appelé une **surface**. Par exemple, imaginons que vous souhaitiez que l'utilisateur puisse dessiner sur l'écran. Il ne devra pas pouvoir le faire directement sur l'écran (car le dessin sera supprimé à chaque nouveau step). Vous souhaiterez donc que le dessin soit fait sur une surface distincte qui sera ensuite copiée à l'écran à chaque step. Ou bien encore vous voudrez utiliser une texture qui devra changer à intervalles réguliers.

L'utilisation de surfaces vous permettra de réaliser tout ceci et de plus, c'est assez simple à utiliser. Vous devrez créer en premier lieu une surface. Ensuite, vous indiquerez les opérations de dessin qui devront s'afficher sur cette surface. A partir de ce moment là, toutes les fonctions de dessin opéreront sur la surface en question. Une fois que vous en aurez terminé, vous réinitialiserez la cible de dessin et les opérations de dessin ultérieurs s'afficheront de nouveau sur cet écran. Vous pouvez dessiner sur la surface de l'écran de différentes manières ou encore utiliser la surface comme une texture. A ce sujet, il y a certaines choses pour lesquelles vous devrez être prudent. Veuillez consulter les remarques à la fin de cette section.

Les fonctions suivantes existent et concernent les surfaces.

`surface_create(w,h)` Crée une surface de largeur et hauteur indiquées. Retourne l'**ID** de la surface qui devra être utilisé dans tous les appels ultérieurs de fonctions. Veuillez prendre note que la surface ne sera pas effacée. Ceci incombe à l'utilisateur de le faire délibérément (définissez-la comme une cible puis appelez la fonction d'effacement appropriée.)
`surface_free(id)` Libère la mémoire utilisée par la surface.
`surface_exists(id)` Retourne si la surface identifiée par l'**ID** indiqué existe déjà.

`surface_get_width(id)` Retourne la largeur de la surface.
`surface_get_height(id)` Retourne la hauteur de la surface.
`surface_get_texture(id)` Retourne la texture correspondant à la surface. Ceci peut être utilisé pour afficher des objets texturés avec l'image de la surface.

surface_set_target(id) Définit la surface indiquée comme étant la cible d'affichage. Ainsi, toutes les opérations futures d'affichages opéreront sur

cette surface. Cette commande réinitialise la projection afin de couvrir simplement la surface.
surface_reset_target() Réinitialise la cible d'affichage sur l'écran de destination courant.

surface_getpixel(id,x,y) Retourne la couleur du pixel correspondant à la position **(x,y)** de la surface. A utiliser avec précaution car cela n'est pas très rapide.
surface_save(id,fname) Sauvegarde la surface en tant qu'image **BMP** dans le fichier de nom **filename**. Utile pour faire une copie d'écran de la surface.
surface_save_part(id,fname,x,y,w,h) Sauvegarde une partie de la surface dans le fichier de nom **filename**.

draw_surface(id,x,y) Affiche la surface à la position **(x,y)**. (sans mélange de couleur (pas de fondu) ni transparence **alpha**).
draw_surface_stretched(id,x,y,w,h) Affiche la surface étendue à la région indiquée.
draw_surface_tiled(id,x,y) Affiche la surface sous forme de tuiles (tiles) de manière à ce qu'elle remplisse entièrement la room.
draw_surface_part(id,left,top,width,height,x,y) Affiche la partie indiquée de la surface avec origine en position **(x,y)**.
draw_surface_ext(id,x,y,xscale,yscale,rot,color,alpha) Affiche la surface avec mise à l'échelle et orientation tout en utilisant une couleur de mélange (utilisez **c_white** pour ne pas avoir de couleur de fondu) et une transparence **alpha** (**0** à **1**).
draw_surface_stretched_ext(id,x,y,w,h,color,alpha) Affiche la surface étendue à la région indiquée. La couleur est celle de la couleur de fondu et **alpha** correspond aux paramètres de transparence.
draw_surface_tiled_ext(id,x,y,xscale,yscale,color,alpha) Affiche la surface sous forme de tuiles (tiled) de façon à remplir complètement la room mais désormais avec une mise à l'échelle à l'aide des facteurs **xscale** et **yscale** et un paramétrage de la couleur et de la transparence.
draw_surface_part_ext(id,left,top,width,height,x,y,xscale,yscale,color,alpha) Affiche la partie indiquée de la surface avec origine en position **(x,y)** mais maintenant avec une mise à l'échelle à l'aide des facteurs **xscale** et **yscale** et un paramétrage de la couleur et de la transparence.

draw_surface_general(id,left,top,width,height,x,y,xscale,yscale,rot,c1,c2,c3,c4,alpha) La fonction d'affichage la plus couramment utilisée. Elle affiche la partie indiquée de la surface avec origine en position **(x,y)** mais désormais avec une mise à l'échelle à l'aide des facteurs **xscale** et **yscale**, un angle de rotation **rot**, une couleur pour chacun des quatre vertices (supérieur gauche, supérieur droit, inférieur droit et inférieur gauche) et une valeur de transparence **alpha**.

surface_copy(destination,x,y,source) Effectue une copie de la surface source à la position **(x,y)** dans la surface de destination. (sans aucune possibilité de fondu ou de mélange de couleur).
surface_copy_part(destination,x,y,source,xs,ys,ws,hs) Copie la partie indiquée de la surface source à la position **(x,y)** dans la surface de destination. (sans fondu ni mélange de couleur).

Veuillez noter qu'il n'existe pas de fonction permettant de copier une partie de l'écran dans une surface (ceci est impossible en raison d'éventuelles différences de format entre l'écran et les surfaces). Si nécessaire, vous devrez définir une nouvelle surface comme destination de rendu puis réafficher la room. Vous pourrez ensuite utiliser les routines de copie de surfaces pour obtenir la partie qui vous intéresse.

Notez qu'il est cependant possible de créer des sprites et des arrière-plans à partir de surfaces. Pour plus d'informations sur ce sujet, veuillez consulter la section sur la modification des ressources (changing resources).

Certaines précautions doivent être prises lors de l'utilisation de ces fonctions. En particulier, faîtes bien attention aux choses suivantes :

- Vous ne devrez jamais modifier la cible d'affichage alors que vous dessinez actuellement sur l'écran soit en d'autres termes, ne jamais utiliser cette cible dans les événements d'affichages (drawing events). Cela pourrait poser de sérieux problèmes avec la projection et le viewport.
- Les surfaces ne fonctionnent pas correctement dans le mode 3D. Vous ne pouvez les utiliser alors que vous n'êtes pas dans le mode 3D (en invoquant la fonction **d3d_end()** avant d'utiliser les surfaces mais une fois que vous rentrerez dans le mode 3D, les surfaces seront détruites.
- Pour des raisons de vitesse, la surface est conservée uniquement en mémoire vidéo. Aussi, vous pourriez perdre la surface si par exemple, vous modifiez la résolution d'écran ou encore si votre effaceur d'écrans se met en fonction.

- Les surfaces ne seront pas sauvegardées lorsque vous effectuerez une sauvegarde de votre jeu.

Tuiles graphiques (Tiles)

Comme vous le savez sans doute déjà, vous avez la possibilité d'ajouter des tuiles graphiques (tiles) dans les rooms. Une tuile (tile) est une partie d'une ressource d'arrière-plan. Les tuiles ne sont juste que des images visibles. Elles ne réagissent pas aux événements et ne génèrent pas de collisions. Elles présentent l'avantage d'être affichées beaucoup plus rapidement que les objets. Toute chose ne nécessitant pas d'événements de collisions devrait être gérée à l'aide de tuiles. Ainsi, on utilisera de préférence une tuile pour afficher de jolis graphiques alors qu'un simple objet sera utilisé pour générer les événements de collision.

Vous disposez de beaucoup plus de contrôles sur les tuiles que vous ne pourriez le penser. Vous pouvez en ajouter lors de la conception de la room mais il est également possible d'en rajouter lors de l'exécution du jeu. Il vous sera possible de changer leur position et même d'effectuer sur elles une mise à l'échelle ou encore les rendre partiellement transparentes. Une tuile présente les propriétés suivantes :

- **background**. La ressource d'arrière-plan dans laquelle se trouve la tuile.
- **left, top, width, height**. La partie d'arrière-plan qui est utilisée.
- **x,y**. La position du coin supérieur gauche de la tuile dans la room.
- **depth**. La profondeur de la tuile. Vous pourrez choisir une profondeur quelconque, permettant ainsi de faire apparaître les tuiles entre les instances d'objets.
- **visible**. Indique si la tuile est visible.
- **xscale, yscale**. Chaque tuile peut être dessinée à une certaine échelle (la valeur par défaut est de **1**).
- **blend**. Une couleur de fondu utilisée lors de l'affichage de la tuile.
- **alpha**. Une valeur **alpha** indiquant la transparence de la tuile. **1** = aucune transparence, **0** = transparence totale.

Pour modifier les propriétés d'une tuile particulière, vous devez connaître son **ID**. Lors des opérations d'ajout de tuiles quand vous concevez les rooms, l'ID est affiché dans la barre d'information en bas de l'écran. Il existe aussi une fonction qui permet de déterminer l'ID d'une tuile se trouvant à une certaine position.

Les fonctions suivantes existent et concernent les tuiles :

tile_add(background,left,top,width,height,x,y,depth) Ajoute une nouvelle tuile dans la room en utilisant les valeurs indiquées (voir la signification

ci-dessus). La fonction retourne l'ID de la tuile qui devra être utilisé dans toutes les fonctions opérant sur cette tuile.

tile_delete(id) Efface la tuile correspondant à l'ID donné en paramètre.
tile_exists(id) Indique si la tuile d'ID **id** existe.

tile_get_x(id) Retourne la position en **x** de la tuile d'ID **id**.
tile_get_y(id) Retourne la position en **y** de la tuile d'ID **id**.
tile_get_left(id) Retourne la valeur **left** (gauche) de la tuile d'ID **id**.
tile_get_top(id) Retourne la valeur **top** (haut) de la tuile d'ID **id**.
tile_get_width(id) Retourne la largeur (**width**) de la tuile d'ID **id**.
tile_get_height(id) Retourne la hauteur (**height**) de la tuile d'ID **id**.
tile_get_depth(id) Retourne la profondeur (**depth**) de la tuile d'ID **id**.
tile_get_visible(id) Indique si la tuile d'ID **id** est visible ou pas.
tile_get_xscale(id) Retourne le facteur d'échelle **xscale** de la tuile d'ID **id**.
tile_get_yscale(id) Retourne le facteur d'échelle **yscale** de la tuile d'ID **id**.
tile_get_background(id) Retourne la valeur d'arrière-plan de la tuile d'ID **id**.
tile_get_blend(id) Retourne la couleur de fondu de la tuile d'ID **id**.
tile_get_alpha(id) Retourne la valeur **alpha** de la tuile d'ID **id**.

tile_set_position(id,x,y) Fixe la position de la tuile d'ID **id**.
tile_set_region(id,left,right,width,height) Fixe la région de la tuile d'ID **id** dans son propre arrière-plan.
tile_set_background(id,background) Fixe l'arrière-plan pour la tuile d'ID **id**.
tile_set_visible(id,visible) Indique si la tuile d'ID **id** doit être visible ou pas.
tile_set_depth(id,depth) Fixe la profondeur de la tuile d'ID **id**.
tile_set_scale(id,xscale,yscale) Fixe l'échelle de la tuile d'ID **id**.
tile_set_blend(id,color) Fixe la couleur de fondu de la tuile d'ID id.
Fonction uniquement disponible dans la version enregistrée !
tile_set_alpha(id,alpha) Fixe la valeur **alpha** de la tuile d'ID **id**.

Les fonctions suivantes concernent les couches de tuiles (layers of tiles) qui sont simplement des collections de tuiles ayant la même profondeur.

tile_layer_hide(depth) Cache toutes les tuiles de la couche de profondeur indiquée.

tile_layer_show(depth) Affiche toutes les tuiles de la couche de profondeur indiquée.

tile_layer_delete(depth) Efface toutes les tuiles de la couche de profondeur indiquée.

tile_layer_shift(depth,x,y) Décale toutes les tuiles de la couche de profondeur indiquée en utilisant le vecteur **x,y**. Peut être utilisé pour créer un scrolling des couches de tuiles.

tile_layer_find(depth,x,y) Retourne l'ID de la tuile de profondeur indiquée **depth** à la position **(x,y)**. Si aucune tuile n'existe à cette position, la valeur **-1** sera retournée. S'il y a plusieurs tuiles de profondeur indiquée **depth** à cette position, la première tuile sera retournée.

tile_layer_delete_at(depth,x,y) Efface la tuile de la profondeur indiquée **depth** à la position **(x,y)**. Dans le cas où plusieurs tuiles existent à la profondeur indiquée **depth** et à cette position, elles seront toutes effacées.

tile_layer_depth(depth,newdepth) Change la profondeur de toutes les tuiles de profondeur **depth** à la nouvelle profondeur **newdepth**. Avec cette fonction, vous pouvez déplacer tout un ensemble de couches de tuiles vers une nouvelle profondeur.

L'Affichage

La zone d'affichage est délimitée par la surface du moniteur. Ce dernier possède une taille (habituellement 1024x768 ou 1280x1024), une profondeur de couleur, c'est à dire le nombre de bits utilisés pour représenter un pixel (généralement 16 = **High Color** ou 32 = **Full Color**) et une fréquence de rafraîchissement, c'est à dire le nombre de fois par seconde pendant lequel l'affichage est réactualisé à l'écran (généralement compris entre 60 et 120). Ces réglages peuvent être modifiés à l'aide des propriétés d'affichage. Cependant pour les jeux et plus particulièrement lorsque ceux-ci tournent en mode plein-écran, il est important de pouvoir changer ces paramètres. Tous ces réglages peuvent être initialisés grâce aux réglages proposés dans **Game Settings**. Pendant le jeu, les fonctions suivantes peuvent être employées. Veuillez noter que la modification de ces réglages pendant le jeu entraînera un certain délai d'attente car il est nécessaire alors de recalculer et/ou de réafficher toutes les choses présentes à l'écran. ***Les fonctions de paramétrage du mode d'affichage ne sont disponibles que dans la version enregistrée.***

`display_get_width()` Retourne la largeur de l'affichage en pixels.

`display_get_height()` Retourne la hauteur de l'affichage en pixels.

`display_get_colordepth()` Retourne la profondeur de la couleur en bits.

`display_get_frequency()` Retourne la fréquence de rafraîchissement de l'affichage.

`display_set_size(w,h)` Règle la largeur et la hauteur de l'affichage en pixels. Retourne le résultat de la commande en cas de succès (seules certaines combinaisons sont autorisées).

`display_set_colordepth(coldepth)` Règle la profondeur de la couleur. En général, seules les valeurs **16** et **32** seront autorisées. Retourne une valeur en cas de succès.

`display_set_frequency(frequency)` Règle la fréquence de rafraîchissement de l'affichage. Très peu de fréquences peuvent être utilisées. Généralement, vous indiquerez une valeur de **60** qui correspond également à la vitesse de la room afin d'obtenir une animation de **60** frames par seconde. Retourne une valeur en cas de succès.

`display_set_all(w,h,frequency,coldepth)` Règle tous les paramètres en une seule commande. Utilisez **-1** pour les valeurs que vous ne souhaitez pas changer. Retourne une valeur en cas de succès.

`display_test_all(w,h,frequency,coldepth)` Teste si les paramètres indiqués sont valides. Cela ne modifie en rien les réglages. Utilisez **-1** pour les valeurs que vous ne désirez pas tester. Retourne si les paramètres sont autorisés.

display_reset() Réinitialise les réglages d'affichage aux valeurs utilisées lors du démarrage du programme.

Parfois, il est utile de connaître la position de la souris ou encore de modifier sa position. Les fonctions suivantes sont dédiées à cet usage :

display_mouse_get_x() Retourne la valeur en **x** de la souris.
display_mouse_get_y() Retourne la valeur en **y** de la souris.
display_mouse_set(x,y) Règle la position de la souris avec les valeurs indiquées.

La Fenêtre

Les jeux actuels se déroulent habituellement dans des fenêtres. La fenêtre de jeu possède un certain nombre de propriétés, comme la présence ou non d'une bordure ou encore un affichage en mode plein écran, etc. Normalement, ces propriétés seront réglées à partir de **Game Settings**. Mais vous pouvez les changer également pendant le jeu. A cet effet, les fonctions suivantes existent :

window_set_visible(visible) Indique si la fenêtre doit être visible ou non. Habituellement, vous souhaiterez que la fenêtre reste visible durant tout le jeu. Le programme ne recevra pas les événements clavier si la fenêtre est invisible.
window_get_visible() Retourne si la fenêtre de jeu est visible.
window_set_fullscreen(full) Indique que la fenêtre doit être affichée en mode plein écran.
window_get_fullscreen() Retourne si la fenêtre est affichée en plein écran.
window_set_showborder(show) Indique que l'on doit afficher la bordure autour de la fenêtre (la bordure ne sera jamais affichée en mode plein écran).
window_get_showborder() Retourne si la bordure autour de la fenêtre est affichée en mode fenêtré.
window_set_showicons(show) Indique que les icônes de bordure de fenêtre (icônification, maximisation et fermeture) doivent être affichées (ces icônes ne seront jamais affichées en mode plein écran).
window_get_showicons() Retourne si les icônes de bordure de fenêtre sont affichées en mode fenêtré.
window_set_stayontop(stay) Indique que la fenêtre doit toujours être affichée par-dessus les autres fenêtres.
window_get_stayontop() Retourne si la fenêtre est toujours affichée par-dessus les autres fenêtres.
window_set_sizeable(sizeable) Indique que la fenêtre est redimensionnable par le joueur (le joueur ne peut le faire que si la bordure de fenêtre est affichée et que la fenêtre ne soit pas en mode plein écran).
window_get_sizeable() Retourne si la fenêtre est redimensionnable par le joueur.
window_set_caption(caption) Renseigne la chaîne contenant le nom de la fenêtre. Habituellement, vous indiquerez ce nom lors de la conception de la room, nom qui pourra être changé grâce à la variable **room_caption**. Aussi, cette fonction ne sera pas généralement utile sauf si vous dessinez la room vous-même

et ne laissez donc pas *Game Maker* le faire pour vous. Le titre de la fenêtre n'est visible que si celle-ci possède une bordure et si on n'est pas en mode plein écran.
window_get_caption() Retourne le titre de la fenêtre.
window_set_cursor(curs) Indique le curseur de la souris à utiliser dans la fenêtre. Il vous est possible d'utiliser les constantes suivantes :

cr_default
cr_none
cr_arrow
cr_cross
cr_beam
cr_size_nesw
cr_size_ns
cr_size_nwse
cr_size_we
cr_uparrow
cr_hourglass
cr_drag
cr_nodrop
cr_hsplit
cr_vsplit
cr_multidrag
cr_sqlwait
cr_no
cr_appstart
cr_help
cr_handpoint
cr_size_all

En particulier, pour cacher le curseur de la souris, utilisez la valeur `cr_none`.
window_get_cursor() Retourne le curseur utilisé dans la fenêtre.
window_set_color(color) Fixe la couleur de la partie de la fenêtre non utilisée pour afficher la room.
window_get_color() Retourne la couleur de la fenêtre.
window_set_region_scale(scale,adaptwindow) Si la fenêtre est plus large que la room actuelle, la room sera normalement affichée dans une région centrée dans la fenêtre. Il est possible d'indiquer que la room doit être mise à l'échelle afin de remplir entièrement la fenêtre ou seulement une partie de celle-ci. Une valeur de 1 signifie pas de mise à l'échelle. Si vous mentionnez la valeur 0, la région

sera mise à l'échelle afin de remplir complètement la fenêtre. Si vous indiquez une valeur négative, la room sera mise à l'échelle à une taille maximale à l'intérieur de la fenêtre tout en maintenant le ratio d'aspect (c'est ce que vous souhaiterez le plus souvent). `adaptwindow` indique si la taille de la fenêtre doit être adaptée dans le cas où la room ne tiendrait pas dans la fenêtre après mise à l'échelle. Le redimensionnement de la taille de la fenêtre ne sera possible que si le facteur d'échelle est positif.

`window_get_region_scale()` Retourne le facteur d'échelle pour l'affichage de la région.

La fenêtre a une position sur l'écran et une taille (lorsque nous parlons de position et de taille, nous voulons toujours indiquer la partie de la fenêtre sans les bordures). Vous pouvez modifier ceci bien que ce ne soit pas facile et souhaitable de le faire durant le jeu. Habituellement, ces paramètres sont déterminés automatiquement ou encore par le joueur. Les fonctions suivantes peuvent être employées pour modifier la position et la taille de la fenêtre. Veuillez noter que ces fonctions concernent uniquement le mode fenêtré. Si la fenêtre est en mode plein écran, on pourra toujours utiliser ces paramètres mais ils ne prendront effet que lorsque le mode plein écran sera désactivé.

`window_set_position(x,y)` Règle la position de la fenêtre (partie de la fenêtre sans bordure) à la position indiquée.

`window_set_size(w,h)` Règle la taille de la fenêtre (partie de la fenêtre sans bordure) selon la taille mentionnée. Si la taille indiquée est trop petite pour remplir la région d'affichage, elle sera élargie afin de remplir totalement cette région.

`window_set_rectangle(x,y,w,h)` Règle la position et la taille de la fenêtre rectangulaire. (effectue les 2 réglages précédents en une seule opération).

`window_center()` Centre la fenêtre sur l'écran.

`window_default()` Donne à la fenêtre la taille et la position (centrée) par défaut sur l'écran.

`window_get_x()` Retourne la valeur actuelle en x de la fenêtre.

`window_get_y()` Retourne la valeur actuelle en y de la fenêtre.

`window_get_width()` Retourne la largeur actuelle de la fenêtre.

`window_get_height()` Retourne la hauteur courante de la fenêtre.

Pour rappel, vous n'utiliserez probablement jamais aucune de ces fonctions de positionnement de fenêtre car *Game Maker* effectue ce travail de manière automatique.

Dans de rares cas, vous souhaiterez connaître la position de la souris par rapport à la fenêtre courante (normalement, vous utiliserez souvent la position de la souris par rapport à la room ou encore selon une vue). Les fonctions suivantes existent si besoin.

window_mouse_get_x() Retourne la valeur en x de la souris par rapport à la fenêtre.

window_mouse_get_y() Retourne la valeur en y de la souris par rapport à la fenêtre.

window_mouse_set(x,y) Règle la position de la souris par rapport à la fenêtre avec les valeurs indiquées.

Les Vues

Comme vous le savez déjà, vous pouvez définir jusqu'à huit vues différentes lors de la création de rooms. Une vue est définie par sa zone d'affichage dans la room et son viewport à l'écran (ou pour être plus précis dans la zone d'affichage de la fenêtre). En utilisant des vues, vous pouvez afficher différentes parties de la room à différents endroits de l'écran. Ainsi, vous pourrez être sûr qu'un objet particulier sera toujours visible.

Vous pouvez contrôler les vues en utilisant du code de programmation. Vous pouvez rendre les vues visibles ou invisibles et modifier l'endroit, la taille des vues dans la zone d'affichage ou encore la position et la taille de la vue dans la room (ce qui peut être particulièrement utile lorsque vous indiquez qu'aucun objet ne doit être visible). Vous avez la possibilité de changer la taille de la bordure horizontale et verticale autour de l'objet visible et pouvez aussi indiquer quel objet doit rester visible dans les vues. Ce dernier point est très important si l'objet principal doit changer pendant le jeu. Par exemple, vous pourrez modifier l'objet du personnage principal en fonction de son statut actuel. Malheureusement, cela signifiera aussi que l'objet ne sera plus visible. Pour remédier à cela, on ajoutera une ligne de code dans l'événement de création de tous les objets principaux possibles (en supposant que cela doit survenir dans la première vue) :

```
{
  view_object[0] = object_index;
}
```

Les variables suivantes existent et ont une influence sur la vue. Toutes, excepté les deux premières, sont des tableaux d'indice allant de **0** (la première vue) à **7** (la dernière vue).

view_enabled Indique si les vues sont disponibles ou pas.
view_current* La vue actuellement affichée (**0** à **7**). A utiliser uniquement dans l'événement d'affichage (*drawing event*). Vous pourrez par exemple vérifier cette variable pour afficher certaines choses dans une certaine vue. Cette variable ne peut être modifiée.
view_visible[0..7] Indique si une vue particulière est visible à l'écran.

view_xview[0..7] Position en **X** de la vue dans la room.
view_yview[0..7] Position en **Y** de la vue dans la room.
view_wview[0..7] Largeur de la vue dans la room.

view_hview[0..7] Hauteur de la vue dans la room.
view_xport[0..7] Position en **X** du viewport dans la zone d'affichage.
view_yport[0..7] Position en **Y** du viewport dans la zone d'affichage.
view_wport[0..7] Largeur du viewport dans la zone d'affichage.
view_hport[0..7] Hauteur du viewport dans la zone d'affichage.
view_angle[0..7] Angle de rotation utilisé par la vue dans la room (dans le sens contraires des aiguilles d'une montre et en degrés).
view_hborder[0..7] Taille de la bordure horizontale autour de l'objet visible (en pixels).
view_vborder[0..7] Taille de la bordure verticale autour de l'objet visible (en pixels).
view_hspeed[0..7] Vitesse horizontale maximale de la vue.
view_vspeed[0..7] Vitesse verticale maximale de la vue.
view_object[0..7] Objet dont l'instance doit rester en permanence visible dans la vue. Dans le cas de multiples instances de l'objet, seul la première sera suivie. Vous pouvez également assigner l'**ID** d'une instance à cette variable. Ainsi, cette dernière instance sera suivie.

Veuillez bien noter que la taille de l'image sur l'écran est déterminée à partir des vues visibles au début de la room. Si vous deviez changer les vues pendant le jeu, il est possible que celles-ci ne tiennent plus entièrement dans l'écran. La taille de l'écran ne sera pas adaptée automatiquement. Aussi, ce sera à vous d'adapter la taille de l'écran en utilisant les fonctions suivantes :

window_set_region_size(w,h,adaptwindow) Règle la largeur et la hauteur de la surface d'affichage de la fenêtre. `adaptwindow` indique si la taille de la fenêtre doit être ajustée dans le cas où la région ne pourrait pas être totalement affichée. La taille de la fenêtre devra toujours être ajustée si vous utilisez des fonctions de mise à l'échelle (se reporter à la fonction `window_set_region_scale()`.)
window_get_region_width() Retourne la largeur actuelle de la zone d'affichage.
window_get_region_height() Retourne la hauteur actuelle de la zone d'affichage.

Dans un jeu, vous aurez souvent besoin de connaître la position du curseur de la souris. Habituellement, vous utiliserez à cet effet les variables `mouse_x` et `mouse_y`. S'il y a plusieurs vues, ces variables donneront la position de la souris en accord avec la vue où se trouve actuellement la souris. Parfois cependant, vous souhaiterez obtenir la position de la souris

par rapport à une vue particulière, même lorsque la souris se trouve en dehors de cette vue. Les fonctions suivantes seront alors à utiliser.

window_view_mouse_get_x(id) Retourne l'abscisse **x** de la souris en accord avec la vue d'index **ID**.

window_view_mouse_get_y(id) Retourne l'ordonnée **y** de la souris en accord avec la vue d'index **ID**.

window_view_mouse_set(id,x,y) Règle la position de la souris en accord avec la vue d'index **ID**.

window_views_mouse_get_x() Retourne l'abscisse **x** de la souris en accord avec la vue où se trouve la souris (identique à `mouse_x`).

window_views_mouse_get_y() Retourne l'abscisse **x** de la souris en accord avec la vue où se trouve la souris (identique à `mouse_y`).

window_views_mouse_set(x,y) Règle la position de la souris en accord avec la première vue visible.

Les Transitions

Comme vous le savez déjà, lorsque vous vous déplacez d'une room à une autre, il vous est possible de choisir un effet de transition. Il est également possible d'effectuer une transition pour la prochaine frame sans changer de room en utilisant la variable `transition_kind`. La transition sera effectuée en assignant une valeur allant de **1** à **13** correspondant à la transition souhaitée (ce sont les mêmes transitions que vous pouvez sélectionner pour les rooms). Une valeur de **0** signifie aucune transition. Les transitions ne sont visibles que lors de l'affichage de la prochaine frame.

`transition_kind` Indique la transition pour la prochaine frame. Vous pouvez utiliser les valeurs suivantes

0 = aucun effet
1 = effet de rideau venant de la gauche
2 = effet de rideau venant de la droite
3 = effet de rideau venant du haut
4 = effet de rideau venant du bas
5 = effet de transition venant du centre
6 = Décalage de l'écran à partir de la gauche
7 = Décalage de l'écran à partir de la droite
8 = Décalage de l'écran à partir du haut
9 = Décalage de l'écran à partir du bas
10 = Effet entrelacé venant de la gauche
11 = Effet entrelacé venant de la droite
12 = Effet entrelacé venant du haut
13 = Effet entrelacé venant du bas

Veuillez noter qu'il est assez facile de créer vos propres effets de transitions en utilisant les diverses fonctions d'affichage. Par exemple, pour créer un effet de fondu au noir, vous pouvez dessiner un grand rectangle remplissant la room avec une valeur **alpha** allant en augmentant. Ou encore vous pouvez modifier la position et la taille de la vue afin que la room se déplace hors de la zone visible.

Réaffichage de l'Ecran

La room est redessinée habituellement sur l'écran à la fin de chaque step. Mais dans de rares circonstances, vous devrez redessiner la room à d'autres moments. Cela arrive par exemple lorsque votre programme prend le contrôle total des opérations. Ainsi, avant une mise en veille prolongée, il sera sans doute nécessaire de redessiner l'écran. De plus, si votre programme affiche un message et demande au joueur de taper sur une touche, il sera indispensable de rafraîchir l'affichage de l'écran entre ces 2 opérations. Il existe deux routines différentes pour effectuer le rafraîchissement de l'affichage de l'écran.

`screen_redraw()` Réaffiche la room en appelant tous les événements d'affichage (*draw events*).
`screen_refresh()` Rafraîchit l'écran en utilisant l'image actuelle de la room (n'invoque pas les événements d'affichage).

Pour bien comprendre la deuxième fonction, il vous est nécessaire de connaître davantage le fonctionnement interne de l'affichage sur écran. Intrinsèquement, tous les affichages se déroulent sur une seule image. Cette image n'est pas visible à l'écran. Uniquement à la fin de chaque step, après que toutes les opérations d'affichage soient terminées, l'image affichée à l'écran est remplacée par cette image interne (c'est ce que l'on appelle le double buffering). La première fonction redessine l'image interne et rafraîchit l'image à l'écran. La seconde fonction ne rafraîchit que l'image affichée à l'écran.

Maintenant, vous devriez mieux comprendre pourquoi vous ne pouvez pas utiliser des actions ou des fonctions d'affichage ailleurs que dans des événements d'affichage (drawing events). Les autres événements affichent ou dessinent des choses dans l'image interne mais ces dernières ne seront pas visibles à l'écran. Et lorsque les événements d'affichage sont exécutés, la room d'arrière-plan est affichée en premier plan, effaçant tout ce qui aurait pu être dessiné sur l'image interne. Mais quand vous utilisez la commande `screen_refresh()` après avoir dessiné quelque chose, l'image réactualisée sera visible sur l'écran à ce moment là. Ainsi, par exemple, un script pourra afficher sur l'écran du texte, appeler la fonction de rafraîchissement puis attendre que le joueur appuie sur une touche. Le code ci-dessous illustre cet exemple.

```
{
  draw_text(screen_width/2,100,'Press any key to continue.');
  screen_refresh();
  keyboard_wait();
}
```

Veuillez bien comprendre que lorsque vous dessinez ailleurs que dans un événement d'affichage (drawing event), vous dessinez en fait simplement sur l'image et non pas dans une vue ! (bien que les coordonnées utilisées soient les mêmes comme s'il n'y avait pas de vue). Soyez prudent lors de l'utilisation de cette technique. Assurez-vous que vous avez bien assimilé les notions d'affichage et ayez en tête que le rafraîchissement d'écran prend du temps.

Lorsque vous dessinez par vous-même la room, cela peut être utile de **NE PAS** laisser *Game Maker* le faire automatiquement à votre place. Par exemple, vous pourrez désirer dessiner la room tous les 5 steps. Vous utiliserez alors les fonctions suivantes :

`set_automatic_draw(value)` Indique si l'affichage de la room doit se faire en automatique (valeur **true** par défaut) ou non (**false**).

Enfin, une fonction existe avec laquelle vous pourrez synchroniser l'affichage avec la fréquence de rafraîchissement de votre moniteur :

`set_synchronization(value)` Indique que la synchronisation de l'affichage doit se faire avec la fréquence de rafraîchissement du moniteur.

Vous avez la possibilité de forcer un temps d'attente avec la prochaine synchronisation verticale en utilisant la fonction suivante :

`screen_wait_vsync()` Attends la prochaine synchronisation verticale du moniteur.

Son et musique

Les sons jouent un rôle très important dans les jeux sur ordinateurs. Les sons sont incorporés dans les jeux sous forme de ressources sonores. Il est important de s'assurer que les noms utilisés soient des noms de variables valides. Comme vous l'avez déjà vu, vous pouvez indiquer quatre types différents de sons: les sons normaux, les musiques d'arrière-plan, les sons 3D et enfin les sons devant être joués à l'aide d'un lecteur multimédia (media player).

Les sons normaux sont utilisés pour les sons avec effets. En général, on utilise des fichiers **wave**. La plupart d'entre eux peuvent être joués simultanément (y compris les instances multiples d'un même son). Vous pouvez leur appliquer toutes sortes d'effets.

Les musiques d'arrière-plan concernent généralement des fichiers **midi** mais on utilise aussi parfois des fichiers **wave**. Des effets sonores peuvent leur être appliqués. La seule différence par rapport aux sons normaux est qu'une seule musique d'arrière-plan peut être jouée à un moment donné. Si vous jouez une autre musique, la musique en cours de lecture sera interrompue.

Les sons 3D vous permettent des effets sonores de type **3D** comme décrit ci-dessous. Ce sont des sons monophoniques (**wave** ou **midi**).

Enfin, si vous souhaitez utiliser un autre type de sons, en particulier le type **mp3**, sachez que ces sons ne pourront être joués à l'aide de **DirectX**. Il faudra alors utiliser un lecteur multimédia pour les écouter. Ceci rend leur usage plus limité. Un seul son peut être joué à la fois. Aucun effet ne peut leur être appliqué (même pas un changement de volume !) et le timing est très pauvre comme par exemple pour les sons en boucle. Il peut également y avoir des pauses de courte durée lors de l'écoute de ces sons. Vous êtes fortement encouragés de ne pas les utiliser (de plus, certains ordinateurs pourraient ne pas les supporter !)

Vous trouverez de l'information sur les sons et les musiques dans les pages suivantes :

Fonctions de base sur les sons

Il existe cinq fonctions de base ayant trait aux sons, deux pour lire un son, une pour vérifier si un son est actuellement en cours de lecture et deux autres pour arrêter les sons. La plupart d'entres elles prennent comme argument l'index du son. Le nom du son représente son index. Mais il vous est possible de stocker cet index dans une variable puis de l'utiliser.

`sound_play(index)` Joue une seule fois le son indiqué. Si le son est de type musique d'arrière-plan, la musique d'arrière-plan courante sera interrompue.
`sound_loop(index)` Joue en continu le son indiqué. Si le son est de type arrière-plan, la musique d'arrière-plan courante sera interrompue.
`sound_stop(index)` Arrête le son indiqué. Dans le cas où plusieurs sons de même index sont en cours de lecture, tous les sons seront stoppés.
`sound_stop_all()` Arrête tous les sons.
`sound_isplaying(index)` Retourne si le son indiqué (ou une copie de ce son) est en cours de lecture. Veuillez noter que cette fonction retourne **true** lorsque le son est joué à travers les haut-parleurs. Après avoir appelé la fonction pour jouer un son, ce dernier n'atteint pas immédiatement les haut-parleurs. Aussi, la fonction peut retourner **false** comme valeur pendant un certain laps de temps. De façon similaire, lorsque vous arrêtez un son, vous pouvez encore l'entendre pendant un certain temps (à cause de l'écho) et la fonction retournera alors **true**.

Il est possible d'utiliser des effets sonores plus évolués. En particulier, vous pourrez changer le volume et le pan, c'est à dire si le son doit arriver du haut-parleur de gauche ou de droite. Dans tous les cas, le volume peut seulement être réduit. Ces fonctions ne fonctionnent pas pour les fichiers joués par le lecteur multimédia (comme les fichiers mp3).

`sound_volume(index,value)` Change le volume du son indiqué (**0** = faible, **1** = fort).
`sound_global_volume(value)` Change le volume global de tous les sons (**0** = faible, **1** = fort).
`sound_fade(index,value,time)` Change le volume du son indiqué avec une nouvelle valeur (**0** = faible, **1** = fort) pendant la période de temps mentionnée (en millisecondes). Cela peut être utile pour créer un effet de **fading out** ou **in** sur une musique.
`sound_pan(index,value)` Change le pan du son indiqué (**-1** = gauche,

0 = centre, **1** = droit).

sound_background_tempo(factor) Change le tempo de la musique

d'arrière-plan (si c'est un fichier midi). `factor` indique le facteur avec lequel le tempo doit être multiplié. Une valeur de **1** correspond à un tempo normal. De plus grandes valeurs indiqueront un tempo plus rapide alors que de plus petites valeurs signifieront un tempo plus lent. Doit être compris entre **0.01** et **100**.

En plus des fichiers **midi** et **wave** (et des fichiers **mp3**), il existe un quatrième type de fichier musical : les fichiers de musique directe (*direct music files*). Ceux-ci ont l'extension **.sgt**. Ces fichiers font référence souvent à d'autres fichiers décrivant la bande (band) ou l'information de style. Pour trouver ces fichiers, le système de sons doit connaître précisément où ces sons sont localisés. A cet effet, vous pourrez utiliser les fonctions suivantes pour déterminer le répertoire de recherche de ces fichiers. Veuillez noter que c'est à vous d'ajouter ces fichiers. *Game Maker* n'inclue pas automatiquement ce genre de fichiers supplémentaires.

sound_set_search_directory(dir) Détermine le répertoire dans lequel les fichiers de type musique directe (*direct music files*) seront cherchés. La chaîne **dir** ne doit pas comprendre de **backslash** (****) à la fin.

Effets spéciaux sur les sons

Cette fonctionnalité est uniquement disponible dans la version enregistrée de Game Maker.

Les effets sonores peuvent être utilisés pour modifier la façon dont les sons ou les musiques d'arrière-plan sont joués. Prenez bien note que les effets sur les sons ne peuvent s'appliquer que sur les fichiers **wave** et **midi** et non sur les fichiers mp3. Cette section décrit les fonctions existantes pour l'utilisation et la modification des effets sonores. Pour bien utiliser ces fonctions, vous devez avoir une bonne connaissance du fonctionnement des sons en général et des synthétiseurs en particulier. Aucune explication des différents paramètres ne sera donnée ici. Veuillez consulter le web ou encore des livres sur le sujet afin d'obtenir davantage d'informations.

Pour appliquer un effet sonore à un son particulier, vous devrez soit mentionner le son lors de la conception de la ressource sonore, soit utiliser la fonction suivante.

sound_effect_set(snd,effect) Détermine un (ou plusieurs) effet(s) sonore(s) sur le son indiqué. `effect` peut être n'importe laquelle des valeurs suivantes :

se_none
se_chorus
se_echo
se_flanger
se_gargle
se_reverb
se_compressor
se_equalizer

Vous pouvez utiliser une combinaison d'effets en ajoutant les valeurs. Par exemple, vous pouvez utiliser :

```
sound_effect_set(snd,se_echo+se_reverb);
```

pour avoir à la fois un effet d'écho et de réverbération.

Tous les effets présentent certains réglages par défaut. Vous ne pouvez modifier ces derniers une fois qu'un effet ait été appliqué à un son. Il est très important de respecter l'ordre. Vous choisirez en premier lieu l'effet sur le son puis déterminerez les paramètres à lui appliquer. Une fois que vous appliquerez des effets sur un son, les réglages précédents le concernant ne seront plus disponibles et vous devrez les redéfinir si besoin est. Veuillez noter que tous les paramètres doivent être compris dans une certaine plage de valeurs comme indiqué ci-dessous. Les fonctions suivantes peuvent être utilisées pour modifier les paramètres des effets sonores :

sound_effect_chorus(snd,wetdry,depth,feedback,frequency,wave,delay,phase) Règle les paramètres pour l'effet **chorus** pour le son indiqué. Les paramètres suivants peuvent être fixés :

wetdry Ratio de la balance du signal des son réverbérés **wet** aux sons directs non réverbérés **dry** (plage : 0 à 100, 50 par défaut).
depth Pourcentage avec lequel le temps d'attente (*delay time*) est modulé par l'oscillateur basse-fréquence, exprimé en pourcent (plage : 0 à 100, 25 par défaut).
feedback Pourcentage du signal de sortie allant dans l'entrée pour les effets (plage : -99 à 99, 0 par défaut).
frequency Fréquence de la LFO (plage : 0 à 10, 0 par défaut).
wave Forme d'onde (*waveform*) de la LFO (**0** = triangle, **1** = wave, 1 par défaut).
delay Nombre de millisecondes pendant lesquelles l'entrée est mise en attente d'être jouée (plage : 0 à 20, 0 par défaut).
phase Différentiel de phase entre les LFO de gauche et de droite (plage : 0 à 4, 2 par défaut).

sound_effect_echo(snd,wetdry,feedback,leftdelay,rightdelay,pandelay) Règle les paramètres pour l'effet d'**écho** pour le son mentionné. Les paramètres suivants peuvent être déterminés :

wetdry Ratio de la balance du signal des sons réverbérés **wet** aux sons directs non réverbérés **dry** (plage : 0 à 100, 50 par défaut).
feedback Pourcentage d'attente vers l'entrée (plage : 0 à 100, 0 par défaut).
leftdelay Délai pour le canal de gauche en millisecondes (plage : 1 à 2000, 333 par défaut).
rightdelay Délai pour le canal de droite en millisecondes (plage : 1 à 2000, 333 par défaut).

pandelay Indique si l'on doit permuter les délais gauche et droite pour chacun des écho successifs (**0** = ne pas permuter, **1** = permuter, 0 par défaut).

sound_effect_flanger(snd,wetdry,depth,feedback,frequency,wave,delay,phase) Règle les paramètres pour l'effet **flanger** pour le son indiqué. Les paramètres suivants peuvent être fixés :

wetdry Ratio de la balance du signal des sons réverbérés **wet** aux sons non réverbérés **dry** (plage : 0 à 100, 50 par défaut).
depth Pourcentage avec lequel le temps d'attente (*delay time*) est modulé par l'oscillateur basse-fréquence, en pourcent (plage : 0 à 100, 25 par défaut).
feedback Pourcentage du signal de sortie allant dans l'entrée pour les effets (plage : -99 à 99, 0 par défaut).
frequency Fréquence de la LFO (plage : 0 à 10, 0 par défaut).
wave Forme d'onde (*waveform*) de la LFO (**0** = triangle, **1** = wave, 1 par défaut).
delay Nombre de millisecondes pendant lesquelles l'entrée est mise en attente d'être jouée (plage : 0 à 20, 0 par défaut).
phase Différentiel de phase entre les LFO de gauche et de droite (plage : 0 à 4, 2 par défaut).

sound_effect_gargle(snd,rate,wave) Règle les paramètres pour l'effet **gargle** pour le son mentionné. Les paramètres suivants peuvent être déterminés :

rate Taux de modulation, en Hertz (plage : 1 à 1000, 1 par défaut).
wave Forme de l'onde de modulation (**0** = triangle, **1** = carré, 0 par défaut).

sound_effect_reverb(snd,gain,mix,time,ratio) Règle les paramètres pour l'effet de réverbération pour le son indiqué. Les paramètres suivants peuvent être modifiés :

gain Gain en entrée du signal, en décibels (dB). (plage : -96 à 0, 0 par défaut).
mix Mixage de la réverbération, en dB (plage : -96 à 0, 0 par défaut).
time Temps de réverbération, en millisecondes (plage : 0.001

à 3000, 1000 par défaut).
ratio Ratio de fréquence (plage : 0.001 à 0.999, 0.001 par défaut).

sound_effect_compressor(snd,gain,attack,release,threshold,ratio,delay) Règle les paramètres pour l'effet **compressor** pour le son mentionné. Les paramètres suivants peuvent être réglés :

gain Gain en sortie du signal après compression (plage : -60 à 60, 0 par défaut).
attack Temps avant que la compression atteigne sa valeur maximale (plage : 0.01 à 500, 0.01 par défaut).
release Vitesse à laquelle la compression est interrompue après que l'entrée tombe en dessous du **threshold** (plage : 50 à 3000, 50 par défaut).
threshold Point à partir duquel débute la compression, en décibels (plage : -60 à 0, -10 par défaut).
ratio Ratio de compression (plage : 1 à 100, 10 par défaut).
delay Temps après que le Threshold est atteint avant que débute la phase d'attaque, en millisecondes (plage : 0 à 4, 0 par défaut).

sound_effect_equalizer(snd,center,bandwidth,gain) Règle les paramètres pour l'effet **equalizer** pour le son indiqué. Les paramètres suivants peuvent être fixés :

center Fréquence centrale, en hertz (plage : 80 à 16000).
bandwidth Largeur de bande (*Bandwidth*), en sémitones (plage : 1 à 36).
gain Gain (plage : -15 à 15).

Sons en 3D

Cette fonctionnalité n'est disponible que dans la version enregistrée de Game Maker.

(NDT : Dans cette section, le terme ***personne*** désigne la personne qui écoute le son en 3D).

Les sons en 3D se réfèrent aux sons ayant une position dans l'espace (mais aussi une vélocité) en accord avec la position de la personne. Bien que les sons en 3D soient plus utilisés dans les jeux en 3D, il vous est également possible de les utiliser dans des jeux en 2D. L'idée de base est la suivante : un son a une position dans l'espace. Pour toutes les fonctions, on supposera que la personne se trouve à la position (0,0,0). Le système calcule la façon dont la personne devrait entendre le son et adapte ce dernier en conséquence. L'effet est particulièrement de qualité si vous disposez de bonnes enceintes mais donne déjà un bon résultat sur de petits haut-parleurs.

En plus d'une position, le son possède également une vélocité. Cela conduit aux effets bien connus appelés effets *doppler* qui sont ici correctement synthétisés. Enfin, le son peut avoir une orientation et sera de nouveau adapté en fonction de la position de la personne.

Game Maker supporte les options de sons 3D à l'aide des fonctions suivantes. Ces fonctions ne fonctionnent que dans des ressources sonores destinées à être en 3D (le désavantage est que les sons 3D seront en monophonie et non en stéréophonie).

`sound_3d_set_sound_position(snd,x,y,z)` Règle la position du son indiqué en accord avec la position spatiale de la personne. Les valeurs sur l'axe des **x** augmentent de la gauche vers la droite, sur l'axe des **y** du bas vers le haut et sur l'axe des **z** du proche vers le lointain. Les valeurs sont mesurées en *meters*. Le volume avec lequel le son est entendu, dépend de ces valeurs de la même façon que dans le monde réel.

`sound_3d_set_sound_velocity(snd,x,y,z)` Règle la vélocité du son mentionné dans l'espace avec le vecteur indiqué. Veuillez noter que le réglage de la vélocité ne signifie pas que la position change. La vélocité est uniquement employée pour calculer les effets *doppler*. Aussi, si vous souhaitez déplacer le son dans l'espace, c'est à vous de modifier la position du son.

`sound_3d_set_sound_distance(snd,mindist,maxdist)` Régler la distance minimale à partir de laquelle le son n'augmentera plus en *loudness* et la distance maximale à laquelle le son ne pourra plus être entendu. Lorsque la distance se situe entre **0** et la distance minimale, le son est à son amplitude maximale.

Quand la distance est comprise entre les distances minimale et maximale, l'amplitude diminue lentement jusqu'à atteindre soit la distance maximale soit le seuil où le son n'est plus audible. Par défaut, la distance minimale est de **1** *meter* et la distance maximale est de **1 billion** *meters*.

sound_3d_set_sound_cone(snd,x,y,z,anglein,angleout,voloutside)

Normalement, le son présente à une distance donnée la même amplitude et ce dans toutes les directions. Vous avez la possibilité de modifier cela en réglant le cône du son et rendre ainsi le son directionnel. **x,y,z** indiquent la direction du cône du son. **anglein** précise l'angle intérieur. Si la personne qui écoute se trouve dans cet angle, elle entendra le son avec un volume normal. **angleout** précise l'angle extérieur. Lorsque la personne est en dehors de cet angle, le volume est indiqué par **voloutside**. Pour être précis, **voloutside** est un nombre négatif qui indique le nombre de centaines de décibels devant être soustraits du volume à l'intérieur. Entre les angles intérieur et extérieur, le volume décroît progressivement.

Musiques de CD

Cette fonctionnalité n'est disponible que dans la version enregistrée de Game Maker.

Il existe également un certain nombre de fonctions ayant traits à la lecture de musiques sur CD.

cd_init() Doit être invoquée avant d'utiliser les autres fonctions. Devrait être également appelée lorsqu'un CD a été retiré/introduit dans le lecteur (ou simplement de temps en temps).
cd_present() Retourne si un CD est présent dans le lecteur CD utilisé par défaut.
cd_number() Retourne le nombre de pistes du CD.
cd_playing() Retourne si un CD est en cours de lecture.
cd_paused() Retourne si un CD est actuellement en pause ou suspendu.
cd_track() Retourne le numéro de la piste courante (**1**= première piste).
cd_length() Retourne la longueur totale du CD en millisecondes.
cd_track_length(n) Retourne la longueur de la piste **n** du CD en millisecondes.
cd_position() Retourne la position actuelle du CD en millisecondes.
cd_track_position() Retourne la position actuelle de la piste en cours de lecture en millisecondes.
cd_play(first,last) Ordonne au lecteur de lire le CD de la piste **first** à **last**. Pour lire entièrement un CD, indiquez comme paramètres **1** et **1000**.
cd_stop() Stoppe la lecture du CD.
cd_pause() Effectue une pause.
cd_resume() Reprend la lecture après une pause.
cd_set_position(pos) Régler la position du CD en millisecondes.
cd_set_track_position(pos) Régler la position de la piste courante en millisecondes.
cd_open_door() Ouvre la porte du lecteur CD.
cd_close_door() Ferme la porte du lecteur CD.

Il existe aussi une fonction très générale permettant d'accéder à des fonctions multimédias de Windows.

MCI_command(str) Cette fonction envoie une commande sous forme de chaîne vers le système multimédia de Windows en utilisant l'interface **M**edia **C**ontrol **I**nterface (*MCI*). La fonction retourne le résultat de la commande sous forme de chaîne. Vous pouvez utiliser cette fonction pour contrôler toute sorte de

périphériques multimédias. Veuillez vous reporter à la documentation de Windows afin d'obtenir des informations sur la manière d'utiliser cette commande. Par exemple, la commande `MCI_command('play cdaudio from 1')` lit le CD en entier (après cependant que vous l'ayez correctement initialisé en utilisant d'autres commandes). Cette fonction est à utiliser uniquement dans le cadre d'une utilisation avancée !

Ecrans Splash, Highscores et autres menus Pop-ups

Dans cette section, nous allons décrire un certain nombre de fonctions pouvant être utilisées pour afficher des écrans Splash comprenant des vidéos, des images, etc., des messages et poser des questions au joueur et afficher aussi la liste des plus hauts scores (highscore list).

De l'information sur les écrans Splash et sur les messages peut être consultée dans les pages suivantes :

Ecrans Splash

De nombreux jeux possèdent ce que l'on appelle des écrans Splash (littéralement : écrans surgissants). Ces écrans affichent une vidéo, une image ou encore du texte. Ils sont utilisés très souvent au début du jeu (en guise d'intro), au début d'un niveau ou à la fin du jeu (pour afficher les crédits par exemple). De tels écrans Splash comprennent du texte, des images ou de la vidéo et peuvent être affichés avec *Game Maker* à n'importe quel moment du jeu. Le jeu sera temporairement suspendu pendant l'affichage de l'écran Splash. Voici les fonctions pouvant être utilisées :

`show_text(fname,full,backcol,delay)` Affiche un écran splash de type texte. **fname** est le nom du fichier texte (**.txt** ou .**rtf**). Il est obligatoire de placer vous-même ce fichier dans le répertoire où se trouve votre jeu. Ainsi, si vous créez une version autonome (Cf. exécutable) de votre jeu, n'oubliez pas de mettre le fichier à cet endroit. **full** indique si l'on doit afficher l'écran splash en mode plein écran. **backcol** correspond à la couleur d'arrière-plan et **delay** indique le délai en secondes avant de revenir au jeu (le joueur pourra toujours cependant cliquer sur l'écran avec la souris s'il souhaite retourner immédiatement au jeu).

`show_image(fname,full,delay)` Affiche un écran splash de type image. **fname** est le nom du fichier image (uniquement des fichiers **.bmp**, **.jpg** et **.wmf**). De même, vous devrez mettre ce fichier dans le répertoire du jeu. **full** signifie que l'écran doit être affiché en plein écran. **delay** est le délai en secondes avant de retourner au jeu.

`show_video(fname,full,loop)` Affiche un écran splash de type vidéo. **fname** est le nom du fichier vidéo (**.avi**,**.mpg**). De nouveau, vous devrez placer ce fichier dans le répertoire du jeu. **full** indique si l'affichage doit se faire en plein écran. **loop** précise si l'on doit lire la vidéo en boucle.

`show_info()` Affiche l'écran d'information sur le jeu.

`load_info(fname)` Charge les informations sur le jeu à partir du fichier de nom **fname**. Le type du fichier doit toujours être **rtf**. Cela permettra d'afficher différents fichiers d'aide à des moments divers.

Messages Popup et Questions

D'autres fonctions existent pour afficher des messages de type *pop up*, des questions, un menu présentant des choix ou encore une boîte de dialogue avec laquelle le joueur pourra entrer un nombre, une chaîne de caractères ou mentionner une couleur ou un nom de fichier :

`show_message(str)` Affiche une boîte de dialogue avec un message défini dans la chaîne **str**.

`show_message_ext(str,but1,but2,but3)` Affiche une boîte de dialogue avec un message sous forme de chaîne **str** et jusqu'à trois boutons. **But1**, **but2** et **but3** contiennent le texte des boutons. Une chaîne vide signifie que le bouton ne sera pas affiché. Dans le texte, vous pouvez utiliser le symbole **&** qui indique que le prochain caractère sera utilisé comme raccourci clavier pour ce bouton. La fonction retourne le numéro du bouton pressé (**0** si la touche **Esc** a été pressée).

`show_question(str)` Affiche une question. Retourne **true** si l'utilisateur a répondu **yes** et **false** sinon.

`get_integer(str,def)` Demande un nombre au joueur à l'aide d'une boîte de dialogue. **str** correspond au message. **def** est le nombre affiché par défaut.

`get_string(str,def)` Demande une chaîne au joueur à l'aide d'une boîte de dialogue. **str** correspond au message. **def** est la valeur affichée par défaut.

`message_background(back)` Détermine l'image de fond pour la boîte pop-up et pour toutes les fonctions vues précédemment. **back** doit être l'un des arrière-plans définis dans le jeu. Si **back** est partiellement transparent, ce sera l'image du message (uniquement pour Windows 2000 et supérieur).

`message_alpha(alpha)` Régler la transparence alpha des boîtes pop-up pour toutes les fonctions ci-dessus. alpha doit être compris entre **0** (complètement translucide) et **1** (non translucide) (uniquement pour Windows 2000 et supérieur).

`message_button(spr)` Détermine le sprite utilisé pour les boutons de la boîte pop-up. **spr** doit être un sprite comprenant trois images, la première représente le bouton lorsque celui-ci est à l'état non pressé et le curseur de la souris non au-dessus, la seconde correspond à l'apparence que doit avoir le bouton non pressé lorsque la souris est au-dessus et enfin la troisième représente le bouton pressé.

`message_text_font(name,size,color,style)` Fixe la police de caractères utilisée dans le texte de la boîte pop-up (cela doit être une police de Windows et non une ressource fontes insérée dans votre jeu !), **style** indique le style de police (**0**=normal, **1**=gras, **2**=italique et **3**=gras-italique).

`message_button_font(name,size,color,style)` Fixe la police utilisée pour

les boutons de la boîte pop-up. **style** indique le style de police (**0**=normal, **1**=gras, **2**=italique et **3**=gras-italique).

message_input_font(name,size,color,style) Détermine la police utilisée dans les champs de saisie de la boîte pop-up. **style** indique le style de police (**0**=normal, **1**=gras, **2**=italique et **3**=gras-italique).

message_mouse_color(col) Fixe la couleur de la police pour les boutons de la boîte pop-up lorsque la souris survole ceux-ci.

message_input_color(col) Fixe la couleur du fond des champs de saisie de la boîte pop-up.

message_caption(show,str) Détermine le titre de la boîte pop-up. **show** indique si l'on doit afficher une bordure (**1**) ou pas (**0**) et **str** correspond au titre lorsque la bordure est affichée.

message_position(x,y) Détermine la position de la boîte pop-up sur l'écran.

message_size(w,h) Fixe la taille de la boîte pop-up à l'écran. Si vous indiquez **0** comme largeur, la largeur de l'image sera utilisée. De même, si vous choisissez **0** comme hauteur, la hauteur sera calculée en fonction du nombre de lignes présent dans le message.

show_menu(str,def) Affiche un menu popup. **str** indique le texte du menu. Celui-ci comprend les différents articles du menu séparés par une barre verticale. Par exemple, *str = 'menu0|menu1|menu2'*. Lorsque le premier article du menu est sélectionné, la valeur **0** sera retournée, etc. Si le joueur ne choisit pas d'article dans le menu, la valeur par défaut **def** sera retournée.

show_menu_pos(x,y,str,def) Affiche un menu popup comme dans la fonction précédente mais à la position **x,y** sur l'écran.

get_color(defcol) Demande une couleur au joueur. **defcol** est la couleur par défaut. Si l'utilisateur presse **Cancel**, la valeur **-1** sera retournée.

get_open_filename(filter,fname) Demande au joueur un nom de fichier à ouvrir avec le filtre indiqué. Le filtre est de la forme *'name1|mask1|name2|mask2|...'.* Un masque contient les différentes options séparées par un trait vertical. ***** signifie n'importe quelle chaîne. Par exemple : *'bitmaps|*.bmp;*.wmf'*. Si le joueur presse **Cancel**, il sera retourné une chaîne vide.

get_save_filename(filter,fname) Demande au joueur un nom de fichier pour la sauvegarde et avec le filtre indiqué. Si l'utilisateur presse **Cancel**, une chaîne vide sera retournée.

get_directory(dname) Demande le nom d'un répertoire. **dname** est le nom par défaut. Si l'utilisateur presse **Cancel**, une chaîne vide est retournée.

get_directory_alt(capt,root) Une autre façon de renseigner le nom d'un

répertoire. **capt** est le titre à afficher. **root** est la racine de l'arborescence du répertoire à afficher. Utilisez la chaîne vide pour afficher toute l'arborescence.

Si l'utilisateur presse **Cancel**, une chaîne vide sera retournée.
`show_error(str,abort)` Affiche un message d'erreur standard (et/ou l'enregistre dans le fichier rapport des erreurs). **abort** indique si le jeu doit être arrêté.

Liste des plus hauts Scores (Highscore List)

La liste des highscores est l'une des fenêtres pop-up. Cette liste est mise à jour après chaque jeu. Les fonctions suivantes existent :

highscore_show(numb) Affiche la table des Highscores. **numb** est le nouveau score. Si ce score est suffisamment bon pour être ajouté dans la liste, le joueur pourra entrer son nom. Utilisez **-1** pour simplement afficher la liste actuelle des scores.

highscore_set_background(back) Détermine l'image d'arrière-plan à utiliser. `back` doit être l'index de l'une des ressources d'arrière-plan déjà définie dans le jeu.

highscore_set_border(show) Détermine si la page de Highscores doit présenter une bordure ou pas.

highscore_set_font(name,size,style) Détermine la police utilisée pour le texte affiché dans la table (cela doit être une fonte Windows et non l'une des ressources fontes). Vous indiquerez le nom, la taille et le style (**0**=normal, **1**= gras, **2**=italique, **3**=gras-italique).

highscore_set_colors(back,new,other) Détermine les couleurs à utiliser pour l'arrière-plan, la nouvelle entrée dans la table ainsi que pour les autres entrées.

highscore_set_strings(caption,nobody,escape) Change les différentes chaînes utilisées par défaut lors de l'affichage de la table des Highscores. `caption` est le titre de la page. `nobody` est la chaîne utilisée quand il n'y a personne déjà inscrit à un grade particulier. `escape` est la chaîne du bas de l'écran invitant à presser la touche **Esc**. Il vous est ainsi possible d'utiliser la langue étrangère de votre choix pour l'affichage des scores dans votre jeu.

highscore_show_ext(numb,back,border,col1,col2,name,size) Affiche la table des Highscores avec un certain nombre d'options (peut être aussi réalisé en utilisant certaines des fonctions vues précédemment). **numb** est le nouveau score. Si ce score est suffisamment bon pour être ajouté dans la liste, le joueur pourra alors entrer son nom. Utilisez **-1** afin d'afficher simplement la liste actuelle des scores. **back** correspond à l'image d'arrière-plan à utiliser, **border** indique si l'on doit ou non afficher la bordure. **col1** est la couleur à utiliser pour la nouvelle entrée, **col2** la couleur pour les autres entrées. **name** est le nom de la police à utiliser et **size** la taille de la police.

highscore_clear() Efface la liste des Highscores.

highscore_add(str,numb) Ajoute à la liste le joueur de nom **str** et avec le score **numb**.

highscore_add_current() Ajoute le score courant dans la liste des Highscores. Il sera demandé au joueur de saisir son nom.

highscore_value(place) Retourne le score de la personne située à la place indiquée (**1** à **10**). Cela peut être utile pour afficher votre propre liste de Highscores.

highscore_name(place) Retourne le nom de la personne se trouvant à la place indiquée (**1** à **10**).

draw_highscore(x1,y1,x2,y2) Affiche la tables des Highscores dans la room et dans la boîte indiquée, en utilisant la police actuelle.

Ressources

Dans *Game Maker*, vous pourrez définir différents types de ressources, comme les sprites, les sons, les polices de caractères, les objets, etc. Dans ce chapitre, vous trouverez de nombreuses fonctions qui vous donneront des informations sur les ressources. Dans le chapitre suivant, vous trouverez des informations sur la façon de modifier et de créer des ressources à la volée.

Vous trouverez des informations sur les ressources dans les pages suivantes :

Sprites

Les fonctions suivantes vous donneront des informations sur les sprites :

sprite_exists(ind) Retourne si le sprite d'index **ind** existe.
sprite_get_name(ind) Retourne le nom du sprite d'index **ind**.
sprite_get_number(ind) Retourne le nombre de sous-images contenues dans le sprite d'index **ind**.
sprite_get_width(ind) Retourne la largeur du sprite d'index **ind**.
sprite_get_height(ind) Retourne la hauteur du sprite d'index **ind**.
sprite_get_transparent(ind) Indique si le sprite d'index **ind** est transparent.
sprite_get_smooth(ind) Indique si le sprite d'index **ind** possède des bords adoucis.
sprite_get_preload(ind) Indique si le sprite d'index **ind** doit être préchargé.
sprite_get_xoffset(ind) Retourne l'offset **x** du sprite d'index **ind**.
sprite_get_yoffset(ind) Retourne l'offset **y** du sprite d'index **ind**.
sprite_get_bbox_left(ind) Retourne le côté gauche de la boîte de rebond du sprite d'index **ind**.
sprite_get_bbox_right(ind) Retourne le côté droit de la boîte de rebond du sprite d'index **ind**.
sprite_get_bbox_top(ind) Retourne le côté supérieur de la boîte de rebond du sprite d'index **ind**.
sprite_get_bbox_bottom(ind) Retourne le côté inférieur de la boîte de rebond du sprite d'index **ind**.
sprite_get_bbox_mode(ind) Retourne le mode de rebond de la boîte (**0**=automatique, **1**=image pleine, **2**=manuel) du sprite d'index **ind**.
sprite_get_precise(ind) Indique si le sprite d'index **ind** utilise la vérification précise pour les collisions.

Sons

Les fonctions suivantes vous donneront des informations sur les sons :

sound_exists(ind) Indique si le son d'index **ind** existe.
sound_get_name(ind) Retourne le nom du son d'index **ind**.
sound_get_kind(ind) Retourne le type du son d'index **ind** (**0**=normal, **1**=arrière-plan, **2**=3D, **3**=mmplayer).
sound_get_preload(ind) Indique si le son d'index **ind** doit être préchargé.

Les sons utilisent beaucoup de ressources et la plupart des systèmes ne peuvent que stocker et lire qu'un nombre limité de sons. Si vous envisagez de faire un gros jeu, vous devrez bien connaître la manière et à quel moment les jeux sont chargés dans la mémoire audio. Vous pouvez mettre à off le switch de l'option de chargement des sons afin d'être sûr que ces derniers ne seront chargés en mémoire uniquement en cas de besoin. Cela présente l'inconvénient que lors de la première utilisation du son, il est possible que vous entendiez un léger bruit. De plus, les sons ne sont automatiquement déchargés quand vous n'en avez plus besoin. Pour davantage de contrôle sur les sons, vous pouvez utiliser les fonctions suivantes.

sound_discard(index) Libère la mémoire audio utilisée par le son indiqué.
sound_restore(index) Restaure le son indiqué dans la mémoire audio afin d'être lu immédiatement.

Arrière-plans

Les fonctions suivantes vous donneront des informations sur les arrière-plans :

background_exists(ind) Indique si l'arrière-plan d'index **ind** existe.
background_get_name(ind) Retourne le nom de l'arrière-plan d'index **ind**.
background_get_width(ind) Retourne la largeur de l'arrière-plan d'index **ind**.
background_get_height(ind) Retourne la hauteur de l'arrière-plan d'index **ind**.
background_get_transparent(ind) Indique si l'arrière-plan d'index **ind** est transparent.
background_get_smooth(ind) Indique si l'arrière-plan d'index **ind** possède des bords adoucis.
background_get_preload(ind) Indique si l'arrière-plan d'index **ind** doit être préchargé.

Polices de caractères (Fonts)

Les fonctions suivantes vous donneront des informations sur les polices de caractères :

font_exists(ind) Indique si la police d'index **ind** existe.

font_get_name(ind) Retourne le nom de la police d'index **ind**.

font_get_fontname(ind) Retourne le nom de police de la police d'index **ind**.

font_get_bold(ind) Indique si la police d'index **ind** est de type **gras**.

font_get_italic(ind) Indique si la police d'index **ind** est de type **italique**.

font_get_first(ind) Retourne l'index du premier caractère de la police d'index **ind**.

font_get_last(ind) Retourne l'index du dernier caractère de la police d'index **ind**.

Chemins (Paths)

Les fonctions suivantes vous donneront des informations sur les chemins :

path_exists(ind) Indique si le chemin d'index **ind** existe.

path_get_name(ind) Retourne le nom du chemin d'index **ind**.

path_get_length(ind) Retourne la longueur du chemin d'index **ind**.

path_get_kind(ind) Retourne le type de connexion du chemin d'index **ind** (**0**=droit (straight), **1**=arrondi (smooth)).

path_get_closed(ind) Indique si le chemin d'index **ind** est fermé ou pas.

path_get_precision(ind) Retourne la précision utilisée lors de la création de chemins arrondis (smoothed).

path_get_number(ind) Retourne le nombre de points définis pour le chemin d'index **ind**.

path_get_point_x(ind,n) Retourne l'abscisse **x** du **n**ième point défini pour le chemin d'index **ind**. **0** correspond au premier point.

path_get_point_y(ind,n) Retourne l'ordonnée **y** du **n**ième point défini pour le chemin d'index **ind**. **0** correspond au premier point.

path_get_point_speed(ind,n) Retourne le facteur de vitesse au **n**ième point défini et pour le chemin d'index **ind**. **0** correspond au premier point.

path_get_x(ind,pos) Retourne l'abscisse **x** à la position **pos** pour le chemin d'index **ind**. **pos** doit être compris entre **0** et **1**.

path_get_y(ind,pos) Retourne l'ordonnée **y** à la position **pos** pour le chemin d'index **ind**. **pos** doit être compris entre **0** et **1**.

path_get_speed(ind,pos) Retourne le facteur de vitesse à la position **pos** pour le chemin d'index **ind**. **pos** doit être compris entre **0** et **1**.

Scripts

Les fonctions suivantes vous donneront des informations sur les scripts :

`script_exists(ind)` Indique si le script d'index **ind** existe.

`script_get_name(ind)` Retourne le nom du script d'index **ind**.

`script_get_text(ind)` Retourne la chaîne texte du script d'index **ind**.

Lignes de temps (Time lines)

Les fonctions suivantes vous donneront des informations sur les lignes de temps :

`timeline_exists(ind)` Indique si la ligne de temps d'index **ind** existe.

`timeline_get_name(ind)` Retourne le nom de la ligne de temps d'index **ind**.

Objets

Les fonctions suivantes vous donneront des informations sur les objets :

object_exists(ind) Indique si l'objet d'index **ind** existe.

object_get_name(ind) Retourne le nom de l'objet d'index **ind**.

object_get_sprite(ind) Retourne l'index du sprite par défaut de l'objet d'index **ind**.

object_get_solid(ind) Indique si l'objet d'index **ind** est solide par défaut.

object_get_visible(ind) Indique si l'objet d'index **ind** est visible par défaut.

object_get_depth(ind) Retourne la profondeur de l'objet d'index **ind**.

object_get_persistent(ind) Indique si l'objet d'index **ind** est persistent.

object_get_mask(ind) Retourne l'index du masque de l'objet d'index **ind** (**-1** si pas de masque).

object_get_parent(ind) Retourne l'index de l'objet parent de l'objet **ind (-1** si pas de parent).

object_is_ancestor(ind1,ind2) indique si l'objet **ind2** est l'un des ancêtres de l'objet **ind1**.

Rooms

Les fonctions suivantes vous donneront des informations sur les rooms :

`room_exists(ind)` Indique si la room d'index **ind** existe.

`room_get_name(ind)` Retourne le nom de la room d'index **ind**.

Veuillez noter qu'en raison que les rooms changent pendant leur affichage, il existe d'autres routines permettant d'obtenir des informations sur le contenu de la room courante.

Modification des ressources

Ces fonctions sont uniquement disponibles dans la version enregistrée de Game Maker.

Il est possible de créer de nouvelles ressources pendant le jeu. Vous pouvez aussi bien entendu modifier des ressources existantes. Ce chapitre décrit les possibilités en la matière. Soyez cependant prévenu. **La modification de ressources peut conduire très facilement à de sérieuses erreurs dans vos jeux !!!** Vous devrez respecter les règles suivantes si vous souhaitez modifier les ressources :

- Ne jamais modifier les ressources en cours d'utilisation. Cela conduira obligatoirement à des erreurs ! Par exemple, ne changez pas un sprite actuellement utilisé par une instance.
- Lorsque vous sauvegardez le jeu pendant son fonctionnement, les ressources ajoutées et/ou modifiées ne seront **PAS** sauvegardées avec le jeu. Ainsi, si vous rechargez le jeu ultérieurement, les ressources ne seront peut-être plus présentes. En général, quand vous manipulez des ressources, vous ne devrez plus utiliser la routine système intégrée de chargement et de sauvegarde des jeux.
- Lorsque vous relancez le jeu alors que ce dernier était en cours, les ressources modifiées ne seront **PAS** restaurées dans leur état d'origine. En général, lorsque vous manipulez des ressources, vous ne devrez plus utiliser l'action ou la fonction destinée à relancer le jeu.
- La manipulation de ressources peut être lente. Par exemple, le changement de sprites ou d'arrière-plans est relativement lent. A ne pas utiliser donc pendant le jeu.
- La création de ressources pendant le jeu (en particulier des sprites et les arrière-plans) consomme de très grandes quantités de mémoire. Soyez par conséquent extrêmement prudent. Par exemple, si vous avez 32 images de 128x128 dans un sprite animé et que vous décidiez d'en créer 36 copies avec rotation, vous utiliserez alors jusqu'à 36x32x128x128x4 = 36 Mo de mémoire !
- Soyez sûr d'effacer les ressources dont vous n'avez plus besoin. Sinon, le système connaîtra rapidement des problèmes de mémoire.

En général, abstenez-vous de modifier une ressource pendant le déroulement du jeu. Le mieux sera de créer et de changer les ressources au début du jeu ou encore au début de la room.

De l'information sur la modification des ressources peut être trouvée dans les pages suivantes :

- Sprites
- Sons
- Arrière-plans
- Polices de caractères
- Chemins
- Scripts
- Lignes de temps (Time lines)
- Objets
- Rooms

Sprites

Les routines suivantes sont disponibles pour modifier les propriétés des sprites :

sprite_set_offset(ind,xoff,yoff) Régler l'offset du sprite d'index **ind**.
sprite_set_bbox_mode(ind,mode) Régler le mode de rebond de la boîte du sprite d'index **ind** (**0**=automatique, **1**=image pleine, **2**=manuel).
sprite_set_bbox(ind,left,top,right,bottom) Paramètre la boîte de rebond du sprite d'index **ind**. Ne fonctionne que lorsque le mode est positionné sur *manuel*.
sprite_set_precise(ind,mode) Détermine si le sprite d'index **ind** doit utiliser la vérification précise lors des collisions (**true** ou **false**).

Les routines suivantes peuvent être utilisées afin de créer de nouveaux sprites ou les supprimer.

sprite_duplicate(ind) Crée une copie du sprite d'index **ind**. La commande retourne l'index du nouveau sprite. Si une erreur survient, la valeur **-1** sera retournée.
sprite_assign(ind,spr) Affecte le sprite indiqué au sprite **ind**. Ceci permet donc de réaliser une copie du sprite *spr*. De cette manière, vous pourrez facilement affecter un sprite existant à un nouveau sprite différent.
sprite_merge(ind1,ind2) Ajoute en fin de sprite, les images du sprite **ind2** dans le sprite **ind1**. Si les tailles ne correspondent pas, les sprites seront adaptés en conséquence. Le sprite **ind2** n'est pas effacé !
sprite_add(fname,imgnumb,precise,transparent,smooth,preload,xorig,yorig) Ajoute l'image stockée dans le fichier **fname** dans les ressources de sprites. Seules les images de type **BMP**, **JPG** et **GIF** peuvent être utilisées. Si l'image est de type **BMP** ou **JPG**, il pourra y avoir une liste de sprites qui contiendront chacun une sous-image. Utilisez **imgnumb** pour indiquer leur numéro (**1** pour une image unique). Pour les images **GIF** (animées), cet argument ne sera pas utilisé : c'est le nombre d'images dans le fichier **GIF** qui sera utilisé. `precise` indique si la vérification précise des collisions doit être employée. `transparent` précise si l'image doit être partiellement transparente. `smooth` indique si l'on doit arrondir les bords. `preload` précise si l'on doit précharger l'image dans la mémoire des textures. `xorig` et `yorig` indiquent la position de l'origine du sprite. La fonction retourne l'index du nouveau sprite que vous pourrez alors utiliser pour l'afficher ou encore l'assigner à la variable *sprite_index* d'une instance.

Si une erreur survient, la valeur **-1** sera retournée.

sprite_replace(ind,fname,imgnumb,precise,transparent,smooth,preload,xorig,yorig) Identique à la fonction précédente mais ici, c'est le sprite d'index **ind** qui est remplacé. La fonction retourne un résultat précisant si la commande a réussi.

sprite_create_from_screen(x,y,w,h,precise,transparent,smooth,preload,xorig,yorig) Crée un sprite en copiant la zone mentionnée de l'écran. Cela vous permet de créer n'importe quel sprite. Affichez l'image sur l'écran en utilisant les fonctions d'affichage classiques puis créez un sprite à l'aide de cette fonction (si vous ne placez pas cette fonction dans l'événement d'affichage, vous pourrez même faire en sorte que le sprite ne soit pas visible sur l'écran simplement en ne rafraîchissant pas l'écran). Les autres paramètres sont les mêmes que ceux vus précédemment. La fonction retourne l'index du nouveau sprite. Une précision importante est à faire ici. Bien que nous parlons ici d'écran, c'est la région d'affichage qui est importante. Le fait qu'il y ait une fenêtre sur l'écran et que l'image peut être redimensionnée dans cette fenêtre n'a pas d'importance.

sprite_add_from_screen(ind,x,y,w,h) Ajoute une zone de l'écran comme prochaine sous-image pour le sprite d'index **ind**.

sprite_create_from_surface(id,x,y,w,h,precise,transparent,smooth,preload,xorig,yorig) Crée un sprite en copiant la zone indiquée de la surface de l'écran d'ID **id**. Cela vous permet de créer tous les sprites souhaités. Affichez l'image sur la surface en utilisant les fonctions habituelles d'affichage puis créez un sprite à partir de cette surface. La fonction retourne l'index du nouveau sprite. Veuillez toutefois noter que les valeurs **alpha** sont préservées dans le sprite.

sprite_add_from_surface(ind,id,x,y,w,h) Ajoute une aire de la surface d'ID **id** comme prochaine sous-image dans le sprite d'index **ind**.

sprite_delete(ind) Efface le sprite de la mémoire, libérant ainsi la mémoire utilisée.

La routine suivante existe pour modifier l'apparence d'un sprite.

sprite_set_alpha_from_sprite(ind,spr) Change les valeurs **alpha** (transparence) du sprite d'index `ind` en utilisant les valeurs de tonalité (hue values) du sprite `spr`. Cette opération ne peut être annulée.

Sons

Les routines suivantes peuvent être utilisées afin de créer de nouveaux sons et les supprimer.

sound_add(fname,kind,preload) Ajoute une ressource sonore dans le jeu. **fname** est le nom du fichier son. **kind** indique le type de son (**0**=normal, **1**=arrière-plan, **2**=3D, **3**=mmplayer).**preload** précise si le son doit être immédiatement stocké en mémoire audio (**true** ou **false**). La fonction retourne l'index du nouveau son qui pourra être utilisé pour jouer le son (**-1** si une erreur survient comme par exemple, dans le cas où le fichier n'existe pas).
sound_replace(index,fname,kind,loadonuse) identique à la fonction précédente mais cette fois-ci, il n'est pas créé de nouveau son mais l'index du son existant est remplacé, libérant ainsi l'ancien son. Retourne un résultat précisant si la commande s'est bien déroulée.
sound_delete(index) Efface le son indiqué, libérant ainsi la mémoire allouée pour ce son. Cette opération ne peut être annulée.

Arrière-plans

Les routines suivantes peuvent être utilisées pour créer de nouveaux arrière-plans ou les supprimer.

background_duplicate(ind) Crée une copie de l'arrière-plan d'index **ind**. La commande retourne l'index du nouvel arrière-plan. Si une erreur survient, la valeur **-1** sera retournée.

background_assign(ind,back) Affecte l'arrière-plan **back** indiqué à l'arrière-plan **ind**. Cette commande vous permet donc de faire une copie d'un arrière-plan.

background_add(fname,transparent,smooth,preload) Ajoute l'image stockée dans le fichier de nom **fname** dans la liste des ressources relatives aux arrière-plans. Seules les images de type **BMP** et **JPG** peuvent être manipulées. `transparent` indique si l'image doit être partiellement transparente. `smooth` précise si l'on doit arrondir les bords. `preload` indique si l'image doit être préchargée dans la mémoire de textures. La fonction retourne l'index du nouvel arrière-plan que vous pourrez ensuite utiliser pour l'afficher ou l'assigner à la variable *background_index[0]* afin de le rendre visible dans la room courante. Si une erreur survient, la valeur **-1** sera retournée.

background_replace(ind,fname,transparent,smooth,preload) Identique à la fonction précédente sauf qu'ici, seul l'arrière-plan d'index **ind** sera remplacé. La fonction retourne un résultat précisant si celle-ci s'est bien déroulée. Dans le cas où l'arrière-plan était déjà visible dans la room, il sera également remplacé .

background_create_color(w,h,col,preload) Crée un arrière-plan de la dimension et de la couleur indiquées. Cette commande retourne l'index du nouvel arrière-plan. Si une erreur survient à ce stade, la valeur **-1** sera retournée.

background_create_gradient(w,h,col1,col2,kind,preload) Crée un arrière-plan de type arc-en-ciel (gradient) avec les dimensions indiquées. **col1** et **col2** précisent les deux couleurs. **kind** est une nombre entre **0** et **5** indiquant le type d'arc-en-ciel : **0**=horizontal **1**=vertical, **2**= rectangle, **3**=ellipse, **4**=double horizontal, **5**=double vertical. La commande retourne l'index du nouvel arrière-plan. Si une erreur survient, la valeur **-1** sera retournée.

background_create_from_screen(x,y,w,h,transparent,smooth,preload) Crée un arrière-plan en copiant la zone indiquée de l'écran. Ceci vous permettra de créer toute sorte d'arrière-plan. Affichez l'image à l'écran en utilisant les fonctions habituelles d'affichage puis créez un arrière-plan en utilisant

cette fonction (si vous ne placez pas la fonction dans l'événement d'affichage, vous aurez la possibilité de créer un arrière-plan qui ne sera pas visible sur l'écran simplement en ne rafraîchissant pas l'écran). Les autres paramètres restent les mêmes que précédemment. La fonction retourne l'index du nouvel arrière-plan. Une précision importante est à faire ici. Lorsque nous parlons d'écran, il est fait référence à la zone d'affichage. Le fait qu'il y ait une fenêtre à l'écran et que l'image puisse être redimensionnée dans cette fenêtre n'a pas ici d'importance.

background_create_from_surface(id,x,y,w,h,transparent,smooth,preload) Crée un arrière-plan en copiant la zone indiquée de la surface d'ID **id**. Cela vous permet de créer tout arrière-plan souhaité. Afficher l'image sur la surface en utilisant les fonctions habituelles d'affichage puis créez un arrière-plan à l'aide de cette fonction. Veuillez noter que les valeurs **alpha** seront conservées dans l'arrière-plan.

background_delete(ind) Efface l'arrière-plan de la mémoire, libérant ainsi la mémoire allouée pour celui-ci.

La routine suivante permet de changer l'apparence d'un arrière-plan.

background_set_alpha_from_background(ind,back) Change les valeurs **alpha** (transparence) de l'arrière-plan d'index **ind** en utilisant les valeurs de tonalité de l'arrière-plan **back**. Cette opération ne peut être annulée.

Polices de caractères

Il est possible de créer, de remplacer et d'effacer des polices durant le jeu en utilisant les fonctions suivantes (cependant, ne remplacez pas une police définie comme police courante ou alors il sera nécessaire de la redéfinir par la suite).

font_add(name,size,bold,italic,first,last) Ajoute une nouvelle police et retourne son index. Vous indiquerez son nom, sa taille, son type (**gras** ou *italique*) et le premier et dernier caractère devant être créés.
font_add_sprite(spr,first,prop,sep) Ajoute une nouvelle police et retourne son index. La police est créée ici à partir du sprite **spr**. Le sprite devra contenir une sous-image pour chaque caractère. **first** indique l'index du premier caractère contenu dans le sprite. Par exemple, utilisez `ord('0')` si votre sprite contient uniquement des chiffres. **prop** indique si la police est de type proportionnel. Dans une police proportionnelle, la largeur de chaque caractère est déterminée selon la largeur de la boîte de bord. Enfin, **sep** indique le nombre de blancs séparant les caractères dans le sens horizontal. Une valeur typique à utiliser est généralement comprise entre **2** et **8** et est fonction de la taille de la police.
font_replace(ind,name,size,bold,italic,first,last) Remplace la police **ind** avec une nouvelle police. Vous indiquerez son nom, sa taille, son type (**gras** ou *italique*) et le premier et dernier caractère devant être créés.
font_replace_sprite(ind,spr,first,prop,sep) Remplace la police **ind** avec une nouvelle police basée ici sur le sprite **spr**.
font_delete(ind) Efface la police d'index **ind**, libérant la mémoire utilisée par celle-ci.

Chemins

Il est possible de créer des chemins (paths) et d'ajouter de nouveaux points à ces derniers. Cependant, ne jamais modifier un chemin en cours d'utilisation par une instance. Cela pourrait conduire à des résultats inattendus. Les fonctions suivantes peuvent être utilisées :

path_set_kind(ind,val) Détermine le type de connexion pour le chemin d'index **ind** (**0**=straight, **1**=smooth).
path_set_closed(ind,closed) Détermine si le chemin **ind** doit être fermé (**true**) ou ouvert (**false**).
path_set_precision(ind,prec) Détermine la précision avec laquelle le chemin de type *smooth* doit être calculé (doit être compris entre **1** et **8**).
path_add() Ajoute un nouveau chemin vide. L'index du chemin est retourné.
path_delete(ind) Efface le chemin d'index **ind**.
path_duplicate(ind) Crée une copie du chemin d'index **ind**. Retourne l'index de la copie.
path_assign(ind,path) Assigne le chemin indiqué **path** au chemin d'index **ind**. Cela permet donc d'effectuer une copie du chemin. De cette façon, vous pourrez facilement affecter un chemin existant à un nouveau chemin.
path_append(ind,path) Ajoute le chemin indiqué **path** en fin de chemin **ind**.
path_add_point(ind,x,y,speed) Ajoute un point au chemin d'index **ind**, à la position **(x,y)** et avec le facteur de vitesse **speed**. Souvenez-vous qu'un facteur de **100** correspond à la vitesse actuelle. Plus la valeur de ce facteur sera faible, plus la vitesse ira en diminuant et vice-versa.
path_insert_point(ind,n,x,y,speed) Insère un point dans le chemin d'index **ind** avant le point **n**, à la position **(x,y)** et avec le facteur de vitesse **speed**.
path_change_point(ind,n,x,y,speed) Change le point **n** du chemin d'index **ind** à la position **(x,y)** et avec le facteur de vitesse **speed**.
path_delete_point(ind,n) Efface le point **n** du chemin d'index **ind**.
path_clear_points(ind) Supprime tous les points du chemin d'index **ind**, rendant ainsi ce dernier vide.
path_reverse(ind) Inverse le chemin.
path_mirror(ind) Effectue un effet miroir du chemin **ind** dans le sens horizontal (en accord avec son centre).
path_flip(ind) Renverse le chemin **ind** dans le sens vertical (en accord avec son centre).

path_rotate(ind,angle) Effectue une rotation du chemin **ind** dans le sens contraire des aiguilles d'une montre, de **angle** degrés (par rapport à son centre).
path_scale(ind,xscale,yscale) Effectue une mise à l'échelle du chemin **ind** en utilisant les facteurs indiqués **xcale** et **yscale** (à partir de son centre).
path_shift(ind,xshift,yshift) Effectue un décalage du chemin **ind** en utilisant les valeurs mentionnées **xshift** et **yshift**.

Scripts

Les scripts ne doivent pas être modifiés pendant l'exécution du jeu. Les scripts font partie de la logique de ce dernier. Modifier les scripts pendant le déroulement du jeu reviendrait à ce que les scripts se modifient par eux-même ce qui conduirait très facilement à des erreurs. Cependant, il existe d'autres moyens pour parvenir à ce résultat. Si vraiment vous souhaitez exécuter une portion de code inconnue lors de la conception du jeu (CAD :à partir d'un fichier), vous devrez utiliser les fonctions suivantes :

`execute_string(str)` Exécute la portion de code contenue dans la chaîne **str**.
`execute_file(fname)` Exécute la portion de code contenue dans le fichier **fname**.

Parfois, vous désirerez stocker un index de script dans une variable pour l'exécuter ensuite. A cet usage, vous utiliserez la fonction suivante.

`script_execute(scr,arg0,arg1,...)` Exécute le script d'index **scr** en utilisant les arguments fournis.

Lignes de temps (Time lines)

Les routines suivantes sont disponibles pour la création et la modification des lignes de temps (*time lines*). Ne pas modifier les lignes de temps en cours d'utilisation !

`timeline_add()` Ajoute une nouvelle ligne de temps. La fonction retourne l'index de la ligne de temps créée.

`timeline_delete(ind)` Efface la ligne de temps d'index **ind**. Soyez certain cependant qu'aucune instance n'utilise cette ligne de temps et ce dans chacune des rooms.

`timeline_moment_add(ind,step,codestr)` Ajoute une action (code) à la ligne de temps **ind** au step du moment. **codestr** contient le code pour les actions. Si le **step** n'existe pas, il sera créé. Vous avez ainsi la possibilité d'ajouter plusieurs actions de code pour un même moment.

`timeline_moment_clear(ind,step)` Vous utiliserez cette fonction pour effacer toutes les actions d'un moment particulier.

Objets

Les objets peuvent être manipulés mais aussi créés pendant le jeu. **NE JAMAIS** modifier ou effacer un objet comprenant déjà des instances. Cela conduira à des effets de bords inattendus car certaines propriétés de l'objet sont stockées avec l'instance et par conséquent, la modification des propriétés de l'objet ne produira pas l'effet souhaité.

`object_set_sprite(ind,spr)` Détermine le sprite de l'objet d'index **ind**. Utilisez la valeur **-1** afin de supprimer le sprite courant de l'objet.
`object_set_solid(ind,solid)` Détermine si les instances créées de l'objet doivent par défaut être de type *solide* (**true** ou **false**).
`object_set_visible(ind,vis)` Détermine si les instances créées de l'objet doivent par défaut être *visibles* (**true** ou **false**).
`object_set_depth(ind,depth)` Détermine la *profondeur* par défaut des instances créées de l'objet.
`object_set_persistent(ind,pers)` Détermine si les instances créées de l'objet doivent par défaut être *persistantes* (**true** ou **false**).
`object_set_mask(ind,spr)` Détermine le masque du sprite de l'objet d'index **ind**. Utilisez la valeur **-1** pour que le masque corresponde au sprite de l'objet.
`object_set_parent(ind,obj)` Détermine le parent de l'objet. Utilisez la valeur **-1** pour que l'objet n'ait pas de parent. Modifier le parent d'un objet modifie également le comportement des instances de ce même objet.

Les routines suivantes sont très utiles pour créer des objets 'à la volée'. Comme pour toute ressource modifiant les routines, il convient d'être très prudent afin d'éviter de créer de nouveaux objets en boucle.

`object_add()` Ajoute un nouvel objet. Cette commande retourne l'index de l'objet créé. Vous pourrez utiliser cet index dans les routines ci-dessus pour fixer certaines propriétés de l'objet puis pour créer des instances de ce même objet.
`object_delete(ind)` Efface l'objet d'index **ind**. Soyez certain qu'aucune autre instance de l'objet n'existe pas déjà dans une autre room.
`object_event_add(ind,evtype,evnumb,codestr)` Afin de donner un comportement à l'objet, nous devons définir des événements pour ce dernier. Seules des actions sous forme de code peuvent être ajoutées à des événements. Il sera nécessaire de préciser l'objet, le type d'événement, le numéro de l'événement (utilisez les constantes qui ont été spécifiées auparavant dans la fonction

event_perform()). Enfin, vous devrez fournir la chaîne contenant le code devant être exécuté. Il est possible d'ajouter plusieurs actions de code pour chaque événement.

`object_event_clear(ind,evtype,evnumb)` Vous utiliserez cette fonction pour effacer toutes les actions d'un événement particulier.

La création d'objets est particulièrement utile lorsque vous concevez des scripts ou bien des librairies d'actions. Par exemple, un script d'initialisation peut créer un objet destiné à afficher un texte donné tandis qu'un autre peut utiliser ce même objet pour afficher un texte particulier. De cette façon, vous disposerez d'un mécanisme simple pour afficher des textes sans avoir recours systématiquement à la création d'objets en utilisant l'interface standard.

Rooms

La manipulation de salles (rooms) 'à la volée' peut être une chose dangereuse. Vous devez bien comprendre que les salles changent à tout moment en fonction de ce qui se passe dans le jeu. Ceci ne concerne normalement que la salle actuellement active et il existe de nombreuses routines décrites dans les sections précédentes destinées à manipuler les instances, les arrière-plans et les tuiles de la salle active. Mais les changements opérés dans la salle active ne perdureront que si la salle est de type persistante. Par conséquent, vous ne devriez jamais manipuler certains aspects de la salle actuellement active ou de n'importe quelle salle persistante qui a déjà été visitée auparavant. De tels changements passeront inaperçus la plupart du temps et pourront parfois mener à des erreurs inattendues. Etant donné que ces salles sont reliées entre elles de manière sophistiquée, il n'existe donc pas de routine pour supprimer une salle.

Les routines suivantes sont disponibles.

`room_set_width(ind,w)` Régler la largeur **w** de la room d'index **ind**.
`room_set_height(ind,h)` Régler la hauteur **h** de la room d'index **ind**.
`room_set_caption(ind,str)` Détermine le titre **str** de la salle d'index **ind**.
`room_set_persistent(ind,val)` Affiche la room d'index **ind** que celle-ci soit persistante ou non.
`room_set_code(ind,str)` Fournit la chaîne de code d'initialisation pour la room d'index **ind**.
`room_set_background_color(ind,col,show)` Fixe les propriétés de couleur de la room d'index **ind** si cette dernière ne possède pas d'image d'arrière-plan. **col** indique la couleur et **show** indique si la couleur doit être affichée ou pas.
`room_set_background(ind,bind,vis,fore,back,x,y,htiled,vtiled,hspeed,vspeed,alpha)` Régler l'arrière-plan d'index **bind** (**0** à **7**) pour la room d'index **ind**. **vis** indique si l'arrière-plan est visible et **fore** si c'est actuellement un premier plan. **back** est l'index de l'image d'arrière-plan. **x,y** précise la position de l'image et **htiled** et **vtiled** mentionnent si l'image doit être affichée sous forme de tuiles. **hspeed** et **vspeed** donnent la vitesse à laquelle l'arrière-plan doit se déplacer et **alpha** correspond à la valeur alpha de translucidité (**1** = solide et le plus rapide).
`room_set_view(ind,vind,vis,xview,yview,wview,hview,xport,yport,wport,hport,hborder,vborder,hspeed,vspeed,obj)` Sélectionne la vue d'index **vind** (**0** à **7**) pour la room d'index **ind**. **vis** indique si la vue est visible. **xview**, **yview**, **wview** et **hview** précisent la position de la vue dans la room.

xport, **yport**, **wport** et **hport** donnent la position à l'écran. Si la vue doit suivre un objet, **hborder** et **vborder** déterminent la bordure visible minimale à conserver autour de l'objet. **hspeed** et **vspeed** indiquent la vitesse maximale avec laquelle la vue peut se déplacer. **obj** est l'index de l'objet ou encore celui de l'instance.
`room_set_view_enabled(ind,val)` Détermine si les vues doivent être autorisées pour la room d'index **ind**.
`room_add()` Ajoute une nouvelle room. Cette commande renvoie l'index de la room. Veuillez noter que la room ne fera pas partie de l'ordre des séquences des rooms. Aussi, la nouvelle room ne possédera pas de room précédente ou suivante. Si vous désirez vous déplacer vers une room que vous avez ajoutée, il sera nécessaire de fournir l'index de cette room.
`room_duplicate(ind)` Effectue une copie de la room d'index **ind**. Cette commande retourne l'index de la nouvelle room.
`room_assign(ind,room)` Assigne la room indiquée à la room **ind**. Cette commande effectue donc une copie de la room.
`room_instance_add(ind,x,y,obj)` Ajoute une nouvelle instance de l'objet **obj** à la room, en la plaçant à la position indiquée. Cette commande retourne l'index de l'instance.
`room_instance_clear(ind)` Supprime toutes les instances de la room indiquée.
`room_tile_add(ind,back,left,top,width,height,x,y,depth)` Ajoute une nouvelle tuile à la room à la position indiquée. La commande retourne l'index de la tuile. **back** correspond à l'arrière-plan à partir duquel la tuile est prise. **left**, **top**, **width** et **height** indiquent la partie de l'arrière-plan qui forme la tuile. **x,y** est la position de la tuile dans la room et **depth** la profondeur de la tuile.
`room_tile_add_ext(ind,back,left,top,width,height,x,y,depth,xscale,yscale,alpha)` Identique à la routine précédente mais cette fois, vous avez la possibilité d'indiquer un facteur d'échelle de direction **x** et **y** ainsi qu'une transparence **alpha** pour la tuile.
`room_tile_clear(ind)` Supprime toutes les tuiles de la room mentionnée.

Fichiers, registres et exécution de programmes

Dans les jeux les plus avancés, vous souhaiterez certainement lire des données à partir d'un fichier que vous mettrez à disposition de votre jeu. Ou encore, vous désirerez stocker des informations qui seront accessibles lors des différentes exécutions du jeu. De plus, dans certaines situations, vous aurez peut-être aussi besoin d'exécuter des programmes externes.

Vous trouverez des informations sur le sujet dans les pages suivantes :

Fichiers

Il est souvent utile d'utiliser des fichiers externes dans les jeux. Par exemple, vous pourriez vouloir créer un fichier décrivant à quels moments certaines choses doivent se passer dans votre jeu. Vous pourriez également souhaiter sauvegarder des informations qui seront utilisées lors d'une prochaine exécution du jeu (par exemple, sauvegarde de la room courante). Les fonctions suivantes permettent de lire et d'écrire des données dans les fichiers de type texte :

`file_text_open_read(fname)` Ouvre en lecture le fichier de nom **fname**. La fonction retourne l'id du fichier qui devra être ensuite utilisé dans les autres fonctions. Il vous est possible d'ouvrir plusieurs fichiers en même temps (**32** au maximum). N'oubliez pas de fermer les fichiers dès que vous n'en avez plus besoin.
`file_text_open_write(fname)` Ouvre en écriture le fichier de nom **fname**, créant ce dernier si celui-ci n'existe pas déjà. La fonction retourne l'id du fichier créé qui devra ensuite être utilisé dans les autres fonctions.
`file_text_open_append(fname)` Ouvre en mode ajout le fichier de nom **fname**, créant ce dernier si celui-ci n'existe pas déjà. La fonction retourne l'id du fichier qui devra ensuite être utilisé dans les autres fonctions.
`file_text_close(fileid)` Ferme le fichier d'id **fileid**.
`file_text_write_string(fileid,str)` Ecrit la chaîne **str** dans le fichier d'id **fileid**.
`file_text_write_real(fileid,x)` Ecrit le nombre réel **x** dans le fichier d'id **fileid**.
`file_text_writeln(fileid)` Ecrit un caractère *newline* (passage à la ligne suivante) dans le fichier d'id **fileid**.
`file_text_read_string(fileid)` Lit une chaîne à partir du fichier d'id **fileid** puis retourne la chaîne lue.
`file_text_read_real(fileid)` Lit un nombre réel à partir du fichier d'id **fileid** puis retourne la valeur lue.
`file_text_readln(fileid)` Saute le reste de la ligne courante du fichier d'id **fileid** puis se place au début de la ligne suivante.
`file_text_eof(fileid)` Indique si l'on a atteint la fin du fichier.

Vous pouvez utiliser les fonctions suivantes pour manipuler les fichiers du système d'exploitation :

file_exists(fname) Indique si le fichier de nom **fname** existe (**true**) ou pas (**false**).
file_delete(fname) Supprime le fichier **fname**.
file_rename(oldname,newname) Renomme le fichier *oldname* en ***newname***.
file_copy(fname,newname) Copie le fichier **fname** dans un nouveau fichier de nom ***newname***.
directory_exists(dname) Retourne si le répertoire indiqué **dname** existe.
directory_create(dname) Crée le répertoire **dname** (avec le chemin complet) si ce dernier n'existe pas déjà.
file_find_first(mask,attr) Retourne le nom du premier fichier satisfaisant le filtre **mask** et les attributs **attr**. Une chaîne vide sera retournée si la recherche n'a pas abouti. Le filtre **mask** peut contenir le nom d'un chemin mais aussi des caractères jokers comme par exemple *'C:\temp*.doc'*. Les attributs vous permettent de fournir les fichiers supplémentaires que vous souhaitez voir (les fichiers '*normaux*' seront toujours retournés s'ils satisfont le masque). Vous pouvez utiliser les constantes suivantes afin de rechercher le type de fichiers souhaité :

fa_readonly fichiers en lecture seule
fa_hidden fichiers cachés
fa_sysfile fichiers systèmes
fa_volumeid fichier d'id de volumes
fa_directory répertoires
fa_archive fichiers archives

file_find_next() Retourne le nom du prochain fichier satisfaisant le masque et les attributs précédemment fournis. Une chaîne vide sera retournée si aucun fichier n'a été trouvé.
file_find_close() Afin de libérer de la mémoire, cette commande doit être invoquée après manipulation des fichiers.
file_attributes(fname,attr) Indique si le fichier vérifie tous les attributs **attr**. Vous pouvez utiliser toute combinaison de constantes vues précédemment.

Les fonctions suivantes peuvent être utilisées pour modifier les noms de fichiers. Veuillez noter que ces fonctions ne fonctionnent pas directement avec les fichiers mais uniquement par le biais de chaînes de caractères.

filename_name(fname) Retourne le nom du fichier indiqué, avec l'extension mais sans le chemin.
filename_path(fname) Retourne le chemin du nom de fichier indiqué, y compris le caractère *backslash* de fin de chemin.
filename_dir(fname) Retourne le nom de répertoire du nom de fichier indiqué, ce qui normalement est identique au chemin sans le caractère *backslash* de fin de chemin.
filename_drive(fname) Retourne des informations sur le lecteur où est enregistré le fichier de nom **fname**.
filename_ext(fname) Retourne l'extension du nom de fichier indiqué, y compris le point séparateur.
filename_change_ext(fname,newext) Retourne le nom de fichier indiqué, avec l'extension (incluant le point) avec ajout de la nouvelle extension **newext**. En utilisant une chaîne vide comme nouvelle extension, il vous est ainsi possible de supprimer l'extension.

Dans de rares situations, vous voudrez lire des données à partir de fichiers binaires. Les routines suivantes de bas niveau sont réservées à cet usage :

file_bin_open(fname,mod) Ouvre le fichier binaire **fname**. Le paramètre **mode** indique les actions possibles sur ce fichier (**0** = lecture, **1** = écriture, **2** = lecture et écriture). La fonction retourne l'id du fichier devant être utilisé dans les autres fonctions. Vous pouvez ouvrir plusieurs fichiers en même temps (**32** au maximum). N'oubliez pas de fermer les fichiers dès que vous n'en avez plus besoin.
file_bin_rewrite(fileid) Réinitialise le fichier d'id **fileid**, c'est à dire l'efface puis recommence à écrire depuis le début.
file_bin_close(fileid) Ferme le fichier d'id **fileid**.
file_bin_size(fileid) Retourne la taille (en octets) du fichier d'id **fileid**.
file_bin_position(fileid) Retourne la position courante (en octets; **0** correspond à la première position) dans le fichier d'id **fileid**.
file_bin_seek(fileid,pos) Déplace le pointeur courant du fichier à la position indiquée. Pour écrire en fin de fichier, déplacez le pointeur en fin de fichier en utilisant la taille de ce dernier avant d'écrire.
file_bin_write_byte(fileid,byte) Ecrit un octet de données dans le fichier d'id **fileid**.
file_bin_read_byte(fileid) Lit un octet de données à partir du fichier puis retourne cette valeur.

Si le joueur a sélectionné le mode *secure* (mode sécurisé) dans ses préférences, pour de nombreuse routines, il ne sera pas possible d'indiquer un chemin. Seuls les fichiers du répertoire du jeu pourront être modifiés.

Les trois fonctions suivantes peuvent être utilisées pour des opérations de lecture seule :

game_id* Correspond à l'identificateur unique du jeu. Cela peut vous être utile si vous souhaitez disposer d'un nom de fichier unique.
working_directory* Il s'agit ici du répertoire de travail du jeu

(*working directory*) (ne comprend pas le caractère *backslash* de fin de chemin).
temp_directory* Répertoire temporaire créé pour le jeu. Vous pouvez y stocker temporairement des fichiers. Les fichiers de ce répertoire seront supprimés à la fin du jeu.

Dans certaines situations, vous désirerez donner aux joueurs la possibilité de fournir des arguments sous forme de ligne de commandes pour le jeu avec lequel ils jouent (par exemple afin de créer des cheats (astuces de jeu) ou encore des modes spéciaux). Les deux routines suivantes permettent d'exploiter ces arguments.

parameter_count() Retourne le nombre de paramètres de la ligne de commandes (le nom du programme constituant l'un de ces paramètres).
parameter_string(n) Retourne les **n** paramètres de la ligne de commandes. Le premier paramètre correspond à l'index **0**. C'est le nom du programme.

Vous pouvez lire le contenu des variables d'environnement en utilisant la fonction suivante :

environment_get_variable(name) Retourne la valeur (sous forme de chaîne de caractères) de la variable d'environnement de nom **name**.

Registres

Si vous souhaitez stocker de l'information qui demeurent accessibles lors des différentes exécutions du jeu, il existe un mécanisme encore plus simple que d'utiliser un fichier. Vous pouvez utilisez la base de registres. La base de registres est une énorme base de données que Windows maintient à jour afin de conserver toutes sortes de paramètres sur les programmes. Une entrée (une clé de registre) possède un **nom** et une **valeur**. Il est possible d'utiliser comme valeur à la fois des chaînes de caractères et des nombres réels. Les fonctions suivantes sont disponibles :

registry_write_string(name,str) Crée une entrée dans la base de registres en utilisant les nom et chaîne mentionnés.
registry_write_real(name,x) Crée une entrée dans la base de registres en utilisant les nom et nombre réel mentionnés.
registry_read_string(name) Retourne la chaîne de caractères que contient le nom de clé indiqué (le nom doit exister. Dans la négative, une chaîne vide sera retournée).
registry_read_real(name) Retourne la valeur réelle que contient le nom de clé mentionné (le nom doit exister. Dans le cas contraire, le nombre 0 sera retourné).
registry_exists(name) Indique si le nom de clé mentionné existe dans la base de registres.

Les valeurs dans la base de registres sont regroupées sous forme de clés. Toutes les routines ci-dessus travaillent avec des valeurs de clef spécialement créées pour votre jeu. Votre programme peut ainsi obtenir certaines informations sur le système que le jeu pourra exploiter. Vous pouvez également lire les valeurs d'autres clés. Vous pouvez même écrire dans ces clés mais soyez très prudent ! VOUS POUVEZ FACILEMENT DETRUIRE VOTRE SYSTEME en procédant ainsi ! (l'écriture n'est pas autorisée dans le mode sécurisé...) Veuillez noter que les clés sont placées à leur tour dans des groupes. Les routines suivantes travaillent par défaut avec le groupe HKEY_CURRENT_USER. Mais il est possible de changer de groupe racine. Ainsi, par exemple, si vous désirez connaître le répertoire temporaire courant, tapez ceci :

```
path = registry_read_string_ext('\Environment','TEMP');
```

Les fonctions suivantes existent.

registry_write_string_ext(key,name,str) Crée une entrée de clé dans le registre avec les nom et valeur de chaîne fournis.
registry_write_real_ext(key,name,x) Crée une entrée de clé dans le registre avec les nom et valeur réelle fournis.
registry_read_string_ext(key,name) Retourne la valeur de la chaîne dont le nom est contenu dans la clé indiquée. (le nom doit exister. Sinon, une chaîne vide sera retournée).
registry_read_real_ext(key,name) Retourne la valeur du nombre réel dont le nom est contenu dans la clé indiquée. (le nom doit exister. Sinon, le nombre 0 sera retourné).
registry_exists_ext(key,name) Indique si le nom donné existe dans la clé indiquée de la base de registres.
registry_set_root(root) Détermine la racine pour les autres routines. Utilisez les valeurs suivantes :

0 = HKEY_CURRENT_USER
1 = HKEY_LOCAL_MACHINE
2 = HKEY_CLASSES_ROOT
3 = HKEY_USERS

Fichiers INI

La finalité des fichiers INI est de fournir aux programmes un mécanisme standard (via Windows) leur permettant de connaître les réglages les concernant. Les fichiers INI comprennent des sections et chacune de ces sections comportent un certain nombre de couples de valeurs préfixées par un nom ou variable. Par exemple, voici un fichier INI typique :

```
[Form]
Top=100
Left=100
Caption=The best game ever
[Game]
MaxScore=12324
```

Ce fichier comprend deux sections, la première nommée *Form* et la deuxième *Game*. La première section contient trois paires de valeurs. Les deux premières paires possèdent une valeur réelle alors que la troisième contient une valeur chaîne de caractères. De tels fichiers INI sont faciles à créer et à modifier. Vous trouverez les fonctions suivantes dans *Game Maker* pour lire et modifier les fichiers INI.

`ini_open(name)` Ouvre le fichier INI de nom **name**. Le fichier INI doit être enregistré dans le même répertoire que le jeu !
`ini_close()` Ferme le fichier INI actuellement ouvert.
`ini_read_string(section,key,default)` Lit la chaîne de la clé **key** de la section **section**. Si la clé et/ou la section n'existent pas, la valeur **default** sera retournée.
`ini_read_real(section,key,default)` Lit le nombre réel de la clé **key** de la section **section**. Si la clé et/ou la section n'existent pas, la valeur **default** sera retournée.
`ini_write_string(section,key,value)` Ecrit la chaîne **value** dans la clé **key** de la section **section**.
`ini_write_real(section,key,value)` Ecrit le nombre réel **value** dans la clé **key** de la section **section**.
`ini_key_exists(section,key)` Indique si la clé **key** existe dans la section **section**.

ini_section_exists(section) Indique si la section **section** existe.

ini_key_delete(section,key) Supprime la clé **key** de la section **section**.

ini_section_delete(section) Supprime la section **section**.

Exécution de Programmes

Game Maker offre également la possibilité d'exécuter des programmes externes. Deux fonctions sont réservées à cet usage : ***execute_program*** et ***execute_shell***. La fonction *execute_program* lance l'exécution d'un programme, avec éventuellement des arguments. La fonction peut attendre la fin d'exécution du programme lancé (mettant ainsi le jeu en pause) ou bien poursuivre le jeu. La fonction *execute_shell* ouvre un fichier. Ce dernier peut être un quelconque fichier pour lequel il existe une association de définie comme par exemple un fichier HTML, un fichier WORD, etc. Cela peut aussi être un programme exécutable. La fonction n'attendra pas la fin d'exécution et poursuivra l'exécution du jeu en parallèle.

`execute_program(prog,arg,wait)` Exécute le programme **prog** avec les arguments **arg**. **wait** indique si la fonction doit attendre la fin d'exécution du programme lancé.
`execute_shell(prog,arg)` Exécute le programme (ou le fichier) **prog** dans un shell, avec arguments **arg** éventuels.

Ces deux fonctions ne fonctionneront pas si le joueur a sélectionné dans les préférences le mode sécurisé (secure mode). Vous pouvez le vérifier en utilisant la variable suivante qui est en lecture seule :

`secure_mode*` Indique que le jeu fonctionne dans le mode sécurisé (secure mode).

Structures de données

Cette fonctionnalité est seulement disponible dans la version enregistrée de Game Maker.

Dans les jeux, il est souvent nécessaire de stocker de l'information. Par exemple, vous souhaiterez sans doute sauvegarder les listes d'objets que transporte une personne ou encore les endroits qui n'ont pas déjà été visités. Il est possible d'utiliser des tableaux pour parvenir à ce résultat. Cependant, si vous désirez réaliser des choses plus compliquées, comme par exemple le tri de données ou la recherche d'un objet particulier, vous serez obligés d'écrire dans ce cas une longue liste de code en GML ; code dont l'exécution ralentira le jeu.

Pour remédier à cela, *Game Maker* intègre un certain nombre de structures de données auxquelles on peut accéder via des fonctions. Actuellement, six types différents de structures de données sont disponibles : les piles de données (stacks), les files d'attente (queues), les listes (lists), les cartes (maps), les files d'attente prioritaires (priority queues) et les grilles (grids). Chacune de ces structures de données est destinée à un usage bien particulier (voir ci-dessous).

Toutes les structures de données fonctionnent globalement de la même manière. Vous créerez une structure de données à l'aide d'une fonction qui retournera un identificateur sur la structure créée. Vous utiliserez cet **id** pour effectuer toutes opérations sur la structure de données. Une fois ces opérations réalisées, vous devrez détruire la structure de données afin d'économiser de l'espace mémoire. Vous pourrez utiliser autant de structures que vous souhaitez. Toutes les structures peuvent stocker des chaînes de caractères ou des nombres réels.

Veuillez noter que les structures de données et leur contenu ne sont pas sauvegardés lorsque vous sauvez le jeu en utilisant les actions et les fonctions dédiées à cet usage. Si vous utilisez des structures de données et que vous souhaitez les sauvegarder, vous devrez créer vos propres routines à cet effet.

Lors de la comparaison de valeurs, par exemple lors de la recherche dans une carte (map) ou lors du tri dans une liste, *Game Maker* doit décider si deux valeurs sont strictement égales. Pour les chaînes de caractères et les nombres entiers, cela ne pose pas de problème mais en ce qui concerne les nombres réels, et en raison des erreurs d'arrondis, un nombre apparemment égal à une valeur peut facilement devenir non égal à cette dernière lors des calculs. Par exemple, l'expression (5/3)*3 ne donnera pas un résultat égal à 5. Afin de contourner ce problème,

il est nécessaire de définir une précision. Quand l'écart entre deux nombres est inférieur à la précision indiquée, les deux nombres seront considérés comme égaux. Par défaut, une précision

de 0.0000001 (1/10 millionième) est utilisée. Il vous est possible de modifier cette précision en employant la fonction ci-dessous :

`ds_set_precision(prec)` Détermine la précision à utiliser lors des comparaisons entre nombres réels.

Cette précision est utilisée pour toutes les opérations sur les structures de données mais n'intervient pas dans les autres comparaisons en GML !

De l'information sur les structures de données est fournie dans les pages suivantes :

Piles de données (Stacks)
Files d'attente (Queues)
Listes (Lists)
Cartes (Maps)
Files d'attente prioritaires (Priority Queues)
Grilles (Grids)

Piles de données

Une structure de piles de données est appelée structure ***LIFO*** (**L**ast-**I**n **F**irst-**O**ut : Dernier entré Premier sorti). Vous pouvez placer des valeurs sur la pile (opération *push*) puis les enlever de celle-ci en les dépilant (opération *pop*). La valeur la plus récemment placée (*push*) sur la pile sera la première à pouvoir être dépilée (*pop*). Les piles sont souvent utilisées lorsqu'il y a des interruptions à gérer ou lors de l'utilisation de fonctions récursives. Les fonctions suivantes existent et concernent les piles :

`ds_stack_create()` Crée une nouvelle pile. La fonction retourne un identifiant **id** de type entier qui sera à utiliser par toutes les autres fonctions pour manipuler la pile créée. Il vous est possible de créer plusieurs piles.
`ds_stack_destroy(id)` Détruit la pile d'ID **id**, libérant la mémoire utilisée par cette dernière. Ne pas oublier d'utiliser cette fonction lorsque vous en avez terminé avec l'utilisation de cette pile.
`ds_stack_clear(id)` Efface le contenu de la pile d'ID **id**, supprimant ainsi toutes les données qu'elle contenait mais ne la détruit pas.
`ds_stack_size(id)` Retourne le nombre de valeurs contenues dans la pile.
`ds_stack_empty(id)` Indique si la pile est actuellement vide. Cela revient à tester si sa taille équivaut à 0.
`ds_stack_push(id,val)` Stocke (*push*) la valeur donnée **val** au sommet de la pile d'ID **id**.
`ds_stack_pop(id)` Retourne la valeur du sommet de la pile (*pop*) puis supprime cette valeur de la pile.
`ds_stack_top(id)` Retourne la valeur du sommet de la pile (*pop*) mais cette fois-ci ne supprime pas cette valeur de la pile.

Files d'attente (Queues)

Une file d'attente est pratiquement similaire à une pile de données mais fonctionne selon le principe **FIFO** (**F**irst-**I**n **F**irst-**O**ut : Premier Entré Premier Sorti). La valeur insérée en premier lieu dans la file d'attente sera donc la première à y être enlevée. Cela correspond au fonctionnement d'une file d'attente dans un magasin. La première personne de la file sera servie en premier. Les files d'attente sont généralement utilisées pour mémoriser des choses restant à faire mais il existe bien d'autres usages possible. Les fonctions suivantes existent (veuillez noter que les cinq premières sont équivalentes aux fonctions des piles de données : toutes les structures de données présentent d'ailleurs ces cinq fonctions).

`ds_queue_create()` Crée une nouvelle file d'attente. La fonction retourne un entier représentant l'ID de la file d'attente et devant être utilisé par toutes fonctions devant manipuler cette file. Plusieurs files d'attente peuvent être créées.
`ds_queue_destroy(id)` Détruit la file d'attente d'ID **id**, libérant ainsi la mémoire utilisée. Ne pas oublier d'invoquer cette fonction lorsque vous aurez terminé de travailler avec la file.
`ds_queue_clear(id)` Efface le contenu de la file d'attente d'ID **id**, supprimant ainsi toutes les données qu'elle contient mais ne détruit pas la file.
`ds_queue_size(id)` Retourne le nombre de valeurs que contient la file d'attente.
`ds_queue_empty(id)` Indique si la file d'attente est vide. Cela revient à tester si sa taille est égale à 0.
`ds_queue_enqueue(id,val)` Insère la valeur **val** en fin de la file d'attente d'ID **id**.
`ds_queue_dequeue(id)` Retourne la valeur la plus ancienne de la file d'attente (= la première valeur de la file) puis la supprime de cette dernière.
`ds_queue_head(id)` Retourne la valeur en début de file d'attente, c'est à dire la valeur la plus ancienne mais ne la supprime pas de la file.
`ds_queue_tail(id)` Retourne la valeur en fin de file d'attente, c'est à dire la valeur la plus récente mais ne la supprime pas de la file.

Listes (Lists)

Une liste permet de stocker une collection de valeurs dans un ordre particulier. Dans cette liste, il vous est possible d'ajouter des valeurs en fin de liste ou encore d'en insérer à un endroit quelconque de la liste. Vous adressez les valeurs de la liste via un index. De plus, vous pouvez trier les éléments de la liste, de manière ascendante ou descendante. Les listes sont utilisées dans différents cas, par exemple pour mémoriser des collections de valeurs destinées à être modifiées. Intrinsèquement, elles sont implémentées sous la forme de tableaux mais, comme la gestion des listes est intégrée dans du code compilé, cela demeure beaucoup plus rapide que si vous deviez utiliser par vous-même des tableaux. Les fonctions suivantes sont à votre disposition :

ds_list_create() Crée une nouvelle liste. La fonction retourne un entier identifiant la liste qui devra ensuite être utilisé dans toutes les fonctions devant accéder à la liste créée.
ds_list_destroy(id) Détruit la liste d'ID **id**, libérant ainsi la mémoire utilisée. N'oubliez pas d'appeler cette fonction lorsque vous aurez terminé d'utiliser la liste concernée.
ds_list_clear(id) Efface le contenu de la liste d'ID **id**, supprimant toutes les données qu'elle contient mais ne supprime pas la liste.
ds_list_size(id) Retourne le nombre de valeurs stockées dans la liste.
ds_list_empty(id) Indique si la liste est vide. Cela équivaut à tester si sa taille est à 0.
ds_list_add(id,val) Ajoute la valeur **val** en fin de liste d'ID **id**.
ds_list_insert(id,pos,val) Insère la valeur **val** à la position **pos** dans la liste d'ID **id**. La première valeur dans la liste se situe à la position 0, la position de la dernière valeur équivaut à la taille de la liste moins 1.
ds_list_replace(id,pos,val) Remplace la valeur à la position **pos** dans la liste d'ID **id** avec la nouvelle valeur **val**.
ds_list_delete(id,pos) Supprime la valeur à la position **pos** dans la liste d'ID **id** (la position 0 correspond au premier élément de la liste).
ds_list_find_index(id,val) Recherche à quelle position se trouve la valeur indiquée **val** dans la liste d'ID **id**. Si aucune valeur n'est trouvée dans la liste, la valeur **-1** sera retournée.
ds_list_find_value(id,pos) Retourne la valeur mémorisée à la position indiquée **pos** dans la liste d'ID **id**.
ds_list_sort(id,ascend) Trie les valeurs de la liste. Si le paramètre **ascend**

est positionné à *true* (**VRAI**) alors les valeurs seront triées selon l'ordre ascendant, sinon c'est l'ordre de tri descendant qui sera employé.

Cartes (Maps)

Dans quelques situations, il vous sera nécessaire de mémoriser des couples de valeurs comprenant une *clé* et sa *valeur* associée. Par exemple, un personnage peut disposer de différents articles et pour chacun d'entre eux en avoir un certain nombre. Dans ce cas précis, l'article sera la clé et le nombre la valeur. Les cartes (*maps*) contiennent de telles paires de valeurs, triées par clé. Vous pouvez ajouter des paires de valeurs dans une carte puis y effectuer une recherche d'une valeur correspondant à des clés particulières. Dû au fait que les clés sont triées dans la carte, il est donc possible de faire une recherche des clés précédentes et suivantes par rapport à une clé donnée. Parfois, il peut être utile également d'utiliser une carte pour stocker uniquement des clés sans aucune valeur associée. Dans cette situation, vous utiliserez simplement une valeur initialisée à 0. Les fonctions suivantes existent :

`ds_map_create()` Crée une nouvelle carte. La fonction retourne l'id de la carte sous forme d'entier, **id** qui devra ensuite être utilisé par toutes les fonctions devant accéder à la carte créée.
`ds_map_destroy(id)` Détruit la carte d'ID **id**, libérant la mémoire utilisée par cette dernière. Ne pas oublier d'appeler cette fonction lorsque vous n'aurez plus besoin d'utiliser une carte particulière.
`ds_map_clear(id)` Efface la carte d'ID **id**, supprimant ainsi toutes les données qu'elle contient mais ne supprime pas la carte.
`ds_map_size(id)` Retourne le nombre de couples clé-valeur contenus dans la carte d'ID **id**.
`ds_map_empty(id)` Indique si la carte est vide. Cela revient à tester si sa taille est égale à 0.
`ds_map_add(id,key,val)` Ajoute le couple clé-valeur donné dans la carte d'ID **id**.
`ds_map_replace(id,key,val)` Remplace la valeur correspondant à la clé **key** par la nouvelle valeur **val**.
`ds_map_delete(id,key)` Supprime la clé **key** et sa valeur attachée de la carte d'ID **id** (en présence de plusieurs entrées avec la même clé, seule la première entrée sera supprimée).
`ds_map_exists(id,key)` Indique si la clé **key** existe dans la carte d'ID **id**.
`ds_map_find_value(id,key)` Retourne la valeur correspondant à la clé **key**.
`ds_map_find_previous(id,key)` Retourne la clé précédent la clé **key** indiquée (veuillez noter que seule la clé est retournée et non pas sa valeur. Vous pouvez

utiliser la routine précédente pour obtenir la valeur correspondant à la clé).

ds_map_find_next(id,key) Retourne la clé suivant la clé **key** indiquée.

ds_map_find_first(id) Retourne la plus petite clé de la carte.

ds_map_find_last(id) Retourne la plus grande clé de la carte.

Files d'attente prioritaires (Priority Queues)

Dans une file d'attente prioritaire, un certain nombre de valeurs sont stockées, avec pour chacune d'entre elles une priorité bien définie. Vous pouvez trouvez rapidement les valeurs de priorité minimale et maximale. En utilisant cette structure de données, il vous sera possible de gérer certaines choses nécessitant un ordre de priorité. Les fonctions suivantes existent :

ds_priority_create() Crée une nouvelle file d'attente prioritaire. La fonction retourne un **id** sous forme d'un entier qui devra par la suite être utilisé par toutes les fonctions devant accéder à la file d'attente créée.
ds_priority_destroy(id) Détruit la file d'attente prioritaire d'ID **id**, libérant la mémoire utilisée par cette file d'attente. N'oubliez pas d'invoquer cette fonction lorsque vous n'aurez plus besoin de cette file d'attente.
ds_priority_clear(id) Efface la file d'attente prioritaire d'ID **id**, supprimant toutes les données qu'elle contient mais ne supprime pas la file d'attente.
ds_priority_size(id) Retourne le nombre de valeurs stockées dans la file d'attente prioritaire d'ID **id**.
ds_priority_empty(id) Indique si la file d'attente prioritaire est vide. Cela équivaut à tester si sa taille est à 0.
ds_priority_add(id,val,prio) Ajoute la valeur **val** de priorité **prio** dans la file d'attente prioritaire d'ID **id**.
ds_priority_change_priority(id,val,prio) Change la priorité **prio** de la valeur **val** appartenant à la file d'attente prioritaire d'ID **id**.
ds_priority_find_priority(id,val) Retourne la priorité de la valeur **val** de la file d'attente prioritaire d'ID **id**.
ds_priority_delete_value(id,val) Supprime la valeur **val** (y compris sa priorité) de la file d'attente prioritaire d'ID **id**.
ds_priority_delete_min(id) Retourne la valeur possédant la plus petite priorité puis la supprime de la file d'attente prioritaire d'ID **id**.
ds_priority_find_min(id) Retourne la valeur possédant la plus petite priorité mais ne la supprime pas de la file d'attente prioritaire d'ID **id**.
ds_priority_delete_max(id) Retourne la valeur possédant la plus grande priorité puis la supprime de la file d'attente prioritaire d'ID **id**.
ds_priority_find_max(id) Retourne la valeur possédant la plus grande priorité mais ne la supprime pas de la file d'attente prioritaire d'ID **id**.

Grilles (Grids)

Une grille est tout simplement un tableau à deux dimensions. Une grille présente une largeur et une hauteur définies par des entiers. La structure de la grille vous permet de déterminer et supprimer les valeurs de cellules de la grille en fournissant un index (devant débuter par 0 et ce pour les deux directions *x* et *y*). Mais il est également possible de déterminer des plages de valeurs, ajouter des valeurs particulières, de déterminer la somme, le maximum, le minimum ainsi que la valeur moyenne d'une plage donnée. Une grille peut être utile pour représenter par exemple un terrain de jeu. Bien que toutes ces fonctionnalités puissent également être réalisées à l'aide de tableaux à deux dimensions, les opérations sur des régions ou plages sont réalisées beaucoup plus rapidement en employant des grilles. Les fonctions suivantes existent :

ds_grid_create(w,h) Crée une nouvelle grille de largeur **w** et de hauteur **h**. La fonction retourne un entier représentant l'**id** de la grille qui devra être utilisé par toutes les fonctions devant accéder à la nouvelle grille créée.
ds_grid_destroy(id) Détruit la grille d'ID **id**, libérant ainsi la mémoire utilisée. Ne pas oublier d'invoquer cette fonction quand vous n'aurez plus besoin d'utiliser cette grille.
ds_grid_resize(id,w,h) Redimensionne la grille d'ID **id** avec les nouvelles largeur **w** et hauteur **h**. Les cellules déjà existantes conserveront leur valeur originale.
ds_grid_width(id) Retourne la largeur de la grille d'ID **id**.
ds_grid_height(id) Retourne la hauteur de la grille d'ID **id**.
ds_grid_clear(id,val) Initialise le contenu de la grille d'ID **id**, avec la valeur indiquée **val** (*val* peut être aussi bien un nombre qu'une chaîne).
ds_grid_set(id,x,y,val) Initialise la cellule indiquée **x,y** de la grille d'ID **id** avec la valeur mentionnée **val** (qui peut être un nombre ou une chaîne).
ds_grid_add(id,x,y,val) Ajoute la valeur **val** dans la cellule indiquée **x,y** de la grille d'ID **id**. En ce qui concerne les chaînes, cela correspond à une concaténation.
ds_grid_multiply(id,x,y,val) Multiplie la valeur **val** par la cellule indiquée **x,y** de la grille d'ID **id**. Cette fonction ne concerne uniquement que les nombres.
ds_grid_set_region(id,x1,y1,x2,y2,val) Initialise toutes les cellules d'une région donnée **x1,y1,x2,y2** de la grille d'ID **id** avec la valeur indiquée **val** (concerne aussi bien les nombres que les chaînes de caractères).
ds_grid_add_region(id,x1,y1,x2,y2,val) Ajoute la valeur **val** aux cellules de la plage **x1,y1,x2,y2** dans la grille d'ID **id**. En ce qui concerne les chaînes, cela correspond à une concaténation.

ds_grid_multiply_region(id,x1,y1,x2,y2,val) Multiplie par la valeur **val** les cellules de la plage **x1,y1,x2,y2** de la grille d'ID **id**. Cette fonction ne concerne uniquement que les nombres.

ds_grid_set_disk(id,xm,ym,r,val) Initialise avec la valeur **val** toutes les cellules du disque avec comme centre **xm,ym** et rayon **r**.

ds_grid_add_disk(id,xm,ym,r,val) Ajoute la valeur **val** à toutes les cellules du disque de centre **xm,ym** et de rayon **r**.

ds_grid_multiply_disk(id,xm,ym,r,val) Multiplie par la valeur **val** toutes les cellules du disque de centre **xm,ym** et de rayon **r**.

ds_grid_get(id,x,y) Retourne la valeur de la cellule indiquée **x,y** de la grille d'ID **id**.

ds_grid_get_sum(id,x1,y1,x2,y2) Retourne la somme des valeurs de cellules de la plage **x1,y1,x2,y2** de la grille d'ID **id**. Ne fonctionne que si les cellules ne contiennent que des nombres.

ds_grid_get_max(id,x1,y1,x2,y2) Retourne le maximum des valeurs de cellules de la plage **x1,y1,x2,y2** de la grille d'ID **id**. Ne fonctionne que si les cellules ne contiennent que des nombres.

ds_grid_get_min(id,x1,y1,x2,y2) Retourne le minimum des valeurs de cellules de la plage **x1,y1,x2,y2** de la grille d'ID **id**. Ne fonctionne que si les cellules ne contiennent que des nombres.

ds_grid_get_mean(id,x1,y1,x2,y2) Retourne la moyenne des valeurs de cellules de la plage **x1,y1,x2,y2** de la grille d'ID **id**. Ne fonctionne que si les cellules ne contiennent que des nombres.

ds_grid_get_disk_sum(id,xm,ym,r) Retourne la somme des valeurs des cellules du disque.

ds_grid_get_disk_min(id,xm,ym,r) Retourne le minimum des valeurs des cellules du disque.

ds_grid_get_disk_max(id,xm,ym,r) Retourne le maximum des valeurs des cellules du disque.

ds_grid_get_disk_mean(id,xm,ym,r) Retourne la moyenne des valeurs des cellules du disque.

ds_grid_value_exists(id,x1,y1,x2,y2,val) Indique si la valeur **val** apparaît quelque part dans la plage **x1,y1,x2,y2**.

ds_grid_value_x(id,x1,y1,x2,y2,val) Retourne l'abscisse **x** de la cellule dans laquelle la valeur **val** apparaît dans la plage **x1,y1,x2,y2**.

ds_grid_value_y(id,x1,y1,x2,y2,val) Retourne l'ordonnée **y** de la cellule dans laquelle la valeur **val** apparaît dans la plage **x1,y1,x2,y2**.

ds_grid_value_disk_exists(id,xm,ym,r,val) Indique si la valeur **val** apparaît quelque part dans le disque.

ds_grid_value_disk_x(id,xm,ym,r,val) Retourne l'abscisse **x** de la cellule dans laquelle la valeur **val** apparaît dans le disque.

ds_grid_value_disk_y(id,xm,ym,r,val) Retourne l'ordonnée **y** de la cellule dans laquelle la valeur **val** apparaît dans le disque.

Création de particules

Cette fonctionnalité est uniquement disponible dans la version enregistrée de Game Maker.

Les systèmes de particules sont destinés à la création d'effets spéciaux. Les particules sont de petits éléments, représentés par un sprite de dimension réduite. De telles particules se déplacent selon des règles prédéfinies et peuvent changer de taille, d'orientation, de couleurs, etc. pendant leur déplacement. Plusieurs de ces particules réunies ensemble pourront créer par exemple des feux d'artifice, des flammes, des explosions, de la pluie, de la neige, des champs d'étoiles, des débris projetés, etc.

Game Maker contient un système de particules sophistiqué pouvant être employé pour créer de grands effets spéciaux. En raison de son aspect général, ce système n'est pas simple à utiliser. Aussi, il est fortement recommandé de lire ce paragraphe avec attention avant de tenter d'utiliser le système.

Si cela vous paraît de prime abord trop compliqué, sachez qu'il existe aussi en base un mécanisme très simple permettant de créer différents types d'explosions, de fumées, de pluies et même de feux d'artifice.

Les systèmes de particules présentent de nombreux paramètres et il n'est pas toujours simple de comprendre la façon de créer les effets souhaités. En premier lieu, nous trouvons les ***types*** *de particules*. Un type de particules définit un genre particulier de particules. Ces types possèdent de nombreux paramètres décrivant notamment la forme, la taille, la couleur et le mouvement des particules. Les types de particules doivent être définis une seule et unique fois pour pouvoir ensuite être utilisés à n'importe quel endroit du jeu.

En second lieu, nous avons les ***systèmes*** *de particules* à proprement parlé. Il peut y avoir différents systèmes de particules dans le jeu. Un système de particules peut avoir des particules de différents types. Un tel système possède des émetteurs (emitters) dont la fonction est de créer les particules, soit de manière continue soit en rafales. On peut trouver également des attracteurs (attractors) qui attirent les particules. Enfin, nous trouvons les destructeurs (destroyers) qui justement ont pour mission de détruire les particules. Dès que les particules ont été créées dans un système de particules, elles seront gérées automatiquement (mises à jour et affichage) par le système de particules.

Des informations sur les particules sont disponibles dans les pages suivantes :

Effets Simples

La façon la plus simple de créer des particules est encore d'utiliser le mécanisme intégré de création d'effets. Les effets sont créés à l'aide du système de particules mais vous n'êtes pas dans l'obligation de vous occuper de tous les détails. Vous indiquerez ici simplement le type d'effet souhaité, la position où il doit être créé, sa taille et sa couleur. Et c'est tout !

Voici les différents genres d'effets mis à votre disposition :

- `ef_explosion`
- `ef_ring`
- `ef_ellipse`
- `ef_firework`
- `ef_smoke`
- `ef_smokeup`
- `ef_star`
- `ef_spark`
- `ef_flare`
- `ef_cloud`
- `ef_rain`
- `ef_snow`

Parfois, vous pourrez créer certains effets en une seule étape (comme une explosion) mais à d'autres moments et pour d'autres effets, il sera nécessaire de les créer en plusieurs étapes (comme pour la fumée ou la pluie). Veuillez noter que la pluie ou encore la neige seront toujours créées en haut de la room. Aussi, la position n'a pas d'importance dans ce cas.

Même si ce mécanisme vous paraît somme toute assez limité, il est cependant suffisant pour créer déjà de grands effets. Par exemple, en créant à chaque step un petit filet de fumée rouge sous un vaisseau spatial en déplacement, un début d'incendie sera créé. Les deux fonctions suivantes existent pour créer des effets:

> **`effect_create_below(kind,x,y,size,color)`** Crée un effet du type indiqué **kind** (voir ci-dessus) à la position **x,y** indiquée. **size** fournit la taille comme suit : **0** = petit, **1** = moyen, **2** = grand. **color** indique la couleur à utiliser. L'effet est créé par-dessous les instances, c'est à dire à une profondeur de *100 000*.

effect_create_above(kind,x,y,size,color) Identique à la fonction précédente mis à part que cette fois-ci, l'effet est créé par-dessus les instances, c'est à dire à une profondeur de *-100 000*.

Si vous voulez supprimer tous les effets, invoquez la fonction suivante :

effect_clear() Supprime tous les effets.

Types de Particules

Un système de particules a pour fonction de décrire la forme, la couleur, le déplacement, etc. d'un type particulier de particules. Il ne sera nécessaire de définir qu'une seule fois le type de particules dans votre jeu. Ceci fait, vous pourrez l'utiliser dans n'importe quel système de particules que comporte le jeu. Les types de particules présentent de nombreux paramètres vous permettant de modifier tous leurs aspects. En paramètrent les types de particules, vous serez à même de créer la plupart des effets possibles. Nous aborderons ultérieurement leur paramétrage.

Un certain nombre de routines sont à votre disposition afin que vous puissiez créer de nouveaux types de particules mais également les détruire :

part_type_create() Crée un nouveau type de particules. Cette commande retourne l'index du type de particule créé. Cet index devra ensuite être utilisé par tous les appels de fonctions (voir ci-dessous) afin de régler les propriétés du type de particules. Aussi, il sera souvent utile de sauvegarder cet index dans une variable globale.
part_type_destroy(ind) Détruit le type de particules d'index **ind**. Appelez cette fonction si vous ne souhaitez plus utiliser ce type de particule. Cela permettra d'économiser de l'espace mémoire.
part_type_exists(ind) Indique si le type de particules d'index **ind** existe.
part_type_clear(ind) Réinitialise le type de particules d'index **ind** à ses valeurs par défaut.

La forme d'une particule

Chaque particule possède une forme (*shape*). Cette forme est représentée en base par un sprite. Vous pouvez utiliser un sprite quelconque pour la définition de vos particules mais sachez qu'il existe déjà 15 sprites réservés à cet usage. La forme d'une particule doit présenter une taille de 64x64 pixels et possède des valeurs *alpha* déjà prédéfinies, de manière à ce que la particule s'accorde le plus harmonieusement possible avec l'arrière-plan. Les constantes suivantes concernent les formes :

- `pt_shape_pixel`
- `pt_shape_disk`
- `pt_shape_square`
- `pt_shape_line`

- pt_shape_star
- pt_shape_circle
- pt_shape_ring
- pt_shape_sphere
- pt_shape_flare
- pt_shape_spark
- pt_shape_explosion
- pt_shape_cloud
- pt_shape_smoke
- pt_shape_snow

Vous pouvez paramétrer la forme d'une particule en utilisant la fonction suivante :

> **part_type_shape(ind,shape)** Détermine la forme d'un type de particule en utilisant l'une des constantes vues plus haut (la forme par défaut est *pt_shape_pixel*).

Vous pouvez également utiliser votre propre sprite afin de définir la forme de la particule. Si le sprite comporte plusieurs sous-images, vous devrez alors préciser ce qui doit être fait avec ces dernières. Vous pourrez choisir aléatoirement une image, animer le sprite, démarrer au début de l'animation ou à un endroit aléatoire, etc. Vous utiliserez la fonction suivante à cet effet.

> **part_type_sprite(ind,sprite,animat,stretch,random)** Affecte votre propre sprite pour le type de particule d'index **ind**. Avec le paramètre **animate**, vous préciserez si le sprite doit être animé (**1**) ou pas (**0**). L'argument **stretch** (**1** ou **0**) indique si l'animation doit être étendue au-delà de la durée de vie de la particule. Avec le paramètre **random** (**1** ou **0**), vous déterminerez si une sous-image aléatoire doit être choisie comme image de départ.

Une fois le sprite choisi pour le type de particule (soit votre sprite ou encore la forme par défaut *shape*), vous devrez indiquer la taille de celui-ci. Une taille de **1** correspond à la taille normale du sprite. Un type de particule peut être défini de telle manière que toutes les particules puissent présenter la même taille ou encore avoir des tailles différentes. Vous pouvez indiquer une plage comprenant plusieurs tailles. Dans ce dernier cas, il vous sera possible d'indiquer si la taille doit changer au-delà de la durée de vie de la particule et si une certaine agitation dans la taille doit survenir afin de créer un effet de clignotement.

> **part_type_size(ind,size_min,size_max,size_incr,size_wiggle)** Fixe les paramètres de taille pour le type de particule. Vous spécifierez la taille de départ minimale, la taille de départ maximale, si la taille doit augmenter à chaque

step (utilisez un nombre négatif pour diminuer la taille) et la valeur d'agitation ou de tremblement (la taille par défaut est de **1** et par défaut, la taille ne change pas).
part_type_scale(ind,xscale,yscale) Détermine l'échelle horizontale et verticale. Ce facteur est multiplié par la taille. Ceci est en particulier très utile lorsque vous avez besoin d'utiliser des échelles différentes dans les directions **x** et **y**.

Les particules peuvent avoir aussi une orientation. De nouveau, l'orientation peut être la même pour toutes les particules ou peut être différente pour chacune d'entre elles, peut changer au-delà de la durée de vie du sprite. Les angles indiquent des rotations exprimées en degrés dans le sens contraire des aiguilles d'une montre.

part_type_orientation(ind,ang_min,ang_max,ang_incr,ang_wiggle,ang_relative) Régler les propriétés de l'angle d'orientation pour le type de particule d'index **ind**. Vous indiquerez l'angle minimum, l'angle maximum, l'incrément à chaque step et la valeur d'agitation dans l'angle (par défaut, toutes ces valeurs sont à **0**). Vous pourrez également indiquer si l'angle donné doit être relatif (**1**) ou absolu (**0**) par rapport à la direction actuelle du déplacement. Par exemple, en paramètrent toutes les valeurs à **0** mais en fixant *ang_relative* à **1**, l'orientation de la particule suivra précisément le chemin de la particule.

Couleur et mélange

Les particules ont une couleur. Il y a différentes manières de préciser les couleurs que doit avoir une particule. La façon la plus simple est d'indiquer une couleur unique. Vous pouvez également préciser deux ou trois couleurs avec lesquelles la couleur de la particule sera interpolée pendant toute sa durée de vie. Par exemple, la particule peut débuter *blanche* puis devenir de plus en plus *noire* et ce jusqu'à la fin de sa durée de vie. Une autre possibilité sera d'indiquer que la couleur de chacune des particules doit être différente et sélectionnée à partir d'une gamme de couleurs. Vous pouvez aussi fournir une plage de couleurs pour le rouge, le vert et le bleu ou encore une gamme de tonalité ou de saturation.

Par défaut, la couleur sera *blanche*. Habituellement, vous utiliserez un sprite possédant ses propres couleurs. Dans ce cas, aucune couleur ne devra alors être précisée.

part_type_color1(ind,color1) Indique une couleur unique à utiliser pour la particule.
part_type_color2(ind,color1,color2) Spécifie deux couleurs entre lesquelles la couleur de la particule sera interpolée.

part_type_color3(ind,color1,color2,color3) Identique à la commande précédente mais cette fois-ci, la couleur sera interpolée en utilisant trois couleurs, de la manière suivante : la première couleur au début, la deuxième couleur à mi-chemin et enfin la troisième couleur à la fin.
part_type_color_mix(ind,color1,color2) Avec cette fonction, vous indiquerez que la particule doit avoir une couleur qui sera un mélange aléatoire des deux couleurs indiquées. La couleur déterminée, elle demeurera invariable pendant toute la durée de vie de la particule.
part_type_color_rgb(ind,rmin,rmax,gmin,gmax,bmin,bmax) Peut être utilisé pour indiquer que chaque particule doit avoir une couleur fixe, déterminée à partir d'une gamme de couleurs. Vous indiquerez une plage de couleurs pour le composant rouge, vert et bleu de la couleur (chaque valeur allant de **0** à **255**).
part_type_color_hsv(ind,hmin,hmax,smin,smax,vmin,vmax) Peut être utilisé pour indiquer que chaque particule doit avoir une couleur fixe mais déterminée à partir d'une plage de couleurs. Vous préciserez une plage de couleurs pour le composant de saturation et de tonalité de la couleur (chaque valeur allant de **0** à **255**).

En plus de la couleur, vous indiquerez également une valeur de transparence *alpha*. Les formes de particules intégrées possèdent déjà une certaine transparence *alpha* mais vous pourrez utiliser ces paramètres pour par exemple faire en sorte que la particule disparaisse à la fin de sa vie.

part_type_alpha1(ind,alpha1) Détermine un paramètre unique de transparence **alpha** (0-1) pour le type de particule.
part_type_alpha2(ind,alpha1,alpha2) Similaire à la commande précédente mais cette fois-ci, des valeurs de début et de fin seront à préciser. La valeur **alpha** sera alors déterminée par interpolation entre ces 2 valeurs.
part_type_alpha3(ind,alpha1,alpha2,alpha3) Cette fois-ci, trois valeurs seront à fournir avec lesquelles la transparence **alpha** sera déterminée par interpolation.

Habituellement, les couleurs des particules seront mélangées avec celles de l'arrière-plan de la même façon que pour les sprites. Mais il est aussi possible d'utiliser un mélange de type additif. Cela peut donner un effet très intéressant, en particulier pour les explosions.

part_type_blend(ind,additive) Indique si l'on doit utiliser le mélange additif (**1**) ou bien un mélange classique (**0**) pour le type de particule.

Vie et mort des particules

Les particules ont une durée de vie limitée, c'est ce que l'on appelle leur temps de vie. Après ce temps, elles disparaîtront. La durée de vie des particules se mesure en ***steps***. Vous pourrez indiquer la durée de vie (ou une plage de durées de vie) pour chaque type de particule. Les particules peuvent créer de nouvelles particules de types différents. Il y a deux manières de procéder. Soit elles créent de nouvelles particules *à chaque step* , soit elles créent des particules *à leur mort*. Mais soyez prudent afin que le nombre total de particules ne devienne pas trop élevé !

`part_type_life(ind,life_min,life_max)` Détermine les limites de durée de vie pour le type de particule (les valeurs par défaut sont de **100**.)
`part_type_step(ind,step_number,step_type)` Détermine le nombre et le type de particules devant être générés à chaque step pour le type de particules indiqué. Si vous employez une valeur négative, une particule sera générée à chaque step avec une chance égale à ***-1/nombre indiqué***. Ainsi, une valeur de **-5** fera qu'une seule particule sera générée en moyenne tous les 5 steps.
`part_type_death(ind,death_number,death_type)` Détermine le nombre et le type de particules devant être générées lorsqu'une particule du type indiqué vient à mourir. A nouveau, vous pouvez utiliser des nombres négatifs afin de créer une particule avec une certaine probabilité de génération. Veuillez noter que ces particules seront uniquement créées quand la particule meurt à la fin de sa vie, et non pas lorsqu'elle meurt à cause de l'action d'un destructeur (*destroyer* : voir ci-dessous)

Mouvement de particule

Les particules peuvent se mouvoir pendant toute leur durée de vie. Elles peuvent avoir une vitesse de départ (ou une plage de vitesses) et une direction : la vitesse et la direction pouvant changer en permanence. De même, la gravité peut être définie pour attirer les particules dans une direction donnée. Les fonctions suivantes sont destinées à cet usage :

`part_type_speed(ind,speed_min,speed_max,speed_incr,speed_wiggle)` Détermine les propriétés de vitesse pour le type de particule (par défaut, toutes les valeurs sont à **0**). Vous indiquerez une vitesse minimale et maximale. Une valeur aléatoire déterminée entre ces deux valeurs sera choisie lorsque la particule sera créée. Vous pouvez indiquer un incrément de vitesse à utiliser à chaque step. Utilisez un nombre négatif pour ralentir la particule (la vitesse ne devra jamais descendre en dessous de **0**). Enfin, vous indiquerez une certaine valeur d'agitation de la vitesse.

part_type_direction(ind,dir_min,dir_max,dir_incr,dir_wiggle)
Détermine les propriétés de direction pour le type de particule (par défaut, toutes les valeurs sont à **0**). A nouveau, vous pouvez spécifier une plage de directions (exprimées en degrés dans le sens contraire des aiguilles d'une montre; **0** indiquant un déplacement vers la droite). Par exemple, pour que la particule se déplace dans une direction aléatoire, choisissez les valeurs **0** et **360**. Vous pouvez spécifier un incrément pour la direction à chaque step et une valeur d'agitation.
part_type_gravity(ind,grav_amount,grav_dir) Détermine les propriétés de gravité pour le type de particule (par défaut, il n'y a pas de gravité). Vous indiquerez la quantité de gravité devant être ajoutée à chaque step et la direction. Par exemple, utilisez la valeur **270** pour obtenir une direction vers le bas.

Systèmes de Particules

Les particules ont une existence dans les systèmes de particules. Aussi, pour que votre jeu dispose de particules, il est donc nécessaire de créer un ou plusieurs systèmes de particules. Il peut y avoir différents systèmes de particules (mais il est préférable d'en avoir un petit nombre). Par exemple, si votre jeu possède un certain nombre de balles et que chaque balle doit avoir une queue de particules, il est très probable que chacune des balles devra avoir son propre système de particules. La façon la plus simple d'aborder les systèmes de particules est justement d'en créer un puis de créer les particules le composant, en utilisant les types de particules que vous aurez définis auparavant. Mais, comme nous le verrons plus loin, les systèmes de particules peuvent comprendre des émetteurs (emitters) qui produiront automatiquement les particules, des attracteurs (attractors) qui les attireront et des destructeurs (destroyers) qui les détruiront.

Dés lors que des particules ont été ajoutées à un système de particules, elles seront automatiquement mises à jour à chaque *step* puis affichées. Aucune action supplémentaire n'est requise. Afin de permettre que les particules puissent être dessinées derrière, devant ou encore entre des instances d'objets, chaque système de particules possède une profondeur qui est similaire à celle des instances et des tuiles.

Les systèmes de particules, après avoir été créés, présentent une durée de vie illimitée. Ainsi, même si vous changez de room ou que vous relanciez le jeu, les systèmes et leurs particules demeureront toujours présents. Il sera donc préférable de les détruire dés lors que vous n'en aurez plus besoin.

Les fonctions de base suivantes traitent des systèmes de particules :

`part_system_create()` Crée un nouveau système de particules. Cette commande retourne l'index du système créé. Cet index devra être utilisé dans tous les appels de fonctions ci-dessous afin de régler les propriétés du système de particules.
`part_system_destroy(ind)` Détruit le système de particules d'index **ind**. A invoquer si vous ne souhaitez plus utiliser ce système de particules (permet de gagner de l'espace mémoire).
`part_system_exists(ind)` Indique si le système de particules d'index **ind** existe.
`part_system_clear(ind)` Réinitialise le système de particules à ses valeurs par défaut, supprimant toutes les particules, émetteurs et attracteurs du système.

part_system_draw_order(ind,oldtonew) Définit l'ordre dans lequel le système de particules dessine les particules. Si le paramètre **oldtonew** est à **true**, les particules les plus anciennes seront dessinées en premier lieu tandis que les plus récentes seront affichées par-dessus les plus anciennes (par défaut). Sinon, les particules les plus récentes seront dessinées en premier. Ceci permet d'obtenir différents effets.

part_system_depth(ind,depth) Détermine la profondeur du système de particules. Cela peut être utilisé afin de permettre aux particules d'apparaître derrière, au premier plan ou encore entre des instances.

part_system_position(ind,x,y) Détermine la position où le système de particules doit être affiché. Cela n'est habituellement pas nécessaire mais si vous souhaitez avoir des particules à une position relative par rapport à un objet en déplacement, vous pourrez de cette façon régler la position des particules par rapport à cet objet.

Comme indiqué précédemment, le système de particules est mis à jour et affiché automatiquement. Mais parfois, cela ne sera pas ce que vous souhaiterez. Pour vous permettre de contourner cela, vous pouvez régler à **off** la mise à jour et l'affichage automatique et décider alors par vous-même le moment où la mise à jour ou l'affichage du système de particules doit avoir lieu. Vous pourrez utiliser à cet effet les fonctions suivantes :

part_system_automatic_update(ind,automatic) Indique si le système de particules doit être mis à jour automatiquement (**1**) ou pas (**0**). **1** est la valeur proposée par défaut.

part_system_automatic_draw(ind,automatic) Indique si le système de particules doit être affiché automatiquement (**1**) ou pas (**0**). **1** correspond à la valeur par défaut.

part_system_update(ind) Cette fonction met à jour la position de toutes les particules du système et autorise les émetteurs à créer des particules. Vous ne devrez appeler cette fonction uniquement lorsque la mise à jour n'est pas réglée en automatique (bien que parfois il est recommandé d'appeler cette fonction deux fois afin d'obtenir un bon fonctionnement du système).

part_system_drawit(ind) Cette fonction dessine les particules du système. A invoquer lorsque l'affichage ne se fait pas en automatique. Elle devrait être appelée dans l'événement d'affichage (draw event) de certains objets.

Les fonctions suivantes ont trait aux particules des systèmes de particules :

part_particles_create(ind,x,y,parttype,number) Cette fonction crée `number` particules du type indiqué à la position **(x,y)** dans le système.

part_particles_create_color(ind,x,y,parttype,color,number) Cette fonctions crée `number` particules du type indiqué à la position **(x,y)** dans le système avec la couleur indiquée. Cela est seulement utile quand le type de particule utilise une couleur unique (ou n'utilise aucune couleur du tout).

part_particles_clear(ind) Cette fonction supprime toutes les particules du système.

part_particles_count(ind) Cette fonction retourne le nombre de particules du système.

Emetteurs (Emitters)

Les émetteurs (emitters) crée des particules. Ils peuvent soit créer un flux continu de particules, soit projeter par intermittence un certain nombre de particules en utilisant la fonction appropriée. Un système de particules peut avoir un nombre arbitraire d'émetteurs. Un émetteur présente les propriétés suivantes :

- **xmin, xmax, ymin, ymax** Indique l'étendue de la région dans laquelle les particules seront générées.
- **shape** Indique la forme de la région. Elle peut avoir les valeurs suivantes :
  - ps_shape_rectangle
  - ps_shape_ellipse
  - ps_shape_diamond
  - ps_shape_line
- **distribution** Indique la distribution utilisée pour générer les particules. La distribution peut prendre les valeurs suivantes :
  - ps_distr_linear Indique une distribution linéaire, ce qui signifie que la distribution est régulière quelle que soit la zone de la région.
  - ps_distr_gaussian Indique une distribution Gaussienne dans laquelle la plupart des particules seront générées au centre plutôt que sur les côtés de la région.
- **particle type** Indique le type de particules devant être généré.
- **number** Indique le nombre de particules générées à chaque step. Si cette valeur est inférieure à **0**, une particule sera générée à chaque step avec une chance de *-1/number*. Ainsi, par exemple, une valeur de **-5** indiquera qu'une particule sera générée en moyenne tous les 5 steps.

Les fonctions suivantes sont disponibles pour paramétrer les émetteurs et leur permettre ainsi de créer des particules. Veuillez noter que chacune d'entre elles demande l'index du système de particules comme premier argument.

part_emitter_create(ps) Crée un nouvel émetteur pour le système de particules indiqué **ps**. Elle retourne l'index de l'émetteur. Cet index devra être utilisé dans tous les appels de fonctions ci-dessous afin de régler les propriétés de l'émetteur.

part_emitter_destroy(ps,ind) Détruit l'émetteur **ind** du système de particules **ps**. Invoquez cette fonction si vous ne souhaitez plus utiliser l'émetteur afin de gagner de l'espace mémoire.

part_emitter_destroy_all(ps) Détruit tous les émetteurs du système de particules **ps** qui ont été créés.

part_emitter_exists(ps,ind) Retourne si l'émetteur indiqué **ind** existe dans le système de particules **ps**.

part_emitter_clear(ps,ind) Réinitialise l'émetteur **ind** à ses valeurs par défaut.

part_emitter_region(ps,ind,xmin,xmax,ymin,ymax,shape,distribution) Détermine la région et la distribution pour l'émetteur **ind**.

part_emitter_burst(ps,ind,parttype,number) Projette en une seule fois **number** particules du type indiqué **parttype** à partir de l'émetteur **ind**.

part_emitter_stream(ps,ind,parttype,number) A partir de ce moment désormais, crée **number** particules du type indiqué **parttype** avec l'émetteur **ind** à chaque step. Si vous indiquez un nombre inférieur à **0**, une particule sera générée à chaque step avec une chance de *-1/number*. Par exemple, une valeur de **-5** générera en moyenne une particule tous les 5 steps.

Attracteurs (Attractors)

En plus des émetteurs, un système de particules peut également comprendre des attracteurs (attractors). Un attracteur attire les particules (ou les repousse). Un système de particules peut avoir plusieurs attracteurs. Il est cependant recommandé d'en utiliser un petit nombre afin de ne pas trop ralentir le traitement des particules. Un attracteur possède les propriétés suivantes :

- **x,y** Indique la position de l'attracteur.
- **force** Indique la force d'attraction de l'attracteur. La manière dont agit la force sur les particules dépend des paramètres suivants.
- **dist** Indique la distance maximale à partir de laquelle l'attracteur cesse d'avoir des effets. Seules les particules inférieures à cette distance de l'attracteur seront attirées.
- **kind** Indique le type d'attracteur. Les valeurs suivantes existent :
    - `ps_force_constant` Indique si la force doit être constante indépendamment de la distance.
    - `ps_force_linear` Indique une force augmentant de manière linéaire. A la distance maximale, la force sera de **0** alors qu'à la position de l'attracteur, la force atteindra la valeur maximale indiquée.
    - `ps_force_quadratic` Indique que la force augmentera de façon quadratique.
- **additive** Indique si la force doit être ajoutée à la vitesse et à la direction à chaque step (*true*) ou si elle doit être appliquée à la position de la particule (*false*). Dans le cas de l'ajout, la particule accélérera en direction de l'attracteur tandis que dans le cas d'une force non-additive, la particule se déplacera à une vitesse constante.

Les fonctions suivantes existent pour définir les attracteurs. Veuillez noter que chacune de ces fonctions demande l'index du système de particules comme premier argument.

part_attractor_create(ps) Crée un nouvel attracteur dans le système de particules **ps**. Cette fonction retourne l'index de l'attracteur. Cet index sera à utiliser dans tous les appels de fonctions ci-dessous afin de paramétrer les propriétés de l'émetteur.
part_attractor_destroy(ps,ind) Détruit l'attracteur **ind** du système de particules **ps**. Appelez cette fonction si vous n'avez plus besoin de l'attracteur afin d'économiser de l'espace mémoire.
part_attractor_destroy_all(ps) Détruit tous les attracteurs du système de particules **ps** qui ont été créés.

part_attractor_exists(ps,ind) Indique si l'attracteur **ind** existe dans

le système de particules **ps**.

part_attractor_clear(ps,ind) Réinitialise l'attracteur **ind** à ses valeurs par défaut.

part_attractor_position(ps,ind,x,y) Fixe la position de l'attracteur **ind** à la position **(x,y)**.

part_attractor_force(ps,ind,force,dist,kind,aditive) Régler les paramètres de la force de l'attracteur **ind**.

Destructeurs (Destroyers)

Les destructeurs (destroyers) détruisent les particules lorsque ces dernières apparaissent dans leur région. Un système de particules peut avoir un nombre arbitraire de destructeurs. Un destructeur présente les propriétés suivantes :

- **xmin, xmax, ymin, ymax** Indique l'étendue de la région dans laquelle les particules seront détruites.
- **shape** Indique la forme de la région. La forme peut avoir les valeurs suivantes :
  - ps_shape_rectangle
  - ps_shape_ellipse
  - ps_shape_diamond

Les fonctions suivantes sont disponibles afin de paramétrer les propriétés des destructeurs. Veuillez noter que chacune de ces fonctions demande l'index du système de particules comme premier argument.

part_destroyer_create(ps) Crée un nouveau destructeur dans le système de particules **ps**. Cette fonction retourne l'index du destructeur. Cet index devra être employé dans tous les appels de fonction ci-dessous afin de paramétrer les propriétés du destructeur.
part_destroyer_destroy(ps,ind) Détruit le destructeur **ind** du système de particules **ps**. Fonction à invoquer si vous n'avez plus besoin du destructeur afin d'économiser de l'espace mémoire.
part_destroyer_destroy_all(ps) Détruit tous les destructeurs du système de particules **ps** qui ont été créés.
part_destroyer_exists(ps,ind) Indique si le destructeur indiqué **ind** existe dans le système de particules **ps**.
part_destroyer_clear(ps,ind) Réinitialise le destructeur **ind** à ses valeurs par défaut.
part_destroyer_region(ps,ind,xmin,xmax,ymin,ymax,shape) Détermine la région du destructeur **ind**.

Déflecteurs (Deflectors)

Les déflecteurs guident les particules lorsqu'elles apparaissent dans leur région. Veuillez noter que seule la position de la particule est prise en compte et non pas son sprite ou encore la taille de la particule. Un système de particules peut avoir un nombre arbitraire de déflecteurs. Un déflecteur possède les propriétés suivantes :

- **xmin, xmax, ymin, ymax** Indique l'étendue de la région dans laquelle les particules seront guidées.
- **kind** Indique le type de déflecteur. Le type peut avoir les valeurs suivantes :
  - `ps_deflect_horizontal` Guide la particule horizontalement; habituellement employé pour les murs verticaux.
  - `ps_deflect_vertical` Guide la particule verticalement; habituellement utilisé pour les murs horizontaux.
- **friction** La valeur de friction suite à impact de la particule avec le déflecteur. Plus cette valeur est importante et plus la particule sera ralentie lors de l'impact.

Les fonctions suivantes sont disponibles afin de paramétrer les propriétés du déflecteur. Veuillez noter que chacune de ces fonctions demande l'index du système de particules comme premier argument.

part_deflector_create(ps) Crée un nouveau déflecteur dans le système de particules **ps**. Cette fonction retourne l'index du déflecteur. Cet index sera à utiliser dans tous les appels de fonctions ci-dessous afin de paramétrer les propriétés du déflecteur.
part_deflector_destroy(ps,ind) Détruit le déflecteur **ind** du système de particules **ps**. A invoquer si vous n'avez plus besoin du déflecteur afin de libérer de l'espace mémoire.
part_deflector_destroy_all(ps) Détruit tous les déflecteurs du système de particules **ps** qui ont été créés.
part_deflector_exists(ps,ind) Indique si le déflecteur **ind** existe dans le système de particules **ps**.
part_deflector_clear(ps,ind) Réinitialise le déflecteur **ind** à ses valeurs par défaut.
part_deflector_region(ps,ind,xmin,xmax,ymin,ymax) Détermine la région du déflecteur **ind**.
part_deflector_kind(ps,ind,kind) Détermine le type du déflecteur **ind**.

part_deflector_friction(ps,ind,friction) Détermine la friction du déflecteur **ind**.

Changeurs (Changers)

Les changeurs de particules modifient certaines particules lorsqu'elles apparaissent dans leur région. Un système de particules peut avoir un nombre arbitraire de changeurs. Un changeur présente les propriétés suivantes :

- **xmin, xmax, ymin, ymax** Indique l'étendue de la région dans laquelle les particules seront modifiées.
- **shape** Indique la forme de la région. La forme peut avoir les valeurs suivantes :
  - ps_shape_rectangle
  - ps_shape_ellipse
  - ps_shape_diamond
- **parttype1** Indique le type de particule à modifier.
- **parttype2** Indique le type de particule en lequel la particule sera changée.
- **kind** Indique le type de changeur. Le type peut avoir les valeurs suivantes :
  - ps_change_motion Ici, seuls les paramètres de mouvement de la particule pourront changer, et non pas sa couleur, sa forme ou les paramétrages de sa durée de vie.
  - ps_change_shape Seuls les paramètres de forme pourront changer ici comme la taille, la couleur et la forme.
  - ps_change_all Modifie tous les paramètres, cela signifie en base que la particule sera détruite puis une nouvelle particule sera créée dans le nouveau type.

Les fonctions suivantes sont disponibles pour paramétrer les propriétés du changeur de particules. Veuillez noter que chacune de ces fonctions demande l'index du système de particules comme premier argument.

part_changer_create(ps) Crée un nouveau changeur dans le système de particules **ps**. Cette fonction retourne l'index du changeur. Cet index devra être utilisé dans tous les appels de fonctions ci-dessous afin de paramétrer les propriétés du changeur de particule.
part_changer_destroy(ps,ind) Détruit le changeur **ind** du système de particules **ps**. Fonction à appeler si vous n'avez plus besoin du changeur et pour économiser de l'espace mémoire.
part_changer_destroy_all(ps) Détruit tous les changeurs du système de particules **ps** qui ont été créés.

part_changer_exists(ps,ind) Indique si le changeur **ind** existe dans le système de particules **ps**.

part_changer_clear(ps,ind) Réinitialise le changeur **ind** à ses valeurs par défaut.

part_changer_region(ps,ind,xmin,xmax,ymin,ymax,shape) Détermine la région du changeur **ind**.

part_changer_types(ps,ind,parttype1,parttype2) Détermine quel type de particule le changeur doit modifier en un autre type.

part_changer_kind(ps,ind,kind) Fixe le type du changeur **ind**.

Exemple de Feu d'artifices (Firework Example)

Voici un exemple d'un système de particules créant un feu d'artifices. Ce dernier utilise deux types de particules : l'une formant une rocket et l'autre créant à proprement parler le feu d'artifices. La rocket génère les particules du feu d'artifices lorsque celui-ci vient à mourir. Nous générons aussi un émetteur dans le système de particules qui projettera de façon régulière des jets de particules de rocket venant du bas de l'écran. Pour accomplir cette tâche, nous aurons besoin d'un objet. Dans son événement de création, nous placerons le code suivant qui créera les types de particules, le système de particules et l'émetteur :

```
{
  // make the particle system
  ps = part_system_create();

  // the firework particles
  pt1 = part_type_create();
  part_type_shape(pt1,pt_shape_flare);
  part_type_size(pt1,0.1,0.2,0,0);
  part_type_speed(pt1,0.5,4,0,0);
  part_type_direction(pt1,0,360,0,0);
  part_type_color1(pt1,c_red);
  part_type_alpha2(pt1,1,0.4);
  part_type_life(pt1,20,30);
  part_type_gravity(pt1,0.2,270);

  // the rocket
  pt2 = part_type_create();
  part_type_shape(pt2,pt_shape_sphere);
  part_type_size(pt2,0.2,0.2,0,0);
  part_type_speed(pt2,10,14,0,0);
  part_type_direction(pt2,80,100,0,0);
  part_type_color2(pt2,c_white,c_gray);
  part_type_life(pt2,30,60);
  part_type_gravity(pt2,0.2,270);
```

```
  part_type_death(pt2,150,pt1);    // create the firework on
death

  // create the emitter
  em = part_emitter_create(ps);

part_emitter_region(ps,em,100,540,480,490,ps_shape_rectangle,ps
_distr_linear);
  part_emitter_stream(ps,em,pt2,-4);   // create one every four
steps
}
```

Et voilà le travail ! Vous devrez cependant vous assurer que le système de particules (mais également les types de particules) est détruit quand vous vous déplacerez dans une autre room. Dans le cas contraire, le feu d'artifices ne cessera jamais...

Jeux multi-joueurs

Cette fonctionnalité n'est disponible que dans la version enregistrée de Game Maker.

Jouer à des jeux contre l'ordinateur est amusant. Mais jouer à des jeux contre d'autres joueurs humains peut l'être encore plus. D'autre part, la réalisation de ce dernier type de jeux est relativement facile à réaliser, ceci en raison que vous n'avez pas à implémenter une intelligence artificielle compliquée pour gérer l'opposant. Vous pouvez bien entendu jouer à un jeu en étant deux joueurs devant le même écran et utiliser différentes touches ou autres périphériques de saisie, mais c'est beaucoup plus intéressant lorsque les deux joueurs se trouvent chacun aux commandes de leur propre ordinateur. Mieux encore, lorsque chaque joueur se situe de part et d'autre de l'océan. *Game Maker* sait gérer le mode multi-joueurs. Veuillez cependant bien noter que la création de jeux multi-joueurs présentant une bonne synchronisation et des temps de réponses corrects, n'est pas tâche aisée. Ce chapitre vous donne une brève description des possibilités en la matière. Sur le site web, un tutorial est disponible avec davantage d'informations.

De l'information sur les jeux multi-joueurs peut être trouvée dans les pages suivantes :

Paramétrage d'une Connexion

Afin que deux ordinateurs puissent communiquer entre eux, il est nécessaire de disposer d'un protocole de connexion. Comme dans la plupart des jeux, *Game Maker* offre quatre types différents de connexions : IPX, TCP/IP, par Modem et Série. La connexion IPX (pour être plus précis, il s'agit d'un protocole) est pratiquement totalement transparente aux utilisateurs. Elle peut être utilisée pour jouer à des jeux avec d'autres personnes se trouvant sur le même réseau local. Pour être effective, elle doit être installée sur votre ordinateur (dans le cas où cela ne fonctionnerait pas, veuillez consulter la documentation de Windows ou aller dans le panneau de configuration de Windows, rubrique "réseau", puis ajoutez le protocole IPX). TCP/IP est le protocole d'internet. Il peut être utilisé pour jouer avec d'autres joueurs n'importe où sur Internet, à condition que vous connaissiez votre adresse IP. Dans le cas d'un réseau local, il n'est pas obligatoire de fournir des adresses. Une connexion par modem s'effectue par modem comme son nom l'indique. Vous devrez fournir quelques réglages du modem (une chaîne d'initialisation et un numéro de téléphone) avant de pouvoir l'utiliser. Enfin, quand vous utilisez une ligne série (c'est à dire une connexion directe entre les ordinateurs), il est nécessaire de paramétrer un certain nombre de ports. Il existe quatre fonctions GML que l'on peut utiliser pour initialiser ces connexions :

`mplay_init_ipx()` Initialise une connexion IPX.
`mplay_init_tcpip(addr)` Initialise une connexion TCP/IP. **addr** est une chaîne contenant l'adresse du site web ou encore l'adresse IP, par exemple *'www.gameplay.com'* ou *'123.123.123.12'*, éventuellement suivie par un numéro de port (exemple : ':12'). C'est seulement quand vous vous joignez à une session de jeu (voir ci-dessous) qu'il est nécessaire de fournir une adresse. Dans le cas d'un réseau local, il n'est pas nécessaire d'indiquer une adresse IP.
`mplay_init_modem(initstr,phonenr)` Initialise une connexion par modem. **initstr** est la chaîne d'initialisation destinée au modem (peut être vide). **phonenr** correspond à la chaîne contenant le numéro de téléphone à appeler (exemple : *'0201234567'*). Un numéro de téléphone n'est nécessaire qu'à l'instant où vous rejoignez une session (voir ci-dessous).
`mplay_init_serial(portno,baudrate,stopbits,parity,flow)` Initialise une connexion série. **portno** est le numéro de port (1 à 4). **baudrate** est la vitesse en bauds à utiliser (100 à 256K). **stopbits** indique le nombre de bits d'arrêt (**0** = 1 bit, **1** = 1.5 bit, **2** = 2 bits). **parity** indique la parité (**0**=aucune, **1**=impaire, **2**=paire, 3=mark). Et **flow** précise le type de contrôle de flux (**0**=aucun, **1**=xon/xoff, **2**=rts, **3**=dtr, **4**=rts et dtr). Retourne une valeur en cas de succès. Un appel typique sera par exemple : *mplay_init_serial(1,57600,0,0,4)*.

Donnez la valeur **0** comme premier argument pour ouvrir une boîte de dialogue permettant à l'utilisateur de modifier les réglages.

Votre jeu devra invoquer l'une de ces fonctions une seule fois précisément. Toutes les fonctions signalent si la commande a réussi. Ces fonctions se terminent en échec si le protocole particulier n'est pas installé ou n'est pas supporté par votre système. Afin de vérifier si la connexion en cours a réussi, vous pourrez utiliser la fonction suivante :

`mplay_connect_status()` Retourne le statut de la connexion courante. **0** = pas de connexion, **1** = connexion IPX, **2** = connexion TCP/IP, **3** = connexion par modem et **4** = connexion série.

Pour clore la connexion en cours :

`mplay_end()` Ferme la connexion courante.

Si vous utilisez une connexion TCP/IP, vous pourrez vouloir indiquer à la personne avec laquelle vous jouez, l'adresse IP de votre ordinateur. La fonction suivante vous le permettra :

`mplay_ipaddress()` Retourne l'adresse IP de votre machine (exemple : *'123.123.123.12'*) sous la forme d'une chaîne de caractères. Il vous sera par exemple possible d'afficher cette information à un endroit précis de l'écran. Veuillez noter que cette routine est lente cependant, aussi ne l'utilisez pas en permanence.

Créer et se Joindre à des Sessions

Quand vous vous connectez à un réseau, il peut y avoir plusieurs jeux se déroulant actuellement sur le même réseau. Nous appellerons cela des sessions. Ces différentes sessions peuvent correspondre à des jeux différents ou encore au même jeu. Un jeu doit s'identifier de manière unique sur le réseau. Heureusement, *Game Maker* fait ce travail pour vous. La seule chose que vous devez savoir est que lorsque vous modifiez l'ID du jeu dans l'écran des options, l'identification du jeu changera également. De cette manière, vous pourrez ainsi éviter que des personnes avec de vieilles versions de votre jeu ne puissent jouer contre des personnes disposant de la nouvelle version.

Si vous souhaitez lancer un nouveau jeu en mode multi-joueurs, il sera nécessaire de créer une nouvelle session. Vous utiliserez cette routine à cet effet :

`mplay_session_create(sesname,playnumb,playername)` Crée une nouvelle session pour la connexion réseau courante. **sesname** est une chaîne de caractères indiquant le nom de la session. **playnumb** est un nombre identifiant le nombre maximal de joueurs autorisés à jouer à ce jeu (utilisez **0** pour avoir un nombre arbitraire de joueurs). **playname** est votre nom de joueur. Retourne une valeur en cas de succès.

Une instance du jeu doit créer la session. L'autre ou les autres instances du jeu vous permettront de vous joindre à cette session. Ceci est légèrement plus compliqué qu'il n'y parait. Vous devrez en premier lieu vérifier quelles sessions sont disponibles puis en choisir une pour vous joindre à une session. A cet effet, il existe trois routines importantes :

`mplay_session_find()` Recherche toutes les sessions qui acceptent encore des joueurs puis retourne le nombre de sessions trouvées.
`mplay_session_name(numb)` Retourne le nom de la session de numéro **numb** (**0** étant la première session). Cette routine ne peut être appelée qu'après l'appel de la routine précédente.
`mplay_session_join(numb,playername)` Vous permet de rejoindre la session de numéro **numb** (**0** étant la première session). **playername** correspond à votre nom de joueur. Retourne une valeur en cas de succès.

Il existe aussi une routine qui peut changer le mode de session. Elle devra être appelée avant de créer une session:

mplay_session_mode(move) Détermine si oui ou non l'on doit transférer la session cliente sur un autre ordinateur lorsque le client quitte la session. **move** ne peut prendre que les valeurs **true** ou **false** (valeur par défaut).

Pour vérifier le statut de la session courante, vous pourrez utiliser la fonction suivante :

mplay_session_status() Retourne le statut de la session courante. **0** = pas de session, **1** = session créée, **2** = session rejointe.

Un joueur peut arrêter une session en utilisant la routine suivante :

mplay_session_end() Termine la session pour ce joueur.

Joueurs

Chaque instance de jeu que rejoint un joueur est une session. Comme indiqué auparavant, chaque joueur a un nom. Il existe trois routines qui ont trait aux joueurs.

`mplay_player_find()` Recherche tous les joueurs dans la session courante et retourne le nombre de joueurs trouvés.

`mplay_player_name(numb)` Retourne le nom du joueur possédant le numéro **numb** (**0** étant le premier joueur, qui sera toujours vous-même). Cette routine ne peut être appelée qu'après l'appel de la précédente routine.

`mplay_player_id(numb)` Retourne l'unique *id* du joueur de numéro **numb** (**0** est le premier joueur, qui sera toujours vous-même). Cette routine ne peut être appelée qu'après l'appel de la première routine. Cet *id* est utilisé pour envoyer et recevoir des messages vers/en provenance de joueurs individuels.

Données Partagées

La communication de données partagées est certainement la manière la plus simple de synchroniser un jeu. Toute communication fonctionne en mode sécurisé. Il existe un ensemble de 1 000 000 de valeurs communes à toutes les entités du jeu (il est préférable d'utiliser seulement les premières valeurs afin d'économiser de la mémoire). Chaque entité peut écrire et lire des valeurs. *Game Maker* fera en sorte que chacune des entités voit les mêmes valeurs. Une valeur peut être soit un réel soit une chaîne de caractères. Il n'existe que deux routines :

`mplay_data_write(ind,val)` Ecrit la valeur **val** (chaîne ou réel) à la position **ind** (**ind** est compris entre **0** et **1000000**).
`mplay_data_read(ind)` Retourne la valeur à la position **ind** (avec **ind** compris entre **0** et **1000000**). Au départ, toutes les valeurs sont à **0**.

Afin de synchroniser les données sur les différentes machines, vous pouvez utiliser soit le mode garantie qui vous assure que les changements parviendront sur l'autre machine (mais qui est lent) soit le mode non garantie. Pour sélectionner le mode, veuillez utiliser la routine suivante :

`mplay_data_mode(guar)` Détermine si l'on doit utiliser la transmission garantie pour les données partagées. **guar** devra être soit à *true* (par défaut) soit à *false*.

Messages

Le second mécanisme de communication supporté par *Game Maker* est l'envoi et la réception de messages. Un joueur peut envoyer des messages à un ou à tous les autres joueurs. Les joueurs peuvent voir si des messages sont arrivés et réagir en conséquence. Les messages peuvent être envoyés en mode garantie dans lequel on peut être certain qu'ils arriveront à leurs destinataires (mais au prix d'une certaine lenteur) ou en mode non garantie, qui est plus rapide.

Les routines suivantes d'adressage de messages existent :

`mplay_message_send(player,id,val)` Envoie un message au joueur indiqué (soit un identificateur ou un nom; utilisez **0** pour envoyer le message à tous les joueurs). **id** est un identificateur entier du message et **val** correspond à la valeur (soit un réel ou une chaîne). Le message est envoyé dans le mode non garantie. Si **val** contient une chaîne, la longueur maximale de la chaîne permise sera de 30 000 caractères.

`mplay_message_send_guaranteed(player,id,val)` Envoie un message au joueur indiqué (soit un identificateur ou un nom; utilisez **0** pour envoyer le message à tous les joueurs). **id** est un identificateur entier du message et **val** correspond à la valeur (soit un réel ou une chaîne). Le message est expédié dans le mode garantie. Si **val** contient une chaîne, la longueur maximale permise sera de 30 000 caractères.

`mplay_message_receive(player)` Réceptionne le prochain message de la file d'attente des messages en provenance du joueur indiqué (soit un identificateur soit un nom). Utilisez **0** pour recevoir les messages de tous les joueurs. La routine indique en retour s'il y avait effectivement un nouveau message. Dans l'affirmative, vous pourrez utiliser les routines suivantes pour en lire le contenu :

`mplay_message_id()` Retourne l'identificateur du dernier message reçu.

`mplay_message_value()` Retourne la valeur ou contenu du dernier message reçu.

`mplay_message_player()` Retourne le joueur ayant envoyé le dernier message reçu.

`mplay_message_name()` Retourne le nom du joueur ayant envoyé le dernier message reçu.

`mplay_message_count(player)` Retourne le nombre de messages restant dans la file d'attente pour le joueur **player** (utilisez **0** pour compter tous les messages).

`mplay_message_clear(player)` Supprime tous les messages en attente dans la file du joueur **player** (utilisez **0** pour supprimer tous les messages).

Quelques remarques sont à faire ici. En premier lieu, si vous souhaitez envoyer un message uniquement à un joueur en particulier, il vous faudra connaître l'unique id du joueur. Comme indiqué précédemment, vous pouvez obtenir cet id grâce à la fonction `mplay_player_id()`. Cet identificateur du joueur est également utilisé lorsque vous recevez des messages d'un joueur particulier. Alternativement, vous pouvez donner le nom du joueur sous forme de chaîne. Si plusieurs joueurs possèdent le même nom, seul le premier recevra le message.

En second lieu, vous vous demandez peut-être pourquoi chaque message possède un identificateur entier. La raison est que cela aide votre application à envoyer différents types de messages. Le réceptionnaire peut vérifier le type du message en utilisant l'id et effectuer les actions appropriées (en raison qu'il n'y a aucune garantie que les messages arrivent, envoyer l'id et la valeur dans différents messages peut causer de sérieux problèmes.)

Utilisation de librairies DLL

Cette fonctionnalité n'est disponible que dans la version enregistrée de Game Maker.

Dans les cas où vous jugeriez que les fonctionnalités du GML ne seraient pas suffisantes, vous pouvez étendre ses possibilités en utilisant des plug-ins. Un plug-in se présente sous la forme d'un fichier DLL (**D**ynamic **L**ink **L**ibrary : Librairie à Lien Dynamique). Dans ce fichier DLL, vous pouvez adresser des fonctions. Les fonctions peuvent être programmées avec n'importe quel langage de programmation supportant la création de fichiers DLL (ex: Delphi, C, C++, etc.) Vous devrez cependant posséder un bon niveau de programmation pour créer des DLL. Les fonctions Plug-in doivent respecter un format spécifique. Elles peuvent avoir de **0** à **11** arguments, chacun de ceux-ci pouvant être un nombre réel (double en C) ou une chaîne terminée par un caractère *null* (s'il y a plus de 4 arguments, seuls les arguments de type réel sont supportés pour l'instant). Ces fonctions doivent retourner soit un réel ou encore une chaîne terminée par un caractère *null*.

Sous Delphi, vous créerez une DLL en choisissant en premier lieu l'article **New** du menu **File** puis vous sélectionnerez DLL. Voici un exemple de DLL que vous pourrez utiliser avec *Game Maker* écrit en Delphi (veuillez noter que ce code est en Delphi, pas en code GML !)

```
library MyDLL;

uses SysUtils, Classes;

function MyMin(x,y:double):double;  cdecl;
begin
  if x<y then Result := x else Result := y;
end;

var res : array[0..1024] of char;

function DoubleString(str:PChar):PChar; cdecl;
begin
  StrCopy(res,str);
  StrCat(res,str);
  Result := res;
end;
```

```
exports MyMin, DoubleString;

begin
end.
```

Cette DLL définit deux fonctions: `MyMin` prenant deux arguments réels puis retourne le minimum de ces deux valeurs et `DoubleString` qui concatène la même chaîne de caractères. Veuillez noter que vous devrez être particulièrement prudent en ce qui concerne la gestion de la mémoire. C'est pourquoi j'ai déclaré la chaîne résultat sous forme de variable globale. Veuillez remarquer également l'utilisation de la convention d'appel nommée cdecl. Vous pouvez utiliser soit les conventions d'appel cdecl ou stdcall. Une fois que vous avez créé la librairie DLL sous Delphi, vous obtiendrez le fichier de nom `MyDLL.DLL`. Ce fichier sera à placer ensuite dans le répertoire d'exécution de votre jeu (ou encore à tout autre endroit où Windows pourra le trouver).

Pour utiliser cette DLL dans *Game Maker*, vous devrez en premier lieu indiquer les fonctions externes que vous souhaitez employer et préciser quels types d'arguments elles devront accepter. A cet effet, il existe la fonction suivante en GML :

> **`external_define(dll,name,calltype,restype,argnumb,arg1type,arg2type, ...)`** Définit une fonction externe. **dll** est le nom du fichier de la DLL. **name** est le nom des fonctions. **calltype** est la convention d'appel utilisée. Utilisez soit *dll_cdecl* ou *dll_stdcall*. **restype** est le type du résultat. Utilisez soit *ty_real* ou *ty_string*. **argnumb** est le nombre d'arguments (0-11). Ensuite, pour chacun des arguments, vous devrez indiquer le type. Utilisez soit *ty_real* ou *ty_string*. S'il y a plus de 4 arguments, tous devront être de type *ty_real*.

Cette fonction retourne l'**id** de la fonction externe, ID devant être utilisé pour appeler la fonction. Ainsi, dans l'exemple ci-dessous, au début du jeu, vous utiliserez le code GML suivant :

```
{
  global.mmm = external_define('MYOWN.DLL','MyMin',dll_cdecl,

ty_real,2,ty_real,ty_real);
  global.ddd =
external_define('MYOWN.DLL','DoubleString',dll_cdecl,
                                      ty_string,1,ty_string);
}
```

Maintenant, si vous souhaitez appeler les fonctions, vous utiliserez la fonction suivante :

external_call(id,arg1,arg2,...) Appelle la fonction externe d'ID **id** avec les arguments indiqués. Il est nécessaire de fournir le nombre et le type corrects pour les arguments (réel ou chaîne). La fonction retourne le résultat de la fonction externe.

Par exemple, vous pourrez écrire :

```
{
  aaa = external_call(global.mmm,x,y);
  sss = external_call(global.ddd,'Hello');
}
```

Si vous n'avez plus besoin d'utiliser la DLL, il est préférable de la fermer.

external_free(dll) Libère la DLL de nom **dll**. Ceci est particulièrement indispensable si le jeu doit supprimer la DLL. Tant que la DLL n'est pas libérée, elle ne pourra pas être retirée. Il est conseillé de le faire à la fin de l'événement du jeu.

Vous vous demandez peut-être comment réaliser une fonction dans une DLL qui réalise quelque chose dans le jeu. Par exemple, vous pourrez vouloir créer une DLL qui ajoute des instances d'objets dans votre jeu. La façon la plus simple de procéder est de faire en sorte que la fonction DLL retourne une chaîne contenant du code GML. Cette chaîne contenant du code GML, pourra être exécutée à l'aide de la fonction GML suivante :

execute_string(str) Exécute du code GML contenu dans la chaîne **str**.

De plus, vous pouvez également faire en sorte que la DLL crée un fichier script qui pourra ensuite être exécuté (cette fonction peut aussi être utilisée pour modifier ultérieurement le comportement du jeu).

execute_file(fname) Exécute le code GML contenu dans le fichier de nom **name**.

Désormais, vous pouvez appeler une fonction externe puis exécuter la chaîne résultat de la manière suivante :

```
{
  ccc = external_call(global.ddd,x,y);
  execute_string(ccc);
}
```

Dans de rares cas, votre DLL aura besoin de connaître le gestionnaire de la fenêtre graphique principale de votre jeu. Ceci peut être réalisé avec la fonction suivante puis le résultat sera transmis à la DLL :

window_handle() Retourne le gestionnaire de fenêtre de la fenêtre principale (window handle).

Veuillez noter que les DLLs ne peuvent pas être utilisées dans le mode sécurisé.

L'utilisation de fichiers DLLs externes est un mécanisme extrêmement puissant. Mais attention : à n'utiliser cependant que si vous savez exactement ce que vous faites.

Graphiques 3D

Cette fonctionnalité n'est disponible que dans la version enregistrée de Game Maker.

Game Maker est un programme orienté jeux en 2 dimensions isométriques. Il existe néanmoins quelques fonctions permettant de créer des graphiques en 3 dimensions. Avant de débuter ce chapitre, il y a certaines choses que vous devez comprendre d'ores et déjà.

- La fonctionnalité 3D implémentée dans *Game Maker* se limite à la partie graphique. Il n'y a pas de support pour d'autres fonctionnalités 3D. A partir du moment où vous utilisez des graphiques en 3D avec *Game Maker*, vous connaîtrez des problèmes avec les autres aspects de *Game Maker*, comme les vues (views), l'ordre de profondeur, etc. La fonctionnalité est limitée et il n'est pas dans mes priorités d'améliorer les fonctions 3D. Aussi, n'attendez pas trop de support en ce qui concerne les modèles d'objets 3D, etc.
- Lorsque vous utilisez la fonctionnalité 3D, il existe un certain nombre d'autres choses que vous ne pourrez plus employer.
    - Il ne sera plus possible d'utiliser des arrière-plans et des avant-plans dans vos rooms (la raison est que ces derniers sont affichés de manière à remplir l'image mais dû aux projections de perspective, cela ne pourra plus fonctionner correctement).
    - Vous ne pourrez plus utiliser les fonctions ayant trait à la position de la souris. La position de la souris ne sera pas transformée en coordonnées 3D. Vous pourrez toujours obtenir la position de la souris à l'écran (dans une vue) mais ce sera à vous d'effectuer le calcul des coordonnées 3D (ou de ne plus utiliser du tout la souris).
    - Vous ne pourrez plus employer de tuiles. Les tuiles, la plupart du temps, ne correspondront plus correctement.
    - La vérification des collisions utilisera toujours les positions 2D des instances de la room. Aussi, il n'y aura plus de détection de collisions en 3D. Parfois, vous pourrez encore utiliser la vérification de collision (si vous utilisez la room comme représentation d'un monde plat (ex: pour les jeux de course ou FPS) mais dans les autres situations, vous devrez faire tout le travail par vous-même.
- Toutes les fonctions 3D utilisent exclusivement du code GML. Vous devrez en conséquence bien connaître le langage GML. Aussi, vous devrez bien comprendre le fonctionnement interne de *Game Maker* sinon vous aurez beaucoup de problèmes.

- Vous devrez avoir de bonnes connaissances en ce qui concerne les graphiques 3D. En particulier, j'emploierai des termes comme projection de perspective, suppression de surface cachée, éclairage et brouillard, sans donner davantage d'explications.
- Il n'existe pas de modèles 3D dans *Game Maker*. De plus, je n'envisage pas d'ajouter un quelconque support pour le chargement des modèles 3D.
- Vous devrez travaillez avec prudence pour conserver une vitesse d'exécution raisonnable. Rien n'a été réellement optimisé pour favoriser la vitesse.

Si tout ceci ne vous a pas découragés, alors lisez ce qui suit.

De l'information sur les graphiques 3D peut être trouvée dans les pages suivantes :

Sélection du mode 3D

Si vous souhaitez utiliser le mode 3D, vous devrez au préalable basculer *Game Maker* en mode 3D. Vous pourrez ultérieurement revenir dans le mode 2D si vous le souhaitez. Les deux fonctions suivantes sont dédiées à cet usage.

d3d_start() Positionne *Game Maker* dans le mode 3D.
d3d_end() Suspend le mode 3D.

Veuillez noter que toutes les fonctions concernant le mode 3D débutent par `d3d_`.

La bascule dans le mode 3D produira les changements suivants. En premier lieu, toutes les suppressions de surface cachée (hidden surface removal) seront positionnées à *ON* (en utilisant un tampon *z* de 16 bits). Cela signifie que pour chaque pixel à l'écran, seul sera affiché le dessin des valeurs *z* les plus petites (= valeur de profondeur) . Si les instances possèdent la même profondeur, ce qui pourra se passer est aléatoire et vous pourrez obtenir des effets de bords. Soyez certain que les instances pouvant se chevaucher, n'ont pas la même valeur de profondeur !

En second lieu, la projection normale orthonormée sera remplacée par une projection en perspective. Cela signifie ceci. Habituellement, la taille des instances à l'écran est indépendante de leur profondeur. Avec une projection en perspective, les instances ayant la plus grande profondeur apparaîtront plus petites. Si la profondeur est de **0**, elle sera égale à l'ancienne taille (sauf si vous avez modifié la projection; voir plus loin). Le point de vue de la caméra est placé à une certaine distance au-dessus de la room (cette distance est égale à la largeur de la room; ce qui donne une projection par défaut acceptable). Seules les instances en face de la caméra seront dessinées. Aussi, n'utilisez pas d'instances avec une profondeur inférieure à **0** (ou tout du moins plus petite que **-w** où **w** est la largeur de la room ou de la vue).

En troisième lieu, l'ordonnée verticale **y** est inversée. Alors que normalement la position **(0,0)** se situe en haut et à gauche de la vue, en mode 3D, la position **(0,0)** sera située en bas et à gauche, ce qui est habituel pour les vues en 3 dimensions.

Actuellement, vous pourrez basculer entre la suppression de surface cachée et la projection en perspective (*ON* ou *OFF*) en employant les fonctions suivantes.

d3d_set_hidden(enable) Active la suppression de surface cachée (**true**) ou la désactive (**false**).

d3d_set_perspective(enable) Active l'utilisation d'une projection en perspective (**true**) ou la désactive (**false**).

Dessin facile

Dès que vous avez sélectionné le mode 3D, vous pouvez utiliser *Game Maker* comme d'habitude (sauf en ce qui concerne les remarques faites au début de ce chapitre). Les objets apparaîtront en différentes tailles selon la définition de leur profondeur. Vous pourrez même utiliser des vues. Une fonction particulière pourra être très utile. Si vous dessinez un certain nombre de choses à l'aide de code GML, vous voudrez certainement changer la valeur de profondeur pour les primitives que vous dessinez. Utilisez alors la fonction suivante :

`d3d_set_depth(depth)` Régler la profondeur utilisée pour les opérations de dessin.

Veuillez noter qu'à partir du moment une nouvelle instance est affichée, la profondeur sera de nouveau fixée à la profondeur de cette instance.

Dessin de polygones en 3D

Le problème lorsque l'on dessine de façon traditionnelle fait qu'un sprite ou un polygone se situe toujours dans un plan xy, c'est à dire que tous ses coins présentent la même profondeur. Pour une vraie 3D, vous désirerez avoir des vertices de différentes profondeurs. A partir de cet instant, nous ne parlerons plus que de coordonnées en **z** et non plus de profondeur. Nous indiquerons les coordonnées sous forme de triplet **(x,y,z)**. Pour cela, il existe un jeu spécial de fonctions avancées de dessin :

d3d_primitive_begin(kind) Débute une primitive 3D de type **kind** : `pr_pointlist`, `pr_linelist`, `pr_linestrip`,`pr_trianglelist`,`pr_trianglestrip` ou `pr_trianglefan`.
d3d_vertex(x,y,z) Ajoute un vertex **(x,y,z)** à la primitive, en utilisant la couleur et la valeur alpha définies au préalable.
d3d_vertex_color(x,y,z,col,alpha) Ajoute un vertex **(x,y,z)** à la primitive avec sa propre couleur et valeur alpha. Cela vous permet de créer des primitives avec des changements progressifs de couleur et de valeur alpha.
d3d_primitive_end() Termine la description de la primitive. Cette fonction dessine aussi la primitive.

Par exemple, pour dessiner un tétraèdre (pyramide à trois côtés) se situant sur plan *z=0* avec son sommet en *z = 200*, vous pourrez utiliser le code suivant :

```
{
  d3d_primitive_begin(pr_trianglelist);
    d3d_vertex(100,100,0);
    d3d_vertex(100,200,0);
    d3d_vertex(150,150,200);
    d3d_vertex(100,200,0);
    d3d_vertex(200,200,0);
    d3d_vertex(150,150,200);
    d3d_vertex(200,200,0);
    d3d_vertex(100,100,0);
    d3d_vertex(150,150,200);
    d3d_vertex(100,100,0);
    d3d_vertex(100,200,0);
```

```
    d3d_vertex(200,200,0);
  d3d_primitive_end();
}
```

Maintenant, si vous utilisez ce code tel qu'il se présente, vous verrez juste la plupart du temps un triangle à l'écran parce que le dessus du tétraèdre se trouvera derrière lui à cause de l'angle de vue. Aussi, en utilisant seulement une couleur, il sera difficile de voir les différentes faces. Ci-dessous, vous verrez les façons de modifier l'angle de vue de la caméra. L'assignation de couleurs peut être réalisée comme cela a été vu auparavant en ajoutant des appels de fonctions `draw_set_color(col)` entre les vertices.

Il est possible également d'utiliser des polygones texturés en 3D. Cela fonctionne exactement de la même façon que les fonctions avancées de dessin décrites dans cette documentation. Mais cette fois-ci, vous aurez besoin de fonctions 3D quelques peu différentes des fonctions de base. Une chose que vous devez comprendre. Dans une texture, la position **(0,0)** correspond au coin supérieur gauche. Mais souvent, lorsque l'on utilise des projections (comme indiqué ci-dessous), le coin inférieur gauche sera la position **(0,0)**. Dans ce cas, il sera peut-être nécessaire de renverser verticalement la texture.

d3d_primitive_begin_texture(kind,texid) Débute une primitive 3D de type **kind** et de texture **texid**.
d3d_vertex_texture(x,y,z,xtex,ytex) Ajoute un vertex **(x,y,z)** à la primitive à la position **(xtex,ytex)** de la texture, avec mélange de la couleur et de la valeur **alpha** définies préalablement.
d3d_vertex_texture_color(x,y,z,xtex,ytex,col,alpha) Ajoute un vertex **(x,y,z)** à la primitive à la position **(xtex,ytex)** de la texture, avec mélange de sa propre couleur et de la valeur **alpha**.
d3d_primitive_end() Termine la description de la primitive. Cette fonction affiche aussi la primitive.

Ainsi, par exemple, vous pourrez utiliser le code suivant afin d'afficher une image de fond qui disparaîtra dans le lointain.

```
{
  var ttt;
  ttt = background_get_texture(back);
  d3d_primitive_begin_texture(pr_trianglefan,ttt);
    d3d_vertex_texture(0,480,0,0,0);
```

```
    d3d_vertex_texture(640,480,0,1,0);
    d3d_vertex_texture(640,480,1000,1,1);
    d3d_vertex_texture(0,480,1000,0,1);
  d3d_primitive_end();
}
```

Un triangle possède une face de devant et une de derrière. La face de devant est destinée à être le côté où les vertices seront définis dans le sens contraire des aiguilles d'une montre.
Habituellement, les deux faces seront affichées. Mais si vous réalisez une forme fermée, ce sera du gaspillage car la face de derrière du triangle ne pourra jamais être aperçue. Dans ce cas, vous pourrez sélectionner le mode face arrière cachée. Cela permettra d'économiser environ la moitié du temps d'affichage mais cela vous obligera à définir vos polygones en utilisant la bonne manière. La fonction suivante existe :

d3d_set_culling(cull) Indique de se placer en mode face arrière cachée (backface culling) (**true**) ou au contraire de l'arrêter (**false**).

Dessin de formes de base

Un certain nombre de fonctions existent pour dessiner des formes de base, comme des blocs ou des murs. Veuillez noter que ces formes sont affichées correctement lorsque le mode face arrière cachée est activé (*backface culling on*).

d3d_draw_block(x1,y1,z1,x2,y2,z2,texid,hrepeat,vrepeat) Dessine un bloc avec la couleur courante aux coins opposés indiqués en utilisant la texture **texid**. Utilisez la valeur **-1** pour ne pas utiliser de texture. `hrepeat` indique de combien de fois la texture doit être répétée le long du bord horizontal de chaque face. `vrepeat` fait la même chose pour le bord vertical.

d3d_draw_cylinder(x1,y1,z1,x2,y2,z2,texid,hrepeat,vrepeat,closed,steps) Dessine un cylindre vertical dans la couleur courante dans la boîte de rebond mentionnée en utilisant la texture indiquée. Utilisez la valeur **-1** afin de ne pas employer de texture. `hrepeat` indique de combien de fois la texture doit être répétée le long du bord horizontal de chaque face. `vrepeat` effectue la même chose mais pour le bord vertical. `closed` indique si l'on doit fermer le dessus et le dessous du cylindre. `steps` indique combien d'étapes de rotations doivent être faites. Une valeur typique est **24**.

d3d_draw_cone(x1,y1,z1,x2,y2,z2,texid,hrepeat,vrepeat,closed,steps) Dessine un cône vertical dans la couleur courante dans la boîte de rebond mentionnée en utilisant la texture indiquée. Utilisez la valeur **-1** pour ne pas utiliser de texture. `hrepeat` indique de combien de fois la texture doit être répétée le long du bord horizontal de chaque face. `vrepeat` fait la même chose pour le bord vertical. `closed` indique si l'on doit fermer le dessus et le dessous du cylindre. `steps` indique combien d'étapes de rotations doivent être réalisées. Une valeur typique est **24**.

d3d_draw_ellipsoid(x1,y1,z1,x2,y2,z2,texid,hrepeat,vrepeat,steps) Dessine une ellipse dans la couleur courante dans la boîte de rebond en utilisant la texture indiquée. Utilisez la valeur **-1** afin de ne pas employer de texture. `hrepeat` indique de combien de fois la texture doit être répétée le long du bord horizontal de chaque face. `vrepeat` effectue la même chose mais pour le bord vertical. `steps` indique combien d'étapes de rotations doivent être faites. Une valeur typique est **24**.

d3d_draw_wall(x1,y1,z1,x2,y2,z2,texid,hrepeat,vrepeat) Dessine un mur vertical dans la couleur courante aux coins indiqués en utilisant la texture **texid**. Utilisez la valeur **-1** pour ne pas utiliser de texture. `hrepeat` indique de

combien de fois la texture doit être répétée le long du bord horizontal de chaque face. `vrepeat` effectue la même chose mais pour le bord vertical.

d3d_draw_floor(x1,y1,z1,x2,y2,z2,texid,hrepeat,vrepeat) Dessine un sol (*incliné*) dans la couleur courante aux coins indiqués en utilisant la texture **texid**. Utilisez la valeur **-1** pour ne pas employer de texture. `hrepeat` indique de combien de fois la texture doit être répétée le long du bord horizontal de chaque face. `vrepeat` fait la même chose mais pour le bord vertical.

Le code suivant dessine deux blocs :

```
{
  var ttt;
  ttt = background_get_texture(back);
  d3d_draw_block(20,20,20,80,40,200,ttt,1,1);
  d3d_draw_block(200,300,-10,240,340,100,ttt,1,1);
}
```

Visualisation du Monde

Par défaut, vous regardez le long d'un axe négatif en z vers le milieu de la room. Souvent dans les jeux en 3D, vous souhaiterez modifier la façon de voir le monde. Par exemple, dans les jeux à la première personne, vous souhaiterez probablement disposer la caméra à une position légèrement au-dessus et le long du plan xy. En terme de graphiques, vous devrez paramétrer la projection de manière correcte. Pour modifier la façon de regarder le monde, il existe les deux fonctions suivantes.

d3d_set_projection(xfrom,yfrom,zfrom,xto,yto,zto,xup,yup,zup)
Définit la manière de regarder le monde. Vous indiquerez le point de vue de départ, le point de vue de la direction à suivre et le vecteur haut.

Cette fonction nécessite des explications. Pour paramétrer la projection, vous avez besoin en premier lieu de connaître la position à partir de laquelle vous effectuez l'observation. Cela est indiqué par les paramètres `(xfrom,yfrom,zfrom)`. Ensuite, vous devrez préciser la direction vers laquelle vous regardez. Cela se fait en donnant un second point de chute. Cela correspond au point `(xto,yto,zto)`. Enfin, vous pouvez diriger la caméra autour de la ligne du point de vue jusqu'au point de fuite du regard. Pour indiquer ceci, nous devons donner un vecteur haut, qui est la direction au-dessus de la caméra. Cela est précisé par les trois derniers paramètres `(xup,yup,zup)`. Prenons un exemple. Pour regarder le long d'un plan xy dans un jeu à la première personne, vous pouvez utiliser ceci

```
{
  d3d_set_projection(100,100,10,200,100,10,0,0,1);
}
```

Dans cet exemple, vous regardez à partir du point **(100,100)**, de **10** au-dessus du plan et dans la direction **(200,100)**. Les points du vecteur haut sont dans la direction en z comme requis. Pour rendre tout ceci légèrement plus compliqué, supposez que vous ayez une instance dans votre room qui indique la position de la caméra. Elle aura une position courante en **(x,y)** et une direction (et peut-être aussi une vitesse). Vous pouvez maintenant indiquer ceci à la caméra en utilisant le code suivant :

```
{
  with (obj_camera)
    d3d_set_projection(x,y,10,
                x+cos(direction*pi/180),y-
sin(direction*pi/180),10,
                0,0,1);
}
```

Tout cela doit vous sembler quelque peu compliqué ! Nous observons à partir de la position de la caméra en **(x,y)**, **10** au-dessus du sol. Afin de déterminer un point dans la bonne direction, nous avons besoin d'effectuer un peu de calcul arithmétique. Ce point est indiqué par les trois prochains paramètres. Enfin, nous utilisons le vecteur haut comme vu ci-dessus.

Faisons ici une remarque importante ! Lorsque *Game Maker* commence à afficher une room, il fixe le point de vue de nouveau à la position par défaut. Aussi, la première chose à faire quand la scène est placée à l'écran, est de préciser la projection souhaitée. Ceci doit obligatoirement être fait dans un événement d'affichage (*drawing event*) !

Il existe également une version plus évoluée de la fonction vue ci-dessus :

> **d3d_set_projection_ext(xfrom,yfrom,zfrom,xto,yto,zto,xup,yup,zup,angle,aspect,znear,zfar)** Une version améliorée de la fonction dans laquelle vous préciserez aussi l'angle définissant le champ visuel, le ratio d'aspect entre la taille horizontale et verticale de la vue et les plans de coupures proches et lointains.

Les paramètres additionnels fonctionnent comme suit. Si vous avez déjà indiqué la position de la caméra, le point à regarder et le vecteur haut, vous pouvez encore modifier la largeur de champ de l'objectif de la caméra. C'est ce que l'on appelle le champ visuel. Une valeur raisonnable est de **45** degrés et cette valeur est celle prise par défaut. Mais il vous est possible de modifier cette valeur si vous le souhaitez. Ensuite, vous préciserez le ratio d'aspect entre la projection horizontale et verticale. Normalement, vous souhaiterez utiliser la même valeur que le ratio d'aspect de la room ou de la vue, par exemple **640/480**. Enfin, vous devrez indiquer les plans de coupures. Les objets qui sont plus proches que `znear` par rapport à la caméra ne seront pas dessinés. Idem pour les objets plus loin que `zfar`. Il peut être important de bien régler ces paramètres à des valeurs raisonnables car ils auront également une influence sur la précision des comparaisons en z. Si vous

créez une plage de valeurs trop grandes, la précision sera mauvaise. Par défaut, nous utiliserons **1** et **32000**. `znear` doit être plus grand que **0** !

Parfois, vous aurez besoin temporairement d'une projection normale orthonormée comme c'est le cas quand nous n'employons pas de 3D. Ou bien vous souhaiterez revenir à la projection en perspective par défaut. Vous utiliserez alors les fonctions suivantes :

d3d_set_projection_ortho(x,y,w,h,angle) Paramètre une projection normale orthonormée du secteur indiqué de la room, orientée selon l'angle indiqué.
d3d_set_projection_perspective(x,y,w,h,angle) Paramètre une projection normale en perspective du secteur indiqué de la room, orientée selon l'angle indiqué.

Une utilisation courante de tout ceci est l'affichage d'un recouvrement pour par exemple montrer le score ou d'autres aspects du jeu. Pour réaliser cela, nous définirons une projection orthonormée. Nous devrons également temporairement déconnecter la suppression de face cachée (*hidden surface removal*) parce que nous voulons que l'information soit affichée indépendamment de la valeur courante de profondeur. L'exemple suivant montre comment créer un recouvrement pour afficher le score.

```
{
  draw_set_color(c_black);
  d3d_set_projection_ortho(0,0,room_width,room_height,0);
  d3d_set_hidden(false);
  draw_text(10,10,'Score: ' + string(score));
  d3d_set_hidden(true);
}
```

Transformations

L'opération dite de transformation vous permet de changer l'emplacement où les choses sont dessinées dans le monde. Par exemple, la fonction qui dessine des blocs ne peut uniquement dessiner des blocs parallèles aux axes. En paramétrant en premier lieu une transformation de rotation, vous pourrez créer des blocs avec une orientation. De même, les sprites sont habituellement toujours affichés parallèlement à un plan xy. En paramétrant une transformation, vous serez en mesure de modifier ce comportement. Il existe deux types de fonctions : les fonctions permettant de paramétrer une transformation et celles qui ajoutent des transformations.

d3d_transform_set_identity() Paramètre la transformation en lui affectant une identité (pas de transformation).
d3d_transform_set_translation(xt,yt,zt) Paramètre la transformation en lui appliquant une translation selon les vecteurs indiqués.
d3d_transform_set_scaling(xs,ys,zs) Paramètre la transformation selon une certaine échelle en utilisant les valeurs indiquées.
d3d_transform_set_rotation_x(angle) Paramètre la transformation en lui appliquant une rotation autour de l'axe des x selon la valeur indiquée.
d3d_transform_set_rotation_y(angle) Paramètre la transformation en lui appliquant une rotation autour de l'axe des y selon la valeur indiquée.
d3d_transform_set_rotation_z(angle) Paramètre la transformation en lui appliquant une rotation autour de l'axe des z selon la valeur indiquée.
d3d_transform_set_rotation_axis(xa,ya,za,angle) Paramètre la transformation en lui appliquant une rotation autour des axes indiqués selon le vecteur et la valeur indiqués.
d3d_transform_add_translation(xt,yt,zt) Ajoute une translation en utilisant le vecteur mentionné.
d3d_transform_add_scaling(xs,ys,zs) Ajoute une mise à l'échelle en utilisant les valeurs indiquées.
d3d_transform_add_rotation_x(angle) Ajoute une rotation autour de l'axe des x avec la valeur indiquée.
d3d_transform_add_rotation_y(angle) Ajoute une rotation autour de l'axe des y avec la valeur indiquée.
d3d_transform_add_rotation_z(angle) Ajoute une rotation autour de l'axe des z avec la valeur indiquée.
d3d_transform_add_rotation_axis(xa,ya,za,angle) Ajoute une rotation autour des axes indiqués selon le vecteur et la valeur indiqués.

Veuillez comprendre que la rotation et la mise à l'échelle se font dans le respect de l'origine du monde, et non pas en fonction de l'objet devant être dessiné. Si l'objet ne se trouve pas à l'origine, il sera également déplacé à un autre endroit, ce qui ne sera pas forcément ce que vous voudrez réaliser. Aussi, pour effectuer une rotation d'un objet selon son propre axe des x, nous devons en premier lieu lui effectuer une translation selon l'origine, puis lui appliquer une rotation pour enfin lui faire une translation une nouvelle fois selon sa position. C'est pourquoi il existe des fonctions pour ajouter des transformations.

Les exemples suivants devraient mieux vous faire comprendre ce mécanisme. Supposons que nous avons un sprite `spr` que nous voulons afficher à la position (100,100,10). Nous pouvons utiliser le code suivant :

```
{
  d3d_transform_set_translation(100,100,10);
  draw_sprite(spr,0,0,0);
  d3d_transform_set_identity();
}
```

Veuillez noter que dû au fait que nous employons une translation, nous devons maintenant afficher le sprite à la position **(0,0)** (ceci suppose que l'instance courante présente une profondeur de 0 ! Si vous n'en êtes pas sûr, régler en premier lieu la profondeur). Si nous utilisons cela dans notre jeu de shoot à la première personne, nous ne verrons pas le sprite. La raison est que le sprite est toujours affiché parallèlement au plan des xy. Nous voulons lui appliquer une rotation de 90 degrés le long de l'axe des x (ou de l'axe des y). Aussi, nous devrons ajouter une rotation. Souvenez-vous de l'ordre : nous devons en premier lieu effectuer une rotation du sprite puis lui appliquer une translation. Ainsi, nous utiliserons le code suivant.

```
{
  d3d_transform_set_identity();
  d3d_transform_add_rotation_x(90);
  d3d_transform_add_translation(100,100,10);
  draw_sprite(spr,0,0,0);
  d3d_transform_set_identity();
}
```

Parfois, vous désirerez temporairement sauvegarder la transformation actuelle, par exemple pour ajouter une transformation supplémentaire puis restaurer l'ancienne transformation (cela arrive souvent lorsque l'on dessine des modèles hiérarchiques). Pour arriver à ce but, vous pouvez pousser la transformation courante sur la pile pour ultérieurement la retirer de cette dernière afin de l'utiliser ensuite comme transformation courante. Il existe pour cela les fonctions suivantes :

d3d_transform_stack_clear() Efface la pile des transformations.
d3d_transform_stack_empty() Indique si la pile des transformations est vide.
d3d_transform_stack_push() Pousse (push) la transformation courante sur la pile. Indique en retour s'il y avait une room sur la pile pour la pousser à cet emplacement (si vous omettez d'effectuer un pop de la transformation, vous obtiendrez à un certain moment un message "out of room on the stack").
d3d_transform_stack_pop() Dépile la transformation du haut de la pile et la désigne comme transformation courante. Indique en retour s'il y avait une transformation sur la pile.
d3d_transform_stack_top() Désigne la transformation située en haut de pile comme la transformation courante mais ne l'enlève pas de la pile. Indique en retour s'il y avait une transformation sur la pile.
d3d_transform_stack_discard() Supprime la transformation du haut de la pile sans la désigner comme transformation courante. Indique en retour s'il y avait une transformation sur la pile.

L'utilisation de transformation constitue un mécanisme puissant. Mais il convient d'être prudent et de toujours replacer la transformation à une identité dès que vous en avez terminé.

Brouillard (Fog)

Du brouillard peut être utilisé dans les jeux 3D afin de permettre aux objets d'apparaître trouble selon la distance voire même de disparaître. Cela permet de créer une certaine atmosphère. Il est même possible de ne pas afficher les objets se trouvant à une certaine distance. La fonction suivante active ou désactive le brouillard :

> **d3d_set_fog(enable,color,start,end)** Active ou désactive l'utilisation du brouillard. `color` indique la couleur du brouillard. `start` précise la distance à partir de laquelle le brouillard doit débuter. `end` signale la distance à laquelle le brouillard doit être maximal et où aucun objet ne pourra être perçu.

Pour bien comprendre comment cela marche, il existe actuellement deux types de brouillard, le brouillard basé sur une table (*table based fog*) et celui basé sur des vertex ou sommets (*vertex based fog*). Le premier type calcule les valeurs de brouillard sur la base d'un pixel. Le second type détermine la valeur du brouillard pour chaque vertex ou sommet puis effectue une interpolation de ceux-ci. Le premier type est meilleur que le second mais n'est pas toujours supporté. *Game Maker* tentera d'utiliser le brouillard basé sur une table si supporté et le second type de brouillard dans le cas contraire (à moins qu'aucun brouillard ne soit supporté). Veuillez noter que certaines cartes graphiques rapportent qu'elles savent gérer le brouillard basé sur une table et offrent à l'utilisateur la possibilité de désactiver cette fonction dans les paramètres avancés d'affichage. Dans ce cas, le résultat sera un écran tout noir !

Eclairage (Lighting)

Les scènes que vous avez pu dessiner à l'aide des fonctions vues précédemment doivent vous paraître bien fades, ceci en raison de l'absence de lumière. La couleur est, dans le cas où les faces sont égales, indépendante de leur orientation. Afin de créer des scènes plus réalistes, vous devrez utiliser de l'éclairage et placer des spots à des endroits judicieux. Néanmoins, créer correctement des scènes avec de l'éclairage n'est pas tâche facile mais les effets peuvent être très bons.

Vous pourrez utiliser la fonction suivante pour activer l'éclairage :

d3d_set_lighting(enable) Active ou désactive l'utilisation de l'éclairage.

Lorsque vous utilisez de l'éclairage, la couleur est déterminée pour chaque sommet du polygone. Ensuite, la couleur des pixels internes est basée sur la couleur de ces vertices. Deux manières de réaliser cela : soit le polygone tout entier est de la même couleur ou soit la couleur est progressivement interpolée sur le polygone. Par défaut, il est utilisé de l'ombrage progressif (smooth shading). Cela peut être modifié en employant la fonction suivante :

d3d_set_shading(smooth) Détermine si l'on doit utiliser ou non l'ombrage progressif.

Pour utiliser de l'éclairage, vous devrez définir des lumières ou spots. Il existe deux types de lumières : les lumières directionnelles (comme le soleil) et celles positionnelles (comme une caméra). La lumière possède une couleur (seule la lumière diffuse est supportée et pas la réflexion spéculaire). Les fonctions suivantes existent pour définir et utiliser des lumières :

d3d_light_define_direction(ind,dx,dy,dz,col) Définit une lumière avec une direction. **ind** est l'index de la lumière (utilisez un petit nombre positif). **(dx,dy,dz)** correspond à la direction de la lumière. **col** est la couleur de la lumière (vous utiliserez souvent `c_white`). Cette fonction n'active pas la lumière.
d3d_light_define_point(ind,x,y,z,range,col) Définit un point de lumière. **ind** est l'index de la lumière (utilisez un petit nombre positif). **(x,y,z**) correspond à la position de la lumière. **range** indique jusqu'à quelle distance doit briller la lumière. L'intensité de la lumière ira en décroissant au-delà de cette valeur. **col** est la couleur de la lumière. Cette fonction n'active pas la lumière.
d3d_light_enable(ind,enable) Active (*true*) ou désactive (*false*) la lumière de numéro **ind**.

La manière dont la lumière se réfléchit sur un objet dépend de l'angle entre la direction de la lumière et de la normale de la surface, c'est à dire du vecteur pointant loin de la surface.

Pour créer des objets avec lumière, vous avez besoin non seulement de fournir la position des vertices mais aussi de leurs normales. A cet effet, il existe quatre fonctions supplémentaires qui permettent de définir les vertices des primitives :

d3d_vertex_normal(x,y,z,nx,ny,nz) Ajoute un sommet **(x,y,z)** à la primitive, avec un vecteur normal **(nx,ny,nz)**.
d3d_vertex_normal_color(x,y,z,nx,ny,nz,col,alpha) Ajoute un sommet **(x,y,z)** à la primitive, avec un vecteur normal **(nx,ny,nz)** et sa propre couleur et valeur **alpha**.
d3d_vertex_normal_texture(x,y,z,nx,ny,nz,xtex,ytex) Ajoute un sommet **(x,y,z)** à la primitive, avec un vecteur normal **(nx,ny,nz)** et à la position **(xtex,ytex)** de la texture, avec mélange de la couleur et de la valeur *alpha* définies précédemment.
d3d_vertex_normal_texture_color(x,y,z,nx,ny,nz,xtex,ytex,col,alpha) Ajoute un sommet **(x,y,z)** à la primitive, avec un vecteur normal **(nx,ny,nz)**, à la position **(xtex,ytex)** de la texture, avec mélange de sa propre couleur et de la valeur **alpha**.

Veuillez noter que pour les formes de base que vous pouvez dessiner, les normales sont automatiquement paramétrées correctement.

Création de modèles

Lorsque vous avez besoin de dessiner de grands modèles, il est très coûteux d'invoquer sans cesse les différentes fonctions de dessin à chaque step. Afin d'éviter cela, vous avez la possibilité de créer des modèles. Un modèle est constitué d'un certain nombre de primitives de dessin et de formes. Dès qu'un modèle est créé, vous pouvez le dessiner à différents endroits en utilisant juste un seul appel de fonction. Un modèle peut également être chargé à partir d'un fichier ou sauvegardé dans un fichier.

Avant de décrire les différentes fonctions disponibles, il y a un chose importante à connaître : la gestion des textures. Comme décrit plus en avant, les textures sont créées à partir de sprites et d'arrière-plans. Les indices des textures peuvent être différents à certains moments. En conséquence, les modèles ne contiennent aucune information sur les textures. C'est seulement quand vous afficherez le modèle que vous indiquerez la texture. Aussi, vous ne pouvez utiliser qu'une texture par modèle. Si vous souhaitez disposer de plus de textures, vous devrez soit les combiner en une seule (et être très prudent avec les coordonnées de la texture) ou utiliser plusieurs modèles. L'avantage de ceci est que vous pourrez dessiner aisément le même modèle avec différentes textures.

Les fonctions suivantes existent et permettent la création, le chargement, la sauvegarde et l'affichage des modèles :

d3d_model_create() Créé un nouveau modèle puis retourne son index. Cet index sera à utiliser dans toutes les autres fonctions ayant trait aux modèles.
d3d_model_destroy(ind) Détruit le modèle d'index **ind**, libérant la mémoire occupée par ce dernier.
d3d_model_clear(ind) Efface le modèle d'index **ind**, supprimant ainsi toutes ses primitives.
d3d_model_save(ind,fname) Sauvegarde le modèle **ind** sous le nom de fichier **fname**.
d3d_model_load(ind,fname) Charge le modèle **ind** à partir du fichier de nom **fname**.
d3d_model_draw(ind,x,y,z,texid) Affiche le modèle à la position **(x,y,z)**. **texid** correspond à la texture devant être utilisée. Utilisez la valeur **-1** si vous ne souhaitez pas utiliser de texture. Si vous désirez effectuer une rotation ou mettre à l'échelle le modèle, vous pourrez utilisez les routines de transformation décrites précédemment.

Pour chaque fonction primitive, il existe une fonction équivalente pour l'ajouter à un modèle. Les fonctions possèdent les mêmes arguments comme auparavant excepté que chacune a comme premier argument l'index du modèle et aucune information de texture n'est nécessaire.

d3d_model_primitive_begin(ind,kind) Ajoute une primitive 3D au modèle de type **kind**: `pr_pointlist, pr_linelist, pr_linestrip,pr_trianglelist,pr_trianglestrip` ou `pr_trianglefan`.
d3d_model_vertex(ind,x,y,z) Ajoute un sommet **(x,y,z)** au modèle.
d3d_model_vertex_color(ind,x,y,z,col,alpha) Ajoute un sommet **(x,y,z)** au modèle, avec sa propre couleur et valeur **alpha**.
d3d_model_vertex_texture(ind,x,y,z,xtex,ytex) Ajoute un sommet **(x,y,z)** au modèle à la position **(xtex,ytex)** de la texture.
d3d_model_vertex_texture_color(ind,x,y,z,xtex,ytex,col,alpha) Ajoute un sommet **(x,y,z)** au modèle avec des valeurs de texture et de couleur.
d3d_model_vertex_normal(ind,x,y,z,nx,ny,nz) Ajoute un sommet **(x,y,z)** au modèle, avec un vecteur normal **(nx,ny,nz)**.
d3d_model_vertex_normal_color(ind,x,y,z,nx,ny,nz,col,alpha) Ajoute un sommet **(x,y,z)** au modèle, avec un vecteur normal **(nx,ny,nz)** et avec sa propre couleur et valeur **alpha**.
d3d_model_vertex_normal_texture(ind,x,y,z,nx,ny,nz,xtex,ytex) Ajoute un sommet **(x,y,z)** au modèle, avec un vecteur normal **(nx,ny,nz)**, avec une position de texture.
d3d_model_vertex_normal_texture_color(ind,x,y,z,nx,ny,nz,xtex,ytex,col,alpha) Ajoute un sommet **(x,y,z)** au modèle, avec un vecteur normal **(nx,ny,nz)**, avec des valeurs de texture et de couleur.
d3d_model_primitive_end(ind) Termine la description de la primitive du modèle.

En dehors des primitives, vous pouvez également ajouter des formes de base aux modèles. A nouveau, les fonctions sont presque les mêmes mais disposent d'un index de modèle et n'ont pas d'information de texture :

d3d_model_block(ind,x1,y1,z1,x2,y2,z2,hrepeat,vrepeat) Ajoute une forme *bloc* au modèle.
d3d_model_cylinder(ind,x1,y1,z1,x2,y2,z2,hrepeat,vrepeat,closed,steps) Ajoute une forme *cylindre* au modèle.
d3d_model_cone(ind,x1,y1,z1,x2,y2,z2,hrepeat,vrepeat,closed,steps) Ajoute une forme *cône* au modèle.

d3d_model_ellipsoid(ind,x1,y1,z1,x2,y2,z2,hrepeat,vrepeat,steps) Ajoute une forme *ellipse* au modèle.

d3d_model_wall(ind,x1,y1,z1,x2,y2,z2,hrepeat,vrepeat) Ajoute une forme *mur* au modèle.

d3d_model_floor(ind,x1,y1,z1,x2,y2,z2,hrepeat,vrepeat) Ajoute une forme *plancher* au modèle.

L'utilisation de modèles peut améliorer considérablement la vitesse d'affichage des graphiques dans vos jeux 3D et vous devriez les utiliser le plus souvent possible.

Mot de la fin

Les fonctions 3D de *Game Maker* peuvent être utilisées pour réaliser de jolis jeux en 3D. Cependant, ces fonctions demeurent limitées en fonctionnalité et demande un important travail de votre part. N'espérez pas pouvoir réaliser avec elles votre propre jeu *Quake*. *Game Maker* est et reste avant tout un logiciel pour créer des jeux en 2 dimensions.

Notes

Notes