VACUUM MARAUDERS

Part 1

Lane Community College

Table of Contents

Last Updated: January 2, 2017

SECTION 0 ►► OVERVIEW

In this game you will become more familiar with the Game Maker Studio Interface and will implement a game similar to Galaga.

This assignment will introduce the following concepts:

Game Design Elements

- Mouse Movement
- Enemy movement
- Attacking
- The Controller
- Fixing Errors
- Checking for Victory
- Garbage Collection

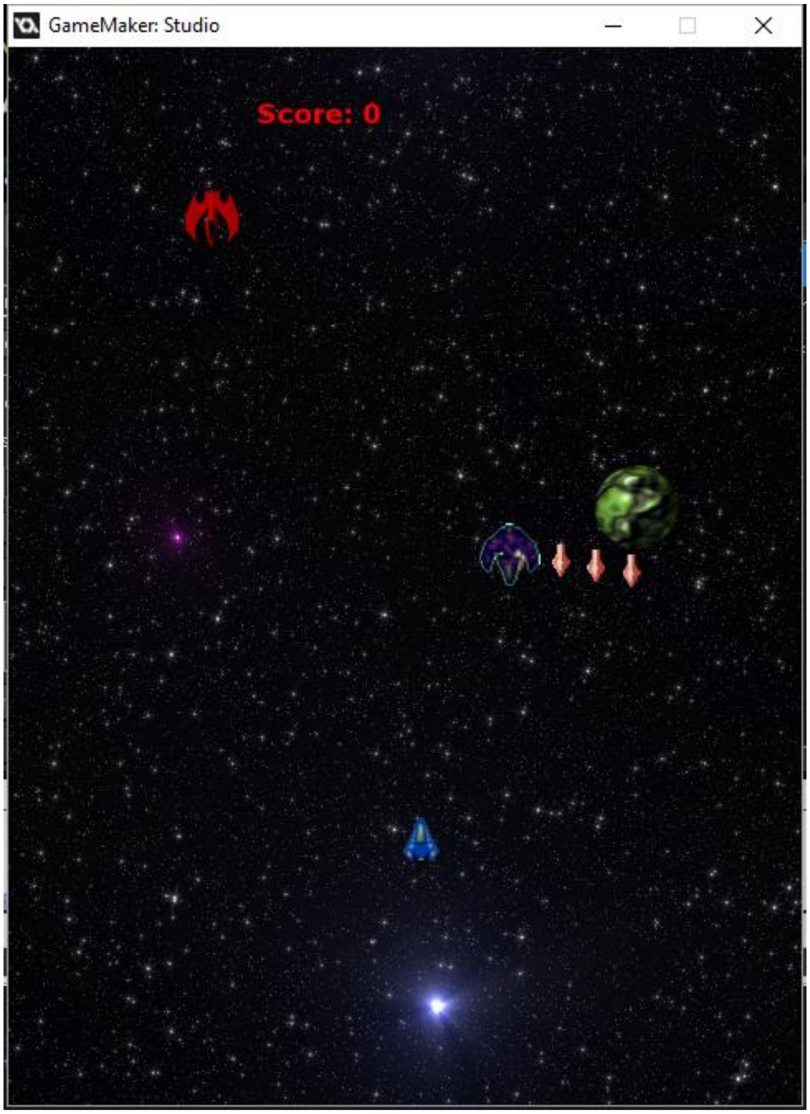
GameMaker Elements

- Loading Resources
- Variables
- Decisions
- Backgrounds
- Instances versus Objects
- What is Applies to?
- What is Relative?

Last Updated: January 2, 2017

SECTION 1 ►► PLAYING THE GAME

The first thing you need to do to get started here is to download and play the Vacuum Marauders demo file that is located on Moodle. This is the fully completed game. When you run the demo file your game will look like the following:



You play the game by moving the mouse around the screen and clicking the left mouse button to fire missiles at the enemy ships. Press the ESC key to re-start the game if you get killed.

The objective is to dodge enemy ships and missiles while fire missiles back at them to destroy all the enemy ships in order to save the Earth

Play the game a bit in order to understand the flow of it before moving onto the next step.

SECTION 2 ►► LOADING RESOURCES

Now that you're a little more familiar with Game Maker, for this assignment you'll be responsible for loading some of your own resources for the game in order to familiarize yourself with the process.

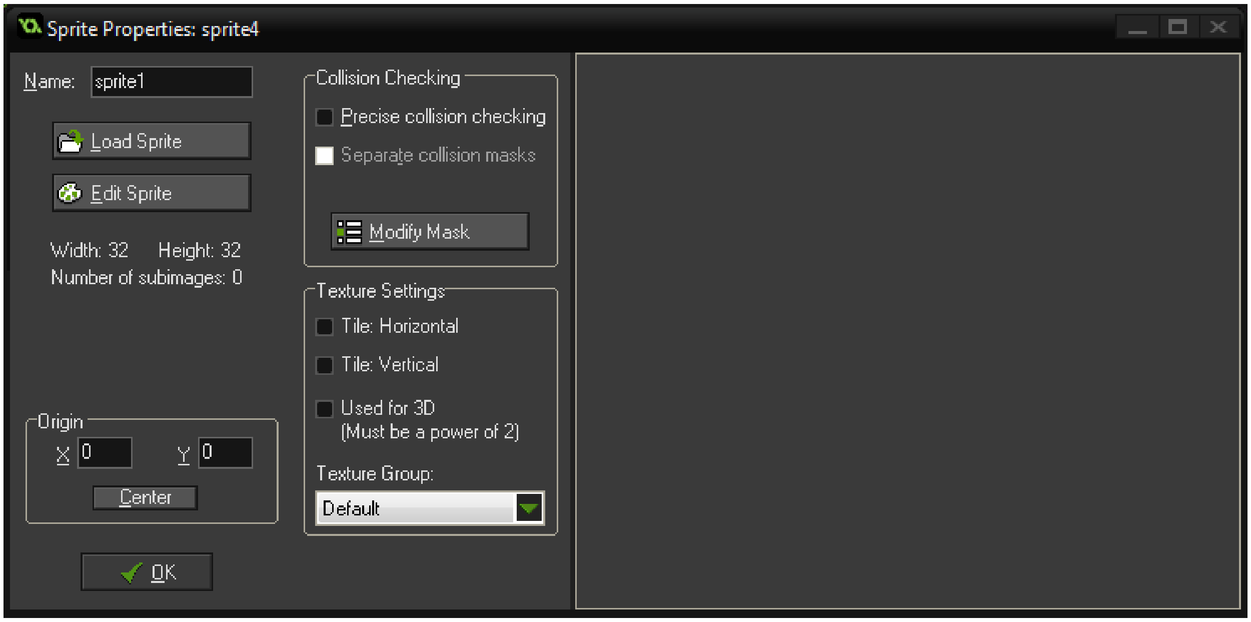
For this game you are going to need to load files for each of the following game resources:

Game Resources Table

Sprites	Sounds	Backgrounds
"Player_Ship_spr"	"Player_missile_snd"	"Room_Background_bkg"
"Enemy_Ship_1_spr"		

Along with a starter file, a .zip folder of image and audio files has been provided to you, please download these files from Moodle before continuing

We will start by creating the needed sprites. Click on Create a Sprite from the toolbar and the following window will open:


Sprite Properties: sprite4

Name: sprite1

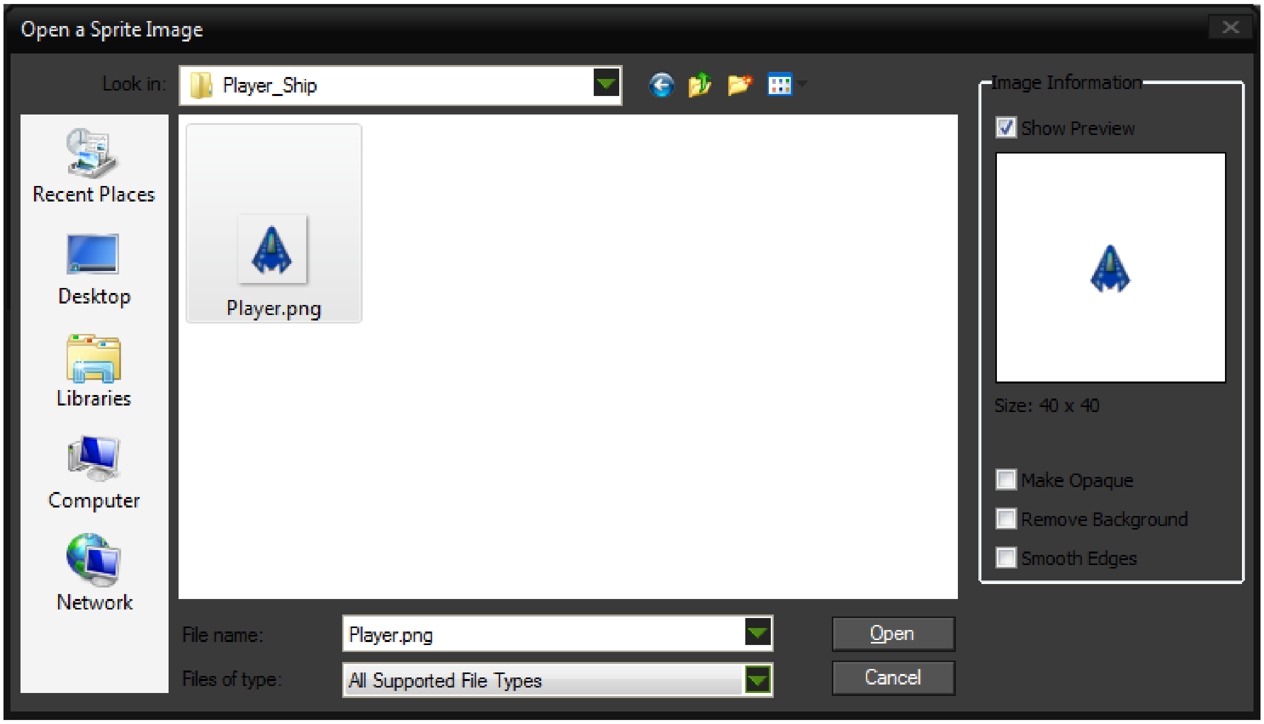
Width: 32 Height: 32
Number of subimages: 0

Origin
X 0 Y 0

Collision Checking
□ Precise collision checking
□ Separate collision masks

Texture Settings
□ Tile: Horizontal
□ Tile: Vertical
□ Used for 3D
(Must be a power of 2)
Texture Group: Default

In the Name: box type Player_Ship_spr (as shown above in the game resources list
R
Next select the Load Sprite button and navigate to the Resources folder. In this folder is a series of subfolders containing the different image and sounds needed for the game. Choose the Player_Ships folder. Within this folder are the Player ship images we will use in the game. Choose the Player.png file and click the **Open** button.



Open a Sprite Image

Look in: Player_Ship

Image Information

☑ Show Preview

Size: 40 x 40

□ Make Opaque

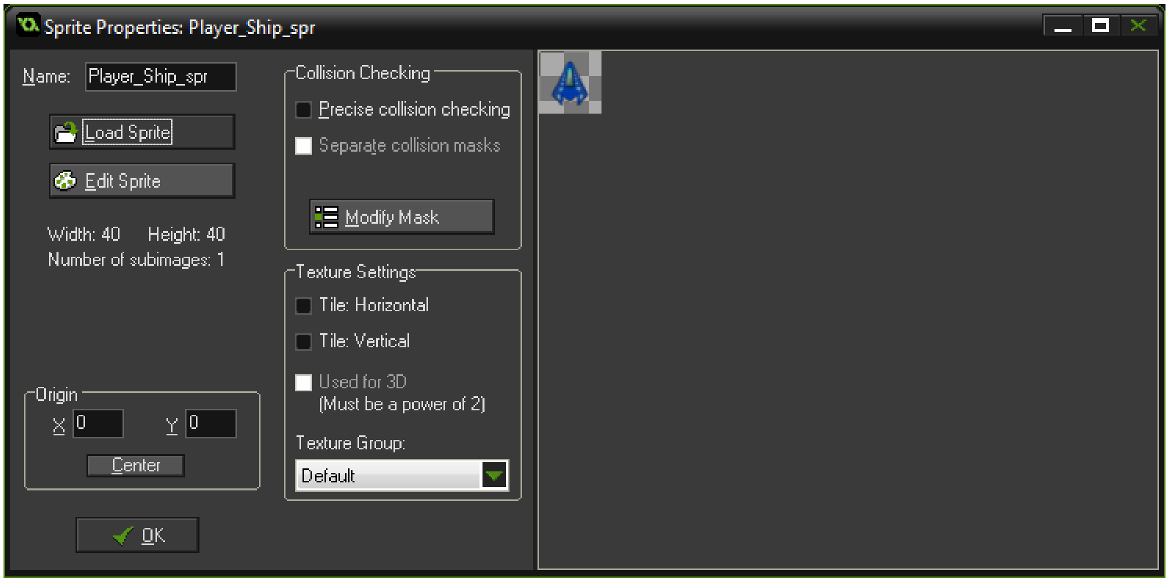
□ Remove Background

□ Smooth Edges

File name: Player.png

Files of type: All Supported File Types

Now you will be back at the Sprite window:



Sprite Properties: Player_Ship_spr

Name: Player_Ship_spr

Width: 40 Height: 40

Number of subimages: 1

Collision Checking

□ Precise collision checking

□ Separate collision masks

Texture Settings

□ Tile: Horizontal

□ Tile: Vertical

□ Used for 3D (Must be a power of 2)

Texture Group: Default

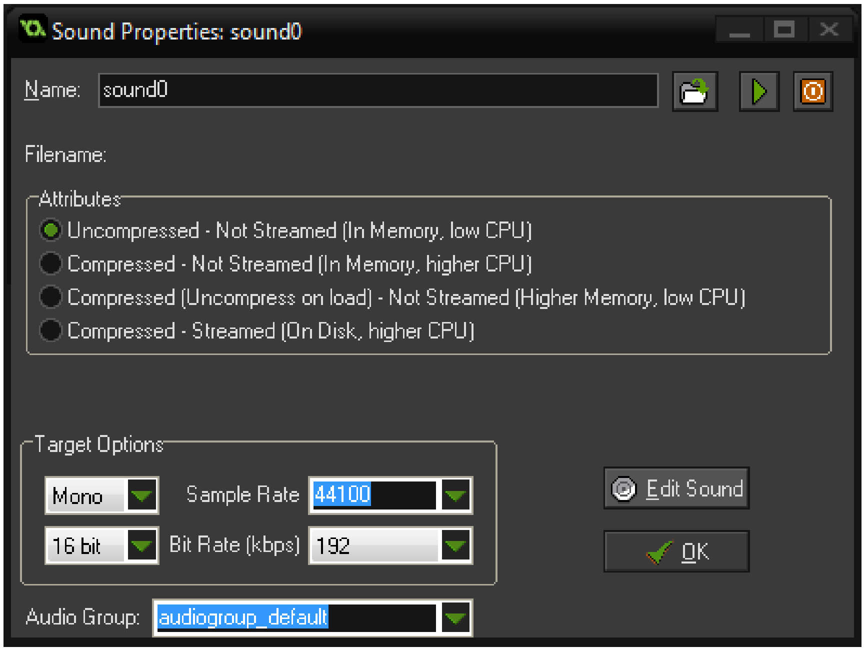
Origin

X 0 Y 0

Be sure to click on the Center button. This will tell GameMaker to use the center of the Sprite as the location to use when placing the Object that uses this Sprite in a room. Only click the center button AFTER you've loaded the sprite!

Please follow the instructions above to now create "Enemy_Ship_1_spr".

Now we need to load in the Sounds effects. Click the Create a sounds button on the toolbar and the following window will open:



Sound Properties: sound0

Name: sound0

Filename:

Attributes

● Uncompressed - Not Streamed (In Memory, low CPU)
○ Compressed - Not Streamed (In Memory, higher CPU)
○ Compressed (Uncompress on load) - Not Streamed (Higher Memory, low CPU)
○ Compressed - Streamed (On Disk, higher CPU)

Target Options

Mono

Sample Rate: 44100

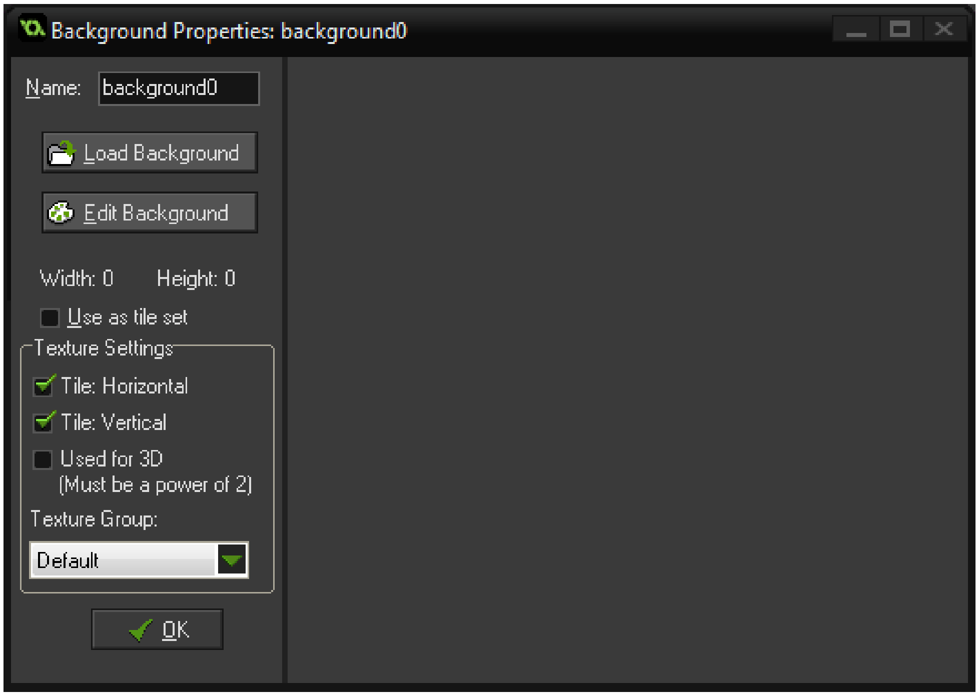
16 bit

Bit Rate (kbps): 192

Audio Group: audiogroup_default

Name this sound Player_missile_snd. Click on the **File Open button** and then choose the Player_Missile.wav from the Sounds folder in your Resources file. Click the open button to load the sound. When back at the Sound Properties window click the **OK** button to save the sound.

Finally we will load the game background image. Click the **Create a Background** button from the toolbar. The following window will open:



Background Properties: background0

Name: background0

Width: 0 Height: 0

☐ Use as tile set

Texture Settings

☑ Tile: Horizontal

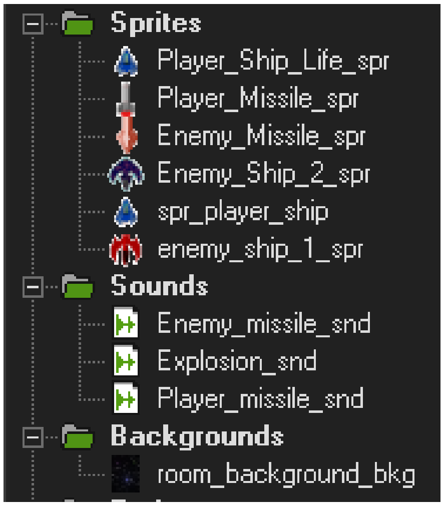
☑ Tile: Vertical

☐ Used for 3D (Must be a power of 2)

Texture Group: Default

Name this Background: Room_Background_bkg. Click on the **Load Background button** and then choose the Room_Background.png from the Backgrounds folder in your Resources file Click the open button to load the background. When back at the Background Properties window click the **OK** button to save the background.

You are now done loading the needed game resources. Your Resources window should now look like this:

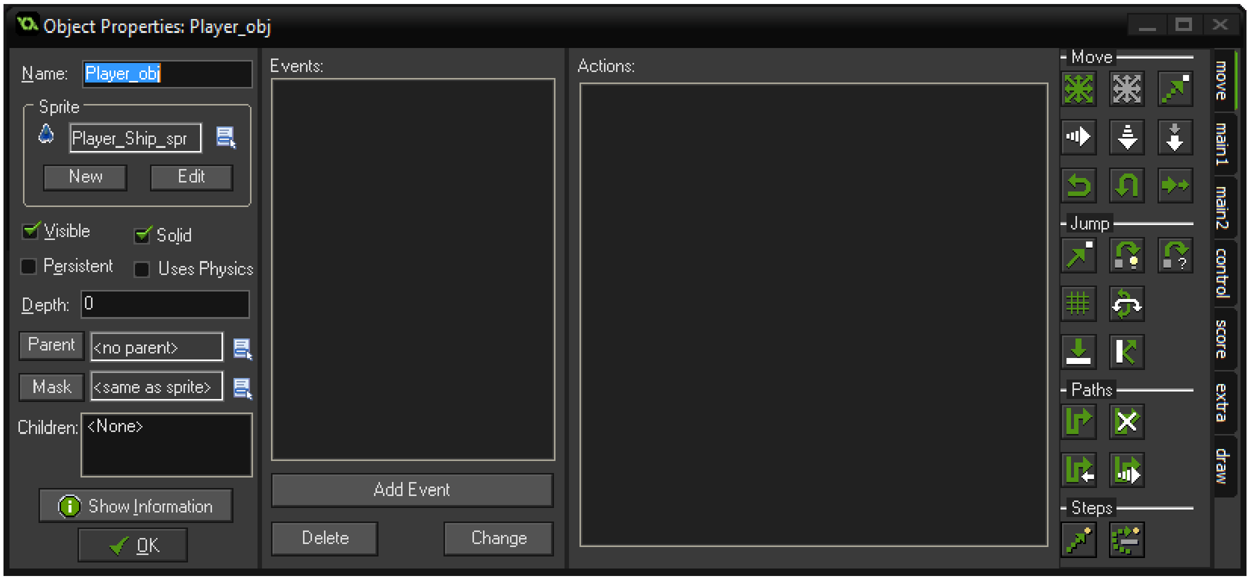


- Sprites
  - Player_Ship_Life_spr
  - Player_Missile_spr
  - Enemy_Missile_spr
  - Enemy_Ship_2_spr
  - spr_player_ship
  - enemy_ship_1_spr
- Sounds
  - Enemy_missile_snd
  - Explosion_snd
  - Player_missile_snd
- Backgrounds
  - room_background_bkg

You might notice here that the resources are not automatically sorted by name. You can re-sort them to your liking by clicking a resource and dragging it to a new location in the folder.

SECTION 3 ►► CREATING THE OBJECTS

Start by creating an object for the player named "Player_obj" and assign it the sprite "Player_Ship_spr". Make sure the Visible and Solid boxes are check marked
 This object is now set up to display the ship sprite whenever it is used in the game.



Object Properties: Player_obj

Name: Player_obj

Sprite: Player_Ship_spr

☑ Visible ☑ Solid

□ Persistent □ Uses Physics

Depth: 0

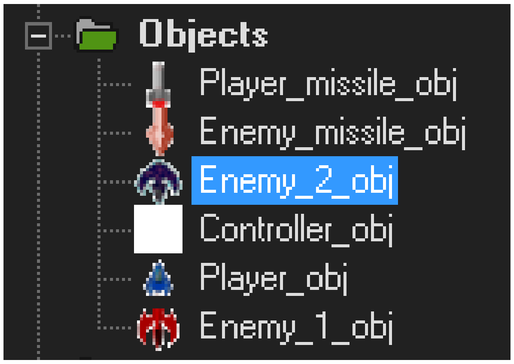
Parent: <no parent>

Mask: <same as sprite>

Children: <None>

Create another object called "Enemy_1_obj", give it the "Enemy_Ship_1_spr" and set **Visible** and **Solid** to check marked.

All of the other objects needed for the game are already created for you. Your Object resource tree should have the following objects:

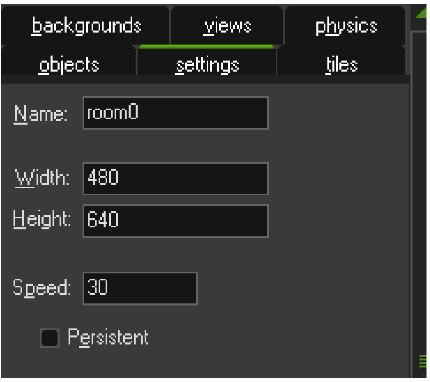


Objects
- Player_missile_obj
- Enemy_missile_obj
- Enemy_2_obj
- Controller_obj
- Player_obj
- Enemy_1_obj

Remember the controller object from Ping? In Vacuum Marauders "Controller_obj" will be responsible for keeping track of the game score and the number of lives the player has left and playing the background music.

SECTION 4 ►► CREATING THE ROOM

Click on the **Create a room button** on the toolbar. Name the room "Main_rm" in the **Settings** tab and set the **Width** = 480 and **Height** = 640 (making the room 480 X 640 pixels in size).



Name: room0

Width: 480

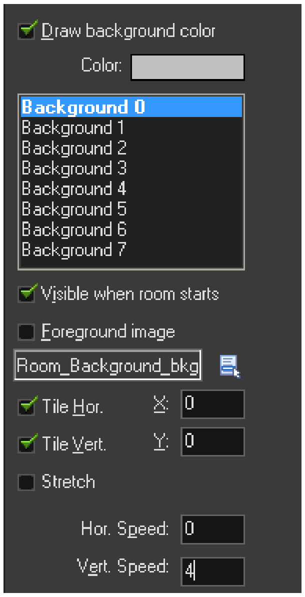
Height: 640

Speed: 30

□ Persistent

d

We will now set the background for the room so that instead of the dull grey, it uses our starscape. Click the **Backgrounds tab** and select "Background 0" from the list. Now choose the background "Room_Background_bkg" from the background listbox. Ensure the **Visible when room starts** box is checked. Put the value 4 in the **Vert**. **Speed** box.



☑ 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

Room_Background_bkg

☑ Tile Hor. X: 0

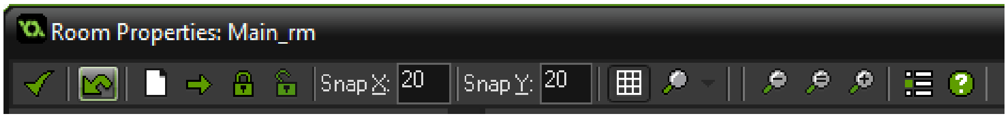
☑ Tile Vert. Y: 0

□ Stretch

Hor. Speed: 0

Vert. Speed: 4

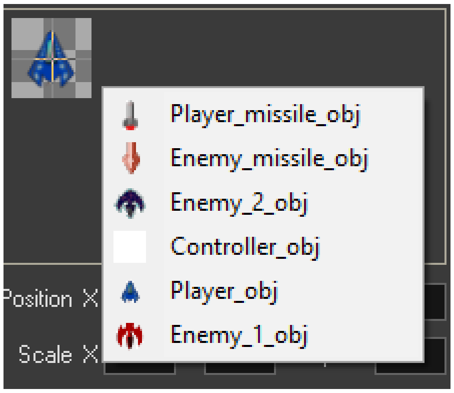
On the Room Properties toolbar, set the **Snap X** and **Snap Y** to be 20. This will make our ship line up better in the grid and make placing objects easier. Grids can also be used to align objects during the game.



Room Properties: Main_rm

Snap X: 20 Snap Y: 20

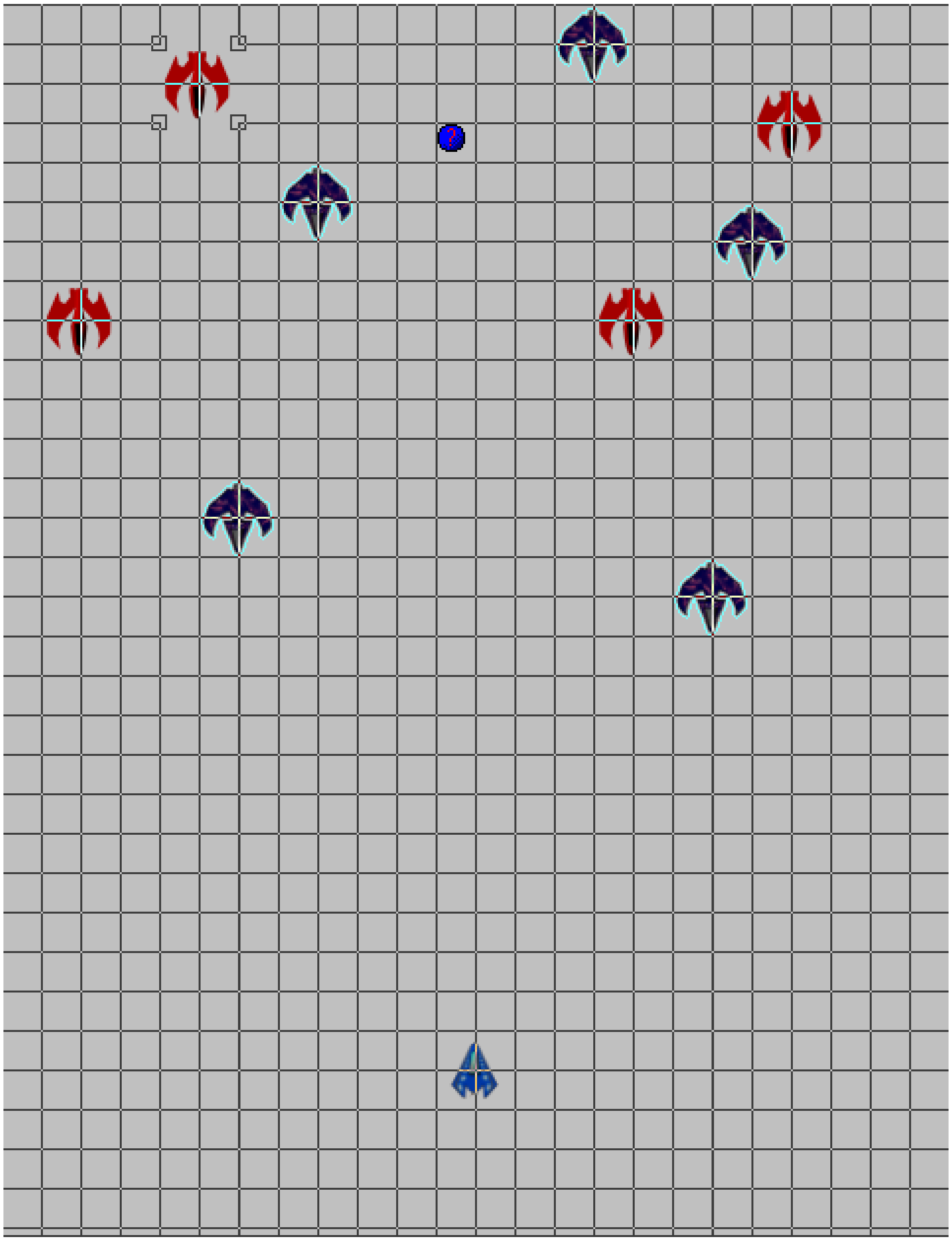
The last thing to do is place the objects into the room. Go to the Objects tab and click in the object window (the large rectangle underneath the word objects) and then select the "Player_obj" object from the listbox.



To place the space ship in the room just move the mouse to some place near the bottom of the room and click the mouse button.

Continue this process by placing several instances of each enemy ship objects in the top half of the screen, and finally place the controller object anywhere on the screen. It doesn't matter where the controller object is since it is not visible and won't be able to collide with any of the other game objects, but it MUST be in the room in order to function!

You should end up with a room that looks something like this (I've turned the background off temporarily in this image to allow you to easily see where the objects have been placed):



Now the room is completed. Close the room by clicking the green check mark to save your work.

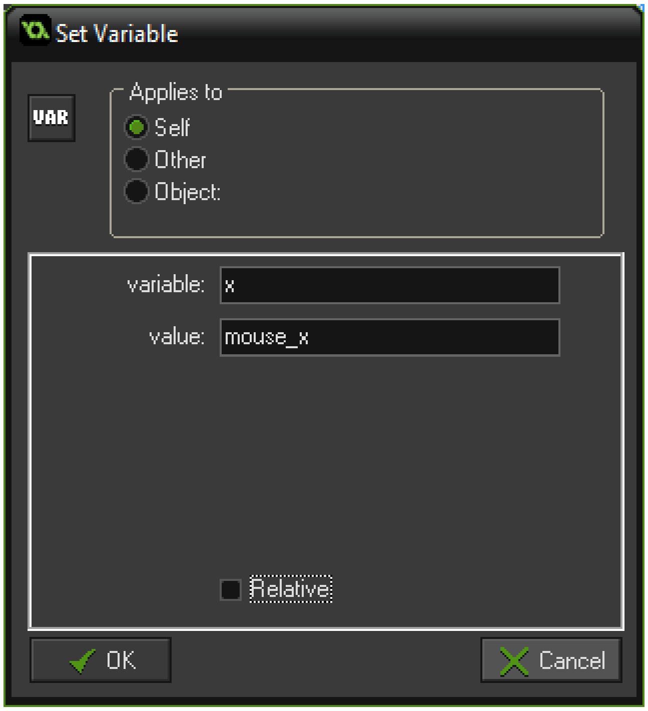
SECTION 5 ►► ADDING PLAYER MOUSE MOVEMENT

Instead of using the keyboard to control the player ship we are going to use the mouse this time.

Open “Player_obj”, select **Add Event**, choose **Step** and then choose **Step** again. The step event is a continuously triggering event. It will complete any actions assigned to it every ‘step’ of the game. A game step is similar to the concept of Frame Per Second (FPS) in a First Person Shooter. In GameMaker the time between each step is determined by the room speed. The initial room speed of all newly created rooms in GameMaker is 30 steps per second. So the Step event in “Player_obj” will complete its assigned actions 30 times each second.

Now assign the **Set Variable** action VAR (Under the Control tab) to the **Step** Event. Set the following values:

- Applies to: Self
- Variable: x
- value: mouse_x
- Relative: unchecked



Set Variable

Applies to

● Self
○ Other
○ Object:

variable: x

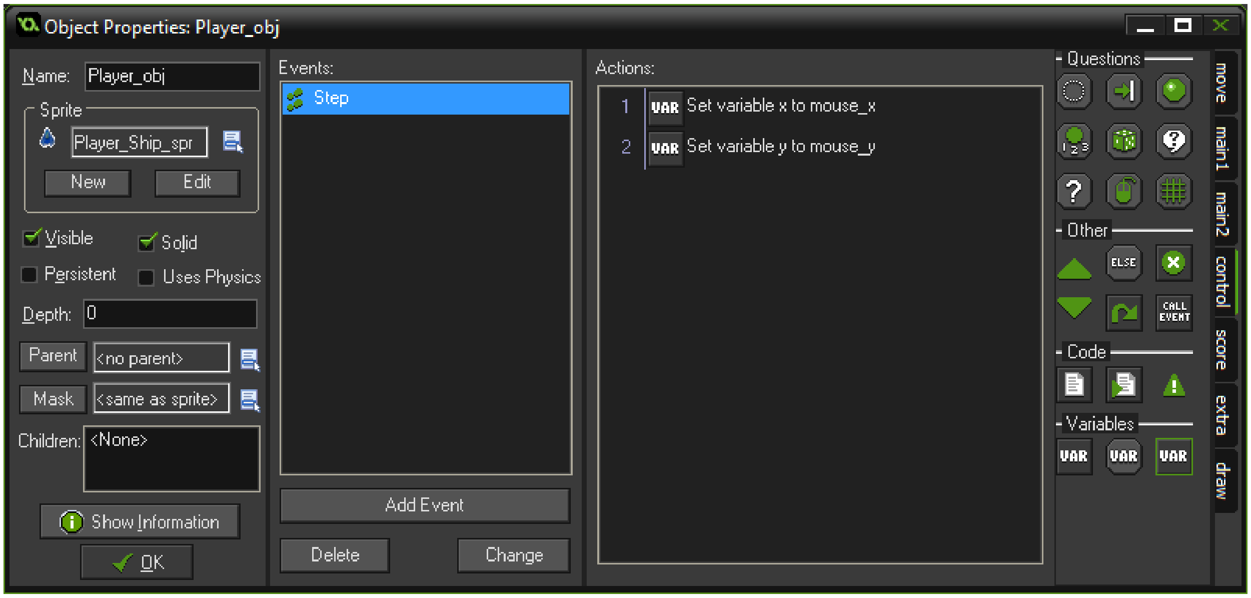
value: mouse_x

□ Relative

Assign one more **Set Variable** action to the **Step** Event. Set the following values:

- Applies to: Self
- Variable: y
- value: mouse_y
- Relative: unchecked

"Player_obj" should now look like this:


Object Properties: Player_obj

Name: Player_obj

Sprite: Player_Ship_spr

☑ Visible ☑ Solid

☐ Persistent ☐ Uses Physics

Depth: 0

Parent: <no parent>

Mask: <same as sprite>

Children: <None>

Events: Step

Actions:

1 VAR Set variable x to mouse_x

2 VAR Set variable y to mouse_y

Go ahead and run the game now. You should see that the ship will follow the mouse cursor around the game screen. Close the running game.

▶▶ Teaching Moment

So how is this all working? We've introduced here a very important concept in Game Design and that is Variables. Variables are a part of every video game and every object within a video game.

So what exactly are variables? I like to think of a variable as something important that varies over time. For instance the amount of money in my wallet is not a constant amount, it varies as I purchase things and get money from an ATM.

In a video game each object has many important aspects to it that can vary over time. The aspect we are concerned with here is the objects position. With "Player_obj" its position on the game screen will be same as the mouse cursor position so we need a way to tell "Player_obj" to go to the same position as the mouse cursor.

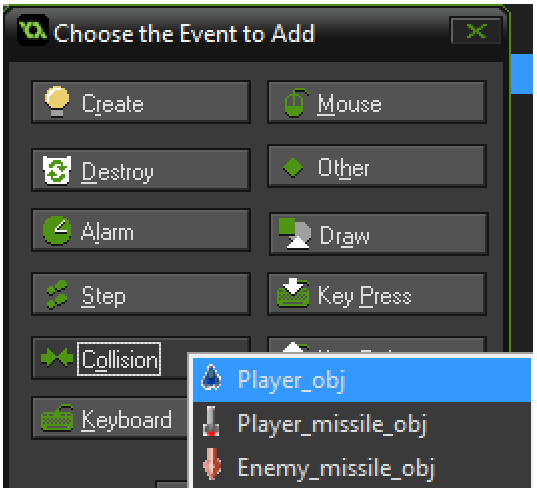
Each object in a video game uses an X, Y (and Z for 3d games) coordinate system as the way to determine where to place the object on the game screen. So for "Player_obj" we can just tell it to use the mouse cursor X position as its X position and the same for Y. This is what the above actions do.

So where did mouse_x and mouse_y come from? These are known as Built in Variables in GameMaker. GameMaker keeps track of numerous aspects of the game play in these special variables for you. You can see all of the Built in Variables by Clicking Scripts on the Main toolbar and then Selecting Show Built in Variables

So you can now move around the room, but you can also move your ship right on top of the enemy ships and nothing happens. We need to take care of this by adding in a collision event.

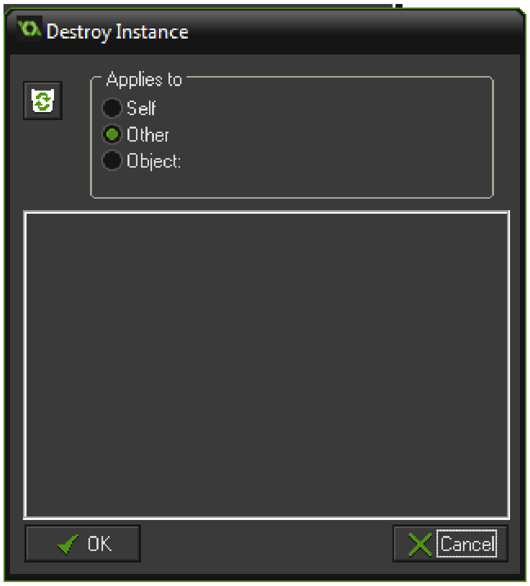
In this case we want a collision between the player and an enemy to result in both being destroyed. We could do this with a collision event in both the enemy and player, but when handling collisions it is often best to have just one object handle all the collision actions. This is so that if something goes wrong it is easier to debug and it also avoids problems about which collision event happens first.

We will handle all these actions through the enemy object, so open "Enemy_1_obj" and add a **collision event** for "Player_obj":



Under the **Collision event** add the **Destroy Instance** action, and set the following:

- Applies to: Other is selected



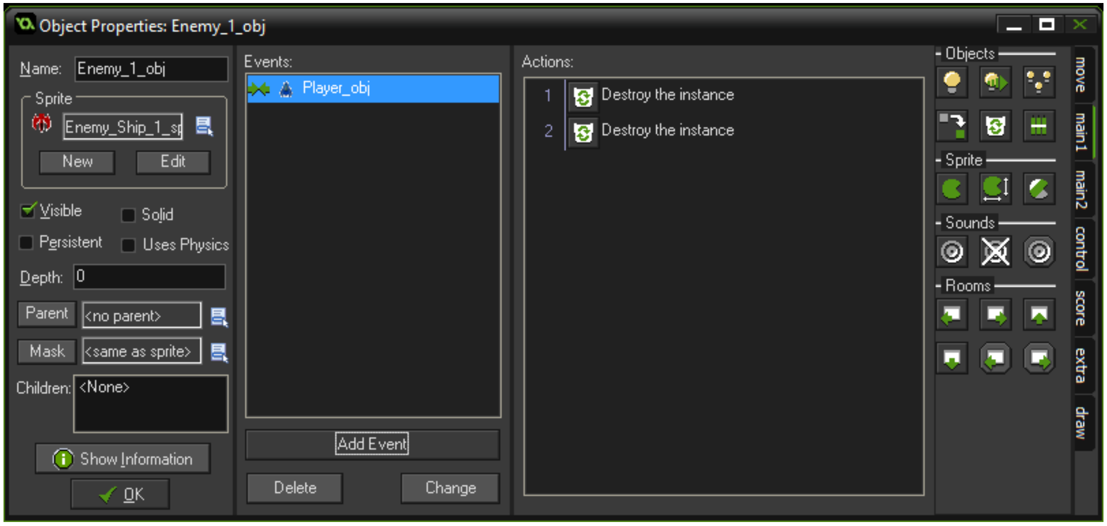
This makes the **Destroy Instance** action apply to the instance that the enemy collided with (the player.) Using the Applies to: Other setting on any action that allows it will have it apply to the instance that the instance declaring the action collided with. This process destroys the player that hit the enemy.

Add another Destroy Instance action , and set the following:

- Applies to: Self is selected

This will destroy the enemy itself.

This is what the Enemy_1_obj should look like now:



Object Properties: Enemy_1_obj

Name: Enemy_1_obj

Sprite: Enemy_Ship_1_sp

☑ Visible ☐ Solid

☐ Persistent ☐ Uses Physics

Depth: 0

Parent: <no parent>

Mask: <same as sprite>

Children: <None>

Events: Player_obj

Actions:

1 Destroy the instance

2 Destroy the instance

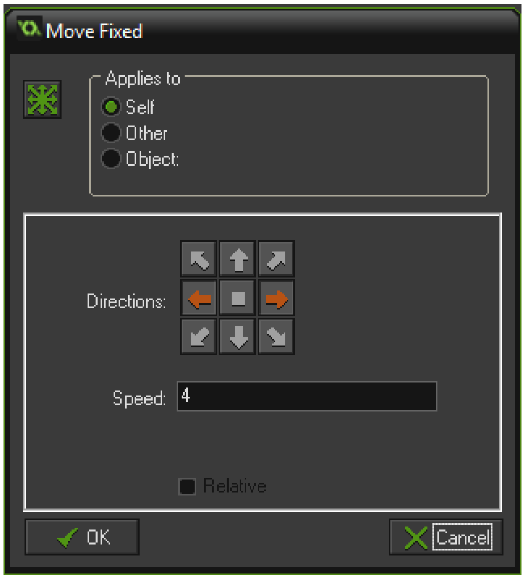
Go ahead and add the above collision with player event and its same actions to the other enemy ship in order for it to be destroyed as well.

SECTION 6 ►► ADDING ENEMY MOVEMENT

The first enemy will be a very dumb enemy. It will just move left and right on the screen while also moving down the screen.

Open "Enemy_1_obj" and add a **Create** event. This event happens as soon as the object is created and is often used to start movement and set up elements the object will need to use. In this case we want to set the enemy moving either to the left or right. Add a **Move Fixed** action (under the Move tab) to the **Create** Event. Set the following:

- Applies to: Self
- Directions: Select both the left and right arrows
- Speed: 4,
- Relative: Unchecked



Move Fixed

Applies to

● Self
○ Other
○ Object:

Directions:

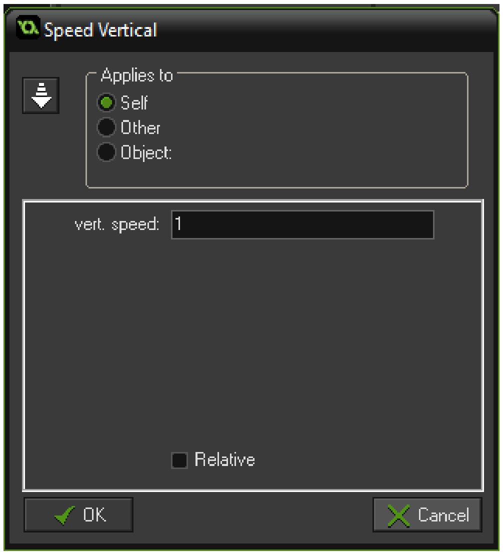
Speed: 4

□ Relative

With both arrows selected the object will randomly choose one of the directions, this means about 50% of the enemies will go left and the other 50% will go right.

Add a **Speed Vertical** action (under the Move tab) to the **Create** event. Set the following:

- Applies to: Self
- Vert. Speed: 1
- Relative: Unchecked


Speed Vertical

Applies to
● Self
○ Other
○ Object:

vert. speed: 1

□ Relative

Go ahead and test the game. You will see that "Enemy_1_obj" moves left or right and down but it will ultimately hit the side of the screen and disappear. This is a problem that we need to fix.

Unlike in Ping where we had walls around the screen that the ball could bounce off of here we need some other method to determine when the sides are hit. We will do this with a new Event called Intersect Boundary. Add the **Intersect Boundary Event** (under the Other Event button) to "Enemy_1_obj". Within this event add a **Reverse Horizontal action** (under the Move tab). Set the following:

- Applies to: Self

Now the "Enemy_1_obj" will move down the screen and move left and right.

►► Teaching Moment

What happens when this enemy moves off the bottom of the screen? Even though it moves off the bottom of the screen it is no longer needed as part of the game but GameMaker is still keeping track of it

Every computer game will have objects that are no longer needed as part of the gameplay, and if they are just kept around, the computer has to keep track of them, and this takes up processor time and memory. If you have objects that are no longer needed get rid of them. In the programming world this is known as "Garbage Collection."

To get rid of the Enemy when if goes off the bottom of the game screen we will add one more event to it. Add an **Outside Room** event (under the Other button) to "Enemy_1_obj" and then add a **Destroy Instance** action (under the Main 1 tab) to it. Set the following:

- Applies to: Self

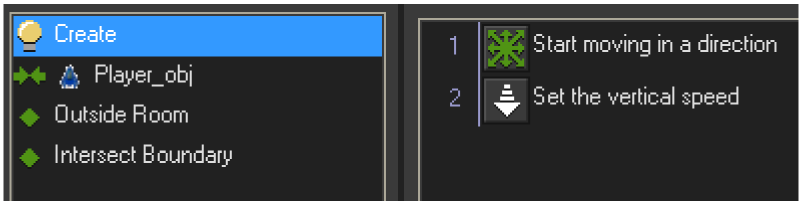
▶▶ Teaching Moment

Have you noticed that the action we use to destroy an object is Destroy Instance and not Destroy Object?

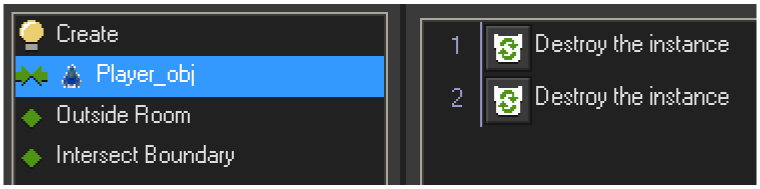
This is an important distinction in Game Design. An Object is like a blueprint. It is what we use to make the actual ‘physical’ items in the game. These ‘physical’ items are known as instances.

While only one type of Object can exist in a game, numerous Instances can be made from an Object. Each of these individual Instance has its own identity and attributes. Just like identical twins look the same, each has their own uniqueness.

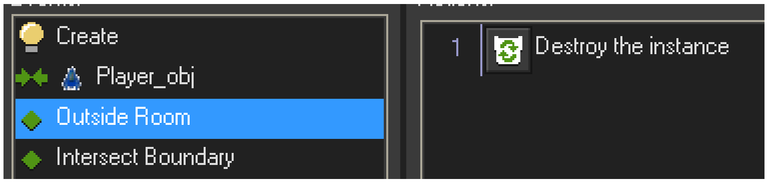
When you are done your events should look like this:



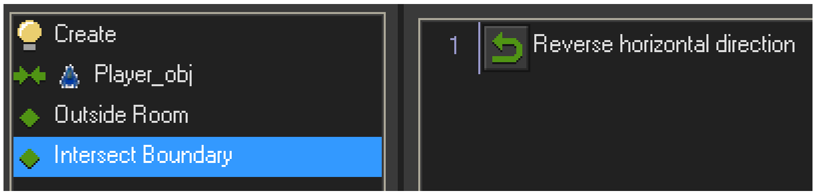
1 Start moving in a direction
2 Set the vertical speed



1 Destroy the instance
2 Destroy the instance



1 Destroy the instance



1 Reverse horizontal direction

Go ahead and close enemy_1_obj as we are done with it for now.

Next let’s add a more intelligent enemy. This enemy will move down the screen directly towards the Player. If the Player can get past (above) the enemy the enemy will then forget about moving towards the Player.

First let's take care of the garbage collection. Open "Enemy_2_obj" an add an **Outside Room** event to it and then add a **Destroy Instance** action to it. Set the following:

- Applies to: Self

Next add a **Step** event (remember choose Step and then choose Step again).

Let's pause for a moment before adding in the actions to get the enemy to move towards the player. We want the enemy to move towards the player until the player gets above the enemy. We in essence want the enemy to make a decision based upon the position of the player.

►► Teaching Moment

Making decisions is a powerful tool to the game designer. You will want your objects to make decisions constantly in order to perform the way you envision your game working. Another term for decisions is conditionals.

For these set of actions we need to first set up a decision action before we let the enemy move towards the player.

We need to decide if the enemy is above the player. We can do this by testing the vertical position of the enemy against vertical position of the player. This would be using the Y axis of the coordinate system.

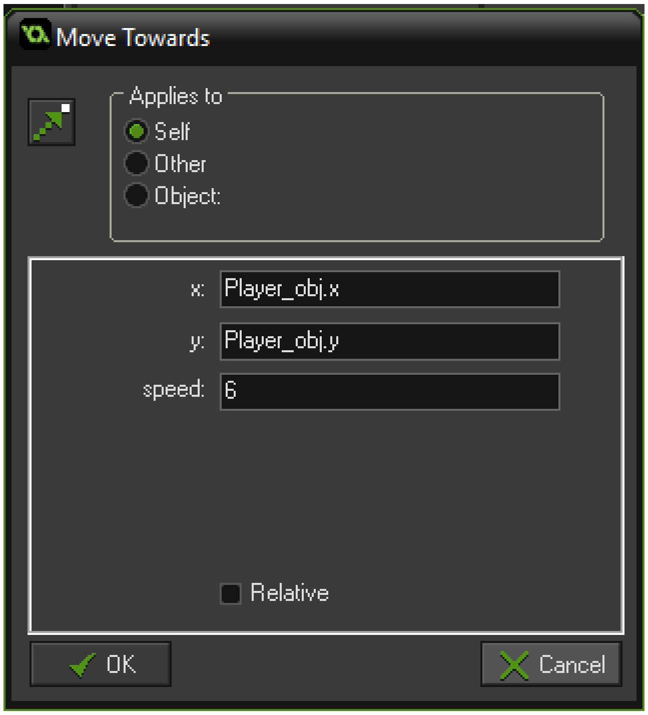
Add a **Test Variable** action VAR (under the Control tab) to the Step Event. Set the following values:

- Applies to: Self
- Variable: y
- Value: Player_obj.y
- Operation: less than
- NOT: unchecked

Next add a **Start Block** ▲ action (under Control tab). There are no attributes to this action.

Next add a **Move Towards** action ↗ (under the Move tab). Set the following:

- Applies to: Self
- x: Player_obj.x
- y: Player_obj.y
- Speed: 6
- Relative: unchecked



Move Towards

Applies to
● Self
○ Other
○ Object:

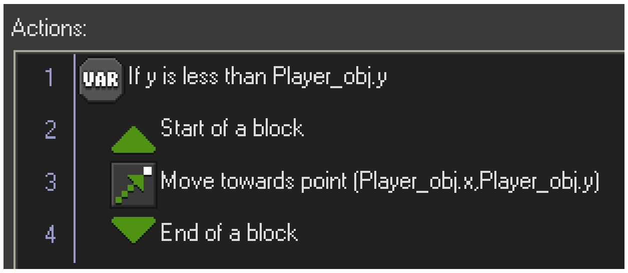
x: Player_obj.x

y: Player_obj.y

speed: 6

□ Relative

And finally add an End Block ▼ action (under Control tab). There are no attributes to this action. Your actions window will look like this:



Actions:

1 If y is less than Player_obj.y
2 Start of a block
3 Move towards point (Player_obj.x,Player_obj.y)
4 End of a block

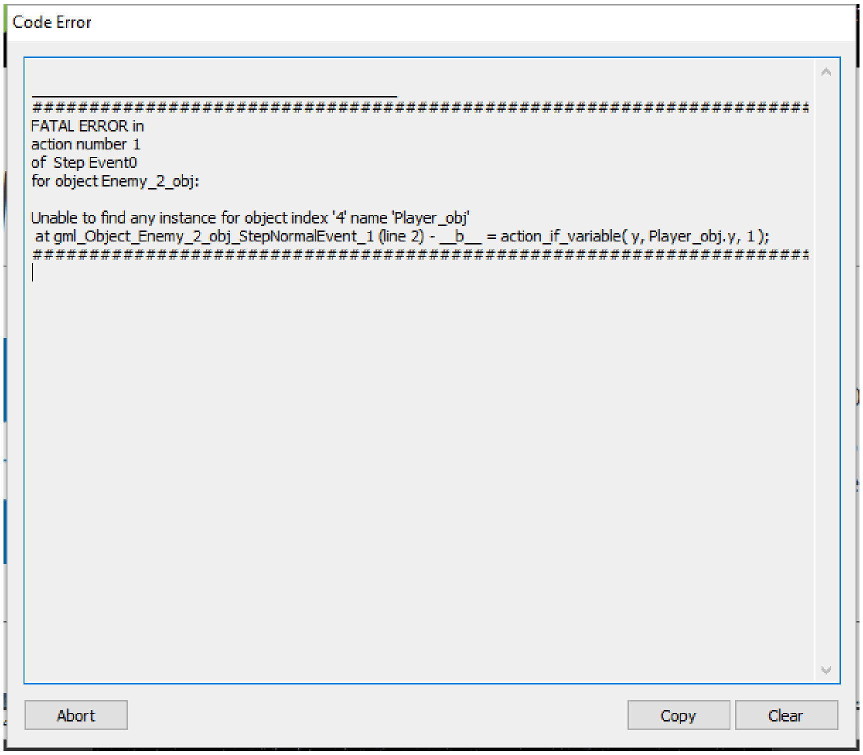
▶▶ Teaching Moment

Notice the variable that was used for the if statement: Player_obj.y? This is a special way of pulling variables out of an instance and is known as Dot notation. What this is saying is: Go look at Player_obj and then pull the y variable from it. Each instance in the game has its own set of variables with it and you can access those variables by using Dot notation.

▶▶ Teaching Moment

So you are probably asking "Why did you put those **Start Block** and **End Block** actions in there?" That's a very good question! When you have an object make a decision, you will always follow that decision with one or more actions. You use the **Start Block** to tell the decision that 'Here is the start of the actions I want you to do if the decision is true'. The **End block** signals the end of the actions to be completed by the decision. You will also notice that the **Start Block** indents to the right a little to also indicate to you that these actions fall within the above decision.

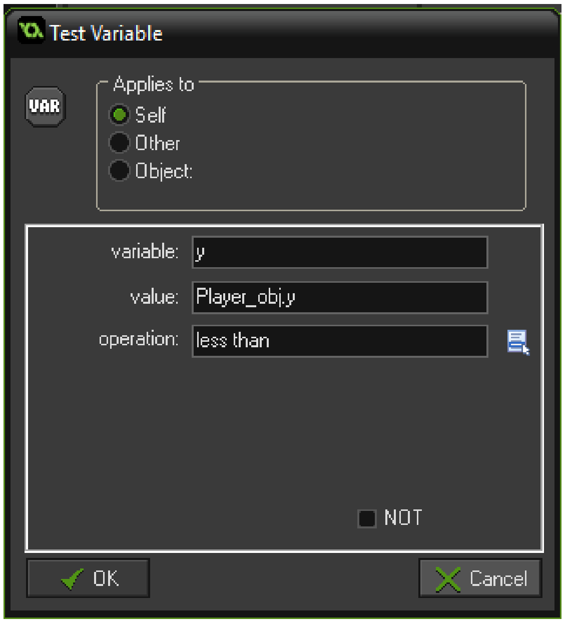
Go ahead and play the game a bit. Now try this: Collide your player ship with Enemy_1_obj but make sure that Enemy_2_obj is still on the screen. You should get the following error:



```
___________________________________
######################################################################
FATAL ERROR in
action number 1
of  Step Event0
for object Enemy_2_obj:

Unable to find any instance for object index '4' name 'Player_obj'
 at gml_Object_Enemy_2_obj_StepNormalEvent_1 (line 2) - __b__ = action_if_variable( y, Player_obj.y, 1 );
######################################################################
```

Hmmm…. What happened here? The error says "Unable to find any instance for object index '4' name 'Player_obj'. So it's telling us that it expected to find Player_obj but it wasn't available. Let's continue. The next part of the error is the important part. It tells us where the error happened, in this case within Enemy_2_obj. Specifically within the if_variable action. If variable is another name for Test variable. So the error happened within the Test variable action. Let's look back at this action:



Applies to
● Self
○ Other
○ Object:

variable: y
value: Player_obj.y
operation: less than
□ NOT

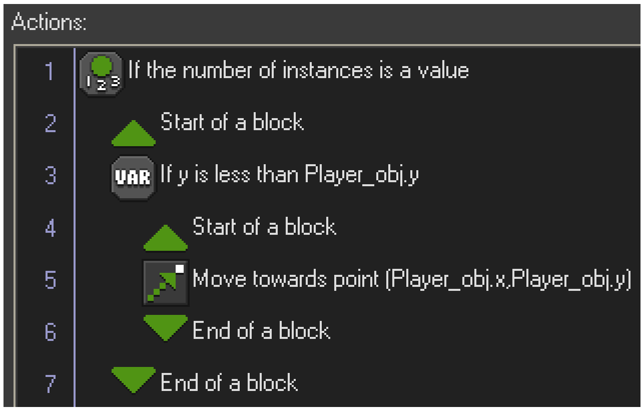
This action tests the y variable within Player_obj against the y variable within the Enemy_2_obj. Ah ha! There is the problem! How can the game do this test if the Player_obj has been destroyed? It can't and that is why the game showed an error.

Since we don't want any errors we need to fix it. To do this we are going to add in an additional test to ensure the Player_obj exists before we do the above test.

At the top of the Enemy_2_obj Step Event add in a Test Instance Count action (under the control tab). Set the following:

- Object: Player_obj
- Number: 0
- Operation: larger than
- Not: unchecked

Then place start and end blocks around the other test code. Your step event should look like this now:



```
If the number of instances is a value
    Start of a block
    If y is less than Player_obj.y
        Start of a block
        Move towards point (Player_obj.x,Player_obj.y)
        End of a block
    End of a block
```

Re-play the game and you should not encounter the error again.

So what is the deal with these Start Blocks and End Blocks? Well, they are used as part of the Test actions (Test Variable and Test Count are just two examples) to ensure that only certain sets of actions are executed if the Test is True. They are very similar to using parenthesis in Math. Using the Step Event picture from above I'll explain how the blocks work.

- At line 1 a Test is performed.
    - If the test is False none of the actions between lines 2 and 7 will be performed.
    - If the test is True then the test at line 3 is performed.
        - If the test is False none of the actions between lines 4 and 6 will be performed
        - If the test is True then the action at line 5 is performed.

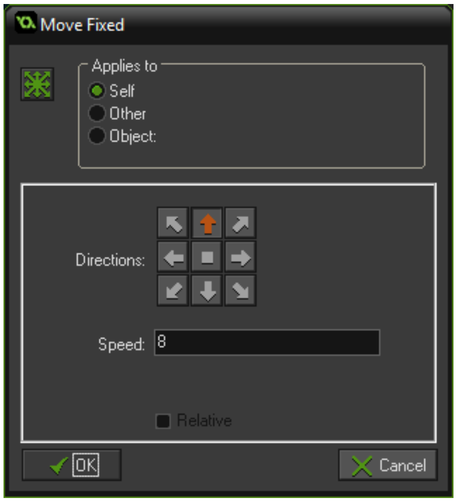
SECTION 7 ►► MAKE THE PLAYER SHOOT

Ok, things are looking good, but we are missing the something here… destruction! We will have to give the player something to shoot at the enemy. The object we will be shooting is "Player_missile_obj".

The whole point of the missile is to travel straight up and collide with the enemy, so we will have it start moving up the screen as soon as it is created.

Go ahead and open "Player_missile_obj" and add a Create event, and put in a **Move Fixed** action and Set the following:

- Applies to: Self
- Directions: Up arrow pressed
- Speed: 8
- Relative: unchecked


Move Fixed

Applies to
● Self
○ Other
○ Object:

Directions: ↑

Speed: 8

□ Relative

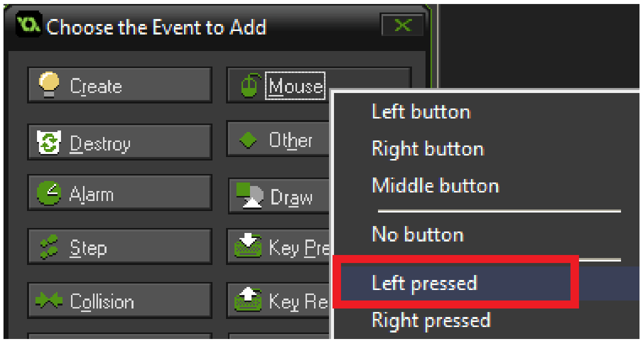
Now add in a Play Sound action with the following set:

- Sound: Player_Missile_snd
- Loop: False

One last thing we need to do here is get rid of the missiles if they miss an enemy and leave the screen. Add the **Outside Room** event and add a **Destroy Instance** action to the event with the following set:

- Applies to: Self

Now we need the player to shoot the missiles. Open "Player_obj" and add a Mouse event for Left Pressed.



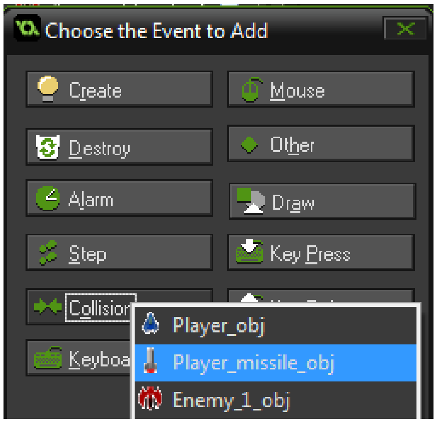
Inside the event, add **Create Instance** action (under the Main1 tab). Set the following parameters:

- Applies to: Self
- Object: "Player_missile_obj"
- X: 0
- Y: 0
- Relative: checked

Did you notice that we are using Relative here? This is a very important tool to use in working with objects. We use Relative because we want it to appear at "Player_obj"s point (0,0) – which is the center of its ship sprite. If we didn't use Relative, it would always appear at the ROOM's (0,0) position which is in the upper left hand corner of the room. Relative always says that you are referring the CURRENT state of the object you are putting the action into.

One last thing we need to do here is have the missile destroy an enemy. We do this by using a **Collision** event.

We will handle all the actions through the enemy object, so open "Enemy_1_obj" and add a **collision event** for "Player_missile_obj":



Under the **Collision** event add the **Destroy Instance** action and set the following parameters:

- Applies to: Other

Add another **Destroy Instance** action and set the following parameters:

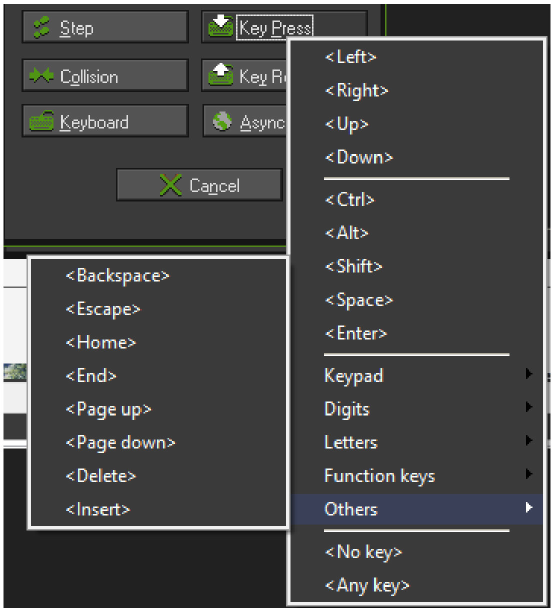
- Applies to: Self
-

This will destroy the enemy itself.

So now when the player shoots a missile it will destroy "Enemy_1_obj". You will need to add the above collision with missile event and its same actions to the other enemy ship in order for it to be destroyed as well.

One more thing you will do here to make testing the game a bit easier is to add in the ability to restart the game while you are playing, instead of having to close the game and re-play it within GameMaker.

Open the Controller_obj and add in a KeyPress Event for the Escape key:



Within this event add in a Restart Game action (under the Main 2 tab). This will allow you to restart the game at any time by pressing the ESC key.

SECTION 8 ►► LET THE ENEMY SHOOT BACK

It's not very fun if we can destroy the enemy but they can't do anything to us. Let's fix this by letting the enemy shoot at us.

Open "Enemy_1_obj" and Add in a **Step** event. Within the **Step** event add a **Test Chance** action 🎲 (under the Control tab). Set the following:

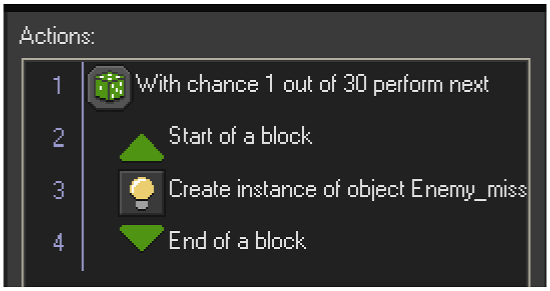
- Sides: 30
- Not: unchecked

This action will test at each game step whether or not to do the actions that follow it. We are setting this to 30 to have the enemy ships only fire at the player about once every 30 steps (about once a second).

Follow the **Test Chance** action with a **Start Block** action. Follow that with a **Create Instance** Action. Within the **Create Instance** action set the following parameters:

- Applies to: Self
- Object: "Enemy_Missile_obj"
- X: 0
- Y: 0
- Relative: checked

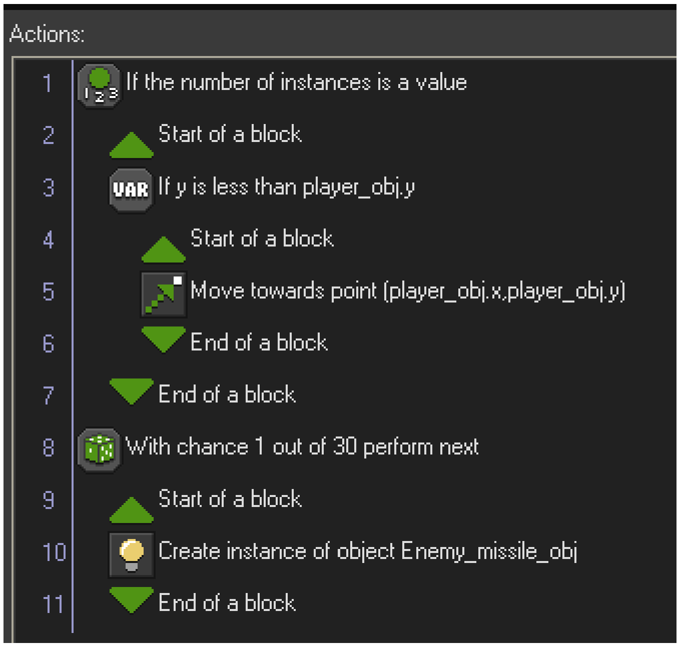
Finish up with an **End Block** action.



Actions:

1 With chance 1 out of 30 perform next
2 Start of a block
3 Create instance of object Enemy_miss
4 End of a block

Repeat the above events and actions for "Enemy_2_obj". The step event for "Enemy_2_obj" should look like this:



Actions:

```
If the number of instances is a value
    Start of a block
    If y is less than player_obj.y
        Start of a block
        Move towards point (player_obj.x,player_obj.y)
        End of a block
    End of a block
With chance 1 out of 30 perform next
    Start of a block
    Create instance of object Enemy_missile_obj
    End of a block
```

We now need to make sure the missile will move down the screen. Open Enemy_missile_obj and add in a Create Event. To this add a Move Fixed Action with the following settings:

- Applies to: Self
- Directions: Down arrow pressed
- Speed: 10
- Relative: unchecked

Next we will destroy the missile if it leaves the bottom of the screen. and add in a Outside Room Event. To this add a Destroy Instance Action with the following settings:

- Applies to: Self

The final thing we need to do is make the missile destroy the player if it gets hit.

Open “Player_obj” and add in a **Collision with** “Enemy_missile_obj” action. Under the Collision event add the **Destroy Instance action**, set the following

- Applies to: Self

Add in one more **Destroy Instance** action with the following settings:

- Applies to: Other

Go ahead and run the game and test that the enemy shoots and the missiles destroy the player ship.

SECTION 9 ►► CHECKING FOR VICTORY

We want the player to win the game after destroying all of the enemy. To do this we will need to do two test statements checking for the existence of both type of enemy objects. Open "Controller_obj" and add in a Step event. Add in a Test Instance Count action and set the following parameters:

- Object: "Enemy_1_obj"
- number: 0
- operation: equal to
- not: unchecked

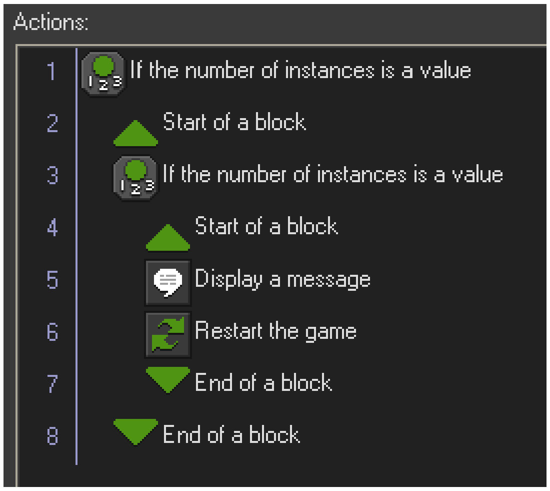
Add in a start block and then add another Test Instance Count action and set the following parameters:

- Object: "Enemy_2_obj"
- number: 0
- operation: equal to
- not: unchecked

Add in another start block and then add a Display Message action (under the Main 2 tab) and set the following parameters:

- message: You've Won!

Follow this action with a Restart Game action and 2 end blocks. Your step event should look like the following:



Actions:

```
If the number of instances is a value
  Start of a block
  If the number of instances is a value
    Start of a block
    Display a message
    Restart the game
    End of a block
  End of a block
```

Test the game and kill all the enemy to ensure your code works.

SECTION 10 ►► FINISHING UP THE TUTORIAL

Now we should have the following features implemented:

- The player ship moves by mouse
- The player ship fires missiles
- Missiles and Enemy are removed from the game when they exit the game window
- Enemy Ships move and fire missiles
- A background that scrolls
- The game checks for victory and displays a message

We will be moving on to Part 2 of Vacuum Marauders next week where we will amp up things by adding much better graphics, better and smarter enemy movement and animation.

SECTION 11 ►► MAKING THE GAME YOURS

As in Ping, making the game yours **is worth 25% of the points** of the assignment. Put about 25% of time into this section you as it took you to complete the tutorial portion of the assignment. You will be graded on the effort shown.

You have several different paths to follow to meet this requirement:

- PROGRAMMING. This path is for students who interested in the Programming path of game design. Some ideas are:
  - Restrict the player movement to the bottom half of the screen
  - Create power-ups dropped by the enemy
- GRAPHIC DESIGN. This path is for students who are interested in the Graphical path of game design. Some ideas are:
  - Create new sprite images.
  - Animate some of the sprites
  - Create a new background
- SOUND DESIGN. This path is for students who are interested in the Audio path of game design. Some ideas are:
  - Create new sounds
  - Create a background sound-track
- NARRATIVE DESIGN. This path is for students who are interested in the Narrative (or story telling) path of game design. Some ideas are:
  - Create a backstory for the game
  - Create dialog appropriate for the characters in the game.

You can use multiple paths to make the game yours if you like. You might also notice that some of the ideas mentioned above talk about doing things that haven't been discussed in this tutorial. You can learn how to do these things by researching the topic at the GameMaker Studio documentation website: docs.yoyogames.com, or by talking with your instructor.

Once you are satisfied with your game, it is ready to be turned in.

To turn in your game Export the game to a .gmz file and upload to the appropriate link in Moodle.