Cosmic Shooter Extended

Cosmic Shooter Part II

My name is Marc Aubanel and I am a video game professional. I worked in the video game industry for 15+ years and have been teaching for the last nine.

This is an exercise where we introduce younger students to computational thinking, simple geometry and a little bit of physics. This can be done with students as young as 12. It is inspired by the classic arcade game Asteroids. I normally do this by live coding in a theater where students come to the computer one at a time and I have the the whole group help solve the problem.

This normally takes anywhere from 40 minutes to 90 minutes. I sometimes purposely create bugs to make the class laugh. This is really about taking students inherit interest or understanding of games and applying them to STEM development practices.

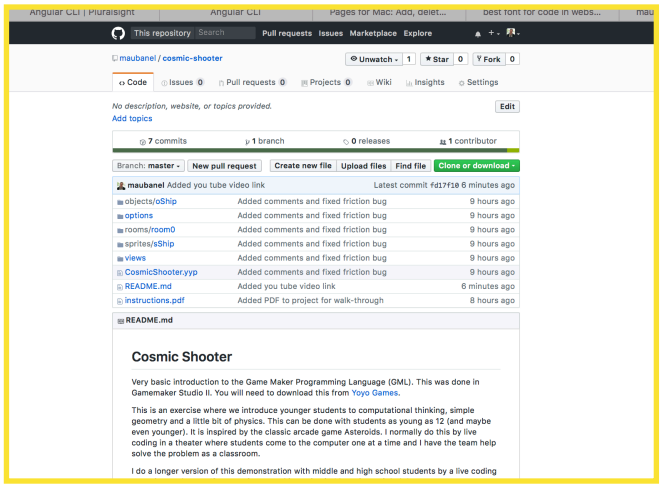
There is no prior experience required and I have the entire project fully solved on GitHub that can be downloaded at: https://github.com/maubanel/cosmic-shooter-extended.

If there are any problems with the PDF please email me at maubanel@cct.lsu.edu.

01

If you have finished Part I, then you can continue from there.

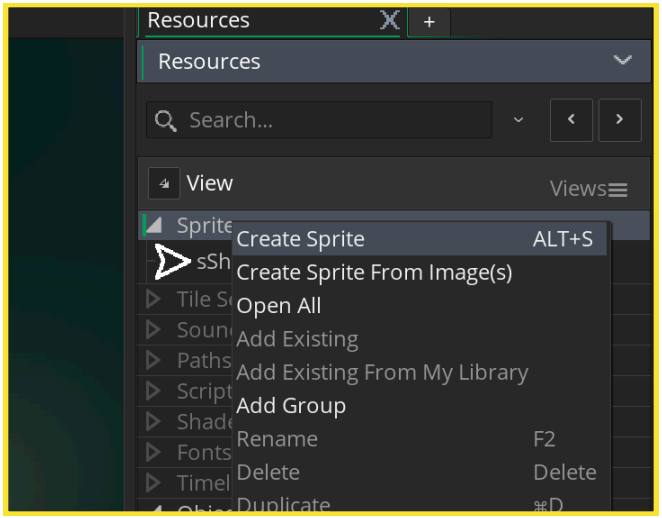
Otherwise you can download the starting point of this exercise at: https://github.com/maubanel/cosmic-shooter



02

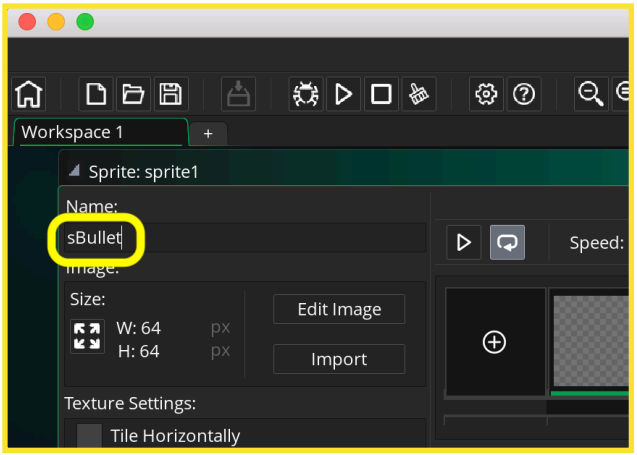
Now it is not much of a Cosmic Shooter without shooting. We want to add the ability to fire bullets.

Lets start by creating a bullet. **Right-Click** the ***Sprites*** menu item under ***Resources*** on the right hand menu bar.



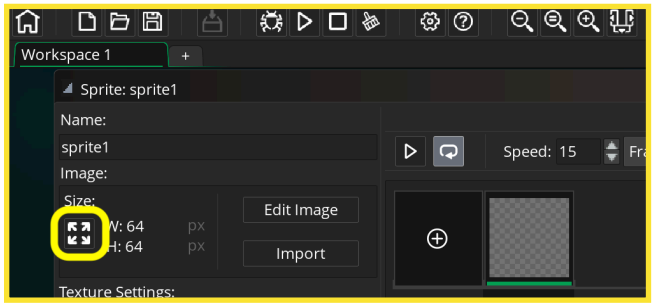
03

Name the sprite **sBullet.**



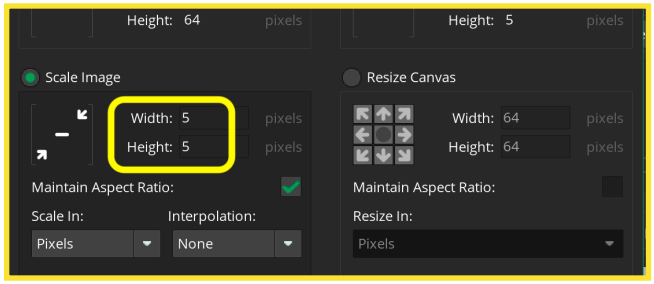
04

Press the four arrow button under ***Size***.



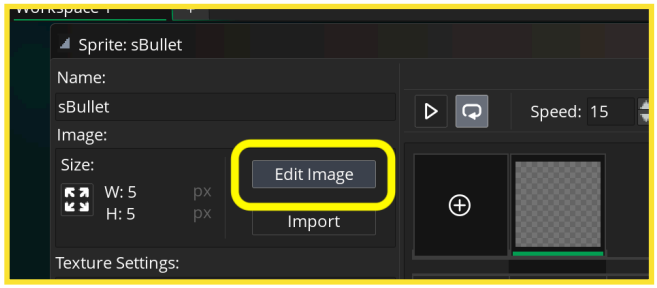
05

Change ***Width*** and ***Height*** to 5 x 5 as we will have a small bullet.



06

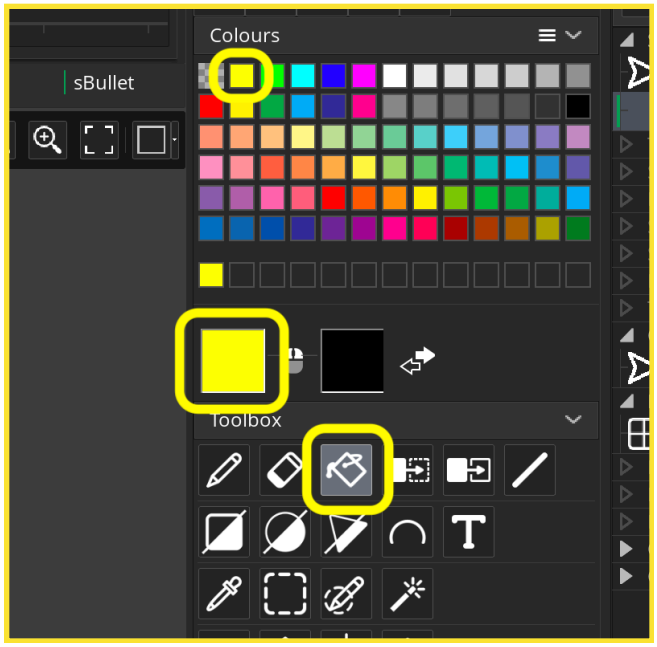
Press the ***Edit Image*** button.



07

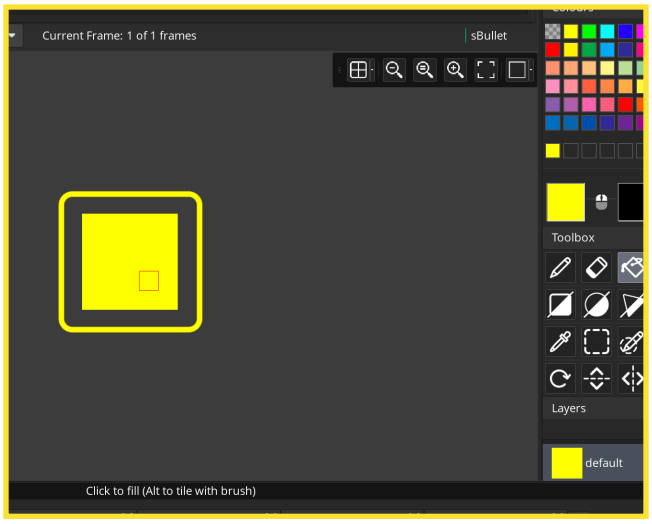
Left-Click on the yellow color under the ***Colours*** menu. This should show up now as the selected color.

Press the **Fill** button (icon that looks like a pouring paint bucket).



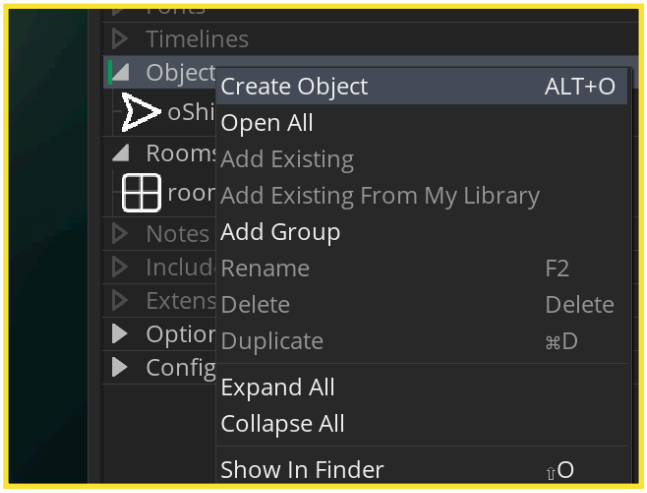
08

Left-Click inside the checker board image area. It will now completely fill with yellow.



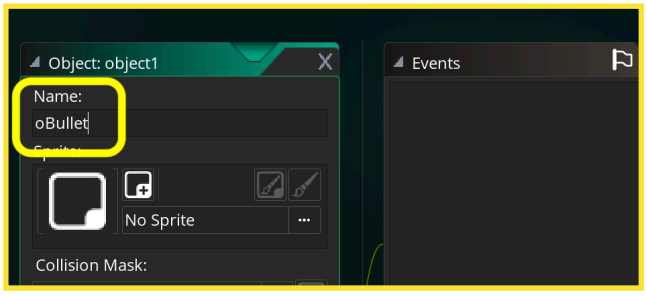
09

Under the ***Resources*** on the right hand side menu **Right-Click *Objects*** and select the **Create Object** menu item.



10

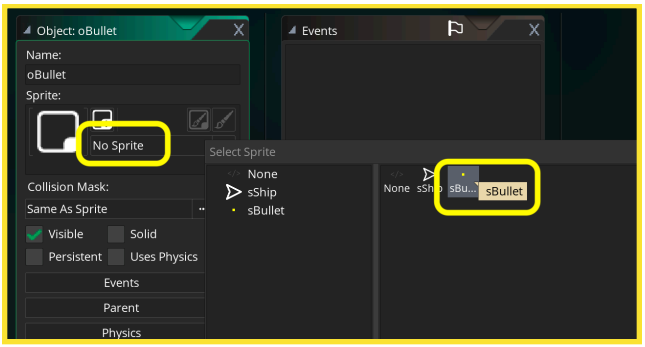
Name the Game Object **oBullet**.



11

Click on ***No Sprite*** and select the **sBullet** sprite and bind it to the Game Object.

Confirm that it works by looking under the ***Resources*** menu and you should see the oBullet should have the yellow icon next to it.



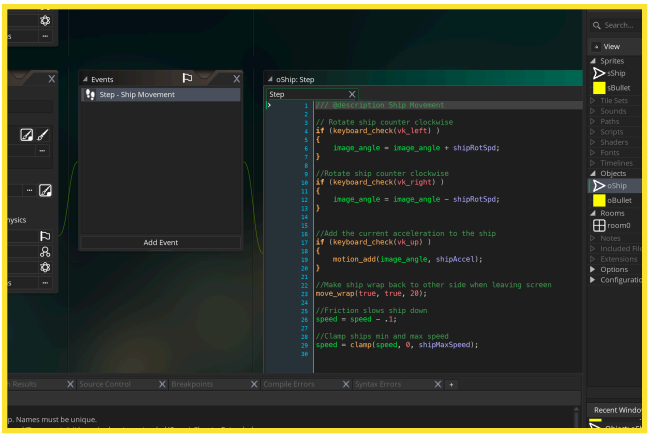
12

Now lets add shooting the oShip Game Object.

Now double click oShip and look at the Step Event script. I am a bit worried about it getting too big.

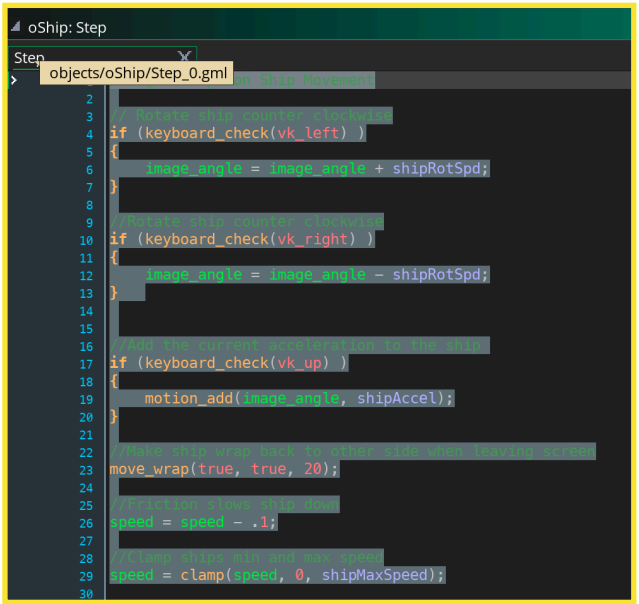
A good rule of thumb is that each script should perform one logical construct. We want to add shooting to this script which is titled movement.

What we can do is we can call other scripts from this event and split it into multiple scripts.



13

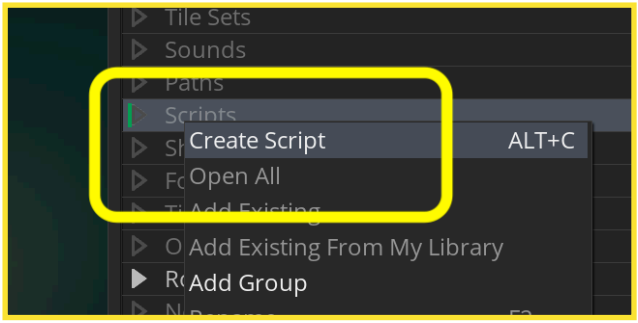
Select all the code in the ***oShip*** Step event script. Press **control C** (or **command C** on the mac) to copy all the text.



14

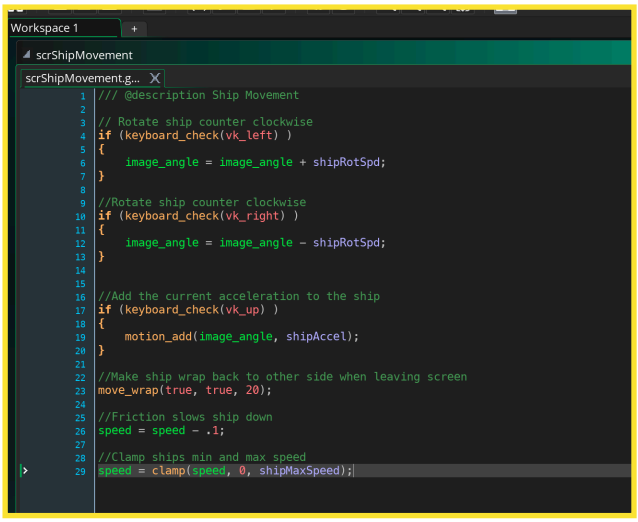
Lets create a new Script by **right-clicking** on ***Scripts*** in the ***Resources*** section in the right hand menu and selecting **Create Script**.

Call the script **scrShipMovement.**



15

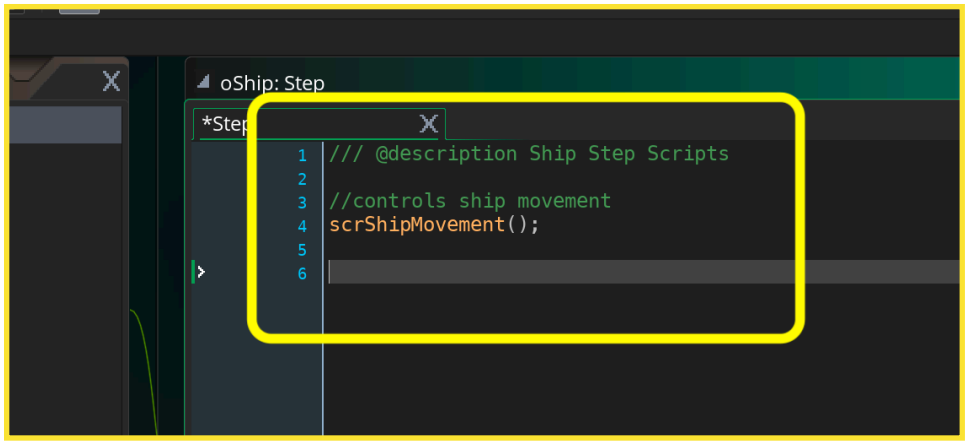
Paste the script into the newly created **scrShipMovement** script by pressing **control c** (or **command c** on the mac)



16

Open the **oShip** Step Event and delete all of the code. Replace it by calling the newly created script. This acts like a function and every frame GameMaker will run the step event script which calls the **scrShipMovement** script:

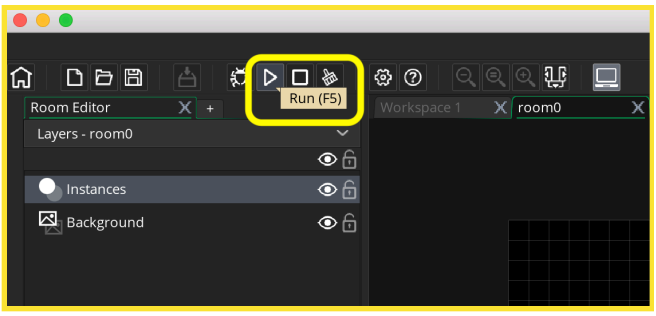
```
/// @description Ship Step Scripts

//controls ship movement
scrShipMovement();
```



17

Run the game and you should have everything working as it was before.

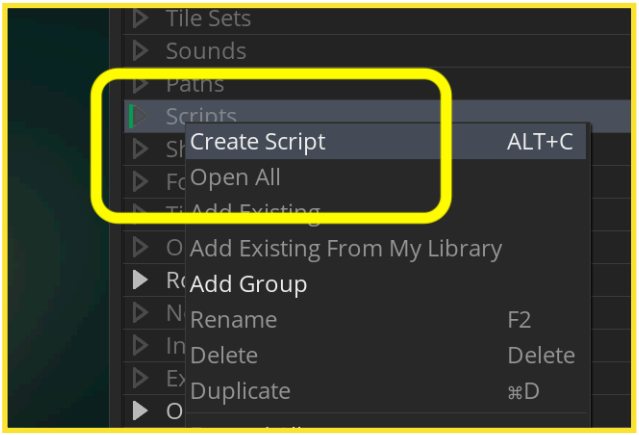
If so you have correctly separated out the movement script



18

Lets create a new shooting script by **right-clicking** on ***Scripts*** in the ***Resources*** section in the right hand menu and selecting **Create Script**.

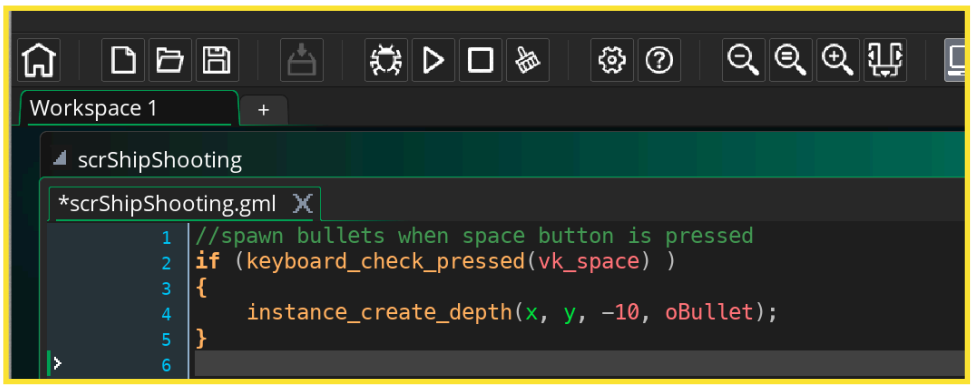
Call the script **scrShipShooting**.



19

How in this script we will check to see if the space button is pressed. Now we will be using k**eyboard_check_pressed (key)** and NOT keyboard_check(key). This is because we don't want to fire a bullet every frame. We want to fire it only once every time the key is hit. The pressed state is only true once every button press. We also want to spawn the bullet behind the frames. So we spawn it at depth -10 which is behind the ship (defaults to depth 0).

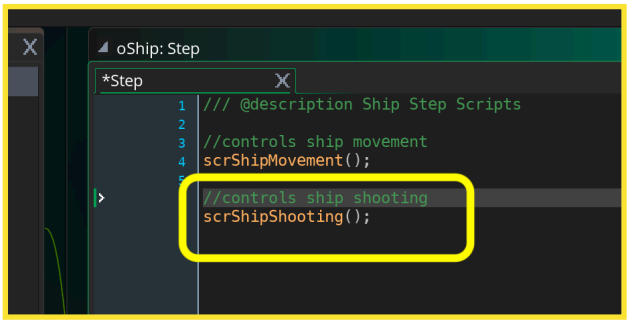
```
//spawn bullets when space button is pressed
if (keyboard_check_pressed(vk_space) )
{
    instance_create_depth(x, y, -10, oBullet);
}
```



20

Open the **oShip** Step Event script and add to the bottom to run the new shooting script:

```
//controls ship shooting
scrShipShooting();
```



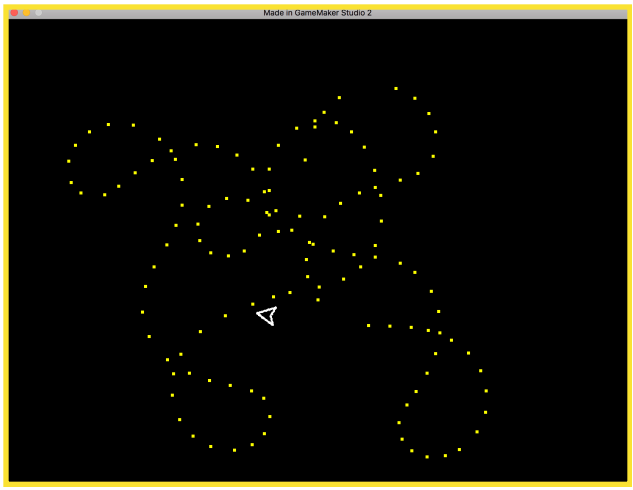
21

Now run the game and press the space bar.

What happens?

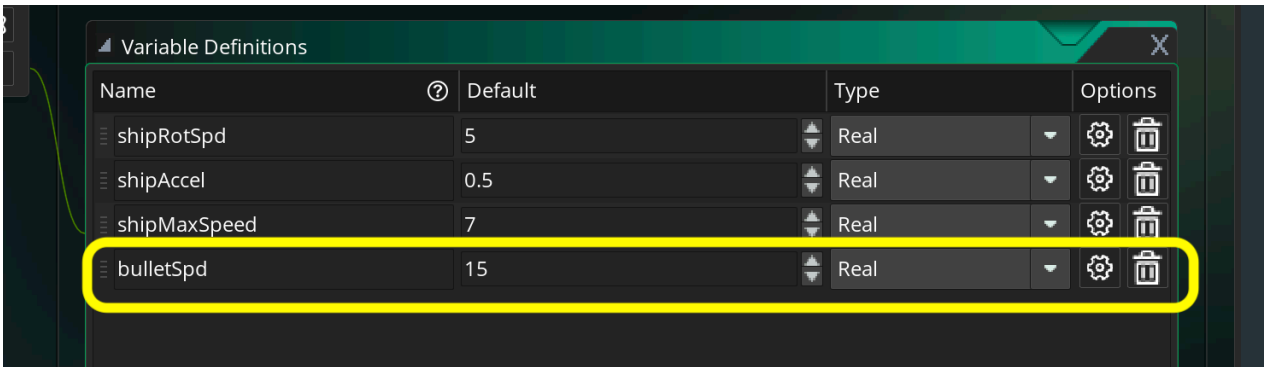
Oh the bullets don't animate so it looks like the plane is laying eggs.

We need to make sure that the bullets fire in the direction the ship is pointing at a a fast speed.



22

Open the ***Variable Definitions*** in oShip and add a **bulletSpd** variable and set it to **15**.



23

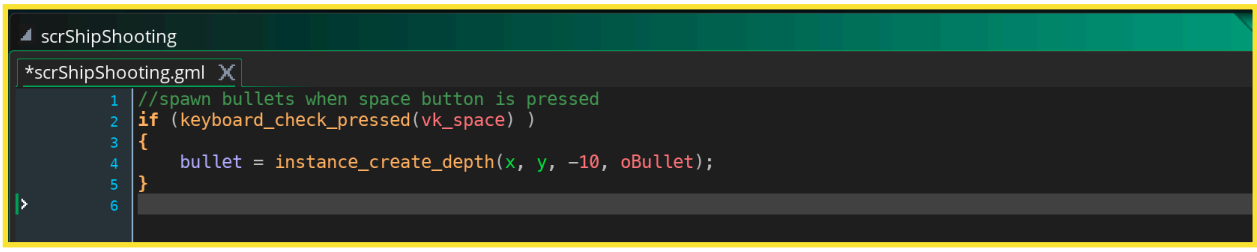
So how do we do this? We are creating the bullet inside the oShip object and we do not have the ability to affect another Game Object's variables, do we?

There is a way. If we look at the documentation for instance_create_depth() (http://bit.ly/2D2k5h7), we see that this function returns the id of the new instance which can then be stored in a variable or used to access that instance.

We can call the instance id that is returned (say bullet) and use the dot to access one of its variables. So we could say bullet.image_angle. This will no affect the image angle of the ship but of the bullet id.

So we can alter the line:

```
if (keyboard_check_pressed(vk_space) )
{
  bullet = instance_create_depth(x, y, -10, oBullet);
}
```

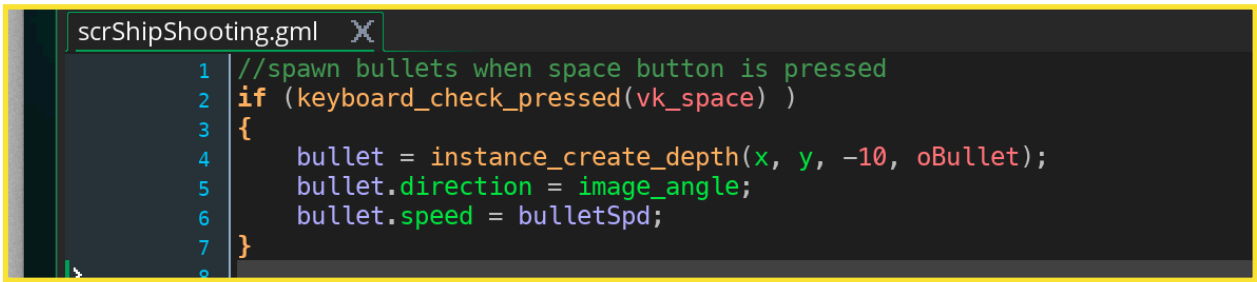


24

So now bullet contains an id to access the bullet object we just created.

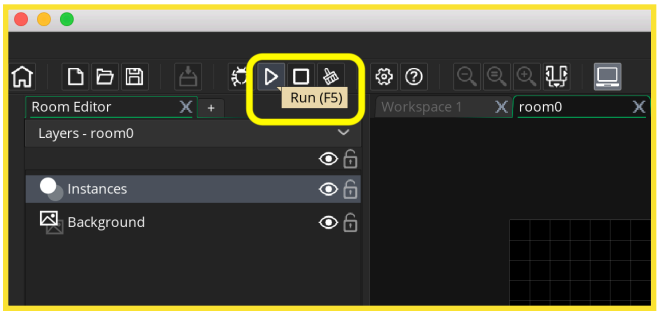
Now we can access that instances space by using the dot (.) notation to access its variables

```
bullet = instance_create_depth(x, y, -10, oBullet);
bullet.direction = image_angle;
bullet.speed = bulletSpd;
```



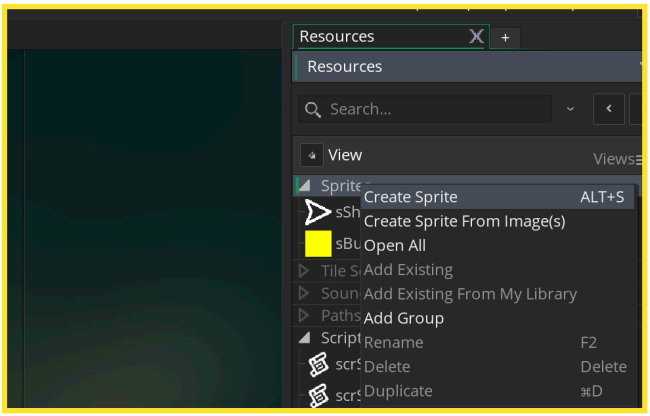
25

Now run the game and press the space bar. Now the bullets should start firing out in the direction the ship is pointing in.



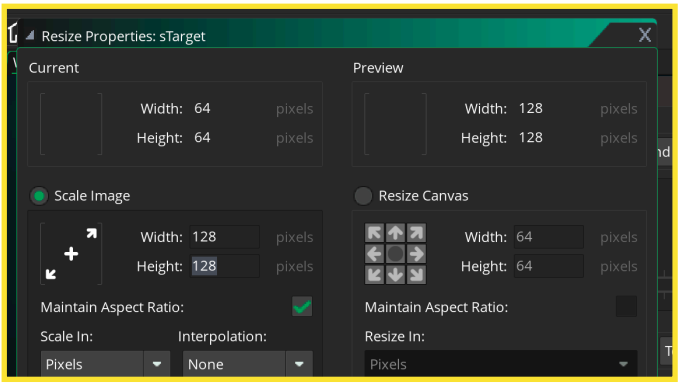
26

Ok, now lets create some targets that the ship can shoot at. **Right-Click** on the ***Sprites*** menu and **press *Create Sprite***.



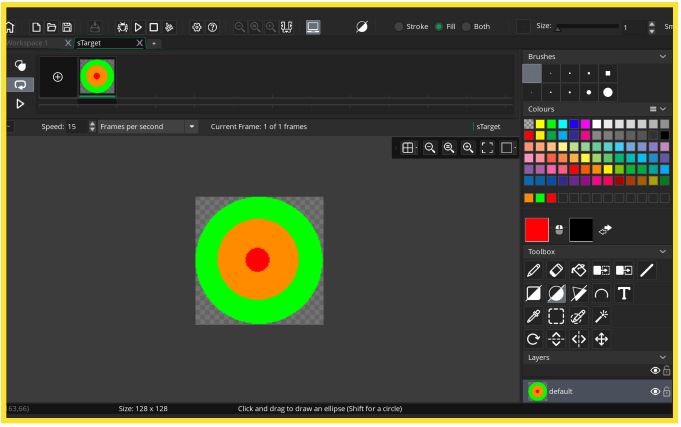
27

Name the new sprite **sTarget** and change the size to 128 by **128** by **pressing** 4 arrows resize button.



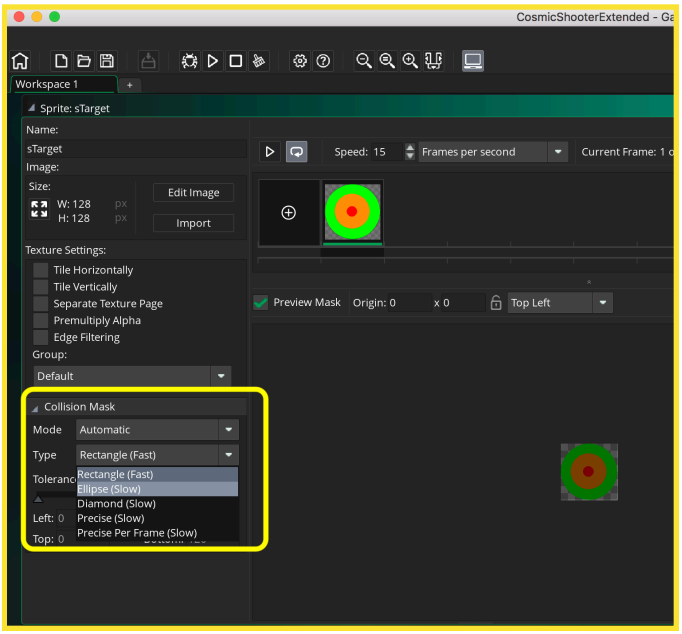
28

Draw a target in any shape and colors that you like. I picked a simple three concentric circle target.



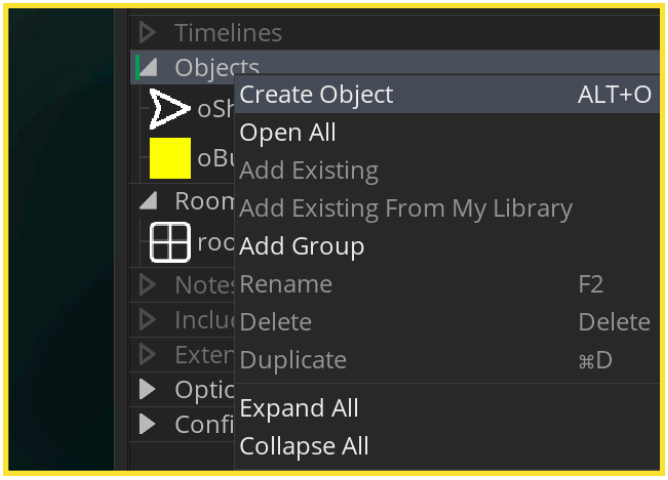
29

Now the collision volume for the shape defaults to a square. Since my shape is circular I am going to change the ***Collision Map*** to an ***Ellipse***. It automatically shapes it to the circle.



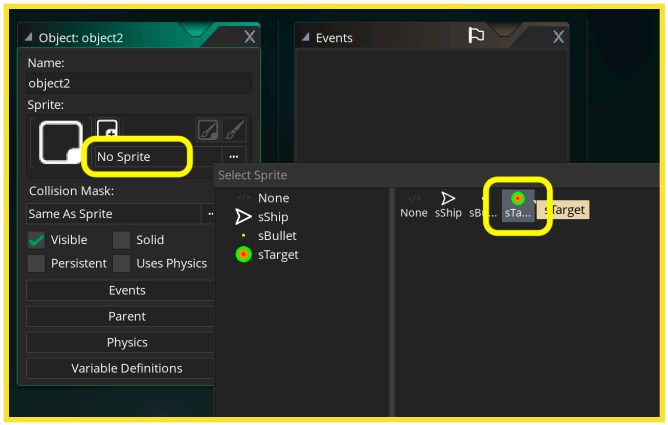
30

Now we need to make a Game Object to hold the sprite. **Right-Click** in the ***Resources*** menu the Objects and press ***Create Object***.



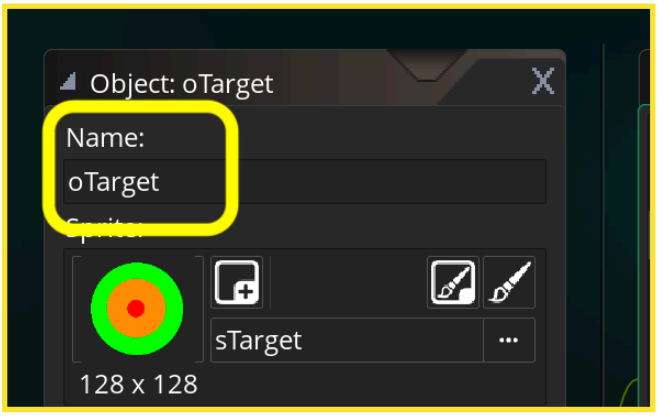
31

Now press No Sprite and select sTarget to bind it to the Game Object.



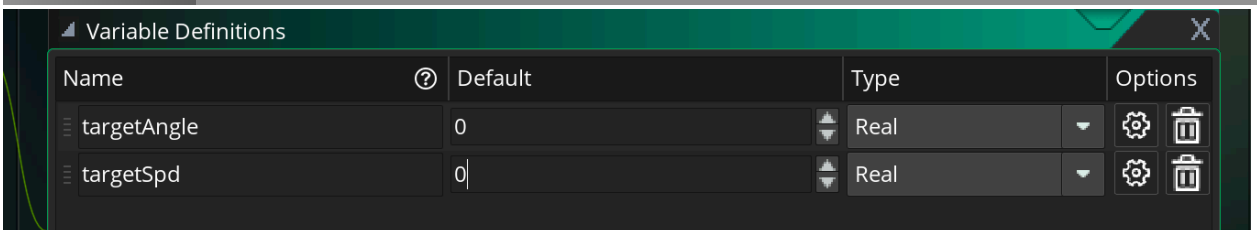
32

Name the object oTarget.



33

Add two ***Variable Definitions*** to **oTarget** for **targetAngle** and **targetSpd** set to **0**.

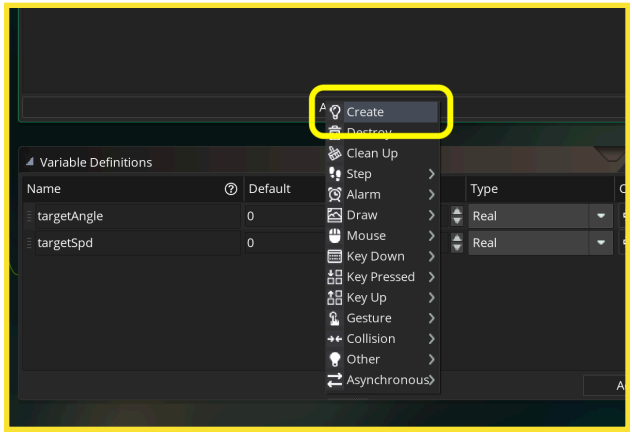


34

We only need to set a direction and speed once. If we put it in a step event it will be needlessly set every frame. Since their speed or direction are not changing we need to set it once when the sprite is created.

There is an event for that. **Click** on **oTarget** ***Add Event*** button and select a Create Event.

This event only runs when the object appears in the room.



35

We are now going to use two new functions:

random() and **choose()**.

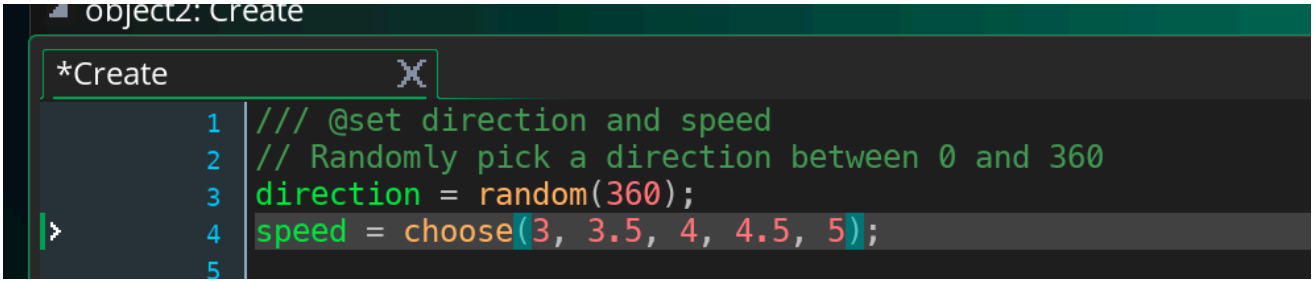
random(num) will create a number between 0 and the num passed.

choose(num, num, num…) will randomly choose a single item passed as a parameter in the parenthesis.

36

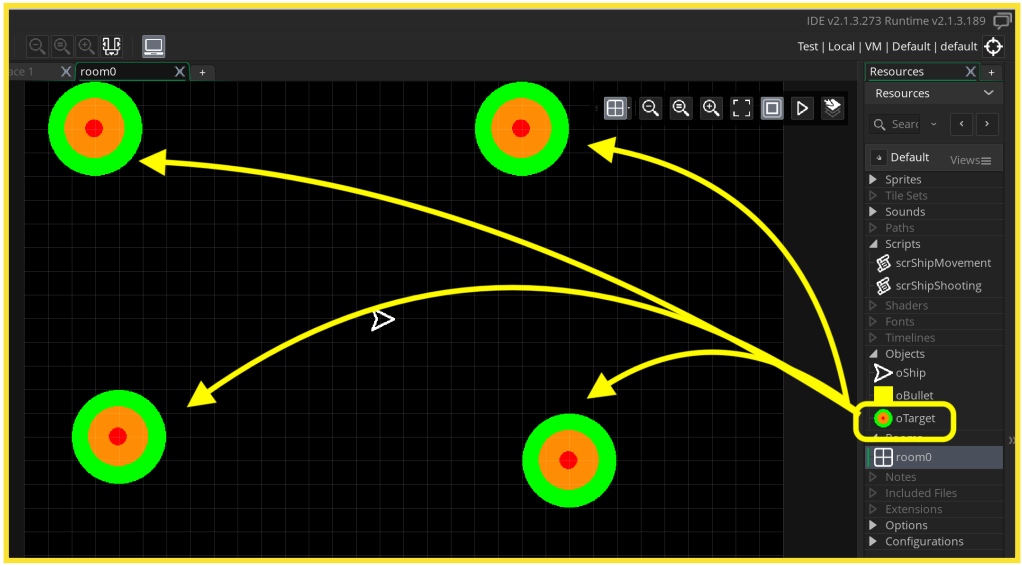
So now in the Create event on **oTarget** type:

```
/// @set direction and speed
// Randomly pick a direction between 0 and 360
direction = random(360);
speed = choose(3, 3.5, 4, 4.5, 5);
```



37

Double-Click the **room0** and drag 4 **oTargets** from the Objects Resources and place them far from the player in the level.

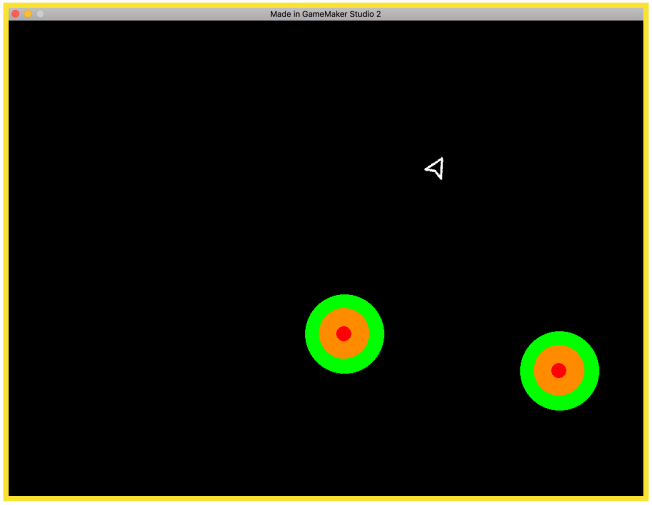


38

Now run the game and you should have 4 targets moving in random directions at different speeds.

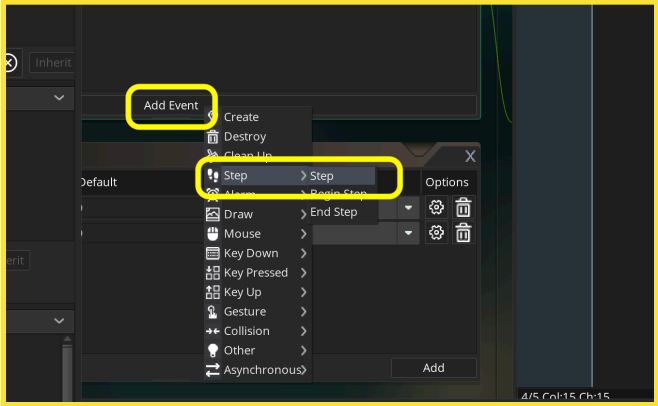
The problem now is that the targets leave the screen.

Lets move wrap the targets but this has to be done in a step event as it needs to check every frame to see if it has left the screen.



39

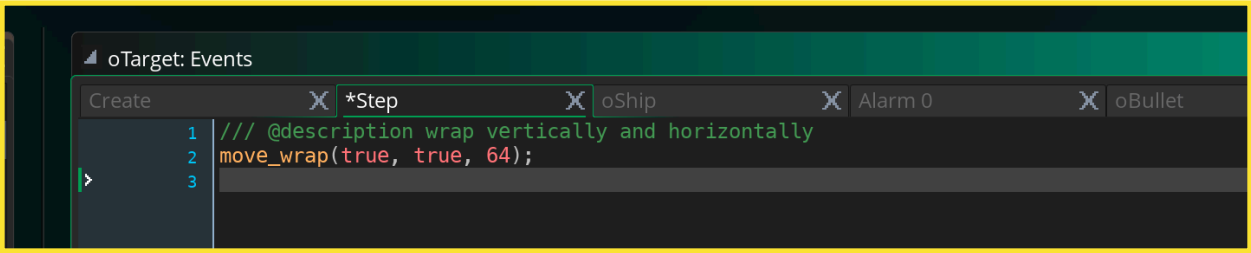
In **oTarget press Add Event | Step | Step.**



40

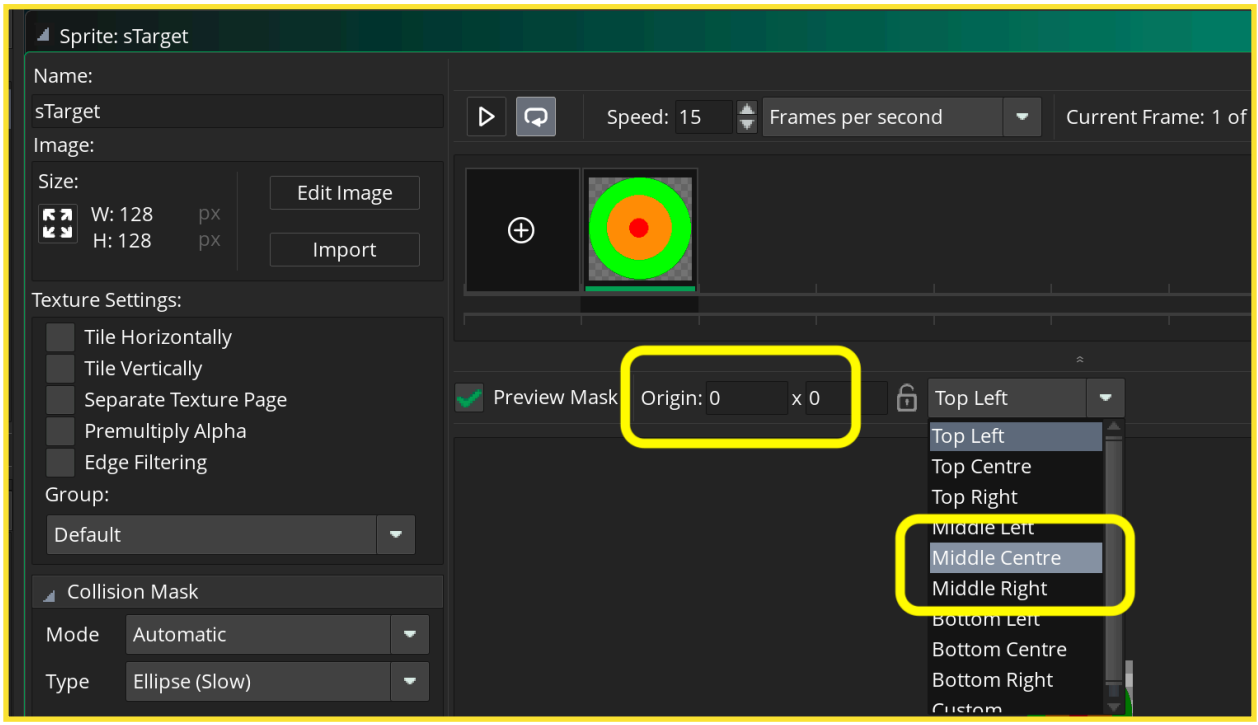
Now lets add to the Step Event the following script to make the targets screen wrap:

```
/// @description wrap vertically and horizontally
move_wrap(true, true, 64);
```



41

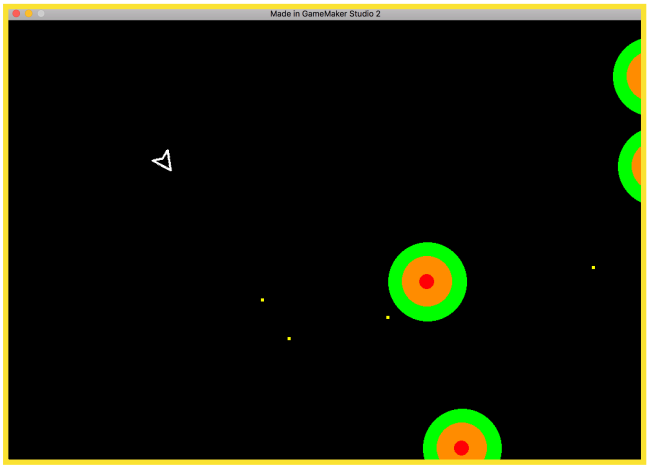
Now **double-click sTarget** and look at its Origin. For screen wrapping to be symmetrical we need to alter the Origin to **Middle Center.**



42

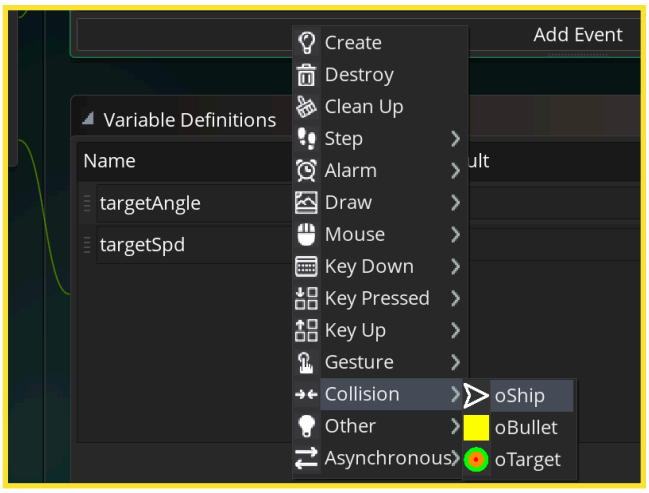
Play the game and now the targets should wrap. But the targets don't damage the player and the bullets don't destroy the targets.

Lets start looking at collisions and have them kill the player.



43

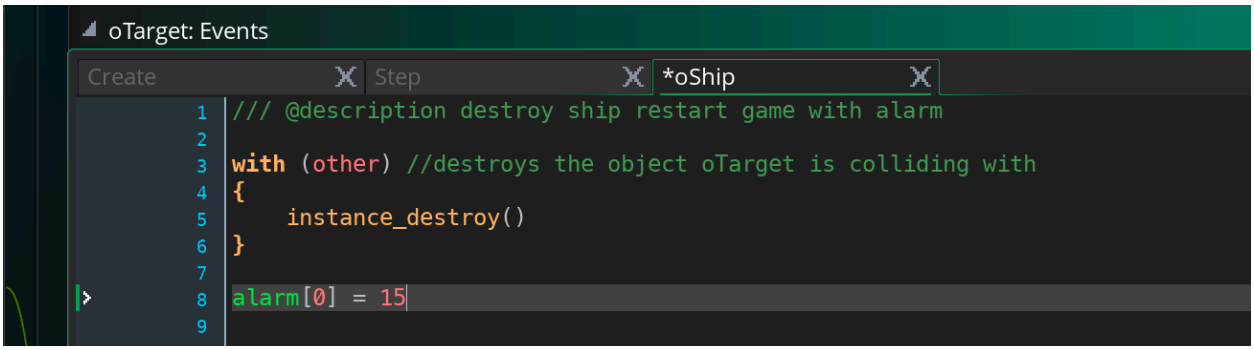
Now **double-click oTarget** and the **press *Add Event*** and select **Collision | oShip**. This script will only run when the oTarget is overlapping oShip's collision volume.



44

In this script type:

```
// @description destroy ship restart game with alarm
with (other) //destroys the object oTarget is colliding with
{
    instance_destroy();
}
alarm[0] = 15;
```



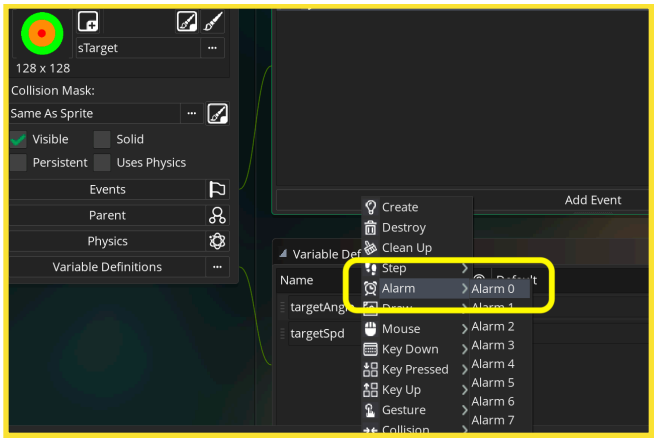
45

We introduced few new concepts. What is **with(other)**? This is unique to Collision Events and allows us to 'enter' the namespace of the object instance we are colliding with. Since the script is on the oTarget we can go into oShip's namespace by calling **with(other)**. Then we **instance_destroy()** the oShip which is self explanatory.

Now what is alarm[0]? This is a trigger that allows us to delay an event. If we ended and restarted the game right away it would be jarring. We are going to let the game continue to simulate without the ship then restart the game.

46

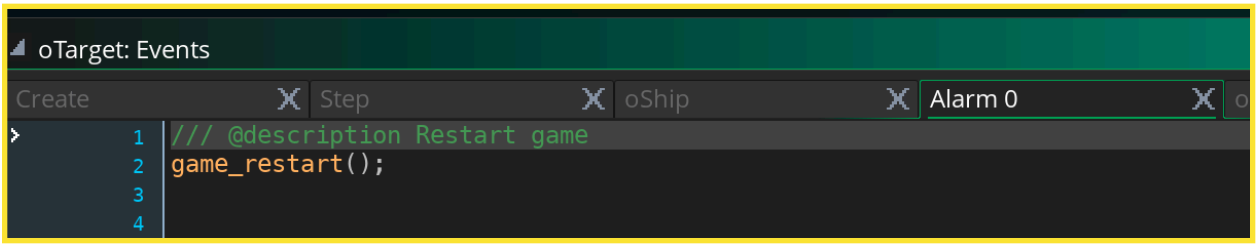
Now lets add an alarm event to **oTarget**. **Press *Add Event*** and select **Alarm | Alarm 0**.



47

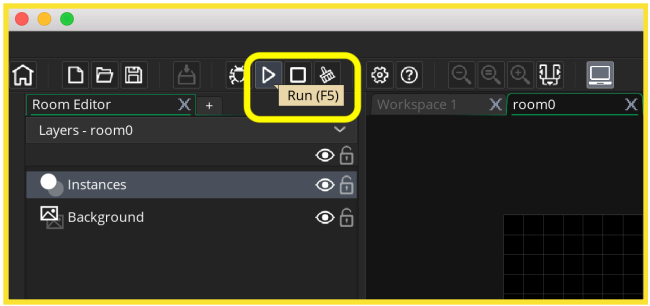
We are going to restart the game in the alarm. Type:

```
/// @description Restart game
game_restart();
```



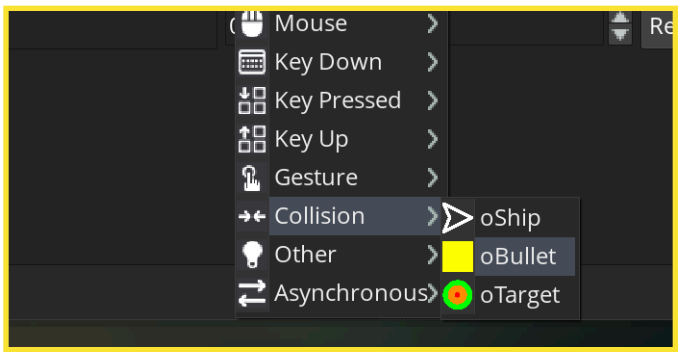
48

Run the game and have the target hit the player. It should destroy them and then half a second later restart the game.



49

Lets now add collision for the bullets hitting the target and destroying it. Go to **oTarget** and **click** on the ***Add Event*** button and add a **Collision | oBullet** event.



50

Now we need to destroy the bullet and the ship. We use the with (other) to destroy the bullet. Now we don't want to necessarily destroy the target. What if the player is killed by a target, then it is hit with a bullet. The alarm would never trigger as it would die with the target and the game would never restart and effectively hang. So what we are going to do is add a rule that to destroy the target the oShip must exist. We use the **instance_exists (object)** for that. Now add:

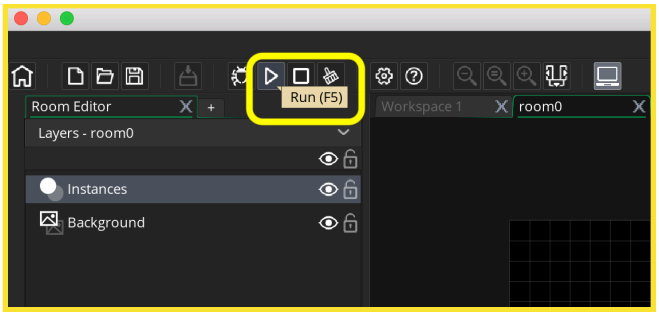
```
/// @description Destroy bullet
 //destroy bullet
 with (other) // destroys bullet
 {
     instance_destroy();
 }
 // Make sure there still is a player in the room.
 if (instance_exists(oShip) )
 {
     instance_destroy();
 }
```

Create | Step | oShip | Alarm 0 | *oBullet

```
1  /// @description Destroy bullet
2
3  //destroy bullet
4  with (other) // destroys bullet
5  {
6      instance_destroy();
7  }
8  // Make sure there still is a player in the room.
9  // If we destroy the target before the alarm is set once the player is killed the game
10 // will never reset.
11
12 //play sound when target is hit with bullet
13 if (instance_exists(oShip) )
14 {
15     instance_destroy();
16 }
17
```

51

Run the game and shoot at the targets. They should now get destroyed when hit by bullets.



52

Now the game is way too short. You kill the two targets right away. What we will do is add logic to spawn new targets when there are fewer than one left.

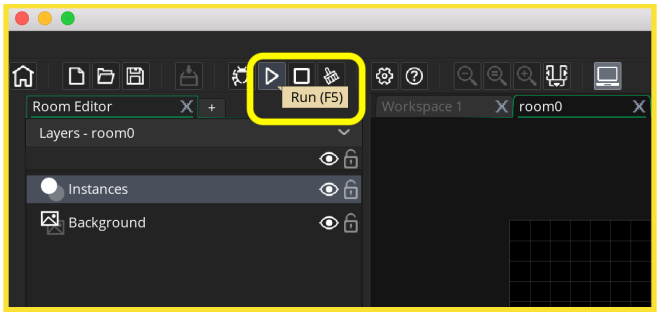
Add to the collision event between oTarget and oBullet and use a function called **instance_numb:er(obj)**. This function will return the number of instances left in the room. So in this case if there are less than 1 left after destroying it, then spawn another target at depth 0.

```
 //If the number of targets gets less than 2 spawn another
bullet offscreen
if (instance_number(oTarget) <= 1)
 {
     instance_create_depth(0, 0, 0, oTarget);
 }
```

```
// will never reset.

//play sound when target is hit with bullet
if (instance_exists(oShip) )
{
    instance_destroy();
}

//If the number of targets gets less than 2 spawn another bullet offscreen
if (instance_number(oTarget) <= 1)
{

    instance_create_depth(0, 0, 0, oTarget); //spawn new rock off screen
}

```

53

Run the game and shoot at the targets. Now they respawn all the time. The problem is that they always respawn in the same area.



54

So we want to spawn it either off the screen on the left/right or off the screen on the top/bottom. Take a look at what I am doing here:

```
if (instance_number(oTarget) <= 1)
{
	targetX = -64; //temporarily create new variables
	targetY = -64;

	vertOffScreen = random(2);
	if (vertOffScreen > 1)
	{
		targetX = choose(-128, room_width + 128);
		targetY = random(room_height);
	}
	else
	{
		targetX = random(room_width);
		targetY = choose(-128, room_height + 128);
	}
	instance_create_depth(targetX, targetY, 0, oTarget);
}
```

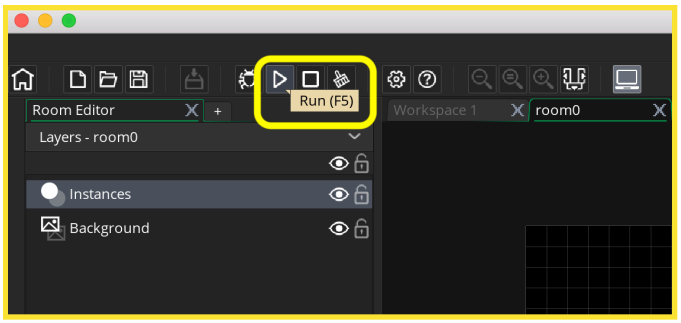
We create two new variables targetX and targetY. We set them off screen. Then we roll the dice to find out if it goes off vertically or horizontally. If it rolls above one we set it off screen to the left or right on targetX and then anywhere along the Y axis within the room_height.

If we roll less than one then we set the targetY off the top or bottom fo the screen and randomly along the x.

```
11
12  //play sound when target is hit with bullet
13  if (instance_exists(oShip) )
14  {
15      instance_destroy();
16  }
17
18  //If the number of targets gets less than 2 spawn another bullet offscreen
19  if (instance_number(oTarget) <= 1)
20  {
21      targetX = -64; //temporarily create new variables
22      targetY = -64;
23
24      vertOffScreen = random(2); //roll dice. Above one means horizontally off screen, beow is vertical
25      if (vertOffScreen > 1) //rolls ver
26      {
27          targetX = choose(-128, room_width + 128); //off screen either on the left or right hand side
28          targetY = random(room_height); //vertical random offset
29      }
30      else
31      {
32          targetX = random(room_width);  //horizontal random offset
33          targetY = choose(-128, room_height + 128); //off screen either on the top or bottom
34      }
35      instance_create_depth(targetX, targetY, 0, oTarget); //spawn new rock off screen
36  }
37
```

55

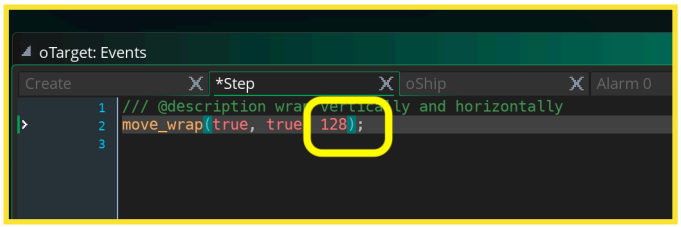
Run the game and shoot targets. Now they are more randomly distributed.



56

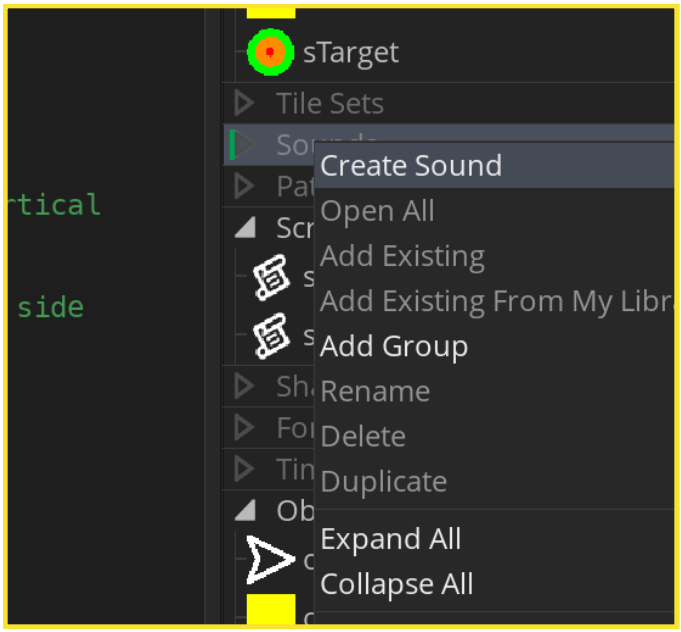
Open the oTarget Step Event script and change the margin in move_wrap to **128**.

This is a little tweak that will make it appear off screen and create a bit more drama.



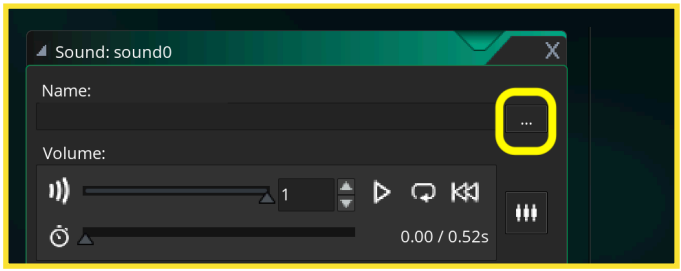
57

Lets add some sound to the game to finish up. **Right-Click** under ***Resources*** the ***Sounds*** menu item and **press Create Sound**.



58

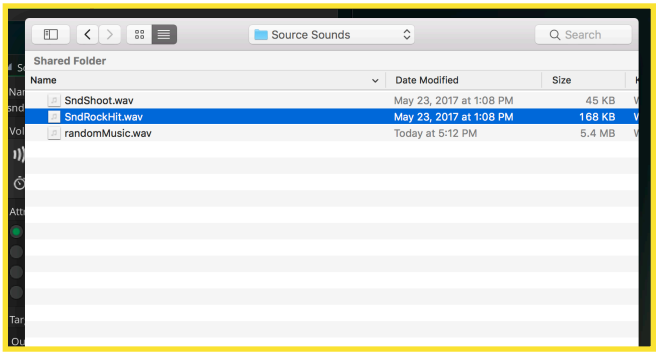
Now press the ***three ellipses*** to open the menu to select a sound.



59

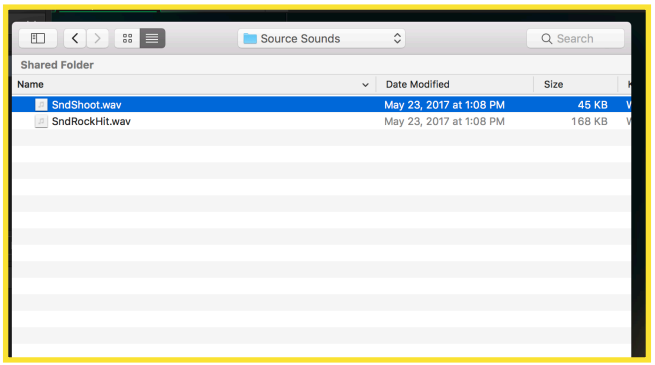
Navigate to the ***Source Sounds*** folder that is included with this project. **Select** the sound called **SndRockHit.wav.**

Hit play and make sure it sounds like the hit sound. Call the file **sndHit**.



60

Create another sound and load **SndShoot.Wav**. Call the sound **sndShoot**.



61

The function **audio_play_sound(index, priority, loop)** plays audio of name index, with a priority which we can set to 1. This is in case you are playing so many sounds at once it prioritizes them in case the device can't play that many. In this game we don't worry about this as we have only 3 sounds. We will also set loop to false. Lets add to oTarget collision event with oBullet and add on the line after we destroy the target:

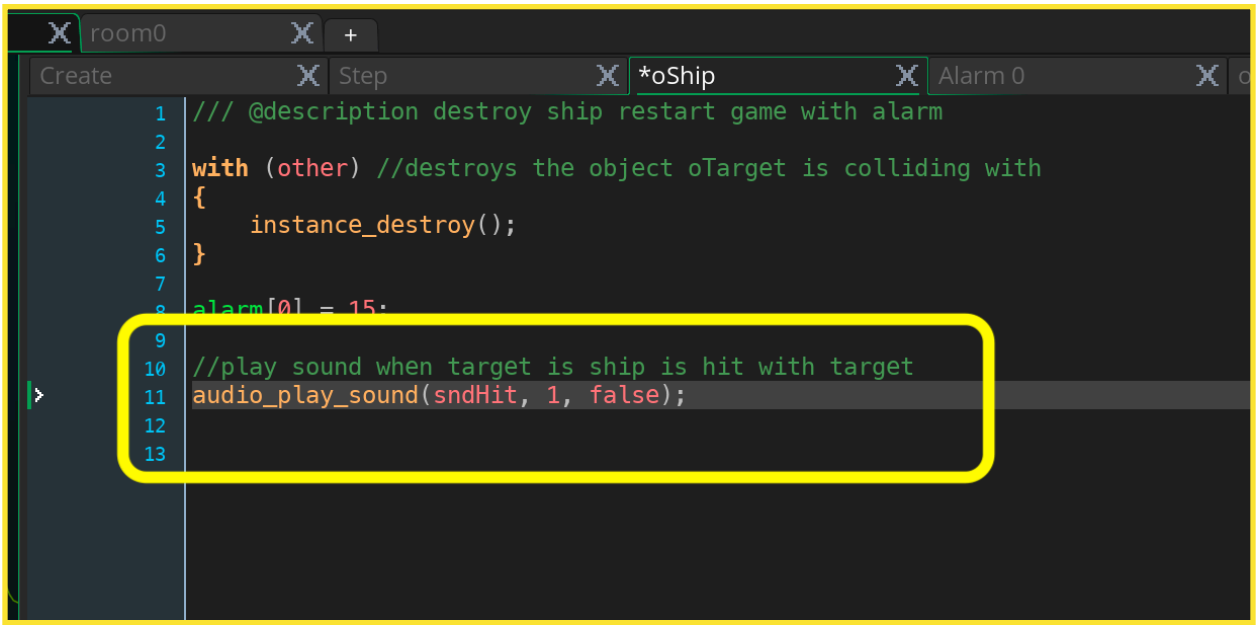
```
//play sound when target is hit with bullet
if (instance_exists(oShip) )
{
     instance_destroy();
     audio_play_sound(sndHit, 1, false);
}
```

```
8  // Make sure there still is a player in the room.
9  // If we destroy the target before the alarm is set once the player is
10 // will never reset.
11
12 if (instance_exists(oShip) )
13 {
14     instance_destroy();
15     audio_play_sound(sndHit, 1, false);
16 }
17
18 //If the number of targets gets less than 2 spawn another bullet offsc
19 if (instance_number(oTarget) <= 1)
20 {
21     targetX = -64; //temporarilly create new variables
```

62

Now lets use the same sound when the target hits the player. Lets add to the bottom of the **oTarget** collision script with **oShip**:

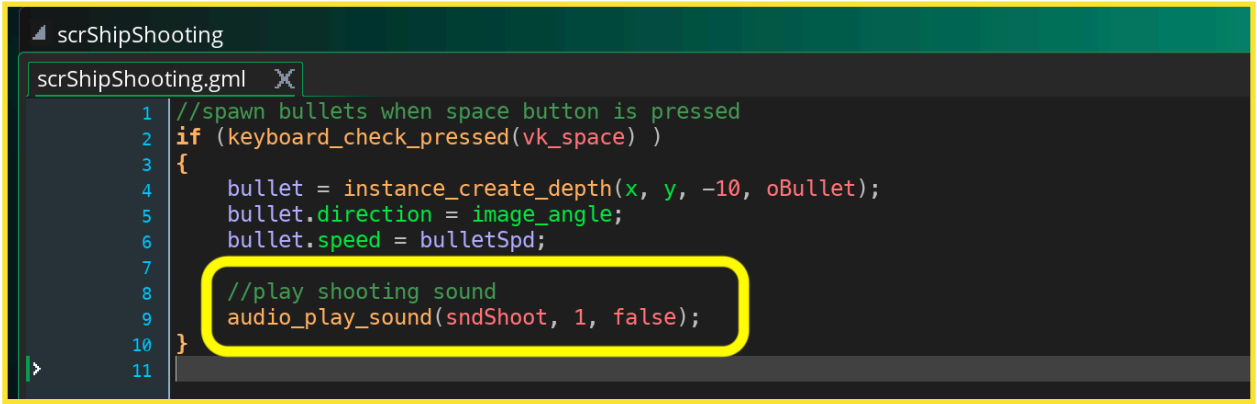
```
//play sound when target is ship is hit with target
audio_play_sound(sndHit, 1, false);
```



63

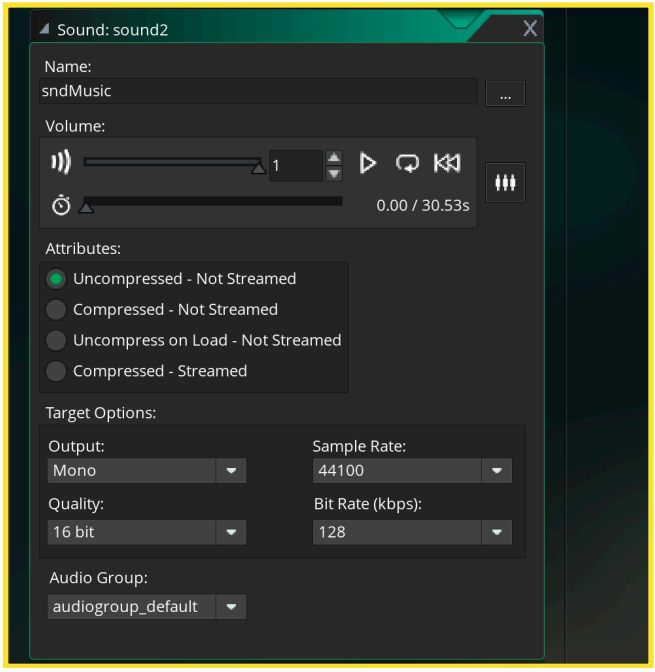
Now lets add shooting sound. **Double-Click scrShipShooting** and add before the last curly brace:

```
    //play shooting sound
    audio_play_sound(sndShoot, 1, false);
}
```



64

Lets add another sound file and this time **load randomMusic.wav**. Call the file **sndMusic**.

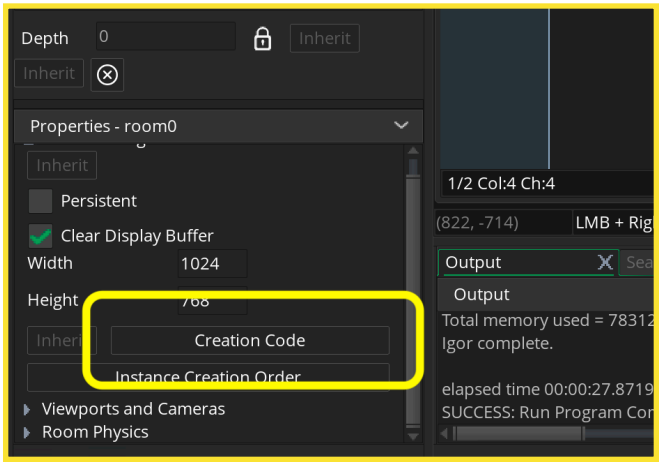


65

Now we **double click** the **room0** and look at the properties at the bottom left.

There is a button that is called Creation Code. This can be used to run a script when the room is loaded up.

This is a perfect place to play our music.



66

We play the sound like previously but this time we do want the music to loop.

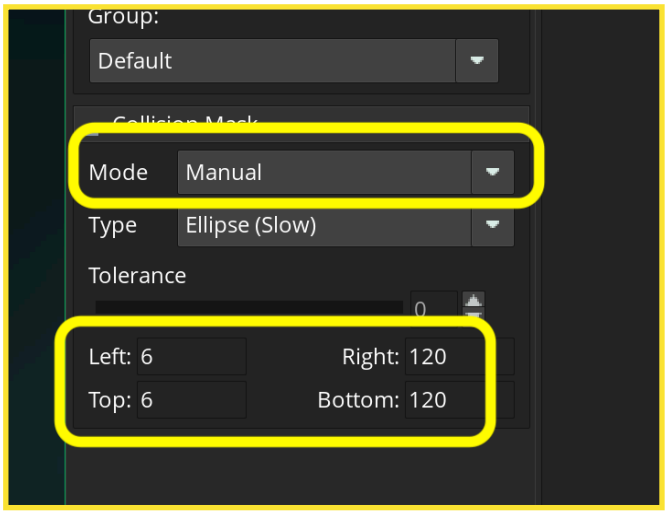
```
// play music
audio_play_sound(sndMusic, 1, true);
```

67

Now play the game and there is one final tweak I want to make.

The collision volume feels like it triggers a bit too soon. I want to make it a bit smaller than the sprite.

Open **sTarget** and change the ***Mode*** to **Manual** and set **Left** & **Top** to **6** and **Right** and **Bottom** to **120**.



That's it for this exercise. You can see what it should look like at https://www.youtube.com/edit?o=U&video_id=K9-naCAbo7o

—The End