Game Maker Tutorial

Creating Maze Games

Written by Mark Overmars

Copyright © 2007-2009 YoYo Games Ltd
Last changed: December 23, 2009
Uses: Game Maker 8.0, Lite or Pro Edition, Advanced Mode
Level: Beginner

Maze games are a very popular type of game and they are easy to create in *Game Maker*. This tutorial shows in a number of easy-to-follow steps how to create such a game. The nice part is that from the first step on we have a playable game that, in further steps, becomes more extended and more appealing. All partial games are provided in the folder `Examples` that comes with this tutorial and can be loaded into *Game Maker*. Also all resources are provided in the folder `Resources`.

The Game Idea

Before starting creating a game we have to come up with an idea of what the game is going to be. This is the most important (and in some sense most difficult) step in designing a game. A good game is exciting, surprising and addictive. There should be clear goals for the player, and the user interface should be intuitive.

The game we are going to make is a maze game. Each room consists of a maze. To escape the maze the player must collect all diamonds and then reach the exit. To do so the player must solve puzzles and monsters must be avoided. Many puzzles can be created: blocks must be pushed in holes; parts of the room can be blown away using bombs, etc. It is very important to not show all these things in the first room. Gradually new items and monsters should appear to keep the game interesting.

So the main object in the game is a person controlled by the player. There are walls (maybe different types to make the maze look more appealing). There are diamonds to collect. There are items that lie around that do something when picked up or touched by the player. One particular item will be the exit of the room. And there are monsters that move by themselves. But let us tackle these things one by one.

第一關：先設計一個容易達成的簡單關卡 -- 只有 player, wall, exit.

A Simple Start

As a first start we forget about the diamonds. We will create a game in which the player simply must reach the exit. There are three crucial ingredients in the game: the player, the wall, and the exit. We will need a sprite for each of them and make an object for each of them. You can find the first very simple game under the name `maze_1.gmk`. Please load it and check it out.

1.先建立三個Sprites - spr_person, spr_wall, spr_goal
2.再建立三個Objects - obj_person, obj_wall, obj_goal
(x solid) (solid) (x solid)
3. 設定obj_goal 的Event: 如果person碰到goal，就要準備進下一個關卡

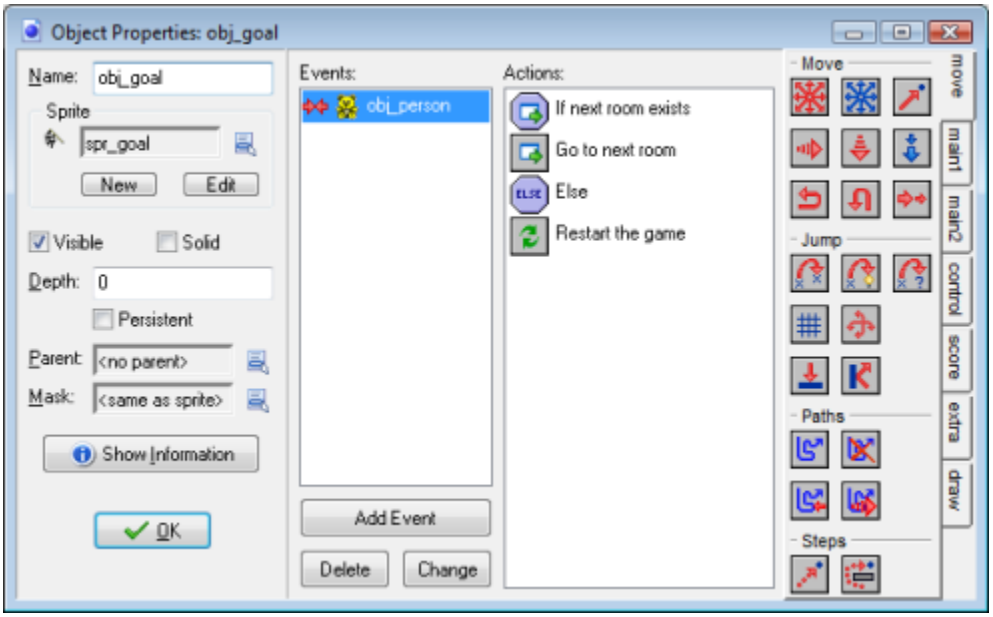
The objects

Let us first create the objects. For each of the three objects we use a simple 32x32 sprite:

Create these three sprites in the usual way and name them `spr_person`, `spr_wall`, and `spr_goal`.

Next we create three objects. Let us first make the wall object. We will give it the `spr_wall` sprite as image, name it `obj_wall` and make it solid by checking the box labeled **Solid**. This will make it impossible for other objects, in particular the person, to penetrate the wall. The wall object does not do anything else. So no events need to be defined for it.

Secondly, let us create the goal object. This is the object the player has to reach. It is a non-solid object. We decided to give it a picture of a finish flag. This makes it clear for the player that he has to go here. When the person collides with it we need to go to the next room. So we put this action in this collision event (it can be found in the tab **main1**). This has one drawback. It causes an error when the player has finished the last room. So we have to do some more work. We first check whether there is a further room. If so we move there. Otherwise we restart the game. So the event will look as follows:



Obviously, in the full game we better do something more when the player finishes the last level, like showing some nice image, or giving him a position in the list of best players. We will consider this later.

Finally we need to create the person that is controlled by the player. Some more work is required here. It must react to input from the user and it should not collide with a wall. We will use the arrow keys for movement. (This is natural, so easy for the player.) There are different ways in which we can make a

4. 接下來 要讓小熊(person)可以動 -- 利用方向鍵 (arrow key)來控制

person move. The easiest way is to move the player one cell in the indicated direction when the player pushed the arrow key. A second way, which we will use, is that the person moves in a direction as long as the key is pressed. Another approach is to keep the player moving until another key is pressed (like in PacMan).

5.這裡簡略描述四個方向Keyboard Event - UP, Down, Left, Right (速度分別設定為 4)

We need actions for all for arrow keys. The actions are rather trivial. They simply set the right direction of motion. (As speed we use 4.) To stop when the player releases the key we use the keyboard event for <no key>. Here we stop the motion. There is one complication though. We really want to keep the person aligned with the cells of the grid that forms the maze. Otherwise motion becomes rather difficult. E.g. you would have to stop at exactly the right position to move into a corridor. This can be achieved as follows. In the **control** tab there is an action to test whether the object instance is aligned with a grid. Only if this is the case the next action is executed. We add it to each arrow key event and set the parameters to 32 because that is the grid size in our maze:

6. 記得要再加設一個Keyboard <no key> 的Event: 就是停在中間點速度設為 0 。讓它可以停得下來。

7. 為了確定小熊的每一步都在格點上，我們再加精細一點的設定：把步驟 5 提到的四個方向按鍵事件，都加上左圖這個「Check Grid」的設定。垂直 水平方向各設為 32.
(ps. 「Check Grid 在 Control 那一類動作中)

8. 下面這一段，要教你處理小熊撞到牆要停下來：方法沒說，妳自己知道如何做嗎?
Hint: collision_with wall --> 停在中間點，速度設為 0

Clearly we also need to stop the motion when we hit a wall. So in the collision event for the person with the wall we put an action that stops the motion. There is one thing you have to be careful about here. If your person's sprite does not completely fill the cell, which is normally the case, it might happen that your character is not aligned with a grid cell when it collides with the wall. (To be precise, this happens when there is a border of size larger than the speed around the sprite image.) In this case the person will get stuck because it won't react to the keys (because it is not aligned with the grid) but it can also not move further (because the wall is there). The solution is to either make the sprite larger, or to switch off precise collision checking and as bounding box indicate the full image.

Creating rooms

9. 接下來，就建立一、二個rooms 來測試一下。

That was all we had to do in the actions. Now let us create some rooms. Create one or two rooms that look like a maze. In each room place the goal object at the destination and place the person object at the starting position.

每一個room 都要在終點放一面旗子(goal)，
在起點放一隻小熊(person)。 中間的迷宮自己設計一下哦!!!

做到這裡，休息一下~~ 存個檔，試玩一下。檔名是 move1.gmk
也可以改變一下小熊的速度、改一下角色的圖示或是再加關卡哦!!!

Done

And that is all. The first game is ready. Play a bit with it. E.g. change the speed of the person in its creation event, create some more levels, change the images, etc.

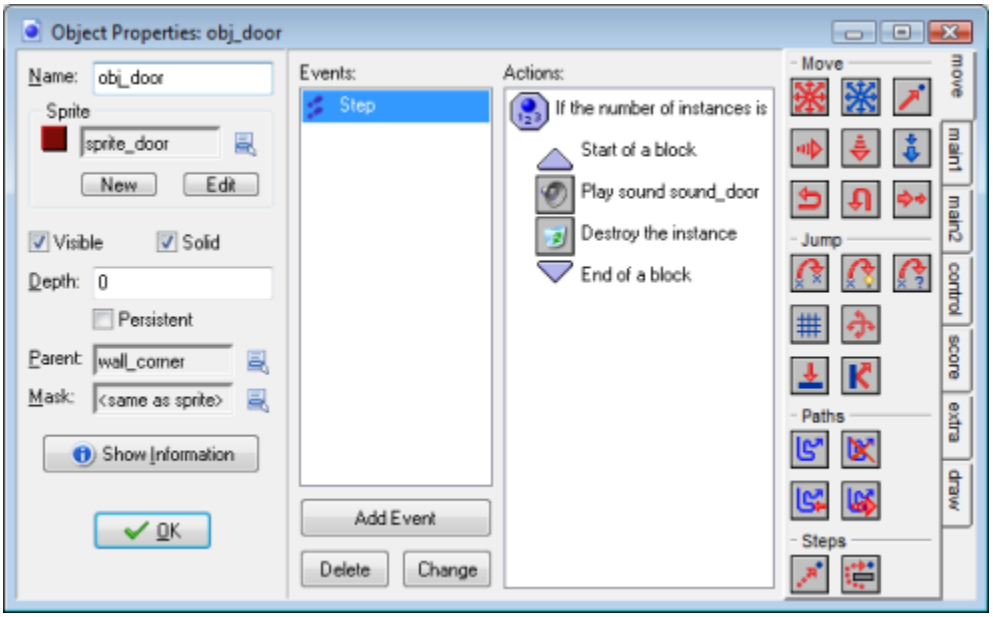
這一段教我們在遊戲中加上收集鑽石的任務，如果鑽石收集完了，在終點擋住goal的這扇門就會消失，你也可以進入下一關。

Collecting Diamonds

But the goal of our game was to collect diamonds. The diamonds itself are easy. But how do we make sure the player cannot exit the room when not all diamonds are collected? To this end we add a door object. The door object will behave like a wall as long as there are still diamonds left, and will disappear when all diamonds have gone. You can find the second game under the name `maze_2.gmk`. Please load it and check it out.

1.所以，我們要先來加兩個物件： diamond, door(當然，要記得先做兩個 Sprites 哦)

Beside the wall, goal, and person object we need two more objects, with corresponding sprites: the diamond and the door. The diamond is an extremely simple object. The only action it needs is that it is destroyed when the person collides with it. So in the collision event we put an action to delete it. The door object will be placed at a crucial place to block the passage to the goal. It will be solid (to block the person from passing it). In the collision event of the person with the door we must stop the motion. In the step event of the door we check whether the number of diamonds is 0 and, if so, destroys itself. There is an action for this. We will also play some sound such that the player will hear that the door is opened. So the step event looks as follows:



2.處理物件的規則
Dimond Object:
(1) collision with person-destroy itself
3. Door Object:
(1) 必須是Solid
(2) 要放在重要的地方，擋住取得goal 的通道
(3) collision with person-stop the motion
(4) step事件-(指的是，當故事進行到某一步時：
如果 dimond 的數量變為0
{
play the sound_door
destory (itself) (門消失)
}

Making it a bit nicer

Now that the basics of the game are in place, let us make it a bit nicer.

以下這一小段在教你如何讓畫面看起來更美。新增3個牆壁物件，1個放在角落(corner)，1個當作垂直方向(vertical)的牆，1個當作水平方向(horizontal)的牆。(別忘了先做sprites)
接著，新增一張背景圖。到場景裡設定背景圖(Background)

The walls look pretty ugly. So let us instead make three wall objects, one for the corner, one for the vertical walls, and one for the horizontal walls. Give them the right sprites and make them solid. Now with a bit of adaptation of the rooms it looks a lot nicer. Giving the rooms a background image also helps.

以下是父子物件的概念：子物件可以繼承父物件的所有行為。(把所有wall物件的parent都設成是 corner)

To avoid having to specify collision events of the person with all these different walls (and later similar for monsters), we use an important technique in *Game Maker.* We make the corner wall object the parent of the other wall objects. This means that the wall objects behave as special variants of the corner wall object. So they have exactly the same behavior (unless we specify different behavior for them) Also, for other instances, they are the same. So we only have to specify collisions with the corner. This will automatically be used for the other wall objects. Also the door object we can give as parent the corner wall.

為的是避免 person 遇見同一類型的物件卻發生不同的行為。

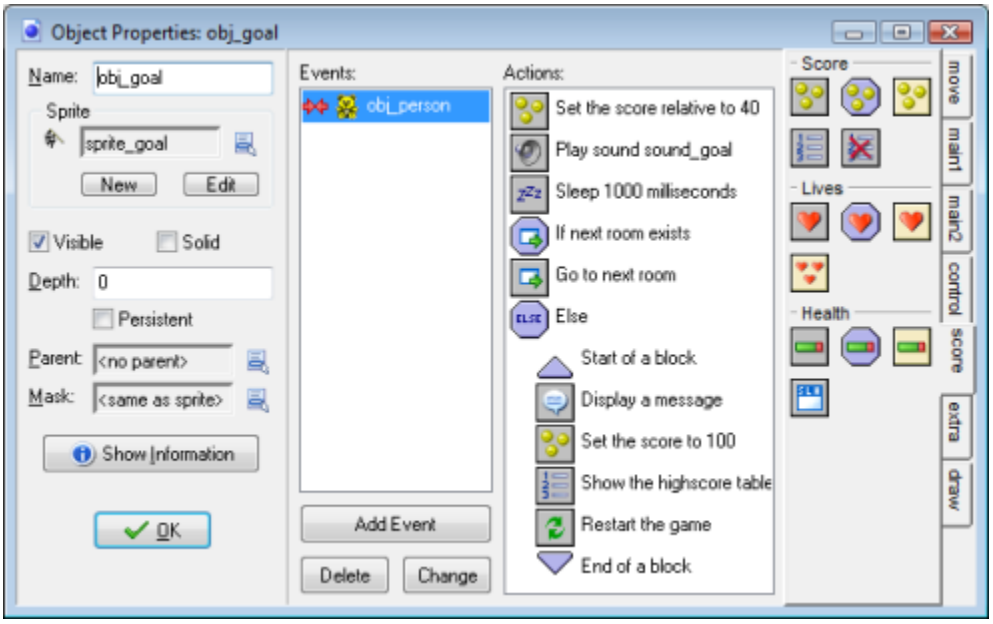
4. 來設一下計分規則： (1) dimond - 被destroy 時，要加 5 分
(2) goal- 目標達成時，要加40分

Score

Let us give the player a score such that he can measure his progress. This is rather trivial. For each diamond destroyed we give 5 points. So in the destroy event of the diamond we add 5 points to the score. Finishing a level gives 40 points so we add 40 points to the score in the collision event for the goal with the person.

想一想，你會怎麼做?

When the player reaches the last room a high-score table must be shown. This is easy in *Game Maker* because there is an action for this. The goal object does become a bit more complicated though. When it collides with the person the following event is executed:



5. 再修一下小熊到達目地的時的細節：

(1) 加 40 分
(2) 發出到達目地的的音效
(3) 暫停 1 秒
(4) 如果下一關存在
則進到下一關
否則
{
show 出遊戲結束訊息
把分數設成 100 分
show 出分數表
遊戲重新開始
}

It adds something to the score, plays a sound, waits a while and then either goes to the next room of, if this is the last room, shows a message, the high-score table, and restarts the game.

Note that the score is automatically displayed in the caption. This is though a bit ugly. Instead we will create a controller object. It does not need a sprite. This object will be placed in all rooms. It does some

註：目前為止，分數是顯示在標題列，看起來有點醜。後面，我們要改變一點作法，自己來畫分數表：
(1) 我們來創一個 controller 物件 (這個物件不需要使用圖像sprites哦)
(2) 這個 controller 物件 必須被放在每一個 rooms 裡，這樣就可以貫穿在整個遊戲中 (global control)

作法：對controller object 設定 「drawing」 event：
Action- set font and color (設定字型和顏色)
draw the score

global control of what is happening. For the moment we just use it to display the score. In its drawing event we set the font and color and then use the action to draw the score.

Starting screen

接下來，教我們如何設計遊戲的開始畫面

It is nice to start with a screen that shows the name of the game. For this we use the first room. We create a background resource with a nice picture. (You might want to indicate that no video memory should be used as it is only used in the first room.) This background we use for the first room (best disable the drawing of the background color and make it non-tiled.) A start controller object (invisible of course) is created that simply waits until the user presses a key and then moves to the next room. (The start controller also sets the score to 0 and makes sure that the score is not shown in the caption.)

1. 在第一個 room 裡，找一張很棒的圖片當背景圖 (留意上文告訴你的設定)
2. 做一個 start_controller 物件：設定(1) 按任意鍵 -> 進入下一個 room (2) creat->score 設為 0

Sounds

A game without sounds is pretty boring. So we need some sounds. First of all we need background music. For this we use some nice midi file. We start this piece of music in the `start_controller`, looping it forever. Next we need some sound effects for picking up a diamond, for the door to open, and for reaching the goal. These sounds are called in the appropriate events described above. When reaching the goal, after the goal sound, it is good to put a little sleep action to have a bit of a delay before going to the next room.3.設定背景音樂： (1) 找一個很棒的 midi 檔當背景音樂
(2) 把設定寫在 start_controller 物件上 - creat -> play the sound(loop)

Creating rooms

Now we can create some rooms with diamonds. Note that the first maze room without diamonds we can simply leave in. This is good because it first introduces the player to the notion of moving to the flag, before it has to deal with collecting diamonds. By giving a suggestive name to the second room with the diamonds, the player will understand what to do.

4.多建幾個關卡- 第一關不要有 diamonds, 第二關命名為 搜集 diamond，玩家就會知道任務是什麼
做到這裡，休息一下。 另存一個檔名 叫做 move2.gmk

Monsters and Other Challenges

The game as it stands now starts looking nice, but is still completely trivial, and hence boring to play. So we need to add some action in the form of monsters. Also we will add some bombs and movable blocks and holes. The full game can be found in the file `move_3.gmk`.

接下來，要增加難度與挑戰。增加 monster, 炸彈、可移動的方塊和一些 洞。

Monsters

Monsters 有三種，各自有自己的移動方式和行為。

We will create three different monsters: one that moves left and right, one that moves up and down, and one that moves in four directions. Adding a monster is actually very simple. It is an object that starts moving and changes its direction whenever it hits a wall. When the person hits a monster, it is killed, that is, the level is restarted and the player looses a life. We will give the person three lives to start with.

Let us first create the monster that moves left and right. We use a simple sprite for it and next create an object with the corresponding sprite. In its creation event it decides to go either left or right. Also, to make life a bit harder, we set the speed slightly higher. When a collision occurs it reverses its horizontal direction.

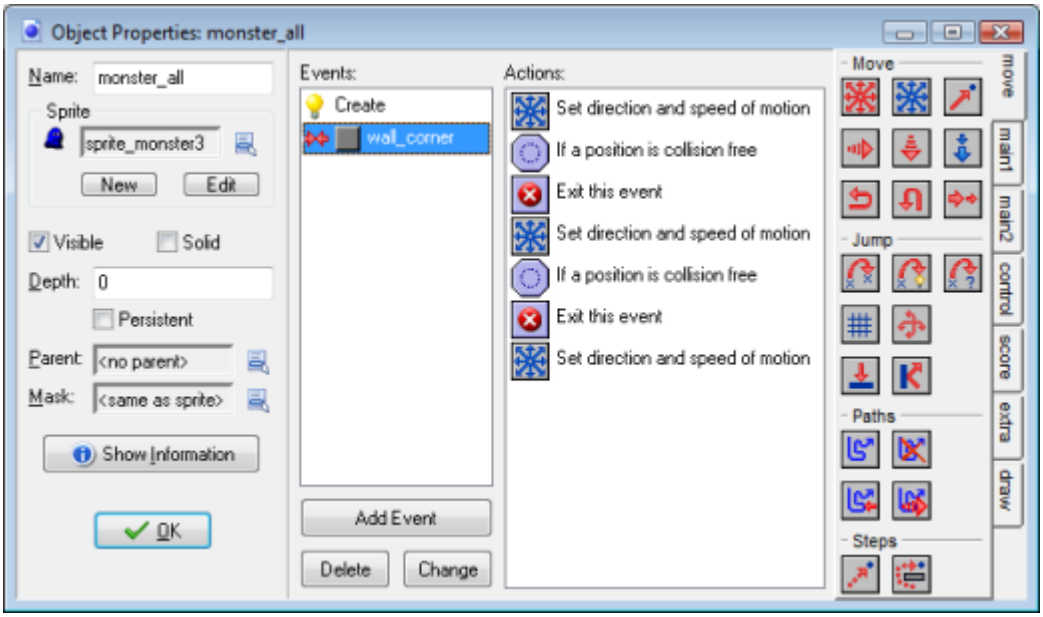
monster1: create -> moves left and right (speed - 設較快些... 你可以想想這個值要多少)
collision_with_wall -> reverse (改變其水平方向) horizontal direction

monster2: create -> moves up and down
collision_with_wall -> reverse (改變其垂直方向)vertical direction

monster3: create -> moves four direction
collision_with_wall -> reverse direction

The second monster works exactly the same way but this time we start moving either up or down and, when we hit a wall, we reverse the vertical direction.

The third monster is slightly more complicated. It starts moving either in a horizontal or in a vertical direction. When it hits a wall it looks whether it can make a left or a right turn. If both fail it reverses its direction. This looks as follows:



To avoid problems with monsters being slightly too small, we uncheck precise collision checking and modify the mask to set the bounding box to the full image.

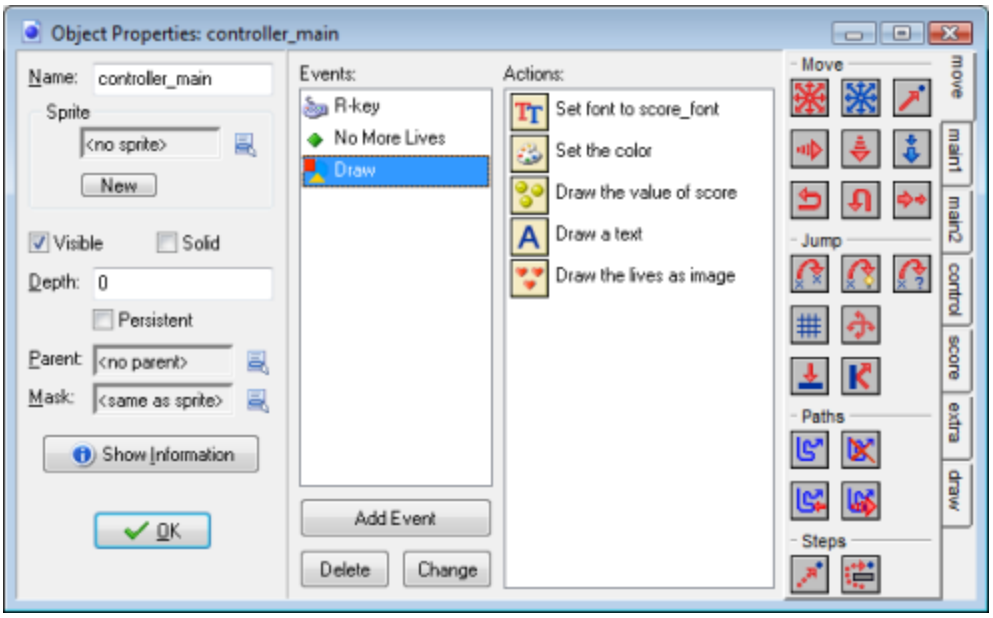
A. When the person collides with a monster, we have to make some awful sound, sleep a while, decrease the number of lives by one, and then restart the room. (Note that this order is crucial. Once we restart the room, the further actions are no longer executed.) The controller object, in the "no more lives" event, shows the high-score list, and restarts the game. B.

Lives

We used the lives mechanism of *Game Maker* to give the player three lives. It might though be nice to also show the number of lives. The controller object can do this in the same way as with the score. But it is nicer if you actually see small images of the person as lives. There is an action for this in the **score** tab. The drawing event now looks as follows:

A. Monster object: collision_with_person -> play some awful sound
sleep a while (1000 millseconds)
decrease lives (1) 把生命值減1
restart the room (這一關要重玩)

B. Controller object: no more lives -> show the high-score list
restart the game



這張圖教我們
如何畫出漂亮的分數表

Note that we also used a font resource to display the score.

加一些炸藥，並且加一些 trigger 趨使它們引爆。

Bombs 想法：當玩家碰到這些 trigger, 所有的 bombs 都會爆炸，並且把週圍的所有東西都炸掉。

Let us add bombs and triggers to blow them up. The idea is that when the player gets to the trigger, all bombs explode, destroying everything in their neighborhood. This can be used to create holes in walls and to destroy monsters. We will need three new objects: a trigger, a bomb, and an explosion. For each we need an appropriate sprite. 1. 現在，你需要再建3 個物件- trigger, bomb, explosion (記得要先建 sprites)

The bomb is extremely simple. It just sits there and does nothing. To make sure monsters move over it (rather than under it) we set its depth to 10. Object instances are drawn in order of depth. The ones with the highest depth are drawn first. So they will lie behind instances with a smaller depth. By setting the depth of the bomb to 10 the other objects, that have a default depth of 0, are drawn on top of it.

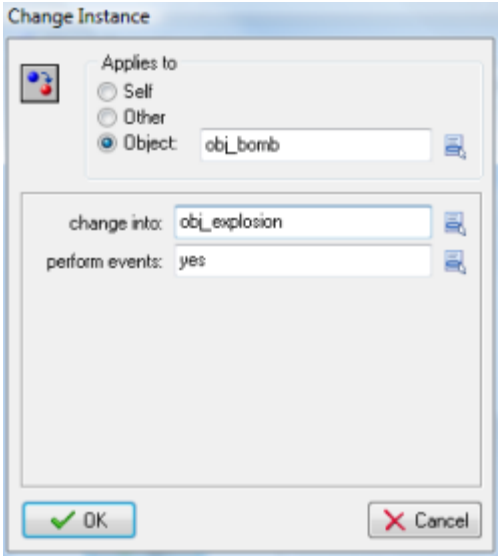
The trigger is also rather simple. When it collides with the person it turns all bombs into explosions. This can be achieved by using the action to change an object in another object. At the top we indicate that it should apply to all bombs.

2. bomb 的設定超簡單的，它就放在那裡，什麼事也不用做。
但是要注意，把這個物件的 「深度」(depth) 設為 10，讓它在比較底層，確保 monster 可以走過它。
（一般物件預設深度是 0，數字越大表示越早畫，所以會放在越底層)

3. trigger 的設定也超簡單的。當它被 小熊(person) 碰到，就把所有 bombs 引爆
怎麼做呢?
collsion Event (with person) -> change an object into another object
(bombs) (explosions)

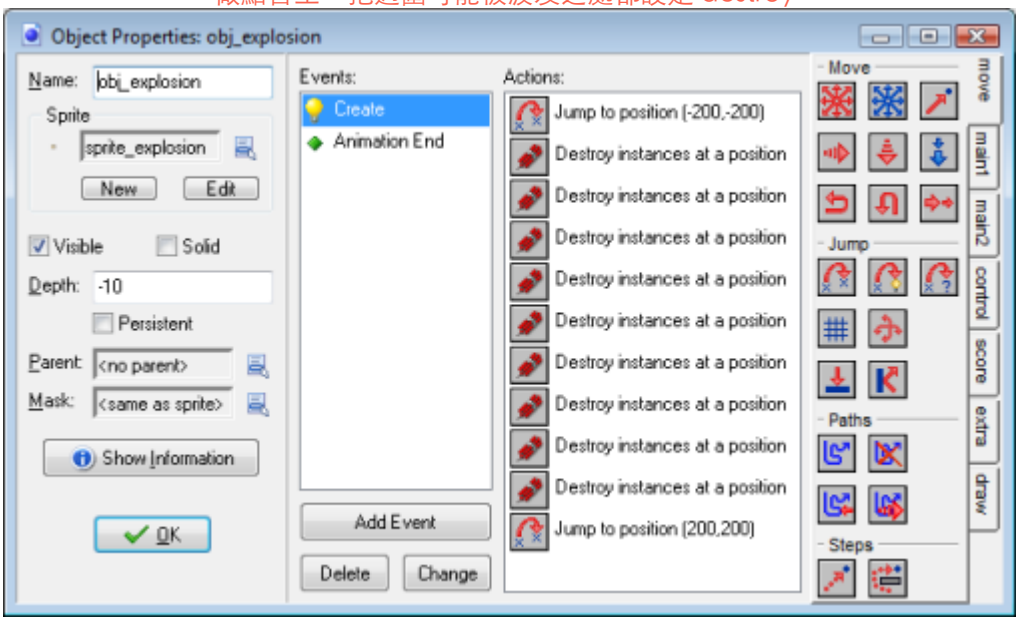
這樣做，會把全部的 bombs 一次全部換成 爆炸的物件 explosion

做完剛才的設定，就會看見像這樣~~



4. 注意： explosion object 是一段動畫，所以，要設定當動畫結束(Animation End), 要把自己 destroy

The explosion object just shows the animation. After the animation it destroys itself. (You have to be careful that the origin of the explosion is at the right place when turning a bomb into it.) The object also must destroy everything it touches. This requires a little bit of work. First of all, we do not want the explosion to destroy itself so we move it temporarily out of the way. Then we use actions to destroy all instances at positions around the old position of explosion. Finally we place the explosion back at the original place. 5. explosion 的爆炸動畫要如何作呢？ Create Event -> 然後如下圖設定
做點苦工，把週圍可能被波及之處都設定 destroy



Note that this goes wrong if the person is next to the bomb! So make sure the triggers are not next to the bombs. It is important to carefully design the levels with the bombs and triggers, such that they present interesting challenges.

這裡要留意： 不要把趨動爆炸事件的trigger 放在bombs 旁邊，
否則，小熊(person) 若靠近時，也會一起被炸掉。

深呼吸… 我們再加點什麼讓遊戲變複雜…

製造一些方塊 (blocks) 讓玩家可以推；製造一下洞 (holes) ，讓它不能直接越過。
可是，玩家可以推動(push)方塊建立新的通道，也可以使用方塊把 monster 抓起來。

Blocks and holes

Let us create something else that will enable us to make more complicated puzzles. We create blocks that can be pushed by the player. Also we make holes that the player cannot cross but that can be filled with the blocks to create new passages. This allows for many possibilities. Blocks have to be pushed in a particular way to create passages. And you can catch monsters using the blocks.

下面這一段教你設定 block物件：

The block is a solid object. This main problem is that it has to follow the movement of the person when it is pushed. When it collides with the person we take the following actions: We test whether relative position `8*other.hspeed`, `8*other.vspeed` is empty. This is the position the block would be pushed to. If it is empty we move the block there. We do the same when there is a hole object at that position. To avoid monsters running over blocks we make the corner wall the parent of the block. This does though introduce a slight problem. Because a collision event is defined between the person and the corner wall and not between the person and the block, that event is executed, stopping the person. This is not what we want. To solve this we put a dummy action (just a comment) in the collision event of the person with the block. Now this event is executed instead, which does not stop the person. (To be precise, the new collision event overrides the collision event of the parent. As indicated before, you can use this to give child objects slightly different behavior than their parents.

這裡要設定 hole 物件：

The hole is a solid object. When it collides with the block it destroys itself and the block. We also make the corner wall its parent to let it behave like a wall.

With the blocks and holes you can create many intriguing rooms. There is though a little problem. You can easily lock yourself up such that the room can no longer be solved. So we need to give the player the possibility to restart the level, at the cost of one life. To this end we use the key R for restart. In this keyboard event for the controller we simply subtract one from the lives and restart the room.

This finishes our third version of the maze game. It now has all the ingredients to make a lot of interesting levels.

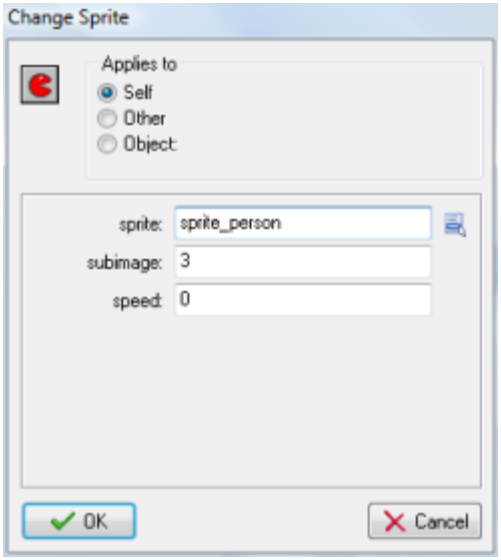
Some Final Improvements

Let us now finalize our game. We definitely should improve the graphics. Also we need a lot more interesting levels. To do this we add some bonuses and add a few more features. The final game can be found in the file `maze_4.gmk`.

Better graphics

The graphics of our current game is rather poor. So let us do some work to improve it. The major thing we want to change is to make the person look in the direction he is going. The easiest way to achieve this is to use a new image that consists of 4 subimages, one for each direction, as follows:

Normally *Game Maker* cycles through these subimages. We can avoid this by setting the variable `image_speed` to 0. When we change the direction of the character, we can change the subimage shown by the action to change the sprite:



A similar thing we can do for all the monsters but here there are no explicit events where we change the direction. It is easier to add a test in the end step event to see in which direction the monster is moving and adapt the sprite based on that.

Bonuses

Let us add two bonuses: one to give you 100 points and the other to give you an extra life. They are both extremely simple. When they meet the person they play a sound, they destroy themselves, and they either add some points to the score or 1 to the number of lives. That is all.

One way streets

To make the levels more complicated, let us add one-way streets that can only be passed in one direction. To this end we make four objects, each in the form of an arrow, pointing in the directions of motion. When the person is completely on it we should move it in the right direction. We do this in the step event of the person. We check whether the person is aligned to the grid in the right way and whether in meets the particular arrow. If so, we set the motion in the right direction. (We set it to a speed of 8 to make it more interesting.)

Frightened monsters

To be able to create Pacman like levels we give every monster a variable called `afraid`. In the creation event we set it to 0 (false). When the person meets a new ring object we set the variable to true for all monsters and we change the sprite to show that the monster is indeed afraid. Now when the person meets the monster we first check whether it is afraid or not. If it is afraid the monster is moved to its initial position. Otherwise, the player looses a life. See the game for details.

Now let's make a game out of it

We now have created a lot of object, but we still don't have a real game. A very important issue in games is the design of the levels. They should go from easy to hard. In the beginning only a few objects should be used. Later on more objects should appear. Make sure to keep some surprises that only pop up in level 50 or so. Clearly the levels should be adapted to the intended players. For children you definitely need other puzzles than for adults.

Also a game needs documentation. In *Game Maker* you can easily add documentation using the **Game Information**. Finally, players won't play the game in one go. So you need to add a mechanism to load and save games. Fortunately, this is very easy. *Game Maker* has a built-in load and save mechanism. F5 saves the current game, while F6 loads the last saved game. You should though put this in the documentation.

You find a more complete game, including all this in the file `maze_4.gmk`. Please load it, play it, check it out, and change it as much as you like. In particular, you should add many more levels (there are only 20 at the moment). Also you can add some other objects, like e.g. keys that open certain doors, transporters that move you from one place in the maze to another, bullets that the person can shoot to kill monsters, doors that open and close from time to time, ice on which the person keeps moving in the same directions, shooting traps, etc.

Finally

I hope this tutorial helped you in creating your own games in *Game Maker*. Remember to first plan your game and then create it step by step (or better, object by object). There are always many different ways in which things can be achieved. So if something does not work, try something else. Good luck!

Further Reading

For further reading on creating games using *Game Maker* you are recommended to buy our book:

> Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice: Game Development for Beginners*, Apress, 2006, ISBN 1-59059-615-3.

The book gives a step-by-step introduction into many aspects of *Game Maker* and in the process you create nine beautiful games that are also fun to play.