Cosmic Shooter

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.

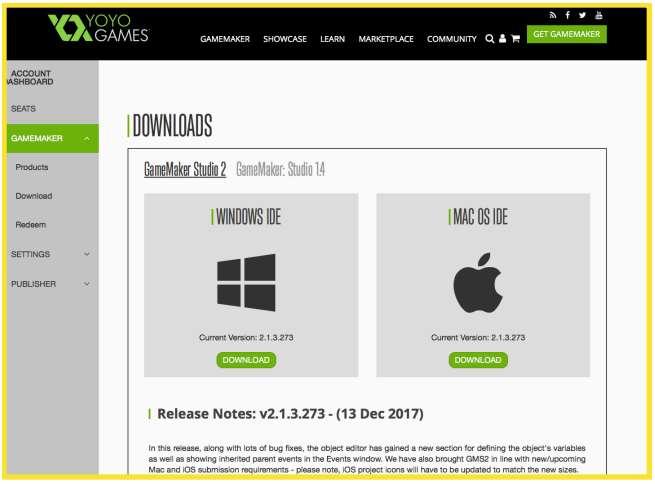
There is no prior experience required and I have the entire project fully solved on GitHub that can be downloaded at: https://github.com/maubanel/cosmic-shooter.

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

01

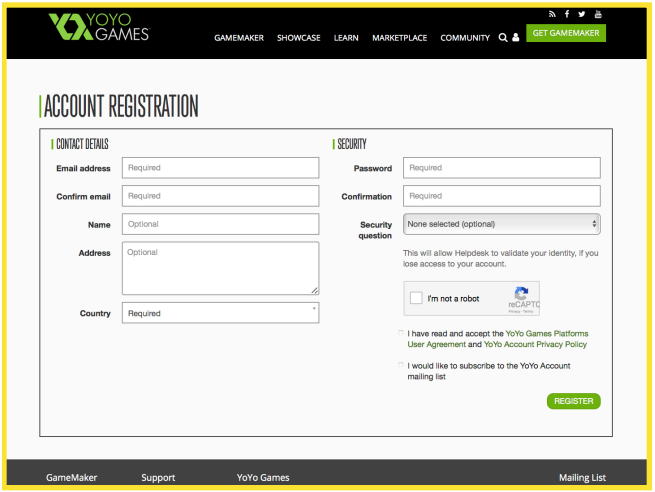
Download the free trial of **GameMaker Studio** at https://www.yoyogames.com/get.

They have a PC and a MAC version.



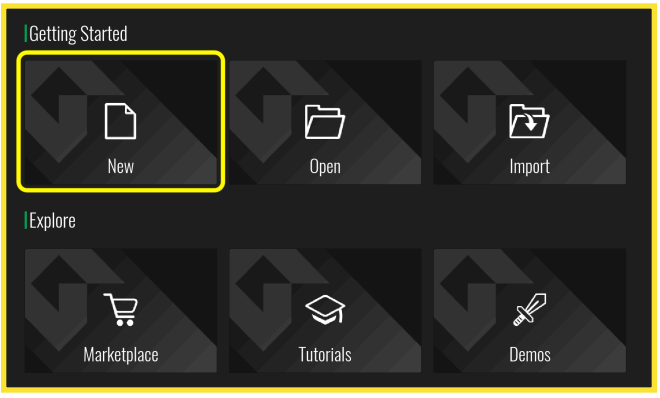
02

As of the writing of this document there was still a free trial. You will have to register with their website.



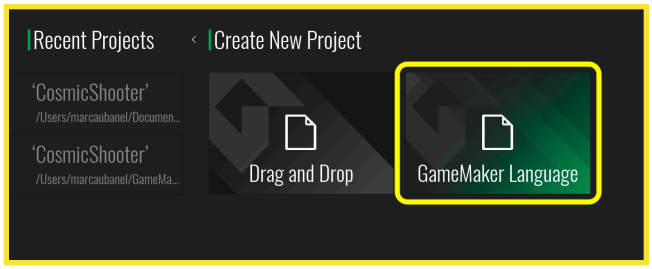
03

Press the ***New*** button in the first Getting Started screen



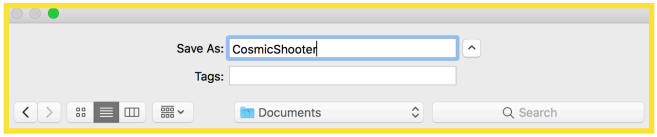
04

Press the ***GameMaker Language*** button



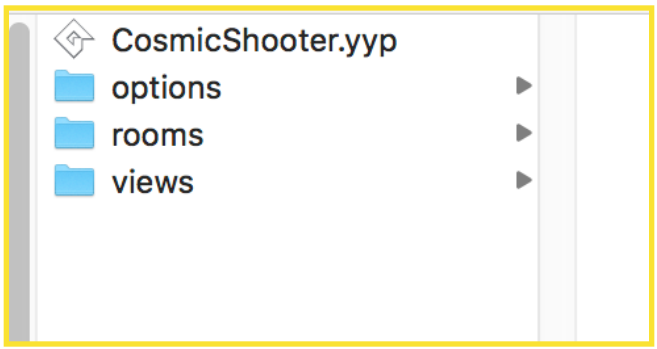
05

Name the project **CosmicShooter** and place it in the directory you want to work in.



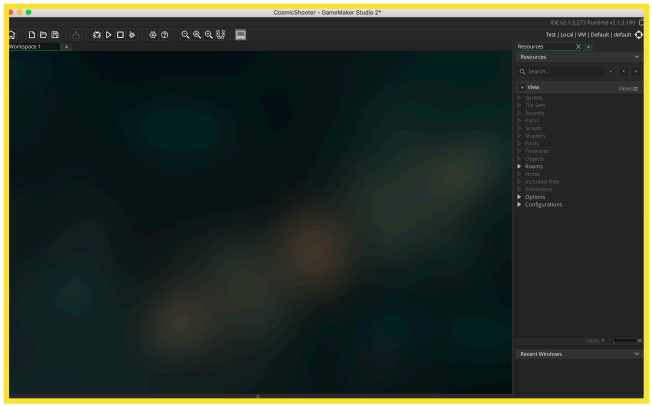
06

This creates a folder called **/CosmicShooter** and puts some files and folders. Feel free to move this later but make sure you have saved it and have quit the GameMaker software.



07

The software automatically launches and you will see a blank project.



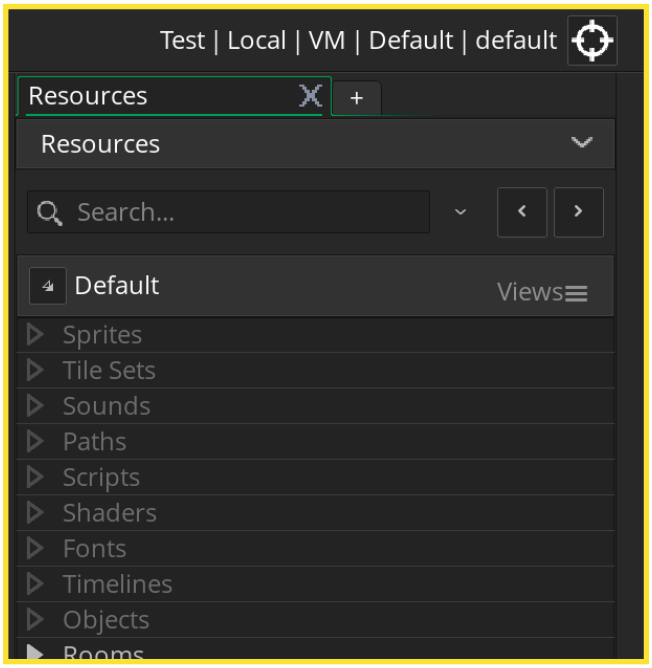
08

If we look on the right hand side we will see a **Resources** menu that holds all of our game assets or content.

We have to be careful with naming these resources though. GameMake does not allow for files to share the same name.

So as a naming convention we typically use camel case with a lower case of Resource type.

Lets see this in action…

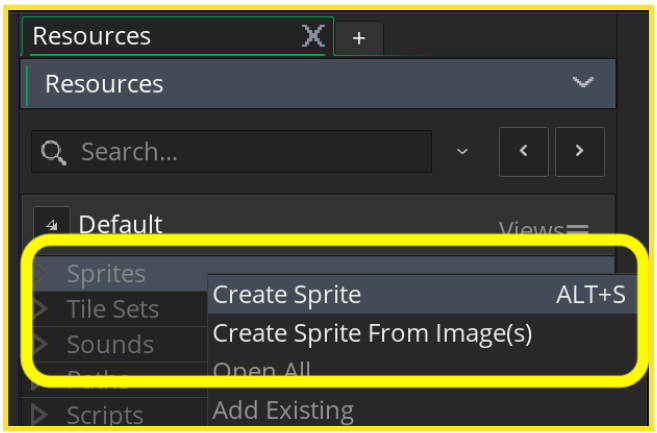


09

The first thing we want to do is add a spaceship. We call an image in games a ***Sprite***.

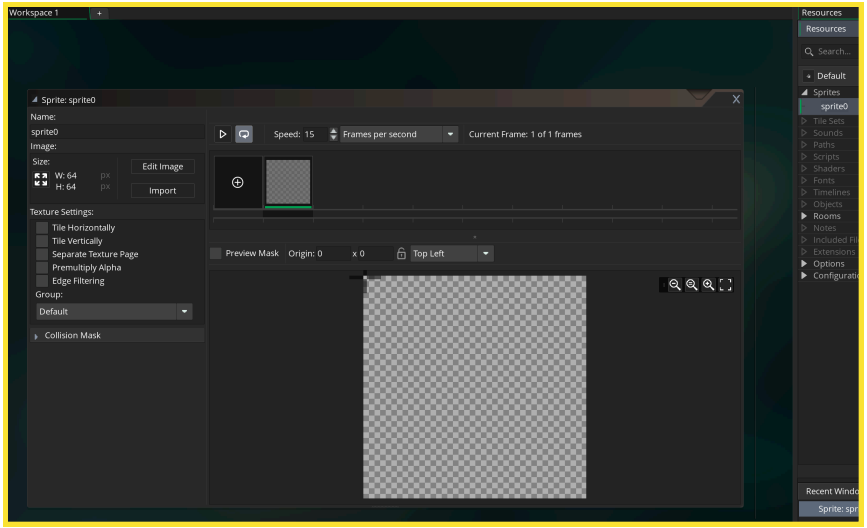
This is an image made up of pixels, very much like what you would create in software such as ***Photoshop***.

Right-Click *Sprites* and select the **Create Sprite** menu item.



10

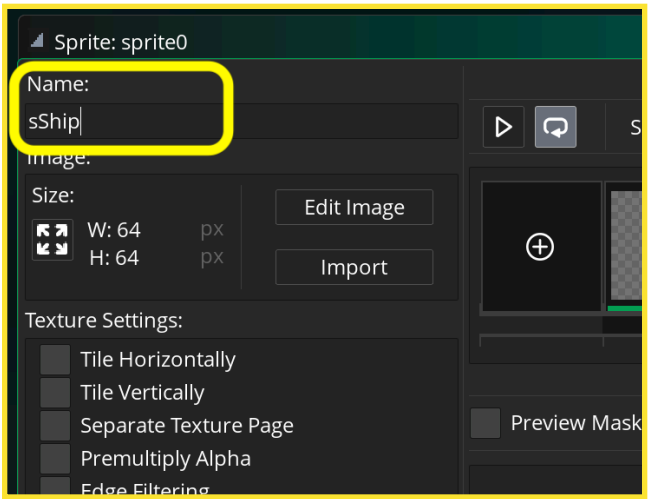
This brings up in our main Workspace Sprite editing window.



11

Click in the input below ***Name:*** and enter the name **sShip**.

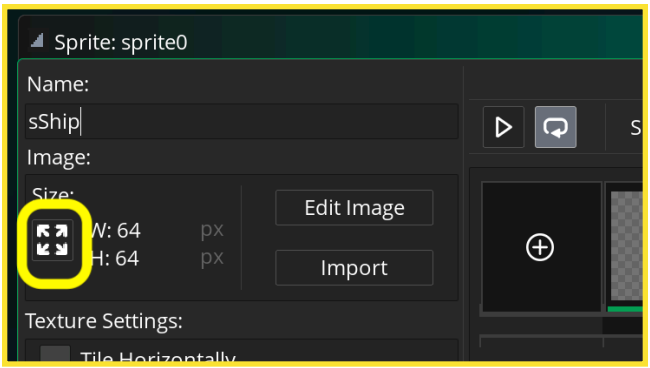
Again we put an small s in front of the name so that we can name other resources that are part of the Ship with Ship in the name without having two files of the same name. We want no collisions.



12

Click in the input below ***Name:*** and enter the name **sShip**.

GameMaker Sprites default to a square 64 x 64 pixels. We want our spaceship to be smaller so we click on the four arrows button…

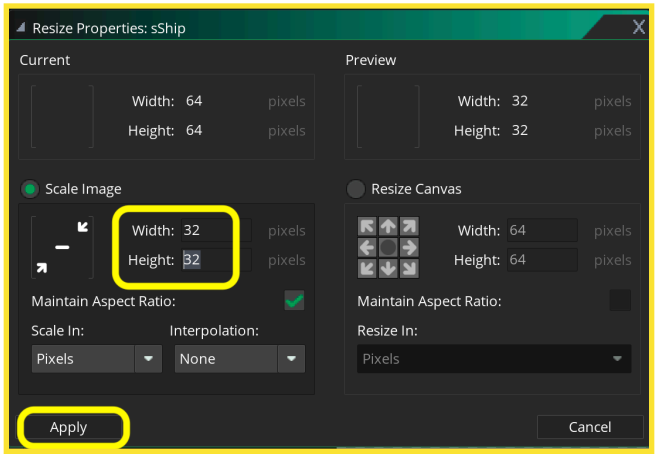


13

This brings up the ***Resize Property*** window in your workspace. Now since there is no image we can either scale the image or resize the canvas.

Be careful though, if there was an image already there, this would make a difference. We want to maintain the square ratio so the ***Maintain Aspect Ratio*** can be selected.

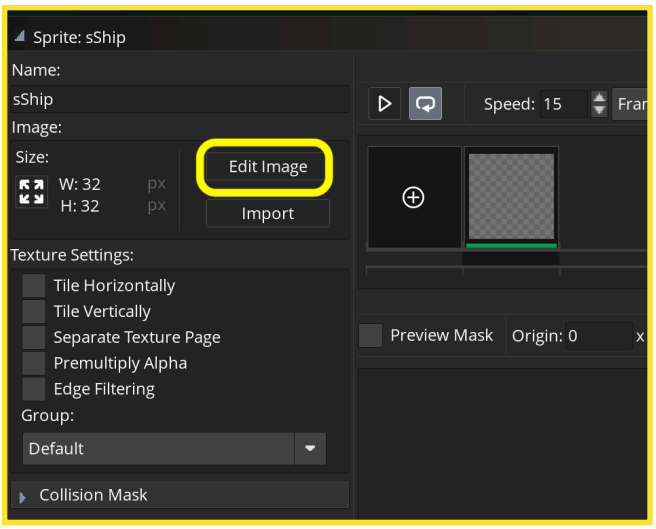
Change the ***Height*** and ***Width*** to **32 x 32** then **press** the ***Apply*** button.



14

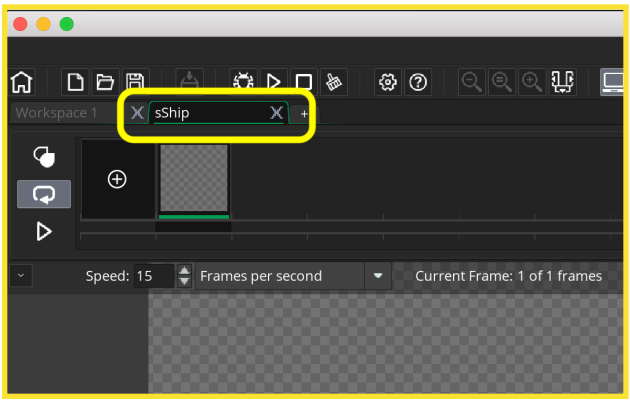
This brings us back to the sprite menu and we want to draw the spaceship.

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



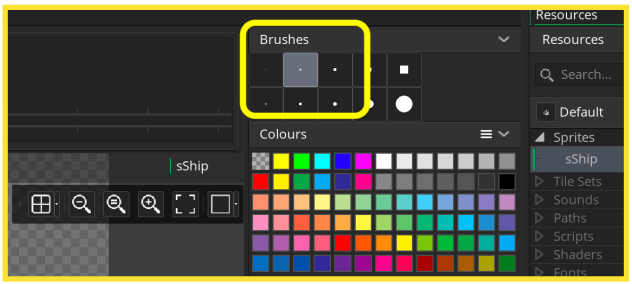
15

We are now no longer in our main ***Workspace***. GameMaker has opened up a new Tab and we are now in an imaging editing menu.



16

On the right hand side there are some ***Brushes***. We want the second smallest brush, so press on the second box.



17

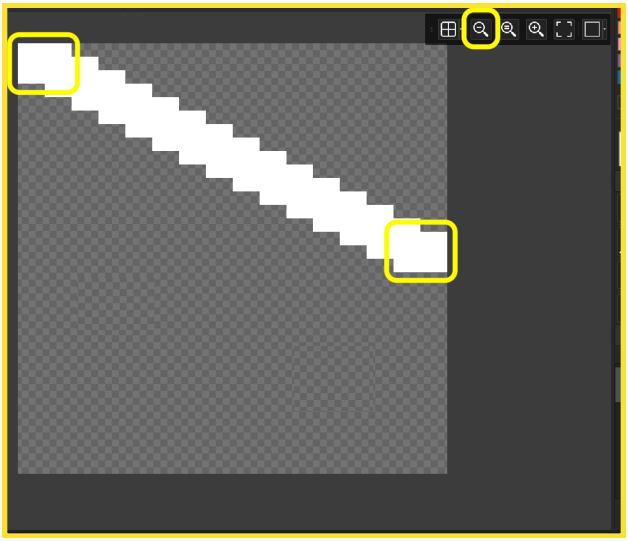
In the ***Toolbox*** below, select the polygon tool (this looks like a triangle. There are two diagonal halves. One is for the **fill** and one is for the **stroke** (to select both you need to press shift and press both halves).

Press the top left half for just the **stroke**.

18

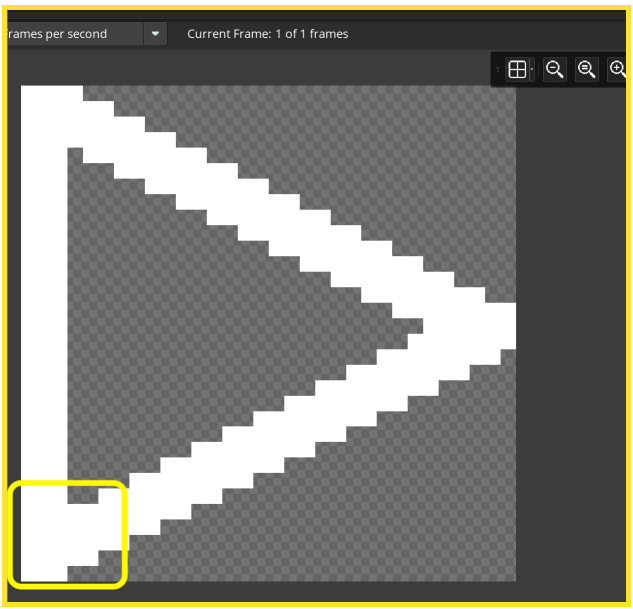
We will stick with the default white color. Zoom out so you can see the entire editing surface (the checkerboard). **Left-Click** the top left corner, then **Left-Click** to right middle edge.

You will see the polygon tool now fills in the line.



19

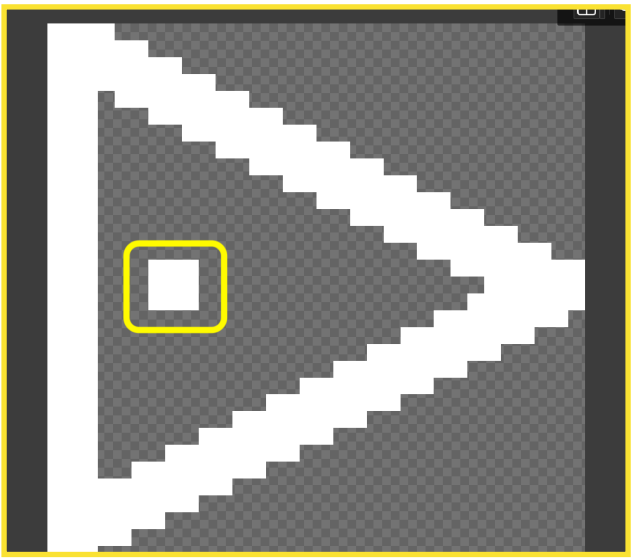
Left-Click the bottom left corner and you will see that the tool fills in the triangle.



20

Now we want the ship pointing to the right. This in GameMaker is an angle of 0° in the polar coordinate system.

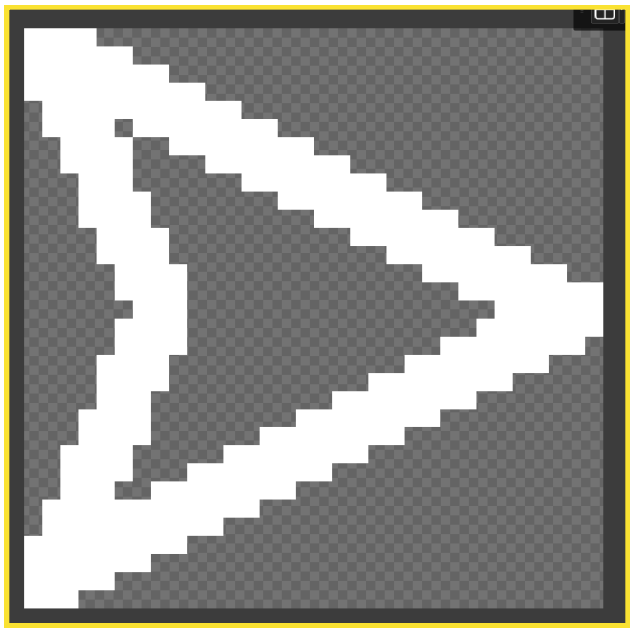
So to make sure we see this as a ship pointing right, lets give it a tail. Put the cursor just inside the middle of the back of the ship on the left hand side.



21

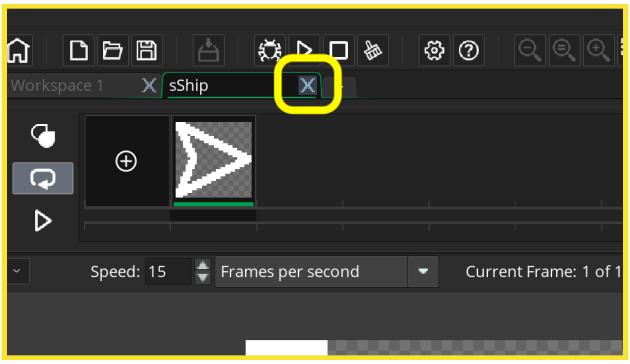
Press down and it alters the shape of the triangle.

Ok, now we have a completed spaceship.



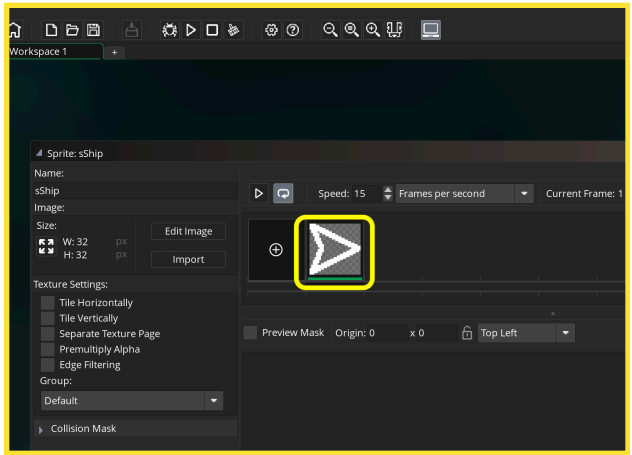
22

Now, we can close the ***sShip*** tab and go back to our main ***Workspace*** tab.



23

Now we are back in our main ***Workspace*** and you should now see the ship sprite we just created.

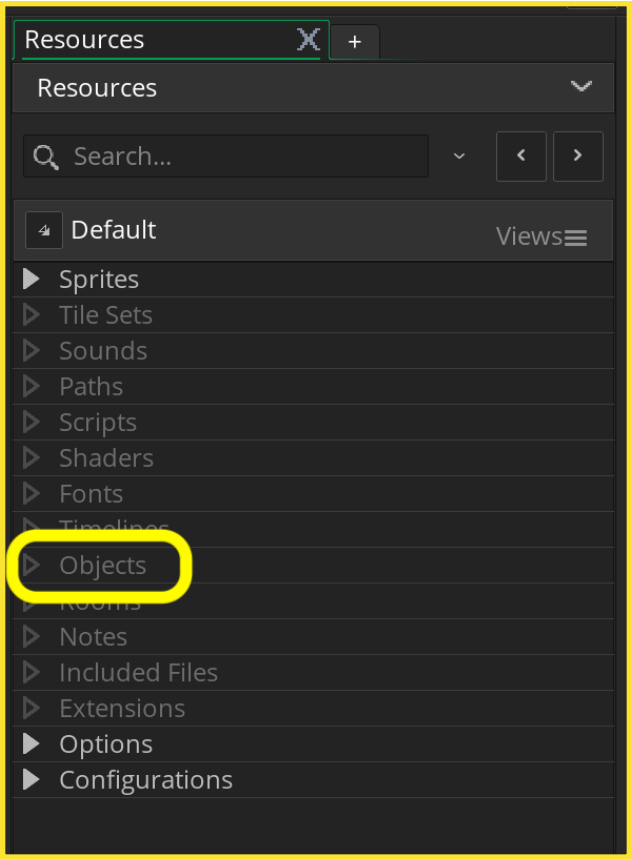


24

The ship is a **game object** that we want the player to control. If it was just a background sprite we could leave it as is.

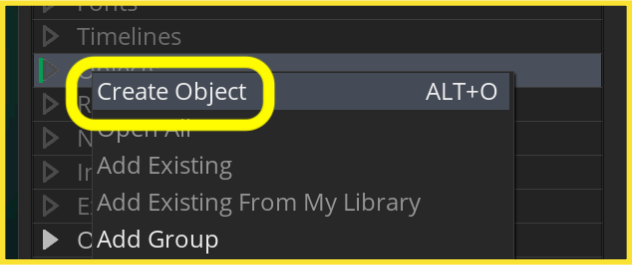
Since we want to add logic and fly the ship, we need to bind it to an ***Object***.

Right-Click the Objects resource…



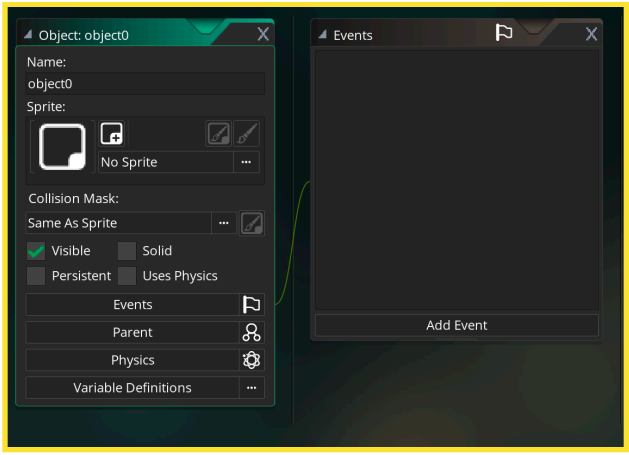
25

Select *Create Object*.



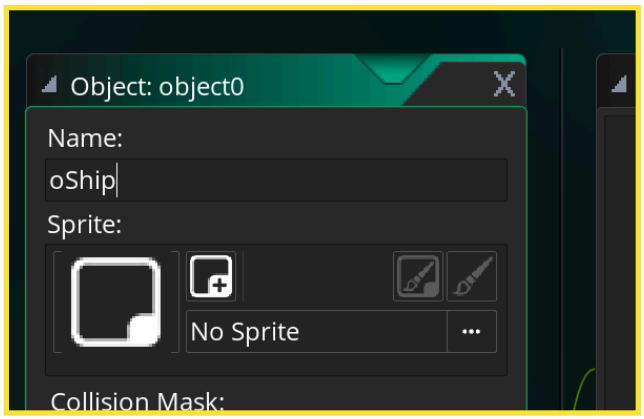
26

This opens up an ***Object*** window in your main ***Workspace***.



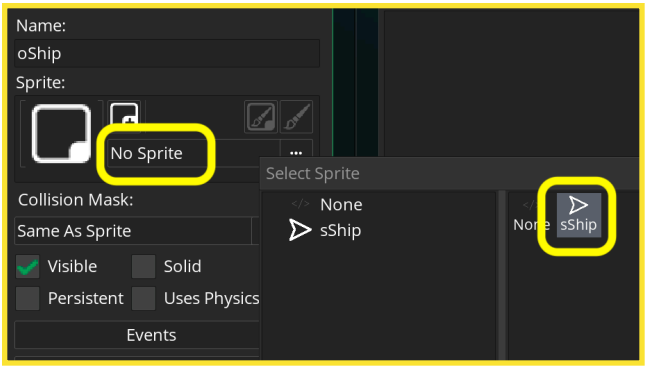
27

Name the object **oShip** with ‘o’ standing for ***Object***.



28

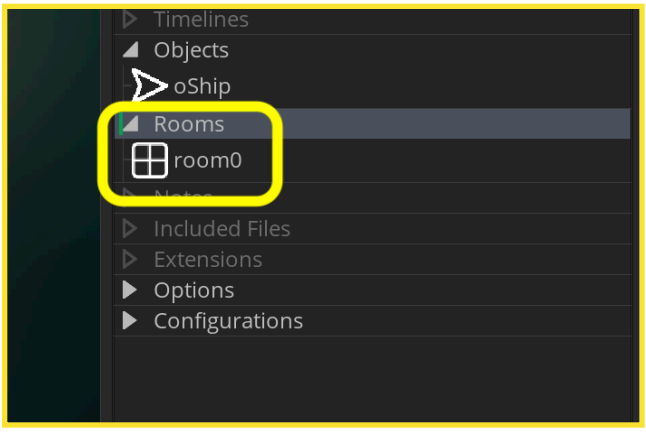
Press the ***No Sprite*** button then bind the **sShip** sprite by **clicking** on it.



29

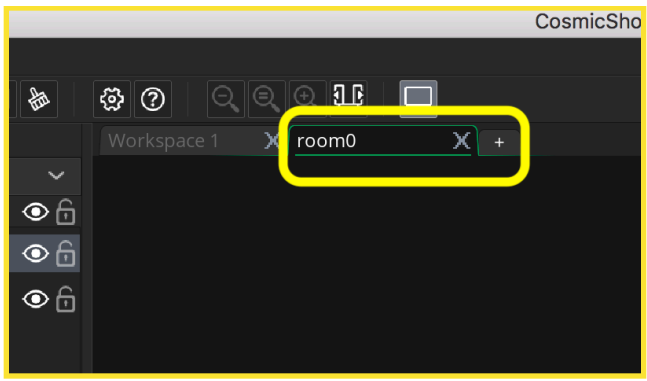
In GameMaker levels are calls ***Rooms***. When we started the game it created a room for us to star with. **Click** the triangle next to ***Rooms*** in the ***Resource*** menu on the right.

You should see a room called **room0**. **Double-Click** on this icon.



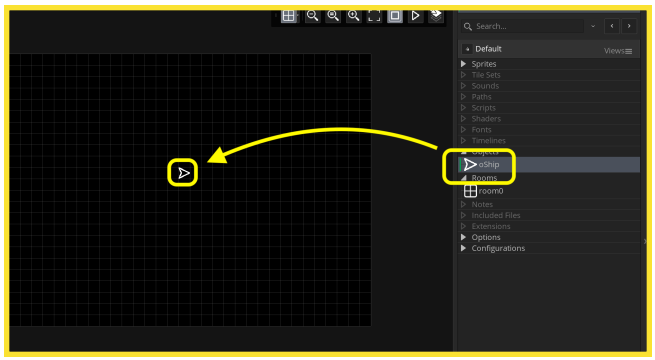
30

This opens up a ***room0*** tab next to your main ***Workspace***.



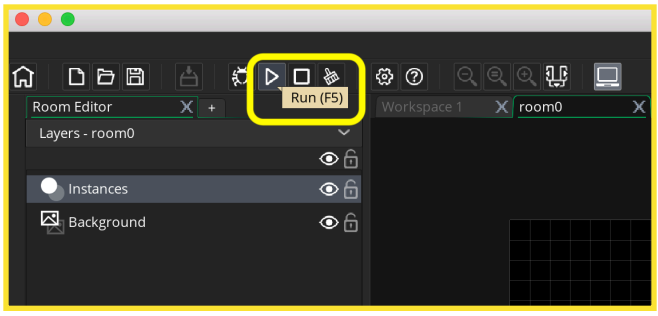
31

Zoom out so you see the whole room. **Left-Click** and **Drag-and-Drop** the ***oShip*** **Object** to the middle of the room.



32

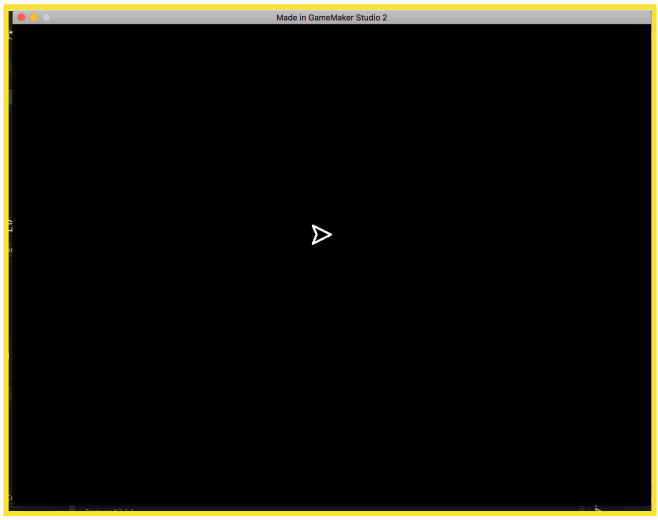
Now we are ready to run the game. In the top menu bar there is a ***run*** button (triangle shape). **Press** the run button to start the game.



33

Our game window should appear. Our game does absolutely nothing. It is a spaceship in the middle of a room with no game logic.

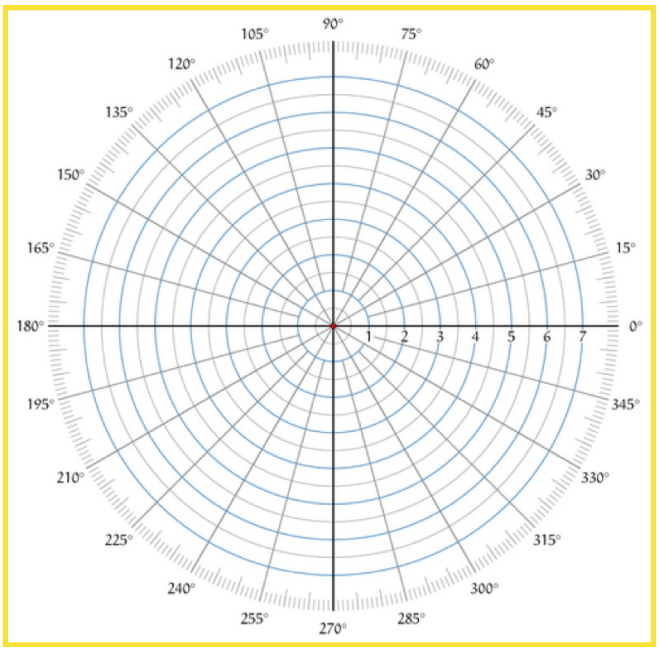
We will need to add some code to alter its behavior.



34

We will be using the polar coordinate system to rotate the ship. GameMake uses degrees for the angle of the ship. 0º is to the right, 90º is pointing up, 180º is to the left and finally 270º points downwards.

GameMaker allows you to go beyond these bounds. It will interpret 361º as 1º so that the ship can keep rotating.



35

We will be using variables to hold numeric information. A variable is a storage location that contains a value. GameMaker only has real numbers (there are no declared types) which is any number either whole or fractional. So here is an example:

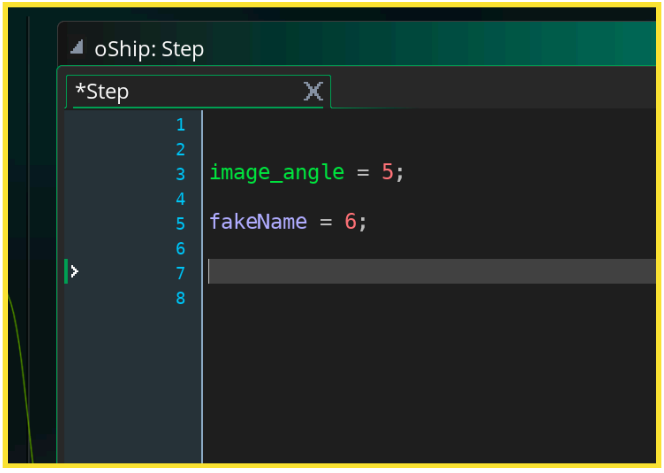
```
width = 5.6;
```

The real number on the right (5.6) is now stored in the variable name width. This means that every time the software sees **width** it gets interpreted as the real number 5.6

36

There are variable names that GameMaker supplied with each Game Object. We can also create our own variable names.

In their editor this is apparent as the GameMaker variables are green in color and our created names are purple.



37

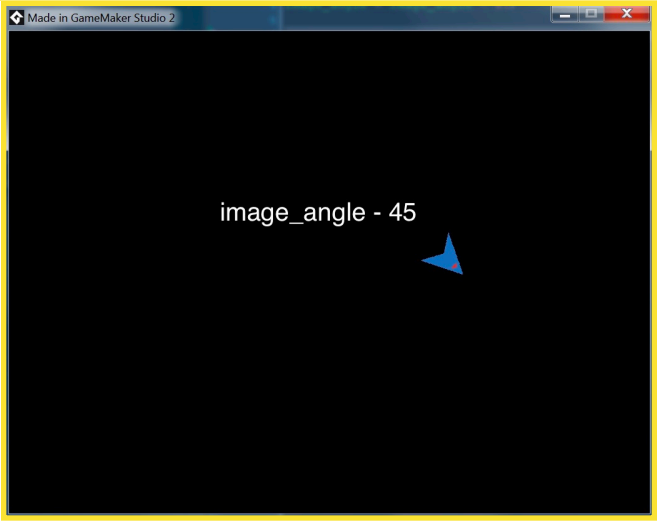
Whenever you create a new Game Object, GameMaker reserves a list of variables that it needs to interpret what you want to do with the Game Object.

The first variable we will look at is **image_angle**. This variable stores the angle of the game object.

We made the spaceship pointing to the right as the default position for all game objects is 0º.

If we make the image angle 45º, then the ship will face at a 45º angle. If we make the image_angle -45º then it will face at 315º (360 - 45).

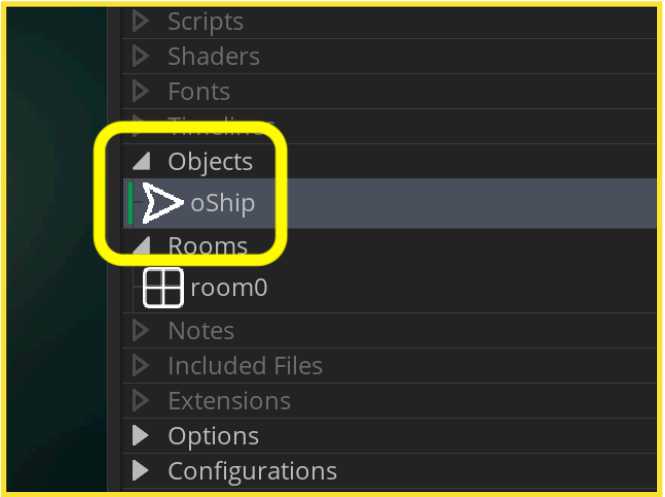
So we are going to add some game logic to rotate the ship by 5º every frame.



38

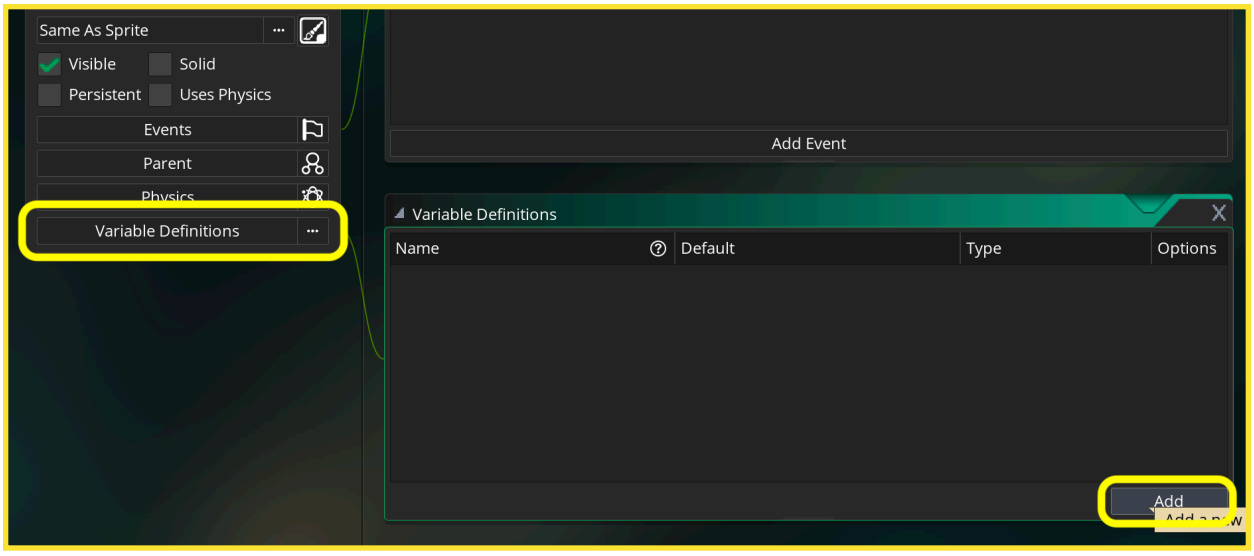
Now we need to create a variable for the ship to hold the variable we will be using for the ship speed rotation.

Double click on on the ***oShip*** in the ***Resource | Object*** menu on the right hand side.



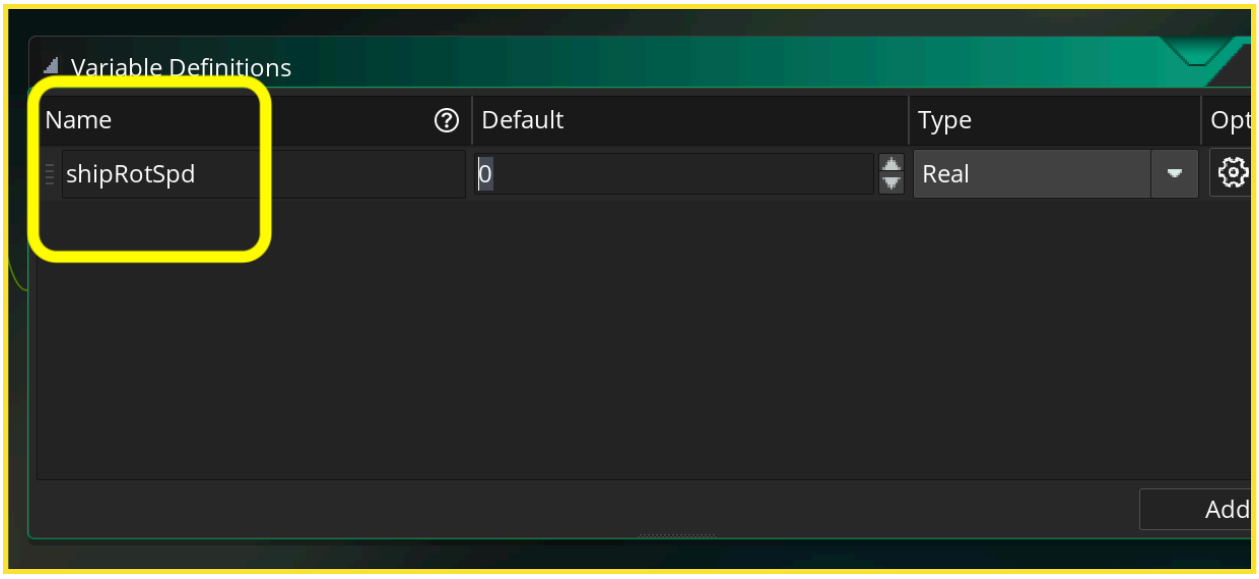
39

Press the **Variable Definitions** button which will fly out a **Variable Definitions** window. **Click** on the **Add** button to create a new variable.



40

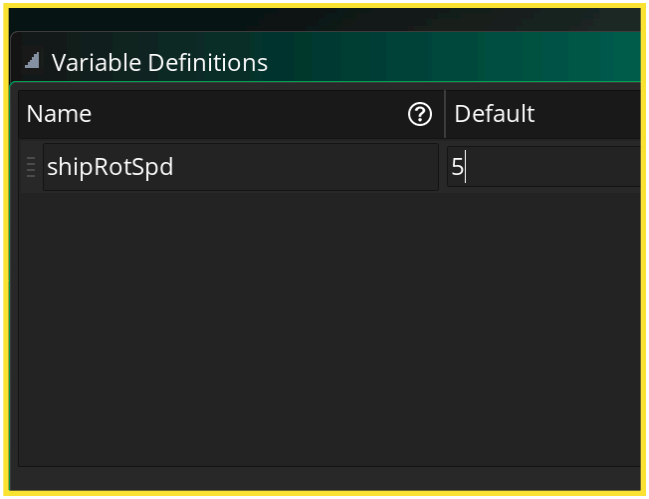
Under ***Name*** type a variable name to describe the ship rotation speed. Variables can't have spaces. I use camel case so that I can quickly tell the difference between the variables I create versus the one's that GameMaker creates which use underscore (foo_foo) between words. Lets call it shipRotSpd to describe the degrees per frame we want to turn the ship..



41

Now under **Default** we will place a value, and we will initialize it to **5**.

We leave ***Type*** as a ***real*** number.

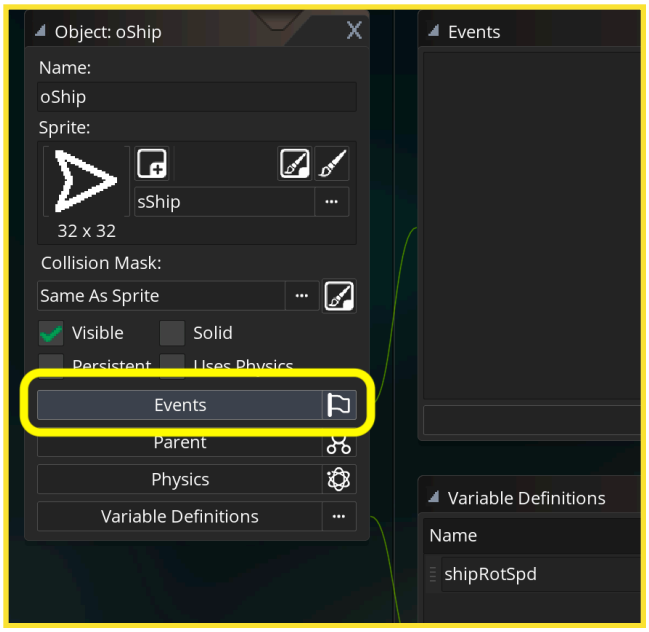


42

Now we need to have this variable affect the ***image_angle*** of the ***oShip*** **Game Object**.

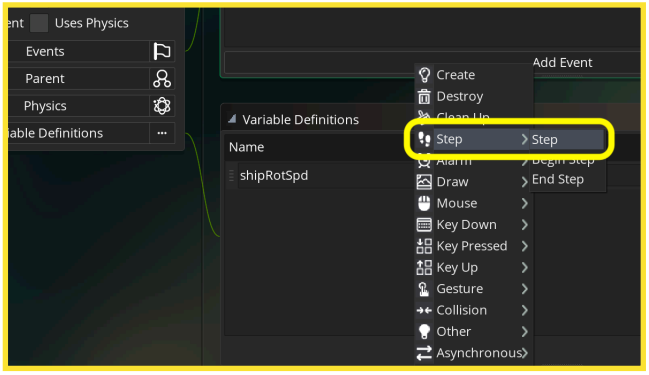
We need to do this every frame. GameMaker has an ***Events*** based system that handles this for us.

We need to go to the ***oShip*** and **press** the ***Events*** button.



43

This brings out another Menu where you will press the ***Step*** menu item, which brings out another menu where you will select the ***Step*** item as well (Events | Step | Step).



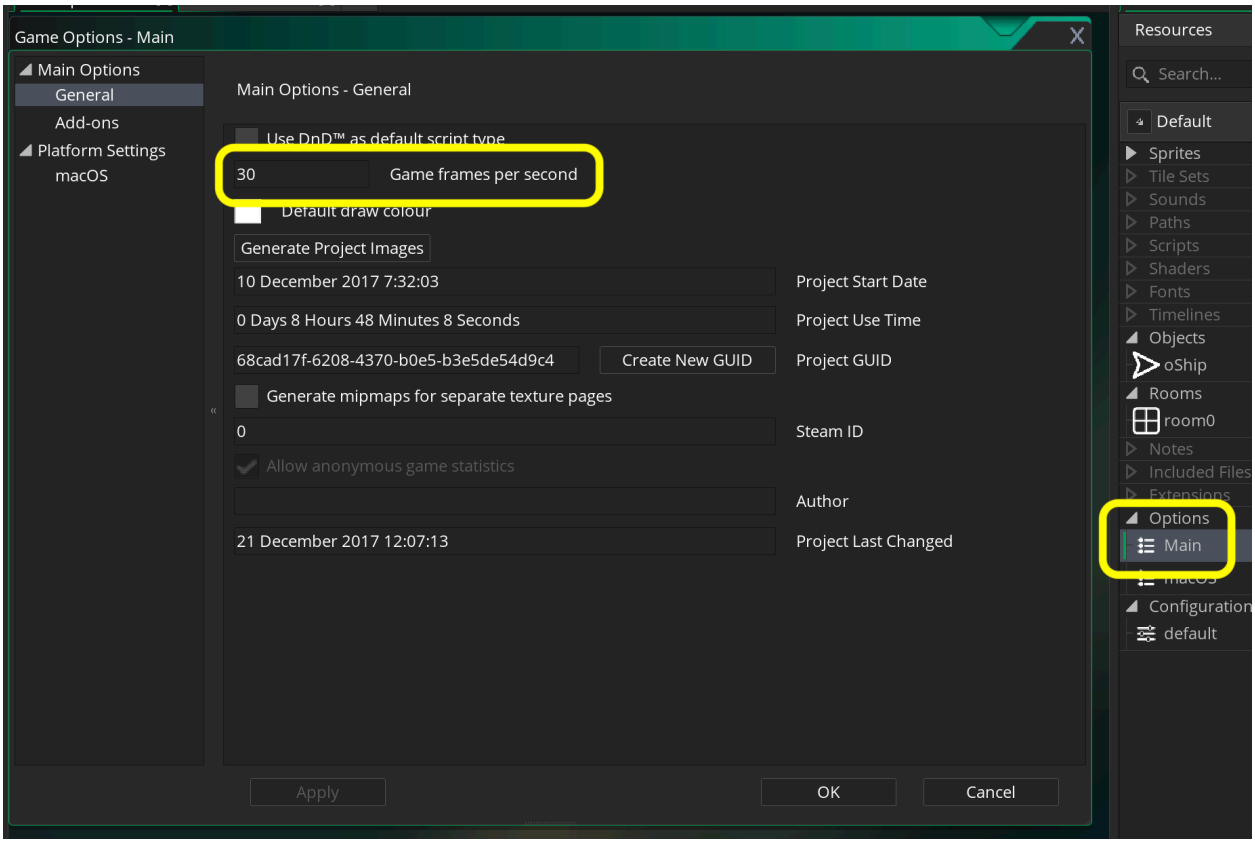
44

The Step Event is the main game loop. It runs every frame when the game is running. The code attached to this event runs every frame.

Typically we instantiate variables in the Variable Definition menu then update them in the Step Event.

45

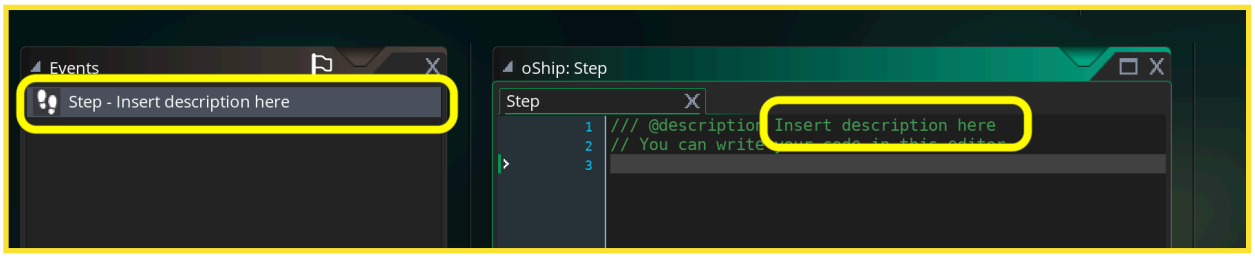
What is a frame rate? This is the number of times the graphics and game logic are updated. It can be found in Options in the Main option menu. You will see that it defaults to 30 Game frames per second.



46

Back in Step Event, it has fly wheeled out another window. This is where we will place our code to rotate the **image_angle** of the ship. The first thing you notice as that the first line has a triple forward slash @description. This is a special kind of comment and does not affect the behavior of the game. Everything after the /// @description will be placed in the side menu to remind you what this script does.

The next line with two forward slashes is also a comment but it can only be seen in the Step event script. Lets change the text after and put Ship Movement after the @description.



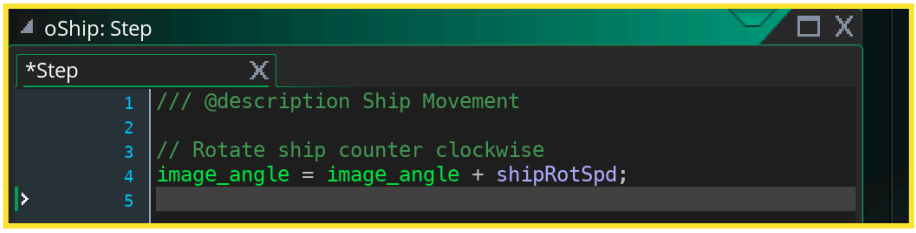
47

Now we add a comment that describes what we will be doing.

In this case we will be adding 5 to the variable image_angle. This way it will go from 0 to 5 to 10 to 15 etc… each game frame in the Step Event.

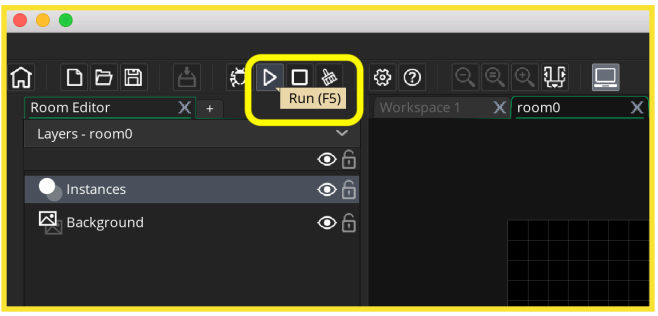
Add to the **oShip** Step Event script:

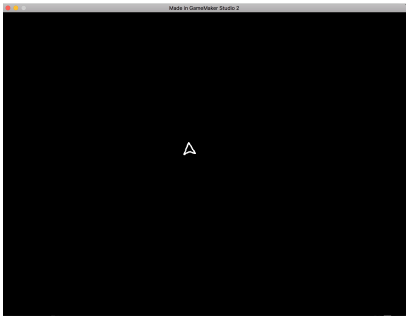
```
image_angle = image_angle + shipRotSpd;
```

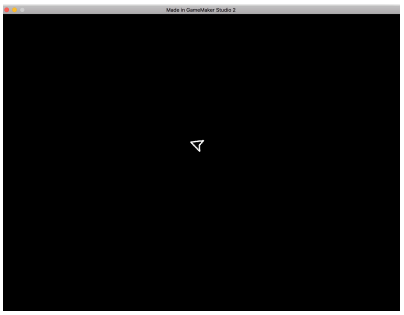


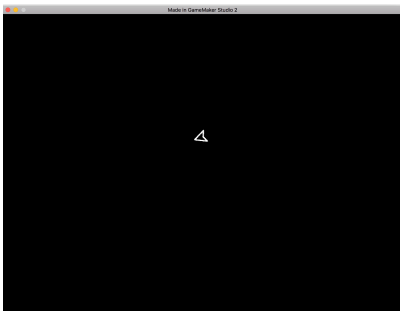
48

Now we are ready to run the game agin. In the top menu bar there is a ***run*** button (triangle shape). **Press** the run button to start the game. The ship should rotate counter clockwise on its own.









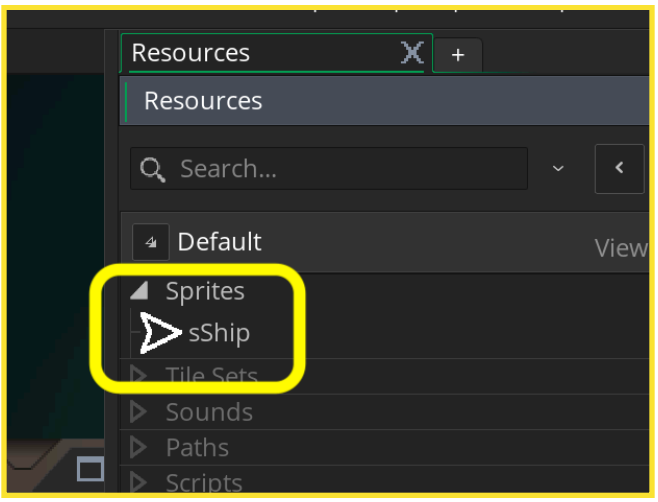
49

You will notice that it rotates on the top left corner of the sprite.

This is because in GameMaker the sprite will rotate around its origin.

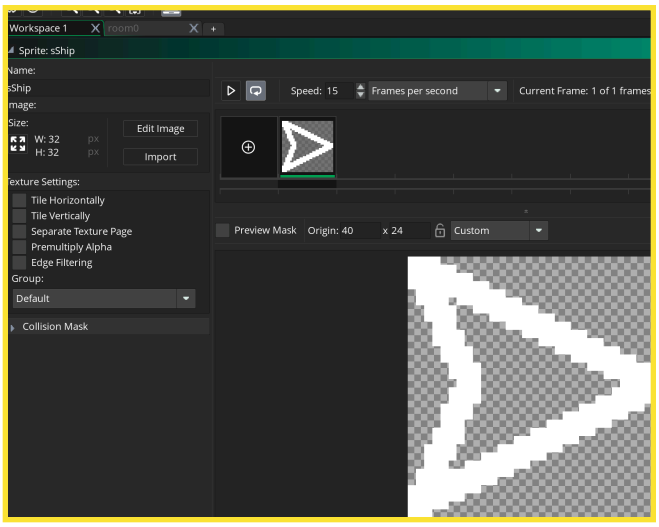
The default location for the origin is on the top left corner of the sprite.

To change this double click on the **sShip** in the Resource | Sprite menu.



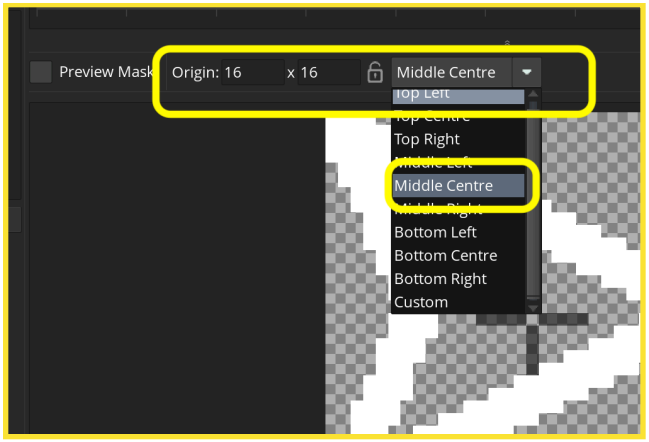
50

We get back to our Sprite Window in our main Workspace.



51

We select the Origin menu option and select **Middle Centre**.

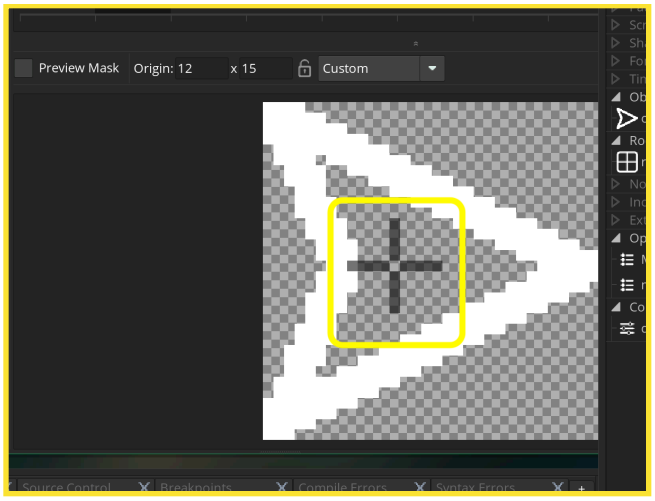


52

The problem is the the middle center of the frame is not the center of gravity for my ship.

You can move the target cursor by left-clicking and moving it to the true center by eye.

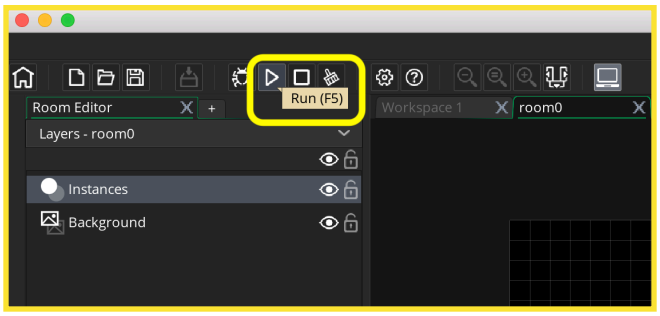
You will see that the option now reads **Custom** which is OK.



53

Run the game again and the ship should now rotate on its true center.

Now we want to have the ship turn only when the player presses the left button.



54

For this we will use a conditional statement and a function.

A conditional if-statement looks at what is within the parenthesis and decides whether the condition is true or false. If the condition is true, the code within the curly braces { … } is run…

```
if (true)
{
        ...run something if if() is true.
}
```

55

A function performs a task in the program. A function can be found by looking for a name followed by ( ) parenthesis (round brackets).

```
function (parameters);
```

56

A function can be passed data that are called parameters. It can also be passed multiple parameters separated by commas.

```
keyboard_check(key);
//keyboard_check is a built in function.
```

57

A function can be returned data so you can assign the output of the function to a variable.

```
function (parameter1, parameter2, parameter3);
```

58

GameMaker functions are written in lower case with _ between words.

```
//Print text between " " to screen
if (keyboard_check(vk_down))
{
        Print ("You are pressing the down key");
}
```

59

Now lets look at the function keyboard_check(key) and see what it does.

We pass it one parameter, a key, and the function returns true or false on whether the button is being pressed.

For example vk_down is the down arrow key on the keyboard. If we pass it this vk_down to the function, it will return true if the key is pressed and false if the key is not pressed.

Or there could an average() function. We would pass it three real numbers and it would return the average.

```
average = AverageFunction (1, 2, 3);
//AverageFunction returns the value 2 which is the average of the above 3 numbers
```

60

Lets put this together and type in **oShip** Step Event and alter the existing script to read::

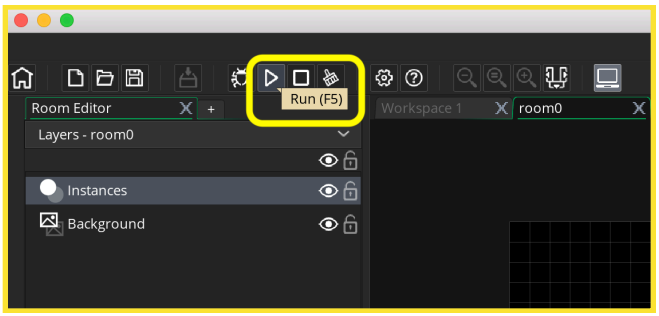
```
if (keyboard_check(vk_left))
{
    image_angle = image_angle + shipRotSpd;
}
```

Step

```
1  /// @description Ship Movement
2
3  // Rotate ship counter clockwise
4  if (keyboard_check(vk_left) )
5  {
6      image_angle = image_angle + shipRotSpd;
7  }
```

61

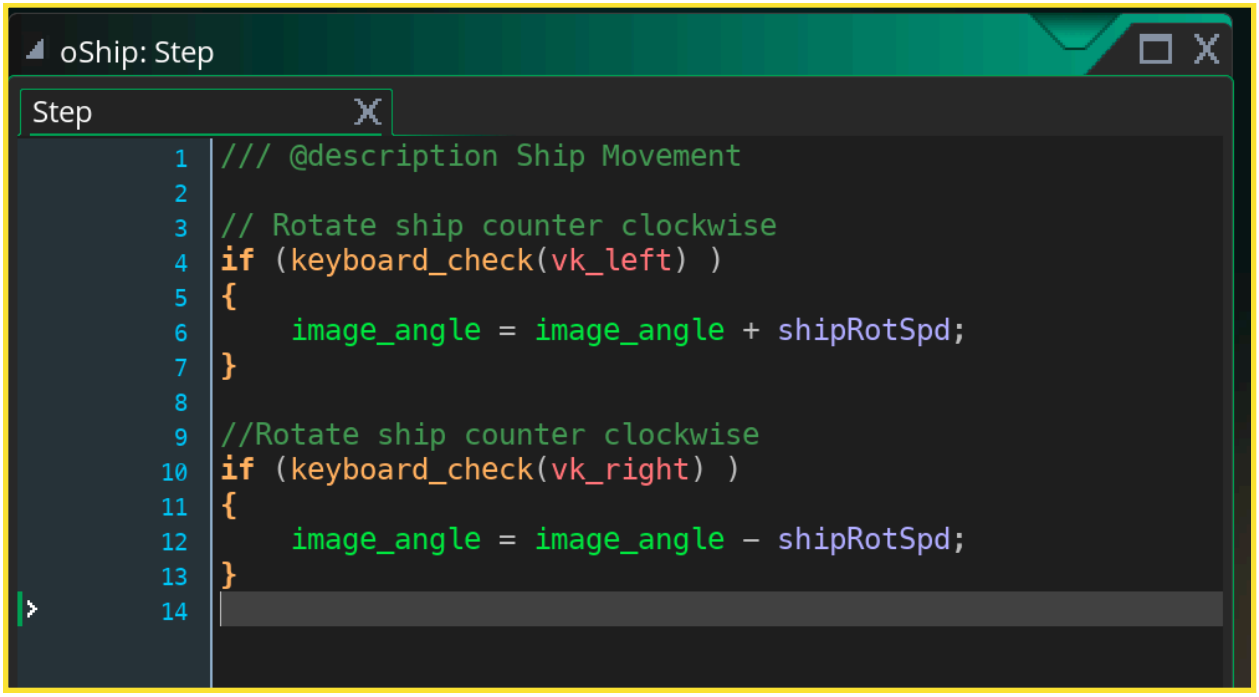
Run the game and the ship should now only move counter-clockwise when the left button is pressed.



62

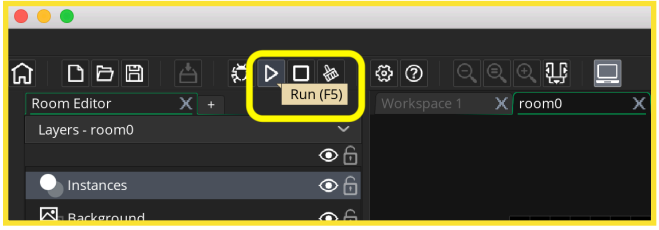
Lets add the ability to turn clockwise. We will need to subtract 5 degrees per frame to turn in the other direction and look for the right button press. Add this text beneath the previous code:

```
//Rotate ship counter clockwise
if (keyboard_check(vk_right) )
{
     image_angle = image_angle - shipRotSpd;
}
```



63

Run the game and the ship now moves both clockwise and counter-clockwise when the left and right button are are pressed.

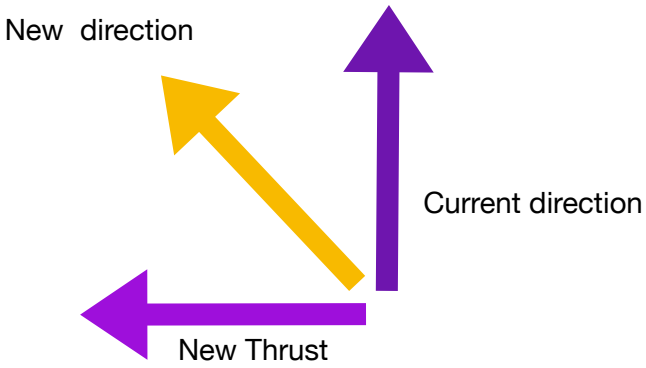


64

Now we want the ship to move forward when we press the up arrow.

We need it to behave like a spaceship in a near frictionless environment. So when you press thrust we want to add it to the existing motion vector.

If the ship is going North and the player thrust due West. The plane will now move at 135°.



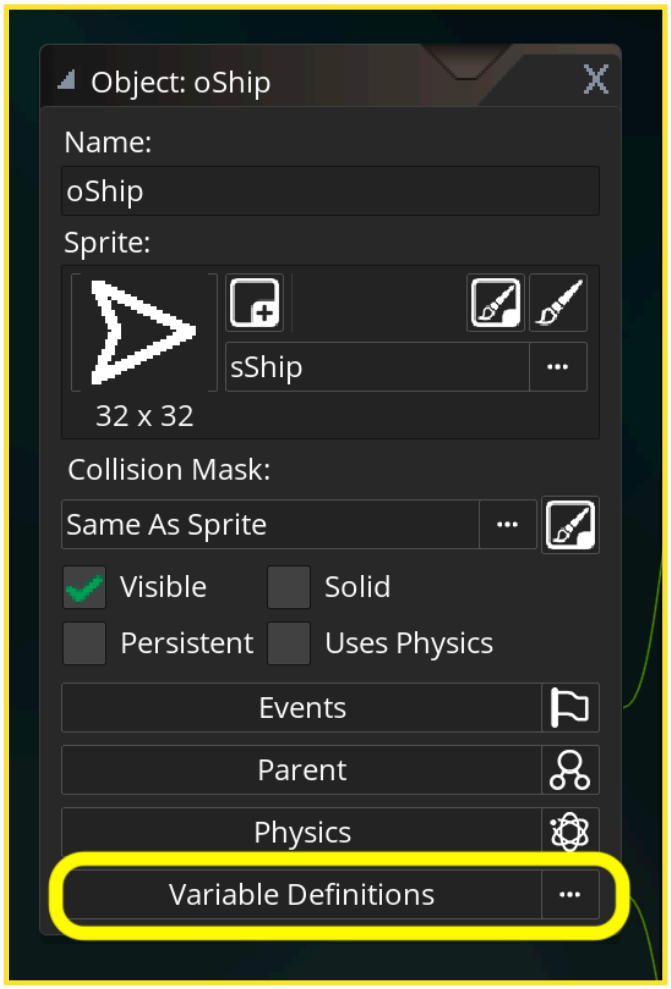
65

There is a function that does this for us. It is supplied by GameMaker and is called **motion_add(dir, speed).** It doesn't return a value it just adds the new direction and speed to the Game Object.

We want the direction to be the forwards facing angle of the ship or it's image_angle.

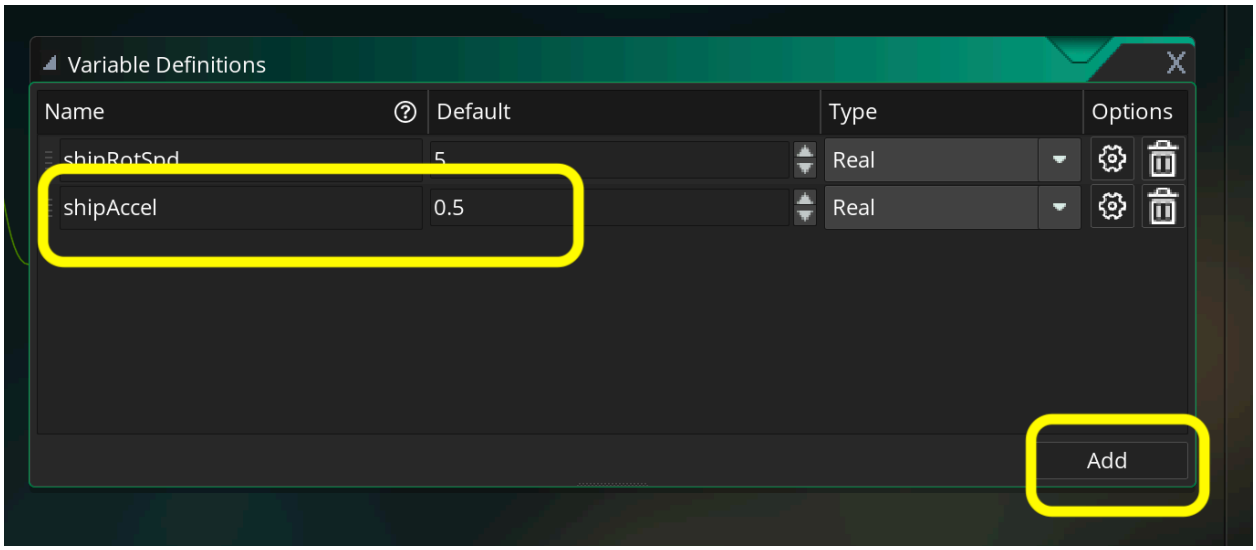
The **speed** variable is the amount it moves per frame in pixels per second.

We will be accelerating at 0.5 pixels per second. Lets create a new variable by pressing **Variable Definitions** on **oShip**.



66

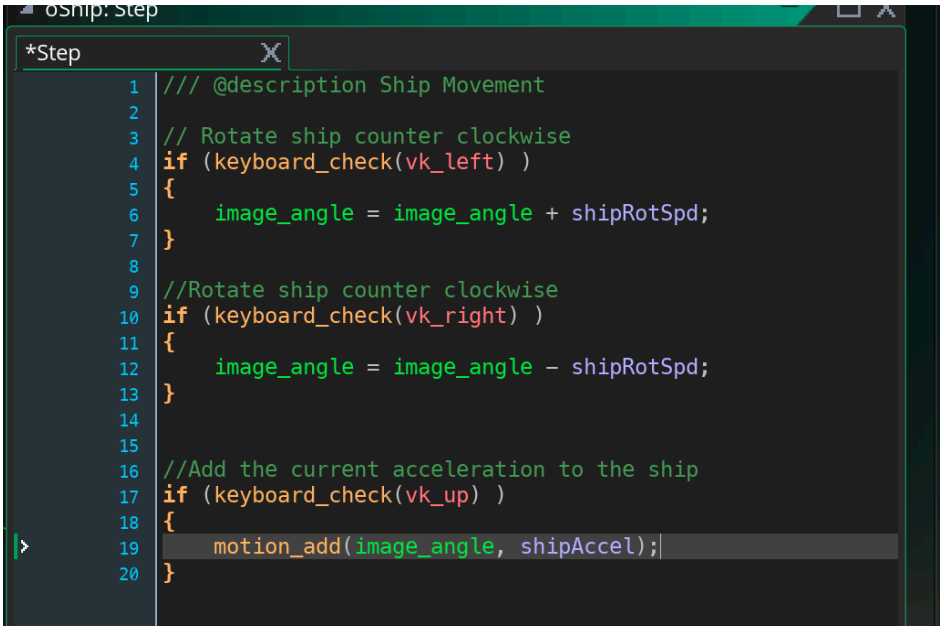
Press the ***Add*** button and then create a new variable called **shipAccel** and set the ***Default*** to **0.5**



67

Now type at the end of the **oShip** Step Event script:

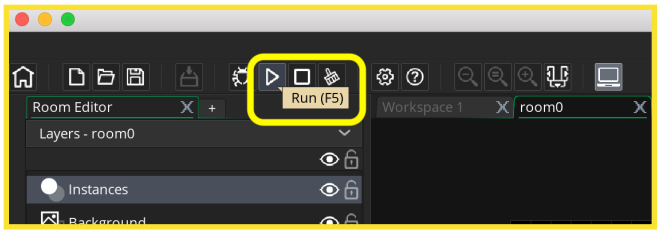
```
//Add the current acceleration to the ship
if (keyboard_check(vk_up) )
{
   motion_add(image_angle, shipAccel);
}
```



68

Run the game and try pressing the up arrow. You should be flying around.

There is a major problem though. When you go off screen you disappear and it is hard to get back on screen.



69

We will want to have the ship wrap horizontally and vertically. So if the ship leaves the top of the screen moving up it will come out of the bottom of the screen.

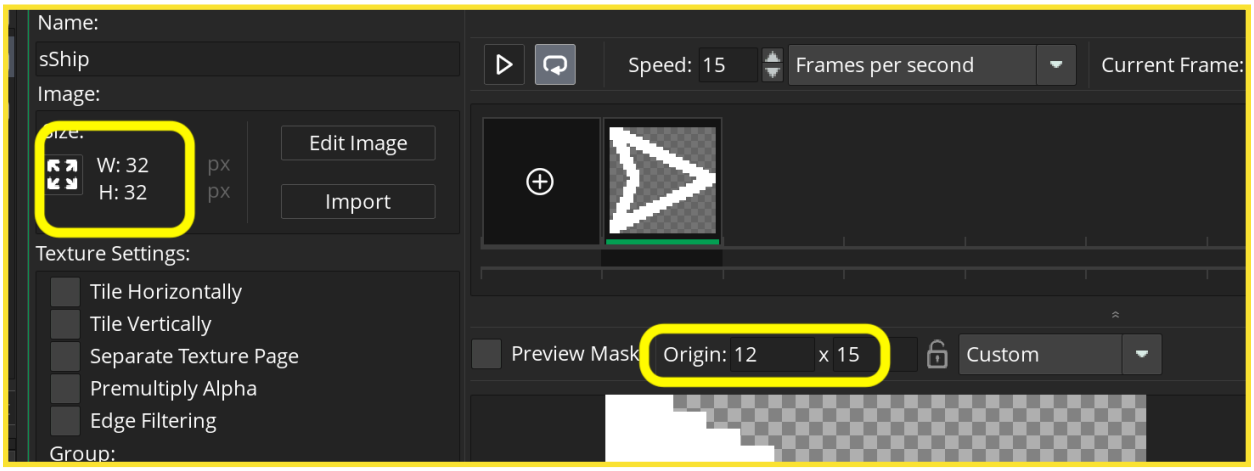
GameMaker gives us a function for this. It is called **move_wrap(hor, vert, margin).**

This function doesn't return any value. The first two parameters are asking whether we want to wrap horizontally and vertically. This is a new variable type it is called a **boolean**. It is two values either true or false.

The final parameter margin How far outside the room, in pixels, the object must be to initiate wrapping.

Look at the screenshot below. If our origins was perfectly middle - center then the offset would be 16 pixels as the entire ship would be off screen.

The origin on the X is at 12. Which means that there are 20pixels to the right of it. So it will only be completely off screen going left after 20 pixels. This as a margin should work for all directions. If you wanted to be absolutely sure you would set it to 32 pixels, but there would be no ship at all for a few brief moments.



70

Now we want the ship to move forward when we press the up arrow.
At the end of the **oShip** Object Step event script add:

```
//Make ship wrap back to other side when leaving screen
move_wrap(true, true, 20);
```

```
12      image_angle = image_angle - shipRotSpd;
13  }
14
15
16  //Add the current acceleration to the ship
17  if (keyboard_check(vk_up) )
18  {
19      motion_add(image_angle, shipAccel);
20  }
21
22  //Make ship wrap back to other side when leaving screen
23  move_wrap(true, true, 20);
24
```

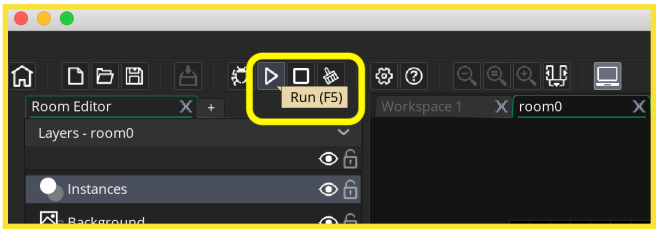
71

Run the game and try flying off screen. Notice that you can no longer leave the game world space.

You ship just wraps endlessly.

Try flying as fast as you can. Woops, we have a problem. You can infinitely accelerate.

We need to clamp your maximum speed. GameMaker has a function for that and it is called **clamp()**.



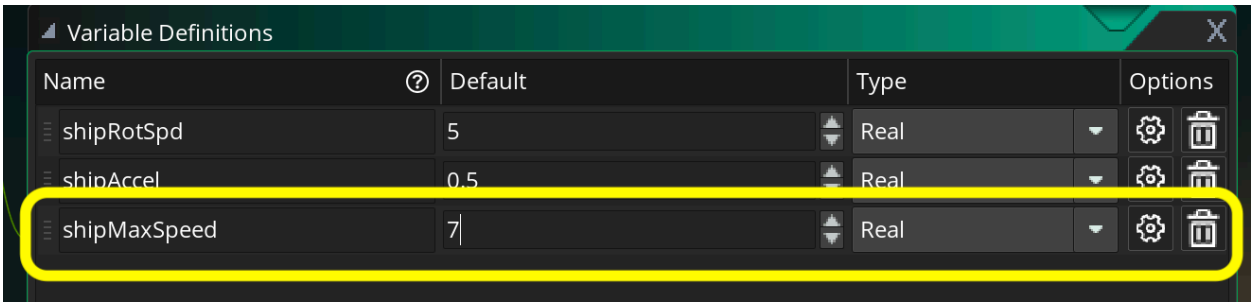
72

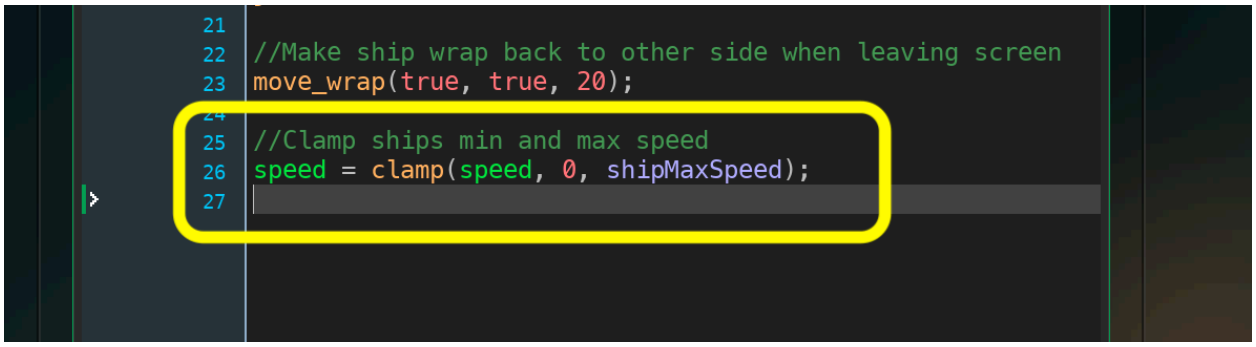
clamp(val, min, max) is the function we will use. This returns a value from variable val with a number that is within the range of min and max (inclusive).

What variable will we clamp? The ships motion vector consists of two parts. The variable **direction** is the angle it is moving at, and **speed** is the pixels per second it is moving at in the given direction. So we need to clamp the **speed** variable.

On **oShip** create a new variable in ***Variable Definitions*** and call it **shipMaxSpeed**. Set its default value to 7 and leave it as a Real Type. Then add to the bottom of the Step Script:

```
//Clamp ships min and max speed
speed = clamp(speed, 0, shipMaxSpeed);
```



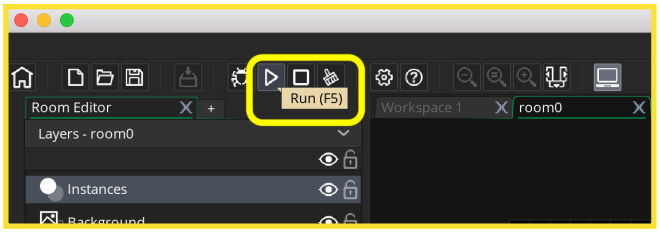


73

Run the game and try flying faster and faster.

You should notice that your speed is now clamped. Now let go, your ship just keeps flying forever. Ok, that's no good.

Lets add friction.



74

Now how will we implement friction. We are already clamping the speed value, so with friction, we can just subtract a small value (smaller than our acceleration) each frame.

Add **BEFORE** the clamp() this line:

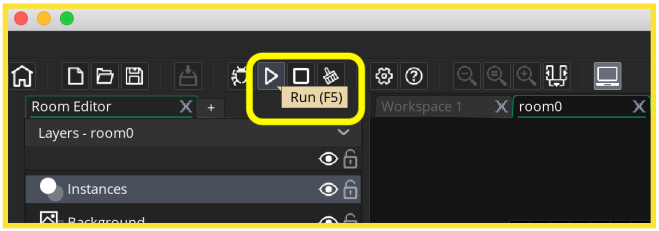
```
//Friction slows ship down
speed = speed - .1;
```

```
20  }
21
22  //Make ship wrap back to other side when leaving screen
23  move_wrap(true, true, 20);
24
25  //Friction slows ship down
26  speed = speed - .1;
27
28  //Clamp ships min and max speed
29  speed = clamp(speed, 0, shipMaxSpeed);
30
```

Before Clamp

75

Run the game and now we have a spaceship that moves nicely through space, wraps and has friction applied to it.



That's it for this exercise. You can see what it should look like at https://www.youtube.com/edit?o=U&video_id=K5CpdXlioHY

I normally add shooting bullets and targets to shoot at. If you are interested in seeing this you can go to:

https://github.com/maubanel/cosmic-shooter-extended