G54GAM – Lab Session 1

The aim of this session is to introduce the basic functionality of Game Maker and to create a very simple platform game (think Mario / Donkey Kong etc). This document will walk you through some of the basic implementation details, however it will not cover everything – this should be a process of "learning by doing" and you are expected to find solutions to problems you may come across, as much of Game Maker is self explanatory. Try to use your imagination to turn this basic implementation into a fun game.

Game Maker Basics

http://www.yoyogames.com/gamemaker/windows#requirements
http://www.cs.nott.ac.uk/~mdf/g54gam/GameMaker-Installer-8.1.exe

Objects and **Instances**

The main entities of the game, specify properties such as solidity (ability to collide with other objects), sprite, actions that respond to certain events (keys pressed, mouse moved, collision with other objects)

Main playable character

Moves on keyboard **events**
Increase score on collision event with a gold coin

Walls, monsters, collectable items etc

Multiple **instances** of a monster **object**

Sprites

Images that are used to represent **objects**

Rooms

Where the game takes place (levels)
Contain **instances** of **objects**

Events

Things that happen to objects during the game – key presses, collisions, timers
Cause **actions** to occur

Actions

Change what an object is doing – position, velocity, animation
Test various parameters – simple conditional logic

Pixels

Units of space

Steps

Units of time - game state updated each step (~1/30 seconds)

Speed

Pixels travelled per step

Gravity

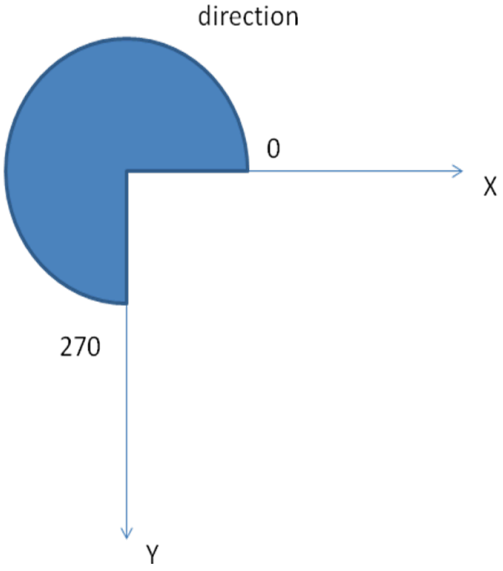
Change in vertical speed per step

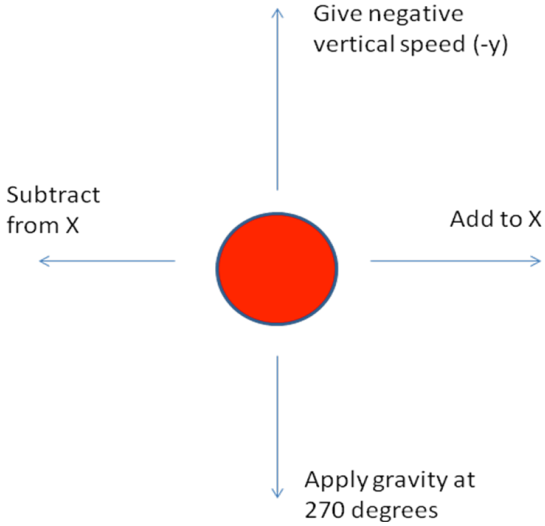
X / Y

The horizontal and vertical coordinates of an object, in pixels

Direction

Degrees from the X+ axis





A Simple Platform Game

- A simple action platform game
- The player can move their character left, right and make them jump
- The aim of the game is to collect all of the gold coins in the level without falling in a pit

Create a blank new game, and start by creating two simple **sprites**. Click "resources -> add sprite" in the menu.

- A square wall sprite to use to build platforms with
- A character sprite that we're going to move around.

Create two solid **objects** that make use of the sprites. Click "resources -> add object" in the menu. In the new object's properties window assign the sprite made earlier.

- A wall object
- A character object

Create a **room** and add some of wall **instances** and one character **instance**.

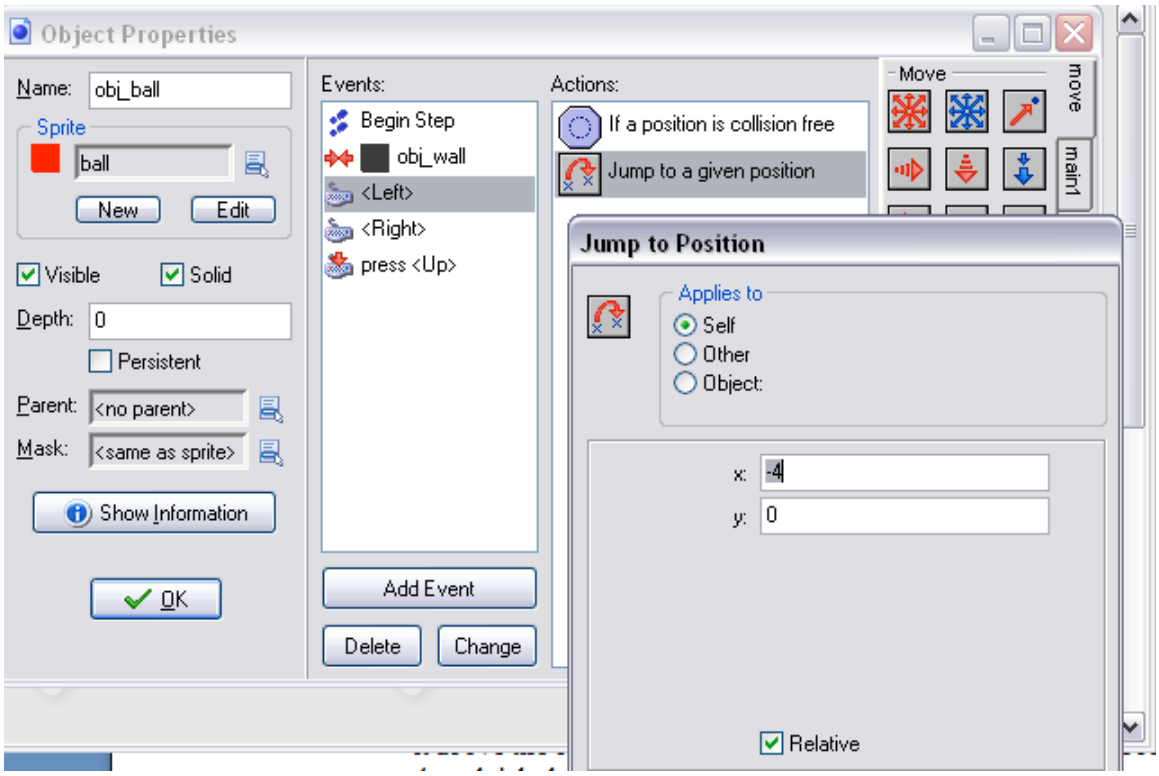
Motion

The core mechanic of a platform game is that our character can move to the left and right, jump into the air and, most importantly, fall to the ground and between gaps in platforms.

This involves implementing some motion logic ("physics") relating to the player's input, or use of the cursor keys, for moving the character object and in turn having the character object react to its surroundings – for example stopping when it reaches a wall, and falling when it is above a gap in the platform.

First we'll add horizontal movement, which is the simplest. When the player presses either the left or the right cursor key, we move the character object by a small amount in the appropriate direction, relative to its current position.

- Add a **Keyboard Event <Left>** event to the character object. Click the "add event" button in the character object properties window. This fires whenever the left cursor key is pressed
- Add a **Jump to Position** action to the **Event**, and set the **x** value to -4, and ticking the **Relative** option. This means that when the left cursor key is pressed, the object will move -4 pixels along the x axis – or 4 pixels to the left.



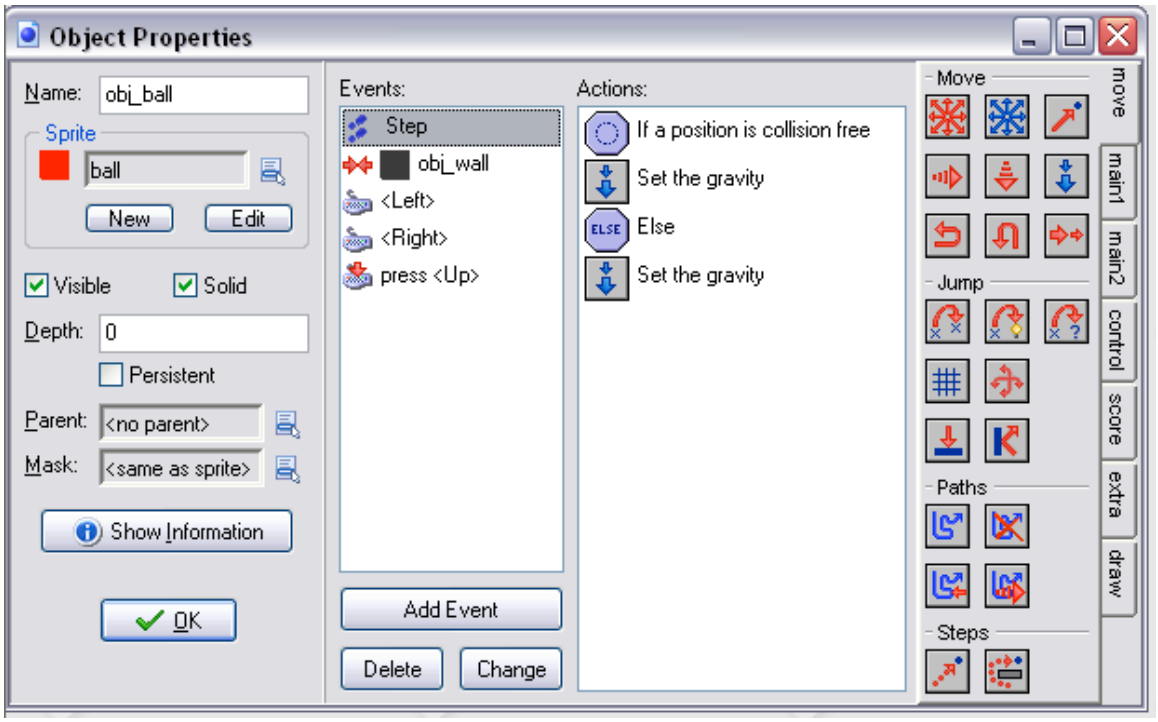
However, this doesn't take into account whether there's anything to the left that might get in our way.

- Find the **Check Empty** action from the **Control** pane of the **Actions** tab, and drag it above the **Jump to Position** action we've just added. Again set the **x** value to -4, and tick the **Relative** option again. This means that we'll check whether the position 4 pixels to the left of the character is empty (not solid) before we actually move.
- Repeat this for the right cursor key, but checking and moving in the opposite direction.
- Create an empty room surrounded by walls and place an instance of the character object in it, as we did for the clown game.
- Run the game – make sure that the character moves left and right with the cursors keys, and stops when you approach one of the walls.

Next we'll add jumping and falling, which is a bit trickier. We want the character to jump into the air when we press the up cursor key, but then fall back down automatically with gravity. First let's add the jumping

- Add a **Key Press <Up>** event to the character object
- Add a **Speed Vertical** action to the event, setting the **vert. speed** field to -10. The y (vertical) axis of the room counts down from the top, so negative speed means we fly upwards.

This makes the character fly up into the air, and we're counting on gravity to bring them back down.

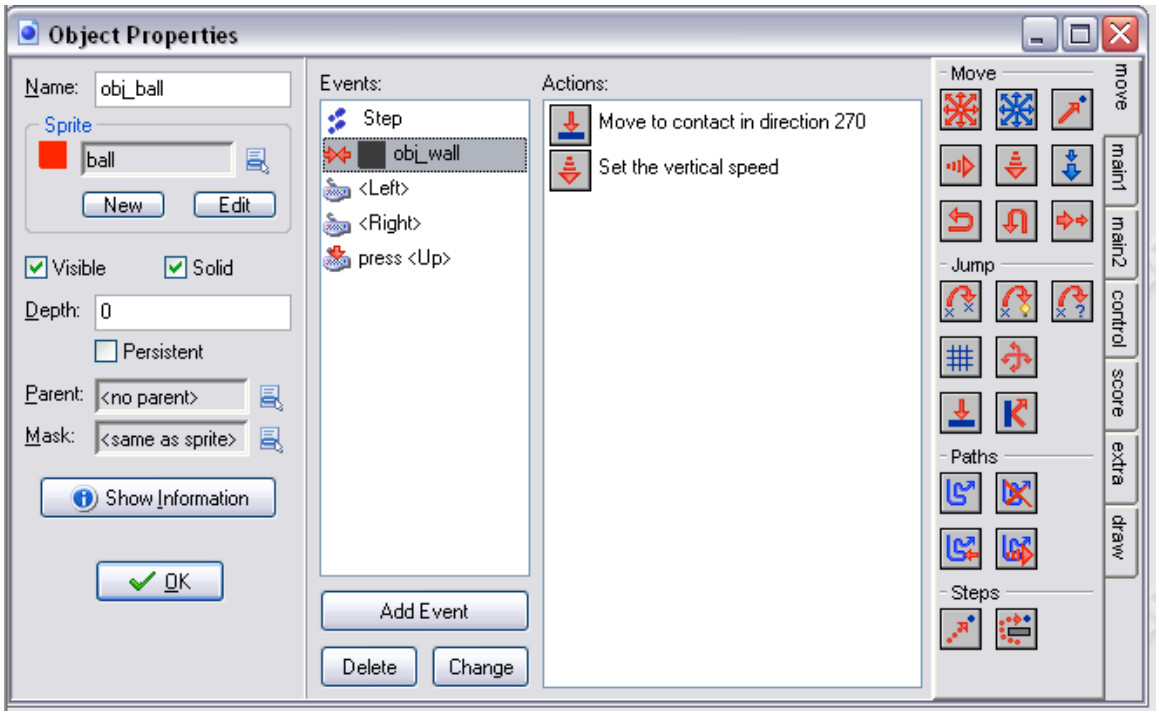


To handle the gravity we need to add some basic conditional logic. If the character is in the air, that is there's nothing underneath them, then we want to apply gravity. If they're not in the air, then we want to stop applying gravity, and stop the downward motion.

First add the gravity:

- Add a **Step** event to the character object. This occurs every step, or frame, that the game is running, so essentially all the time.
- Add a **Check Empty** action again, but this time checking the pixel **x=0**, **y=1** – or the pixel directly below us.
- If it's empty, then we set the gravity, so next add a **Set Gravity** action with **direction** as 270 (270 degrees = straight down), and with **gravity** as 0.5
- Now if it isn't empty, we've reached the platform so we stop applying gravity. Add an **Else** action from the **Control** tab.
- Add another **Set Gravity** action, again with **direction** as 270, but this time with gravity as 0.

The final step is to stop the vertical movement of the character when it hits the platform, otherwise it'll fall straight through it.



- Add a **Collision** event that fires when the character object hits the wall object.
- Add a **Move to Contact** action with **direction** set to 270, and **maximum** set to 12. This makes the character cleanly stop on the platform even when we're moving quickly
- Finally add a **Speed Vertical** action that sets the speed 0.

Extending Game Play

Now that you have implemented the basic movement mechanics of our platform game, the next step is to give the player something to do, and attempt to make an engagine game. For example, challenge the player to collect all of the gold coins in the level, without falling in any of the hazardous pits.

Think about how this could be achieved using **Collision Events**:

- A hazardous pit object uses the **Restart Game** action when the player collides with it
- A gold coin object increases the player's score on collision (Hint: what happens to the gold coin? How could you implement a simple scoring mechanism?)