

# Game Maker Tips

version 1.4 (June 1, 2000)

by

**Mark Overmars**

The possibilities of **Game Maker** are much larger than you might think. In this documents I will give some ideas that might help you in making your games nicer. Examples using some of these ideas are provided.

## General

### Controller

It is often useful to add a controller object to your game. This object can control certain general things, like playing background music, initializing certain variables, checking whether the game is over, creating random monsters, etc. The first thing such an object should do in its creation event is to move itself to a position outside the visible room, e.g. to position (-100,-100). I always give the controller a computer image. (Don't forget to place one in each room.)

### Gravity

Assume you want to add gravity to your game. So objects should fall down. The speed with which they fall should increase every step. First give the objects a downward direction but, at the start, set the vertical speed to 0. Now we make a step event that increases the vertical speed with, say, 0.25 in each step. When the object hits the floor it should either set the vertical speed to 0 or bounce, by setting the vertical speed to the value  $-vspeed$  (don't change the vertical direction; this will cause problems cause then you suddenly have to decrease the vertical speed rather than increase it). Now whenever there is no floor below an object, it will start falling down with increasing speed. (See the Falling Balls demo for an example.)

### Continues keyboard events

Keyboard events are generated when a key is pressed (and when it is repeated by the system). It is not a continuous event as long as the key is being pressed. In some games you want this continuous behavior, E.g. you want an object to move as long as the key is pressed. There is a very simple way to achieve this behavior. In the step event of some existing object (e.g. your controller object if you have one) place a code action with only the line

```
lastkeypressed = keypressed;
```

keypressed is the currently pressed key. lastkeypressed was the key pressed last, which is used to generate keyboard events and which is then reset to 0. By the above line of code it is set back to the currently pressed key such that a new keyboard event will be generated.

### Joystick support

Though this might not be obvious, the program actually has joystick support. Movements of the joystick create keyboard events <NUMPAD>1 to <NUMPAD>9 as in the numeric keypad. The four buttons generate keyboard events for the letters A, B, C and D. Let some object in your game react to these events and you are done. You can actually use the joystick in the Pacman game.

## Your own score

Scores are normally displayed in the window caption. If you want to make to layout of your game a bit more fancy, you might prefer to put the score somewhere in the playing area itself. This can be done as follows. First of all, you should not use the score system of Game Maker. Instead, use your own global variable `global.myscore`. Add and subtract score using this variable. Now make an object that is going to display the score. Give it an icon to be able to place it when designing rooms, but this is not important cause we are not going to draw it. Instead, use the advanced drawing feature for the object and as code use something like:

```
{
    draw_text(x,y,'score: ' + string(global.myscore));
}
```

That is all. You might want to choose a more fancy font, size, style and color for it. See the language guide for the functions for this.

## Creating rooms

### Objects on the foreground

In a number of cases, you might want objects to move over other objects. Objects are drawn in the order in which they are added to the room. So make sure that you add the objects that should go on top last. But what if you need two objects on top of each other at the start? There is a very simple trick to solve this problem. Assume for example that you need at the start a ball on a hole. Create the ball and the hole object. Now create an extra object that we call `ball_on_hole`. On creation, this object creates a ball and changes itself into a hole. That's all. The `ball_on_hole` object no longer exists, you have a ball and a hole, and, because the ball was created last, it lies on top of the hole (and on top of everything else).

### More precise positioning

If you want to position your objects more precisely in the room, set the cell size to half the actual object size. Now you can place objects at many more places. (You can also make objects partially overlap this way.)

### Scrolling background

Scrolling backgrounds are a nice way of adding a feeling of motion to your games. You can use them to e.g. have a plane fly forward while avoiding objects (and shooting other planes), to have a car drive over a road, etc. Make sure some other objects move with the same speed as the background, to increase the illusion. For smooth scrolling, make sure the background picture can nicely be tiled. You can also use a big background picture, e.g. a long road.

You can set the moving speed when creating the room, but you can also control the speed during the game using pieces of code. This makes it possible to e.g. leave an object in the middle of the room but have the world move when you press an arrow key.

## Images and sounds

### Background music

To create background music for your game, find a nice (and long) midi file. Now add this file to your list of sounds. Create a special object called music that, in its creation event, plays the sound and destroys itself. Place it in the first room, and you are done. The midi file will keep repeating itself until you stop the game.

### Finding images and sounds

If you cannot find the images, backgrounds, or sounds you like in the provided collections, the best way is to search the web. On the web huge collections of free images, sounds, backgrounds, animations, etc, are available. To get you started, here are some of the sites that store such material:

- <http://www.coolarchive.com/index.cfm>
- <http://www.arcadia-animations.com/frameindex.htm>
- [http://www.developer.com/downloads/d\\_images.html](http://www.developer.com/downloads/d_images.html)
- <http://www.clipart.com/>
- <http://www.kidsdomain.com/icon/index.html>

These sites again have many further links.

## Object with multiple appearances

In some games you have objects that should have a different appearance (image) in certain situations, but should have the same behavior. For example, in Pacman, the image looks different depending on the direction of motion. To this end you can create multiple copies of the same object with a different image and for the rest the same behavior, but this is a lot of work. Using advanced drawing you can solve this problem in a much easier way. You create separate objects with the different images but without any behavior. There is just one active object. In the advanced drawing code you draw the correct image, depending on the situation. For example, for the Pacman game the advanced drawing code could look as follows:

```
{
    if (hdir < 0) draw_image(x,y,pacleft)
    else if (hdir > 0) draw_image(x,y,pacright)
    else if (vdir < 0) draw_image(x,y,pacup)
    else if (vdir > 0) draw_image(x,y,pacdown)
    else draw_image(x,y,self);
}
```

If the image is not animated, you can also use sub images for this and choose the right sub image.

## Clever Code

### Lives

Many games have multiple lives. To use this in your games you need to write a little bit of code. First of all, you need a global variable called e.g. lives. The controller object should set this in the first room of the game using in its creation event a piece of code like:

```
{
    if (room==1) global.lives = 3;
}
```

To show the number of lives to the player, you need an object, e.g. called life, whose image shows a life. You can draw them at the bottom of the room using e.g. the code:

```
{
    destroy(life);
    i = 1;
    while ( i <= global.lives)
    {
        create(32+i*32,roomheight-32,life);
        i += 1;
    }
}
```

You can also have one life object and use advanced draw to draw it a number of times depending on the global variable lives.

Now when the person dies you check whether global.lives is 0 (or whether the number of life objects is 0). If so, the game ends. Otherwise, decrease global.lives and draw the life objects as above.

## Defining functions

The built-in programming language does not allow you to define functions. But there is a trick to do it. Make an active, non-solid object called e.g. function\_XXX. In its creation event put the piece of code that you want to use as a function. Also, put in it an action to kill itself. Now to call the function in another piece of code, use

```
create(-100,-100,function_XXX);
```

That will do the trick. You can pass parameters through global variables. (You can even create a recursive function this way but make sure the recursion depth is small.)

## Finding key codes

To use the variables lastkeypressed and keypressed you have to know the keycodes for the different keys on the keyboard. This can be found out by running the program, starting the debug mode (<Ctrl>-D) and clicking on global variables. Whenever you press a key the value of the two variables is shown.

## Saving a game

You can use the (crude) file IO to save a game for later use. To this end, create a controller object in each room that executes the following piece of code in its creation event:

```
{
  if (room==1 && read(1)>1)
    goto_room(read(1))
  else
    write(1,room);
}
```

You might want to precede this with a question to the user whether he or she want to continue a saved game. The program can use write(1,0) when the person dies, to avoid continuing in that case.

## Internals

### Image colors and transparency

As described in the documentation, object images are stored as gif images. This means that they will be reduced to 256 colors. If you load a bitmap image with more colors, this means that colors can change. Images are always considered to be partially transparent. For the transparency color the left bottom pixel of the image is used. (If you want an image that fully covers the area (e.g. a brick) simply make it one pixel higher and add there a line with a different color. Because the bounding box will be based on the non-transparent area this works fine.)

**Game Maker** must choose a color that it uses for the background when you it saves the image. For this a dark green color is used (very ugly). This might cause a problem when the image itself uses an almost similar dark green color. In this case holes might appear in the image or, worse, in some cases, the background stops being transparent. The solution here is to choose a slightly different color in your image.

## Order of events

In some cases it is important to know the order in which the events in the game are being processed. This order is as follows:

1. Handle the alarm events.
2. Handle the keyboard and mouse events.
3. Handle the step events.
4. Set each active object in its new position, based on speed and direction of motion.
5. Handle the collision events until no collisions occur anymore (when a collision occurs, the object is put back into its previous position to let the event change the motion after which it is moved again). If there is still a collision after 16 tries the program gives up and leaves the object at its previous position.
6. Handle the meeting events.