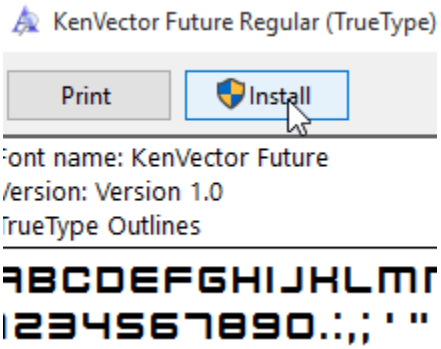
# 1 Space Shooter Guide

## 2 SETUP

Unzip the **SpaceShooterFiles.zip** file to a convenient location.

If you want to use the font provided, double click the **KenVectorFuture.ttf** file and then install it (you have to install it before opening Gamemaker or close and reopen Gamemaker to see it):




Start a new Gamemaker project. Name it **SpaceShooter**. Make sure to pay attention to where you save the project.

***NOTE: THE INSTRUCTIONS IN THIS GUIDE ASSUME YOU HAVE BUILT SHOOTINGGALLERY AND SPACERESCUE. THEY RAPIDLY STEP THROUGH TECHNIQUES AND IDEAS YOU WERE EXPOSED TO IN THOSE GAMES. IT SLOWS DOWN TO COVER NEW IDEAS, BUT THINGS WILL LIKELY BE CONFUSING IF YOU HAVE NOT BUILT THE PREVIOUS GAMES.***

# 3 BACKGROUND, ROOMS AND FONT

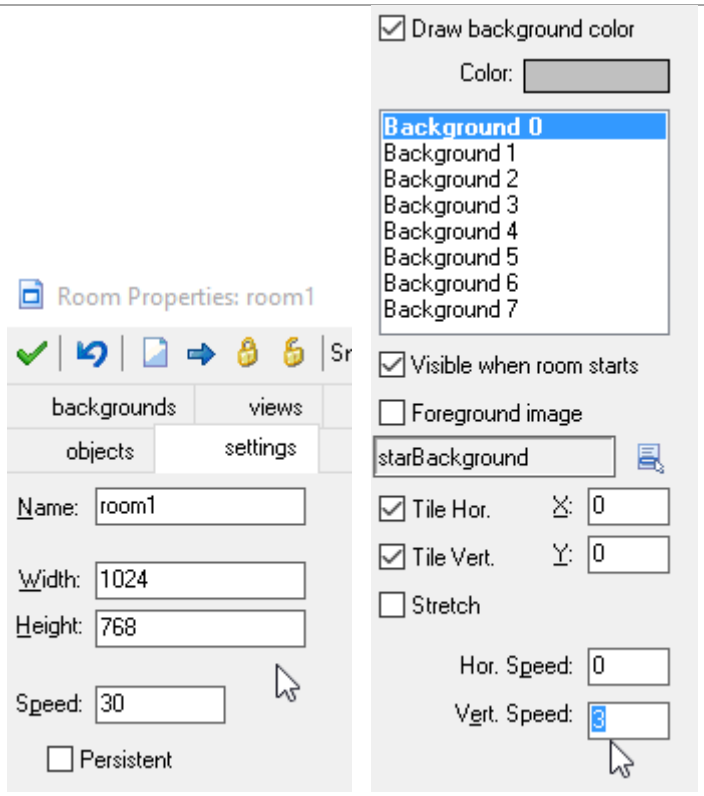
Load the image **starBackground.png** as a **Background**.

Make a new room called **level1** and set it to use that background. Set the **Vert. Speed** to **3**. This will make the background scroll down at a speed of 3 pixels per step. This will help make it look like the player are moving even if their sprite is just sitting in one place.

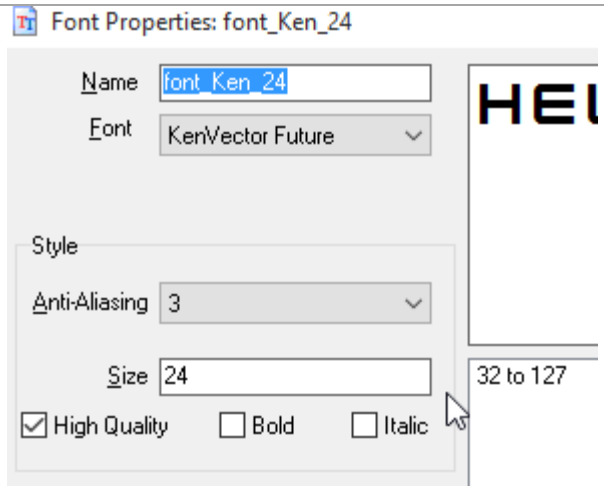
Duplicate it and call the room **level2**

Duplicate it again call the room **level3**

Duplicate it once more to make **roomEnd**



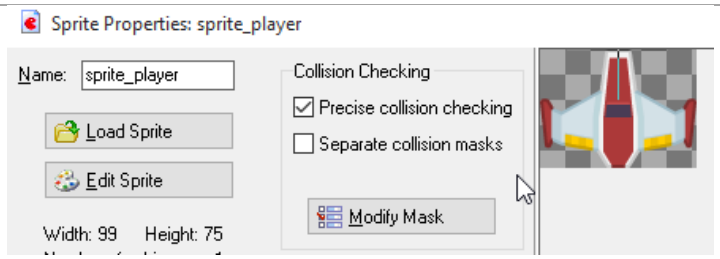

Set up a font of about size 24 for use later.



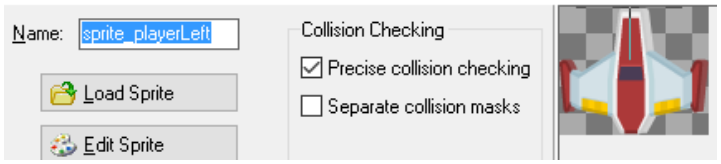
# 4 SPRITES

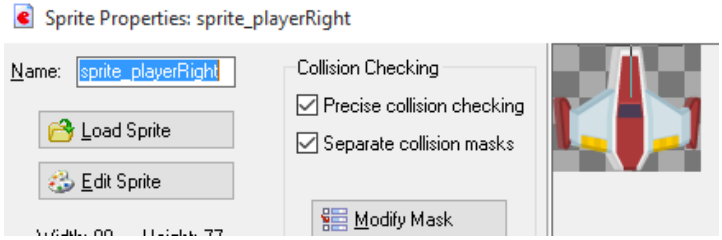
Load the image **player.png** as a **Sprite** called **sprite_player**

Set Precise collisions and move the **Origin** of the sprite to the middle of the nose (49, 0).



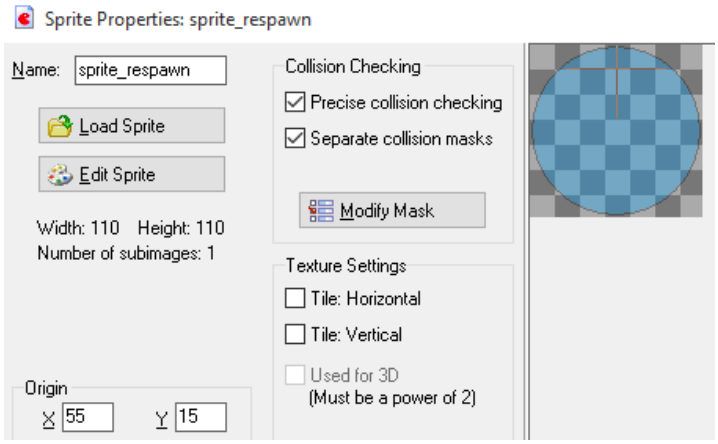

Do the same for **playerLeft** and **playerRight** images/sprites. For each one make sure to place the **Origin** in the middle of the nose (x: 42 for left and x: 47 for right).
We want the ship not to "jump" when we switch from one sprite to the other. Thus we need the sprites to be visually lined up.




Make a **sprite_respawn** from **respawn.png**

Set the **Origin** to **55, 15**
That should be about right so that when this sprite is lined up over the player ship sprite, the ship is centered in the circle.



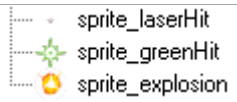

Bring in the sprites shown to the right from the corresponding images.

All should use **Precise Collision Checking** and center the **Origin** of each sprite.

The last three sprites are all animated.

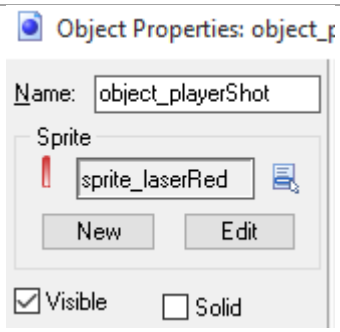
For each one, create a sprite, then Edit it and use File menu Add From File to select the additional frames. _0 is the first from for each, _1 is the second, _2 the third…

All three sprites should have a centered **Origin**. They do not need precise collisions, they will not be colliding with other things.

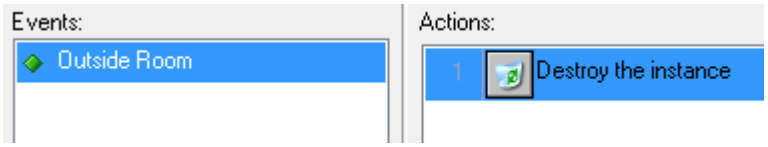



# 5 THE PLAYER & SHOOTING

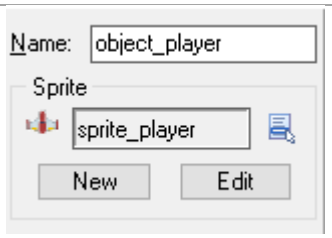
Make an **object_playerShot** using the red laser sprite




The only needed action is to destroy itself when outside the room.



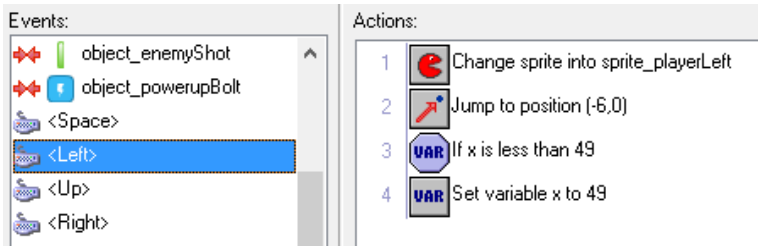

Make an **object_player** using the player sprite




Start with a **Left** event (keyboard, not press)
First change the sprite to show the playerLeft sprite

Then **Jump to Position -6, 0 Relative**
That should make the ship move 6 to the left

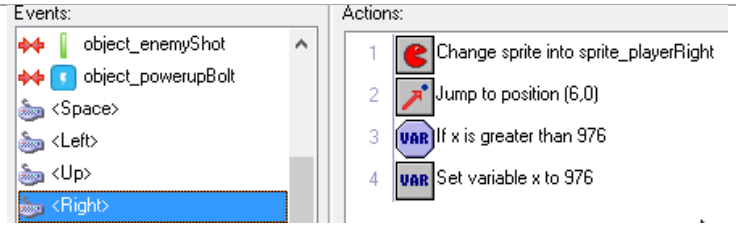
Finally, to prevent the player from moving off screen, **Test Variable x** to see if it is **less than 49**
If so, we will **Set Variable x** to be **49**
*(49 is the distance from the Origin of the player sprite to the edge).*

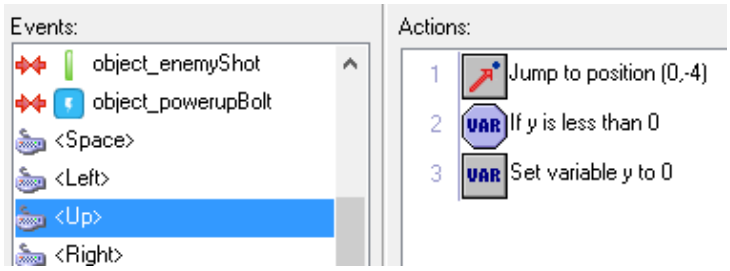


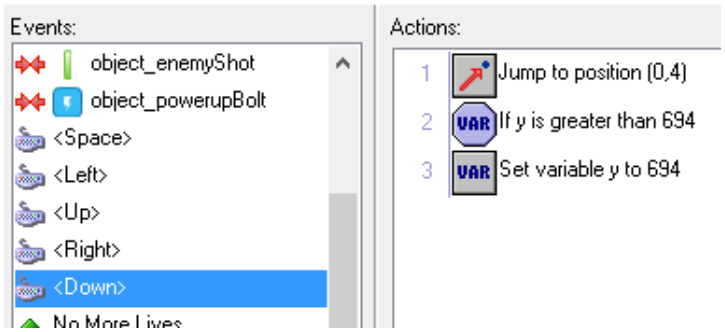
Now do the other directions according to the pictures to the right.

Note that the **Jump to Position** is always **Relative** – we are trying to shift by a few pixels, not set the position to an absolute location.

The **Test Variable** values have been chosen to keep the ship's sprite on the screen.



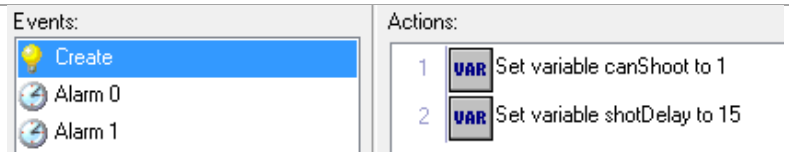

Add a **Key Release Left** and **Key Release Right** event
Both should just set the sprite back to the normal player sprite.


Go place an object_player in the bottom middle of level1 and try out the movement – make sure you can't fly outside the room in any direction.

Now to add the shooting.

Start with a Create event that makes two variables, **canShoot** set to **1** and **shotDelay** set to **15**

canShoot will be 1 to indicate the ship is ready to fire and 0 to indicate it needs to wait. The shotDelay will be used to space out shots – storing the number in a variable will make it easier to change later.



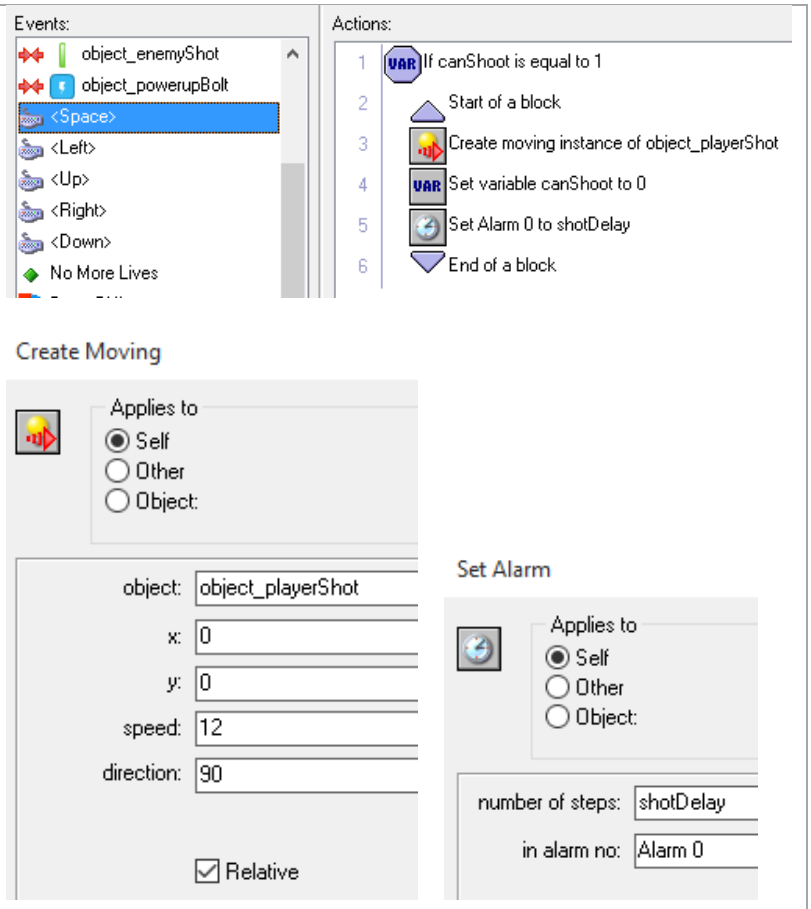
Add a **Space** keyboard event

First, test to see if canShoot is 1. We only do anything if that is true.

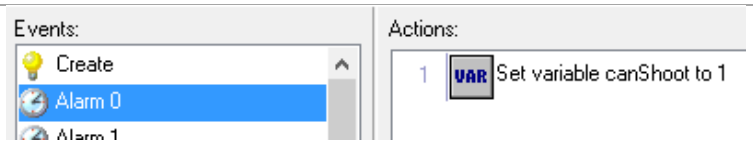
The details for **create moving** and the **alarm 0** are shown below.

Setting canShoot to 0 will prevent the player from firing again immediately. We set the alarm to reenable shooting after a little time has passed.

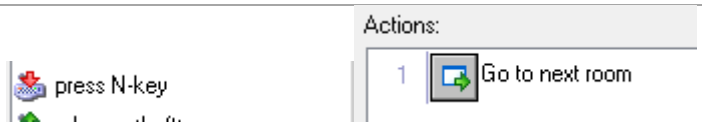
**shotDelay** was set in the Create event to 15. So putting “shotDelay” in as the number of steps is kind of like just putting 15. But, by changing shotDelay later we will be able to change how long the alarm is set for after each shot.




**Alarm 0** is responsible for just setting **canShoot** back to **1**




Finally, add a cheat to skip rooms – this will help for later testing

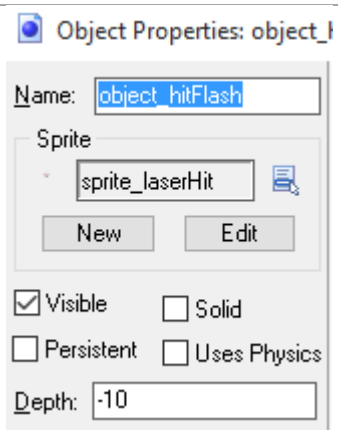



Test shooting with the player object. You should be able to hold the space bar and shots will fire at a steady rate, but not in a giant machine gun stream.

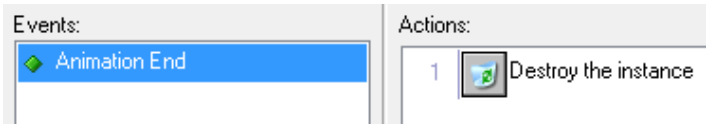
# 6 ENEMIES & METEORS

First make an **object_hitFlash** that will be used to show when a bullet hits an enemy.

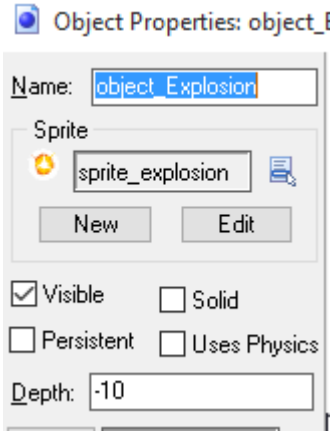
Note the depth of -10 to show up above other objects.

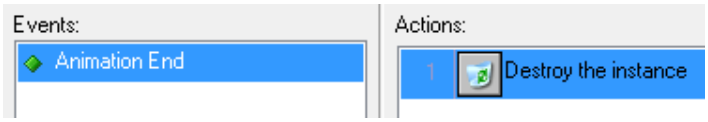



The only action is to destroy itself once the animation is done.

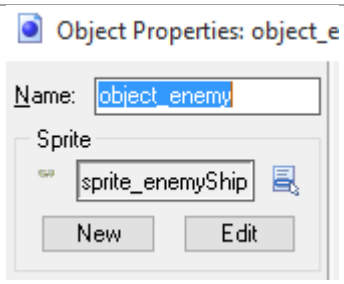



Do the same to make an **object_Explosion**

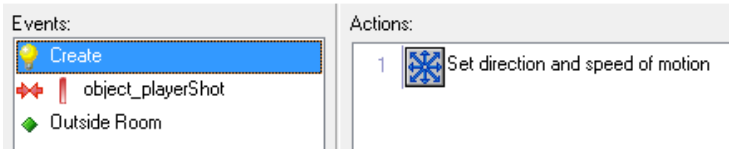


Make an **object_enemy** using the player sprite

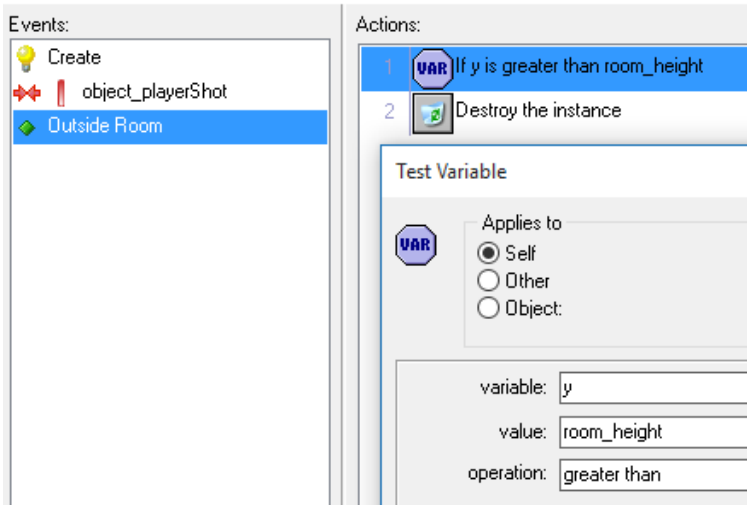



Create should set it moving direction **270** (down) at speed **7**




Outsides room should test if **y** is **greater than room_height**
Place a destroy action after the test so that the enemy is only destroyed if they go off the bottom of the screen. This will prevent them from being destroyed when they are entering from the top of the screen.

***ROOM_HEIGHT IS A BUILT IN VARIABLE THAT ALWAYS EQUALS THE VERTICAL HEIGHT OF THE ROOM. WHILE YOU COULD TYPE 768 FOR THE VALUE, IF YOU EVER RESIZED THE ROOM, YOU WOULD HAVE TO FIND THAT NUMBER AND CHANGE IT. BY USING ROOM_HEIGHT IT WILL CHANGE AUTOMATICALLY. THERE IS ALSO A ROOM_WIDTH YOU CAN USE.***




If you want, you could go adjust the player object movement code to use room_width and room_height instead of hard coded numbers. Instead of 976, right could use **room_width - 48** Down could use **room_height - 74** instead of 694. By calculating these values using an expression instead of in advance, the game will adjust itself if you change a room size.

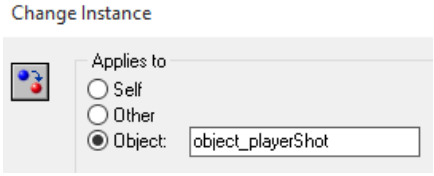
Finally, add Collision with object_playerShot.

It should **destroy** the enemy and **100** to the score (set to 100 relative)

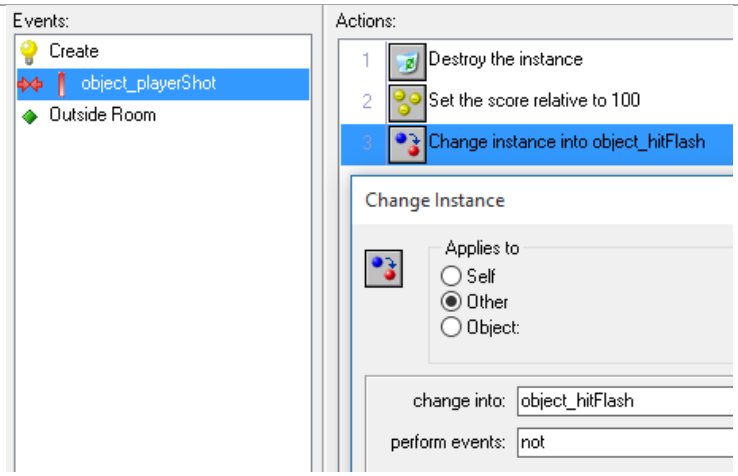
Finally, we need destroy the shot and create a flash. We can do both by turning the shot into a hitFlash. Add a **Change Instance**, but for the **Applies to** click **Other**. This means that the enemy won't do this, the other thing (playerShot) will do this action.

***APPLIES TO OTHER ONLY WORKS IN COLLISION EVENTS. SELF IS THE OBJECT THE EVENT IS IN. OTHER IS THE THING BEING COLLIDED WITH. IF YOU TRY TO USE OTHER IN A DIFFERENT EVENT (LIKE CREATE), NOTHING WILL HAPPEN… THERE IS NO "OTHER" OBJECT INVOLVED.***

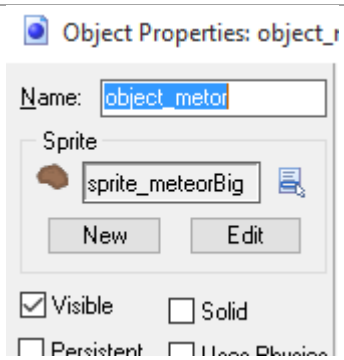
***THE THIRD POSSIBILITY, OBJECT: ALLOWS YOU TO PICK A TYPE. ALL OBJECTS OF THAT TYPE WILL THEN DO THE ACTION. FOR EXAMPLE, THIS:***




***WOULD MAKE EVERY SHOT TURN INTO A HITFLASH. WE DO NOT WANT THAT. (THOUGH YOU MIGHT WANT TO USE THAT FOR A SUPER WEAPON THAT CLEARED THE SCREEN OF ENEMIES).***



Now make an **object_meteor**

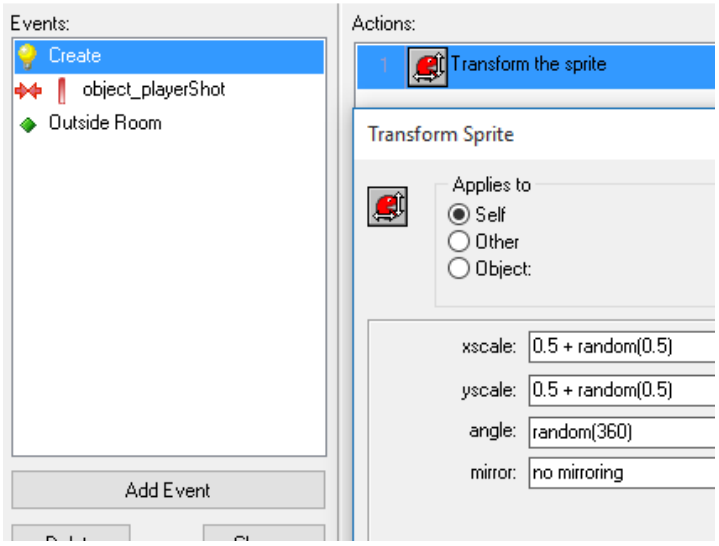



Use **Create** to randomize the appearance of the meteor.

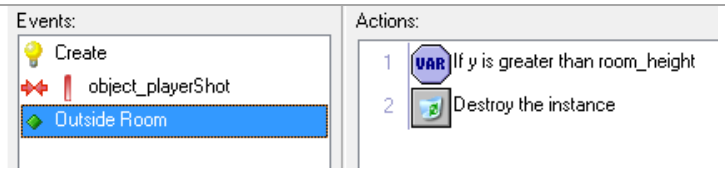
The recipe: **0.5 + random(0.5)**
Will add 0.5 to a random value from 0 to 0.49999 giving a total of 0.5 to 0.99999

Set the xscale and yscale to use that recipe – this will make each meteor have a unique size (from ½ the sprite's normal size to almost full size).

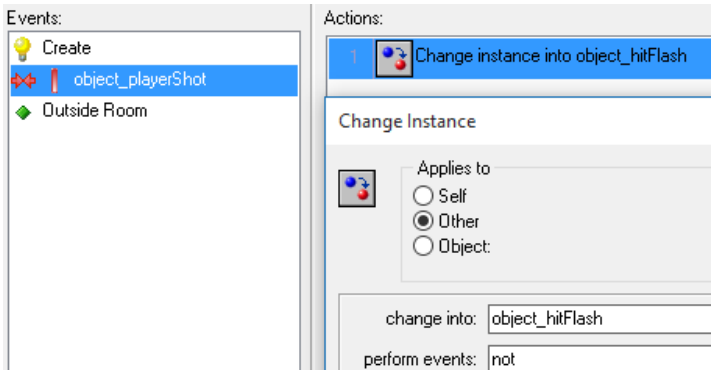
**random(360)** will make the angle of each be random as well




The outside room event should look just like the enemy one




While collision with playerShots should turn the shot (the **Other** thing) into a hitFlash and do nothing else. We want shots to be stopped by meteors without damaging the meteor.

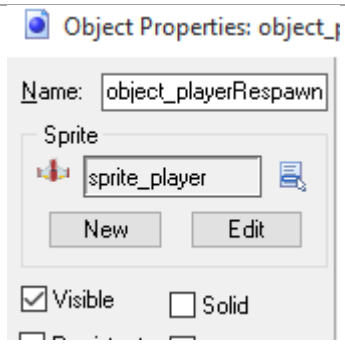



Put a few enemies in and a few meteors. Right now the meteors won't move and nothing hurts the player. Make sure you can shoot enemies and destroy them but can't destroy meteors.

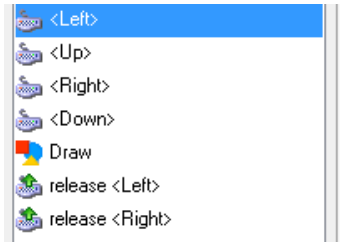
# 7 RESTART AND COLLISIONS

When the player hits something, we will take away a life and "respawn" them at their starting location. Since that location may not be safe, we will make them invulnerable for a few seconds so they can get their bearings and ready to start playing again.

Right click **object_player** and **Duplicate** it to make **object_playerRespawn**




We want the player to be able to fly around, but not shoot. So in the respawn object, **delete** all the events except for the ones involving the direction keys.
*We will add the draw in a second…*

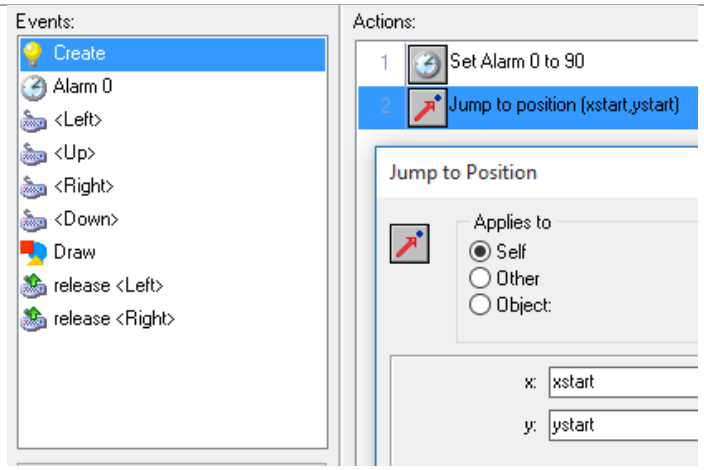



Add a new **Create** event

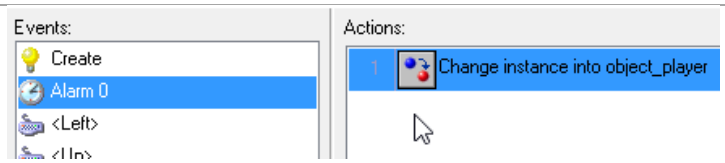
It should set an alarm for 3 seconds (90 steps)

It should **Jump to Posistion** x: **xstart** y: **ystart**

***YSTART AND YSTART ARE BUILT IN VARIABLES THAT REMEMBER WHERE AN OBJECT STARTED OUT. WE WILL TURN THE PLAYER INTO THIS RESPAWN OBJECT WHICH MEANS THAT THE RESPAWN OBJECT WILL KEEP THE STARTING LOCATION THAT THE PLAYER HAD.***



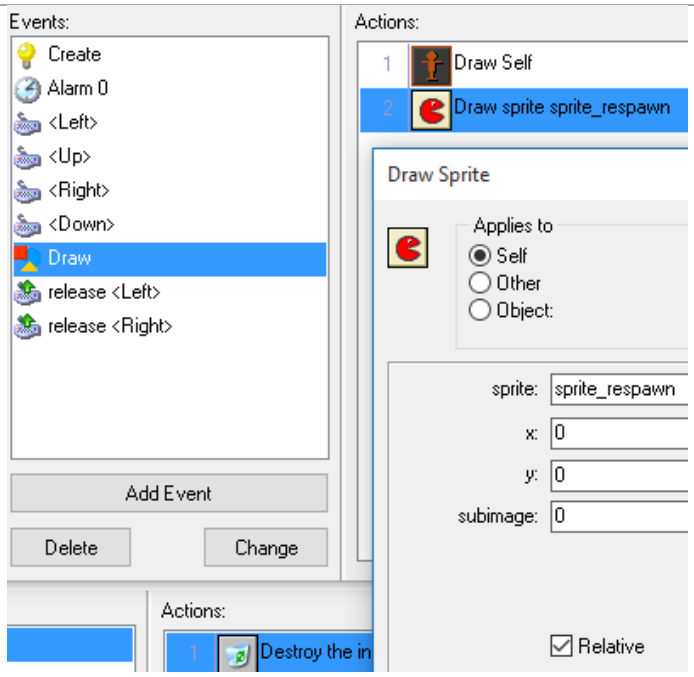

When the alarm goes off, change this object back into an **object_player**. **Perform events** should be **Yes** so that we do the **Create** event of the player and reset the shooting variables.



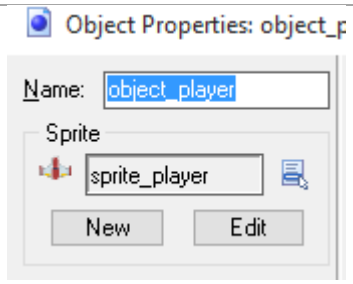
Finally, add a **Draw** event to change how this object is drawn

First, do the normal **Draw Self**

Then (want to draw this part on top, so it comes later) do a **Draw Sprite** and specify the **sprite_respawn** bubble.
**0, 0** relative will ensure we draw the bubble right on top of the ship (as long as you set the sprite Origins correctly).

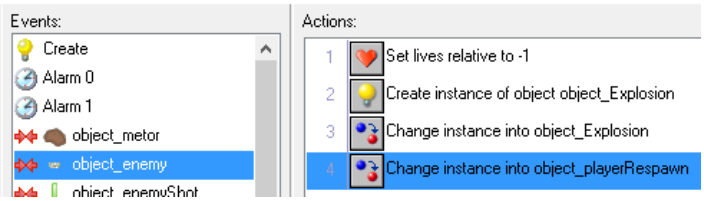



Go back to the regular object_player

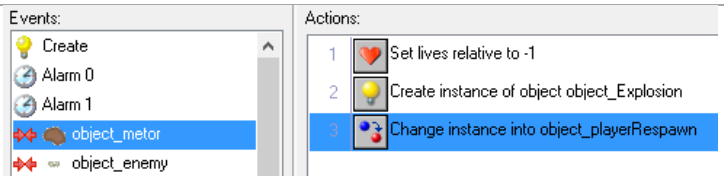



Add a **collision** with **object_enemy**. It should

- Subtract 1 life (-1 relative)
- Create an explosion at 0, 0 relative to show the ship blew up
- Change the Other (the enemy) into an explosion to blow it up
- **Change Self** (the player) into a playerRespawn
  Just like before, we need **Perform Events Yes** to say that we want to run object_playerRespawn's creation code (set the alarm).



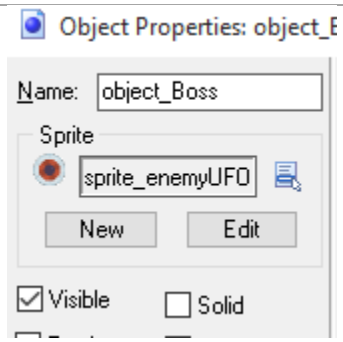

Add a **collision** with **object_meteor**. It should do mostly the same, but NOT turn the Other into an explosion. We do not want the meteor to be destroyed.




Test to make sure that you die when you ram things and respawn appropriately. The respawn player should be able to fly but not shoot and does not collide with anything. After three seconds it should go back to a normal player.

# 8 BOSS & CONTROLLER

Make an **object_boss** using the UFO sprite.

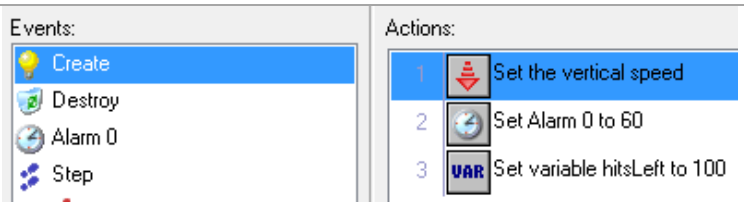



Create should:

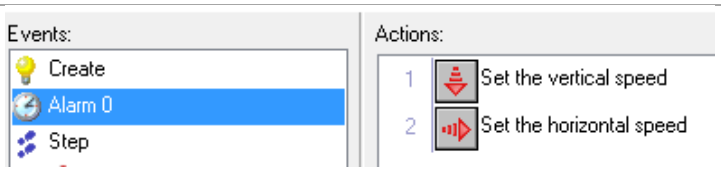
- Set the vertical speed to **2**
- Set **Alarm 0** to **60**
- Set variable **hitsLeft** to **100**

The alarm will be used to stop the boss's movement down into the room and start it moving side to side.

**hitsLeft** will be used to track the boss's health While we could use the built in health, that would prevent us from later having multiple bosses (they would have to share a health) or using health for the player.



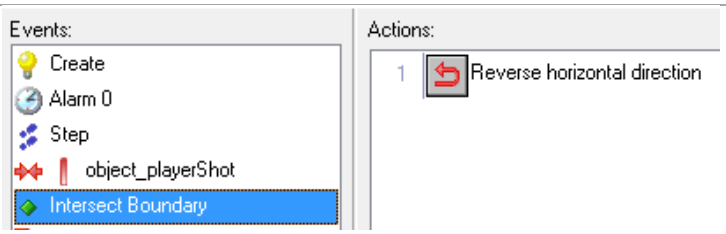

When the **alarm** goes off:
Set the **vertical speed** to **0**
Set the **horizontal speed** to **6**




**Other Event: Intersect Boundary:**

**Reverse horizontal direction**

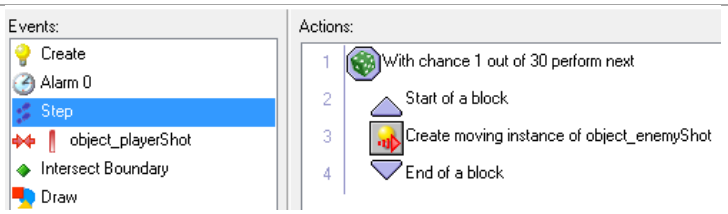
Once it starts moving side to side we want it to bounce between the two sides




Put an object_boss at the top of your room and make sure it moves down for 2 seconds then starts going side to side.

**Step Event:**
With a chance of 1 out of **30** do:
**Create moving** instance of object **object_enemyShot** at **relative** position (**0,0**) with speed **10** in direction **270**

This should make the boss shoot about once per second on average, through the shots will come randomly.



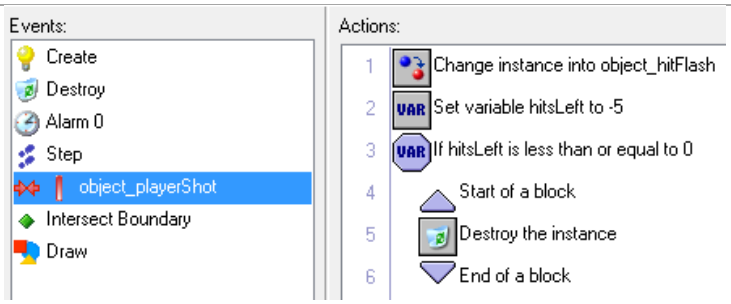
**Collision Event** with object **object_playerShot:**

For **other** object (the shot): **change the instance** into object **object_hitFlash**, not performing events.
This shows that the shot hit and destroys the shot so it does not keep flying through the boss and doing damage at each step.

***WHEN YOU HAVE A COLLISION THAT REDUCES HEALTH – LIKE A BULLET HITTING SOMETHING, YOU USUALLY WANT TO DESTROY ONE OF THE COLLIDING OBJECTS. OTHERWISE, THEY WILL CONTINUE TO COLLIDE AND DO DAMAGE EVERY STEP AS THEY PASS THROUGH.***

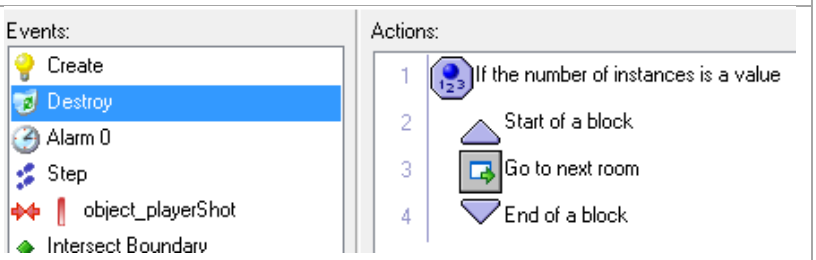
Set variable **hitsLeft relative** to **-5**
To subtract 5 "health" from the boss.

if **hitsLeft** is **less than or equal** to **0**
    **destroy the instance**
Once the boss reaches 0 health, destroy it.




**Destroy Event:**

if **number of** objects **object_Boss** is **equal to 1**
    **Go to next room**
When a boss is being destroyed, check to see if it is the last one, if so, level is done. This way if you want a harder level you can spawn two bosses close together and the player must defeat both to move on.

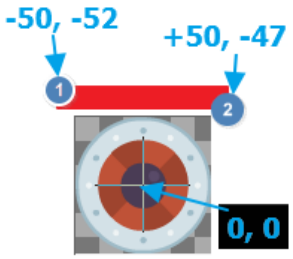


We need a custom **Draw Event** to draw not only the normal sprite, but a health bar above it.

**Draw Self** to draw the normal sprite
Then **set the drawing color** to **red**

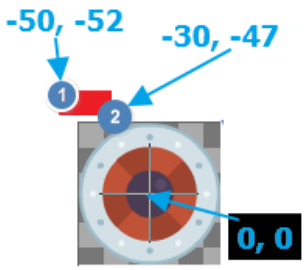
Then, to draw the health bar:
**Draw rectangle** with **relative** vertices (**-50, -52**) and **(-50 + hitsLeft, -47**), **filled**

x1 and y1 are interpreted as the coordinates of the upper left corner of the rectangle to draw as seen in the picture below, x: -50 and y: -52 mean a location about 5 pixels to the right of the sprite and 7 pixels above it (the sprite is 90 pixels wide and tall, so the edge is 45 pixels from the origin). Point 1 in the image below:

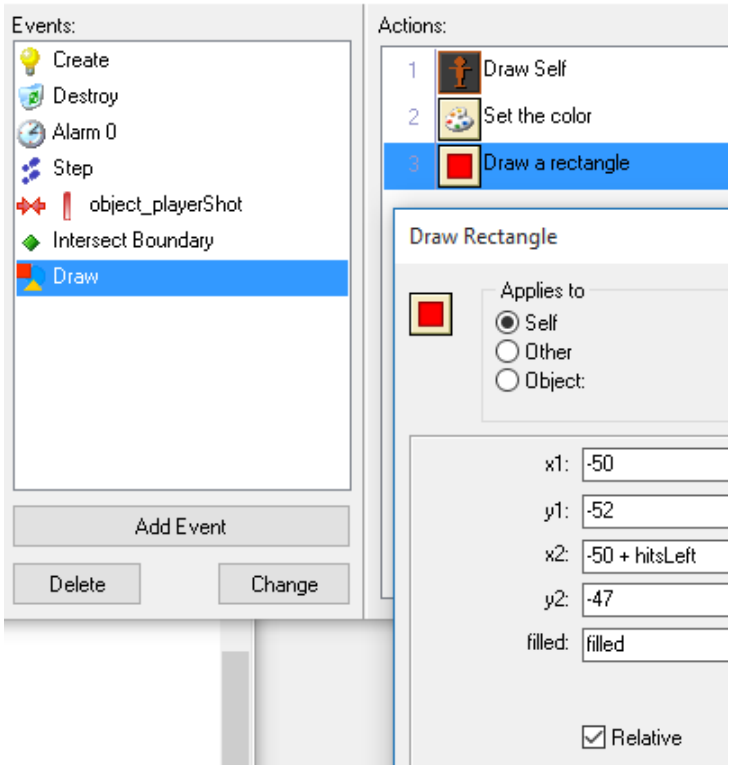


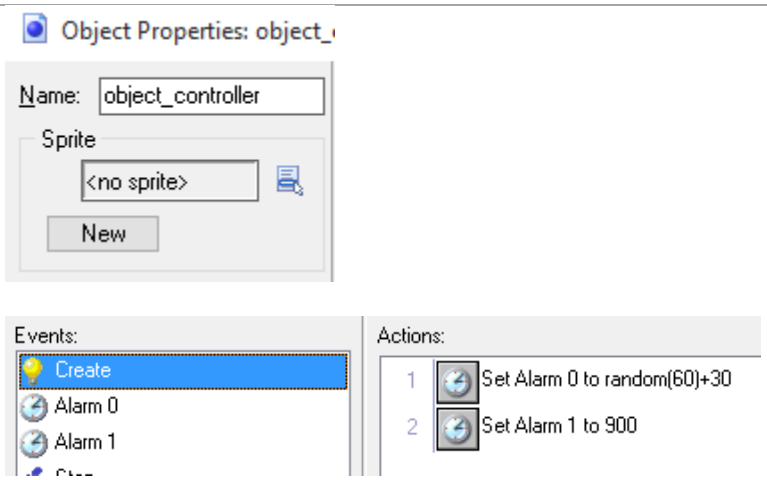
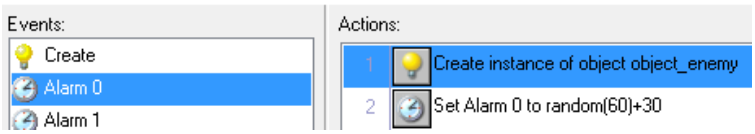
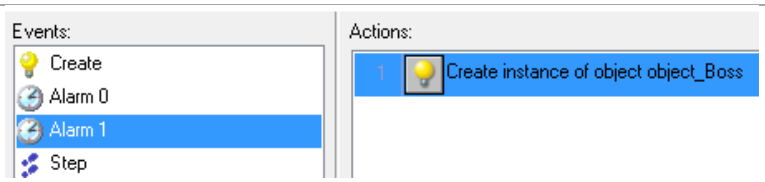

x2, and y2 are interpreted as the lower right corner of the rectangle. x: 50 and y: -47 mean a location about 5 to the right of the edge of the image and 2 pixels above the top (point 2 in the image).

Note that for x2, we use **-50 + hitsLeft** as the expression. Initially, hitsLeft is 100 so that expression = -50 + 100 or +50.
As the boss takes damage, the value will get less and less. When the boss is down to 20 hitsLeft, the expression would = -50 + 20 = -30. if x2 is -30, it means the right edge of the rectangle would be at -30 relative to the sprite and would look like the next picture:



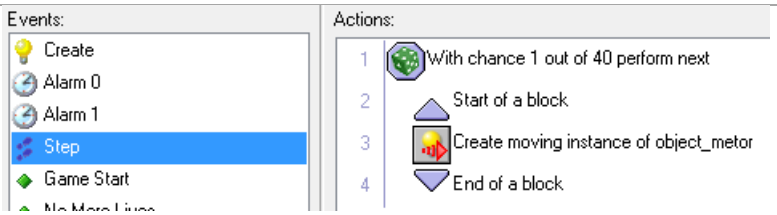

This way the health bar shrinks as the boss takes damage.



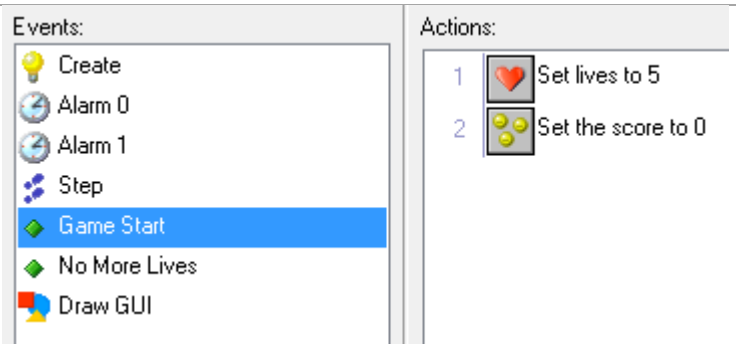
| | |
|---|---|
| Test that your object_boss shoots and that you can shoot it. Once you are sure that works, you can delete the boss. | |
| Now make an **object_controller**<br><br>The create event should set two alarms, one to spawn an enemy, the other two spawn a boss.<br><br>**random(60) + 30** will make a timer that lasts between 1 and almost 3 seconds (30-89 steps). The other timer is for 30 seconds (900 steps). |  |
| **Alarm 0** should<br>**Create instance** of object **object_enemy** at position x: **random(924) + 50** y: **0**<br>That should be somewhere along the top edge, but not right at the sides (x can be 50-974 in a room that is 1024 wise). If you want to make it flex with the room size, you could set it to: **random(room_width - 100) + 50**<br><br>It then resets the alarm to another 1-3 seconds: **random(60)+30** |  |
| **Alarm 1:**<br><br>**Create instance** of object **object_Boss** at position (512, -45)<br>This is just off the middle of the top of the screen. The boss should move down into view from off the screen once it is created.<br><br>The “flexible” version that would scale with rom size would be to set x: to **room_width / 2** |  |

**Step Event:**

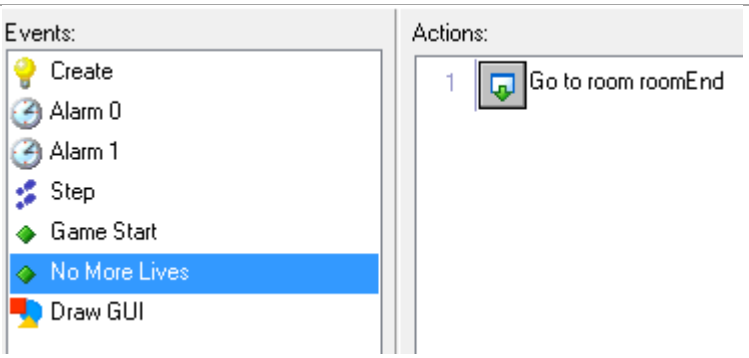
With a chance of 1 out of **40** do:
**Create instance** of object object_metor at position x: **random(room_width)** y: **-50** with speed **6** in direction **270**
The enemies are forced to space out by at least a second each. The 1/40 chance for meteors means they also should come a little less than once per second. But if the dice are "lucky" two may come a fraction of a second apart or if the dice are "unlucky" there may be a big gap between them.




**Game Start** should do the setup shown to the right

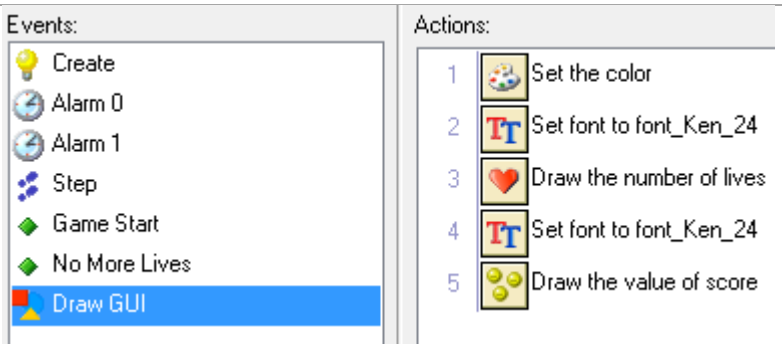



When there are **No More Lives** go to the ending room




**Draw GUI** event will draw the score and lives:

- Set the drawing color to **white**
- Set the font for drawing text to use your font and align **left**
- At position x: **5** y: **5** (upper left of the screen) **draw the number of lives** with caption **Lives:**
- Set the font for drawing text to your font, but aligned **right**
- At position x: **room_width – 5** y: **5** draw the value of score with caption **Score:**

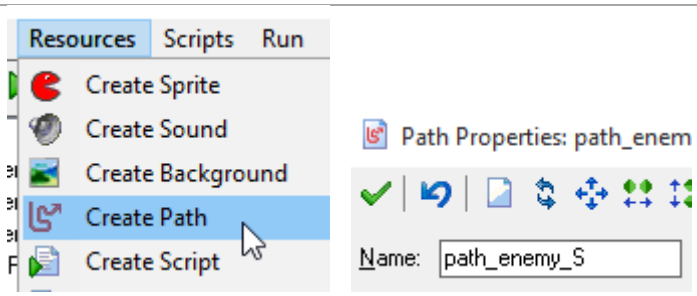



Put a controller in the first room and try it out. Make sure you get enemies and meteors, and after 30 seconds a boss appears.

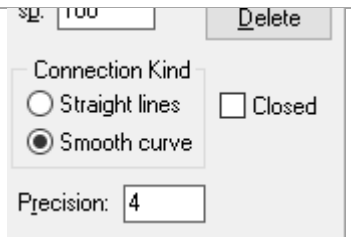
# 9 PATHS AND LEVEL 2

***A PATH ALLOWS YOU TO DEFINE A COMPLEX PATH FOR AN OBJECT TO FOLLOW INSTEAD OF JUST MOVING IN A SET DIRECTION.***

Create a Path called **path_enemy_S**




Frist make the path NOT closed (not a loop) and **Smooth**

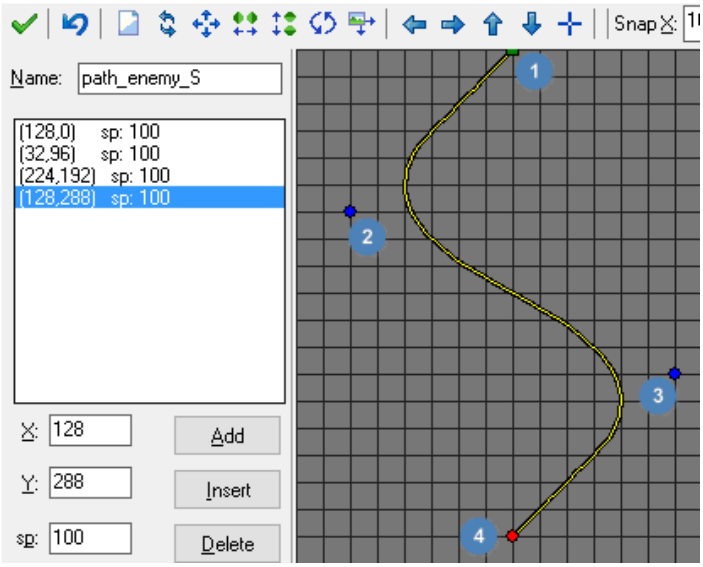



Try clicking in the window at the four shown locations. Each time you click another location will be added to the path. You can right click to delete points.

You can adjust a point by dragging it or by selecting it and typing an X and Y into the boxes.

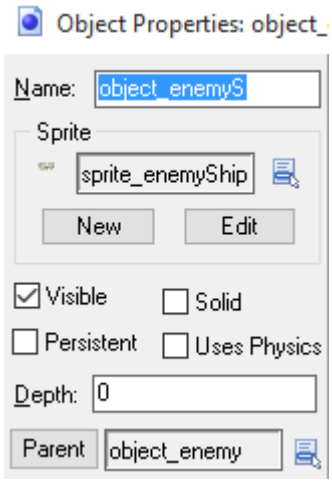
The exact locations are not critical – but try to make the end point (red one) directly below the start (green one) and a roughly S shape.

These arrows can be used to shift the view around:



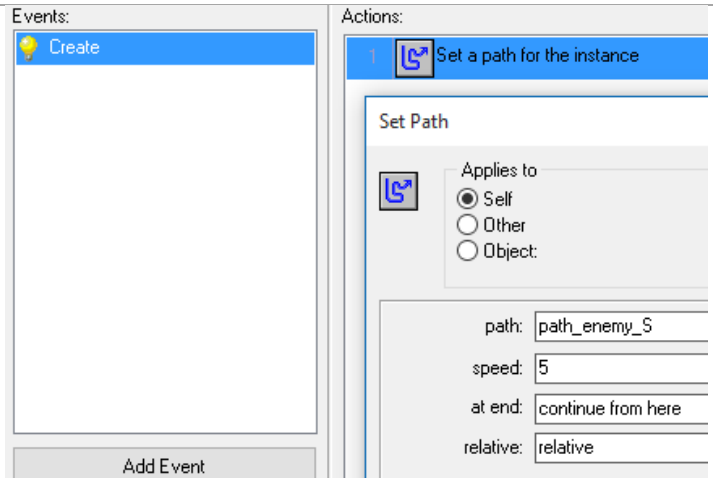

Now make an enemy to use that path

Make an **object_enemyS** with **object_enemy** as its parent

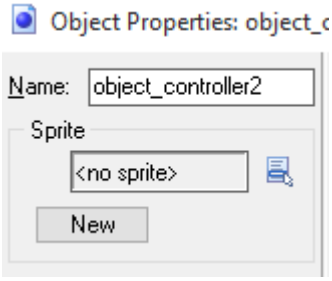


Make a **Create** event to override the one from object_enemy

In it, **Set Path** to use the path you created at a speed of 5. When the object is done with the path we want it to do it again from its current location (**continue from here**). **relative** means that the path will start from where the object is, not a set place in the room.



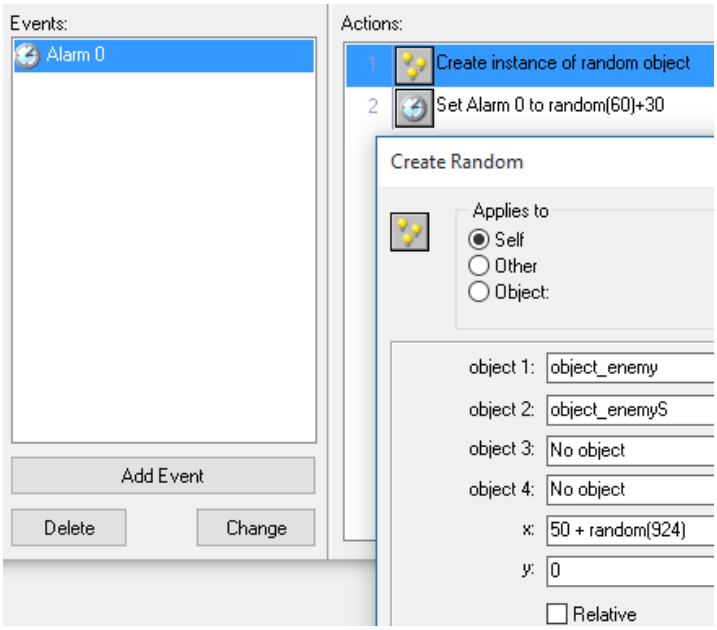

Make an **object_controller2** with **object_controller** as its parent to control room 2




The only thing we will change is what enemies get spawned. Make an **Alarm 0** event to override the one from the parent. (The parent create event will still set Alarm 0 for us, we are just changing what happens when it goes off.)

**Create Random** will randomly pick from a list of possible objects. We want to randomly pick either an enemy or enemyS and create it at the top of the screen at a random x position **50 + random(924)**

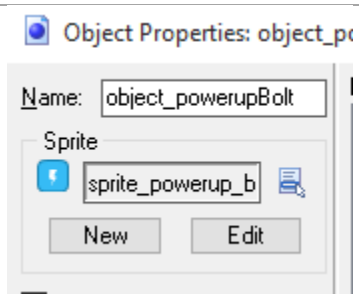
***"NO OBJECT" WILL NEVER BE SELECTED. YOU CAN LIST SOMETHING MORE THAN ONCE: IF WE HAD ENEMY, ENEMYS AND THEN ENEMY AGAIN, THERE WOULD BE A 2/3 CHANCE OF SPAWNING AN ENEMY AND A 1/3 CHANCE OF AN ENEMYS.***



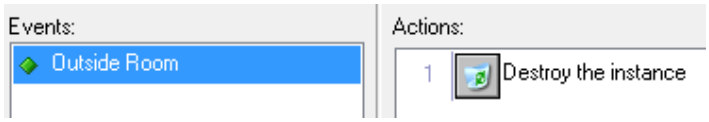

Place a controller2 in level2 (as well as a player object!). Use your cheat code (N) to skip to that level and try it out. You should get a mix of the enemy types, with enemyS types doing a series of S slaloms down the screen. Otherwise it should work like level1.

# 10 POWERUP AND LEVEL 3

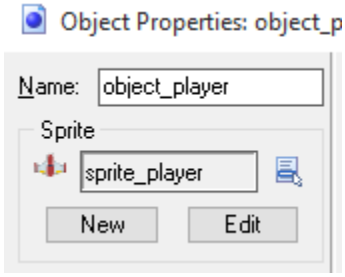
Create an **object_powerupBolt**




The only action it needs is to **destroy** itself when it leaves the room.

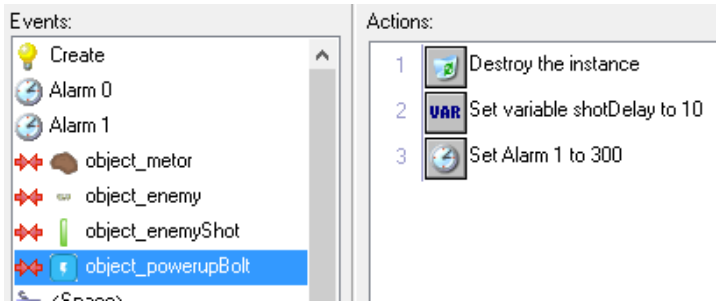



Go back to **object_player**




Add a **Collision** Event with **object_powerupBolt**:

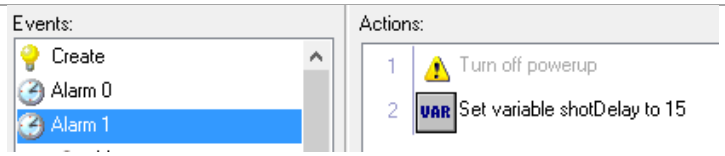
- For **other** object (the powerup): **destroy** the instance
- Set variable **shotDelay** to **10**
- Set **Alarm 1** to **300**




Setting the **shotDelay** to 10 will mean that every time the player shoots they only have to wait 10 steps to shoot again. We set the alarm to disable the powerup after 10 seconds.

**Alarm 1** just needs to set **shotDelay** back to the starting value of 15

***THE EXCLAMATION ACTION IS A COMMENT OR A NOTE FOR PEOPLE THAT THE COMPUTER IGNORES. IT IS A NICE WAY TO LEAVE YOURSELF REMINDERS OF WHAT A PIECE OF CODE IS DOING.***

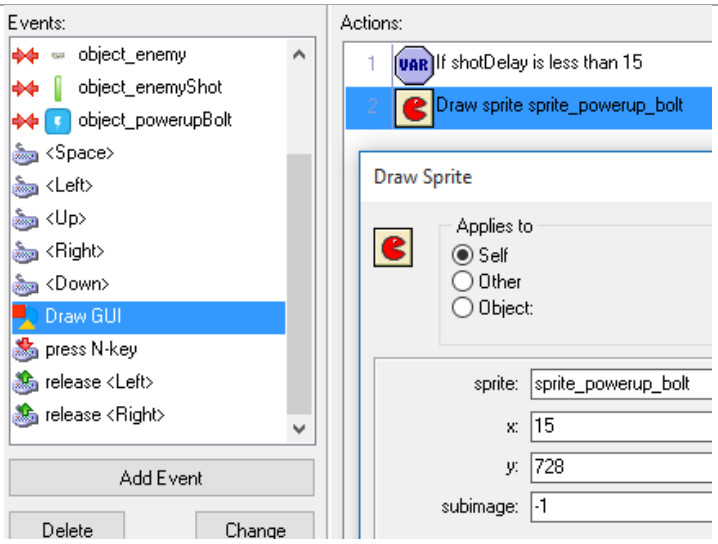


Finally, add a **Draw GUI** event that we will use to display an icon when the player has a powerup active

We know they have the powerup if their shotDelay is lower than the original 15. So:

If **Test Variable shotDelay is less than 15** do: **Draw Sprite sprite_powerup_bolt** at x: 15 y: 728 (lower left corner).

If the **shotDelay** is the normal 15, we will not draw anything in the **Draw GUI**



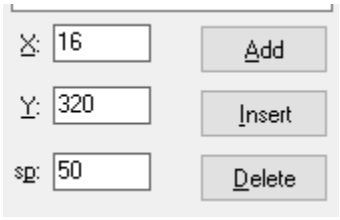

Test out the powerup – put one in room 1 right above the player and go grab it. Make sure it speeds up your shot and that it goes away after 10 seconds.

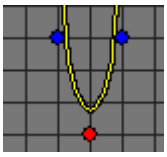
Now we will make an enemy to drop the powerup. First create a Path called **path_enemy_U**

Draw something like the picture to the right. Once again, the exact numbers are not critical, but the ending point (red one in picture) should be above the starting point.

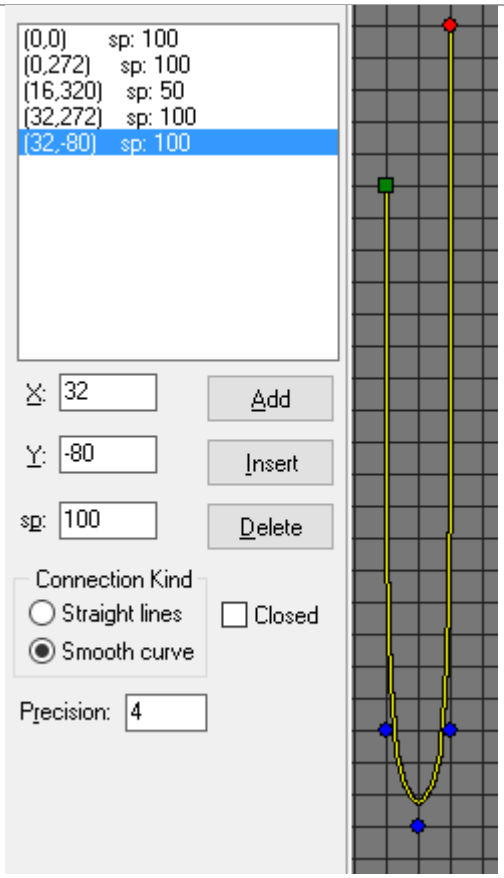
***NOTE THAT FOR EACH POINT YOU CAN SPECIFY A SPEED SP. CHANGING THE SPEED OF A POINT MAKES THE OBJECT GO SLOWER OR FASTER WHEN NEAR THAT POINT.***

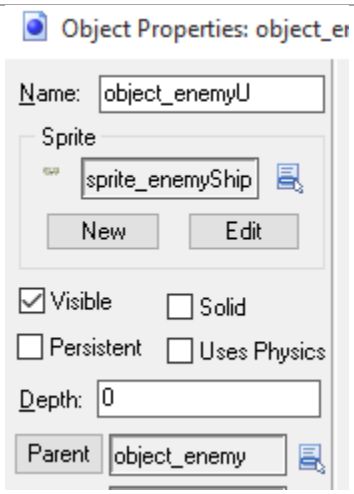
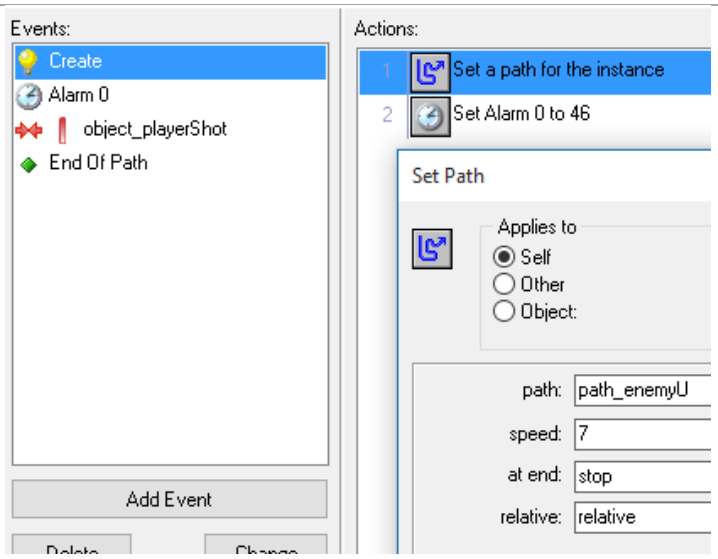
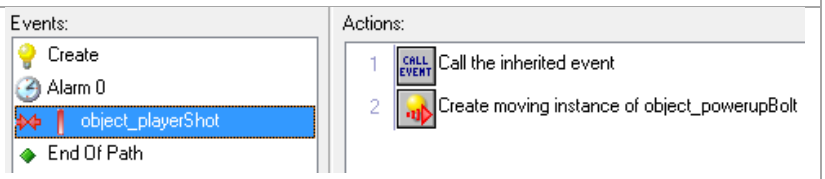
On my path, the very bottom point has a speed of 50:

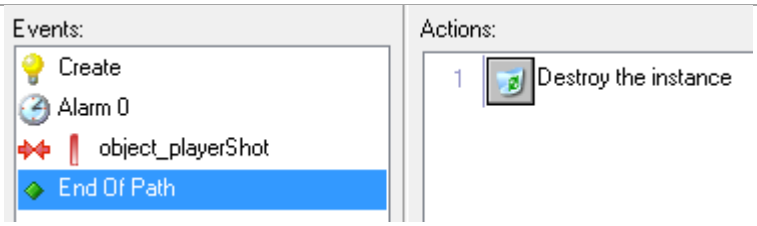
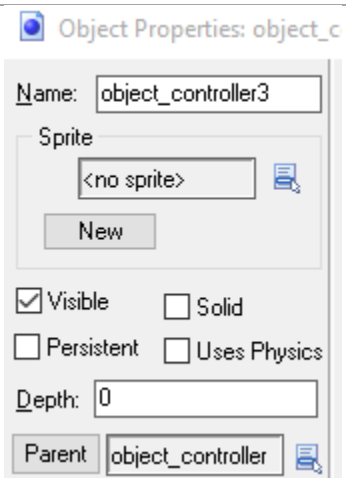
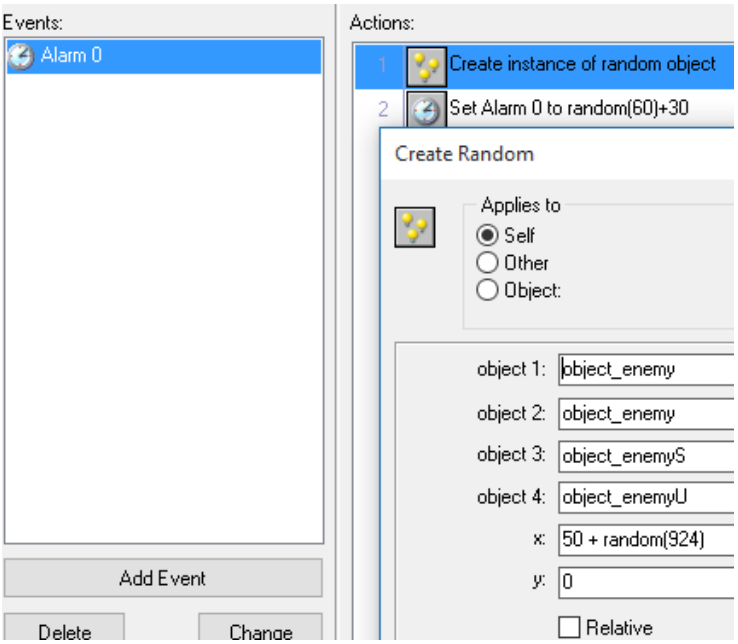


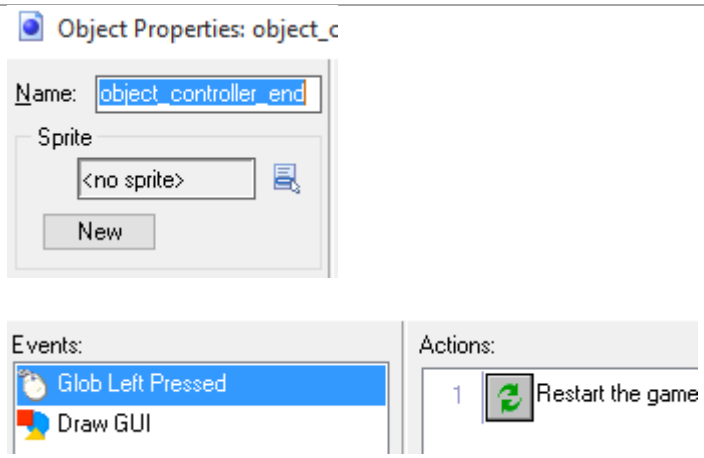
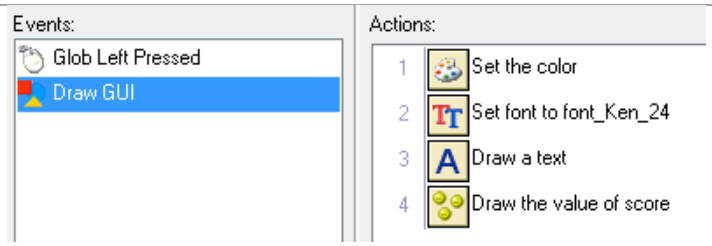
This will make it slow down slightly as it passes the bottom of the U shape.



| | |
|---|---|
| Make an **object_enemyU** with **object_enemy** as its parent again. |  |
| **Create Event:**<br><br>Set the path to **path_enemyU** with speed **7** and at the end **stop** – it should be **relative** so it starts the U shape from wherever the object is.<br><br>Set **Alarm 0** to **46**<br><br>46 is the amount of steps that it takes the enemy to travel to the bottom of my U at speed 7 (it is a distance of about 320 pixels divided by speed of 7). You may need to adjust that number so the alarm goes off at the bottom of your U. |  |
| **Alarm Event for alarm 0:**<br><br>Create instance of object **object_enemyShot** at **relative** position (**0**, **0**) with speed **10** in direction **270**<br>This should make a bullet drop from the enemy at the bottom of its U. |  |
| **Collision Event** with object **object_playerShot:**<br><br>**Call** the inherited event of the parent to do the normal “hit by shot logic”<br>Then **Create instance** of object **object_powerupBolt** at relative position (**0, 0**) with speed **3** in direction **270**<br>This will make a powerup “drop” when you shoot the U enemy. |  |

| | |
|---|---|
| In **Other → end of the path**, the enemyU should destroy itself.<br>At this point it should be back off the top of the screen. |  |
| You may want to test the enemyU by placing one at the top of the screen right above the player in room1 so it is easy to shoot. Shoot it and make sure it drops a powerup. Then try not shooting it and make sure it drops a shot and flies back off the screen. | |
| Now Duplicate **controller2** to make **object_controller3** (It too will use controller as its parent).<br><br>Change the **Create Random** so it looks like the picture. This should spawn a regular enemy half the time (2/4) and either an enemyS or enemyU the other half of the time. |  <br>  |
| Place an object_controller3 in room three and use your cheat key to go test it. | |

# 11 ENDING

<table>
<tr>
<td>Create an object_controller_end<br><br>When Global Left Mouse is pressed, it restarts the game</td>
<td> 
</td>
</tr>
<tr>
<td>Draw GUI Event:<br><br>• Set the drawing color to white<br>• Set the font for drawing text to font_Ken_24 and align center<br>• Draw text at position (512, 450) text: Click To Play Again<br>• At position (512, 350) draw the value of score with caption Score:</td>
<td> 
</td>
</tr>
<tr>
<td>Place an object_controller_end object in roomEnd. Test it out by finishing the game and also by running out of lives.</td>
<td></td>
</tr>
</table>

# 12 EXTRAS

Time to improve on the game.

Before you start making changes, make a backup copy of your project. Copy the WHOLE SpaceShooter.gmx folder, or whatever you called it, to another location. You should spend a couple of hours experimenting and trying to add some features to the game.

Focus on adding new features (objects and behaviors) not on the looks of the game or just adding a bunch of new rooms using the same objects. While you could probably make the game better by adding a detailed timeline instead of the random enemies, I don't want you to spend hours tweaking a timeline – focus on adding new features!

The ExtraArt folder has additional sprites you can use. If you are familiar with basic graphics programs you can also use the image editor in gamemaker to customize sprites. Do not spend lots of time trying to get graphics just right – I want you to focus more on making things happen than looking pretty.

## IDEAS:

You do not have to do all of these – you do not even have to do any of them if you have your own ideas. As you add features, try to think about how they affect the balance of the game.

- Bonus "beacons" or coins that can be caught for money
- Other powerups – use different variables to record which powerups the player has. A usual recipe is:
  Create event of player : Set Variable hasSuperGun to 0

Player collides with powerup : Set Variable hasSuperGun to 1 and set a new alarm
Alarm goes off : set hasSuperGun back to 0
In other events, Test Variable hasSuperGun to decide what bullets to shoot, or if the player actually gets hurt while colliding with an enemy, or how fast they move, etc…

- Other enemies
- Allow the player to aim with the mouse (hint – shots should move towards **mouse_x**, **mouse_y**)