G54GAM – Lab Session 2

Game Description

Square Shooter

A vertical scrolling shoot-em-up. You control a square flying over square-land. You encounter an increasing number of enemy squares that try to destroy you. You should avoid these or shoot them. Stay alive for as long as possible and destroy as many enemy squares as you can.

Mechanics

The level is formed by a scrolling background with some square islands. The player's square flies over this background. You can shoot bullets that destroy enemy squares. There are four types of enemy squares: the first just flies in a straight line towards you, the second fires bullets straight ahead, the third fires bullets towards the player's square, and the forth is a big boss that fires bullets towards you.

Controls

The player controls their square using the arrow keys. The space key fires a bullet.

Levels

There is only one level, but more and more enemy squares attack the player – starting with the easy ones, followed by more difficult ones

Create the illusion of motion

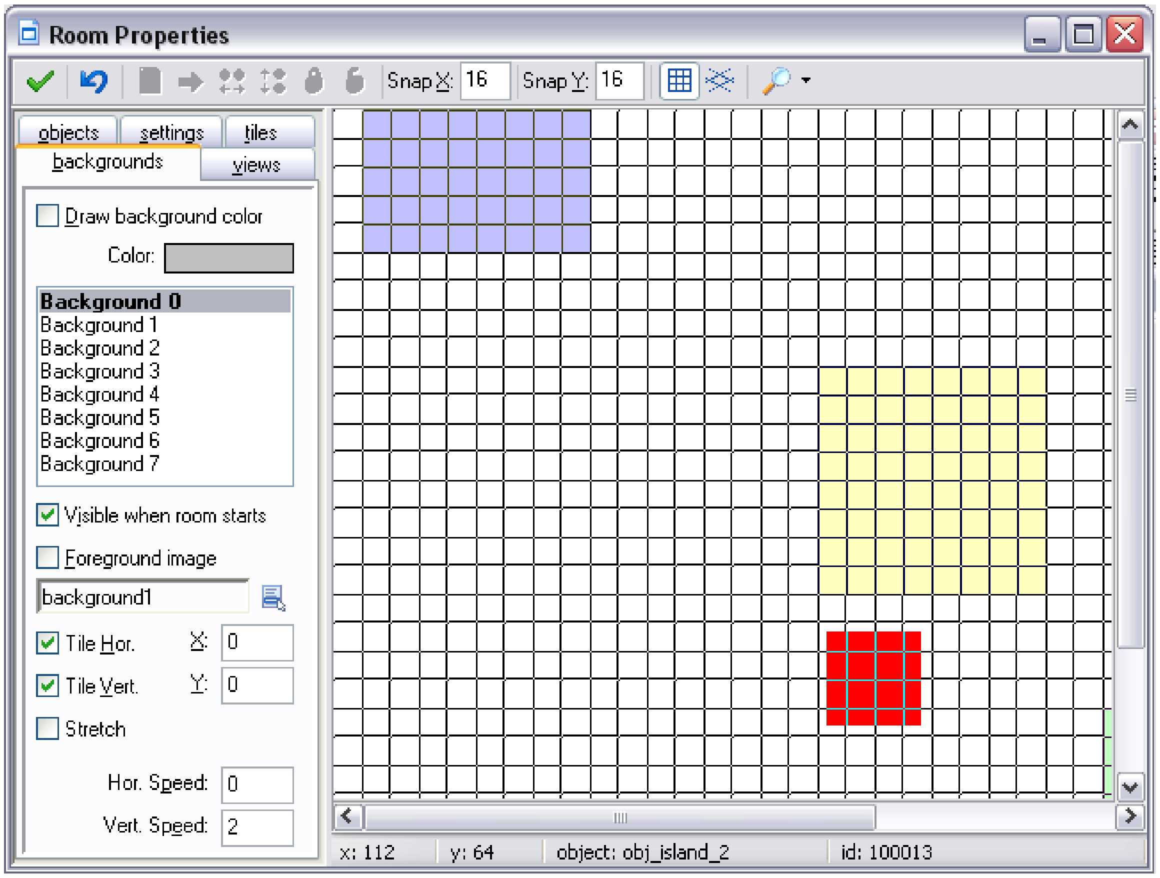
The first task is to create the environment that our hero square flies through. First we need to create resources.

- **Sprites**
  - A **sprite** for the main character square that the player will control
  - 3 island **sprites** that will make the environment look interesting
- **Background**
  - A **background** tile that will scroll to give the illusion of movement

Use the **Add** menu to create these resources, and either use the built-in editor to design some simple sprites or find something on the web. It is very easy to waste a lot of time making these resources look nice so keep them simple, you can always come back and improve them later.

- Create a single **Room** and set the background property to use the background resource that we just created.

- Set the **vert. speed** to 2 – this is the vertical speed at which the background will scroll.



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 [ ]

background1

Tile Hor. [✓] X: 0

Tile Vert. [✓] Y: 0

Stretch [ ]

Hor. Speed: 0

Vert. Speed: 2

x: 112 y: 64 object: obj_island_2 id: 100013

Next we need to add islands to make the scrolling background interesting and less repetitive. We'll use object inheritance to avoid duplicating the functionality. Islands have two actions. They move down the screen at the same speed as the background, and when they've disappeared from the bottom they reappear at the top in a different position.

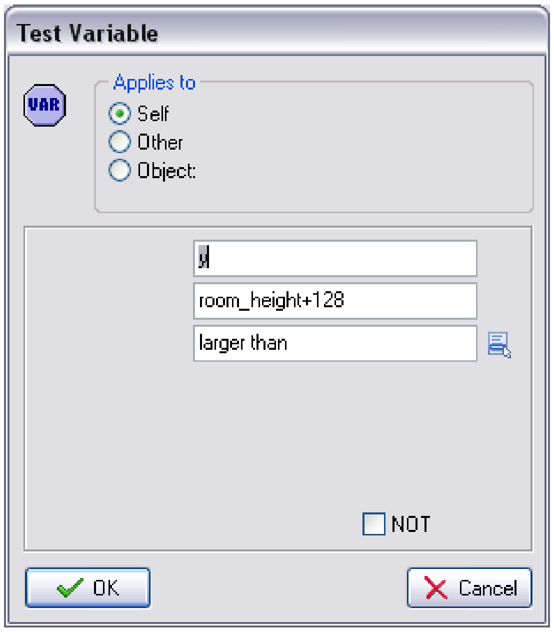
For this we need to a) find out where the islands is, and b) compare it to the size of the room, so we can see if it's gone off the bottom. Each island instance has the following variables that define where it is:

- **x** – the x-coordinate of the instance
- **y** – the y-coordinate of the instance

Remember y increases as the island moves downwards! The game has global variables that tell us the size of the current room:

- **room_width** – width of the room in pixels
- **room_height** – height of the room in pixels

- Create a base **object** for the islands, call it something like obj_base_island. This is just the parent object so it doesn't need to have a sprite.
- Set the **Depth** field to 10000 so the islands appear underneath other objects
- Add a **Create** event to the object, and use a **Move Fixed** action to set it moving down the screen at speed 2
- Add a **Step** event to the object. This is fired every time the game updates the screen, so we can use it to check where the island currently is.
- Add a **Test Variable** action to this event, and check the **y** property of the object to see if it is greater than the height of the room, plus a bit to account for the size of the island itself.


Test Variable

Applies to: [✓ Self] Other Object:

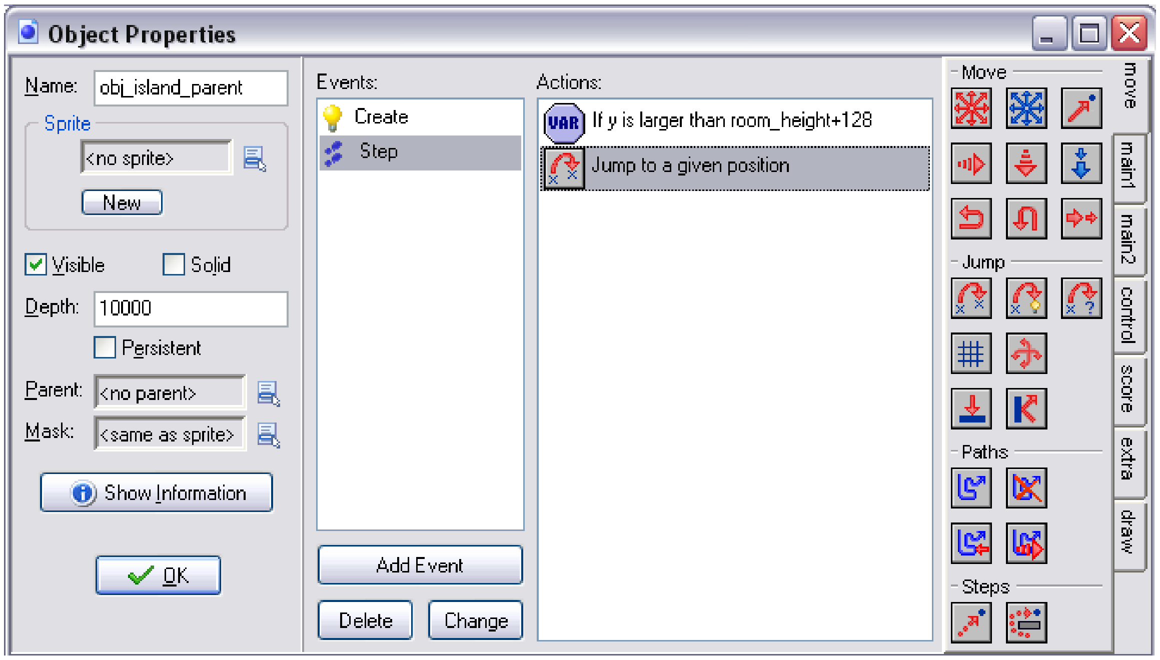
y

room_height+128

larger than

NOT [ ]

- Add a **Jump to Position** action to move the island back to the top of the screen


Object Properties

Name: obj_island_parent

Sprite: <no sprite>

[✓ Visible] Solid

Depth: 10000

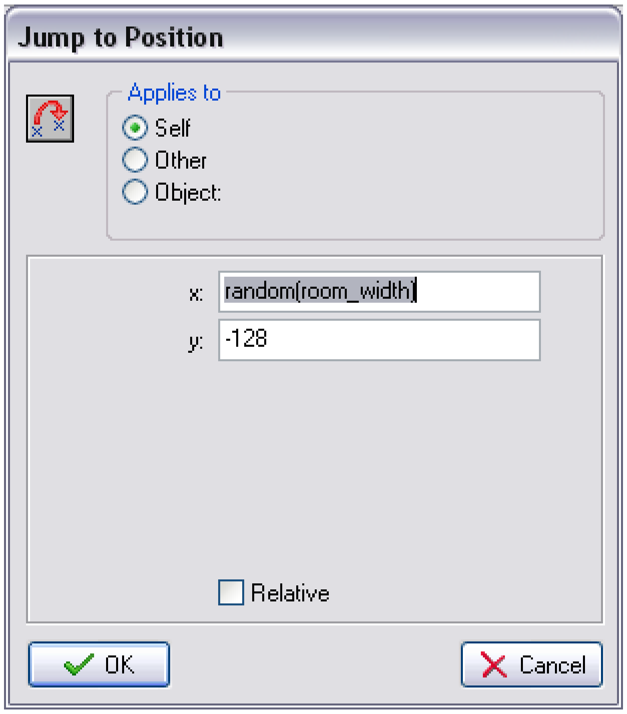
Persistent

Parent: <no parent>

Mask: <same as sprite>

Events:
- Create
- Step

Actions:
- If y is larger than room_height+128
- Jump to a given position


Jump to Position

Applies to: [✓ Self] Other Object:

x: random(room_width)

y: -128

Relative

- Use the **random(room_width)** function to place the island at a random **x** coordinate. This makes the player think that it’s a new island, but we’re just recycling an old one.

Now we’ve sorted out the island’s core functionality, we can make some different island objects to make use of it.

- Create a new object, and set its parent to be the root island object we created earlier. Give it one of the island sprites, and place it somewhere in the room.
- Repeat this process to create two further islands with different sprites

Test the game. Do islands should move to the bottom of the screen and then reappear at the top? Do they move at the same speed as the scrolling background? Do they appear at a random x coordinate each time?

Next create the main character square and allow the player to control it with arrow keys. We'll use the same variable checking mechanism to stop the player flying outside the boundaries of the room.

- Create a new object, assign it the appropriate sprite
- Place an instance of the object somewhere near the bottom of the room
- Add a **Keyboard** event to handle the <Down> arrow being pressed
- Check the **y** coordinate of the object against the **room_height**, should we be allowed to move down?
- If so, use a **Jump to Position** action to move the object down by 4 pixels
- Repeat this for <Up>, <Left> and <Right> arrow, checking against the appropriate **room_height** and **room_width** variables as necessary. Don't forget that the top-left corner of the room is at the coordinates 0, 0.

Test the game again. Do you have a character that the player can fly around with the arrow keys? Do all four boundaries of the room stop the player flying through them?

Bullets and Enemies

Now that we've created an environment, we need to create enemies and bullets to satisfy the premise of the game and to create obstacles and a challenge for the player.

Bullets

Bullets fired by the player appear to come from the main character, fly up the screen and, if they don't hit anything, disappear through the top of the screen.

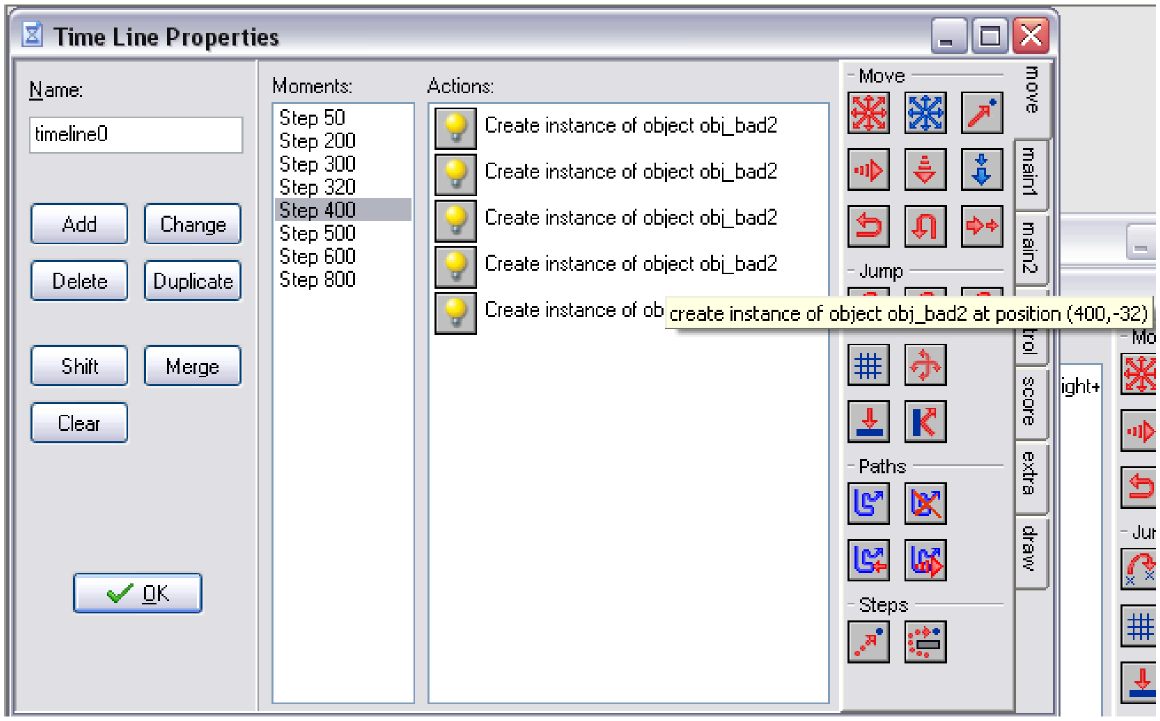
- Create a bullet **sprite** and **object**
- Return to the main character object. Handle the <Space> **Key Press** event, and the **Create Instance** action to create a new instance of the bullet object, remembering to make its position **relative** to the character's current position.
- Handle the bullet object's **Create** event to give it a **Vertical Speed** that makes it move upwards -8.
- Handle the bullet object's **Step** event to check if it's reached the top of the screen
- If it has, use the **Destroy Instance** action to destroy the unused bullet.

Test the game and make sure that the character can fire bullets, and they appear from the right place and move at the appropriate speed. Tweak the **relative** position if necessary to make them appear from just in front of the character. We need to destroy the instance once it is out of sight to avoid wasting memory and resources handling bullets that aren't needed anymore.

Enemies

We've just used the space-bar being pressed to create bullets that fly up the screen. Now we'll use a time line to create waves of enemies that fly down the screen towards the player.

- Create an enemy **sprite** and **object**
- As with the bullet, give it a **Vertical Speed** on creation, this time moving down the screen, and **Destroy Instance** when it gets past the bottom using the **Step** event.



Time lines allow us to schedule events to be fired in a timely manner. Time starts at 0 when the time line is started, and progresses in steps. A step occurs every time the screen redraws, and defaults to 30 steps per second.

- Create a new **Time Line** resource
- Add a **Moment** to the time line to occur at step 50.
- Add a **Create Instance** action at this **Moment** that creates an instance of our enemy object at x = 320 and y = -32. This means that the enemy will start in the middle of the screen but also off the top – note the negative y value. It will then

fly into view, past the player and vanish from the bottom, where it will destroy itself

- Add multiple **Create Instance** actions to the same **Moment** to build up a line of enemies with different starting **x** coordinates.
- Add multiple **Moments** to create waves of enemies that the player has to fight through

Finally we need to start the time line when the game starts

- Create an object with no sprite to act as the time line controller
- Handle the **Create Event**, and use the **Set Time Line** action to start the time line we've just made at position 0.
- Place an instance of the controller object somewhere in the room.

Test the game and ensure that the enemies appear and disappear in the correct manner.

The final step in making our game playable is to make the main character, the bullets and the enemies interact with one another, and for this we'll use collision events and subsequent destruction of instances. The two collision events we need to handle are as follows:

- A bullet collides with an enemy
- An enemy collides with the main character

Remembering to make sure that all of the relevant objects have the **Solid** property set, let's first handle the bullets and enemies.

- Handle the **Collision** event for the enemy with the bullet
- Destroy both the bullet and the enemy object instances. You'll need two actions, one to destroy the **self** (the enemy) and the **other** that it collided with (the bullet)

When the main character collides with an enemy, we're going to just restart the game for now.

- Handle the **Collision** event between the two objects
- Use the **Restart Game** action to send the player back to the beginning.

Test the game. Can the player shoot and destroy the incoming enemies? Does crashing into an enemy restart the game? Try adding more or less enemies and making them fly faster or slower to make the game easier or harder.

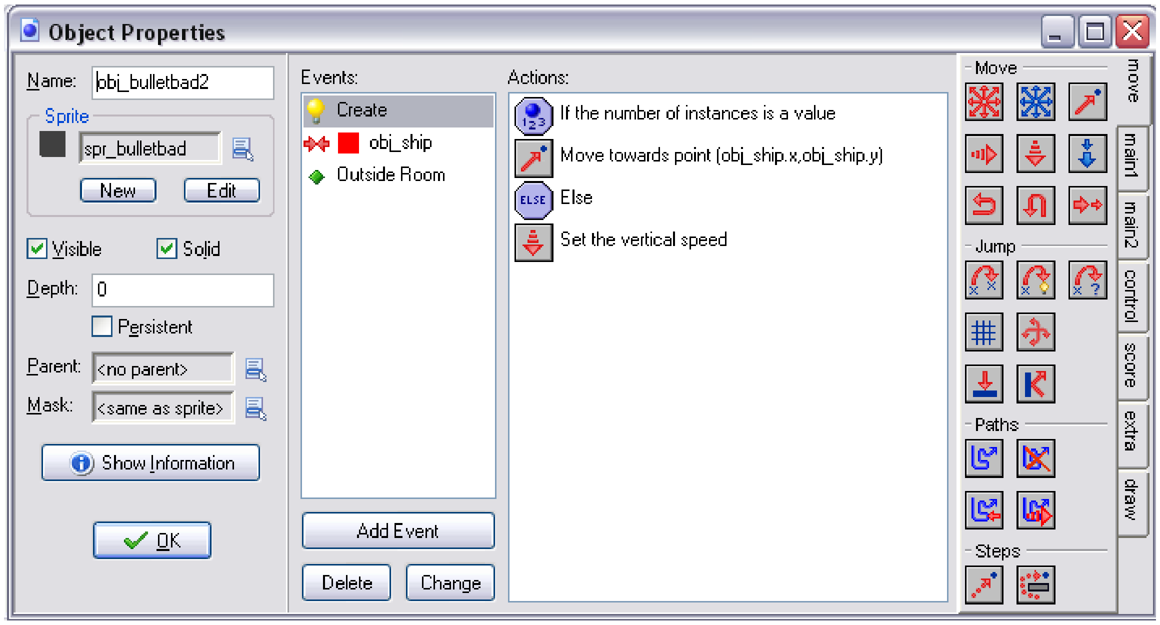
Extending the Game

Now that we've got a playable game we can extend it to make it richer, more challenging and give it longevity.

Better Enemies

Wave after wave of the same enemy isn't that interesting, so create a variety of enemies and place them so that the game gets harder as it goes along

- Create an object hierarchy of enemies with different sprites
- Have different enemy objects move at different speeds
- Create an enemy that fires bullets at you. You'll need to create an enemy bullet object that flies down the screen towards the player. Use the **Test Chance** action with a 1 in 30 chance per step that the enemy instantiates a new bullet object. Handle the collision between the main character and the enemy bullet to restart the game
- Create an enemy that fires targeted bullets. This is trickier, as you need to check if there's a character object to fire at using **Test Instance Count** (otherwise you're risking a crash) and then use the **Move Towards** action to aim at the player.



Name: obj_bulletbad2
Sprite: spr_bulletbad
✓ Visible ✓ Solid
Depth: 0
☐ Persistent
Parent: <no parent>
Mask: <same as sprite>

Events:
- Create
- obj_ship
- Outside Room

Actions:
- If the number of instances is a value
- Move towards point (obj_ship.x,obj_ship.y)
- Else
- Set the vertical speed

- Create a final boss enemy that appears at the top of the screen, then stops and remains in front of the player. Give it a health variable that is reduced every time a bullet hits it, and destroys the instance when the health drops below 0.

Lives, Health and Score

It seems a bit mean to restart the game whenever the player hits an enemy or an enemy bullet. Conveniently Game Maker has built-in support to keep track of **Score**, **Lives** and **Health,** so we can have the player's health reduce when they hit something, lose a life when they run out of health, and end the game when they run out of lives.

- Create an invisible controller object and place it in the room

- In the **Create** event, initialize the score, number of lives and health using the **Set Score, Set Health** and **Set Lives** actions. Also add a **Score Caption** action to show these values in the window's title bar.
- Have the controller object handle the **No More Health** event (in the **Other** event menu) to reduce the life count by one
- Handle the **No More Lives** event to show the scoreboard, using the **Show Highscore** action and restarting the game
- Return to the various **Collision** events between our objects. Instead of restarting the game, use **Set Score** to increase the score and **Set Health** to decrease the health a little depending on what the player has collided with.
- When the boss enemy has been destroyed, display the score board and restart the game.

Test your game with different values for the health decrements, and try and tune the game so it's hard, but not impossible, to reach the end and defeat the boss.