Tutoriel: Création de Jeux de Plates-formes

Dernière modification : le 4 septembre 2004
Nécessite : Version 6.0 de Game Maker, en mode avancé
Niveau : Intermédiaire

Traduction française : Philippe Ragni (alias Xierra54) – Mai 2006

Les jeux de plates-formes sont présents sur tous les systèmes, en particulier sur les consoles de jeux comme la Game Boy. Dans un jeu de plates-formes, vous observez généralement la scène de côté. Le joueur dirige habituellement un personnage qui se déplace dans un monde comprenant des plates-formes. Le joueur peut marcher sur les plates-formes, sauter sur ces dernières ou encore tomber d'une plate-forme sur une autre, utiliser des échelles ou encore des cordes pour aller à différents endroits, etc. Sur les plates-formes se trouvent des objets à collecter, des ennemis à éviter ou à tuer (très souvent en leur tirant dessus ou en leur sautant sur la tête), des interrupteurs à utiliser pour ouvrir des passages, etc. Il est donc nécessaire pour le joueur de posséder une certaine habileté pour déjouer les pièges ou éviter les zones dangereuses. Dans certains jeux de plates-formes, il vous sera possible de visualiser le niveau dans sa totalité mais le plus souvent, seule la partie autour du personnage sera visible. Trouver son chemin dans ces conditions constituera un challenge supplémentaire qui renforcera l'intérêt du jeu.

La création d'un bon jeu de plates-formes n'est pas chose facile, avec ou sans Game Maker.
Il y a trois aspects importants à prendre en compte :

- Création de mouvements naturels pour le(s) personnage(s).
- Création de variations suffisantes en ce qui concerne les monstres, le décor, etc.
- Conception méticuleuse des niveaux de jeu de façon à ce que ceux-ci soient amusants à jouer mais aussi présentent une difficulté croissante.

Dans ce tutoriel, je donnerai quelques conseils et astuces sur la manière de concevoir des jeux de plates-formes à l'aide du logiciel Game Maker. Le tutoriel est accompagné de nombreux jeux de démonstration. Ce ne sont pas des jeux complets. Ils ne comprennent juste qu'un niveau afin de mettre en évidence un aspect particulier de la conception de ce type de jeu. Vous êtes autorisé à les utiliser comme base pour vos propres jeux de plates-formes.

Les bases d'un jeu de plates-formes

Nous commencerons par un jeu de plates-formes le plus simple qui soit. Vous le trouverez dans le fichier ***platform_1.gm6***. Dans tout bon jeu de plates-formes qui se respecte, il existe au minimum deux objets de base : un héros ou personnage principal, contrôlé par le joueur et un objet de type bloc utilisé pour les sols (plates-formes) sur lequel le joueur pourra marcher. Le même bloc est d'ailleurs souvent utilisé pour les murs que le joueur quant à lui ne pourra pas traverser. Nous aurons donc besoin de deux sprites : un pour le personnage et un autre pour le bloc. Pour le héros, nous utiliserons une simple balle. Pour le bloc, nous emploierons un carré noir (non transparent) Nous créerons aussi deux objets. D'une part, l'objet bloc qui est simplement un objet solide ne possédant ni évènement ni action. Sa fonction est juste d'être présent dans la room. L'objet du personnage, quant à lui, est bien entendu beaucoup plus élaboré.

Les déplacements

Un des aspects importants que nous traiterons dans cette première section concernera la définition des déplacements du personnage. Le problème est que le personnage doit marcher exclusivement au dessus du sol. Il ne doit pas bien entendu pénétrer dans celui-ci. Si le personnage saute ou tombe sur une plate-forme, il doit atterrir correctement sur la plate-forme suivante. Il existe d'ailleurs différentes situations où le personnage peut marcher, sauter et tomber. Chaque jeu de plates-formes utilise sa propre méthode. D'ordinaire, on n'utilise seulement que trois touches pour contrôler les déplacements du joueur. La touche flèche gauche devrait permettre au personnage d'aller sur la gauche, la touche flèche droite d'aller sur la droite et la flèche haut ou encore la touche espace de réaliser un saut.

En premier lieu, considérons uniquement les déplacements pour aller à gauche ou à droite. Le premier choix à faire sera de savoir quand le joueur peut changer de direction de mouvements selon qu'il soit sur une plate-forme ou encore dans les airs, à quel moment il peut sauter ou tomber. En raison du fait que la seconde option n'est pas naturelle (il est plutôt difficile de se déplacer vers la gauche lorsque l'on est en train de tomber !), nous décidons d'opter pour la première option qui autorise uniquement les déplacements horizontaux quand le héros se trouve sur une plate-forme. Cela contribuera à rendre l'utilisation du jeu encore plus agréable mais c'est également la méthode la plus simple à implémenter.

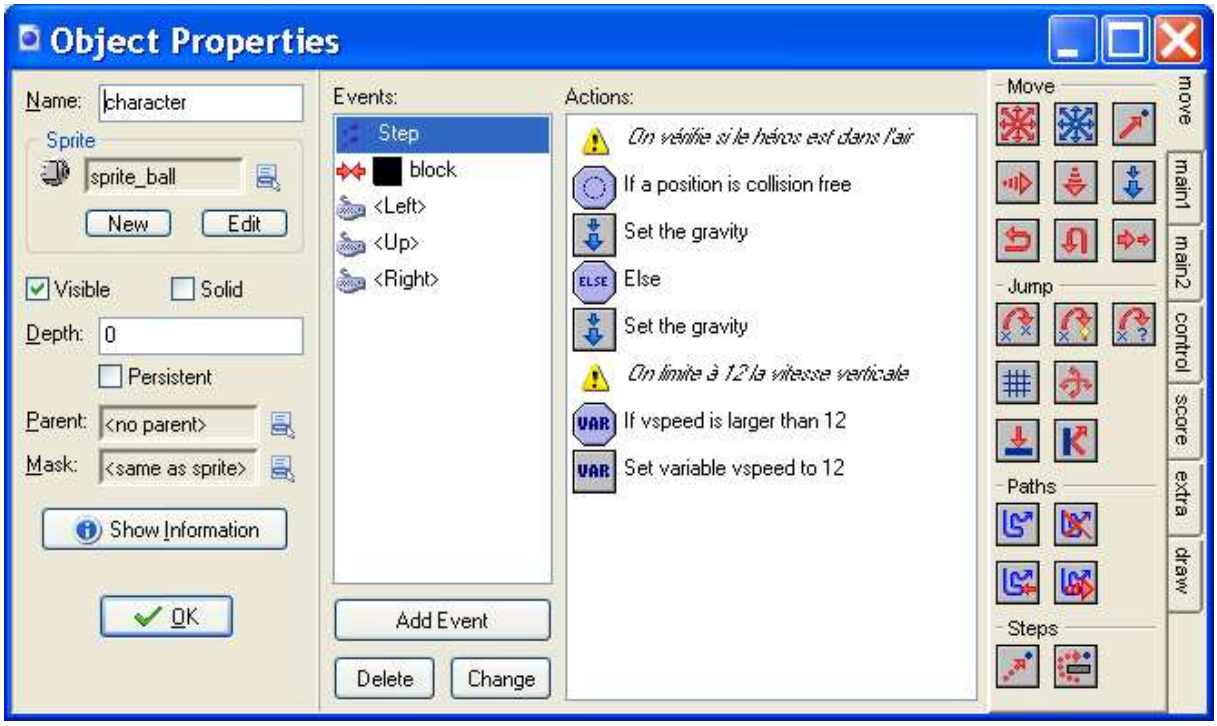
Le second choix est plus indiqué lorsque le déplacement du personnage se fait à vitesse constante ou va en s'accélérant lors de l'appui prolongé sur une touche. Pour des raisons de simplicité, nous opterons pour le premier choix. Afin d'avoir une accélération offrant le meilleur gameplay possible, le joueur devra commencer à courir depuis une certaine distance avant de pouvoir effectivement sauter au-dessus d'un trou. Comme vous le savez déjà sans doute, il existe différentes manières de déplacer un personnage. Nous pouvons régler globalement une vitesse de déplacement ou plus simplement gérer directement par nous-mêmes les déplacements du personnage. Dans les jeux de plates-formes, la solution la plus simple consiste à régler le déplacement vertical de façon automatique (comme nous le verrons ci-dessous) et à nous occuper de la gestion des déplacements horizontaux. Cela nous simplifiera beaucoup la tâche. Dans l'événement clavier concernant la touche flèche gauche, nous vérifierons si l'emplacement est libre à la position relative (-4,0) Dans l'affirmative, nous autoriserons le personnage à sauter directement à cette position. Nous traiterons le cas de la touche flèche droite de façon similaire. Pour une meilleure compréhension, je vous invite à consulter le jeu donné en guise d'exemple.

Les sauts

Nous avons besoin ici de définir le déplacement vertical. C'est une tâche plus difficile. Afin de permettre au personnage de tomber, nous pouvons utiliser la gravité. Mais nous devrons veiller à l'arrêter quand nous atteindrons le sol. Ainsi, nous souhaiterons normalement plafonner la vitesse de chute du héros sinon le personnage se déplacera trop vite (ceci n'est pas très agréable mais cela peut également créer des problèmes lors de l'implémentation. Par exemple, le personnage pourrait traverser un plancher s'il se déplaçait trop vite) Pour résoudre ce problème, dans l'évènement *STEP* du personnage, nous vérifierons si la position immédiatement en dessous du héros est libre de collision. Dans l'affirmative, cela indiquera que le personnage est dans l'air et nous pourrons alors régler la gravité à une valeur positive (NDT :un paramètre correct pour la gravité se situe aux alentours de 0.6) Dans le cas contraire, nous paramétrerons la gravité à 0. Nous devrons également vérifier la variable ***vspeed*** qui concerne la vitesse verticale.

Si la vitesse se révèle supérieure à 12, nous lui affecterons la valeur 12. De cette manière, nous limiterons la vitesse verticale à 12.

L'évènement *STEP* ressemblera par conséquent à ceci :



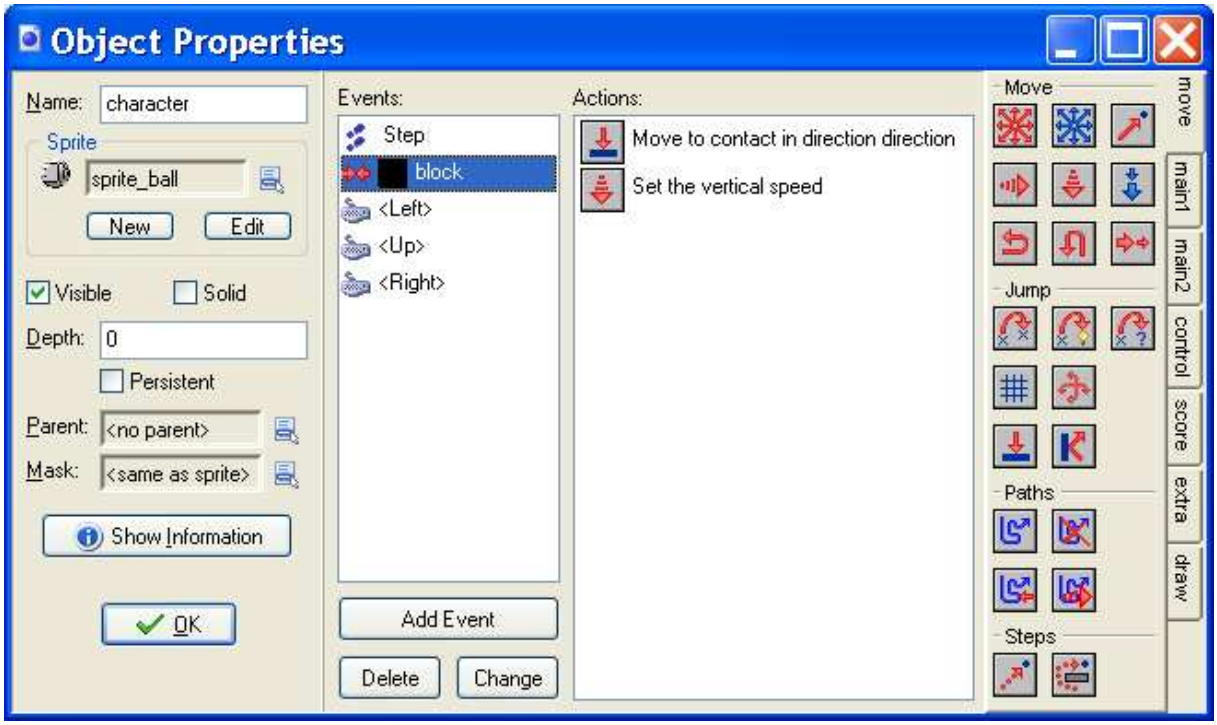
A ce stade, nous devons maintenant faire atterrir le personnage correctement sur le sol. Cela est plus difficile qu'il n'y paraît au premier abord. Cela doit uniquement survenir lorsque le héros entre en collision avec l'objet bloc. Dans l'évènement de collision, nous devrons régler le déplacement vertical à 0. Mais cette façon de faire pourrait également conduire à placer le personnage à une position légèrement au-dessus du sol (ceci est dû au fait que le héros est replacé à sa précédente position, celle qu'il avait juste avant la collision) En fin de chute, nous devrons placer le héros au point précis où la collision est survenue. Heureusement, Game Maker met à notre disposition une instruction pour réaliser tout cela :

Move to contact position (*Déplacer à la position de contact*)

Avec cette action, vous pourrez déplacer une instance dans une direction donnée jusqu'à ce qu'une position de contact soit vérifiée avec un objet. S'il existait déjà auparavant une collision à la position courante, l'instance ne sera pas déplacée. Dans le cas contraire, l'instance sera placée à sa nouvelle position qui correspond à l'emplacement juste avant que la collision ne survienne. Vous pouvez indiquer la direction mais aussi une distance maximale pour le déplacement. Vous pourrez indiquer également si l'on doit prendre en compte uniquement les objets solides ou encore tous les objets.

Nous utiliserons donc cette dernière action. Pour fixer la direction, nous emploierons la variable *direction* qui contient la position courante de déplacement de l'instance. Comme distance maximale, nous spécifierons 12 (ici, cela n'est pas vraiment nécessaire) :

L'évènement de collision entre le personnage et le bloc devrait ressembler à ceci :



Vous pourrez bien sûr penser que nous n'aurons à faire ces actions que lorsque le joueur heurte le sol. Mais nous souhaitons aussi nous déplacer à la position de contact dans le cas où le personnage toucherait un plancher situé en dessous de lui mais également si le héros entre en collision avec un mur sur le côté. Il apparaît ici une chose très importante qui est souvent source de problèmes pour les débutants : nous supposons que le personnage est hors collision à la position qu'il occupait précédemment. Vous pouvez effectivement supposer que cela est vrai sans vouloir le vérifier mais attention, cela n'est pas toujours le cas. Une autre erreur souvent commise est d'ignorer que, lorsque le personnage possède une image animée, le masque de collision change également à chaque step. Ceci explique que la nouvelle image puisse encore générer une collision à la position précédente. Ainsi, il est préférable de vous assurer que le personnage possède un masque de collision (vous reporter à la section suivante pour plus de détails)

Enfin, nous devrons permettre au personnage d'effectuer un saut lorsque la touche flèche haute sera pressée. Mais cela ne doit survenir cependant que lorsque le héros est réellement sur le sol. Nous devrons donc vérifier avant toute chose que la position située juste sous le personnage peut créer une collision et, dans l'affirmative, régler par exemple la vitesse verticale à -10. Vous devrez certainement paramétrer à 10 la vitesse verticale et à 0,5 la gravité, ceci afin d'obtenir le mouvement désiré.

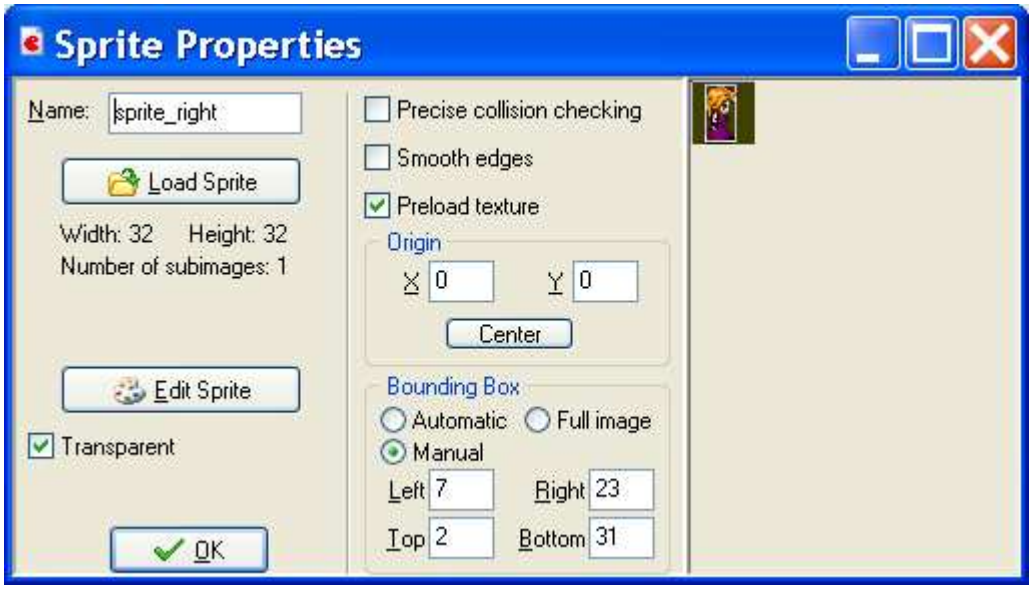
A ce stade, la base du jeu de plates-formes est totalement définie. Concevez ensuite un niveau avec d'autres sols et murs, construits à partir des instances de l'objet bloc. Placez une instance du personnage dans la room et vous en aurez terminé.

Des graphiques à améliorer

Le jeu de plates-formes que nous avons créé dans le paragraphe précédent fonctionne correctement mais n'est pas très élaboré. Il existe deux aspects que nous devons absolument changer : la présentation graphique du personnage ainsi que celle du décor. Le jeu prenant en compte ces modifications se trouve dans le fichier ***platform_2.gm6***.

Les images du personnage

Commençons avec les graphiques concernant le personnage. Nous utiliserons deux sprites différents (non animés) : un pour le personnage se tournant vers la gauche et un autre pour celui s'orientant vers la droite. Le plus simple maintenant sera de placer une action dans l'évènement gérant la touche flèche gauche afin de changer le sprite actuel par celui tournant vers la gauche. De la même façon, et en ce qui concerne la touche flèche droite, vous permuterez avec le sprite affichant le personnage s'orientant vers la droite. Il est très important que vous mettiez la vérification précise des collisions (*precise collision checking*) à **Off** pour les deux sprites. Il existe un certain nombre de raisons à cela. La première d'entre elles évite que le sprite reste à moitié positionné sur le bord de la plate-forme. La deuxième raison se justifie par le fait que lorsque le sprite passe du profil gauche au profil droit, il doit toujours cependant utiliser le même masque de collision, sinon le héros pourrait rester coincé. C'est encore plus important si on utilise des sprites animés. Pour la même raison, vous devrez vous assurer que la boîte de rebond (*bounding box*) de chaque sprite soit toujours la même. Vous pouvez utiliser des boîtes de rebond manuelles pour cela. Ainsi, lors de l'ajout de sprites, les paramètres à utiliser devront ressembler à ceci :

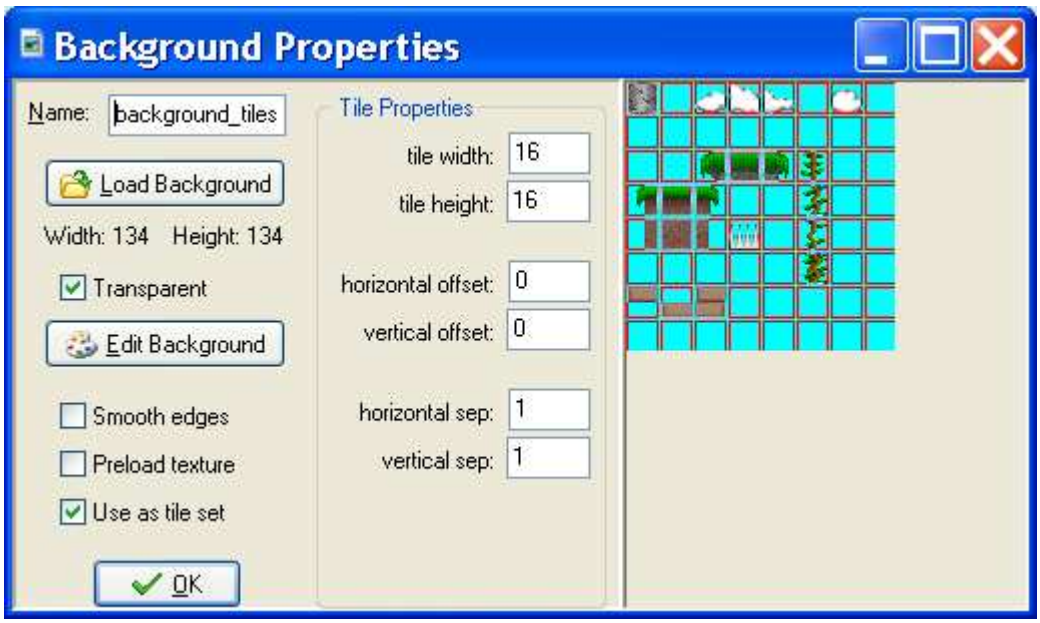


Dans les jeux plus élaborés, vous souhaiterez très certainement utiliser des sprites animés. Dans ce cas, vous aurez également besoin d'un sprite pour le personnage lorsque ce dernier ne se déplace pas. De la même façon, vous ajouterez des sprites pour le héros lorsqu'il doit sauter, tomber, tirer, etc. Par conséquent, vous devrez changer l'aspect du sprite dans les évènements à de multiples endroits.

En particulier, dans l'évènement <*no key*> (aucune touche), vous réglerez très certainement le sprite avec une image fixe représentant le héros à l'arrêt. Alternativement, vous avez la possibilité de dessiner le sprite correspondant à cette situation dans l'évènement d'affichage (*drawing event*) Par exemple, vous devrez vérifier si **`xprevious < x`** afin de savoir si le héros s'est depuis déplacé vers la droite. Comme mentionné précédemment, il est préférable de s'assurer que tous les sprites présentent la même boîte de rebond et de décocher la case « precise collision checking".

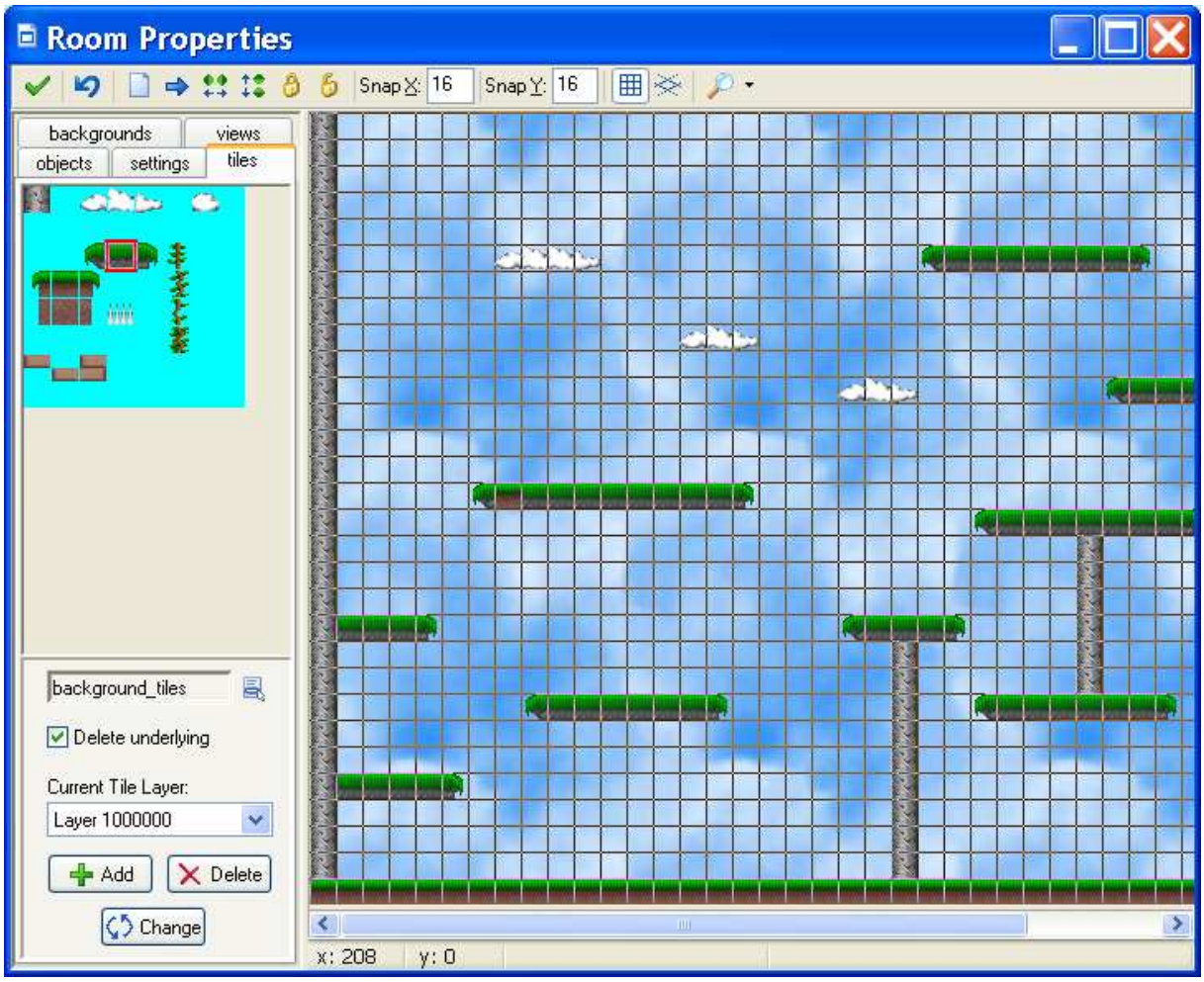
Les plates-formes et les murs

En second lieu, nous souhaiterons également améliorer l'aspect concernant l'arrière-plan et les plates-formes proprement dites. Nous utiliserons ici une technique standard. Au lieu d'utiliser des objets pour chaque élément différent composant les murs et les sols, nous allons utiliser ce que nous appellerons des **tuiles** (*tiles*) Les tuiles sont des parties d'images du décor dessinées à des endroits précis de la room. Elles ne comprennent pas d'évènements associés et ne génèrent aucune collision. Comme avantages, on pourra citer que les tuiles s'affichent rapidement et utilisent peu de mémoire. Il vous sera ainsi possible de créer des rooms gigantesques en n'utilisant que de petites images. Pour ajouter des tuiles dans vos rooms, vous aurez besoin en premier ressort d'une image de fond pour le décor qui quant à lui contiendra les tuiles. Les tuiles contenues dans une image d'arrière-plan, devront de préférence avoir une taille fixe et présenter une petite bordure ou séparation (1 pixel) entre elles, de façon à pouvoir être facilement manipulables. Pour notre petit jeu de plates-formes, nous fabriquerons nos propres tuiles que vous pourrez trouver dans le fichier source *« .gm6 »*. Nous les ajouterons en tant que ressources transparentes d'arrière-plan sous le nom **`background_tiles`**. Lors de leur ajout dans le jeu, vous indiquerez dans les propriétés d'arrière-plan qu'elles doivent être utilisées comme un jeu de tuiles (*tiles set*) et renseignerez correctement les valeurs relatives à leur taille et à la séparation, comme mentionné ci-dessous :

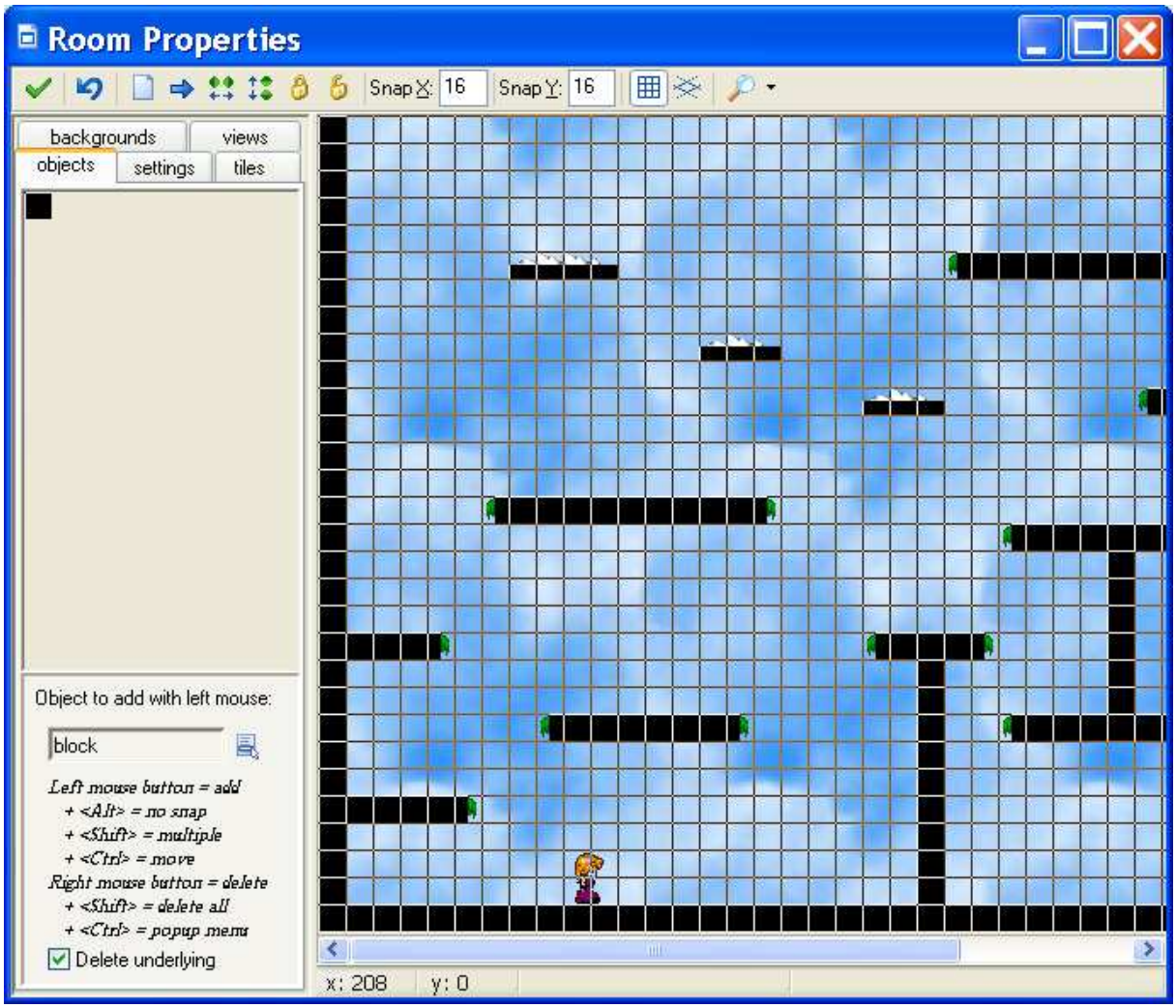


Lors de la création d'une room, il vous sera désormais possible de cliquer sur l'onglet **Tiles**. Vous choisirez ensuite le jeu de tuiles désiré (c'est à dire la ressource d'arrière-plan appropriée) Puis, vous placerez la tuile souhaitée sur la room en cliquant dessus, de la même manière que pour les objets. L'utilisation du bouton droit de la souris effacera les tuiles. A vous de vous laisser guider par votre imagination afin de créer de splendides rooms

(Veuillez bien noter qu'il vous est possible de placer les tuiles à différents niveaux de profondeur (depth layers) grâce à l'ajout de **layers** (*couches*). Par exemple, vous pourrez créer une couche (layer) de tuiles positionnées devant tous les personnages en mouvements. Nous ne les utiliserons pas dans notre jeu mais leur utilisation pourrait contribuer à créer un effet 3D des plus réussis)



Il subsiste toujours néanmoins un problème. Comme indiqué précédemment, les tuiles ne sont que de jolis graphiques, destinées à embellir le décor. Par conséquent, elles ne génèrent ni évènements ni collisions. Ainsi, le héros les traversera sans peine lors d'une chute. Afin d'éviter cela, nous aurons besoin des objets blocs décrits auparavant. Nous placerons les objets blocs aux endroits les plus appropriés comme sur le dessus des murs et des plates-formes que vous avez créées à l'aide des tuiles d'arrière-plan. Ensuite, en rendant invisibles les objets blocs, vous n'apercevrez pas les blocs noirs mais au contraire de belles tuiles à leur place. Mais les objets blocs restent bien présents et le personnage ne pourra donc pas traverser les murs et marchera correctement au-dessus des plates-formes.



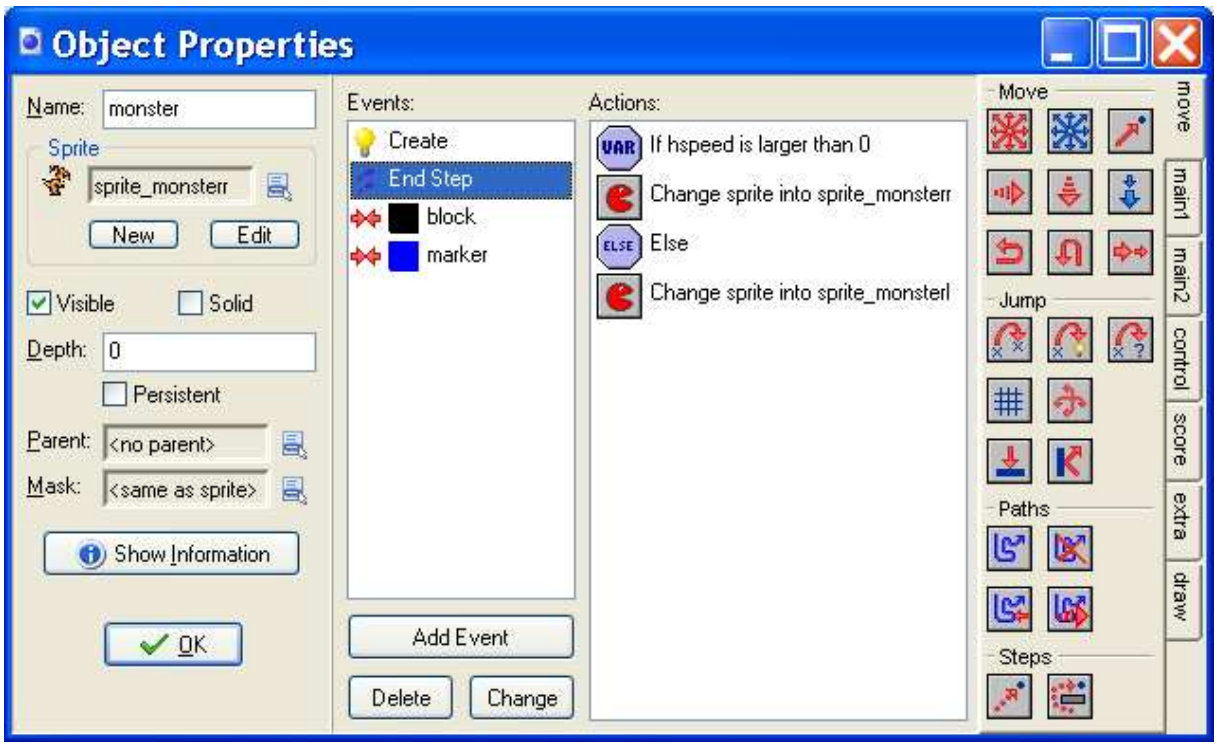
Cela pourrait d'ailleurs être un problème ici. Les objets blocs dont la taille est de 16x16, seront trop grands pour couvrir précisément l'arrière-plan. Nous allons donc créer d'autres objets blocs d'une taille inférieure comme 16x8 et 8x16. Nous les choisirons également de type solide. Afin d'éviter d'avoir à gérer des évènements de collision pour tous ces objets supplémentaires, nous utiliserons **un mécanisme d'héritage** (*parent mechanism*) Ce système est très puissant et vous devriez apprendre à l'utiliser. Définition : si un objet A est le parent d'un objet B, B présentera le même comportement que A. L'objet B héritera par conséquent de toutes les caractéristiques de l'objet A (sauf si vous modifiez par la suite certaines propriétés héritées, ce qui conduirait à modifier le comportement de l'objet B) Ainsi, les collisions avec l'objet B seront traitées de la même façon que celles survenant pour l'objet A. En ce qui concerne les plus petits blocs, nous règlerons le parent sur le plus gros bloc. De cette manière, les petits blocs seront gérés comme les plus gros.

Challenges et Objectifs

Permettre uniquement au héros de sauter d'une plate-forme à une autre pourrait se révéler ennuyeux. Il est nécessaire d'ajouter quelques challenges et objectifs. Dans cette section, nous en détaillerons un certain nombre. Consultez le jeu ***platform_3.gm6*** pour en avoir un aperçu.

Les Monstres

Ajoutons tout d'abord quelques monstres. Nous en créerons principalement deux : un qui se déplacera de gauche à droite sur la plate-forme et un autre qui évoluera dans le ciel, également dans le sens gauche - droite. Pour le premier monstre, on pourra le tuer en lui sautant dessus alors que pour le second, il devra être évité en permanence. Commençons par le monstre se déplaçant sur les plates-formes. Nous aurons besoin de deux sprites pour le représenter : un sprite pour le monstre debout vu de gauche et un autre pour le monstre debout vu de droite. Une nouvelle fois, il est préférable de ne pas utiliser le réglage *« precise collision checking*" pour les même raisons évoquées précédemment et de bien régler le paramètre *« bounding box »*.
Créons maintenant l'objet *monster* (*monstre*) Dans l'évènement de création, nous indiquerons que nous souhaitons qu'il se déplace vers la droite à une certaine vitesse. Dans le cas où il heurterait un mur, il devra inverser sa vitesse horizontale. Pour paramétrer correctement le sprite du monstre, nous utiliserons l'évènement **end step**. Cet évènement survient juste avant que les instances soient affichées. Dans cet évènement, nous règlerons le sprite concerné en agissant sur la valeur de la variable `hspeed` qui influe sur la vitesse horizontale.
Dans le cas où la vitesse serait inférieure à 0, nous afficherons le monstre se tenant debout vu de gauche et vu de droite dans le cas contraire (si vous possédez la version enregistrée de Game Maker, vous aurez même la possibilité d'utiliser une action pour inverser l'image. Vous n'auriez donc besoin dans ce cas que d'un seul sprite)



Afin de pouvoir éviter les monstres tombant des plates-formes, nous introduirons un autre objet que nous appellerons **marqueur** ou encore **repère** (*marker*)

Ce repère sera constitué d'un bloc rouge invisible. Si un monstre touche ce dernier, il devra changer de direction. Le fait de disposer de repères invisibles est en général une bonne astuce qui permettra aux instances d'effectuer certaines actions à des endroits précis de la room. Vous pourrez bien entendu employer des marqueurs pour que vos instances puissent tirer, lâcher des bombes, etc.
Lorsque le personnage heurtera un monstre, le héros devra mourir. Mais dans notre jeu et comme dans la plupart des jeux de plates-formes, nous souhaiterons donner la possibilité au personnage de sauter sur le monstre afin de l'écraser. Aussi, dans l'évènement de collision du héros avec le monstre, nous devrons vérifier à quel moment nous heurtons le monstre pour être en mesure de l'aplatir. Nous utiliserons à cet effet le test suivant :

```
vspeed > 0 && y < other.y+8
```

Le résultat du test sera positionné à *true* si la variable interne ***vspeed*** est supérieure à 0, ce qui revient à dire que le héros pourra se déplacer vers le bas et que la position en **y** du personnage est proche du bord supérieur du monstre (NDT : en fait inférieure à 8 pixels), ce qui voudra dire selon notre exemple que nous pouvons lui sauter dessus. Dans ce cas, le monstre devra être détruit (dans l'exemple, nous afficherons le monstre en utilisant un autre sprite qui représente un monstre aplati, qui s'autodétruira après un certain laps de temps. Cela produira un effet graphique très agréable) Dans ce simple jeu de plates-formes, la mort du personnage principal conduira au redémarrage du niveau de jeu actuel, redémarrage qui sera effectué à l'aide d'actions simples proposées en base par Game Maker.
Le monstre volant est encore plus facile à réaliser. Nous procéderons exactement de la même manière. La différence se situe dans l'évènement de collision du héros avec le monstre volant. Aucun test supplémentaire ne sera nécessaire ici parce que nous ne pouvons pas par définition écraser l'objet représentant le monstre volant.
Il vous sera également possible d'ajouter d'autres monstres en utilisant par exemple différentes vitesses afin de rendre les choses plus difficiles. Vous pouvez aussi créer un monstre ou un roc qui inlassablement, s'élève puis retombe brusquement. A vous de faire travailler votre imagination !

Trous

La plupart des jeux de plates-formes nécessitent une gestion précise du timing en ce qui concerne les sauts afin par exemple d'éviter de tomber dans des trous (pits) Le fait de tomber dans un trou entraîne généralement la mort du personnage. Dans ce but, nous ajouterons un nouvel objet nommé *death* (*mort*) Cet objet est lui-aussi un bloc rouge qui sera également invisible et que vous placerez au fond d'un trou. Dans l'évènement de collision du héros avec l'objet *death*, on fera jouer un son pendant un certain temps puis on relancera la room. Vous pourrez aussi créer des trous de profondeur infinie. Dans ce dernier cas, vous devrez certainement contrôler la position du personnage dans l'évènement *outside* de l'objet *heros*, en incluant le test suivant ***y > room_height*** (NDT : dans le cas présent, héros sorti par le bord inférieur de la room) afin d'être certain que le héros ne puisse pas sauter en dehors de la room et se trouver par conséquent hors de la surface de jeu.

Collecte des points

La plupart des jeux de plates-formes proposent un mécanisme avec lequel le joueur peut amasser des points. Généralement, vous devrez ramasser certains objets ou prendre des choses bien précises. Dans notre exemple, le joueur devra collecter des champignons. Par conséquent, nous créerons un objet *mushroom*. Pour diversifier encore davantage le jeu, le sprite du champignon contiendra en fait 10 sous-images différentes de champignons. L'objet *mushroom* choisira une sous-image au hasard lors de sa création en utilisant l'action suivante nécessaire pour paramétrer le sprite utilisé par l'objet :

Nous pouvons voir ici que nous avons réglé le choix aléatoire de la sous- image avec l'instruction ***random(10)***. `random(10)` est un appel de fonction. Le résultat retourné par la fonction sera un nombre déterminé aléatoirement et de valeur inférieure à l'argument fourni (soit en dessous de 10 dans notre exemple)

Nous avons paramétré la vitesse à 0 pour empêcher le défilement des autres sous-images. Dans l'évènement de collision du héros avec l'objet *mushroom*, nous jouons un son, détruisons l'objet *other* (c'est à dire l'objet *mushroom*) puis ajoutons 10 au score actuel. Dans certains jeux de plates-formes, la collecte d'objets est même plus importante que l'obtention du plus haut score. Par exemple, vous pourrez obtenir une vie supplémentaire quand vous aurez suffisamment ramassé d'objets. On pourra également trouver des objets qui régénèrent votre capital force (en supposant cependant que les monstres ne vous tuent pas mais ne font que vous endormir), vous permettent de vous déplacer plus rapidement, vous fassent sauter encore plus haut, etc.
Tout ceci peut facilement être incorporé dans votre jeu.

Niveau suivant

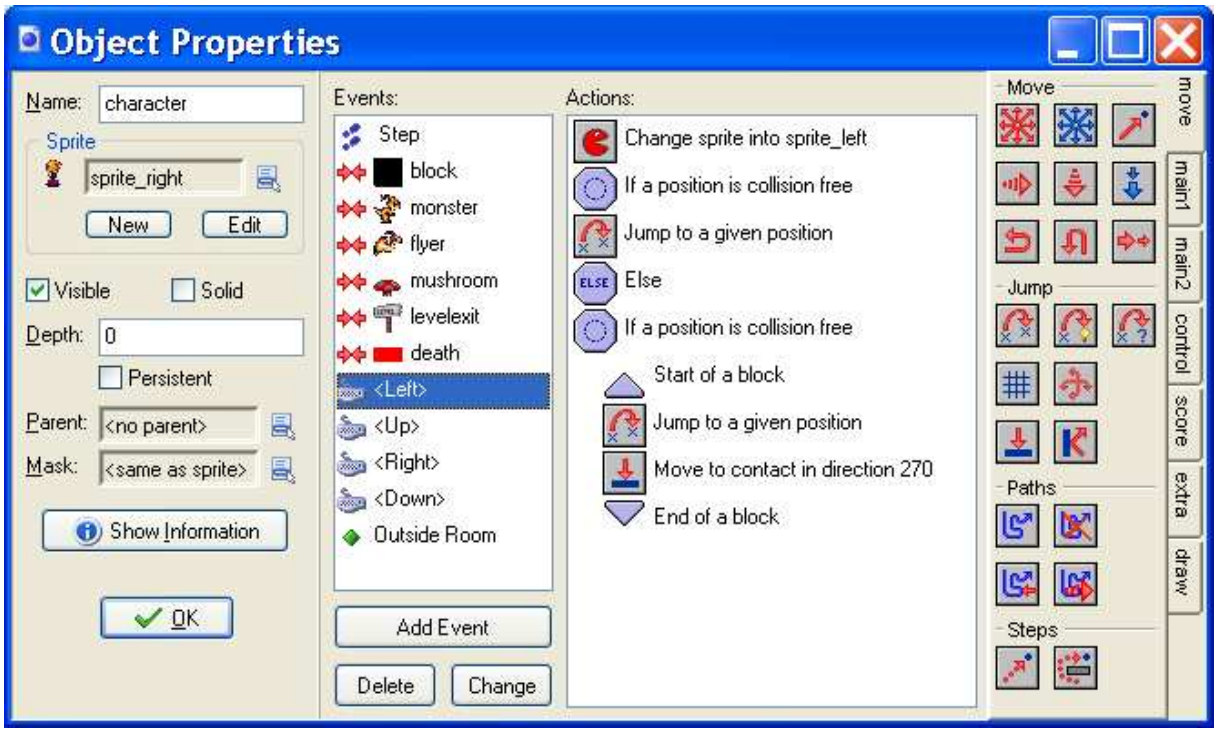
Bien entendu, il doit toujours y avoir une solution pour terminer chacun des niveaux, de manière que l'on puisse garantir que le joueur pourra accéder au niveau suivant. Dans ce but, nous créerons un nouvel objet nommé ***levelexit***. Lorsque le personnage atteindra cet objet, il passera au prochain niveau du jeu. Dans notre exemple, nous avons implémenté ceci de façon très simple. Nous ajouterons une action testant si la room suivante existe. Si le résultat du test est **vrai** alors nous afficherons le niveau suivant. Dans le cas contraire, la liste des plus hauts scores (highscore list) sera affichée puis le jeu recommencera depuis le début.
Vous pourrez faire en sorte que l'objet ***levelexit*** n'apparaisse uniquement que, par exemple, si tous les champignons ont déjà été ramassés. Pour ce faire, dans l'évènement de création de l'objet ***levelexit***, placez ce dernier à la position –100, -100 (**soit en dehors de l'écran**)
Maintenant, dans l'évènement STEP de ce même objet, nous vérifierons en permanence si le nombre d'objets *mushroom* est égal à 0 (il existe une action pour réaliser cela) et, dans l'affirmative, vous déplacerez l'objet *mushroom* à une position visible qu'il devrait avoir sur la room (là-encore, une action permet de faire ceci simplement)
Tout est très simple en fait.

Encore plus de déplacements

Notre actuel jeu de plates-formes présente des possibilités limitées en matière de déplacements. Le héros peut se déplacer à gauche, à droite ou encore sauter. Afin de rendre les choses plus intéressantes, ajoutons quelques possibilités concernant les déplacements. Le résultat peut être observé dans le fichier de jeu nommé ***platform_4.gm6***.

Rampes

Ce serait mieux encore si le joueur pouvait gravir des pentes (pour les descendre, cela se fera automatiquement dû au fait que le joueur tombera dans ce cas de figure)
Pour réaliser ceci, nous allons modifier le code situé dans l'évènement touche flèche gauche (*left arrow key event*) Nous incorporerons les choses suivantes :
Plutôt que de tester uniquement si la position à gauche est exempte de collision, nous testerons également s'il n'y a pas de collision à la position située 8 pixels plus haut. Dans l'affirmative, nous déplacerons le personnage à cet endroit puis utiliserons une action pour faire atterrir le héros (landing action) jusqu'à une position dite de contact. La description de l'évènement devrait ressembler à ceci :



La touche flèche droite sera gérée de façon similaire.

Echelles

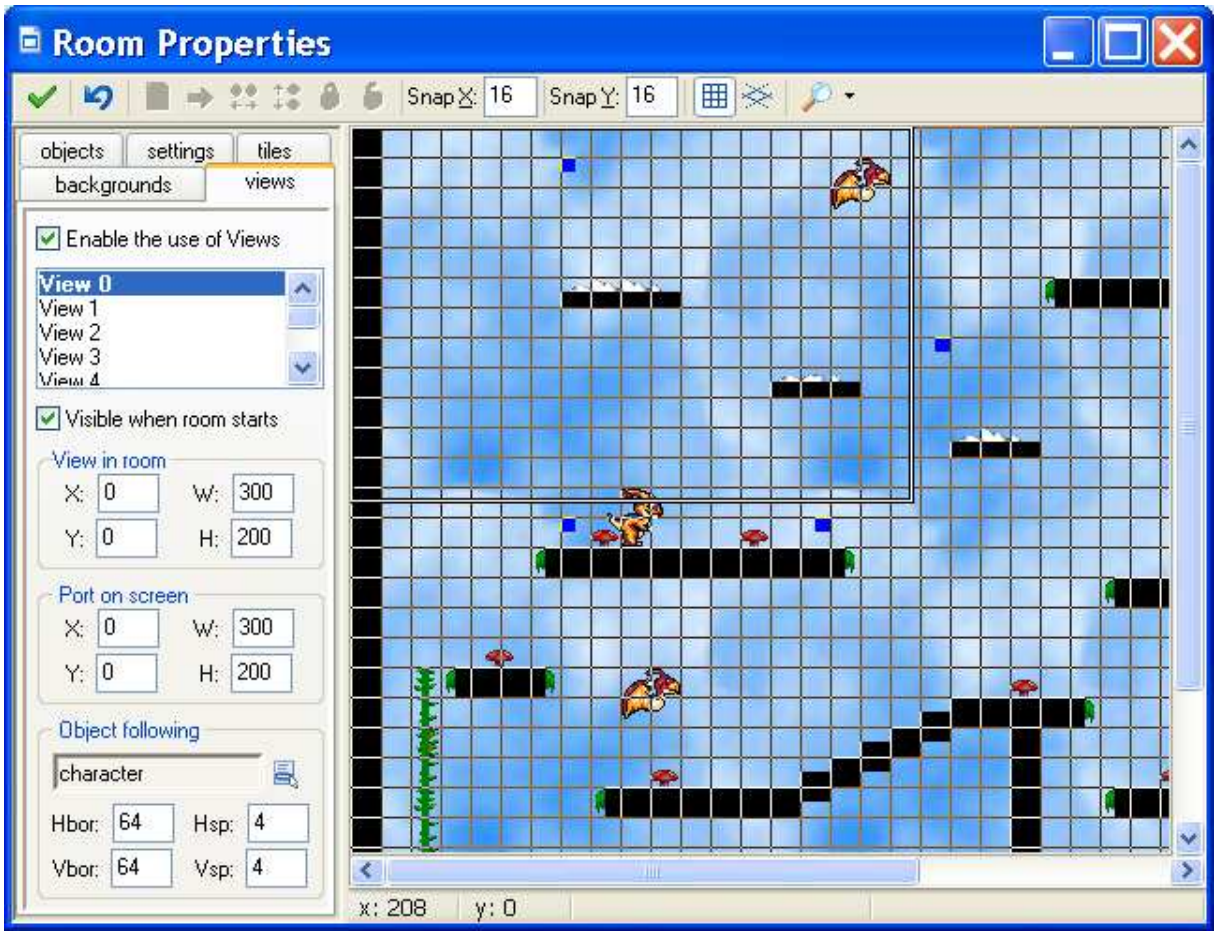
Les joueurs apprécient généralement d'avoir des échelles à leur disposition dans les jeux de plates-formes, avec lesquelles ils peuvent faire aller le héros d'une plate-forme à une autre. Cela nécessite cependant un peu plus de travail lors de la conception de votre jeu. Une échelle sera représentée par un bloc fin vertical invisible (l'échelle véritable utilisée pour faire grimper le joueur est quant à elle affichée à l'écran en utilisant des tuiles (tiles) et ne doit pas être de type solide) Quand le héros n'est pas en contact avec une échelle, les déplacements doivent se faire comme à l'accoutumée. Par contre, si le personnage entre en contact avec l'échelle, les choses sont alors à gérer différemment.
En premier lieu, le héros ne doit pas pouvoir tomber. Ainsi, dans l'évènement STEP, nous devrons effectuer quelques modifications pour prendre en compte cet aspect en ajoutant des actions comme régler la vitesse verticale à 0 ainsi que la gravité. Nous devrons donc afficher un sprite *climbing* (sprite montant) dans ce cas.

La seconde chose qu'il est nécessaire de modifier, est l'évènement de gestion de la touche flèche haute. Quand le héros est sur une échelle, la touche flèche haute doit lui permettre de l'escalader et non pas de lui faire faire un saut.

Nous aurons donc besoin là encore de quelques actions supplémentaires pour réaliser tout cela. Nous testerons si le joueur est en contact avec une échelle et, dans l'affirmative, le ferons monter de quelques pixels. Nous utiliserons ce même genre d'actions en ce qui concerne la touche flèche basse.

Utilisation d'une vue

Jusqu'à maintenant, nous avons toujours affiché la room dans son intégralité. Pour la plupart des jeux de plates-formes, cela ne sera pas généralement l'effet souhaité. Nous voudrons la plupart du temps ne pouvoir afficher qu'une petite partie de la room, celle qui se trouve proche du personnage que nous devons contrôler. Ceci rendra le jeu encore plus intéressant et parfois plus difficile car le joueur devra essayer de trouver son chemin à travers les plates-formes sans pour autant disposer d'une vue d'ensemble de la room. Vous pouvez aussi camoufler des trésors à des endroits de la room que vous jugerez difficiles d'accès.
Heureusement, la réalisation de tout ceci est extrêmement simple avec Game Maker. Lors de la conception de la room, cliquez sur l'onglet **Views** (*Vues*) Cochez ensuite la case **Enable the use of Views** (*Permettre l'utilisation des Vues*) pour autoriser l'utilisation des vues. Choisissez ensuite la première vue puis cochez la case **Visible when room starts** (*Visible lors du lancement de la room*) afin d'être sûr que cette vue sera visible. Donnez-lui une largeur de 300 pixels ainsi qu'une hauteur de 200 pixels (ou toutes autres valeurs que vous souhaitez) Comme nous désirons seulement que la vue suive le personnage, il ne sera donc pas nécessaire d'indiquer les positions gauche et supérieure de la vue dans la room. De même, parce que nous n'utilisons qu'une seule vue, nous n'aurons pas à préciser les positions **x** et **y** du port de l'écran (voir figure ci-dessous) En bas de la fenêtre, nous indiquerons quel est l'objet que doit suivre la vue. Nous choisirons ici bien entendu le héros. Désormais, la vue se déplacera automatiquement et suivra les déplacements du personnage. Comme nous ne souhaitons pas que le héros soit trop près de la bordure, nous réglerons à 64 les valeurs **Hbor** et **Vbor**. Par conséquent, il y aura en permanence une zone de 64 pixels visible autour du personnage. Enfin, pour obtenir un déplacement doux et progressif, nous paramètrerons à 4 la vitesse maximale de la vue (ceci permettra d'avoir, au début de l'affichage de la vue, un très joli effet qui fera apparaître lentement le héros) La figure ci-dessous nous montre l'ensemble des réglages :



L'utilisation d'une vue rendra le jeu plus attirant mais aura aussi pour effet de réduire la taille de la fenêtre où se déroule le jeu. Afin d'éviter cela, nous devrons indiquer sous la rubrique **General Game Settings** (*Réglages généraux du jeu*) une échelle fixe de 200 pourcents. Vous pourrez faire varier ces valeurs afin d'obtenir l'effet souhaité.

Quelques améliorations supplémentaires

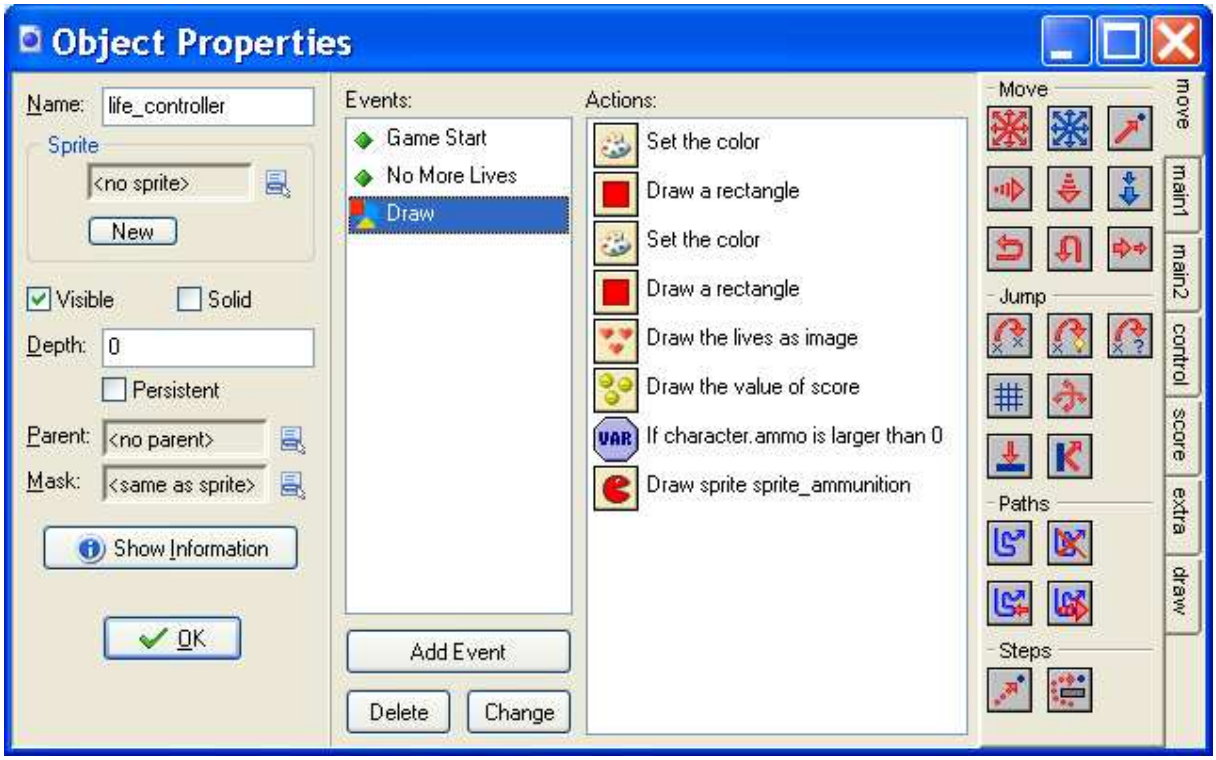
Tirer sur les monstres

La prochaine étape consistera à permettre au joueur de tirer sur les monstres. Pour améliorer quelque peu le gameplay, le joueur aura besoin en premier lieu de munitions avant de pouvoir tirer. A cet effet, nous allons utiliser une variable nommée ***ammo*** qui indique le nombre de munitions dont dispose le joueur. Dans l'évènement de création du personnage, nous réglerons celle variable à 0 en utilisant l'action ad hoc permettant de paramétrer une variable ( ) L'objet *ammunition* possède un sprite unique qui ne fait rien de particulier. Ce sprite attend seulement d'être sélectionné par le joueur. Quand le héros entre en collision avec l'objet *ammunition*, nous ajoutons la valeur 10 à la variable ***ammo*** (en lui affectant 10 en relative) puis détruisons l'instance *ammunition*. Ensuite, nous aurons besoin d'un objet *bullet*. Lorsque le joueur pressera la touche *<Espace>*, une instance de cet objet devra être créée, si toutefois le joueur dispose encore de munitions puis la variable ***ammo*** sera décrémentée de 1. Mais il y a un aspect encore plus important à prendre en compte. Nous souhaiterions que la balle (*bullet*) se dirige dans la direction du héros. Pour ce faire, nous allons vérifier la valeur de la variable ***sprite_index***. Cette variable contient l'index du sprite pour l'objet du personnage. A partir du contenu de cette variable, nous créerons une balle dans le sens de déplacement approprié. Dans le cas où le joueur tomberait, aucune balle ne sera créée (car il n'est pas possible de tirer tout en tombant...) En résumé, l'évènement *space* pourrait se présenter comme suit :

Il nous reste à détruire la balle lorsqu'elle heurte un mur ou encore sort de la room mais il faudra aussi préoir de tuer le monstre quand la balle l'aura touché. Tout cela est facile à réaliser. Consultez le fichier de jeu ***platform_5.gm6*** pour avoir la dernière version du jeu.

Tableau des scores

A ce stade, le joueur possède un score et des munitions. Nous allons aussi lui donner quelques vies. Le fait de toucher un monstre ou de tomber dans un trou coûtera une vie au joueur. Game Maker propose un système simple pour assurer la gestion de vies dans les jeux.
Nous allons pour cela créer un objet spécial nommé ***life_controller***. Il n'est pas nécessaire de lui associer un sprite. Dans son évènement de création, nous initialiserons à 3 le nombre de vies. Lorsque le joueur devra mourir, nous diminuerons le nombre de vies d'une unité. Dans l'évènement **No more lives** ("*Plus de vie*") de cet objet contrôleur, nous afficherons la table des plus hauts scores (*highscore*) puis relancerons le jeu.
Mais cela serait encore mieux si nous pouvions voir en permanence à l'écran le nombre de vies, le score, le nombre de munitions restantes, etc. Dans ce but, nous allons créer un petit panneau reprenant ces informations. Nous afficherons tout ceci dans l'évènement **Draw** de l'objet contrôleur. Il subsiste cependant un problème. Où devons-nous afficher tout ceci ? Nous ne pouvons pas afficher cela à un endroit fixe de la room car la vue change à tout moment et nous souhaitons que le panneau soit toujours présent dans la vue. Fort heureusement, il nous est possible à chaque instant de connaître la position de la vue. Les deux variables suivantes ***view_xview*** et ***view_yview*** indiquent respectivement la position gauche et supérieure de la vue. Nous pouvons donc désormais afficher le panneau en utilisant ces informations. Vous trouverez ci-dessous les actions de l'évènement **draw** de l'objet contrôleur :



Veuillez noter que nous pourrions aussi afficher une image lorsque le joueur a la possibilité de tirer. Le résultat apparaît dans l'image ci-dessous et peut être trouvé dans le fichier de jeu ***platform_6.gm6*** :

Et la suite ?

Ce tutoriel vous a présenté certaines bases que vous pourrez utiliser lors de la conception de vos propres jeux de plates-formes. Mais c'est maintenant à vous de jouer ! Vous devrez utiliser ces techniques mais également les vôtres ainsi que vos propres idées afin de pouvoir créer un bon jeu de plates-formes. N'oubliez-pas que les niveaux de jeu (*levels*) constituent la partie la plus importante des jeux de plates-formes.

Commencez à créer un par un vos propres "levels". Testez-les en y jouant vous-mêmes jusqu'à ce que vous soyez satisfait de chacun d'entre eux. N'hésitez-pas à introduire de nouveaux aspects dans votre jeu pour améliorer son gameplay. Voici quelques idées supplémentaires que vous pourrez utiliser :

- Ajoutez des monstres différents, des balles rebondissant ou des monstres qui tirent.
- Insérez des clés que vous devrez trouver avant de pouvoir ouvrir des portes.
- Placez à certains endroits des mines qui explosent lorsqu'un monstre (ou encore le joueur) marche dessus.
- Affichez de l'eau où il sera possible de nager (cela changera d'ailleurs complètement la gestion des déplacements : plus de gravité ou encore une légère pesanteur allant en augmentant jusqu'à ce que vous atteigniez la surface, temps limité avant que vous ne manquiez d'air, des bulles d'air à saisir, etc.)
- Placez des murs et des sols que vous pourrez détruire par exemple en tirant dessus ou en sautant dessus avec force.
- Ajoutez des trampolines vous permettant de sauter encore plus haut.
- Insérez des plates-formes qui apparaissent et disparaissent.
- Mettez en place des rues à sens unique.
- Affichez des plates-formes se déplaçant latéralement ou verticalement (cela n'est pas facile à implémenter !)
- …etc...

Traduction française par Philippe Ragni alias Xierra54

Il vous faudra bien sûr une bonne dose de persévérance !

Je vous souhaite à tous bonne chance dans la réalisation de vos jeux avec Game Maker...

NOTES

NOTES