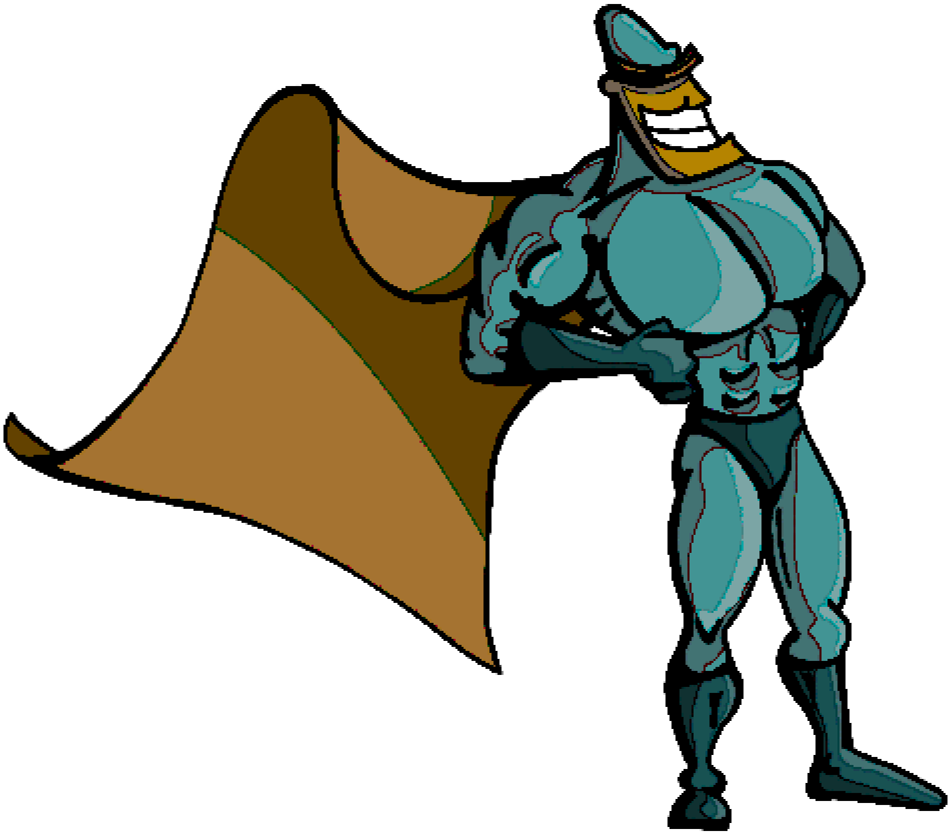
Game Maker



The Tutorials

This booklet has been prepared by Richard Lancaster for students enrolled with eCentral Campus of Central Institute of Technology in Perth, Western Australia in Semester 1, 2010 in the Certificate III in Media. The booklet is designed to be used in the unit *Author Interactive Sequence*.

Thanks to Adil Mistry for testing the games. Any errors remaining are Richard's fault. Comments and corrections to:

richard.lancaster@central.wa.edu.au

The purpose of this booklet is to introduce students to 2D game design and development using the Game Maker program created by Mark Overmars. Some of the tutorials are based on material originally devised by Professor Overmars and grateful acknowledgement is made to both his software and his teaching material.

Screen captures are from versions 7 and 8 of Game Maker and reproduced with permission of the copyright owner, YoYo Games

Cover art is SUPER Hero by Ryan Goggin.

Disclaimer: Every effort has been made to ensure this booklet is free from error or omissions. However, you should conduct your own enquiries and seek professional advice before relying on any fact, statement or matter contained in this book. Central Institute of Technology is not responsible for any injury, loss or damage as a result of material included or omitted from this booklet.

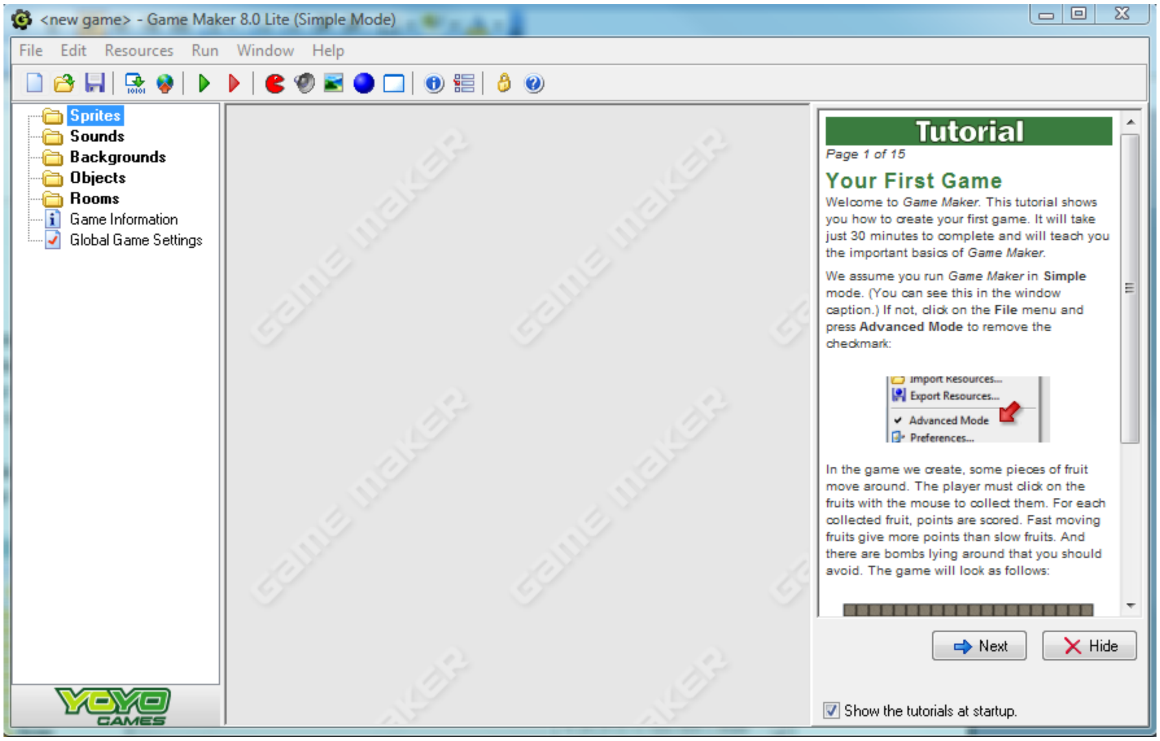
Contents

Character sketch in pencil by Todd Millias

Getting You Ready

Game Maker is an easy to use program for creating 2D computer games. It runs on the Windows and Macintosh operating systems and comes in two versions, Lite and Pro. The Lite version is free and can be downloaded from www.yoyogames.com. All the games in this book can be made using the Lite version.

This is what the screen looks like when you open the Lite version of Game Maker 8 in Simple mode for the first time (Advanced mode gives you some extra controls on the left). If you have the internet connected, you will see a News screen if it's the first time the application has been opened on that day.



Tutorial

Page 1 of 15

Your First Game

Welcome to *Game Maker*. This tutorial shows you how to create your first game. It will take just 30 minutes to complete and will teach you the important basics of *Game Maker*.

We assume you run *Game Maker* in **Simple** mode. (You can see this in the window caption.) If not, click on the **File** menu and press **Advanced Mode** to remove the checkmark:

In the game we create, some pieces of fruit move around. The player must click on the fruits with the mouse to collect them. For each collected fruit, points are scored. Fast moving fruits give more points than slow fruits. And there are bombs lying around that you should avoid. The game will look as follows:

☑ Show the tutorials at startup.

Down the left side you see the folders of resources and the controls you will use to create your games.

Sprites: Are small images. They can be animated gifs which contain sub-images, so a character can appear to walk or the propellers spin on an airplane.

Sounds: You can play background music during a game and you can play sound effects, such as the noise of an explosion.

Backgrounds: Are what you see in the background! They can be one big image or made up of a number of smaller ones.

Objects: These are normally what you see moving around. They usually take on the appearance of a sprite and can be given other characteristics such as how they move or react to other objects. For example, two objects can collide and the player will see an explosion with one of the objects disappearing.

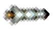
If there is more than one of an object visible, you are probably not looking at different but identical objects, but at *instances* of an object. Say you want to make a squadron of planes appear in your game. Each plane looks the same but has an existence of its own, so each can move differently and one can be shot down while the others remain. You would create one sprite first, then one object which has the appearance of that sprite. In your game you can add as many *instances* of that object as you want. All the instances will look and behave the same but you could shoot one down without affecting the others.

Rooms: Are where the games take place. You give a room a background and place your objects inside. If you are making a game with more than one level, then each level is a separate room.

All of these make up the *resources* you need to make the game.

Game Information: Tells a player about the game and how to play it. It appears when the player presses the F1 key.

Global Game Settings: Let you control features such as the speed at which the game plays.



There are two more things you need to know before you make your first game. After you make your objects you will want something to happen to them. You want your plane to fly, to shoot missiles and the target to explode when hit! These are called *events*. When an event happens you can attach *actions* to it. For example, if a target is hit by a missile, a *collision event* occurs The actions that might be attached to the collision event could be the instance of the target is destroyed, an image of an explosion shown, a sound played and the score increased.

Let’s make our first game!

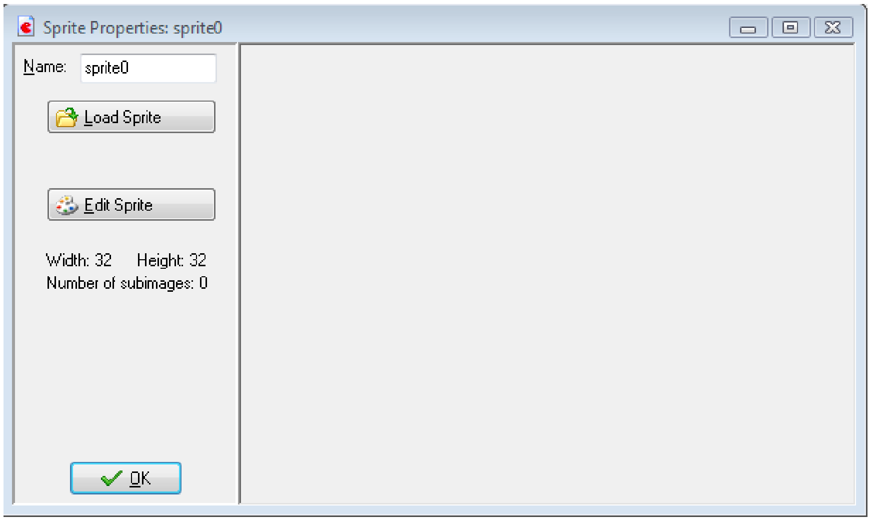
Your First Game

The first step is always to write a short description of a new game:

> *A balloon floats around a room, bouncing off the walls. Clicking on it will pop the balloon. Each popped balloon is worth ten points.*

Start Game Maker. These instructions and screen captures assume you are in Simple mode but will work exactly the same in Advanced mode.

This game requires two objects, a balloon and a brick to make walls. First we need to load two images (sprites) into the resources of our game. Click on the icon to Create a New Sprite You will see a Properties window like this:



Name: sprite0

Width: 32 Height: 32
Number of subimages: 0

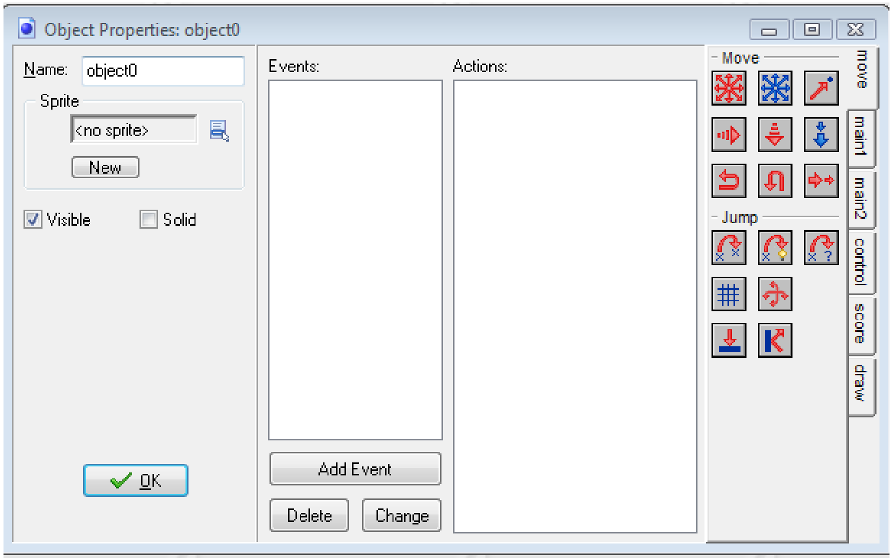
In the name box type spr*_red_balloon*. It is a good habit to always give names which have no spaces in them and are descriptive. You do not have to start a name with a three letter prefix indicating which type of object it is (spr for sprite, snd for sound, etc) but it is a common convention. Spaces are not allowed in names when using code (which we will do in later sessions.) Click on the Load Sprite button and select the red_balloon.gif file in the Game 1 folder on the network drive. Be careful that Game Maker does not mislead you about location, by default it looks in its own sub-folders first and Game Maker comes with some resources. You will see that the size of the image is 32 pixels square and since there is only one subimage it is not an animation (which would be a looping sequence of subimages). Close the Properties window by clicking on its Close button and choosing to save. Or click on the OK button, if you can see it, which saves and closes.

Add the second sprite you need. Click on the icon to Create a New Sprite. Name it *spr_brick* and load brick.gif from the same Game 1 folder. Close and save the Properties window.

Time to save our game. Click on the menu item File⇨Save and save the game to your network account if possible, or a removable drive, as a last resort use the Desktop. Save as *my first game*. The extension (the bit after the full stop in a filename) will be gmk. You can see the filename in the caption of the program window. A gmk file can only be opened with the Game Maker program.

Try running the game to see if there is anything visible yet, to run a game click on the green triangle in the icon bar. The game will be saved and an executable version displayed. However, you do not yet have a game as you need a room for a game to exist.

You will not see a sprite in a game unless it has been assigned to an object so let's do that now. Click on the Create an object icon ●. Enter the name obj_red_balloon and click on the blue icon next to the Sprite name and select spr_red_balloon. The Properties window should look like this:



Object Properties: object0

Name: object0

Sprite

<no sprite>

☑ Visible ☐ Solid

Click on the OK button to close the window (you may have to resize the window to see the OK button at the bottom). It is dangerous to have a sprite and an object with exactly the same names so we have made them different in order that Game Maker will not get confused.

Create another new object, name it obj_brick and assign the brick sprite to it. Close the Properties window and save the information.

We know that when a balloon is popped we want a sound effect, so let's add that to the resources of the game. Click on the icon for Create a sound 🔊. Name the sound "snd_pop" and choose "pop.wav" from the Game 1 folder. You can play the sound to hear what it is like (it will also make sure the speakers are working). Click on OK to close the window.

Time to add a background into our resources ready to apply it to a room in the next step. Click on the Create a Background icon 🖼. Change the name to "back1". Load the file "back1.gif". You will see it is larger than the sprites but will still not fill a room of any decent size. Game Maker will **tile** the image for us (that is, repeat it as many times as is needed to cover the area). Close the Properties window with the OK button or the red Close button and save.

Now for the room in which the game will take place. Click on the Create a room icon . Then select “backgrounds” from the three tabs. Turn off the checkbox for Draw a background color and click on the blue icon to the right of “no background”, assign “back1” as the image. Click on the settings tab and give the room a caption “Destroy the Evil Balloons”. Click on the icon with a green tick to save and close the room.

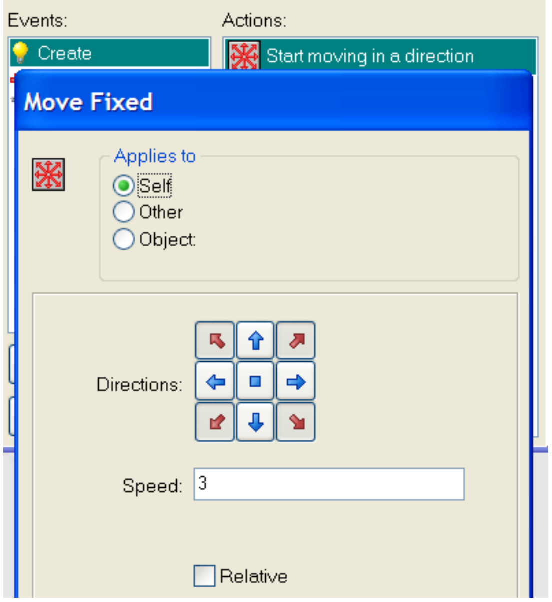
Click on the objects tab. You should make sure both your Room Properties window and the program window are maximised at this point or you will find you are not seeing the full room and you will put a wall where a wall shouldn’t be. You will only be able to see one object at the time on the left but if you left click on it you can choose from all the objects. Choose the brick object first. If you hold down the Shift key as you drag the mouse around the outside edge of the room you will be able to lay a line of bricks. If you make a mistake you use the right mouse button to delete an instance of an object. When the wall is done, change to the red_balloon object and put one of those in the room, anywhere you like. Where you place it will become its starting position.

Close the Room Properties window (it is sometimes called a Properties form), saving it if prompted. The green icon to the left with a tick on it ✓ is the quickest way. It may say that you have placed instances outside the room, and should it destroy them, if it does, choose Yes. If your hand wandered a little as you placed instances, you could have put some outside the room, where you cannot see them, but Game Maker can.

Let’s play the game, we know that nothing will happen but we can at least see what we have made so far. Click on the floppy disk icon first to save your work and then click on the green Run the game icon ▶. If you cannot see the bricks it is either because spr_brick is set to transparent or there is tick on the backgrounds tab of the room against foreground image.

Making Things Happen with Events and Actions

Looking good? Now to make things happen. Close the game and return to Game Maker. Double left click on the icon for the red_balloon object to open the Properties form. Click on the Add Event button and from the Event Selector choose Create. Since this event will happen when a balloon first appears, we can **assign an action to the event**, we will start the balloon moving. From the Move menu on the right, click on Move Fixed and drag it into the Actions column. A dialog box will appear. The eight blue arrows will control in which direction the balloon will move. Click on the four corner arrows and they will go red. Do not click on the blue rectangle in the middle.



Applies to
● Self
○ Other
○ Object

Speed: 3

☐ Relative

This means the computer will start the balloon moving in a randomly chosen direction from any of the four. Set the speed to 3. Click on OK to close the dialog box.

Let's run the game to see if our first event and action are working (the balloon will sail out of the room).

Now to set how the balloon behaves when it hits a wall. On the Object Properties window choose Add Event and Collision, when it asks with what, select the wall object. To tell it how to act, from the Move tab on the right, click on Bounce and drag it into the Actions column. You do not need to adjust any settings, just OK.

Run the game.

The balloon still floats through the bricks! Your first instinct is to look at the Collision event and action again, but there is nothing wrong with them. The problem is earlier. We have not changed the default setting on either balloon or brick to make them solid. If an object is not made solid it behaves like a ghost and cannot collide with other objects.

Open the property window of obj_red_balloon and put a tick in the checkbox for solid. Run the game.

Now the balloon sticks to a brick. Let's try making obj_brick solid and running the game.

Now it works as we expect. So we know it's important to make objects solid if we want a collision and it makes a difference which is solid.

The whole point of the game play is for the user to destroy balloons so add a new event to obj_red_balloon's properties, the Mouse event, from the drop down menu that appears choose Left button. On the Main 1 tab on the right you will see the Play Sound icon. Drag that into the Actions column. In the dialog box choose the pop sound and leave it as Loop False since you only want one pop. We will add three more actions to this event, so four things will happen at seemingly the same time when the player clicks on a balloon.

The player needs to be rewarded. Game Maker has a built in mechanism for recording and displaying scores. From the Score tab drag the Set Score icon into the Actions column for the left button event. Set the New Score to 1 and put a tick in the Relative box. This means one point will be added to the existing score, which will be displayed in the title bar of the game next to the caption when it is running. If you did not tick Relative the score would always stay the same as 1 would be replaced by 1. Press OK.

If there was only one balloon the game would not last very long and we do not want each balloon appearing in the same place or the player will soon learn where to lie in wait. On the Move tab you will see the Jump to Random icon. Drag it into the Actions column. You do not need to change any of the settings. Press OK. Finally, we need to start it moving again. Drag another Move Fixed icon into the Actions column and again set it off in any direction randomly at speed six.

If you hover your cursor over any icon the Tool tip will tell you what it does. Hover over an Action and you see a brief description of what will happen.

That’s it. Cross your fingers, click on the floppy disk icon to save the game and then run it.

If it does not do what you think, re-read these instructions and double check everything. If it still does not work, call the lecturer over and ask for help.

Every game needs instructions for the user. Double click on Game Information . Enter your instructions (be sure to give your name as the game’s creator). When the game is playing this information is displayed by pressing the F1 key (if you can’t see it, be sure the F lock on the keyboard is turned on). Esc key closes the information window.

While a game is playing, if you press the F4 key the display goes to full-screen. You will not see the title bar but the Escape information key will end the game or F4 returns to normal size.

Challenges

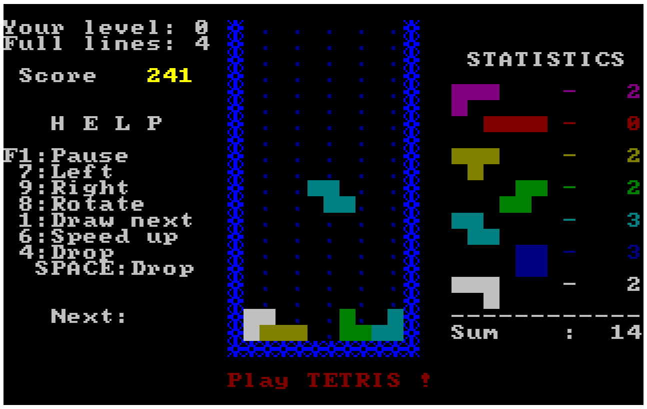
When it is working, if there is still time, why not improve the game play? At present the balloon moves very slowly, this is fine while testing but if you want to speed it up, increase the speed in the Create/Move fixed action of the balloon.

At present a player can hold down the left mouse button and sweep it around, which makes the game easier. We can change this by right clicking on the mouse event in the obj_red_balloon properties window, choosing Change Event and altering the event to “Left pressed”.

You could add a balloon of a different colour, there is a yellow balloon image in the Game 1 folder. You would add the sprite and the object, and could put several instances of them in the room (perhaps the actions make it move slower but be worth a lower score than the red). If you put several instances of the yellow balloon object in the room they will serve to distract the player.

Congratulations: you are a maker of games!

(P.S. Don’t forget if you used the Desktop to move your file to more permanent storage)



```
Your level: 0                     
Full lines: 4                STATISTICS
 Score     241                    -   2
                                  -   0
   H E L P                        -   2
                                  -   2
F1:Pause                          -   3
 7:Left                           -   3
 9:Right                          -   2
 8:Rotate                  ------------
 1:Draw next               Sum    :  14
 6:Speed up
 4:Drop
  SPACE:Drop

   Next:

              Play TETRIS !
```

The first MS_DOS version of Tetris,

developed in Russia in 1985,

one of the most famous computer games.

Recreating Pong

Pong was the game which first made video gaming popular when it was released in 1975. It was the first video game that most people had ever played. It was the first game produced by Atari, in the days before personal computers, and is still played today over thirty years later, for example there is a PlayStation version. The Classic Gaming website has this to say:

> *Pong, while not the first videogame, was the first coin-operated arcade game and the first mainstream videogame that was available to almost everyone. Pong was the impetus for the development of the videogaming industry, almost single-handedly creating both the home and the arcade videogame markets.*
>
> http://www.classicgaming.com/museum/pong/

We will recreate this classic in this session and three more classics in latter sessions. Our learning curve in how to create game mechanics using the Game Maker software will follow the arc of videogame evolution.

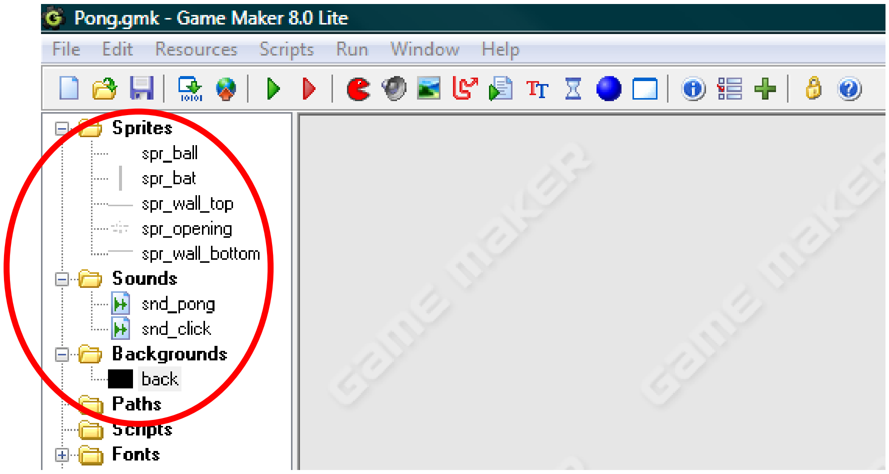
The first step is always to write a short description of a new game:

Pong is table tennis (ping pong) on the screen. It can be played by one or two players. Each controls one bat and tries to hit the ball. If the ball misses the bat, the other side wins a point.

In each session you will start by playing the game from the network folder so you have a clear mental picture of what you are making. Do that now and then read Appendix 1 pages 91 and 92 as far as the section on Depth. To make Pong we need to understand the X,Y coordinate system Game Maker uses to locate objects.

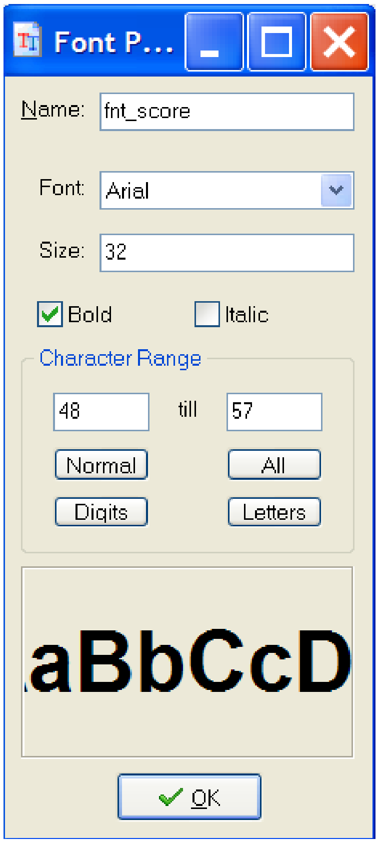
Start Game Maker. These instructions and screen captures assume you are in Advanced mode. You toggle between the two modes through the File menu. Advanced mode gives us some extra elements we need, such as the ability to include a font in the resources.

The graphics and sounds needed have already been created so you can load them into your resources tree. Add these eight resources from the Session 2 folder on the network drive:



Double click on each to examine it. Our game is in black and white like the original. Spr_opening is a graphical set of instructions which a player will see when starting the game. We will make the room background black and the white lettering will show clearly. The sprites wall top and bottom will be used to put a white line, in effect a wall, along top and bottom of the screen so the ball will bounce back into play. The two sprites are very similar, in fact one was flipped to make the other. In a gif any one colour can be transparent. In our first game this allowed us to see only the balloon when it floated over the blue background, we did not see that it was really a rectangular image with a coloured balloon in a white surround. Game Maker in version 8 has changed how it shows a transparent colour. By default it makes the background colour transparent and shows this with the grey chequered pattern which other applications use. Because this is an old game, from when displays could not show colours other than black and white, these gifs have been created so black is the background colour and hence transparent. The walls, as they have been drawn, do not need to be transparent, but the bat and the ball do. Otherwise you will see the black rectangle when the ball passes over objects like the scores.

We want to be able to draw the scores on the screen so we need a font in our resources. Click on the Create a Font icon and make this font.


Font P...

Name: fnt_score

Font: Arial

Size: 32

☑ Bold ☐ Italic

Character Range

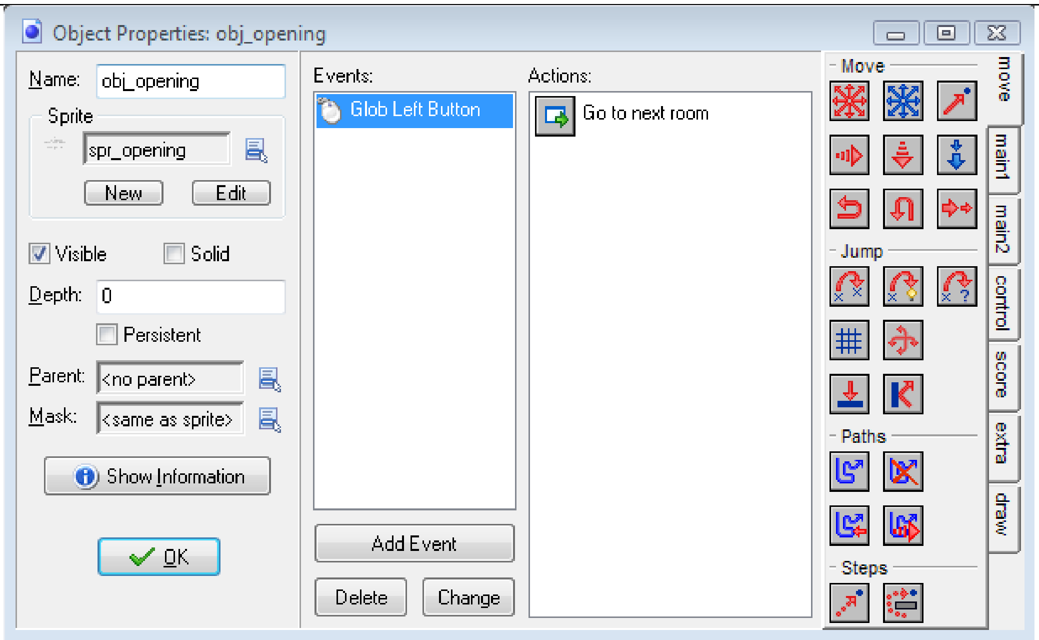
48 till 57

To alter the Character Range you click on the Digits button. Since the scores will be numbers we do not need all the other characters like letters and punctuation. Omitting them will make the font smaller and so take less memory.

The last step in making a game is to save the file as an executable you can distribute to anyone who has a Windows computer. When you do this the font will be saved also, so it will appear in the game whether or not the players have it on their computers. If you give people the Game Maker file, the .gmk, then they will need the Game Maker program and the font on their computers.

Our game will have two “screens”. The first screen will explain the game. When the player presses the left mouse button, the game will begin and the first screen will dissolve into the second. In Game Maker each screen is a room. The first will display the instructions, which have been made as a graphic and in our resources are spr_opening. You will notice the graphic is 640 by 480 pixels, the same size as the default setting of a room.

Create an object named “obj_opening” to display the sprite spr_opening. Add the Mouse event: Mouse/Global mouse/Global left button event and then add the Go to Next Room action from the main1 tab. Choose any transition effect you like.



Object Properties: obj_opening

Name: obj_opening

Sprite: spr_opening

☑ Visible ☐ Solid

Depth: 0

☐ Persistent

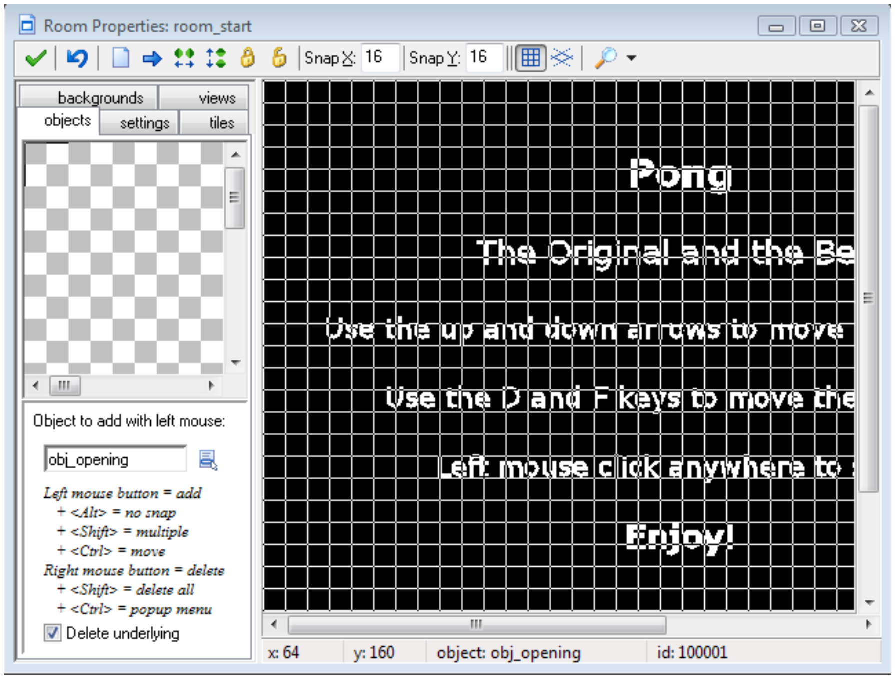
Parent: <no parent>

Mask: <same as sprite>

Events: Glob Left Button

Actions: Go to next room

Now we have an object, we can make the first room. Create a room. Under the objects tab add obj_opening. To do this left click in the top left of the room as the graphic is the same size as the room. If you make a mistake, remove it with a right click. Under the settings tab, set the caption to say Pong and as the Name use “room_start”. Under the backgrounds tab set the background colour to be black, as by default in Game Maker 8 the background of a sprite is transparent. Click the green tick to save and close.



Room Properties: room_start

Snap X: 16 Snap Y: 16

Object to add with left mouse: obj_opening

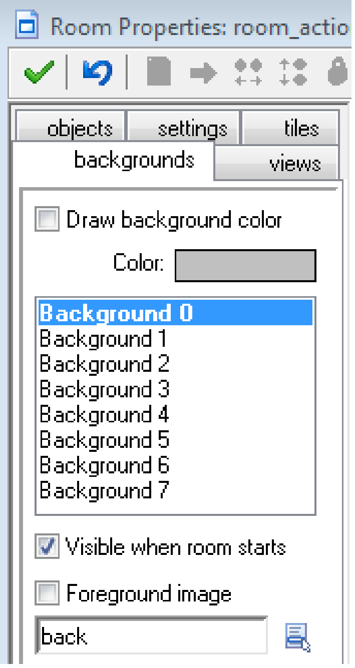
Left mouse button = add
+ <Alt> = no snap
+ <Shift> = multiple
+ <Ctrl> = move
Right mouse button = delete
+ <Shift> = delete all
+ <Ctrl> = popup menu

☑ Delete underlying

x: 64 y: 160 object: obj_opening id: 100001

backgrounds
views
objects
settings
tiles
Pong
The Original and the Be
Use the up and down arrows to move
Use the D and F keys to move the
Left mouse click anywhere to
Enjoy!

We might as well create the other room now, we will be back and forth between the second room and the large number of objects we need to create.



Room Properties: room_actio

□ Draw background color

Color:

Background 0
Background 1
Background 2
Background 3
Background 4
Background 5
Background 6
Background 7

☑ Visible when room starts

□ Foreground image

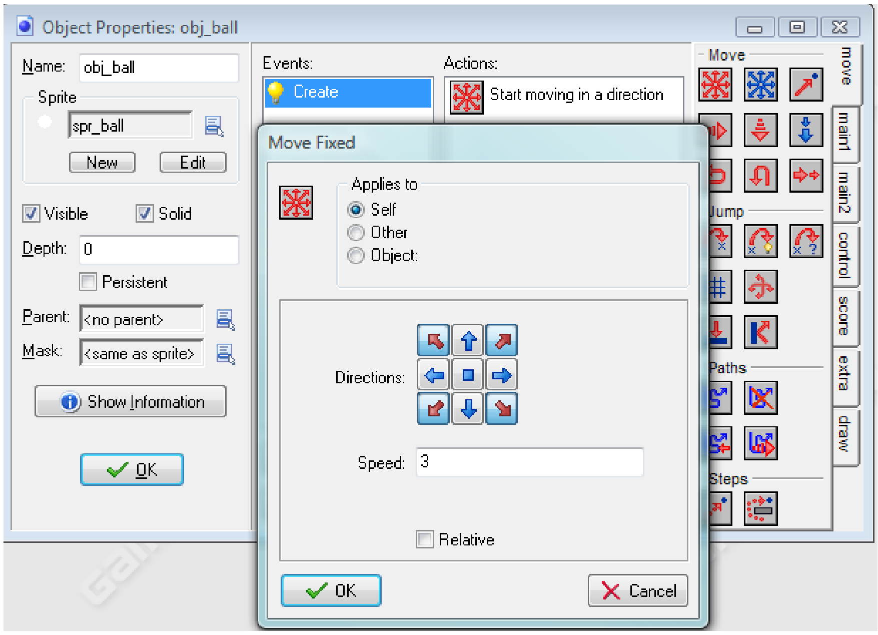
back

Create a new room. On the backgrounds tab click where it says “no background” and add “back”. The background you imported which is named “back” is also a 640 by 480 black and white graphic made in an image editor. Its purpose is to give us the net down the middle.

On the settings tab set the caption for the room to also say Pong (the term “ping-pong” was already copyrighted so the original game could not use it). Change the Name to “room_action” and click on the green tick to save it.

The second room could have been named anything as long as it is underneath room_start in the resources tree on the left. Rooms are shown in their order in the tree, starting with the top one. You can drag and drop to alter the position of rooms so you can create different levels in any order. The vertical order of sprites, objects and sounds in the Resources tree does not matter; it does with rooms.

Time to start working on the objects, the heart of the game. Let’s put the ball in first. Create a new object called obj_ball with the sprite spr_ball, it needs to be solid so it can collide and give it the Create event. As an action make it Move Fixed diagonally, if it moves straight horizontally or vertically it will just bounce back and forth by itself and never come near a bat. Put the speed at 3, this will be slow while we test the game, we can speed it up later.



Object Properties: obj_ball

Name: obj_ball

Sprite: spr_ball

☑ Visible ☑ Solid

Depth: 0

□ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events: Create

Actions: Start moving in a direction

Move Fixed

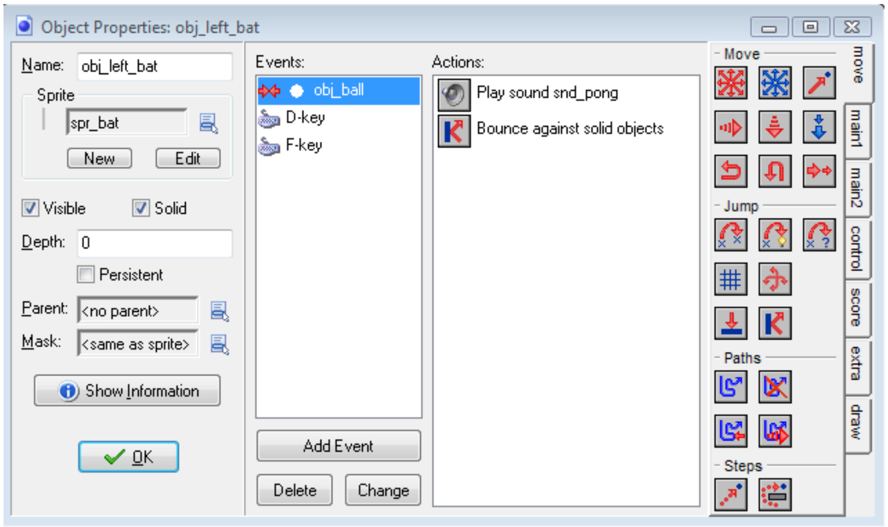
Applies to: ● Self ○ Other ○ Object:

Directions:

Speed: 3

□ Relative

Now to make the bats. Each bat will have three events, the ball can hit it and the player can move it up and down. The left bat moves vertically with the D and F keys; the right bat moves with the up and down arrow keys. Start by creating an object called obj_left_bat which shows spr_bat, is solid and looks like this:



Object Properties: obj_left_bat

Name: obj_left_bat

Sprite: spr_bat

☑ Visible ☑ Solid

Depth: 0

□ Persistent

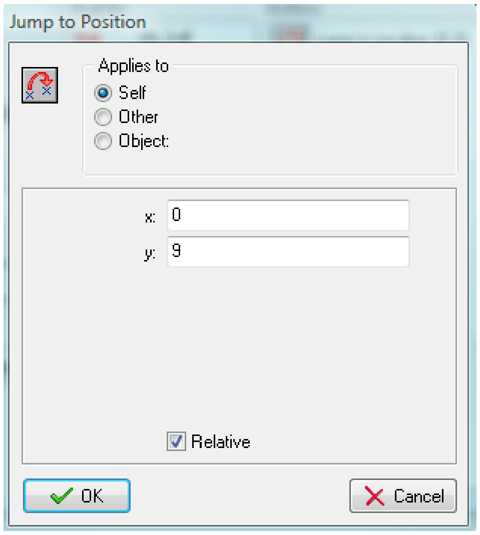
Parent: <no parent>

Mask: <same as sprite>

Events:
- obj_ball
- D-key
- F-key

Actions:
- Play sound snd_pong
- Bounce against solid objects

The three events start with a collision with the ball, where you must set the action so **the other** object bounces (that is the ball bounces when it is colliding with the bat). Add a collision event with obj_ball. Then add the action from the main1 tab to Play a Sound, use snd_pong, no looping. The second action is on the move tab, Bounce. Apply the Bounce to Other, as you want the ball to bounce not the bat.
The other two events are under Add Event ⇨ KeyBoard ⇨Letters. The action that occurs with each is the Jump to Position. This graphic shows the D-key:



Jump to Position

Applies to
● Self
○ Other
○ Object:

x: 0
y: 9

☑ Relative

We want the bat to move slightly every time the key is pressed, and to repeat if the key is held down. So we use the KeyBoard event not the Key Press(which does not repeat).
The y axis is the vertical. This is the D key so we want the bat to move down 9 pixels (remember zero y is the top of the screen, see Appendix 1).

The x position, of course, is unchanged at zero as we do not want the bat to move sideways.
Make the jump relative, so its nine pixels relative to its current position.

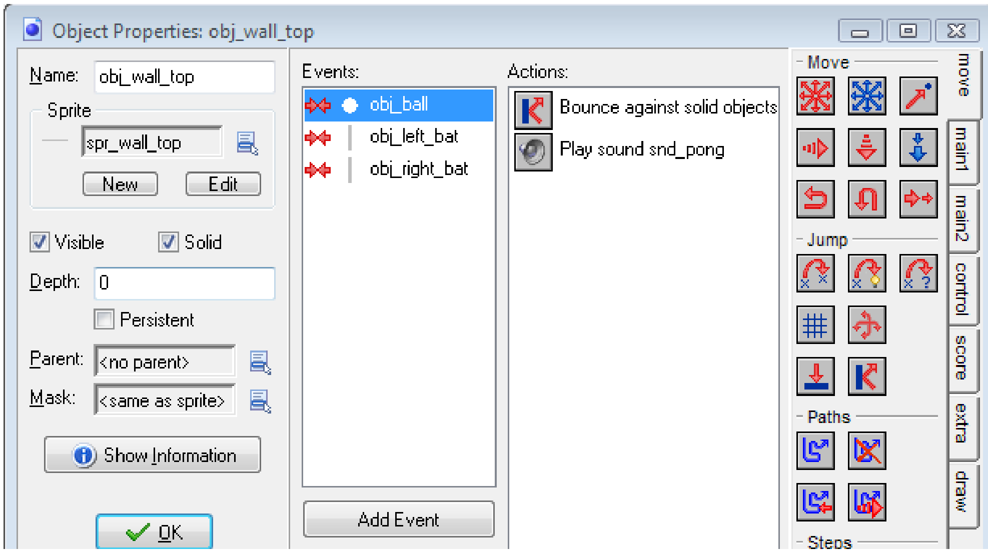
With the F key we want the bat to move up so the y will be -9. X remains at zero and tick relative.

Create a obj_right_bat object with similar properties but acting on the up and down arrows.

Let's put our ball and bats in the room so we can test the game. Open up the second room and place a ball object anywhere and the bats in their opening positions. It can be hard to see with the white grid lines. Turn them off with this icon ▦.

Save the game by clicking on the floppy disk icon, call it Pong, and then run the game. You should see the opening screen with its instructions and can left click to start the game play. Watch to see the ball appears (don't worry that it soon disappears) and test that the bats move up and down.

Now it is time to place a wall along the top and bottom of the screen to keep the ball in play. Both walls will be the same. Let's start with the top. Create an object called "obj_ wall_top", assign it spr_wall_top, make it solid, and give it three events. The first is to make the ball bounce off it, so we need a collision event with the ball where "the other" is affected in the bounce action (same as with the bat objects) and then there is the pong sound effect.


Object Properties: obj_wall_top

Name: obj_wall_top

Sprite: spr_wall_top

☑ Visible ☑ Solid

Depth: 0

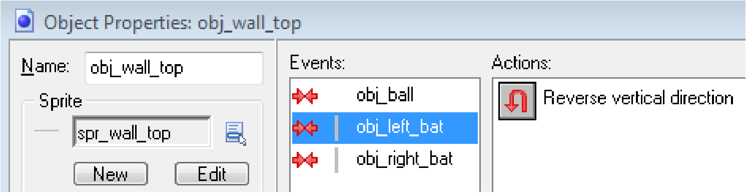
☐ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events:
- obj_ball
- obj_left_bat
- obj_right_bat

Actions:
- Bounce against solid objects
- Play sound snd_pong


Object Properties: obj_wall_top

Name: obj_wall_top

Sprite: spr_wall_top

Events:
- obj_ball
- obj_left_bat
- obj_right_bat

Actions:
- Reverse vertical direction

Then come two more collision events, one each with the two bats. The action in both cases is to reverse the direction of "the other". This will make the bat bounce off the wall, otherwise it would disappear off the screen.

Create an object called obj_wall_bottom and give it the same events and actions as obj_wall_top.

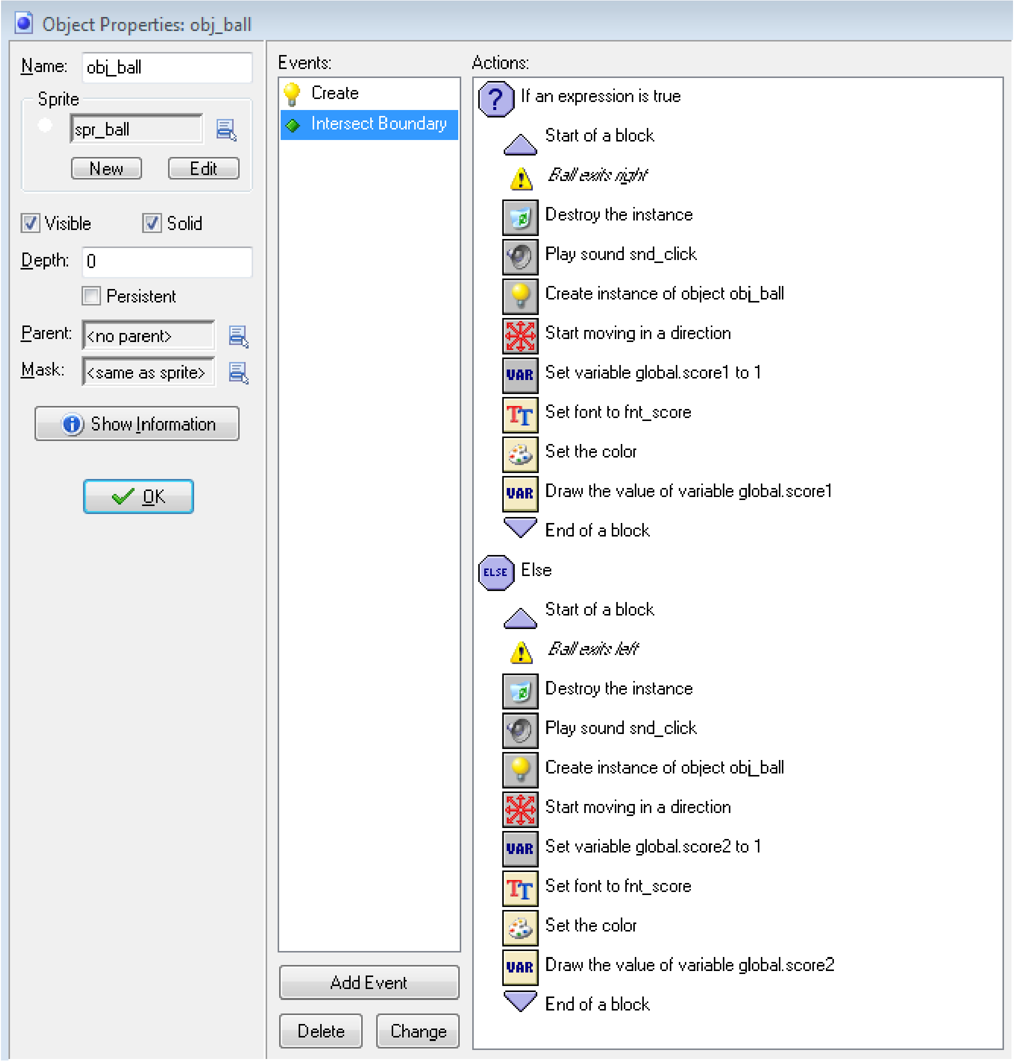
Add the two wall objects to room_action. This is much easier with the room maximized and the grid off. Even then it is impossible to see the white lines against the edges. So go to the backgrounds tab and set it to no background. Place the wall_top along the top edge and wall_bottom along the bottom. Then go to the backgrounds tab and turn the "back" sprite back on. Save and choose to destroy any instances which have strayed outside the room.

Save and run the game to ensure the walls work.

When a Ball Leaves the Room

Now we are getting to the core of the game, the most difficult part of the design. We want to program what happens when the ball leaves either side of the court: the other player needs to win a point and a new ball must appear, a sound clue would be nice.
The event we can use is Intersect Boundary. But how do we know if it's the right or left side? One solution is to check the value of the x co-ordinate. Remember that the default width of the room is 640 pixels and zero is the left edge. So if the ball when it intersects the boundary is past, say 600 pixels, then it must be on the right side; if its less than 600 it must be going the other way. Thus we check for the value of x and then have a branching structure (in computing jargon an If-Else) to increase the score on the left or the right.

Re-open obj_ball, after the Create event add an Other/Intersect Boundary event. In the tables on the next page are the settings you need for all the actions shown below. What you are implementing here is a branching structure, the top half deals with the ball exiting on the right, the lower half with the ball exiting on the left. The computer keeps tracking an object after it has left the screen and this takes processing time. Since we don't want the ball which has bounced out any more, we destroy it and create a new ball positioned just in front of the bat of the player who lost the point. The last four actions in each branch are there to update the value of the score and to display the new value.



Name: obj_ball

Sprite: spr_ball

☑ Visible ☑ Solid

Depth: 0

□ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events:

- Create
- Intersect Boundary

Actions:

```
If an expression is true
    Start of a block
    Ball exits right
    Destroy the instance
    Play sound snd_click
    Create instance of object obj_ball
    Start moving in a direction
    Set variable global.score1 to 1
    Set font to fnt_score
    Set the color
    Draw the value of variable global.score1
    End of a block
Else
    Start of a block
    Ball exits left
    Destroy the instance
    Play sound snd_click
    Create instance of object obj_ball
    Start moving in a direction
    Set variable global.score2 to 1
    Set font to fnt_score
    Set the color
    Draw the value of variable global.score2
    End of a block
```

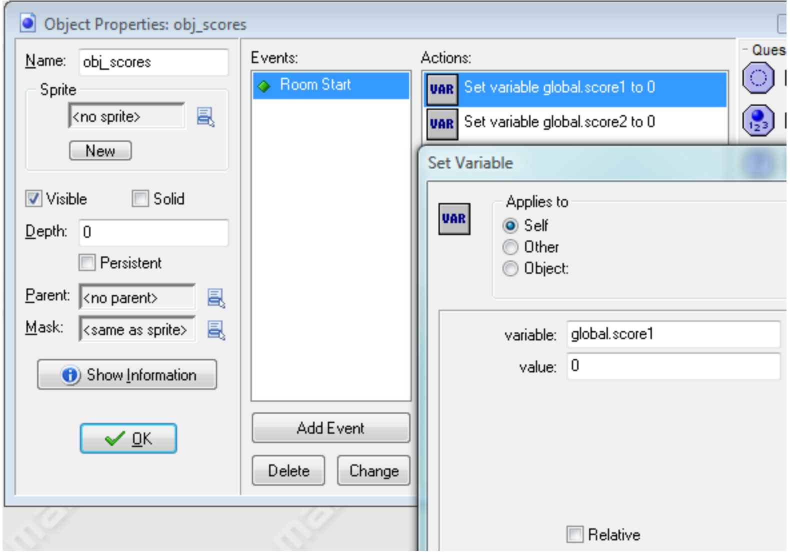
Action	Settings	Comment
If an expression is true	obj_ball.x >600	
Destroy the instance	self	This saves memory as the computer does not keep tracking the ball.
Play sound	snd_click	
Create instance	Object: ball X: obj_right_bat.x-20 Y: obj_right_bat.y	If the ball goes out on the right, a new ball appears in front of the right bat.
Start moving in a direction	Diagonally to the left Speed: 3	Right ball moves to the left.
Set variable	Variable: global.score1 Value:1 Relative √	A variable is a container. A global variable is a container whose contents are available everywhere in a game. We are creating a variable called global.score1 and adding one to its value. Relative means add to any existing score. If you do not do that, the score resets to 1 each time instead of adding 1 to the existing total.
Set font	Fnt_score Align: right	Sets the font we will use to write the score.
Set the color	White	The default is white. The box shows the colour, it doesn’t say the name.
Draw the value of variable	Variable: global.score1 X: 260 Y: 10	The left score is drawn just left of centre and down the screen 10 pixels.

If the ball did not disappear on the right, it must have gone out the left side. We need the same actions but to alter the settings.

Action	Settings	Comment
Create instance	Object: ball X: obj_left_bat.x+20 Y: obj_left_bat.y	If the ball goes out on the left, the new ball appears in front of the left bat
Start moving in a direction	Diagonally to the right Speed: 3	Left ball moves to the right.
Set variable	Variable: global.score2 Value: 1 Relative √	Global.score2 is the variable keeping track of the other player’s score.
Set font	Fnt_score Align: right	Sets the font we will use to write the score.
Set the color	White	
Draw the value of variable	Variable: global.score2 X: 350 Y: 10	The right score is drawn just right of centre and down the screen 10 pixels.

If you run the game now you will get an error message. The problem will be with the variables. When the variables to hold the scores are first created they contain nothing, and you cannot add a number to nothing. We need to give the variables an initial value. This can be zero as zero is a number and therefore you can add another number to it.

We need to start with the scores set to zero and displaying when the game begins. To control this we will create a new object called obj_scores. It needs to be visible yet with no sprite and not solid as nothing need collide with it. The first event is Other/Room Start. The first action is to set both scores to zero, in computing this is called *initializing the values*. Notice that you do **not** tick Relative.



Object Properties: obj_scores

Name: obj_scores

Sprite
<no sprite>

☑ Visible □ Solid

Depth: 0

□ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events: Room Start

Actions:
- Set variable global.score1 to 0
- Set variable global.score2 to 0

Set Variable

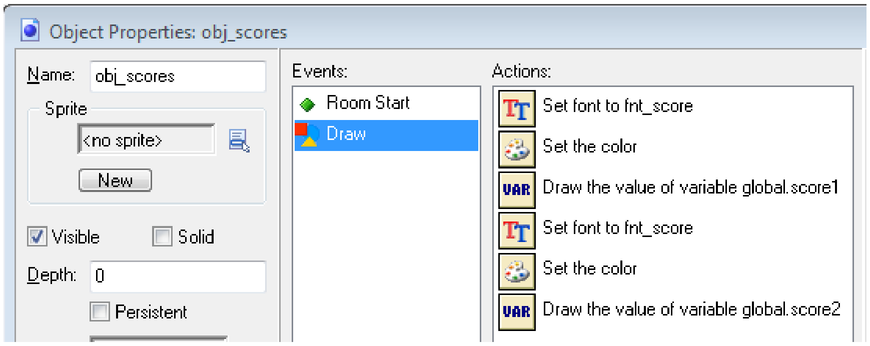
Applies to
● Self
○ Other
○ Object:

variable: global.score1

value: 0

□ Relative

The second event is to draw the scores for the first time.



Object Properties: obj_scores

Name: obj_scores

Sprite
<no sprite>

☑ Visible □ Solid

Depth: 0

□ Persistent

Events:
- Room Start
- Draw

Actions:
- Set font to fnt_score
- Set the color
- Draw the value of variable global.score1
- Set font to fnt_score
- Set the color
- Draw the value of variable global.score2

Global.score1 should be drawn in the same location as it will appear when updated: X 260, Y 10. Global.score2 is at 350, 10. Save and close the object.

For the object to do anything it has to be in a room. Place obj_scores anywhere in the second room, room_action. It appears as a blue circle with a question mark since no sprite was assigned to it. If you forget to do this, you will get an error message when you run the game.

That’s it! You have worked hard. Now for the test. Save your game and then run it.

Debug if there is anything not right. Ask the lecturer for assistance if you are stuck.

After it is working, are you sure you understood now why you made the events and actions you did? You may need to go back once you have made the game work to look at the parts and to become comfortable with why it works. Unless you understand both the how and the why you will not be able to make games of your own.

Remember that if you want to take your game to another computer to play and show off, you can drop down the File Menu and choose Create an Executable. The exe will be over two megabytes as it will contain all the resources but it will play on any Windows computer without needing the Game Maker program. The two Game maker files made when you first save the game are by comparison tiny, under 100kb.

Challenges

If you have time, try and make the game more challenging. The easiest ways to do this are to speed up the ball and to make the bats smaller. Can you think of other ways?

The game would be better if the player who gets to 21 first is declared the winner. Three actions useful for this would be the Test Variable and Display Message, followed by Restart Game.

This is a real challenge. There is a bug in this game to do with the re-creation of the ball. Can you see it when you play? Can you figure out what the problem is and how to fix it?

The original Pong
(home not arcade version).

Notice the controls.

* * * * * * * * *

Recreating Space Invaders

Space Invaders was first released in 1978 in Japan. It became so popular the government had to mint more coins as so many were being put into arcade consoles like this.

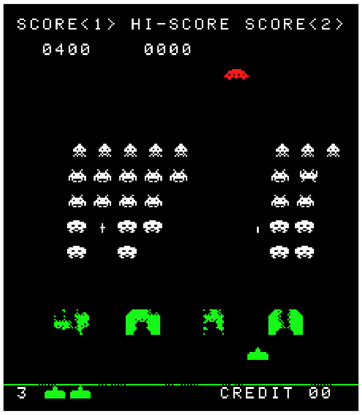
It is one of the most successful and influential video games of all time.

The game description is:

> *Alien spaceships are attacking. You as the defender can return fire with your laser canon. There are blocks you can hide behind to protect you from their fire, but your own fire will destroy your cover.*

Play the game placed in the network folder to get a mental picture of what you will be making.

Start a new game in Game Maker. The original Space Invaders had three types of alien spaceships in five rows which advanced down the screen firing at the laser canon of the defender at the bottom. We will start by making sprites for three types of invaders by using the built-in sprite editor.



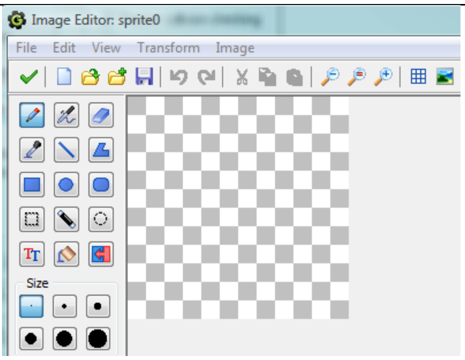
This original display on the left was monochrome with an overlay to make some objects look colored. But later versions used colour graphics and so can we!

We will make three sprites, each one a different colour. Later we will animate them.

Create a new sprite and click on the Edit Sprite button.

Drop down the File menu, choose New and accept the default size of 32X32.
Double click on the box which says image 0 and the Sprite Editor window will open.

It opens too small for us to work, so click on the plus magnifying glass a few times to make the work space larger.
Then click on the grid symbol on the right in order to overlay the grid as a placement aid. The default size is only 32 pixels by 32 pixels, we might as well see each one as we colour it. If you make it big enough the grid will match the squares.



Click on the Pencil tool top left and then move your cursor over the colour pickers to the right and left click on your choice of drawing colour. Draw on the image. Hold down the left button to drag a colour. Right click to erase and leave the pixel transparent.

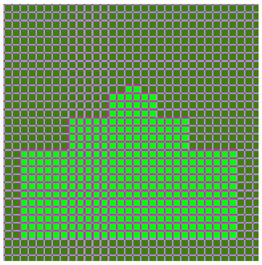
Our simple sprite will move over the room's black background.

When you have finished creating your first invader, click on the green tick to close the Image Editor and then again to close the Sprite Editor (where you will work on animations). In the Sprite Properties window, click on the Center button (you need to be in Advanced not Simple mode) to put the origin at 16,16. Notice the cross hairs over the image. Later when it is firing bullets it will appear more realistic for the bullet to emerge from the centre of the spaceship not the top left corner, which is the default point of origin of a sprite. Name the sprite "spr_Invader1", save and close.

Create your second sprite in a different colour. Name it "spr_Invader2". Do not spend the whole session making sprites using pixel art or you will not have time for the game mechanics. You can always return and replace or upgrade a sprite.

Your third sprite will go on the top row and in the original game was worth the most points. To be sporting we will make it smaller and thus harder to shoot. When the Sprite Editor is open drop down the Transform menu and choose Resize Canvas. Make this sprite 16 by 16 pixels. Create it, name it "spr_Invader3", save and close.

We will return to the sprites later and animate them.
For now let's make a 32X32 sprite for the laser canon which will defend Earth against the invaders from space.



As you can see, this sophisticated device can be drawn by using the rectangle tool.

Be sure to put the point of origin in the centre.

Name it "spr_Defender", save and close.

Now to create a new object, name it obj_Defender and assign spr_Defender to it.

We want it to be able to move sideways and to fire, when the player presses the appropriate keys.

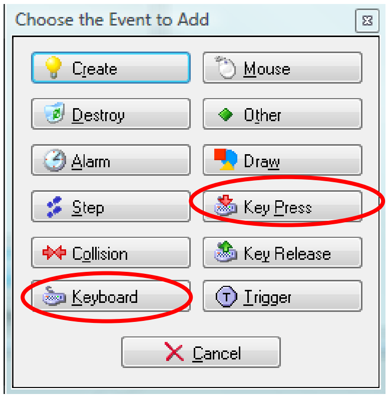
Create a Keyboard event on the right arrow and as an action choose Jump to a Position from the Move tab. We want it to move a little to the right but not up or down so set x to 5 and leave y as 0. Tick the relative box. Now add a Keyboard event on the left arrow with a Jump to a Position action. Since we want it to move to the left, we set x to -5, y as 0 and tick the relative box. We will use the Space bar to fire but we don’t have any bullets yet so we must first create one.

Create a sprite called “spr_Bullet”. Use the Image Editor to make a 16 by 16 image. The same bullet will be fired by both invaders and our defender so the bullet should look good moving either up or down.

The sprite by itself doesn’t do anything, you must create an “obj_Defender_Bullet” and assign spr_Bullet to it.
We want the bullets fired by the two sides to be separate, even if they look the same, as it is easier to program their behaviour when they move and when they hit something. So create an “obj_Invader_Bullet” and assign spr_Bullet to that. Arms dealers profit from both sides in a space war.

Now we can reopen obj_Defender and add a KeyPress/Space bar event (I don’t think the pun was intended). As an action choose Create a Moving Instance of from the main1 tab. Applies to self. Set the object to be obj_Defender_Bullet, leave x and y as 0, set the speed to 15 and the direction to 90 (which means straight up, see Appendix 1) and tick relative (so it seems to be fired from the defender). If you want to be able to fire a continuous stream of bullets you would use the Keyboard/Space event.

A Keyboard event repeats if the key is held down



A Key Press event does not repeat

Letting Invaders Move and Fire

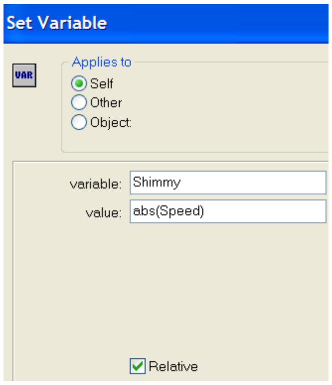
Now it is the turn of the invaders to move and fire. This is more complicated as we want them to move sideways and to advance down the screen, firing as they go. And we have three different types of alien yet we do not want to repeat a complicated set of properties three times. Game Maker has a way of avoiding such repetition, it uses the programming idea of **inheritance**. This means we can create a parent object and give it all the properties and actions we need. Then we can say the three invaders all have this object as their parent, thus each will inherit the parent’s properties and abilities.

Create a new object and name it obj_Invader. Do not assign a sprite to it. This object is the heart of the game and where it is the most complicated so you need some patience with this part. We will create two variables, one will be used to set the sideways movements and the other to set the speed (as we want the invaders to speed up as the game progresses). First we create and initialize their values.

Add a Create event to obj_Invader. Give it a Set Variable action on the Control tab. Call the variable Shimmy and give it the value of 80 but do NOT tick relative (it might give the game engine a problem as there is no pre-existing value to make it relative to). Add another Set Variable action, call the variable Speed and the value of 1, again NOT relative. Be very careful how you spell and capitalize the variable names. Variables are case sensitive. Shimmy is a different variable from shimmy. The word Shimmy does not mean anything to Game Maker, it's been chosen as it's descriptive for humans.

A game made with Game Maker works at a default speed of 30 steps a second. A Step event therefore happens 30 times a second. We will add a Step event and through its actions control the sideways and downwards movements. The software is checking for any Step event 30 times a second so its very useful for things which happen often.

Add the Step/Step event to obj_Invader and a Set Variable action which resets the variable Shimmy to a value of abs(Speed), make sure you tick relative. Here you are calling a built-in function which returns the absolute value of Speed, in practice this means a positive number.


Set Variable

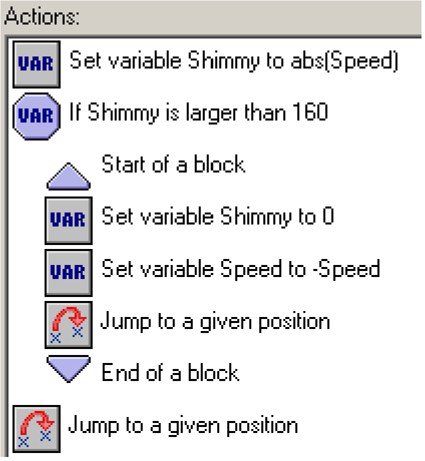
Applies to
● Self
○ Other
○ Object:

variable: Shimmy
value: abs(Speed)

☑ Relative

Now to check how far sideways it has moved. For this we need an If structure. Add a Test Variable action to the Step event with the variable set to Shimmy, the value to 160 and the operation larger than.

Start a block then two Set Variable actions. The first sets Shimmy to 0 and the second sets Speed to a value of –Speed, both not relative. The effect of these is to switch direction.
Add a Jump to Position, x = 0 and y = 5, relative. This will move the object down the screen five pixels.
End the If block.
Jump to Position, x = Speed and y = 0, relative. This is the else part of the structure if it hasn't moved sideways enough yet.


Actions:

Set variable Shimmy to abs(Speed)
If Shimmy is larger than 160
Start of a block
Set variable Shimmy to 0
Set variable Speed to -Speed
Jump to a given position
End of a block
Jump to a given position

Save and close obj_Invader.

Now to make this object the parent of the other three invaders so they can inherit this pattern of movement. Create three objects: (obj_Invader1, obj_Invader2, obj_Invader3) and assign the appropriate sprite to each. For each of the three set its parent to be obj_Invader.

What good is a bullet if it doesn't destroy something? Re-open obj_Defender_Bullet and add a Collision event with obj_Invader. As actions add two Destroy Instance actions from the main1 tab. Use one action to destroy the bullet itself. The other to destroy the other (the invader).

We have created some objects so time to place them in a room and test the game so far.

Create a room named rm_Action with the background colour set (what else) to black. Place in it one defender near the bottom and three rows of invaders, one of each type, with the smallest, obj_Invader3, on the top row. (The original arcade game used five rows of eleven but our screen is rather too small for that many.)

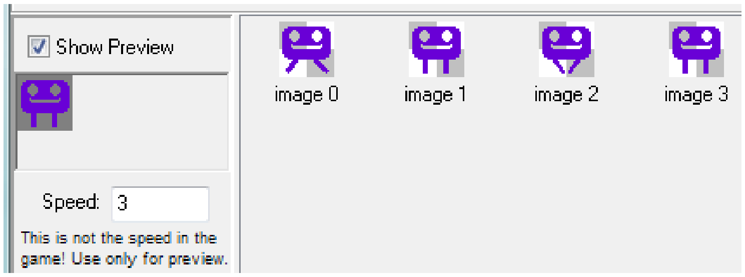
Click on the floppy disk icon to save the game and name it Space Invaders. Play the game. You should be able to move and shoot with the defender and the invaders should move across and down the screen and be destroyed if they are hit. Debug if necessary.

The game will be better if we can hear the bullets being fired (George Lucas says sound carries in a vacuum).

Create a sound and load fire.wav from the resources folder. Add a Play sound action to the Press Space bar event of obj_Defender.

Animating the Invaders

Would you like to animate the invaders? They are more fun to destroy if they wriggle. Open up the first invader sprite. In the Sprite Editor, drop down the Edit menu and choose Copy and then Paste. Open up the copy and alter it in some way. A complete cycle will be at least four sub-images so design your changes over that many so that when it comes back to the start the fourth is already returning to its appearance, otherwise there will be too much of a jump. For example, note the fourth is a copy of the second:



☑ Show Preview

Speed: 3

This is not the speed in the game! Use only for preview.

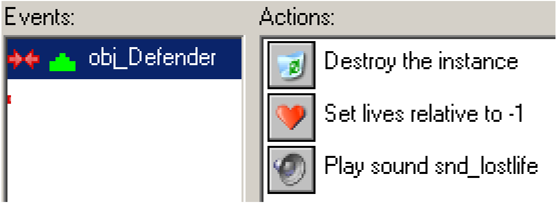
You can tick Show Preview and set a slow speed of 3 to examine the animation. Fine tune until you are happy. Then do the next one until all three sprites are ready.

If an animation flickers too much, if it goes too fast at the game speed of 30, double the number of sub-images by copying each one and pasting the copy next to the original (the arrow keys let you move a sub-image). The effective speed will be halved since each identical sub-image is shown twice.

Play the game and shoot your creations.

We are being unfair to the aliens having them with no weapons. Create a new object named obj_Invader_Bullet, assign spr_Bullet to it. The bullet will kill a defender so we need to add a collision event with obj_Defender set to destroy self. We want the bullet to disappear but the graphic of the Defender to remain (it will lose a life instead)

The game would be over immediately if there was only one defender so we will allow a player three lives. If a defender is destroyed the number of lives should be decreased. Add a Set Lives action from the score tab and set the value to -1, relative.



Events:
obj_Defender

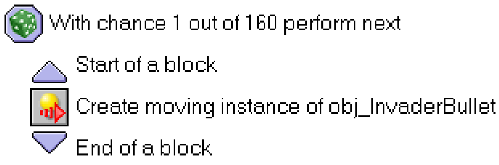
Actions:
Destroy the instance
Set lives relative to -1
Play sound snd_lostlife

While we are here we might as well play a sound effect. Create a new sound named snd_lostlife, load the sound "lostlife" from your Resources folder and add a Play Sound action to the collision event.

A bullet which has missed and gone out is no longer of use but the computer will keep tracking it unless we stop it doing so. Add an Outside Room event from Add Event/Other and a Destroy the Instance action, self. Click OK to save and close the object properties window.

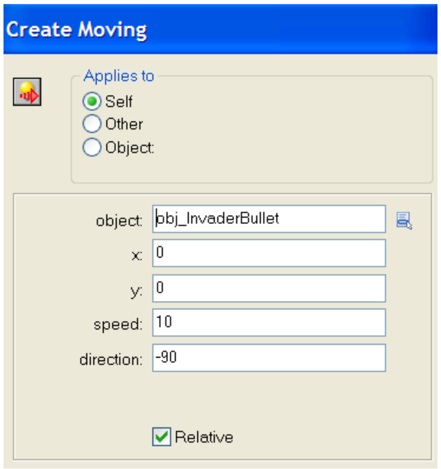
Getting Invaders to Fire

Now to get the invaders to fire their bullets. We want this to happen at random but at a pace we can control as the game designer. Game Maker has in effect a dice we can roll to determine whether or not to fire at each step. Go to obj_Invader and to the existing Step event just after the second Jump to Position action, at the bottom add from the control tab these actions:



With chance 1 out of 160 perform next
Start of a block
Create moving instance of obj_InvaderBullet
End of a block

These are the settings for the invader's bullets on the Create Moving Instance action so they fire downwards.



Create Moving

Applies to
● Self
○ Other
○ Object:

object: obj_InvaderBullet
x: 0
y: 0
speed: 10
direction: -90

☑ Relative

Run the game to test the bullets are firing.

If you want more or less bullets you can decrease or increase the number of sides from 160.

If you cannot see any bullets, it is possible that they are being fired but are destroyed before they emerge from the object. See Point of Origin in Appendix One.
If you have this problem, open up the sprites for the invaders and alter the points of origin to low down, X=16, Y=28. This will ensure the bullet is created low down in the object which is firing it down and so the bullet can emerge without colliding.

Did you notice there is a problem with the bullets being fired?

The bullets from the higher rows of invaders pass through aliens in the lower rows.

We can rectify this by opening up obj_Invader_Bullet and adding a Collision event with obj_invader with the action that it destroys self. That is the bullet is destroyed (disappears) if it hits another invader.

Creating Blocks to Protect the Defender

We will create the blocks next, so the defender can hide from the bullets. Create a sprite with the name spr_Block. Edit the sprite and use the rounded rectangle tool to make the basic shape you can then hollow it out using the background colour.

Create an object called obj_Block, assign your new sprite to it and add a Collision event with obj_Defender_Bullet which will destroy the other (the bullet). The invader bullets also will not harm a block so add a Collision event with obj_Invader_Bullet, with a Destroy Instance, other action.

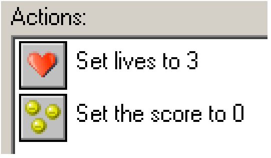
Go to the room and add three or four of the Block objects in a row above the defender.

Save the game and run it to test the blocks work. If they allow bullets through, and they are solid, it can happen that the top of the block is so thin that occasionally a bullet passes through before the collision is registered. Your blocks are simpler than those in the network game, which have been made far more sophisticated so they show damage.

Controlling When the Game Ends

We have all the elements of the game now but need a mechanism to control the game, to keep track of the lives and end the game if the last defender dies or if there are no more invaders.

Create a new object named obj_Controller with no sprite.
First we must initialize the values, so add a Create event with two actions (both on the scores tab). Set lives to 3, not relative and Set score to 0, also not relative, as these are the opening values.



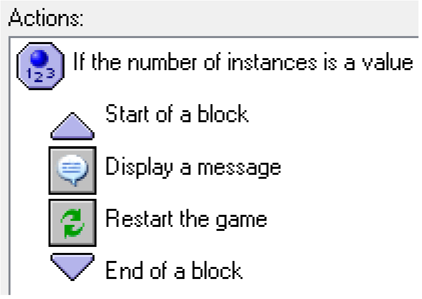
Actions:

Set lives to 3

Set the score to 0

We will deal with the destruction of all the invaders first by adding a Step event to obj_Controller with these actions.

The first is to check if the number of obj_Invader is equal to zero. If that is true a message will be displayed saying something like “My hero. You have saved the planet.” And the game will restart.



Actions:

If the number of instances is a value

Start of a block

Display a message

Restart the game

End of a block

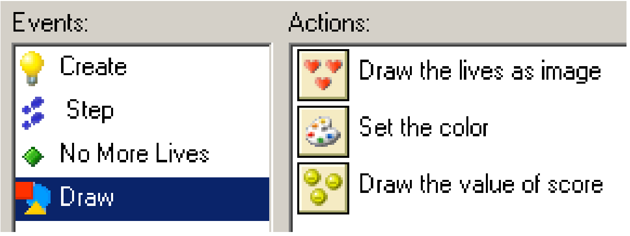
The alternative is easier. Under Other there is an event No More Lives. Add this and attach to it the action to Display a Message (from the main2 tab), saying something like, “Oops, you have been defeated, Earth will be destroyed” and an action to Restart the Game.

We make the score and lives appear on the screen by adding a Draw event with three actions:

Draw the lives as image, using spr_Defender, x =150, y = 12

Set the color, to white

Draw value of score, x = 10, y = 10, caption = Score.



Add obj_Controller to the room somewhere, it doesn’t matter where. It will not work until it is in the room.

Just a couple of things left to do. The first is to set the scores for each type of invader. Open up Invaders 1 to 3 and add a Destroy event with a Set score action. Invader1 is worth 10 points. Invader2 is worth 20 and Invader3 is worth 30, all scores relative.

Making the Invaders Speed up as they Move Down

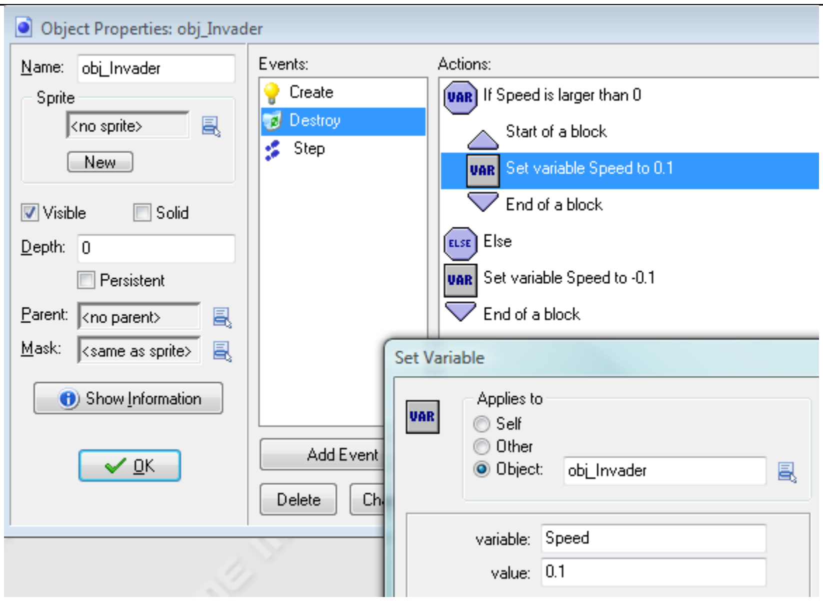
The final issue is to make the invaders speed up as they move down the screen, this will gradually increase the pace of the game, making it more exciting.

Open up obj_Invader and add a Destroy event before the Step event.

The screen capture on the next page will help you create this.

The logic is to test the value held in the variable Speed and then adjust it. The value of the Speed variable can be a positive or a negative number, we have to find out which before we can add to it. So we need an If structure.

We will increase the speed by 0.1. Which means increasing Speed by 0.1 relative if it is already larger than zero and by -0.1 relative otherwise.


Object Properties: obj_Invader

Name: obj_Invader

Sprite: <no sprite>

☑ Visible □ Solid

Depth: 0

□ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events:
- Create
- Destroy
- Step

Actions:
- If Speed is larger than 0
- Start of a block
- Set variable Speed to 0.1
- End of a block
- Else
- Set variable Speed to -0.1
- End of a block

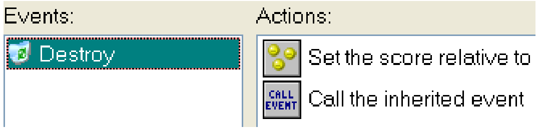
Set Variable

Applies to
○ Self
○ Other
● Object: obj_Invader

variable: Speed

value: 0.1

We have given this ability to the parent so we would expect its children to inherit. Unfortunately here each invader has its own Destroy event, and a local event will take precedence over any inherited event. We can get around this by opening up each of the three invader objects and adding to the Destroy event a Call the inherited event action from the control tab (only available in Advanced Mode). This adds the parent's Destroy actions to the children yet allows each child to have its own score.


Events:
- Destroy

Actions:
- Set the score relative to
- Call the inherited event

And that's the lot. Save your game and run it.

Debug if necessary.

Every game should have instructions, click on Game Information and claim the credit for your game as well as giving information on how to play the game.

Be sure to drop down the File Menu and create an executable which you can take home and show off. An executable should work on any computer running Windows, it doesn't need Game Maker.

If you want to finish off a game or work on your own games at home, you can use Game Maker for free on a Windows computer. A copy of the free version entitled gmaker80.exe at over 10MB is in the network drive or you can download it from yoyogames.com.

Challenges

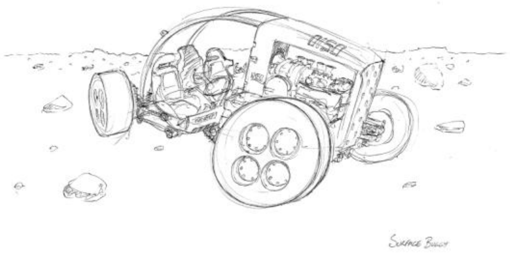
The defender can slide sideways off the screen. One way to stop this is to put an invisible block in the way. When the defender hits the block, its movement is reversed. You can make a small sprite, all black, not transparent, assign it to a solid object with the collision event and appropriate action, then place one instance in each of the bottom corners.

In the original a defender could only have one bullet on screen at a time. Recreate this with a Test instance count action when a defender fires and other appropriate actions. Hint: you will need to destroy a defender's bullet if it leaves the room or else you will never be able to get off another shot if one misses.

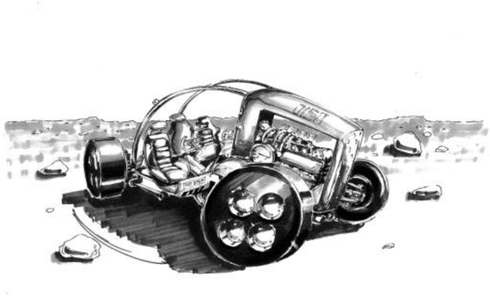
Also in the original, the blocks could be damaged by the defender's fire. There are several ways you could do this. Each block could have a Health and be destroyed or you can create two or three more graphics of a block showing gradual damage and substitute those with a Change Instance action each time a block is hit until there are no more, as in the game on the network. A tactic arcade players used was to damage their own block in order to shoot through it.

The network version has some added features such as an opening screen, you know how to do this from the previous session.

Concept sketch for a game by Todd Millias of a Mars buggy



The sketch after rendering



Recreating Pac-Man

Pac-Man was first released in 1979 in Japan. The *Killer List of Videogames* rates *Pac-Man* as the #1 video game of all time on its "Top 10 Most Popular Video games" list.

The game was developed by Toru Iwatani over eighteen months. The original title was pronounced *pakku-man* and was inspired by the Japanese phrase *paku-paku taberu* (the sound of a mouth opening and closing). Although it is often claimed that the character's shape was inspired by a pizza missing a slice, Iwatani admitted later that it was a half-truth and the character design also came from simplifying and rounding out the Japanese character for mouth. Iwatani's efforts to appeal to a wider audience — beyond the typical demographics of young boys and teenagers playing sports games or space shooters like Space Invaders—lead him to adding elements of a maze.

It was the first video game which appealed equally to female and male players. The sequel, Ms Pac-Man, was even more popular and was the first major game to star a female character in the lead.

The Japanese sales flyer from 1980 displays the Japanese title, *PUCK MAN*, as well as the original character design.

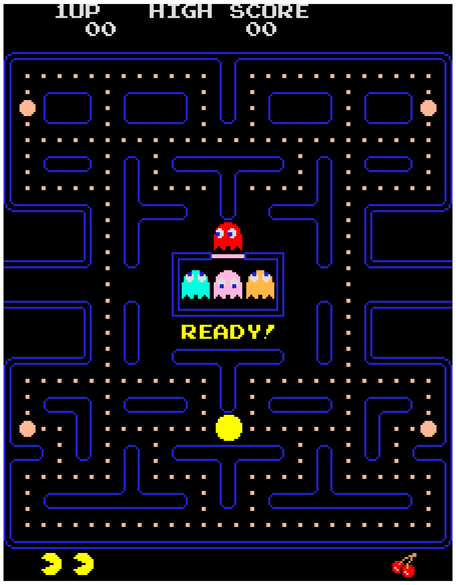
In America it was thought that Puck Man would be too easy for vandals to alter so the name was changed to Pac-Man.

Pac-Man was so popular it inspired a television cartoon series and a Top 40 single, the character appeared on the covers of Time and Mad magazines and led to over 30 sequels.

The player controls Pac-Man as he moves through a maze, eating pac-dots. When all dots are eaten, Pac-Man is taken to the next level. Four ghosts roam the maze, trying to catch Pac-Man. If a ghost touches Pac-Man, a life is lost. When all lives have been lost, the game ends.

Near the corners of the maze are four power-up objects known as "energizers" or "power pellets", which provide Pac-Man with the temporary ability to eat the ghosts. The ghosts turn a lighter colour when Pac-Man eats an energizer, and they are vulnerable to being eaten by Pac-Man.

The original game has 256 levels and can never be completed. Throughout the game "fruits" occasionally appear and give extra points if Pac-Man eats them. The information about them was kept in a single byte, which can only hold a number up to 255. So on level 256 the screen became garbled and the maze cannot be navigated. We will only create the first two levels but we will use the game design principle of increasing game difficulty with each successive level.



A screenshot from the original arcade version of the game, showing the four ghosts in their starting positions at the center of the screen and Pac-Man below. Four Energizers are visible near the corners of the screen. It was the first major game to use power-ups.

Play the game from the network drive to get a feel for what you are about to make. Start a new game in Game Maker.

Create sprites with these names, loading them from the Pac-Man folder:

spr_pacman
spr_pacman_left
spr_pacman_right
spr_pacman_up
spr_pacman_down

Check out what they are. Notice that the first is just a single image, it is the one facing forward when the character is not moving. The other four will be used when the player is pressing a key to move the character. Each of the four is a short animated sequence.

Now load sprites for the food and the wall:

spr_food
spr_wall

The wall is ugly but it will suffice while we build the maze and create the game mechanics, later we could alter its appearance. All the sprites are the same size, 32X32, even the small image of the food. A maze game is built on a grid and it is very important that objects are all the same size as the grids.

The walled corridors through which Pac-Man will move will be 32 pixels wide, the same width as the character itself, which will make changing direction difficult unless the player times their key presses exactly right. To overcome this alignment issue and make play smoother, Game Maker has a Check Grid action. This is a conditional action which checks to see if the alignment is correct. If we combine this with making sure that there is not a wall directly in front of the direction of movement, then we have the basis of smooth movement in any maze game.

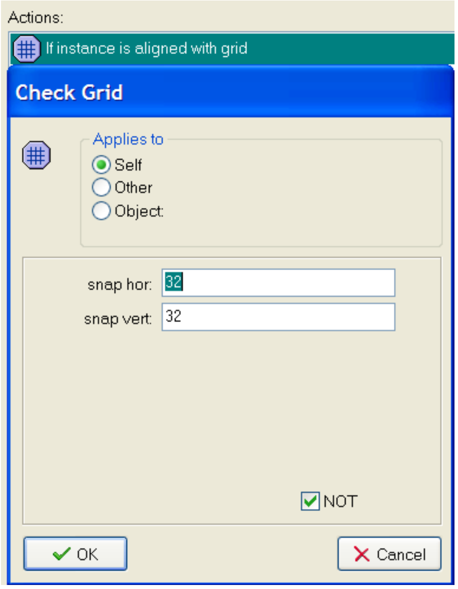
The idea is that we will first check to see if Pac-Man is aligned properly, if it isn’t a key press has no effect. If it is, and there is nothing in front of it, the character can move in the direction the player wants. If there is a wall blocking it, then the standing still appearance will remain.

Create an object, called obj_wall, assign the wall sprite and make it solid.

Pac_Man

Create an object, called obj_pacman, assign spr_pacman.
Add a Keyboard<Right> event and drag in to the action column Check Grid from the control tab. Set both Snap Hor and Snap Vert to 32 (as our grid size will be 32 square). Enable the NOT option, as we are looking to see if it is not aligned with the grid.

If it is not aligned properly we want nothing to happen, so add an Exit Event action from the control tab.



Actions:

If instance is aligned with grid

Check Grid

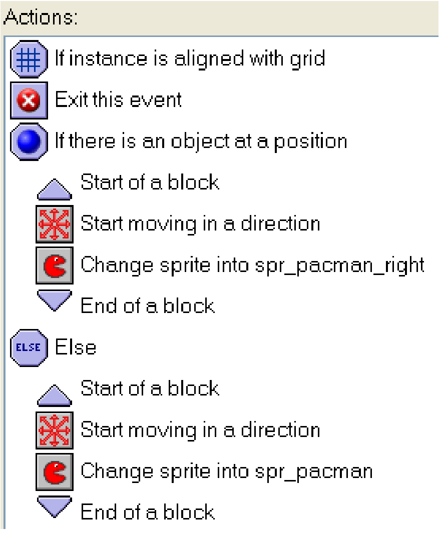
Applies to
● Self
○ Other
○ Object:

snap hor: 32
snap vert: 32

☑ NOT

Now we check to see there is nothing in the way, and if there isn’t anything, move our character.

Add a Check Object action from the control tab. Select the wall object and turn on NOT. Set X to +32 and Y to 0, select the Relative option so it checks there is no wall object one square to the right.
Add a Start Block action followed by a Move Fixed. Select the Right arrow and set Speed to 4 (the speed must divide exactly into the grid size or the character will not stop exactly in the right spot).
Include a Change Sprite action and select spr_pacman_right. If you set the sub-image to -1 the animation keeps playing. Leave speed as 1.
End the block.



```
Actions:
If instance is aligned with grid
Exit this event
If there is an object at a position
    Start of a block
    Start moving in a direction
    Change sprite into spr_pacman_right
    End of a block
Else
    Start of a block
    Start moving in a direction
    Change sprite into spr_pacman
    End of a block
```

If there is something in the way we will stop Pac-Man from moving. Add an Else action, start a block and include Move Fixed. Select the middle square and set Speed to 0. This Move command means the object stays still!

Add a Change Sprite action using spr_pacman.
End the block.

We have made our character move right but we want the other three directions. The quickest way is to click on the first action and Shift-Click on the last, right click in the selection and choose Copy. Add the next movement event and Paste into the Action column. Modify the Move Fixed, the Change Sprite actions and set the X and Y values for Check Object. (Remember that X increases as you move across the screen from left to right while y decreases as the object moves up the screen, so moving up vertically means X=0 and Y=-32).

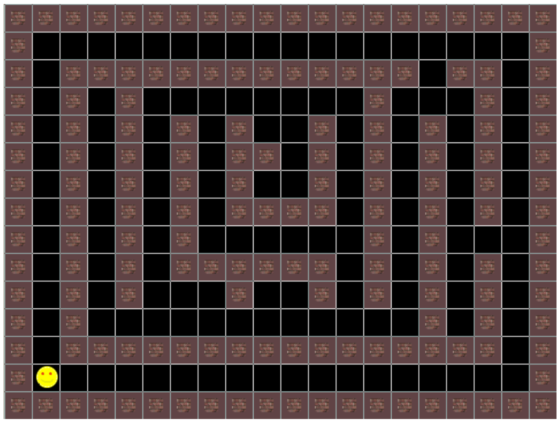
We should save our work.

Time to create a maze and test our hero.

Insert a new room. On the settings tab name it rm_1 and give an appropriate caption. On the backgrounds tab set the background colour to black. Move to the objects tab and set the Snap X and Snap Y to our grid size of 32 pixels. The default is too small at 16 by 16.

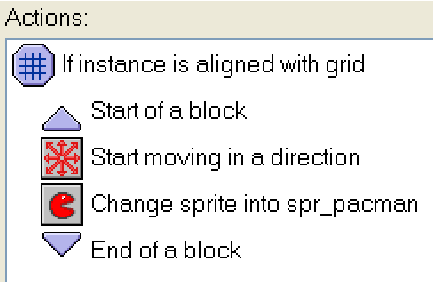
Select the objects tab and create a maze that looks something like this from instances of obj_wall. We want a central area for the ghosts. The area will also function as the way Pac-Man gets to the next level once he eats all the food.

Put Pac-Man in the starting position and test the game.



We have a problem. The Keyboard actions start Pac-Man moving but nothing stops him except a wall. We need him to stop when the player stops pressing any arrow key.

There is a <No Key> event on the Keyboard menu which we can use to pause the character.
Add this and then a Check Grid action with the Snaps set to 32 **but leave NOT disabled**.
Include a start block and a Move Fixed. Choose the centre square with a Speed of 0 to hold it in place.
Change the sprite to spr_pacman.
End the block.

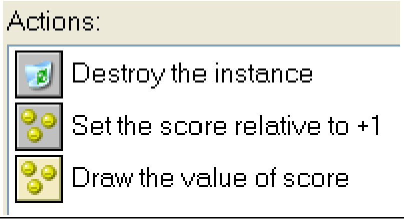


Actions:

If instance is aligned with grid
Start of a block
Start moving in a direction
Change sprite into spr_pacman
End of a block

Try it now and notice the improvement.

Pac-Man hungry!
Create an obj_food, assign spr_food. We need Pac-Man to gobble up his food so add a Collision event with obj_pacman which destroys self, a Set Score which adds one, relative and then a Draw Value.



Actions:

Destroy the instance
Set the score relative to +1
Draw the value of score

Add some pac-dots, the food, to the room but be very careful you do not over-write a wall as you go.

Save the work and test the game play.

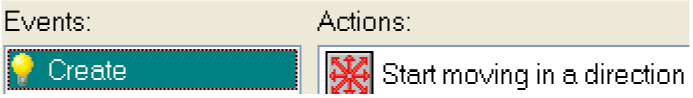
Blinky the Red Ghost

Now for the opposition.

The original game has four ghosts, each with different characteristics. In this first level we will add the first ghost, whose name is Blinky.

Create a new sprite, spr_Blinky and load the image from the network folder. You will see it is animated, which is how it got its name. Blinky will kill Pac-Man if it catches him. Assign the sprite to an obj_Blinky

We can give Blinky initial movement on a Create event because we know where it will start and where there is free space around it. Select a movement with a speed of three (so Pac-Man is faster, on this first level at least). The initial direction must fit where you locate Pac-Man in the maze and where there is freedom to move. Later you may need to fine tune the initial movement or to remodel your maze layout.



Events: Create

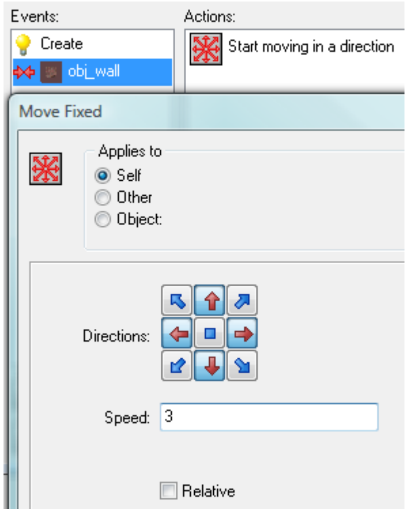
Actions: Start moving in a direction

Blinky is going to start bumping into walls and we want it to not just reverse direction, or it will become easily trapped, but to be capable of moving to both sides as well as reversing.

Add a Collision event with the wall.

Blinky will be moved by the computer. It will choose at random which of the four directions it will choose.
This has the practical benefit of making it less likely that it will become trapped.

However, you do need to watch it moving in the game and may well need to reposition walls in the maze if Blinky can become trapped or if there is an opening in a T junction it will never take because there is no collision to make it turn at that point.



Events: Create, obj_wall

Actions: Start moving in a direction

Move Fixed

Applies to: ● Self ○ Other ○ Object:

Directions:

Speed: 3

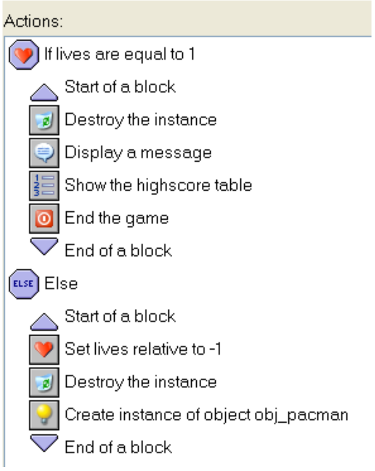
□ Relative

direction is the name of a built-in variable, which keeps track of which way a graphic is facing. Variable names are case sensitive in Game Maker so direction is NOT equal to Direction.
If you turn in direction+90 you are turning at right angles in an anti-clockwise manner.

When Blinky Meets Pac-Man

Blinky will destroy Pac-Man. But that could end the game too quickly, so we will give Pac-Man three lives and each encounter will cost a life. When he is down to his last life and is touched by Blinky then a message will show and the game can end.

We need to start by checking if he is down to his last life.
Open obj_pacman and add a Collision event with Blinky.
On the right are the actions.
Both Destroy instances are set to Self.
The message says what you want said when Blinky wins.
The Create instance action creates a new Pac-Man. You need to give it a new location away from the nasty ghost. I used the original location (X=32, Y =416)

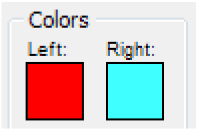

Actions:
- If lives are equal to 1
  - Start of a block
  - Destroy the instance
  - Display a message
  - Show the highscore table
  - End the game
  - End of a block
- Else
  - Start of a block
  - Set lives relative to -1
  - Destroy the instance
  - Create instance of object obj_pacman
  - End of a block

Power-Ups

Pac-Man pioneered the use of power-ups. These are things which if picked up in some way increase the capability of a character. In this game if Pac-Man eats an energizer it will turn the ghost a pale blue for a turn, and while it is that colour Pac-Man can destroy it and earn extra points.

Create a sprite called spr-energiser and load energiser.gif from the Resources folder. Create an object called obj_energiser and assign the sprite to it. Place three of them into the room in the free corners.

Right click in the resources tree on spr_Blinky and choose Duplicate. Name the copy spr_Blinky_scared. Click the Edit sprite button. Double click on image 0 to open it. We will swop all the red pixels for pale blue ones.
Choose the eyedropper tool and left click on a red pixel in the image. Right click on the pale blue square in the colour picker to the right. You should now have set the left and right colours.



Left click on the Change all pixels with the same color button and then right click in the image (the cursor will have changed).

When you have swapped the colours in image 0, to do the others you just have to open each in turn and left click on the Change all colours button/ right click in the image. Save and close the sprite.

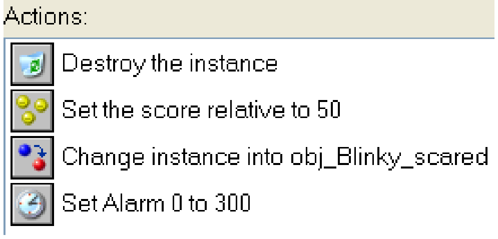
Now we can change the game so that Blinky becomes scared for a time after Pac-Man eats an energizer. And with good reason, Blinky becomes vulnerable.

Create an obj_Blinky_scared and assign the sprite. Make Blinky the parent of Blinky_scared so it can inherit the ability to move. Unfortunately if we do that, it will also inherit the ability to destroy Pac-Man in a Collision event.

First, though, we need to deal with Pac-Man eating an energizer.
We need to put this also into an event which happens to Pac-Man rather than to the energizer. See if you can work out why. Open up obj_Pacman, add a Collision with Energiser event and four actions:

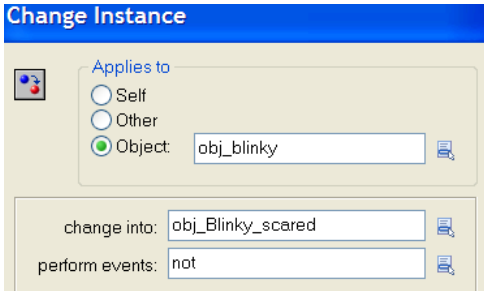
Destroy Instance, other
Set the score to 50, relative

Change the instance of obj_Blinky to obj_Blinky_scared
Set an alarm to 300 steps



Actions:

Destroy the instance
Set the score relative to 50
Change instance into obj_Blinky_scared
Set Alarm 0 to 300



Change Instance

Applies to
○ Self
○ Other
● Object: obj_blinky

change into: obj_Blinky_scared
perform events: not

This is our first use of an alarm but its function is obvious. You set it to when you want it to go off, remembering that there are 30 steps in a second. So we have given the player ten seconds to attack Blinky_scared.

We have set an alarm, so now we need to say what will happen when it rings. Add an Alarm Event and choose Alarm0 (you can set up to 12 alarms). As an action choose Change an instance and reset it so that Blinky-scared turns back into scary Blinky.

Have you figured out why we put this into obj_Pacman rather than into the energizer? When their collision occurs, the energiser is destroyed. It will not be around in ten seconds so the change back won't occur.

We are about to see another example of when it makes a difference to which object to give the collision event.

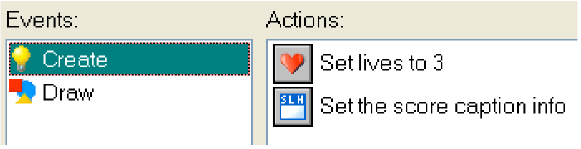
We will deal with the collision between Pac-Man and Blinky_scared as a property of Pac-Man. Open up obj_Pacman and add a Collision event with obj_Blinky_scared. The two actions we need are to Set the score to +200 relative and then to Destroy the instance, other. Save and close the object. Giving the actions to Pac-Man saves a problem with Blinky_scared's inherited abilities as otherwise Blinky-scared would inherit from its parent the ability to destroy Pac-Man.

Test the game and check that when Pac-Man eats an energizer Blinky changes to Blinky_scared and can be destroyed.

Scores and Lives

We will display the scores and the lives in the bottom row. For the lives we will use an image. Create a sprite spr_lives and load pacman_lives from the network folder.

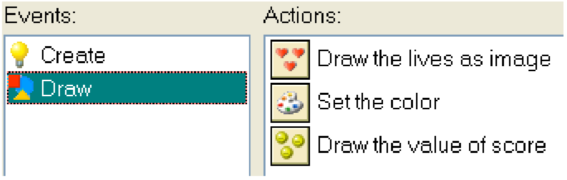
We have to make a controller to initialize the number of lives and to display the initial information. Create an obj_controller with no sprite. In a Create event place two actions. The first from the scores tab will set lives to 3. Unless we turn it off, the scores will automatically be displayed in the title bar. Add a Set Score Caption and turn all three options to Don't Show.



Events:
- Create
- Draw

Actions:
- Set lives to 3
- Set the score caption info

Then in a Draw event add these three actions: Draw the lives as spr_lives. Set the drawing colour to white and Draw the value of the score. (You won't get things to display unless you put them in a Draw event.)
Place obj_controller somewhere in the room.

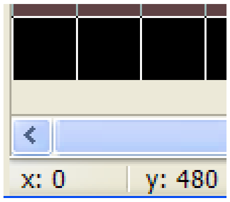


Events:
- Create
- Draw

Actions:
- Draw the lives as image
- Set the color
- Draw the value of score

Save the game and test it.

Doesn't it look ugly where the scores and lives are drawn? We can improve this by adding an empty row to the bottom of the room and drawing in that space. Open up the room and add 32 (the size of a standard grid) to the Height on the settings tab (change the default 480 to 512). If you move your cursor over a grid cell you will be able to see its coordinates in the status bar at the bottom. Therefore in the two Draw actions you know what values of x and y to enter. My values were 32 and 140 X and 480 Y for both.



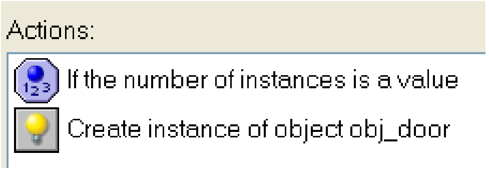
x: 0 y: 480

Finishing a Level

The player must have a way to complete the level. When all the food has been eaten, a door will appear in the center of the maze. If Pac-Man can reach it, he will move to the next level. To accomplish this we need to hide the door until a check reveals all the instances of food have been eaten. If Pac-Man gets to the door, the game moves to the next level.

First we need to create another level to move to, we will develop it more later. Right click on rm_1 and choose duplicate. Rename the new room rm_2. In many games the levels look different but here the layout will stay the same.

Create a new sprite called spr_door and load door.gif from the network folder. Assign this sprite to a new object called obj_door.
Add a collision event between the door and Pac-Man which triggers a Go to Next Room action from the main1 tab.
Open up obj_pacman and add a Step event. Attach to it two actions. The first to check if the count of obj_food is equal to zero. The second to create an instance of obj_door at 320, 180 (your numbers may be different if your layout is different).


Actions:
If the number of instances is a value
Create instance of object obj_door

Save and test the game. It takes a while!

We want players to carry over their score and lives from the first level. This is done by making obj_controller Persistent. On its Properties sheet turn this on so that the object retains its values when the player changes rooms. All objects are available in a new room but not the values they may be carrying.

That should complete Level 1. Test the game now and debug until you are satisfied. Ensure that when Pac-Man eats all the food the door appears and lets him through to the next level with his lives and score intact.

Level Two

Now we want to build a second level, after all the original game had 256. Each level in a game needs to challenge a player with progressive difficulty. There are a number of ways this can be done, such as new opponents and faster play. We will add a second ghost and make it actively target Pac-Man and move towards him, we can also speed up Blinky to make him harder to avoid or catch.

Speeding up Blinky is simple as long as we realize that the same object is the same in each room. If we speed the ghost up in level 2, it will be faster in level 1. The simple solution? Duplicate obj_Blinky, name it obj_Blinky2 and reset the speed of the copy in the movement actions from 3 to 5.

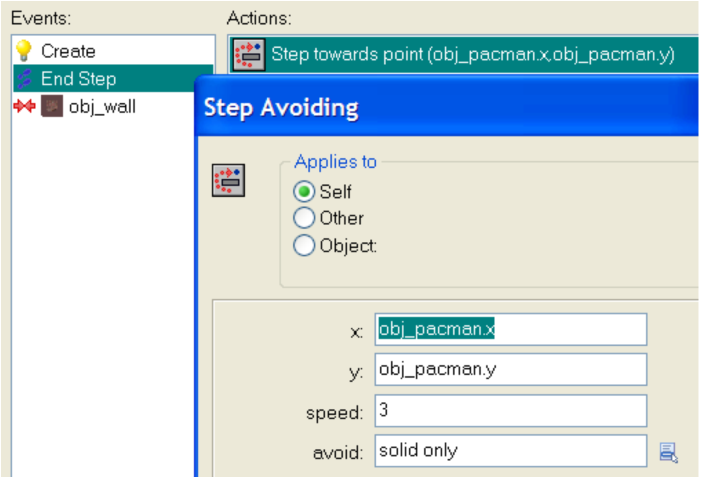
Now for the second ghost. Create a spr_Pinky using the gif in the Resources folder and make an obj_Pinky. Add it to rm_2, might as well alter the caption on the settings tab to warn the player of the new ghost. There will be one difference in behaviour between Blinky and Pinky, Pinky will actively hunt Pac-Man by moving towards him. Thus the actions will be the same with one addition.

Duplicate spr_Pinky, call the copy spr_Pinky_scared. Edit the sub-images as you did for Blinky so that it turns a light colour. Create objects for both, with obj_Pinky_scared make obj_Pinky the parent. Now we only have to give events and actions to the parent.

Duplicate obj_Blinky's events and copy and paste the actions into obj_pinky.

Pac-Man needs modifiying to deal with the new opponents. Make new collision events with both variants of Pinky and copy and paste the actions from Blinky. You will also need to modify Pac-Man's collision with the energizer, which then leads you to modify the alarm as well so both ghosts revert after ten seconds.

Now we have two ghosts which behave the same, so we can introduce the difference.



Step towards point (obj_pacman.x,obj_pacman.y)

● Self
○ Other
○ Object:

x: obj_pacman.x
y: obj_pacman.y
speed: 3
avoid: solid only

Open up obj_Pinky.
Add an End step event with a single action. From the move tab drag in a Step avoiding action and set:
X = obj_pacman.x
Y = obj_pacman.y
Speed = 3
Avoid solid only (only the walls are solid)
These will tell Pinky to try to move towards Pac-man. This technique is useful for homing missiles which follow their target as it moves.

Save your hard work and test the second level.

The more complicated the game, the more information a player needs. Game Maker uses the F1 key to provide Help. Double click on Game Information in the Resources tree to open up the mini word processor and enter instructions on how to play.

One game feature that increases motivation to play is to compete for position on a high score table. This is a built-in feature of Game Maker. Open obj_Pacman and to the collision event with obj_Blinky add a Show High Score action between the Display a message and End a game actions.

Challenges

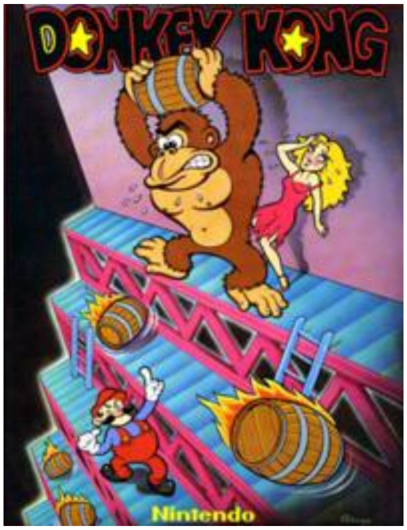
You want sound with that? There are some sounds in the resources folder which will increase playing pleasure. The background music has to be attached to obj_controller with its loop property turned on.

Pac-Man, the original, had some “fruits” which appeared at random and were worth extra points if eaten. In the Resources folder you will find cherry.gif. Add this as a sprite and an object. Place an instance of it in the bottom right of the first room. How to make it suddenly appear? Try a timer with a Jump to position action.

Add to the difficulty of level 3. Perhaps you could make a third ghost?

Pac-Man, the cereal offender

* * * * * * * * * *



A flyer for Donkey Kong, the game where Mario first appeared.

Game Maker

Recreating Mario

Mario is a 155 cm, chubby Italian plumber who lives in the Mushroom Kingdom. He is one of the most famous video game characters of all time. Mario has been the official mascot for Nintendo for over 25 years.

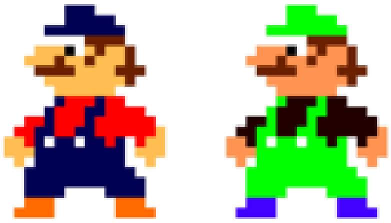
He has a brother, Luigi, but no surname, which hasn't stopped him appearing in three animated television series and a film, where he was played by Bob Hoskins.

Mario was first seen in the game *Donkey Kong*, where he was called Jumpman. Because of his ability to jump, Mario usually appears in platform games.

Mario loves pizza.

Mario can run and he can jump. He has to avoid the crabs or jump on them and can eat mushrooms. When he eats pizza he will be closer to his goal of reaching the Princess.
He has to watch out for a deadly fly and a pit but can climb up and down vines.

Mario's distinctive look is due to technology restrictions in the mid-1980s; with limited pixels and colors, the programmers could not animate Mario's movement without making his arms "disappear". Making his shirt a solid color fixed this. They also did not have the space to give him a mouth or ears, and they could not animate hair, so Mario got overalls, a moustache, sideburns, and a cap to bypass these problems. Mario's creator, Shigeru Miyamoto, has said that Mario wears a cap because he finds it difficult to draw hair.



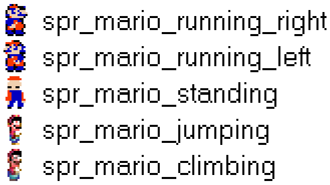
The original Mario and Luigi

Play the game in the network folder. In the last session we made a maze game, in this session we will make another type, a platform game.

Start a new game in Game Maker. Save it as Mario. This will be the largest game you make, most students will not finish it in the session but will need to complete it on their own time.

First we will load some resources to use for Mario and a block from which we can make floors and platforms.

Create these five sprites, loading from the network folder. We need one each for Mario moving left, right, standing, jumping and climbing.
We only need one object, obj_Mario. Initially it can be assigned spr_mario_standing. The other sprites will take over depending on which key the player presses.

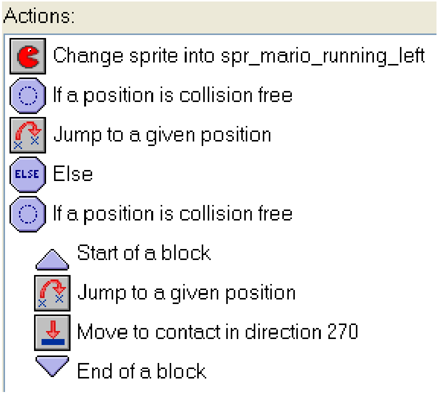

spr_mario_running_right
spr_mario_running_left
spr_mario_standing
spr_mario_jumping
spr_mario_climbing

Create a sprite and object for the block, using block.gif. Make obj_block solid so it can collide with moving objects like Mario and not visible, we can see it in the room as designers but it will not be visible to a player of the game. There are two smaller block gifs which we will use later. These will form the platforms on which Mario jumps.

Making Mario Move

Now we can define how the character will move. Mario will be moved left and right with the left and right arrow keys, and jump with the up arrow. We will also allow the player to move the character sideways while he is jumping. But we will not allow him to accelerate, that is his horizontal speed will be constant. Mario must also be able to fall. Game Maker makes this easy by letting us set the gravity but we must make sure he does not go too fast, and he must land on the floor, not embedded in it. Climbing comes later.

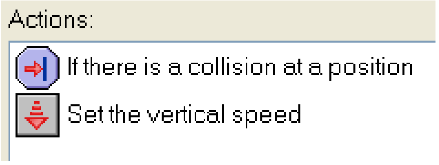
Let’s start with a Keyboard-Left event for obj_Mario. First change the sprite (to spr_mario_running_left, minus 1, 1). The minus 1 in the sub-image box ensures that the sprite is animated and the sub-image does not change if it is already facing in that direction.
Then check to see if there is anything in the way (x minus 4, y zero, only solid, relative).
If there will be no collision, then jump to position X minus 4, y zero, relative.
Now to allow for moving left while jumping.
Place an else button and check for a collision at minus 4, minus 8, only solid, relative.
Start a block, jump to position minus 4, minus 8, relative.


Actions:
Change sprite into spr_mario_running_left
If a position is collision free
Jump to a given position
Else
If a position is collision free
Start of a block
Jump to a given position
Move to contact in direction 270
End of a block

But we do not want Mario to become either embedded in the floor or there to be a small but visible gap. Insert a Move to contact (direction 270, maximum 8, only solid). With this action you can move the object in a given direction (here 270 degrees means straight down) until a contact is made, if there is already a collision it will not be moved. End the block. At a collision the moving object is placed back at its position just before the collision, this can leave a visible gap. Move to contact edges the object to the point where the collision occurs.

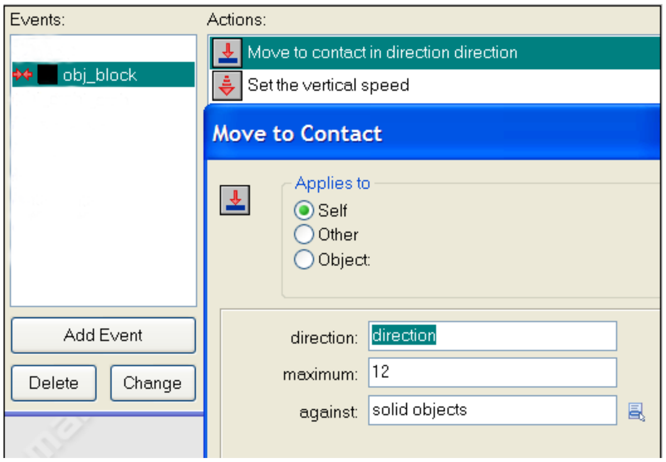
Add a Keyboard-Right event and copy and paste the same actions into it. Change the sprite to Mario_running_right. Remove the minus sign from x in the second and third actions. In the else block set the collision checking and the jump so the values are four for x and minus 8 for y.

Now to make Mario jump with an up arrow. Insert a Keyboard_Up event.
From the control tab select a Check collision action with x as zero, y as one, only solid, relative. This will ensure Mario can only jump when he is on a surface. Insert a Set vertical speed with the value minus 10, not relative.

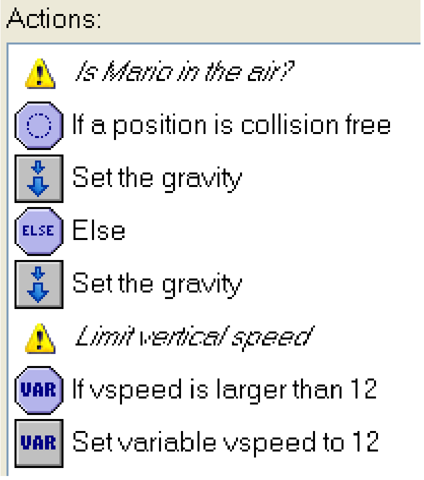

Actions:
If there is a collision at a position
Set the vertical speed

Mario has to interact with a block used for a platform in the same way as with one in a floor, that is neither embedding nor leaving a gap. Create an event of Collision with obj_block. There are two actions.
The Move to contact action uses the values: direction (type the word direction in the box. It's the name of a built-in variable Game Maker uses to keep track of which way an object is facing. Here it means whichever direction Mario is already moving in), maximum 12, against solid objects. The maximum limits how much the image can nudge in.

Set the vertical speed to zero (so he stops).


Events:
obj_block
Actions:
Move to contact in direction direction
Set the vertical speed
Move to Contact
Applies to
● Self
○ Other
○ Object:
direction: direction
maximum: 12
against: solid objects

One last event before we can test our hero. Mario cannot defy gravity. After a jump, he must be pulled downwards.
Add a Step event and then these actions.
From the control tab I have placed two comments to make it clearer what we need. First if Mario's jump is over, he can fall unless he has landed on something. However, in real life a falling body only accelerates to a certain speed then does not get any faster so we will limit Mario's falling speed.
Collision: x= zero, y = one, only solid, relative.
Set the gravity: direction 270, gravity 0.5
Else set the gravity to direction 270, gravity zero (yes, you could make gravity work in any direction, handy for a figure drifting in space or swimming).

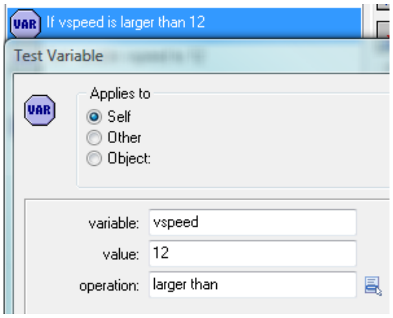

Actions:
Is Mario in the air?
If a position is collision free
Set the gravity
Else
Set the gravity
Limit vertical speed
If vspeed is larger than 12
Set variable vspeed to 12

Now we will use the built-in variable called vspeed.

Game Maker keeps a lot of continuously updated information about objects in a game, such as the direction in which they are facing and their vertical and horizontal speeds.

From the control tab choose a Check Variable action and set it to: vspeed larger than 12. Use a Set Variable to set vspeed to 12, not relative.

Now Mario cannot fall faster than a speed of 12.



If vspeed is larger than 12

Test Variable

Applies to

● Self
○ Other
○ Object:

variable: vspeed
value: 12
operation: larger than

One more last thing, when the player is not pressing a key we want Mario to be just standing there. Add a Keyboard, No key event and a Change Sprite action which displays spr_mario_standing.

We have given Mario the movements he needs, so let's check him out. Create a new room, called rm_1. In it place some blocks as a border around the room (or else Mario could fall out of the room) and a few as a floating platform. Place Mario in the room, standing on the blocks, save the game and run it.

Did Mario move left, right, jump and fall as expected? If not, debug until he does.

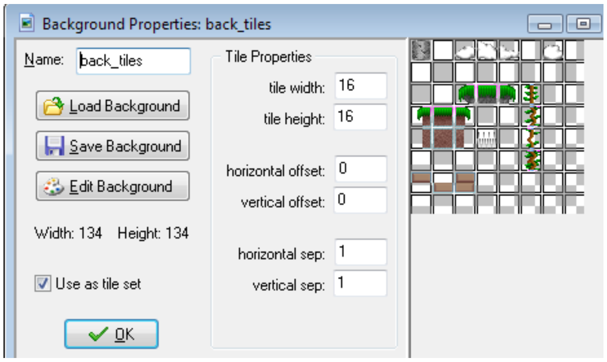
Using a Tileset

We have the basic movements but the appearance of the room needs some work. The background is straight forward, it's going to be the sky. Add a background using the sky graphic in the Resources folder. Use it in rm_1 as the background image.

For the platforms we need the block object because it is solid, but it is ugly. We will cover it up with a tileset. Tiles look good but they do not generate events, so a character cannot collide with them. Tiles are fast to load and use little memory, which makes them good in larger rooms. If we make the blocks invisible in their object property (after we have placed them in the room!) and place tiles on top of them, we get the combination we need of solidity and appearance.

A tile set is a number of images combined into a set with a small border between them to allow separation. Add as a background the image called tiles.gif.

Use these settings:



Background Properties: back_tiles

Name: back_tiles

Load Background
Save Background
Edit Background

Width: 134 Height: 134

☑ Use as tile set

OK

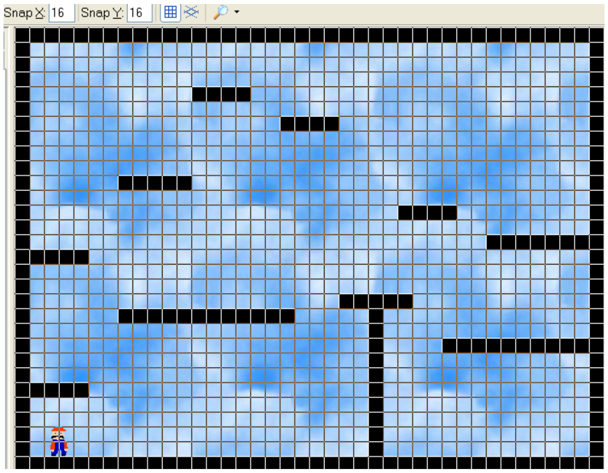
Tile Properties

tile width: 16
tile height: 16

horizontal offset: 0
vertical offset: 0

horizontal sep: 1
vertical sep: 1

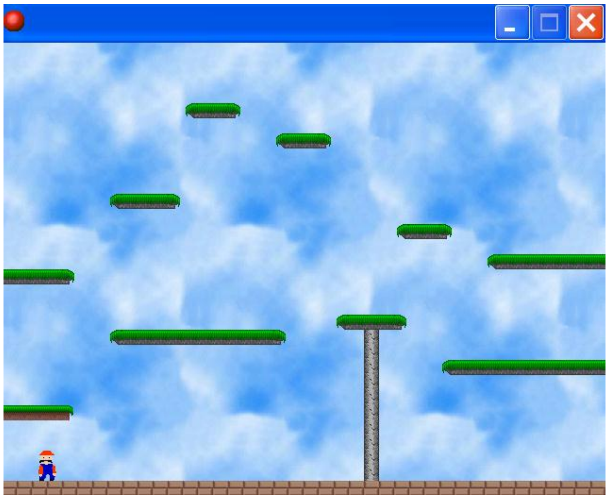
Be sure to set the horizontal and vertical separations to 1 and you can see that the divisions will be accurately placed.
Tick the box for Use as tile set.



In the graphic to the right and below you can see two alternative ways of viewing in the Room properties. The one to the right is the object tab and shows the blocks. Then the tiles tab was selected and the tile set used to overlay the objects. You click on one part of the tileset to place it in the room. However, it can be hard to see the blocks while you do this, so every now and again you need to run the game and check that it is looking good and that the platforms are in a suitable position to allow for jumps. You need to go backwards and forwards a few times to get it right.

You can tell if an invisible object is at a location by moving your mouse over a grid square and looking in the room's status bar. You can trigger the Visible property of obj_block on and off.

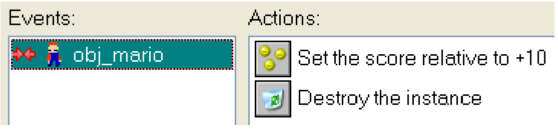
If there is a problem with the edge of a platform, you can substitute the last full block with two other objects, in the Resources folder there is a blockv and a blockh, both smaller and therefore a better fit with the rounded end tile.



When the game looks like that on the right and Mario moves well, we can put in some incentives and some opponents to make game play more interesting. The incentives will be some mushrooms, after which the Kingdom is named. In the lower right there will also be a pizza, Mario's favorite food, and when he eats this, he can move to the next level.

His opponent on this introductory, and deliberately easy, level will be crabs. They will destroy Mario if they meet him but if Mario jumps on them, the crabs can be destroyed and extra points earned.
Add three sprites: spr_crab, spr_mushroom and spr_pizza, using the images in the Resources folder. Create an object for each.

First the mushroom. When it collides with Mario he should score 10 points and the mushroom should be destroyed.



Events:
obj_mario

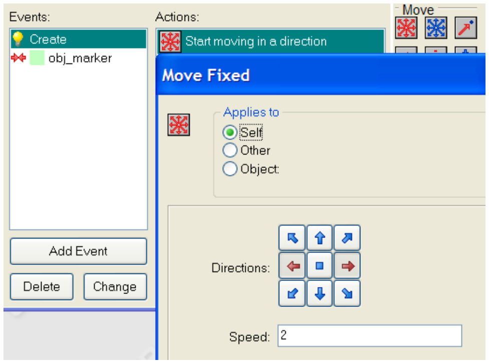
Actions:
Set the score relative to +10
Destroy the instance

Place some mushroom instances in the room and test the game.

Crabs

Now for the crabs. We want them to scuttle sideways first one way then the other. A common technique to use for this is to have an invisible marker, when the character collides with the marker, then the object reverses. Add a spr-marker, using the image marker.gif and create an object for it, leave the tick in the box for visible for now, it will help us see where we are placing instances. Later we can change it. Do not make obj_marker solid (we do not want Mario to interact with it).

Open the Properties box for obj_crab.
Make it solid so at least one of the objects is and therefore there can be a collision.
Give it these two events.
On the Create event make it Move Fixed to either side at a speed of 2, to start it moving.
When it collides with obj_marker, it should Reverse Horizontal Direction.



Applies to
● Self
○ Other
○ Object:

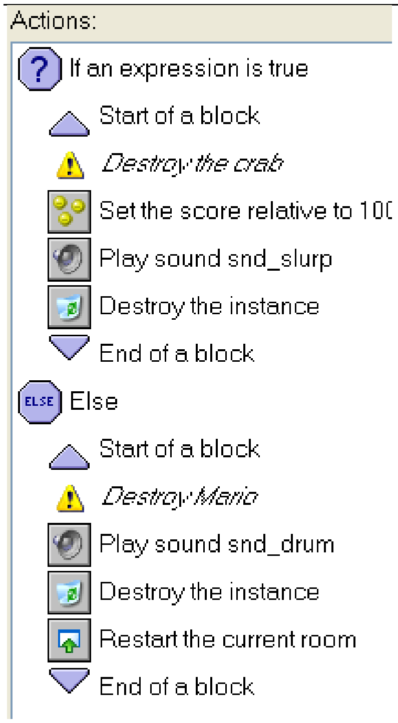
Speed: 2

Open up rm_1 and place some markers and instances of crabs.

Save and test the game so far.

Did you notice the quivering crabs? First check that you have made the Reverse direction for the crab not the marker. If it still quivers, the problem is the collision checking. Often with two sprites, especially if they are animated, you need to turn off Precise collision checking (different sub-images may collide at different points so the image seems to quiver as it moves the sub-images). If the image area within the sprite is quite small you may also need to set the Bounding Box to manual, although here all the central images almost fill the box anyway. Change the collision settings for the crab and you might as well do it for the Mario sprites while you are about it.

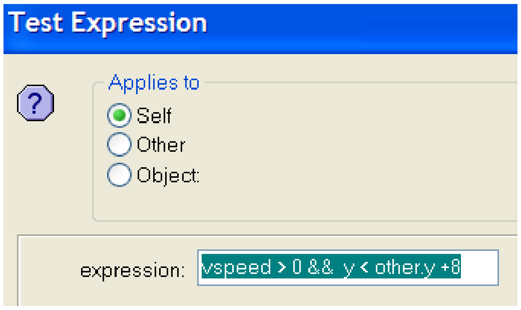
Now to let Mario crush crabs. We want two possible outcomes. If Mario is moving sideways when he collides with a crab, it is Mario who will be destroyed. But we want Mario to be able to jump and land on a crab and crush it, earning extra points. Thus we need to know when they collide if Mario is moving downwards, that is hitting it from above.


Actions:

- If an expression is true
  - Start of a block
  - *Destroy the crab*
  - Set the score relative to 10(
  - Play sound snd_slurp
  - Destroy the instance
  - End of a block
- Else
  - Start of a block
  - *Destroy Mario*
  - Play sound snd_drum
  - Destroy the instance
  - Restart the current room
  - End of a block

These are the actions which will take place for obj_Mario at a collision event with obj_crab. We first test to see if Mario is moving vertically and then if he is above the crab. Both can be done in one expression

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


Test Expression

Applies to
● Self
○ Other
○ Object:

expression: vspeed > 0 && y < other.y +8

The double ampersand && means “and also”.
If this expression is true then Mario must be moving down on top of a crab, in which case Mario gets 100 points, a sound will be played (you have yet to add the sounds so you need to add these two), and the crab is destroyed.

Otherwise Mario is not jumping on the crab so another noise will be played, the instance of Mario is destroyed and the room restarted.

Don’t forget the start and end blocks or it will not work.

Save and test Mario can jump on a crab and destroy it.

At last, Pizza

The end of the level has to be attainable. Create a spr_pizza using pizza.gif and assign it to an obj_pizza. In the object’s properties add a collision event with Mario. The two actions are to add 50 points relative to the score and to go to the next room. Place an instance of the object in the lower right corner of rm_1. Save the room and close it. Right click on rm_1 in the resources list, choose duplicate and rename the new room as rm_2.

We have almost completed level one. Let’s enhance game play with some sounds. Create in total five sounds using these files from the network folder:

• spring.mp3	(attach to when Mario jumps)
• poiee.mp3	(attach to when the mushroom collides with Mario)
• drum.wav	(attach to when Mario is eaten by the crab)
• slurp.wav	(attach to when Mario crushes the crab)
• level.wav	(attach to when the pizza collides with Mario)

Save, test and debug.

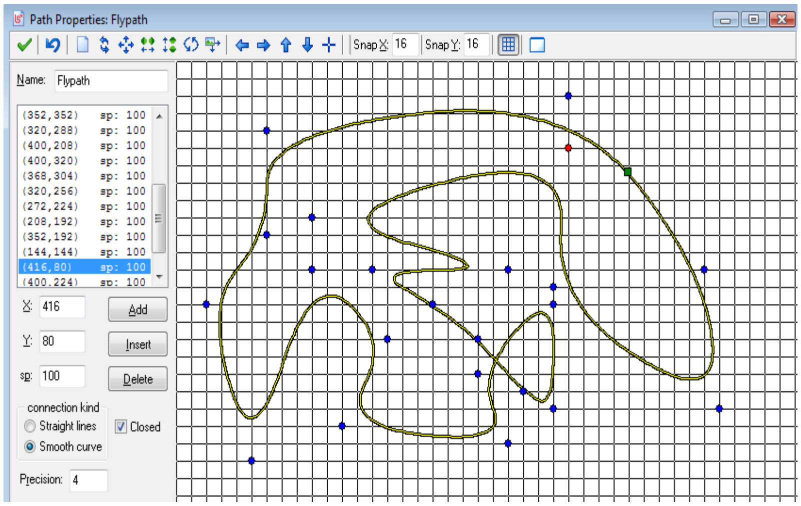
One level down, one to go.

The Fly on a Path

In the next level we are going to introduce two new concepts. Firstly we will add a new opponent, a fly. The crab was very predictable scuttling sideways in a straight line. The fly will move in a looping fashion. Secondly we will only allow a player to see part of a room, thus forcing them to explore.

The fly will move along a **path**. If we make this path quite convoluted, it will make it harder for the player to predict its movements.

From the icon bar in Game Maker click on the Create a Path icon. Name the path Flypath. Left click within the grid to place locations. Do not try to replicate the path in the graphic to the left, make your own. As you click, you will initially see red squares and straight lines joining them.



Path Properties: Flypath

Name: Flypath

```
(352,352)   sp: 100
(320,288)   sp: 100
(400,208)   sp: 100
(400,320)   sp: 100
(368,304)   sp: 100
(320,256)   sp: 100
(272,224)   sp: 100
(208,192)   sp: 100
(352,192)   sp: 100
(144,144)   sp: 100
(416,80)    sp: 100
(400,224)   sp: 100
```

X: 416

Y: 80

sp: 100

connection kind: ○ Straight lines ● Smooth curve

☑ Closed

Precision: 4

Snap X: 16 Snap Y: 16

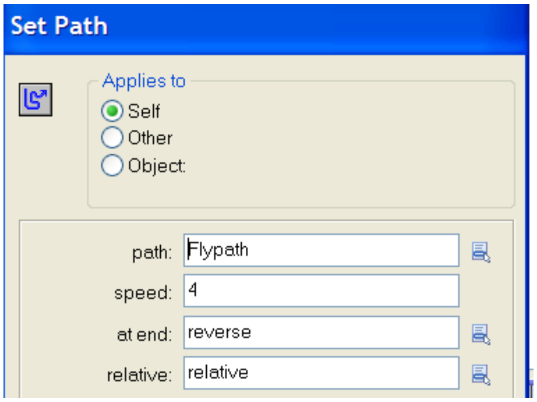
You can move the squares and see their positions detailed in the list. Ensure none of the positions is outside the 640 by 480 of the room.

To smooth the path click on the Smooth Curve radio button. The squares turn to blue but you can still reposition the path. Click on the green tick to save and close the Properties window.

Create a new sprite called spr_fly and load fly.gif from the Resources folder. The graphic is a little dirty and you might like to edit it to make it cleaner.

Create an obj_fly, assign spr-fly and on the Create event add a Set Path action from the move tab.

Make the settings match the graphic to the right.



Set Path

Applies to: ● Self ○ Other ○ Object:

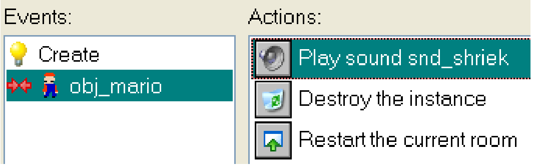
path: Flypath

speed: 4

at end: reverse

relative: relative

We want a sound effect so add a new sound, snd_shriek to the Resources. To make the fly hurt Mario, add a Collision event and the three actions to the right: Play sound, Destroy the other and Restart the current room.



Events:
- Create
- obj_mario

Actions:
- Play sound snd_shriek
- Destroy the instance
- Restart the current room

When a game has multiple levels it is tedious for the designer if she has to play every earlier level before she can test the latest. We can create a cheat which will jump to the next level. This will work on the letter N key. Open up obj_Mario and add a Keyboard/Letters/N event. Attach to it a Go to next room action from the main1.tab.

Save the game and test the second level, using the cheat.

The Pits

Platform games are based on the characters jumping, often they have to jump over a pit, falling into it will kill the character.

Create a new spr_pit, loading pit_gif from the network folder. Make an obj_pit, assign the sprite and make sure the object is solid. You can leave it to be visible while you test it, but later will make it not visible. Add a Collision event with obj_Mario with three actions. Play snd_shriek, Destroy the other (Mario) and Restart the level.

Open up the objects tab of rm_2, and replace two or three of the floor blocks with instances of obj_pit. Move to the tiles tab and overlay the spikes from the tile set.

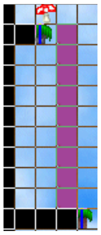
Now the good bit. Play the game and make Mario commit suicide.

Climbing the Vine

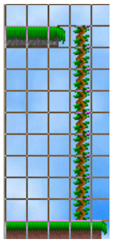
Platform games often include a ladder or something similar which the character can climb. The ladder will be a thin, vertical column of blocks which a player cannot see, it will be overlain in this game with a vine from the tile set. When the character comes into contact with the object, the character's sprite will change and the up and down keys will move the character vertically rather than by jumping.

Create a new sprite spr_vine_marker using vine_marker.gif, turn off Precise collision checking. Assign the sprite to a new object, obj_vine_marker, not solid, not visible, which does not need any events. Place some instances of the object in one of the rooms.

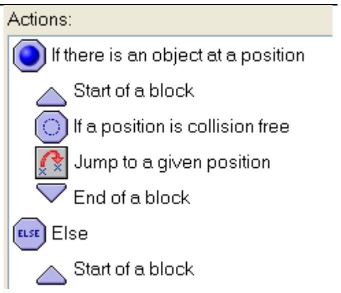
This is the view of the instances.



This is the view in the tiles tab where the magnifying glass icon 🔍 has been selected and Show objects turned off.



The up key event for the Mario object can now be modified by adding these actions at the top.
First check to see if there is an obj_vine_marker 0,0 relative.
If there is, then check for a collision at 0, minus 3, relative, only solid.
Jump to 0, minus 3, relative.
End one block and after an else start another which contains the actions already there. Don't forget to end the second block.

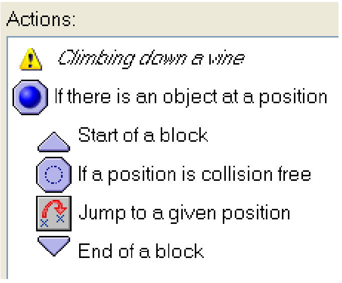


Actions:
If there is an object at a position
Start of a block
If a position is collision free
Jump to a given position
End of a block
Else
Start of a block

We should let Mario climb down ladders as well.

Add a Keyboard/Down event with the same actions.

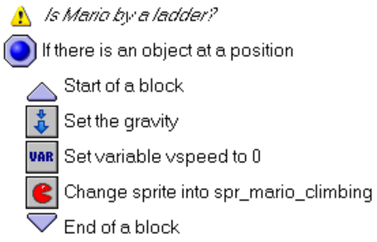
The values are the same except that negative 3 is positive 3.



Actions:
Climbing down a vine
If there is an object at a position
Start of a block
If a position is collision free
Jump to a given position
End of a block

You might think this would be enough, but we need to allow for when Mario first comes into contact with the vine.

This happens during the Step event, so open that up and add these actions at the bottom.
Object at position: obj_vine_marker, 0,0, relative.
Set the gravity to 0 in direction 270.
Set variable vspeed to 0, (so he doesn't fall)
Change sprite into spr_mario_climbing.



Is Mario by a ladder?
If there is an object at a position
Start of a block
Set the gravity
Set variable vspeed to 0
Change sprite into spr_mario_climbing
End of a block

Save and play the game and test you know how to make a character climb.

Now for Mario's goal, attainment of which ends the game.

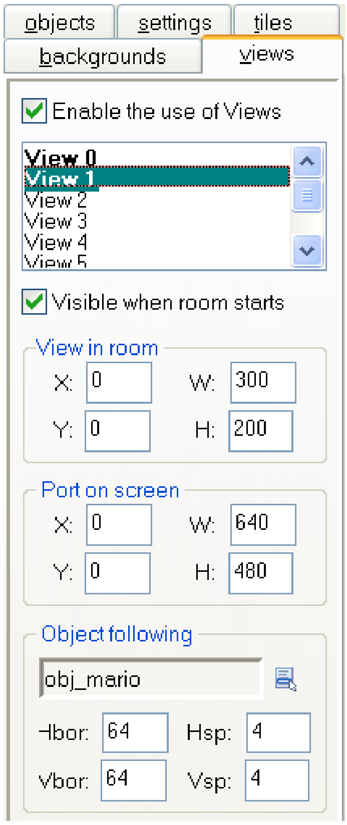
To Rescue a Princess

Level 1 ends with a pizza slice but nothing can beat saving a Princess.
Level 2, and currently the game, will end when Mario rescues the Princess (her original name was Princess Toadstool but now she is the Princess known as Peach).
Create a new sound, snd_fanfare and load fanfare.mp3.
Create a new sprite spr_princess and load princess.gif.
Create a new obj_princess, assign the sprite and give her a collision event with Mario. There is no joke action so we will give the event these two actions: Play a sound (snd_fanfare), Go to next room.
We will make the final room shortly.

The resources list is becoming quite long. The term used in Game Maker is not folder but group. Right click on Sprites and choose Create Group. Name the group Mario and drag into it all his sprites.

Views

So far in our games we have always let the player see the whole of a room. Game Maker allows the designer to choose to show only part of a room. This may be because the room is very large. Or it may be to restrict vision and make discovery lead to a more challenging game. Here we will use a view for the second reason but you are learning the technique because it is such a useful one in 2D game design. It is very common, for example, in sideways scrollers, where the room is very wide. Incidentally, using a digital photograph as a background can give a really nice effect.



☑ Enable the use of Views

View 0
View 1
View 2
View 3
View 4
View 5

☑ Visible when room starts

View in room
X: 0 W: 300
Y: 0 H: 200

Port on screen
X: 0 W: 640
Y: 0 H: 480

Object following
obj_mario
Hbor: 64 Hsp: 4
Vbor: 64 Vsp: 4

Open up the Properties sheet for rm_2. Select the views tab.
Check the box Enable the use of Views.
Select View 1.
Check Visible when room starts.
Set the view in the room to be W:300 H:200.
The view will change as Mario moves so set Object following to obj_mario.
We do not want the character to get too close to the edge of the screen so set Hbor and VBor to 64. These determine how close the character gets to the border of the view window before it scrolls.
In order for the view motion to be smooth, set Hsp and Vsp to 4.

It’s that easy.

Save and play the second level to see the effect.

You can play with the Port on screen to see what it does, essentially it zooms in. Here it was left at the original room setting, which almost doubles the magnification. There is a related setting in Global Game Settings/Graphics. You can set the image to scale if you want.

Back to the Beginning

Now we have completed both levels, we can make the front end, what the player will see first. This will consist of a screen, shown on the next page, with five buttons from which to choose.

Create these seven sprites using the images from the network folder:

spr_start_button
spr_load_button
spr_help_button
spr_scores_button
spr_quit_button
spr_title
spr_mario_menu

Create a new sound called snd_background and assign asian_psycho.mp3.

Mario
Start
Load
Help
Scores
Quit

Create a new object called obj_start_button, assign the sprite. Making the object solid seems to make a menu item easier to click on. Add a Mouse, Left Pressed event. Add the Next Room action from the main1 tab.

Create a new object called obj_load_button, assign the sprite and add a Mouse, Left Pressed event. Add the Load Game action from the main2 tab. Leave the filename as the default. F6 will still load a game unless you disable it under Global Game Settings/Other.

Create a new object called obj_help_button, assign the sprite and add a Mouse, Left Pressed event. Add the Show Info action from main2 tab. F1 will still work as Help. This will display the Game Information.

Create a new object called obj_scores_button, assign the sprite and add a Mouse, Left Pressed event. Add the Show Highscore action from the scores tab.

Create a new object called obj_quit_button, assign the sprite and add a Mouse, Left Pressed event. Add the End Game action from the main2 tab. Pressing the Esc key will still end a game unless you choose to disable it under Global Game Settings/Other.

Create a new object called obj_title, assign the sprite.

Create a new object called obj_menu_mario, assign the sprite.

Create a new object called obj_title_controller and give it a Create event which has an action to set the score to zero. Add an Other/Game Start event with a Play sound action from the main1 tab. Select snd_background and set Loop to true. This sound will clash with the fanfare when Mario rescues the Princess, so open up her Collision event and add a Stop Sound action before the Play Sound (snd_fanfare).

Create a new room, name it rm_start using the settings tab. Give it a white background and then place the five buttons and three other objects on it. I found it easiest to set the Snap to grid to 0,0 when doing this. Remember that if you hold down the Control key when you left click and hold then you can move an object around.

When the player clicks on Help (or F1) they will see the Game Information. Open that in the resources list and enter some information.

Save the game and then run it, test the three buttons which should work initially and listen for the music.

We have the front-end, now for the …. completion screen. Create a new sprite, spr_congrats, loading congrats.gif. Assign the sprite to a new object, obj_congrats.

Create a new room, rm_congrats (make sure the four rooms are vertically in the order you want them to appear). Give it an appropriate caption on the settings tab and on the objects tab add obj_congrats. We can reuse two of the buttons from the starting screen. Add the scores and quit buttons (careful to turn off Delete underlying). However we cannot reuse the start button. Create a new sprite spr_restart. On the object for it give just one Mouse/left pressed event with a Restart game action. Add the object to the room.

My hero!
How can I thank you?
Re-start
Scores
Quit

Save the room and the game, then give it a final test.

Challenges

This is a large complete game, of the size you will be making for your major assignment in Term 2, but that sprite mario_climbing is dreadful, what about improving it?

Making a mistake is terminal. Game play can be prolonged if a character is given lives or health. Add one or other to the game.

Ramps are quite common in platform games. Going down a ramp is automatic because of the falling motion, but going up needs to be arranged. The left and right arrow keys need modification. We need to test whether a position 8 pixels higher is collision free, if it is then the character is moved there and shifted to the contact position.

See if you can program this and place a ramp in a room to test it.

* * * * * * * * * *

Mario and Luigi as they first appeared together.

Musk Cow, a pre-production sketch by Todd Millias

Code and Scripts

Game Maker is so easy to use and popular because it has a drag and drop interface. Events and actions are a readily understandable way to make stuff happen. Adding a collision event between a bat and a ball and setting the ball to bounce is a no brainer, there has been no need to become a nerd, it's been a programmer free zone. Along the way though we have had to pickup some programming terms. For example, that a variable is a container whose contents can vary. If we appreciate that, we can use a Set Variable action and add to a score. Did you suspect it wasn't going to stay that way for ever? One of the objectives of this unit is to introduce you to coding concepts.

If we want to extend what we can do with Game Maker we need to get behind the drag and drop interface. When we were dragging and dropping Game Maker was writing the code for us. Every action has code behind it. These are three common actions and their code:

Icon	*Action Name*	*Code*
	Destroy Instance	instance_destroy()

	Play a Sound	sound_play(sound); //plays sound once.
		sound_loop(sound); //loops sound.

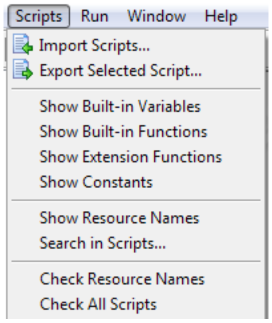
	Go to Next Room	room_goto_next();

This code is written in a programming language created by Mark Overmars called GML (Game Maker Language). We can write in this language ourselves and use it in two ways: as a code snippet or as a script. A script is usually longer and its advantage is it can be called over and over, so we only have to write it once and can use it many times.

On the Control tab of the Action menu there are three icons to do with code:

Icon	*Action Name*
	Execute Code
	Execute Script
⚠	Comment

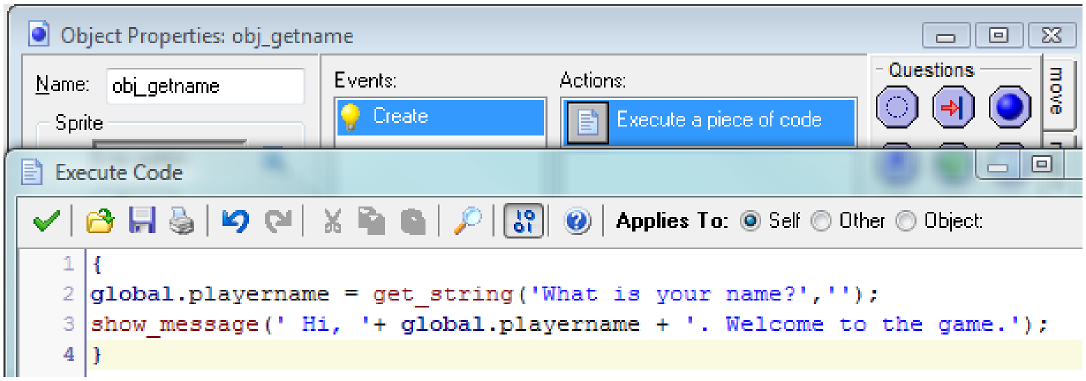
There is also an icon in the Toolbar to create a new script resource and a drop down menu for dealing with scripts.



When we start programming we have extra power. In Game Maker there are under 200 actions to drag and drop but almost 1 000 functions (the lines of code which perform some task, for example there is a function to generate a random number).

In this tutorial we will write some code and execute it directly and also create some scripts we can use more than once.

Start a new game using Advanced mode. Add a new object called obj_getname without any sprite. We will write some code which will prompt the users for their names and then welcome them. Introduce a Create event and attach an Execute Code action. In the dialog box which appears type the code you can see below:



Name: obj_getname

Events: Create

Actions: Execute a piece of code

Applies To: ● Self ○ Other ○ Object:

```
{
global.playername = get_string('What is your name?','');
show_message(' Hi, '+ global.playername + '. Welcome to the game.');
}
```

Click on the green tick to save the code.
Add a room and place an instance of obj_getname in it.
Run the game to see what you have done.

Let’s examine what you typed:

{ } You need curly brackets to indicate the start and end of the block of code. Brackets ain’t brackets. Different shape brackets mean different things.

; The semi-colon indicates the end of a **statement**, a command. This block of code contains two statements.

Colour coding Game Maker uses colour when it recognises something. Comments turn green, variables light blue and functions red. Here it knows two built-in functions. Get_string is a function that fetches some text, by creating a message box. Show_message causes another message box to pop up.

global. We met this before when we were keeping track of scores through different rooms. It means the variable is available everywhere in the game. Notice it is followed by a dot.
The alternative is to use a local variable, that is one restricted to this block of code. If we had wanted to do that we would have started the commands with *var playername;*
Local variables use less memory and speed things up but here we used a global variable as we want to use the name again later.

playername Is a name I made up to be descriptive, it’s a variable. We don’t know in advance what the player of the game will type, its **value** can change. There are built-in variables like *speed* and *direction*; we have just created a user defined variable.

Variable names can only consist of letters, numbers and the underscore character. They cannot start with a number nor contain a space. Never give a variable a name of another resource, like a sprite or an object.

Variable names are case sensitive.

( ) These brackets enclose the **arguments** we are passing to the functions. A function is a process for doing something; the arguments are the raw materials it is going to use to do it with. Think of a function as a machine, like a bread maker, while the argument given to it is the dough.

= One equals sign does not mean “equal to” in GML (or in many other programming languages). For that you would need two. For example, if x is the same as 1 and y is the same as 3, then this would be true:

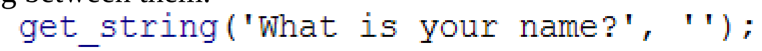
x + y == 4

In GML a single equals sign means a variable is being **assigned** a value, that is something is being put in the container. Playername is being assigned the value returned by the function get_string.

get_string needs two arguments. The message to display and the value to put in the text box initially. Notice that because these are text (not numbers) they go in single inverted commas and are separated by commas. What happens if you leave out a required argument? Find out by deleting the second argument (the comma and ‘Anon’). Save and run the game.

Game Maker warned you twice there was a problem. In the code window the red bar over the line number was the first warning. The second is the run-time error message. It does pinpoint exactly where the error is and what is wrong.

If a function is not given what it needs to work, it causes an error. Add an empty string instead, that is replace the fdsgdsfgcomma followed by single inverted commas with nothing between them.



```
get_string('What is your name?', '');
```

Save and run the game.

The function is fine as long as it has the number of arguments it needs, it doesn’t care what is in the second string (it could be an empty string), it just needed two arguments.

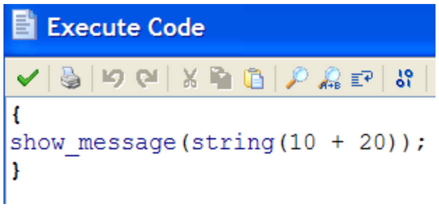
show_message This function only needs one argument but we need to assemble the message from three parts as we want the name entered to go in the middle. Notice how the two text strings go inside single inverted commas whereas the variable does not. The three parts are stuck together with plus signs. If you want spaces and punctuation in the message, you have to type them inside the strings. So the third part starts with a full stop and a space as we want them, the code does not care.

What you have just achieved in prompting for a name, using it and storing it for use later can only be done by writing code, you could not have done it by using drag and drop.

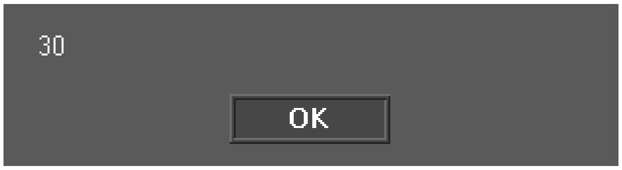
Writing Scripts

Code blocks like this work fine if you only execute them once. Sometimes you may find you use some code over and over with different objects. Then it is better to place the code into a separate script which you can **call** any number of times. You don't have to keep re-typing it and if there is a change, you only need make it once.

Remember that many functions need arguments to work. An argument is a raw material a function needs. For example if we had this simple code block it would produce the result next to it:



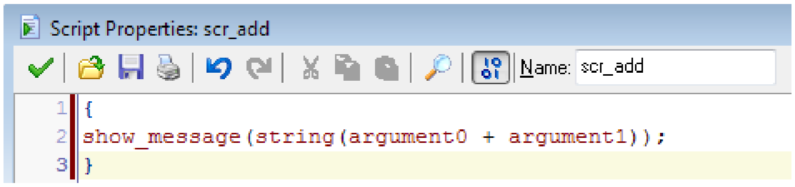
```
{
show_message(string(10 + 20));
}
```



```
30
```

The statement produces a message box which shows the result when you add 10 to 20. The numbers 10 and 20 are the arguments.

In Game Maker, click on the Create a Script icon in the tool bar and type this, naming it scr_add:

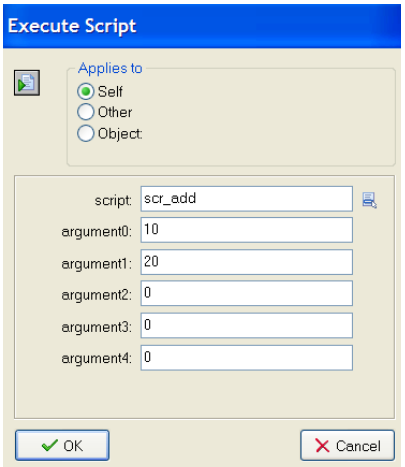


Name: scr_add

```
{
show_message(string(argument0 + argument1));
}
```

When we call a script we can **pass any arguments** needed. This allows us to vary the values of the arguments (sometimes called parameters) we pass at any particular time. Notice next to the Name box is an icon to check the code. This checks for syntactical errors. Unfortunately it will not tell you if a script will do what you want!

To the Create event of obj_getname after the Execute Code action add an Execute Script action .
This will prompt a dialog box, shown opposite.



● Self
○ Other
○ Object

script: scr_add
argument0: 10
argument1: 20
argument2: 0
argument3: 0
argument4: 0

Select the script name from the drop down and enter the value of the arguments.

Notice how the first argument is argument0, the second is argument1 and so on. Computers start counting at 0 not 1.

You could run the same script somewhere else and pass different values.

Click on OK and run the game. After the name prompting, you will see the message box showing the result of the addition, 30.

Since one script can call another, it is better to write a number of small scripts which each do something specialised rather than have one big script. It is easier to swap out or modify a part. If we wanted to call scr_add from within another script, passing the arguments 10 and 20, we would type:

scr_add(10, 20)

To call a script we say its name, the round brackets contain any arguments, separated by commas. We could call the script in another place and pass different arguments:

scr_add(40+50)

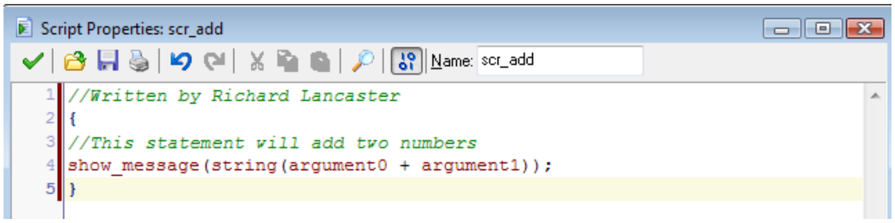
In this example the order of the arguments does not matter as you are adding numbers but usually you must be careful of their order (for example, if you were subtracting). The number of the arguments does make a difference. If you wrote:

scr_add(10, 20, 50)

The result would be 30, not 80 since there is no argument2 in the script.

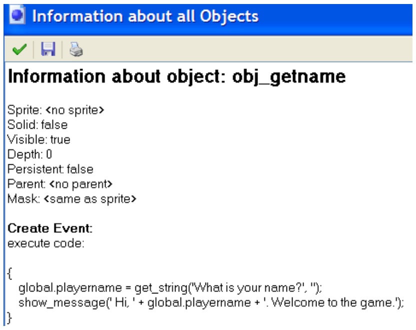
A script can call itself but you should be very careful of this. It is called recursion and the danger is the computer can get stuck in an infinite loop and never stops going around in a circle.

You can place a comment in a code block by typing a double back-slash first. Everything else on that line will be ignored by the computer except for turning it green.



Name: scr_add

```
//Written by Richard Lancaster
{
//This statement will add two numbers
show_message(string(argument0 + argument1));
}
```



Information about object: obj_getname

Sprite: <no sprite>
Solid: false
Visible: true
Depth: 0
Persistent: false
Parent: <no parent>
Mask: <same as sprite>

Create Event:
execute code:

```
{
  global.playername = get_string('What is your name?', '');
  show_message(' Hi, ' + global.playername + '. Welcome to the game.');
}
```

It sometimes helps to see a summary of the properties and the code, a summary which can be printed out or saved as a file for examination or archiving.

Drop down the Edit menu and click on Show Object Information to see a list like this.

Now you know enough about writing blocks of code to make a game which requires scripting to work.

Creating a Game using Scripting

Play the game Code and scripts. It is very simple, the object is to move Frankie so he avoids the bats. But the technique behind it is one very commonly used in games, especially in first person shooters where enemies come towards the protagonist.

The idea is to give a sense of depth by having objects appear bigger the closer they are to us. We could get this 3D effect by having a lot of versions of the same image, at different sizes. For example, a road with cars getting smaller the further they are away. This takes quite a lot of memory so instead we can **scale sprites** – alter the apparent size depending on how far away they are.

Remove the Execute Script action as we do not need a message box showing numbers in our game but leave the Execute Code action as we will still want to ask for a player's name. The game will start with a message box. When a message box is open, everything else is paused so the room will not fully load or the game begin until the box is closed.

Creating the Environment

Create these three sprites using the resources in the folder.

spr_bat
spr_skyline
spr_frankie_right

Assign each sprite to an object: obj_bat, obj_skyline, obj_frankie.
Modify the room, first by resizing it on the Settings tab so it is 1260 by 480. Add an appropriate caption while you are there. Then place in the top left corner of the room using the Objects tab an instance of obj_skyline. Use the Backgrounds tab to turn the background color to black. Now it is a suitable environment in which to place an instance of obj_frankie. Put him in the middle of the room at the bottom. Obj_bat does not need placing in the room. There will be a controller object to do this.

Frankie

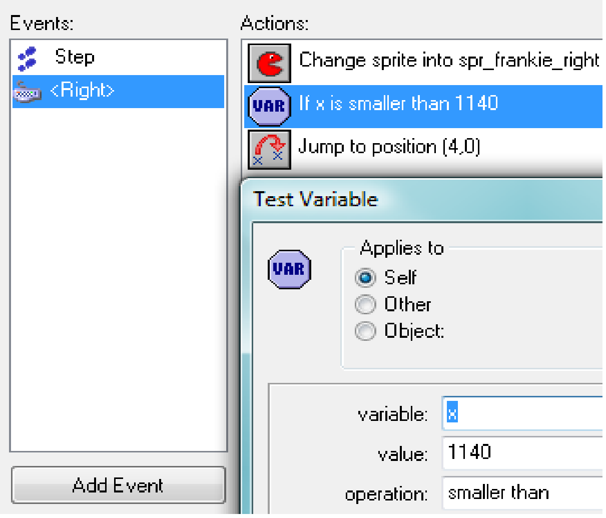
He will get a point at each Step, if he survives, and the player will be able to move him right and left with the arrow keys. We want Frankie to be facing the way he is shambling so we will have one object but change the sprites. Right click on spr_frankie_right and duplicate the sprite, call the copy spr_frankie_left. Click on the Edit button, drop down the Transform menu and select Mirror/Flip, the defaults are what we want. Say OK, then save and close.

Open up obj_frankie and add a Step/Step event with an action of Set Score, with a value of 1, relative. There ware 30 steps in a second, so Frankie earns 30 seconds for every second he survives.

Add a Keyboard/Right event. We need three actions. The first will change the sprite. Add a Change Sprite action from the main1 tab, set it to spr_frankie_right, sub-image -1 and speed 1.

The second action is Test Variable on the control tab. We want to make sure that Frankie cannot get too close to the right hand edge of the room (the x dimension). We know the room is 1260, so if we subtract, say 120 pixels, we won't let him get closer than 1140.

If he is not too close, then he can move right. The third action is to Jump to a Position, with values X:4, Y:0, relative.



Change sprite into spr_frankie_right

If x is smaller than 1140

Jump to position (4,0)

Applies to

● Self

○ Other

○ Object:

variable: x

value: 1140

operation: smaller than

Now for the other way, and again we will ensure Frankie cannot get closer than 120 pixels to the edge.
Add a Keyboard/Left event with a Change Sprite action (spr_frankie_left, -1, 1), then a Test Variable action. Set it to x larger than 120. Then add a Jump to Position action: X:-4, Y:0, relative.
Finally add a Keyboard/NoKey event with a Change Sprite action, choose either Frankie but set the subimage to 0 (this will ensure the animation does not play)
Play the game to see that Frankie and the scoring are working properly.

The View

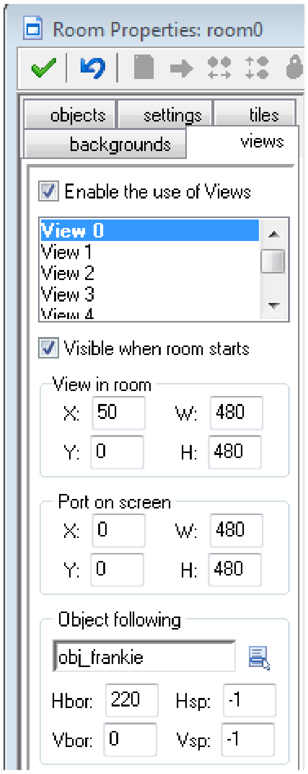
The room is large to allow Frankie to move sideways but we do not want the player to see all of it at once. We will use a view to do this. This makes the window smaller but able to scoll.

Open up the Properties window of the room and on the Views tab set it so that it has these settings.

You will notice that the height of the view and of the port are the same as the room itself – vertically the slice is all visible. Horizontally what is visible is more restricted.

When you have finished entering the settings you will notice a rectangle has been drawn to show the opening view (it is easier to see if you turn the grid off). If Frankie is not in the area, move him in.

Save and run the game to test the appearance and that Frankie moves left and right as far as he is allowed while the view follows him.



☑ Enable the use of Views

☑ Visible when room starts

View in room

X:	50	W:	480
Y:	0	H:	480

Port on screen

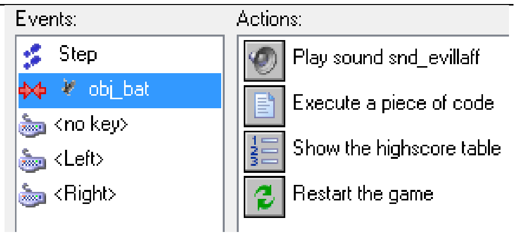
X:	0	W:	480
Y:	0	H:	480

Object following: obj_frankie

Hbor:	220	Hsp:	-1
Vbor:	0	Vsp:	-1

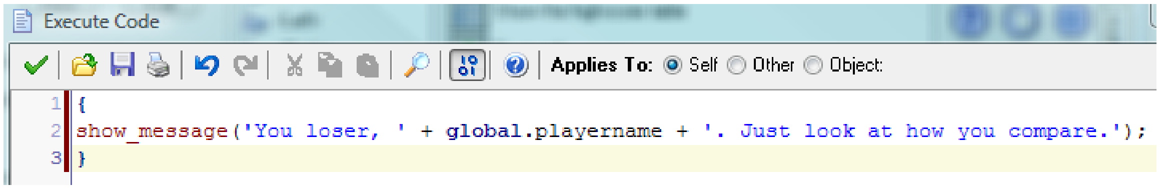
What happens when a bat strikes? It's all over for Frankie. Let's add the sounds first to our resources. Load sounds snd_evillaff and snd_psycho. One is a wav, the other a midi. Game Maker can handle both file types.

Now to obj_frankie add a Collision event with obj_bat. Add the actions in the graphic to the right.
The code will display a message and it can be personalised as we have already asked for the player's name.



Events:
- Step
- obj_bat
- <no key>
- <Left>
- <Right>

Actions:
- Play sound snd_evillaff
- Execute a piece of code
- Show the highscore table
- Restart the game



Applies To: ● Self ○ Other ○ Object:

```
{
show_message('You loser, ' + global.playername + '. Just look at how you compare.');
}
```

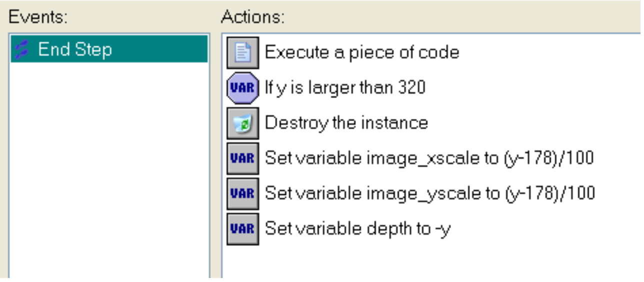
The Bats

We only need one event with obj_bat, the End Step event. But the actions need some explanation. This is the core of the game mechanics and we are going to be clever with our coding (which means we use someone else's code for the hard bit!)

There are three aspects we have to deal with: the size, the position on the screen, and the depth. We want the bats to be visually realistic; meaning as they get "closer" they become bigger, move down the screen, seem to fly faster and diverge from our eyepoint. Oh, and the newer ones must be behind those created earlier. It takes a lot to simulate 3D reality using a 2D computer monitor.

Bats will be created at y-position 180. In the End Step event we scale the sprite based on its y-position. For this we can change the variables image_xscale and image_yscale. We use a scale factor that is close to 0 when the y-position is 180 and increases when the bat moves down (towards us). To be precise, we use a scale value of (y-178)/100. You can play a bit with the scale factor if you like (make sure it is always positive).

The second thing to deal with is the depth. In the End Step event we set the depth to –y. This means that bats closer to the bottom of the screen (with larger y-coordinate and closer to the viewer) have a smaller depth and are drawn on top of the others, as we want. We also need to give the player an appropriate depth to keep him at the correct position among the bats.

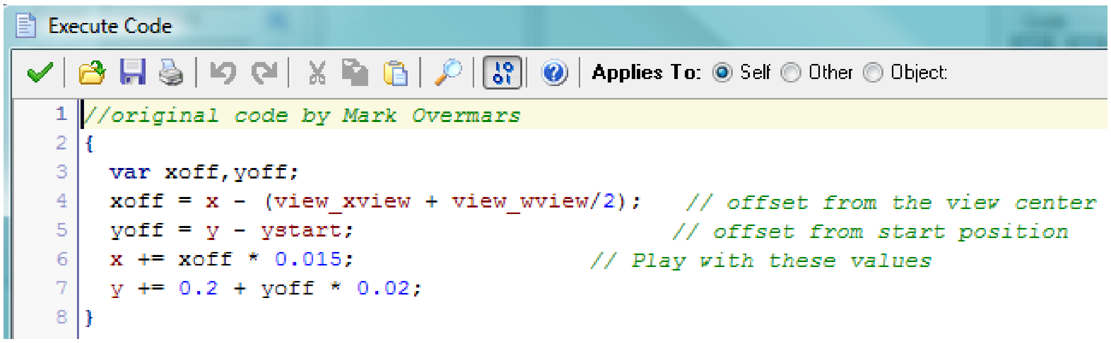


Events:
- End Step

Actions:
- Execute a piece of code
- If y is larger than 320
- Destroy the instance
- Set variable image_xscale to (y-178)/100
- Set variable image_yscale to (y-178)/100
- Set variable depth to -y

Finally we must control the movement. This is harder than it may seem. We can let the bats move toward us with constant speed but, as you will notice, this will visually slow them down (this is because objects close by seem to move faster than objects far away). So we must increase the vertical speed.
Also, objects move towards the sides if they get closer (because parallel lines meet in a point at infinity). So, based on the position of the bat relative to the middle of the view, we must adapt its horizontal speed. We do this using code.

This is the code to enter on the Execute Code action:

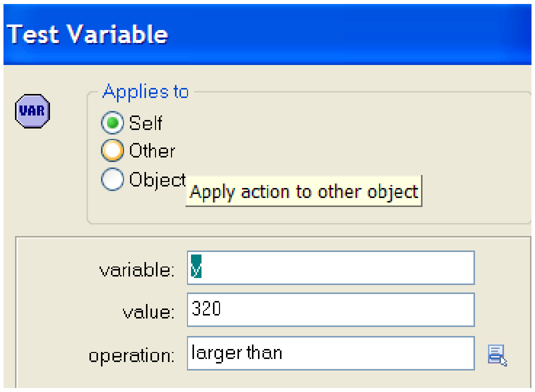


Applies To: ● Self ○ Other ○ Object:

```
//original code by Mark Overmars
{
  var xoff,yoff;
  xoff = x - (view_xview + view_wview/2);   // offset from the view center
  yoff = y - ystart;                        // offset from start position
  x += xoff * 0.015;                  // Play with these values
  y += 0.2 + yoff * 0.02;
}
```

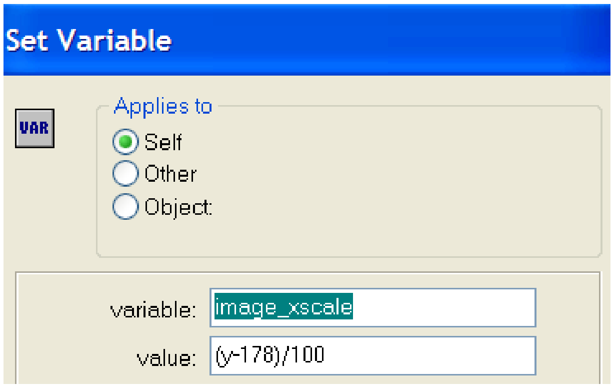
Note the use of local variables xoff and yoff. The numbers have been determined experimentally. You might want to change them a bit.

Below are the settings for the other actions. The Destroy Instance is set to self so a bat will disappear once it gets near the bottom of the screen (that is it appears as though it is passing Frankie).



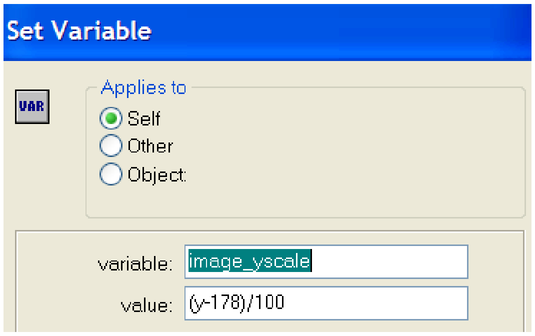
Applies to
● Self
○ Other
○ Object

variable: y
value: 320
operation: larger than



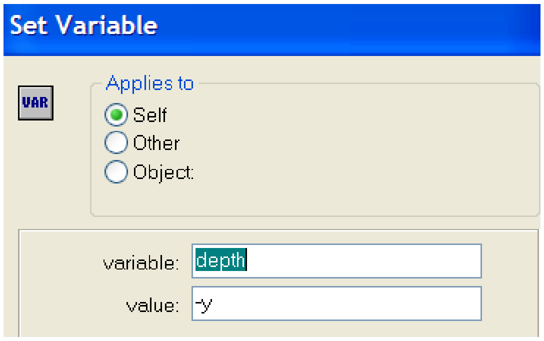
Applies to
● Self
○ Other
○ Object:

variable: image_xscale
value: (y-178)/100



Applies to
● Self
○ Other
○ Object:

variable: image_yscale
value: (y-178)/100



Applies to
● Self
○ Other
○ Object:

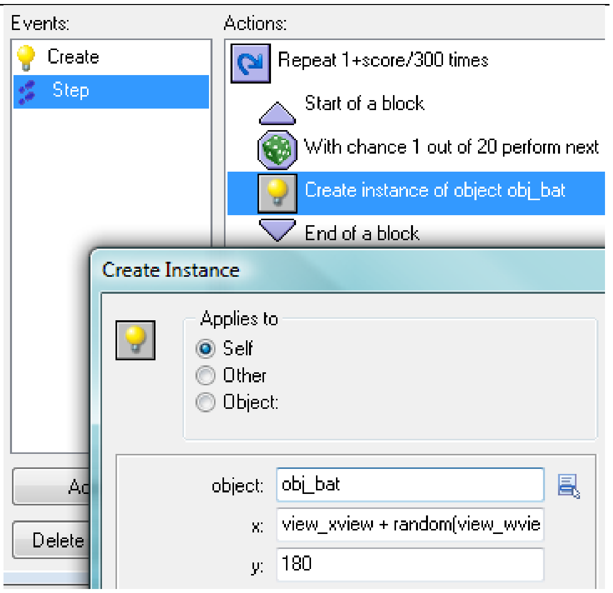
variable: depth
value: -y

The Controller

To make the bats appear in the room we will create an obj_controller, which does not need a sprite.

Add a Create event and play the background music, snd_psycho with looping set to True.

Now add a Step event with the actions shown on the right.
The Repeat action on the Control tab lets us make bats faster as the score mounts (the game gets harder as it progresses).
We will use a value of 20 for the dice but you can alter this to make more or fewer bats.
The Create Instance action will determine where they appear. The cleverness lies in the horizontal location as this must change as the view alters:
view_xview+ random(view_wview)


Events: Create, Step

Actions:
- Repeat 1+score/300 times
- Start of a block
- With chance 1 out of 20 perform next
- Create instance of object obj_bat
- End of a block

Create Instance

Applies to: ● Self ○ Other ○ Object:

object: obj_bat
x: view_xview + random(view_wvie
y: 180

That’s it. Place an instance of obj_controller in the room, then run the game and test it. Hitting the Esc key will help you exit if it is looping. If the game is over before it begins every time, try opening up both sprites for Frankie and ticking Separate collision masks.

Let’s make the highscore table looks more customised. In the network folder for the game is a font (bloody.ttf) and a background (highscore.gif) which can be used with the high score table. The font needs to be copied to C:\Windows\Fonts. When an executable file is created in Game Maker, the font will be packaged with it, but for it to be usable while developing the game it must be installed with all the other fonts. Create a new background using highscore gif. Then open obj_frankie, the collision with the bat event, and modify the highscore table action.

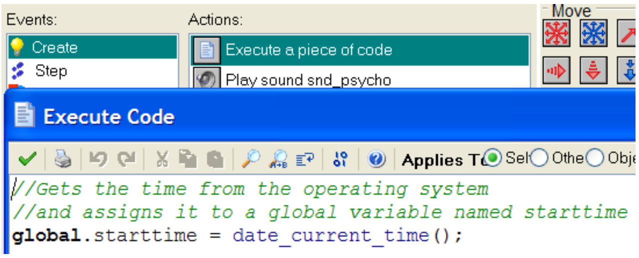
Save the game, then run it to test it.

A Timer

As the game is about how long you can get the character to survive, elapsed time is important. We will create a timer which will show the passage of time.

Open obj_controller and to the Create event add an Execute Code action.
Enter this line of code:

global.starttime = date_current_time();


Events: Create, Step

Actions:
- Execute a piece of code
- Play sound snd_psycho

Execute Code

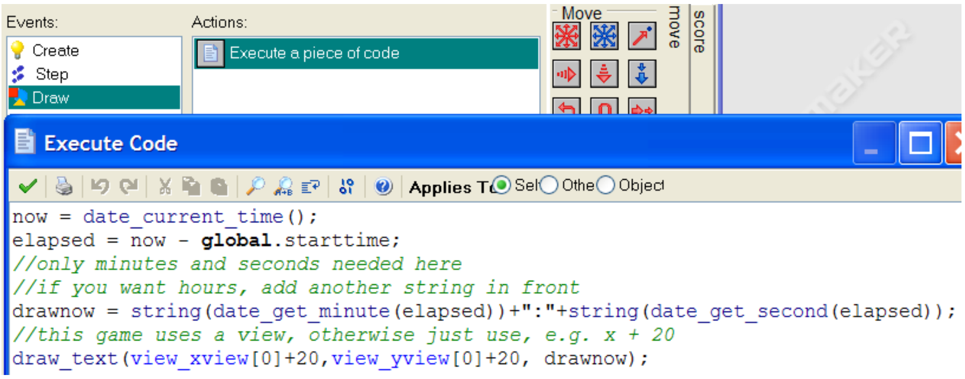
Applies To ● Sel ○ Othe ○ Obj

```
//Gets the time from the operating system
//and assigns it to a global variable named starttime
global.starttime = date_current_time();
```

This creates a user-defined variable called starttime, which is available globally, and assigns it the value of the current time using a built-in function which asks the Window operating system for when it is.

Add a Draw event and an Execute Code action. Add these lines:

now = date_current_time();
elapsed = now - global.starttime;
drawnow = string(date_get_minute(elapsed))+":"+string(date_get_second(elapsed));
draw_text(view_xview[0]+20,view_yview[0]+20, drawnow);



```
now = date_current_time();
elapsed = now - global.starttime;
//only minutes and seconds needed here
//if you want hours, add another string in front
drawnow = string(date_get_minute(elapsed))+":"+string(date_get_second(elapsed));
//this game uses a view, otherwise just use, e.g. x + 20
draw_text(view_xview[0]+20,view_yview[0]+20, drawnow);
```

Save the game, run and test it. If you cannot see the timer, check that obj_controller is not just in the room, but is in the view with Frankie. Obj_getname can be anywhere in the room but not the controller.

Challenges

Add an Execute Code action which shows a message box saying “Goodbye, *player’s name*” when the game is finished.

Make drawnow a global variable so you can alter the message when Frankie collides with a bat by adding the elapsed time.

If you want to add the player’s name automatically to the high score table when it is displayed, you would delete the existing Show Highscore action and add the two commands below to the Execute Code action when obj-Frankie collides with the bat.

```
//Next line adds the name and score to the highscore list, if its within the top ten scores
highscore_add(global.playername,score);
//Next line displays the table. The arguments in brackets are:
//-1 means current highscore list,1 is the index number of the background image, 1 shows the border,
// c_red is the font colour of the new score, c_red is the font colour of the other scores (here both are the same)
//fnt_bloody is the font to use, 14 is the font size
highscore_show_ext(-1,1,1,c_red,c_red,fnt_bloody,14);
```

Check out the detailed information on the Game Maker Language in the Game Maker program itself under Help/Contents. For example, the code above could have been worked out from typing the keyword *highscore* in the Index of Help.

In the network folder is a Word document called Game Maker Documentation. This is a manual for the application written by its creator, Mark Overmars. Much of the manual is concerned with GML.

A good source of scripts is http://www.gmlscripts.com/

* * * * * * * * **

Musk Ox, a pre-production sketch by Todd Millias

3D Effects

The large commercial games on the market today are 3 dimensional. To create them with their complicated lighting, camera movements, collision checking, etc, takes a team of developers many months and requires a budget of millions of dollars. Game Maker does not directly support 3-dimensional game worlds although there are functions for 3-dimensional graphics. We can gain many of the same effects much more modestly if we use a few tricks of perspective.

We will use these:

Objects that lie behind other objects are (partially) invisible

You normally cannot look through solid objects. When for example a character is partially hidden by a rock the viewer immediately knows that the character is behind the rock. In this way the viewer gets a better insight into the 3- dimensional relationship between the objects.

So to make a world look 3-dimensional we should only display the (parts of) objects that are actually visible. This process is normally called **hidden-surface removal**. In the Network icon to the left we know the monitor is in front because it partially obscures the globe and is further down.

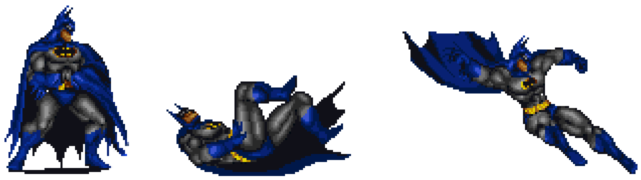
Because Game Maker provides a 2-dimensional world, we have to deal with this ourselves. The way to achieve this is to draw the sprites that correspond to the objects in the correct order. The object that lies furthest way is drawn first. The next object is drawn on top of it, and so on. Because closer objects are drawn on top of objects behind them, they will (partially) hide the other objects. In Game Maker we can indicate this order by setting the **depth** for the different objects. Instances of objects with larger depth are drawn earlier than instances of objects with a smaller depth and, hence, appear behind them.

Objects that are further away appear to move more slowly

An object that lies far away takes longer to traverse from the left to the right through your view, while an object that lies closer moves more quickly through your view. You can use this to give the viewer an indication of the depth of the objects. One particular aspect of this is parallax scrolling that we will make in a game today. Also, objects that move away from you or towards you hardly change position on the screen. So normally the speed in the vertical direction on the screen (which often corresponds to movement towards you or away from you) should be smaller than the horizontal speed. We saw this in the previous game we made by manipulating the bats. In the last game we achieved 3D effects using scaling and by altering the speed

Objects tend to have a shadow

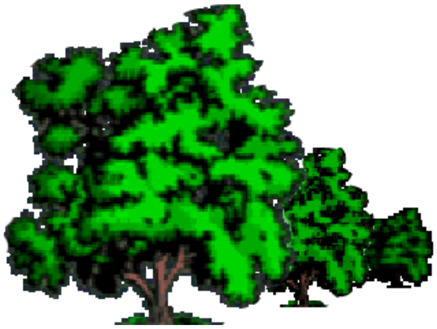
Objects cast shadows. It is difficult to compute the shadows that are cast on other objects but it is relatively easy to have shadows on the floor. And even if there is no precise shadow it is good to have a little dark spot around the bottom of a sprite. This gives the impression that the object stands on the floor. Make sure the shadows are consistent, that is, all objects have shadows (or none) and that they have the same size, colour and are in the same direction.



The two Batmans to the left look like they are on a surface because of their shadows. The semblance of a jump in the nearest is partly because of the absence of a ground shadow.

Objects further away are smaller

A simple way to achieve a sense of distance is to make objects further away seem smaller. For example, this tree has been resized twice.
If the largest image is placed in the foreground, the middle sized image in the middle and the smallest in the distance, we get a sense of depth. Actually all three are of course being placed on a flat screen. That one appears closer than another is a function of size, of hidden surface removal and that the base of the nearest tree is lowest. For each of these trees we create an object. To make sure that the small tree is behind the middle tree which again must be behind the large tree we give the objects different depths. If we move sideways, the “most distant” tree will appear to be obscured by those in front. The nearer we want an object to appear to be, the more we move it down the screen (increasing its y value).



Objects that lie further away appear vaguer

When objects lie further away you cannot see them so well any more. This also works the other way around. When an image appears vague we interpret it as lying further away. Hence, adding a feeling of mist and blurring images that lie further away gives a good sense of distance. This can be achieved by using different sprites for the same instance, depending on its depth. Another technique is to slightly overlap several instances of the sprite, which gives a blurred outline.

Parallel lines appear to meet in a point

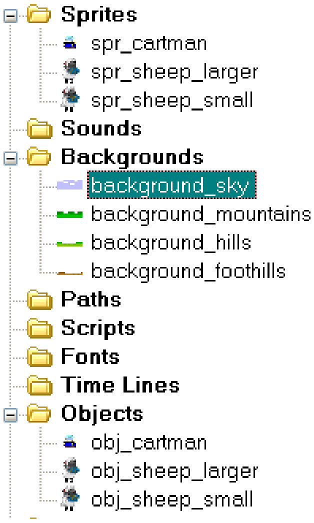
As you probably know, parallel lines in a 3-dimensional world meet in the distance in a point. For example, the two sides of a road meet in the distance. Stated more precisely, our view generates a **perspective projection**. This in some sense deforms the objects depending on their depth but this deformation is exactly what gives the 3-dimensional feeling. In the photograph to the right, the further away the road gets, the narrower it appears, eventually, if we could see far enough, the two sides would seem to meet.

In the last game we used the idea of parallel perspective. This means that objects farther away seem smaller to us and that two receding parallel lines will seem to meet in the distance.

Parallax Scrolling in a Game

Run the executable called parallax.exe and notice as you move left and right with the arrow keys not just that the background moves, but different parts of it move at different speeds. One of the small sheep looks like a baby because of its placement, the other looks further away.

Start a new game in Game Maker and create these resources using the graphics from the Network folder:



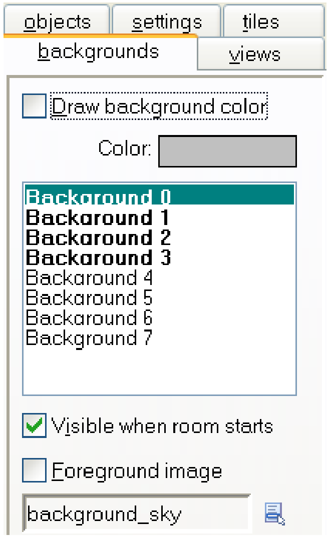
- Sprites
  - spr_cartman
  - spr_sheep_larger
  - spr_sheep_small
- Sounds
- Backgrounds
  - background_sky
  - background_mountains
  - background_hills
  - background_foothills
- Paths
- Scripts
- Fonts
- Time Lines
- Objects
  - obj_cartman
  - obj_sheep_larger
  - obj_sheep_small

Create the four backgrounds in the order shown in the screen capture. The only difference in the four is that background0 (background_sky as you will rename it) does not contain a transparent area and the others do.

We want Cartman to pass in front of the sheep so we will change the depth of both sheep objects to 5. This will give the effect that they are behind him when he moves in front of them.

Notice how the original sheep graphics are pngs and both have a shadow. When brought into Game Maker the shadows are lost and the nature of the conversion to a bit-mapped sprite makes them hard to re-create.

Create a new room,call it rm_game. On the settings tab make the width 1280 to match the backgrounds.
Move to the backgrounds tab, turn off the Draw background color as we do not need it and add each of the four backgrounds in turn.
Background 0 is assigned background_sky.
Background 1 is assigned background_mountains.
Background 2 is assigned background_hills.
Background 3 is assigned background_foothills.



□ Draw background color

Color:

Background 0
Background 1
Background 2
Background 3
Background 4
Background 5
Background 6
Background 7

☑ Visible when room starts

□ Foreground image

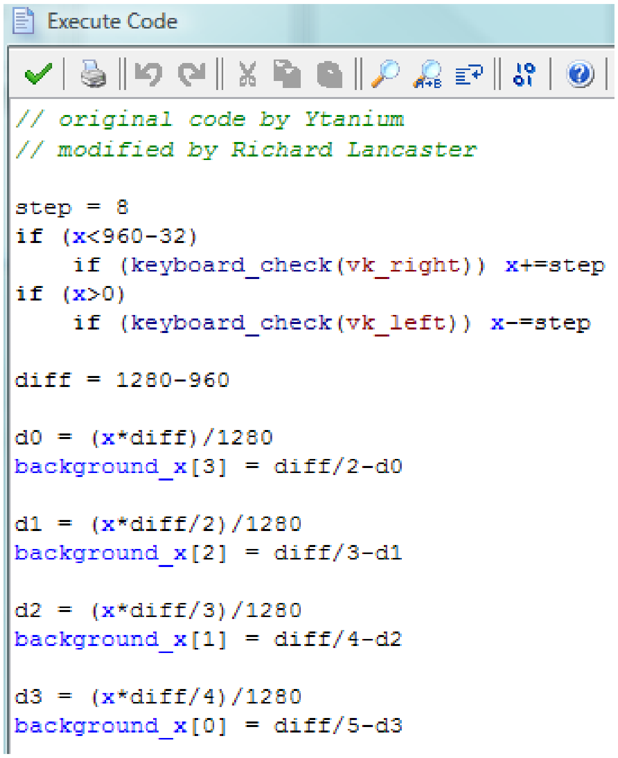
background_sky

Save the game and run it to see it looks alright so far.
If it does not look right, check the transparencies.

The clever stuff is the code to move the backgrounds.
Open obj_cartman and on the Step event place an Execute a piece of code action from the control tab.
The graphic to the right shows the code.
To save you retyping it, the code has been placed in a text file called parallax code.txt

Notice that we will not need to use Keyboard Right and Left arrow events to move Cartman as the code checks for them.

There are four sections to the complete background in the game world (backgrounds0 to 3), each is controlled using its index number in the square brackets and X coordinate.



```
// original code by Ytanium
// modified by Richard Lancaster

step = 8
if (x<960-32)
    if (keyboard_check(vk_right)) x+=step
if (x>0)
    if (keyboard_check(vk_left)) x-=step

diff = 1280-960

d0 = (x*diff)/1280
background_x[3] = diff/2-d0

d1 = (x*diff/2)/1280
background_x[2] = diff/3-d1

d2 = (x*diff/3)/1280
background_x[1] = diff/4-d2

d3 = (x*diff/4)/1280
background_x[0] = diff/5-d3
```

Place some of the sheep and an instance of Cartman in the room.

Save and run the game to test the movement.

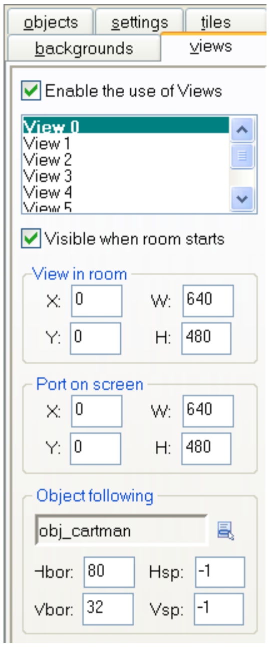
The room is quite wide but we can give even more of a sideways scrolling effect if we use a more restricted view.

On the views tab change the settings so they match those to the right. We will only see half of the width, which makes it a conventional size room, and follow Cartman.

The Hbor has been increased to alter the distance he can get to the right side of the room.

Save and give the game another test run.

It is not really a game as not much happens but we have made a game world which is less flat and static, more 3 dimensional



☑ Enable the use of Views

View 0
View 1
View 2
View 3
View 4
View 5

☑ Visible when room starts

View in room
X: 0 W: 640
Y: 0 H: 480

Port on screen
X: 0 W: 640
Y: 0 H: 480

Object following
obj_cartman
Hbor: 80 Hsp: -1
Vbor: 32 Vsp: -1

Letting the Player Choose a Character

Some games allow the player to choose which character to use. We will give our player a choice of Cartman or Kenny.

Add a spr_kenny and assign it to an obj_kenny. Add a Step event and copy and paste the Execute Code action from Cartman to Kenny.

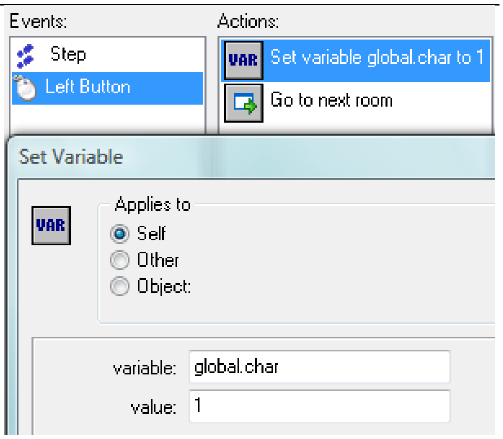
Create a new room, call it rm_choose, and drag it above the existing room in the Resources tree so it will be shown first. Remember that rooms are shown in their order vertically in the Resources tree, working from the top down.
Place one instance each of Cartman and Kenny in the room.

We will show a message box at the beginning of the game telling the player to select a character. On the Settings tab of rm_choose, click on the Creation Code button and add this code, which will be executed when the room is created (just before it is drawn to the screen). The first command will declare a user-defined variable called *char* and initialise it as zero. **global** means the variable will be available to all the rooms.

```
{
global.char = 0;
show_message('Left click on the character you want in the game.');
}
```

Now we will use the left mouse click to set the variable called char we just initialised. Open obj_cartman and add these two actions to the left mouse button.

Do the same for obj_kenny but set the value of the variable global.char to 2.



Events:

- Step
- Left Button

Actions:

- Set variable global.char to 1
- Go to next room

Set Variable

Applies to

● Self
○ Other
○ Object:

variable: global.char

value: 1

In the Creation code of rm_game enter this code which will check the value of the variable called char and draw the appropriate character using the x,y coordinates:

```
// This displays the character determined by the global variable
if global.char=1
{
instance_create(240,400,obj_cartman)//alter the values of x and y to suit
}
if global.char=2
{
instance_create(240,400,obj_kenny)
}
```

Since this code will create either character depending on the value of our globally available variable called char we can remove the instance of obj_cartman from the room.

Save the game, run and test it. Try left clicking on the character in the second room.

Houston, we have two problems. If we left click on a character in the second room, the game crashes because it tries to go to the next room and there isn’t one. This could be solved by putting objects in the first room that look like the characters but are really each the same sprite but a different object. This will also mean the “characters” in the first room do not have any other abilities such as being able to move.

The second problem is that the view is set to follow Cartman and doesn’t follow Kenny. You will have to figure that one out for yourself! Alright, the hint is ‘inheritance”.

Another idea, let’s change the cursor. To change a cursor we only need to do three things:

- ✶ load the image to be used as the cursor
- ✶ type one line of code
- ✶ turn off the default cursor

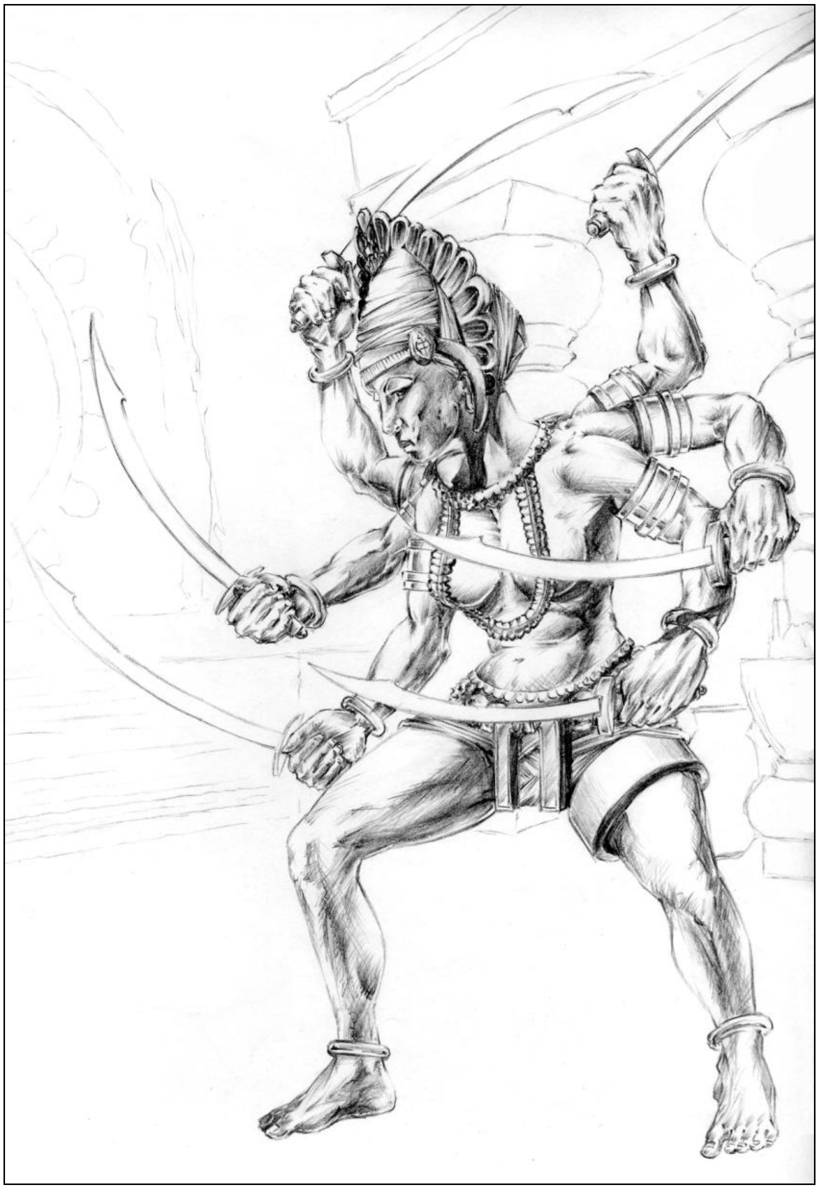
1. Create a new sprite called spr_heart using heart.jpg.

2. In the Creation Code of rm_choose (on the Settings tab), add another command to the code in between the two already there:

```
{
global.char = 0;
cursor_sprite=spr_heart;
show_message('Left click on the character you want in the game.');
}
```

3. Turn off the standard Windows cursor using in the Resource tree: Global Game Settings/Graphics.

Save, run and give it a final test.



Character sketch in pencil by Todd Millias

Layered Backgrounds

To finish off this session we will quickly create using a different technique another environment where the backgrounds move, which is easier to make but not so subtle.

It is possible to apply more than one background to a room and to give them different speeds. If you make them transparent so one can be seen through the other, then the eye sees the things on the fastest moving one as the closest and the things on the slowest moving one as furthest away. Think of looking out of the window of a moving vehicle, objects further away move more slowly across your vision. We will use this parallax effect to give the illusion of 3D depth.

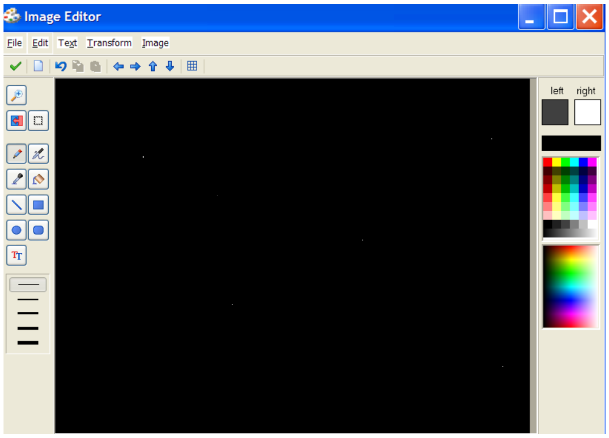
We will make a starfield, a space ship will appear to move across it and the further away stars will appear dimmer and move more slowly. Check out Layered backgrounds in the network folder.

Start a new game. Save it as Moving Backgrounds.

Create a new background, leave its name as background0. Click on the button to Edit Background. First we will make it the same size as a standard room. Drop down the Transform menu, choose Resize Canvas, turn off Keep Aspect Ratio, and set it to be 640 wide by 480 high. Click OK.

Select the eye dropper tool. Select black from the colour swatch on the right and then change to the paint bucket tool and fill the image with black.

Change the drawing colour to white and with the pencil drawing tool place just four or five white dots anywhere on the image. These represent the nearest stars.



Since the image size is the default room size there will be four or five close stars. Click on the green tick to close the Image Editor. This is the first layer, the stars are brightest and biggest, they will move the quickest. The combination of all three makes the viewer see them as closest.

We will make two more backgrounds, called background1 and background2. Both will be the same size 640 by 480, but there are two differences. In background1 place six or seven stars using a light grey, not white. Leave the default transparency as we want background0 (which is technically underneath) to show through and its black will provide the backdrop.

In background2 place seven or eight stars using a darker grey and again tick transparent.

Now we are ready to place them in a room, so create a new room.

On the backgrounds tab of the Room Properties turn off the background colour as we have our own. Choose Background 0 and tick Visible when room starts.

Click on the blue icon and choose background0.

This background represents the closest stars so we want them to move the quickest. Set the Vert Speed to 4.

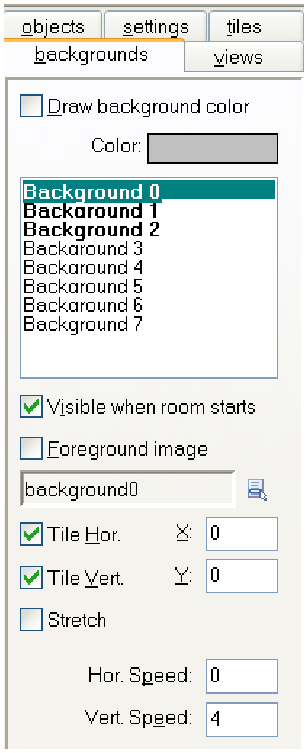
Add the other two backgrounds using the same settings except for the vertical speeds and background names.

We want them to scroll more slowly as they are further away.

Set the vertical speed for Background 1 to 3.

Set the vertical speed for Background 2 to 1.

You can adjust these speeds and the number of stars later if you wish.



□ Draw background color

Color:

Background 0
Background 1
Background 2
Background 3
Background 4
Background 5
Background 6
Background 7

☑ Visible when room starts

□ Foreground image

background0

☑ Tile Hor. X: 0

☑ Tile Vert. Y: 0

□ Stretch

Hor. Speed: 0

Vert. Speed: 4

Save the game and run it.

Fine tune the appearance if you wish.

What is space without a spaceship?

Create a new sprite called spr_spaceship and load the graphic.
Make a new object, obj_spaceship, and place an instance in the bottom of the room.

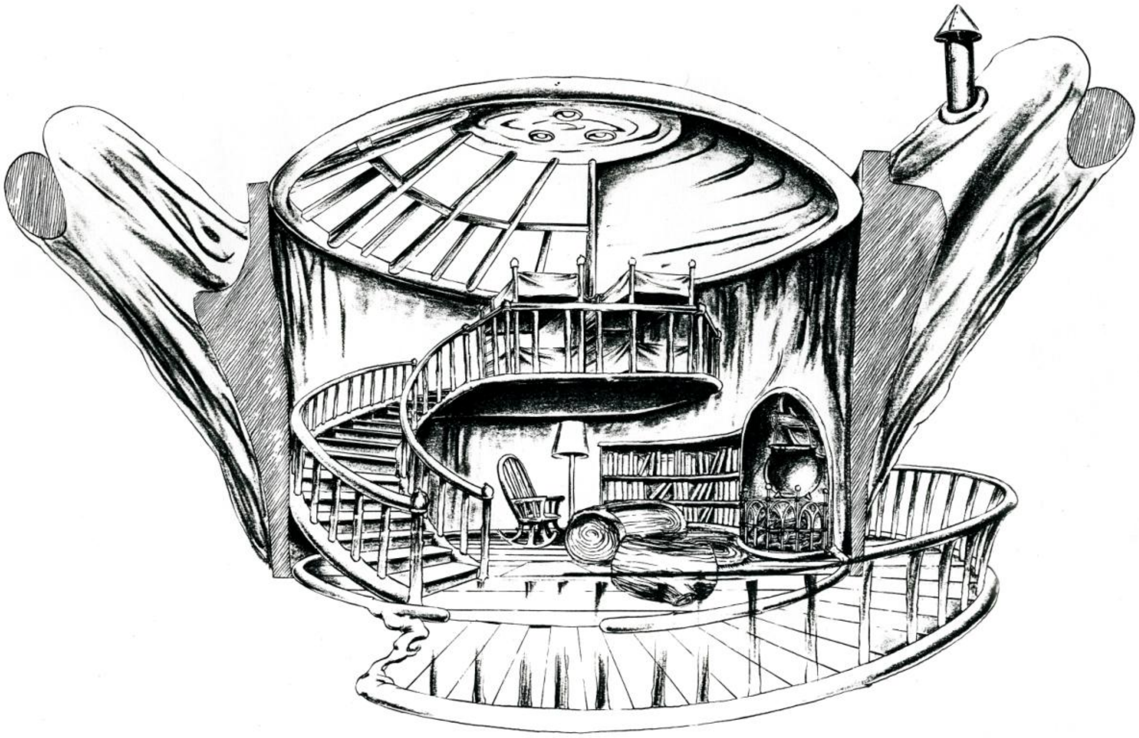
Save and run the game.

You will appreciate the limitation of this technique.

The backgrounds are always going to be moving, regardless of what the central protagonist is doing. The moving backgrounds using code is a more subtle effect. This allows us to be more sophisticated and the backgrounds will only move when the character moves. However, it is useful to make clouds a scrolling background as they do move independently.

Challenges

If you wish to make some game play, the following resources have been placed in the network account: ufo.gif, missile.gif, explosion.gif and explosion_small.wav.
You can make the spaceship move left and right with jump to position actions.
You will need to alter the spaceship's point of origin to be realistic with the missile.
The trick with the explosion object is to add an animation end event with a destroy self action (this will mean the animation of the explosion only plays once then disappears).



Sketch for a level by Todd Millias

Parallel Projection in an RPG Game

Even though true 3-dimensional games should use perspective projection we can use parallel projection to create a feeling of depth. In a parallel projection the size of the objects stays the same independent of the distance. Also parallel lines remain parallel. This makes it a lot easier to use sprites because sprites have a fixed size (although they can be scaled). Clearly, a parallel projection gives a distortion of the view. To limit this distortion we will make sure that the player cannot see objects that are far away. So she will not really notice the errors. We achieve this by looking down on the world at an angle (normally 45 degrees).

We will practise this technique with a small role playing game (RPG) in which the player controls a character who can walk through a world with some trees. Play the game called RPG parallel.exe in the Network folder.

There are three things we must handle to create the environment in such a game. We must make sure that hidden-surface removal is done correctly. So when the character is behind a tree it should be drawn before the tree is drawn to make it partially invisible. Secondly, we have to correctly deal with the speed of the character. Moving back and forth (away and towards us from our view point) must be slower than moving left and right. Finally, and the most difficult, we must handle the collisions correctly.

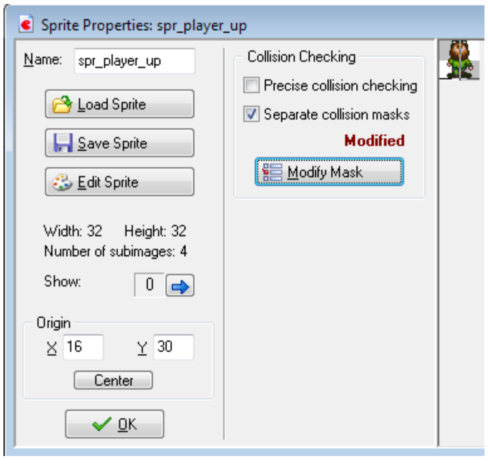
Start a new game and create six sprites using the graphics in the RPG and Parallel folder: wall, tree and the four players.

Hidden surface removal

First we must allow for the hidden-surface removal. As you know every sprite has an origin. By default this is the top-left corner of the sprite image. When we place an instance of an object at a particular position we actually put the corresponding sprite with its origin at that position. When we deal with a 3-dimensional world it is easiest to take as the origin of the objects the position where they stand on the ground.

For the tree and each of the player sprites we must:

- ♦ disable Precise collision checking
- ♦ set the point of origin
- ♦ modify the collision mask to set a small manual bounding box at the bottom of the character and tree. So collisions will not be performed with the whole sprite but only with this small rectangle. Player is done but not the tree.



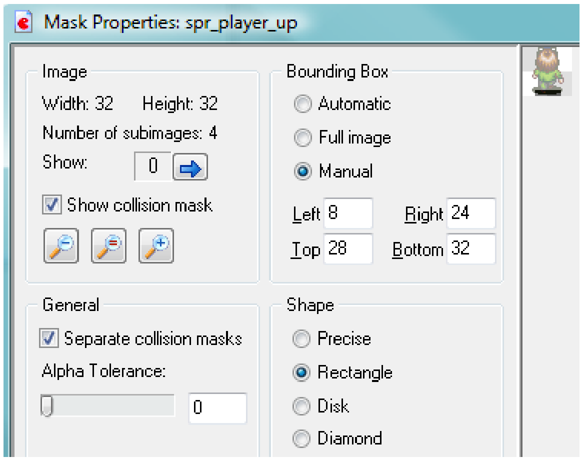
Sprite Properties: spr_player_up

Name: spr_player_up

Width: 32 Height: 32
Number of subimages: 4
Show: 0

Origin
X 16 Y 30

Collision Checking
☐ Precise collision checking
☑ Separate collision masks
Modified



Mask Properties: spr_player_up

Image
Width: 32 Height: 32
Number of subimages: 4
Show: 0
☑ Show collision mask

Bounding Box
○ Automatic
○ Full image
● Manual
Left 8 Right 24
Top 28 Bottom 32

General
☑ Separate collision masks
Alpha Tolerance: 0

Shape
○ Precise
● Rectangle
○ Disk
○ Diamond

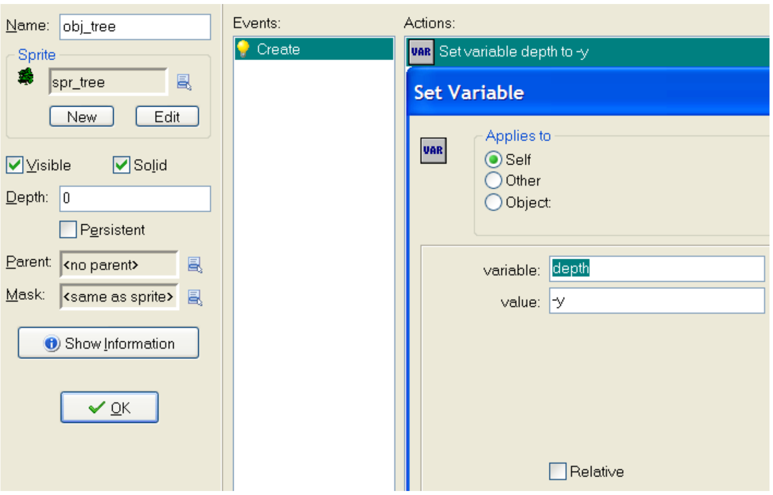
The wall is a single block of colour, later it will be covered over.

Create these three objects and assign the appropriate sprites: obj_wall, obj_tree, obj_player. With the latter assign spr_player_up. With the wall make it solid but not visible, we will tile over it and it will have no events attached to it. Make obj_tree solid and visible. Double check the bounding box for spr_tree covers the base of the trunk.

To make it possible for a character to pass in front or behind of another object and to cover up part of it or to be obscured itself, we need to use the depth property of the character and the other object. Remember that instances of objects with a greater depth are drawn first and therefore appear behind. Since an object which appears closer does so because it is further down the screen, its y value is greater. If we invert the y values and make those the depth, the objects further down the screen will be drawn after (on top of) those higher up (further away).
In most situations the order in which the objects must be drawn is the same as the order in which they stand on the ground. So by setting the origin as we did above, the y-coordinate of the instance determines the order (the y value is calculated from the point of origin). Instances with a low y-coordinate are close to the top of the window and must be drawn before the ones with a higher y-coordinate. So a higher y-coordinate must correspond to a smaller depth. This can be achieved by setting the depth to –y using the Set Variable action.

For the static objects, the trees in our example, we need to do this just once in the Create event.

Open up obj_tree and add a Create event with a Set Variable action. The variable is the word depth (a built-in variable name) and the value is minus y.



Name: obj_tree
Sprite: spr_tree
☑ Visible ☑ Solid
Depth: 0
☐ Persistent
Parent: <no parent>
Mask: <same as sprite>

Events:
Create

Actions:
Set variable depth to -y

Set Variable
Applies to
● Self
○ Other
○ Object:
variable: depth
value: -y
☐ Relative

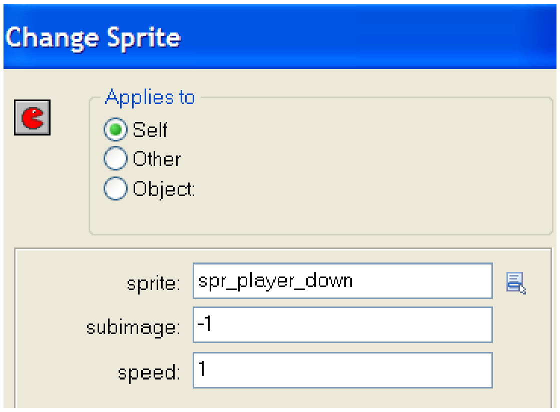
For the dynamic objects, the character in our example, we must do this at the end of every step. Open up obj_player and add the End Step event to make sure that all other events are processed before the depth is adapted. Attach the same Set variable action and settings.

Speed of the character

Secondly, we must deal with the motion of the character. We can do this in the usual way using the arrow keys. In each of the arrow keys event we will set the motion in the correct direction and set the correct sprite for the instance. Also, to get an animation of the sprite while moving, and no animation when the character is not moving, we use the variable image_speed. This variable indicates the speed with which the sub-images in the animation must be shown. Setting it to 1 plays the animation at a normal speed. Setting it to 0 will stop the animation. So whenever the user presses an arrow key we choose the correct sprite and set the image speed to 1 using the Change Sprite action.
(We use -1 for the sub-image to avoid a change in sub-image when we already move in the same direction.)

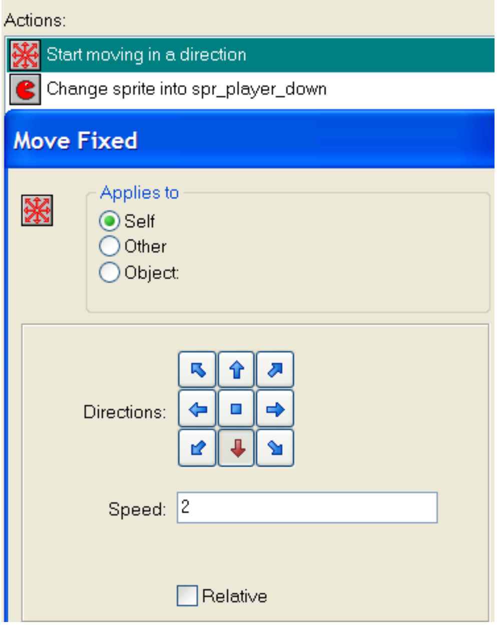
The screen capture to the right shows the actions for the Keyboard/Down event and the Move Fixed action.

The capture below shows the Change sprite action.


Change Sprite

Applies to
● Self
○ Other
○ Object:

sprite: spr_player_down
subimage: -1
speed: 1


Actions:
Start moving in a direction
Change sprite into spr_player_down

Move Fixed

Applies to
● Self
○ Other
○ Object:

Directions: ↓
Speed: 2
□ Relative

The only thing we do have to take care of in a 3- dimensional game is the difference in speed, left to right compared with towards and away. For the horizontal motion we will use a speed of 3 while for the vertical motion we will use a lower speed of 2. You must experiment with these numbers to get the best effect for your game.

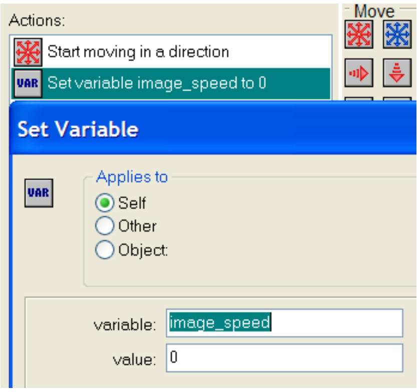
Create a Keyboard/Up event with the same settings except for the sprite and direction of motion.

Create Keyboard/Left and Right events with the appropriate sprites, directions of motion and set their speeds to 3.

Add a Keyboard/No key event. Add two actions. The Move Fixed action, obviously the centre button for no movement, with a speed of zero. But follow with a Set Variable action to set image_speed (a built-in variable) to 0 (so the character is not animated).

Collisions

Finally we must deal with collisions. Normally a collision occurs when the sprites partially overlap. This is fine for 2-dimensional game but not for 3-dimensional games. When the character stands in front of a tree the sprites overlap but there is no collision. So we need another area that must define the collision. Normally this is the area of the instance on the ground. Only when the feet of the character overlap with the bottom of the trunk of the tree is there a real collision. The easiest way to achieve this is to define a very small bounding box for all the sprites at the bottom and not to use precise collision checking. A collision is only reported when the bounding boxes partially overlap, which is exactly what we want. That is why we set these properties in the sprites. The character has been done for you but not the tree, you must change its bounding box to cover the trunk. The bottom will be the same as the base of the graphic but experiment with the other three values to get the black rectangle where it will look realistic as the character brushes through the thin, outer but is stopped by the thicker, inner branches.


Actions:
Start moving in a direction
Set variable image_speed to 0
Set Variable
Applies to
● Self
○ Other
○ Object:
variable: image_speed
value: 0

To obj_player add a Collision event with obj_wall.

Add two actions:

The first is, ironically, to Start moving in a direction but you click in the centre button so it does not move and enter a Speed of zero.

The second is to Set a Variable. In the variable name box type the built-in variable image_speed with a value of zero.

The character will stop when he hits a wall, just as he does when no key is being pressed.

To obj_player add a Collision event with obj_tree. Give it the same two actions with the same two settings. If it is not working correctly, a thing to check is the Point of Origin for the tree and character. Ensure it is not high in the tree as the Point of Origin is related to the depth of a sprite and the depth is important to which sprite is drawn in front. You will need to experiment with the numbers setting the point of origin so it looks realistic as the character brushes through the branches.

With more complex shapes a small rectangle at the bottom might not indicate the correct collision area. In that case we can use another feature of Game Maker. An object can have a different **mask** than the sprite used to represent the instance. So we can make a second sprite that has the shape of the required collision areas and use that as a collision mask.

Tiling

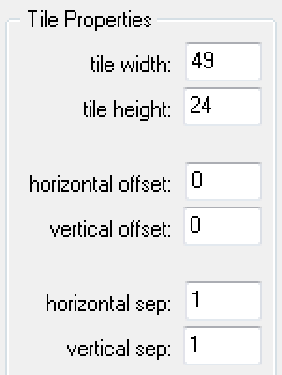
We now have all the ingredients for the 3-dimensional game we want to create. Create a room and ensure on the Settings tab it is 720 by 480 with an appropriate caption. On the objects tab set Snap X to 49 and Y to 24 (this will be the size of an individual tile and the wall sprite was sized to match). Place wall objects around the outside to form an invisible boundary (you are about to cover them over).

To make it look nicer we will use a tileset for the background in the room. Tiles have the advantage of being very small files and they are inert, the computer does not regard them as objects to be tracked constantly.

Add as a new background the file back_landtiles.bmp. This is a graphic found on the web.

Its designer made channels where the individual tiles are designed to be cut. You can see how the large graphic is divided.

Tick in the box to use as a tileset. Then set the Tile Properties to match these, they work for me but you may need to fine tune.



Tile Properties

tile width: 49

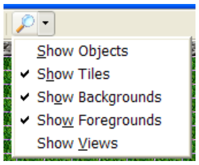
tile height: 24

horizontal offset: 0

vertical offset: 0

horizontal sep: 1

vertical sep: 1



Then place the tiles, it is easier to place tiles if you turn off Show Objects.
You will need to widen the middle column to easily see the entire width of the tileset without scrolling.

You will be back and forth between the Objects tab and the Tiles tab as you place the solid but invisible wall objects with which a character can collide and the more attractive but insubstantial tiles. Do not try to match the world of the game you played, make your own. Do not use the trees from the tileset as we will use our tree objects. Do not spend forever making the room look good, you can always return and improve it later.

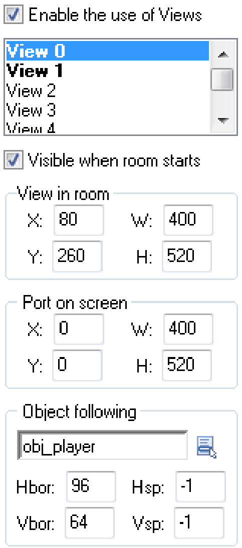
Finally add some instances of the tree object and a character. Turn off delete underlying to place on instance on top of something.

Save, run and test to see how it looks.

We can improve the room. We will use a view in the room such that the player sees only part of the world. It will add an element of exploration.

We will scale the view but at the same time allow for space at the bottom. In many games there is an information panel below the scrolling world. (Many developers put an information panel at the bottom of the screen rather than at the side for this type of game. See the illustration from Sim City later).
It is so common in RPGs we will do it ourselves later.

Use the settings in the graphic to the right.



☑ Enable the use of Views

View 0
View 1
View 2
View 3
View 4

☑ Visible when room starts

View in room
X: 80 W: 400
Y: 260 H: 520

Port on screen
X: 0 W: 400
Y: 0 H: 520

Object following
obj_player
Hbor: 96 Hsp: -1
Vbor: 64 Vsp: -1

Save the game, run and test it.

It's not a game so far but we have created the environment. Many puzzle games use this type of environment. Now we will turn it into a role playing game where the hero needs to get to the next level in pursuit of his heroic quest. To do this our hero will have to ask directions of characters he meets and be prepared to answer riddles (monster slaying is not required).

Resizing the Room

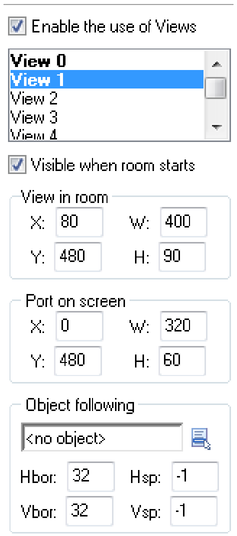
We will start stage two by increasing the size of the room and splitting it into two views. This will allow us to place messages in the bottom part in a second view and have them visible wherever the character is in the room, which is shown in the first view above it.

Open up the room and on the Settings tab increase the height to 540.

On the Backgrounds tab set the background color to be black. This will give us a dark area along the bottom.

On the Views tab use the graphic to the right to create a second view (View1).

View0 will follow the character but View1 will not move. It is where dialogue and the contents of the inventory will be displayed.



☑ Enable the use of Views

View 0
View 1
View 2
View 3
View 4

☑ Visible when room starts

View in room
X: 80 W: 400
Y: 480 H: 90

Port on screen
X: 0 W: 320
Y: 480 H: 60

Object following
<no object>
Hbor: 32 Hsp: -1
Vbor: 32 Vsp: -1

The Graphical Resources

Let's add all the remaining graphics and set the properties of the sprites:

trigger	bounding box full image on Mask properties
key	origin to center
house	bounding box to full image on Mask properties
troll	be polite, its very strong
statue	turn off Precise collision checking, origin x =20 y = 70, bounding box to lower half of image
door	origin x =8, y =8, bounding box to full image

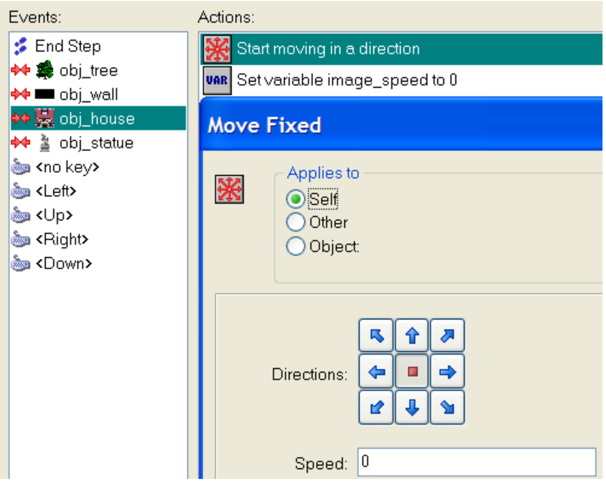
The House

This will be the entry next level in the game (although we won't have time to develop it today). But our hero will not be able to enter until he has found the key and the key will not appear until he has correctly answered THE QUESTION.

Create an obj_house and assign spr_house. It should be visible and solid but needs no events.

Open up obj_player and create a collision event with obj_house.

The two actions are the same as we have recently used. Start moving in a direction which is really no direction with a speed of zero; and set the variable image_speed to zero so the animation stops.



Start moving in a direction

Set variable image_speed to 0

Move Fixed

Applies to

● Self

○ Other

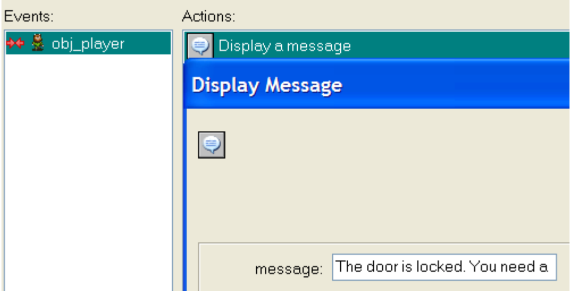
○ Object

Directions:

Speed: 0

We will place a barrier across the door. When the character collides with the barrier it will cause a message to appear saying the door is locked. If the hero answers THE QUESTION, the key will magically appear and the barrier will magically disappear.

Create an obj_trigger and assign spr_trigger. The object should be solid but not visible.
Add a Collision event with obj_player.
Drag and drop a Display a Message action from the main2 tab.
The message is:
The door is locked. You need a key to enter.



Display a message

Display Message

message: The door is locked. You need a

Place an instance of the house in the room and an instance of obj_trigger in front of the door, blocking access. Remember to turn off delete underlying to put one object on top of another.

Save and run the game. Notice how easy it is to display a message using drag and drop, but that is no good for dialogue.

The Troll

Speaking of dialogue, let's add a troll, it should be good for a chat.

Create an obj_troll, assigning spr_troll and making it visible and solid (well it is a troll!).

Place an instance of it in the room.

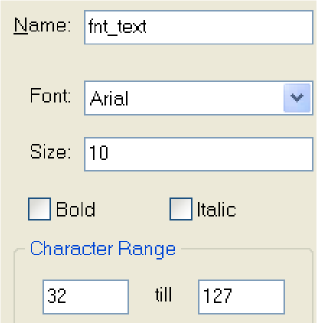
We want our character to be able to go up to the troll and converse with it. We want there to be more than one line of dialogue. We cannot use the drag and drop Message action, we need a script.

The First Script

We have already used the built-in show message function, so to extend our range we will use other functions (all of which are detailed in the Help section of Game Maker).

These will need a font to write with so we start by adding a font.

Name it fnt_text. I used Arial 10 point, which looks big enough on a screen.



Name: fnt_text

Font: Arial

Size: 10

□ Bold □ Italic

Character Range

32 till 127

When the dialogue appears we will precede each line with the name of character speaking. To show off we will use code to make the names appear in a different colour but use another font to make them appear in bold.
Create a second font, call it fnt_text_name with the same settings except for a tick in the box for Bold.

The script I have written will allow for two lines of dialogue (you could cut it to one or add more but this amount fits well in the space). Each line will have the speaker's name appear in yellow, followed by the words.

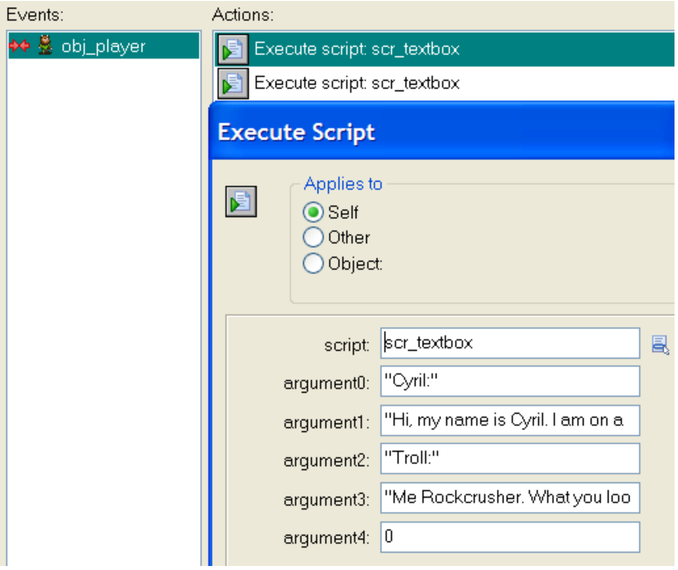
To save time and typong errorrrs the script is in the Network folder. You can drop down the Script menu in the toolbar and choose Import Scripts, bring in scr_textbox.gml.

Open up the script and examine it. Notice the pattern: two colours, yellow and white; two different fonts; the draw_text function called four times. Each time the words will appear in the second view (View1) in a slightly different place to avoid overlapping. The words (in programming vocabulary the strings) are not given in the script, although they could be. Instead there are four arguments. An argument is the raw material and we will supply this to the script from a drag and drop action. The advantage of this is that the same script can be called more than once and each call can give it different strings (words) as arguments.

```
        screen_redraw();
        draw_set_color(c_yellow);
        draw_set_font(fnt_text_name);
        draw_text(view_xview[1]-70,view_yview[1],argument0);
        draw_set_color(c_white);
        draw_set_font(fnt_text);
        draw_text(view_xview[1]-20,view_yview[1],argument1);
        draw_set_color(c_yellow);
        draw_set_font(fnt_text_name);
        draw_text(view_xview[1]-70,view_yview[1]+15,argument2);
        draw_set_color(c_white);
        draw_set_font(fnt_text);
        draw_text(view_xview[1]-20,view_yview[1]+15,argument3);
        screen_refresh();
        io_clear();
        keyboard_wait();
```

Now we will call the script and supply the arguments. To obj_troll add a Collision event with obj_player, drag over an Execute script action from the control tab.

Set the script to scr_textbox. Notice in the graphic to the right that the words to appear **must** be placed in double quotation marks (to show they are strings). Argument1 is (or make up your own):
"Hi, my name is Cyril. I am on a quest."
Argument3 is:
"Me Rockcrusher. What you look for?"



● Self
○ Other
○ Object:

script: scr_textbox
argument0: "Cyril:"
argument1: "Hi, my name is Cyril. I am on a
argument2: "Troll:"
argument3: "Me Rockcrusher. What you loo
argument4: 0

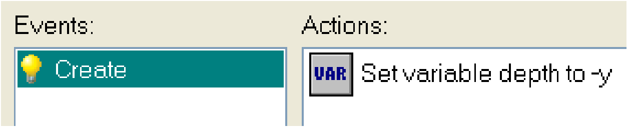
The script uses four arguments, for two lines of dialog. In an Execute script action you can only enter five arguments. One way to get more lines is to call the same script again and give it different arguments. Drag in another Execute script action.
This time argument1 reads, "I am questing for a magic keyring."
Argument3 reads, "Me, no know." Arguments 0 and 2 are the same as before (the names of the speakers).
If you want to force a line break in a text string you use the # symbol.

Save the game and test it. Bump the hero into the troll, note which arrow keys trigger the second script.

The Statue of Knowledge

The troll was a non playing character, an NPC in RPG speak. We will use another one to showcase a different technique. How to ask a question.

Create an object named obj_statue, assigning spr_statue, make the object visible and solid.



Set variable depth to -y

We want our hero to be able to walk around the statue in the same way as he did with trees (we could have done it with the troll but who wants to see a troll's backside?) So we add a Create event with a Set variable action where depth is set to –y.

Put an instance of the statue in the room.

We will use another script to ask a question and handle the answer. We could use arguments here also but for a change we will place the question and the response if correct as strings in the script itself.

Drop down the Scripts menu in the Game Maker toolbar, choose to Import Scripts and select scr_question from the Network folder.

Open the script and examine it.

```
screen_redraw();
ask=show_question("Are you having fun yet?");
if ask=true then
{
//place the key in the inventory area and show a message
draw_sprite(spr_key,-1,view_xview[1]-30,view_yview[1]+15);
draw_set_color(c_white);
draw_set_font(fnt_text);
draw_text(view_xview[1]+15,view_yview[1],"Now you can enter the
Shop, my hero");
//deactivate the object which blocks the door to the shop
instance_deactivate_object(obj_trigger);
screen_refresh();
io_clear();
keyboard_wait();
}
```

Notice that the show_question function returns True if the true button is pressed. A variable called ask is used to trap the response. Since this is a True/False question and we do not need any action taken on False (the wrong answer), we only need to code the response to True, the correct response. A sprite of a key and a message will appear. Then the barrier in front of the door, which is an instance of obj_trigger, will be deactivated. Deactivation is a built in function which in effect disposes of the instance.

Open obj_player, add a Collision event with obj_statue to which an Execute script action is attached. The script is scr_question and there are no arguments.

Save and test the game. You should see the feedback, including the use of a sort of inventory, but will not be able so easily to see if the invisible barrier has disappeared.

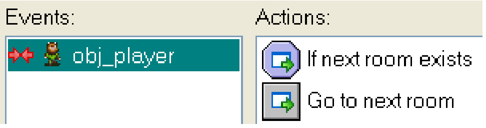
Is The Way Clear?

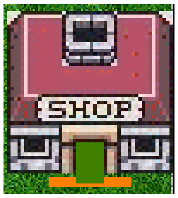
We need another level. Add a new room.

There is one sprite left unused, it will work for us as an invisible cover for the door of the shop. When the player touches it, he will be taken to the next level.

Create an obj_door and assign spr_door as the sprite. Make it solid but not visible.
Add a Collision event with obj_player to which are attached two actions from the main1 tab:

Check if the next room exists
Go to next room.


Events:
obj_player
Actions:
If next room exists
Go to next room



We want to place obj_door over the door to the shop.

It does not matter if it slightly overlaps the top of the trigger.

Save the game and run it, testing the barrier disappears and our hero can ascend.

Persistent

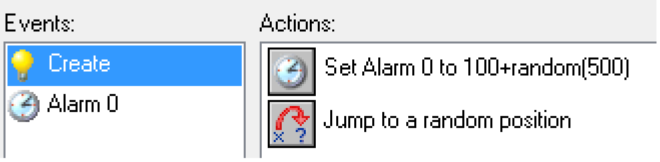
Both rooms and objects can be made persistent. In an RPG game it is common to make a room persistent, so if a room is re-entered it is the way it was last left (whereas otherwise a room is reset to its initial settings). If we do that here (on the room’s settings tab) our hero could re-enter the room and not have to answer the question again in order not to be blocked from exiting.
Tick the box to make obj_player persistent. If you run the game again you will find Cyril is present in the second room in the same location as in the first.

The Orb of Power

It is common in games for a character to be able to pick up an object which goes into its inventory, for example ammunition or something to confer a special power.
We will use an orb which magically appears and disappears at random within the field of play. When Cyril collects the orb, it will appear in his inventory.

Create a spr_orb and load the graphic orb. Now create two objects, obj_orb and obj_orb2, and assign both the same sprite. Why we need two objects which look the same but have different actions will become apparent.

Open obj_orb and add a Create event with, from the main2 tab, a Set Alarm action for Alarm0 and set number of steps to 100+random(500).



Events:
- Create
- Alarm 0

Actions:
- Set Alarm 0 to 100+random(500)
- Jump to a random position

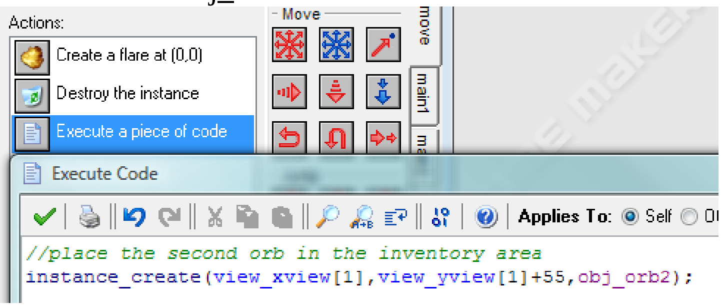
This will result in the alarm ringing at a random time between 100 and 600 steps. Add a Jump to Random action with the default parameters. This will move the instance to a random empty position when it first appears, but it only happens once.

Add an Alarm/Alarm0 event with a Set Alarm action for Alarm0 with Number of Steps set to 100+random(500), then a Jump to Random action. This will cause the instance to keep moving at random after the create event, as the alarm keeps ringing.

The orb will now magically appear and disappear.

When Cyril picks up the orb we want a flash of light, the orb to disappear from the field of play and appear in the inventory.

Open obj_player and add a Collison event with obj_orb. From the Draw tab drag over a Create an Effect action (if you are using the Pro edition). Set type to flare, size to large, where to above objects and tick Relative.
Now for a Destroy Instance action set to other (so the orb is destroyed).
Drag over an Execute Code action and type this:



Actions:
- Create a flare at (0,0)
- Destroy the instance
- Execute a piece of code

```
//place the second orb in the inventory area
instance_create(view_xview[1],view_yview[1]+55,obj_orb2);
```

```
instance_create(view_xview[1],view_yview[1]+55,obj_orb2);
```

Place an instance of obj_orb somewhere in the room. Save the game and run it. Check that the orb appears and disappears and that when the player picks up the orb the three things happen (you can’t use a sprite effect in the Lite edition).

You will now realise why we needed an orb2. If we had used the same one in the inventory area, its create event would have started it jumping again.

We have created a simple RPG game using techniques which can be changed to fit other purposes. For example, if we want an item to appear permanently in the inventory when a character acquires it, we could put the line of code in the Draw event.

Challenges

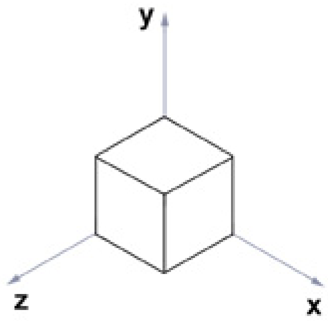
The draw_sprite function could be used to place an icon of the character speaking in the text area instead of the name. You will need to create/acquire two small graphics and modify the script.

Our final game used the **isometric effect** (when you look down at an angle on the game world).

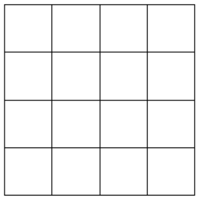
In isometric projection all three of the major axes are foreshortened equally.

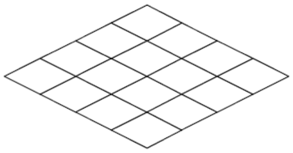
The principle of an isometric game is that the 3D world is viewed under a fixed 45-degree angle. Again parallel projection is used, that is, there is no perspective, and so objects in the distance will not become smaller. Assume the world consists of square cells as in the left picture below.

An isometric view will look like the right picture. Each square has become a diamond shape.



Isometric Projection of a Cube



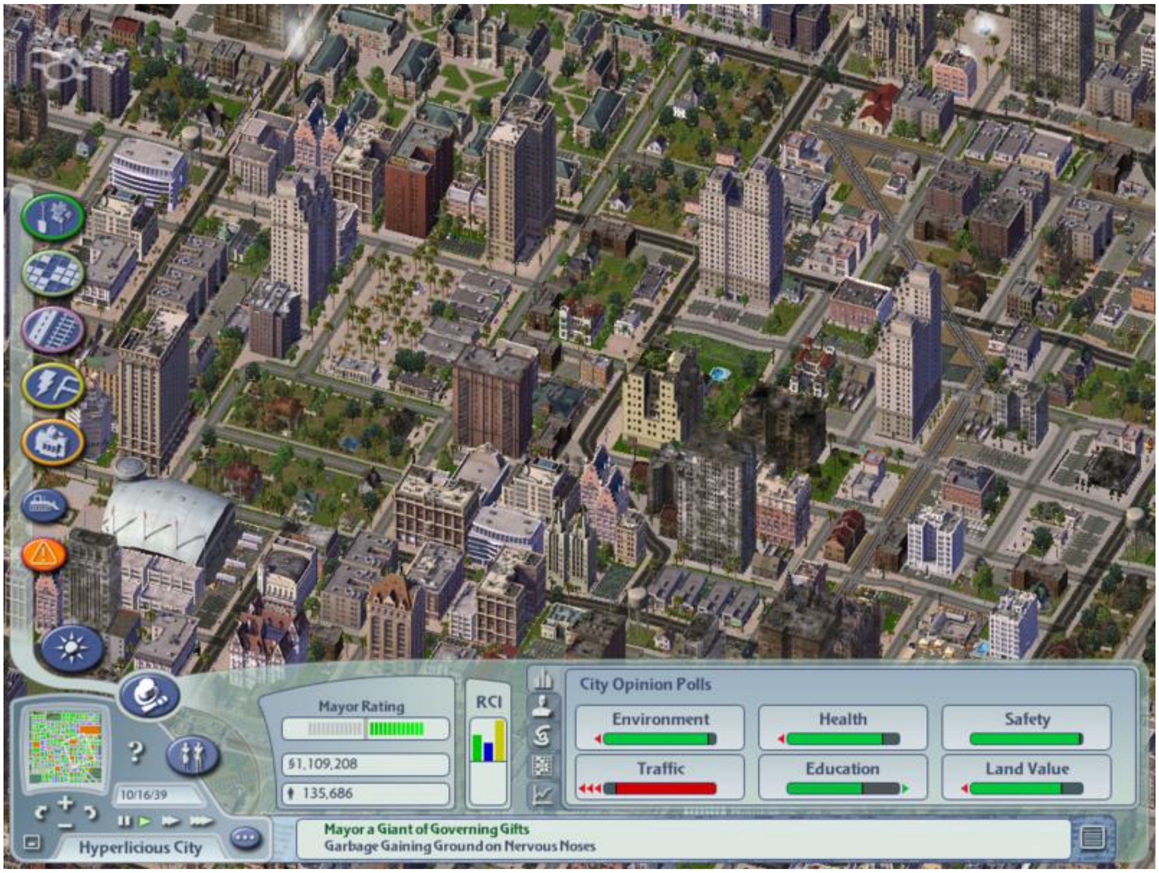


For a more complete coverage, see the Adding Depth tutorial by Mark Overmars on the network drive.

If you are interested in developing a first person shooter game, which typically uses many of these effects, see the folder GM Tutorial First Person Shooter.

* * * * * * * * **

Isometric Projection is used in Sim City 4



Mayor a Giant of Governing Gifts
Garbage Gaining Ground on Nervous Noses

Appendix 1

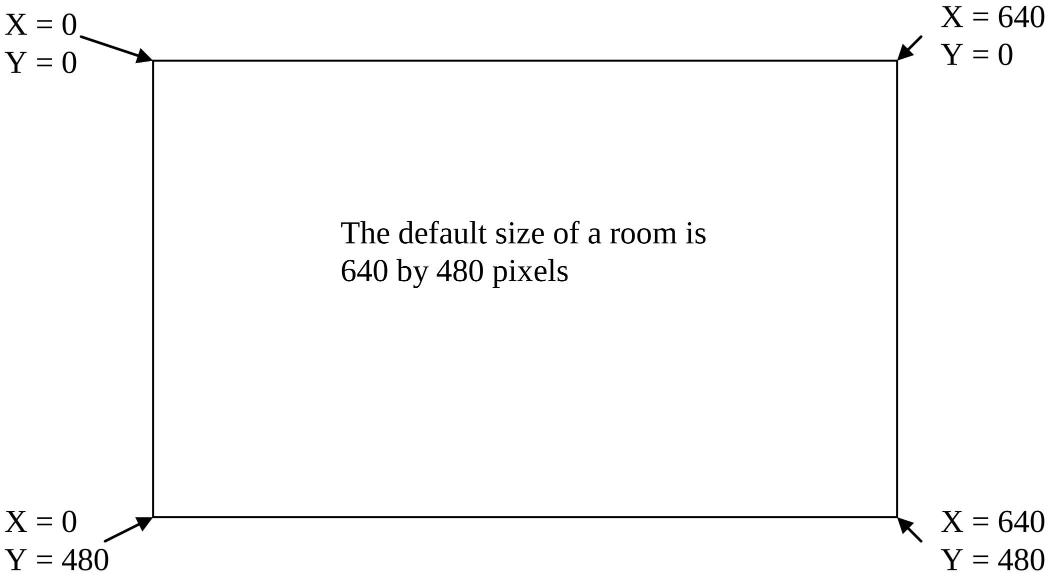
Understanding Location, Direction and Speed in Game Maker

Location

An instance in a game is located by a co-ordinate system. Location horizontally across the screen is on the x axis. Location vertically up and down the screen is on the y axis. With a combination of x and y you can place something in two dimensions. The Perth street directory uses the same co-ordinate system.

Point of Origin

The top left corner of the room is the point of origin, that is where x= 0 and y =0.



The default size of a room is 640 by 480 pixels

If you move an instance to the right that is a positive increase in x; if you move to the left that is a negative change in x. For example x + 10 will move ten pixels to the right; x -10 will move ten pixels to the left. Y behaves the same: +Y moves down; -Y moves up the screen.

A sprite also has a point of origin at the top left corner of the image. So if you put a sprite at x= 10 and y =20 in the room, the top left corner of the image will be at those co-ordinates. This effects you when you place an object in a room, where you left click will be the point of origin.

You can alter the point of origin of a sprite. For example if you want a shell to emerge from the main gun of a tank (and not from the top left corner of the overall image), you would shift the point of origin of the sprite. We will do this in later games.

There are two other ways in which knowing about the point of origin is important. Sometimes, you may find an object fires a bullet but you never see it. The reason is that if the bullet is much smaller than the object it may be created so far inside the object that the front of the bullet hits the inside edge of the object and is destroyed before it emerges. This is especially the case with objects which fire down and the point of origin is left at the default value of top left corner. This may happen to you in Space Invaders

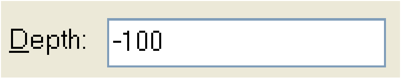
The Importance of Relative:
In dialog boxes where coordinates are involved there will be a check box for Relative. If you do not understand the significance of this you will find that sometimes your objects jump to the top left corner of the screen.

For example in Pong you want to move a bat down the screen but not sideways, so the value of X will stay the same and the value of Y will increase. In the dialog box for a small movement you might enter X=0, Y=5.
If you do **not** tick relative, the bat will move to the top left of the games window, X:0, Y:5 **of the room**.
If you do tick relative, the bat will move five pixels down the screen **relative to its present position**.

So ticking or not ticking relative is an important decision.

Depth

The third dimension is depth. In some systems, such as layers in an XHTML document, the third dimension uses z. In Game Maker it is the word “depth” itself and is a property of an object.

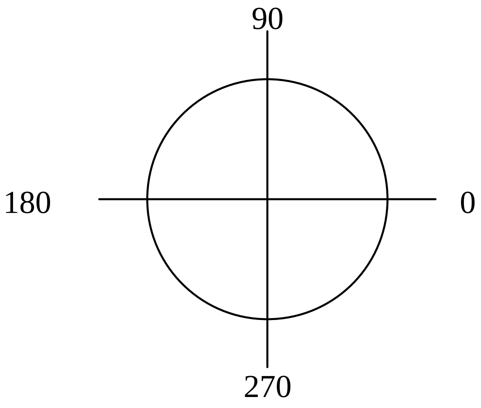


Depth: -100

The surface is given a depth of 0. Anything with a positive number is below the surface; anything with a negative number is above it. Objects have a depth property you can set in their property form. You will use this, for example, if you have a plane flying over the ground. If the ground (usually the background of a room) is 0, then the plane could be -100. If you want the plane to fly under some clouds, then the clouds could be -200.

Direction

Direction is set by an angle. Game Maker uses this system:

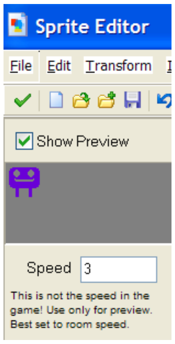


You will need to use this to control direction. For example, if you wanted an object to move diagonally up and to the right, you can select the Move Free action and give it a direction of 45.

Speed

You need to understand two ideas about speed in Game Maker. The first is that when you set the speed with which an object moves it means how many pixels it moves each step. So with the Move Free action above if you set direction to 45 and speed to 5, the object would move diagonally up and to the right at 5 pixels each step.

The second idea is how long is a step? The default setting in a room is 30 steps a second. So in the example above the object would move at a rate of 5 X 30 = 150 pixels a second. You can change the default step size. Doing so in a room will slow down or speed up every movement which takes place there.



☑ Show Preview

Speed 3

This is not the speed in the game! Use only for preview. Best set to room speed.

You can also set the speed when you are experimenting with the animation of a sprite. Setting a low number like 3 lets you slow down an animation so you can check how it is working. The default setting of 30 is the speed of the game.

Appendix 2

Programming Words You Need to Know

Assign

A value is assigned to a variable. That is, something is put into the container.

Boolean

Developed by the English mathematician and computer pioneer George Boole, a **Boolean** consists of operators such as "AND", "OR", and "NOT". Booleans are commonly used in programming and search engines. Boolean expressions are expressions that result in the value of either TRUE or FALSE.

Conditional

A conditional statement is commonly met in an If_Then_Else expression. For example: *If* it is raining, *Then* drive the car to TAFE; *Else* take the train.

Debug

To find and remove an error.

Function

A set of instructions that produces a result, such as a value from a calculation.

Loop

Repeating the same instructions over and over. Can be for a pre-determined number of times, a counted loop. Or not known in advance and until directed to stop.

Instance

A copy of an object. An object is like a blueprint, it can be copied endlessly.

Statement

A single line of code, a single command.

Variable

An address set aside to hold temporary data. It’s a container, the contents of which can vary. A **global** variable is understood by all parts of a program. A **local** variable is more restricted, in Game Maker to only the room in which it is first used.
In many programming languages, including that behind Game Maker, a variable must be **initialised** before it can be used: that is given an initial value.

Appendix 3

Game Maker 7 Registered Version: Short Action Names (list for 8 not available at time of writing)

Move	Main1	Main2	Control
Move Fixed	Create Instance	Set Alarm	Check Empty
Move Free	Create Moving	Sleep	Check Collision
Move Towards	Create Random	Set Time Line	Check Object
Speed Horizontal	Change Instance	Time Line Position	Test Instance Count
Speed Vertical	Destroy Instance	Display Message	Test Chance
Set Gravity	Destroy at Position	Show Info	Check Question
Reverse Horizontal	Change Sprite	Show Video	Test Expression
Reverse Vertical	Transform Sprite	Restart Game	Check Mouse
Set Friction	Color Sprite	End Game	Check Grid
Jump to Position	Play Sound	Save Game	Start Block
Jump to Start	Stop Sound	Load Game	End Block
Jump to Random	Check Sound	Replace Sprite	Else
Align to Grid	Previous Room	Replace Sound	Exit Event
Wrap Screen	Next Room	Replace Background	Repeat
Move to Contact	Restart Room		Call Parent Event
Bounce	Different Room		Execute Code
Set Path	Check Previous		Execute Script
End Path	Check Next		Comment
Path Position			Set Variable
Path Speed			Test Variable
Step Towards			Draw Variable
Step Avoiding			

Score	Extra	Draw
Set Score	Create Part System	Draw Sprite
Test Score	Destroy Part System	Draw Background
Draw Score	Clear Part System	Draw Text
Show Highscore	Create Particle	Draw Scaled Text
Clear Highscore	Particle Color	Draw Rectangle
Set Lives	Particle Life	Horizontal Gradient
Test Lives	Particle Speed	Vertical Gradient
Draw Lives	Particle Gravity	Draw Ellipse
Draw Life Images	Particle Secondary	Gradient Ellipse
Set Health	Create Emitter	Draw Line
Test Health	Destroy Emitter	Draw Arrow
Draw Health	Burst from Emitter	Set Color
Score Caption	Stream from Emitter	Set Font
	Play CD	Set Full Screen
	Stop CD	Take Snapshot
	Pause CD	Create Effect
	Resume CD	
	Check CD	
	Check CD Playing	
	Set Cursor	
	Open Webpage	

Appendix 4

central

INSTITUTE OF TECHNOLOGY

Certificate III in Media
D192
CT40
CUF30107

Author Interactive Sequences

D1108

CUFDIG302A

Learning Plan

Semester 1, 2010

Lecturers:

Andy Hawkins *Day telephone 6211 2465* *E-mail* andy.hawkins@central.wa.edu.au	*Richard Lancaster* *Day telephone 6211 2464* *E-mail* richard.lancaster@central.wa.edu.au
Adil Mistry *Day telephone 6211 2465* *E-mail* adil.mistry@central.wa.edu.au	Sue Williams *Day telephone* TBA *E-mail* sue.williams@central.wa.edu.au

Description:

This is a national unit, the description of which is:

This unit describes the performance outcomes, skills and knowledge required to use an authoring tool to produce discrete interactive sequences.

This unit will introduce you to authoring 2D games using a program called Game Maker. No previous knowledge of authoring or creating games is required. In the first part you concentrate on learning the software, how to make a game using the latest version of Game Maker, V8, released in December 2009. In the second part of the unit you concentrate on designing and creating an original game. Because the game industry works in teams, you will work with a partner in the second part.

Duration:

50 hours (two and half hours a week for one semester)

Prerequisites:

None

Resources

You will be supplied with a printed copy of this student workbook. If you lose it, you will have to print out another for yourself from the softcopy in the network folder. The images and sound files mentioned in the workbook are available on the student network

Elements of Competency:

- Plan use of authoring tool
- Prepare to use authoring tool
- Produce interactive sequences
- Check functionality of interactive sequence

Individual Learning and Assessment Needs

Central Institute of Technology recognises that students have different learning styles and needs. Please let your lecturer know if there is anything that may have an effect on your learning.

ASSESSMENT SUMMARY

DUE	ASSESSMENT
To obtain an overall pass mark, you must successfully pass in **each** of the following. A percentage has been given to indicate the relative weighting of each:	
Session 6	Analysis of an existing game. 15%
Session 10	Completion of the games developed in the workshops. 10%
Session 12	Design documentation for a new game, in collaboration with a partner. 25%
Session 17	A complete, original game. 50%

Support Materials

There are many books on game design in the Central Institute of Technology Library, including:

Chris Crawford (2003), *Chris Crawford on Game Design*, New Riders, Indiana.

Jason Darby (3rd edition, 2008), *Awesome Game Creation: No Programming Required*, Thomson Learning, Boston.

Jacob Hapgood and Mark Overmars (2006), *The Game Maker's Apprentice: Game Development for Beginners*, Apress, New York. (On closed reserve in library)

Jeannie Novak (2008), *Game Development Essentials: Game Project Management*, Thomson Delmar, New York.

Nanu Swamy and Naveena Swamy (2006), *Basic Game Design and Creation for Fun and Learning*, Charles River Media, Massachusetts.

Online

YoYo games, the home of Game Maker	http://www.yoyogames.com/

There are several Word documents in the network drive which list various types of resources (tutorial sites, sites with games made in Game Maker, graphics and sound resources). They are left as soft copy so you can click on a link to open it without retyping.

Class Schedule

There are three Public Holidays on a Monday during Semester One and one on a Friday.

This will effect which week some classes cover a topic. Your lecturer will advise you.

Session	Topic	Assessment
1	Introduction to the unit and 2D game design using Game Maker.	
2	Recreating classic video games: Pong	
3	Recreating classic video games: Space Invaders	
4	Recreating classic video games: Pac Man	
5	Recreating classic video games: Mario	
6	Presentation to class of your chosen game and your analysis of it.	**Assessment Item 1**
7	Using code and scripting	
8	3d and isometric effects	
Break	**No Classes**	
9	Parallel effects and scripting in an RPG	
10	Demonstration to lecturer of all eight completed games	**Assessment Item 2**
12	Appointment with partner(s) and lecturer	**Assessment Item 3**
13	No class, private work on Assessment 4	
14	No class, private work on Assessment 4	
15	No class, private work on Assessment 4	
16	Appointment with partner(s) and lecturer to display prototype of Assessment 4	
17	Presentation to class, hand in files on CD	**Assessment Item 4**
18	Resubmission if required	

Results and appeals.

Please refer to the Central Institute of Technology website for information about the assessment process. The information can be found at www.centraltafe.wa.edu.au. The path is; home – current students- your studies – assessment.

Assessment Item 1 Due Week Six

Your game review will consist of two to three typed and spell checked pages which you should hand to the lecturer attached to a New Media Assessment Cover Sheet. You should cover these issues:

- Title, creator, year, intended audience, genre
- Description of the game (add some pictures). Do **not** make this section too long, it must be no more than one third of the review
- Role of the story (if any)
- Quality and appropriateness of graphics, sounds, and music
- Features versus appearance
    - What is essential in the game and what is not
- Game play
    - forms of decision making
    - balance between the different features
    - motivation to start and keep playing
    - variation, long-lasting appeal
- What could have been left out without hurting the game
- What should have been added to improve the game

Your review should be written from a game designer's perspective not just from a game player's.

Your game should be 2D rather than 3D and must have been made using Game Maker. In the network drive there is a Word document of links to websites with such games.

In class you will briefly show the game and summarise (not read) your review.

Assessment Item 3 Due Week Twelve

Create with your partner(s) a concept document (sometimes called the pitch document). The purpose of the document is to sell the idea so that development will be authorised/funded.

In one or two sentences state the basic idea of your game. Imagine this as a statement that could be used in the packaging of the game to sell it.

Say who the intended players of the game are. Make sure you give the target age and any other relevant demographic or geographic information. Say which genre (such as action, puzzle, role playing) your game fits into (players tend to buy and play games in their favourite genres).

Describe the characters and the game play. The game must be created in Game Maker. You will have to work within its limitations but also be able to use its capabilities.

List the features (such as enemies and power-ups), with information on what their function is and where and how they appear in the game. State how rewards and penalties will be handled.

Describe the user interface.

Include a description of the various levels in the game.

List music and sound effects needed (you will not be expected to create original audio).

In a page give a timeline for the production as a three column table, with detail on who is responsible for what and when by.

All this should be done within five or six typed pages.

Attach concept drawings and sketches of characters and scenes (you will not be expected to create original graphics).

Note that the final game seldom matches exactly the design document. You can and should amend your design as new ideas or circumstances dictate.

Assessment Item 4 Due Week Seventeen

Warning: In the first part of the unit you followed detailed instructions to make games. This will have led you to underestimate how long it takes to make a game.
It will take you much longer than you think to make a game and you will often hit problems when you try to make what you have imagined. **Allow plenty of time.**

You will present your game to the class and give it to the lecturer on a CD, which contains both the exe file and the Game Maker .gmk file. The lecturer will load the executable into the network drive so everyone can play it later.

During your presentation you should state what changed during development, where you altered your design, and what were the areas of difficulty you experienced.

Your game should be complete and fun to play. It should have an intro (e.g. starting screen), help, credits and at least two levels of progressive difficulty. It does not have to be a very long game though.
Please add a cheat using the N key to skip through the levels for testing purposes.

* * * * * * * *