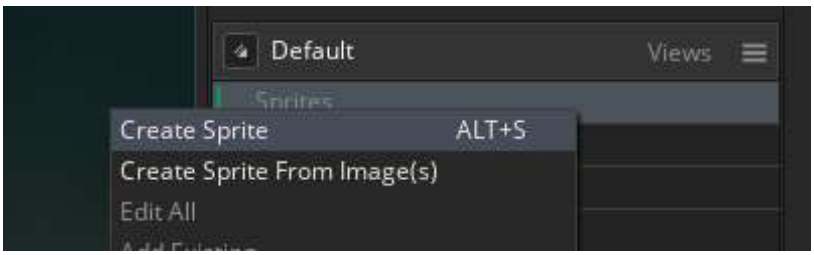
3.1 Spaceship game

For the game we'll need some resources to start with. In this example we've resorted to open license images from websites like http://opengameart.org.

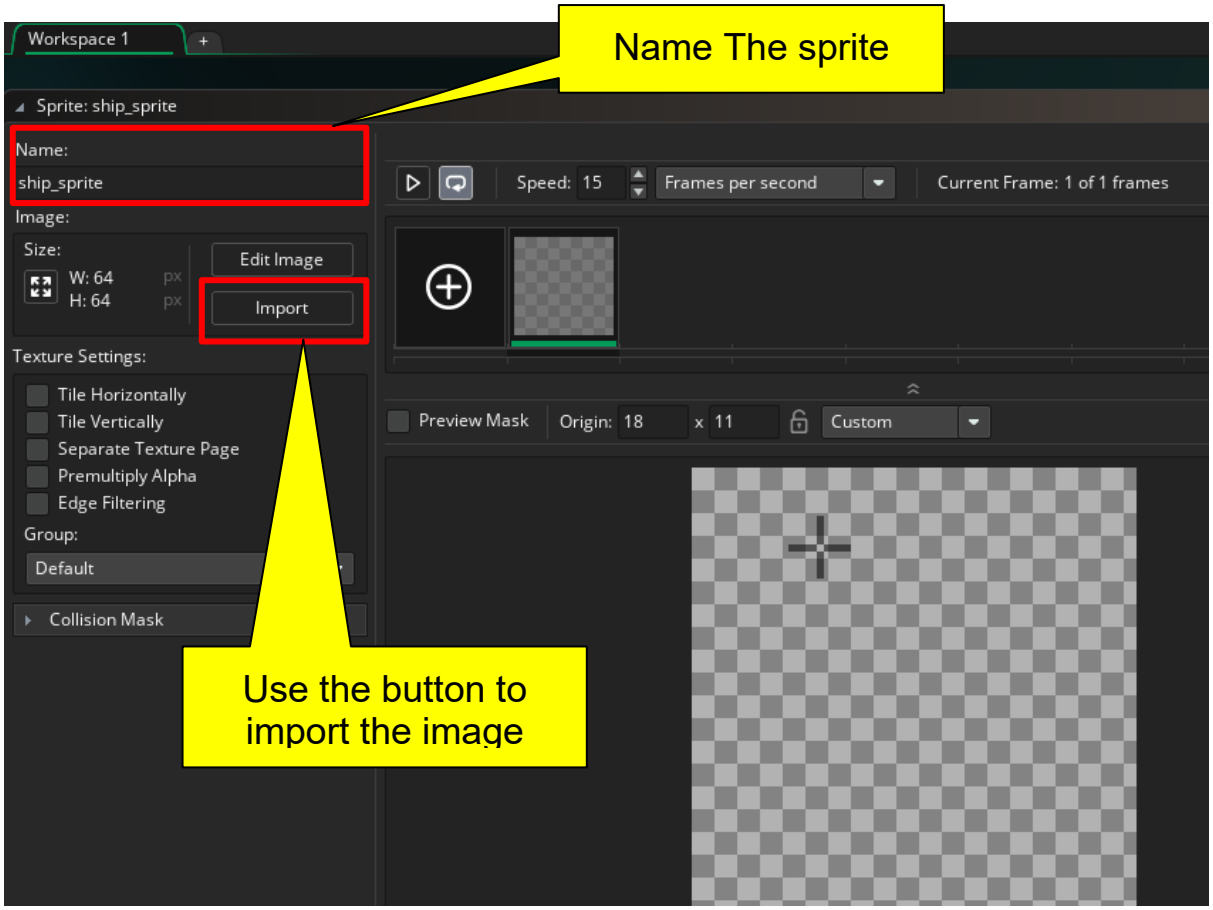
For the spaceship image we'll be using the image found in https://opengameart.org/content/spaceship-9 . which has a "public domain" license. The very first thing is to create a new sprite. Right -click with the mouse over the "sprites" item in the "resources" list at the right side of the IDE. Chose the "Create Sprite" menu entry



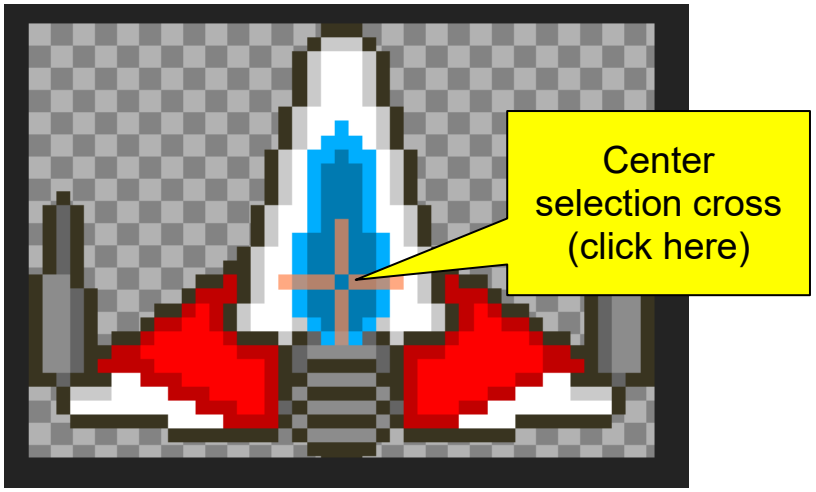
A new tab called workspace1 appears for the created sprite with several controls and areas. We'll be using this to create our sprite. We need to do the following:

1. Name the sprite. For example, set the name to ship_sprite.
2. Import a image (just one in this case)
3. Set the center of the sprite to coincide with the intended object's center.

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



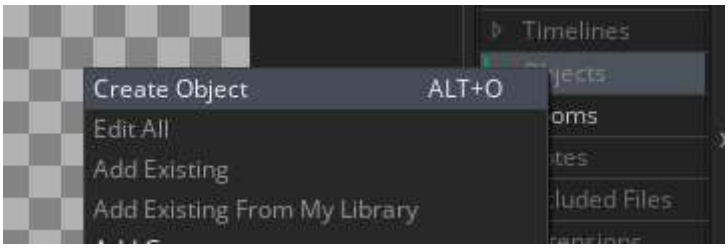
To set the center of the ship, just left-click in the point where you consider the center to be. It is no problem if you miss by some pixels.



Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

If you don't see the complete image in your workspace, you can use the middle button of your mouse to move things around and the mouse wheel combined with the ctrl key to zoom in or out.

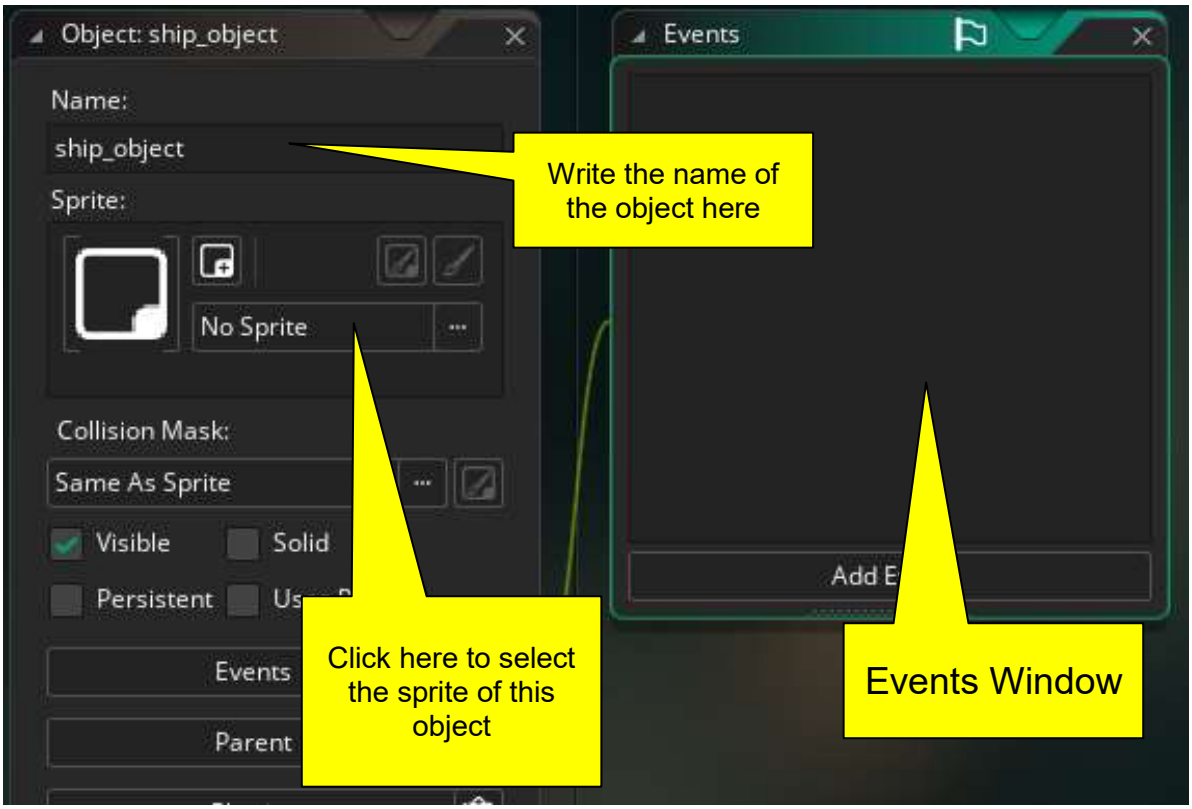
Now that the sprite is created we move on to the next usual step which is to create an object and make it visible, even if we don't give it any behaviour. To do so, right -click with the mouse over the "objects" item in the "resources" list at the right side of the IDE.



You'll see two new forms or dialogs placed in the workspace1 and linked by a line. The main form is shown to configure the main object properties, while the linked one is used to control the behaviour of the objects by events. Sprites and objects can share the same workspace and forms are arranged in it. You can freely move around with the center button of the mouse.

We need to name the object to something meaningful like "ship_object" and to select the sprite we already created for this object.

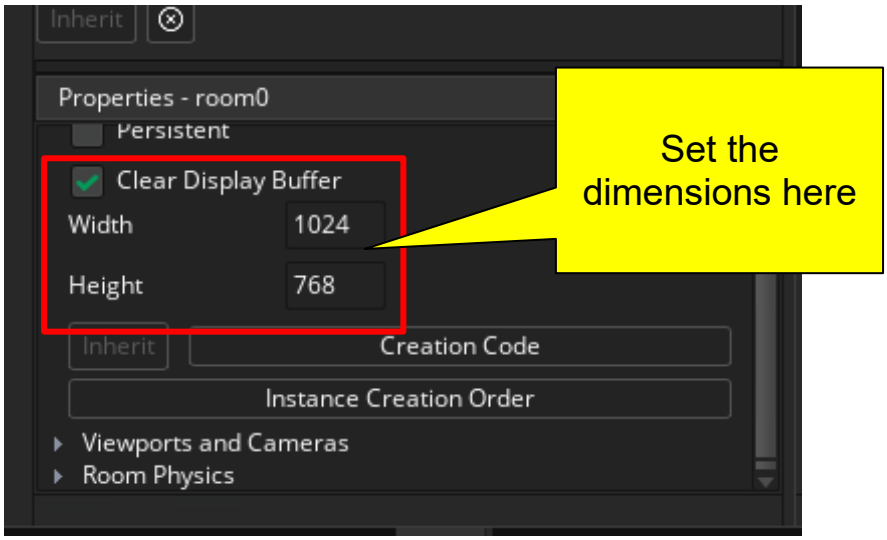
Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



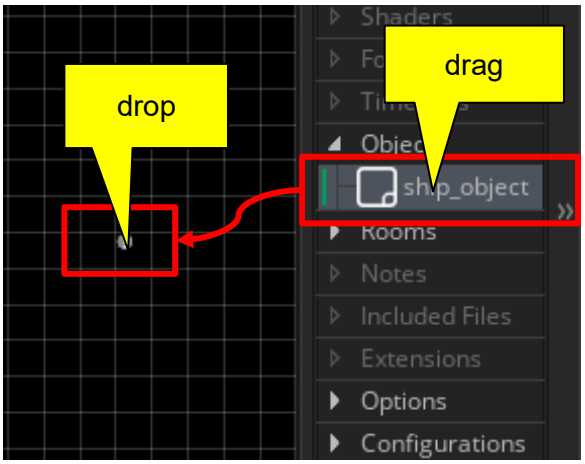
Before we start adding some behaviour to this object (like moving with the keys) we are going to go first to test the game (though we can not name it "game" at this early stage). As told above, the game happens in areas called "rooms" (other names with similar meaning are "levels", "stages", "screens", etc.). There is always at least one room in a game, therefore Game Maker creates one initially and names it room0. If you take a look at the Resource Menu in the right area of the IDE, you can notice it by clicking in "Rooms" and watching the "room0" already created.

Clic in the room0 so it will appear right in the center of the IDE as a new tab (the workspace1 tab is still quickly accessible) . The room editor and properties appear inside a panel placed in the left side of the IDE. We first, are going to to set the dimensions of the room to a popular screen size

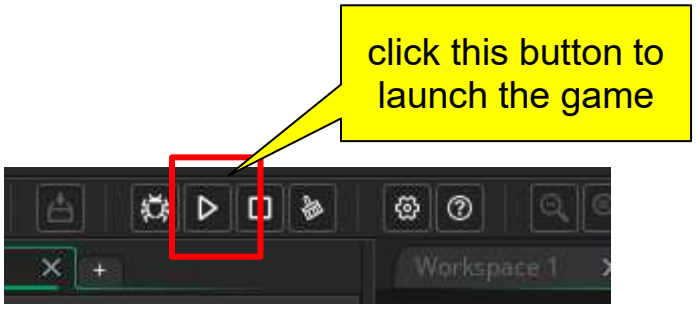
Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



To place the ship, simply drag and drop from the object to the room:



It's time to launch the game which will only show the ship in the middle of an emtpy dark window. Use the "build" menu and then the "run" entry, press F5 or click the "play" button in the button toolbar



It will take some time to "compile" the game, create the executable and launch it. But if everything goes ok, you'll see a new windows in dark black and the ship where you dropped it. From now on, you may launch the game at any time checking your progress at your will. If anything goes wrong, Game Maker will tell you where the error is. Carefully , read the error description and go to the offending line or element to correct it.

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Let's move on to add the ability to move the ship with the keyboard. This goal, like most of the behaviour of objects is achieved by using events and actions. But, what event is the one needed for this case? and What action has to be performed consequently? The answer to these questions is the key to make games. Every requirement has to be analyzed and split into simple actions. For example, moving the space ship will consist in the following 4 event-actions pairs:

- If the "down" key is pressed, the ship will take the down direction in the room or screen.
- If the "up" key is pressed, the ship will go up
- If the "right" key is pressed, the ship will go to the right
- If the "left" key is pressed, the ship will go the left.

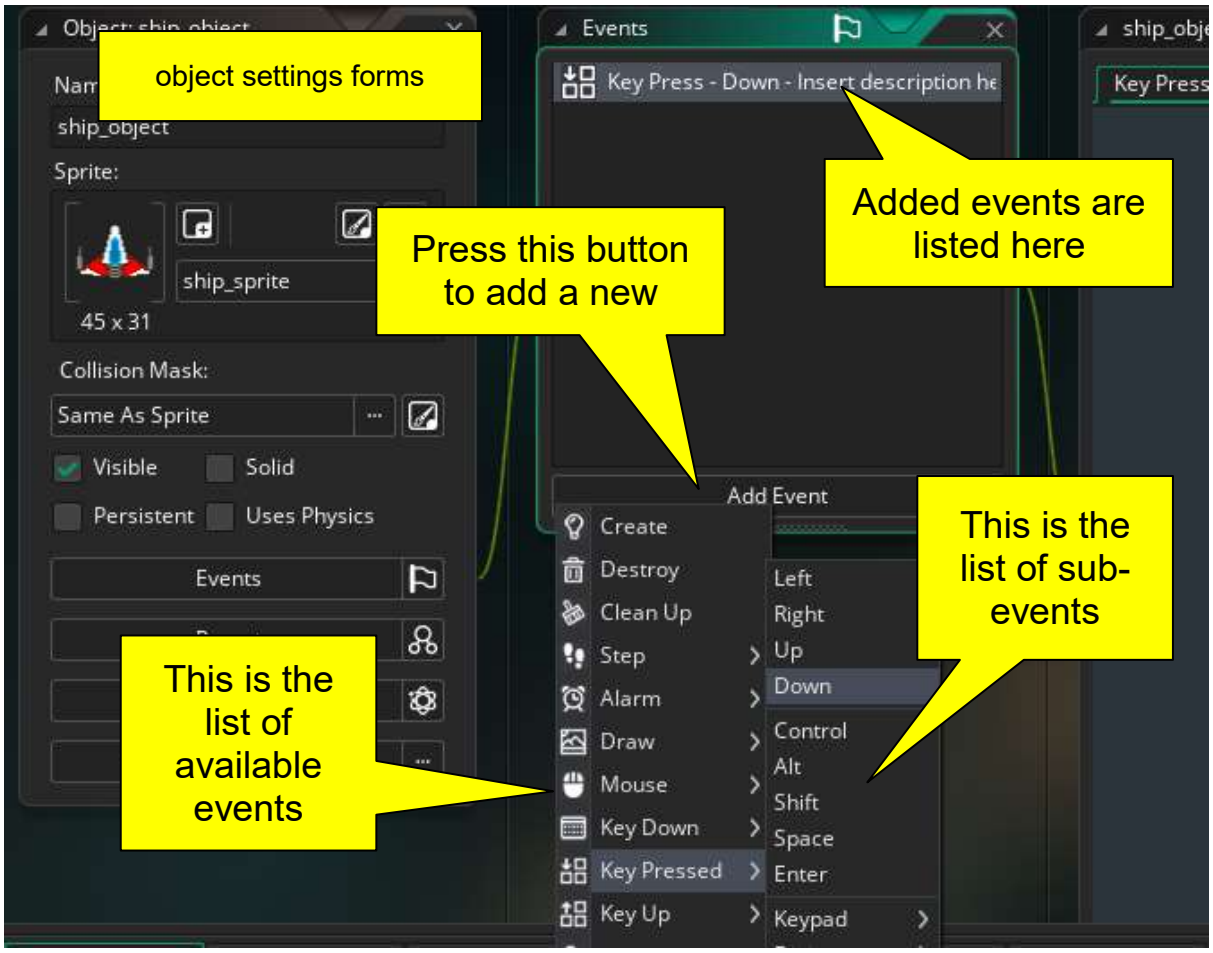
We need to work these one by one. There are several ways to do it, some of them leading to slightly different ways of piloting or maneuvering the ship during actual game. For learning purposes, it doesn't really matter if the ship handling is not the expected.

To move the ship down, we first have to work on the ship object. Double click it to center the forms concerning the object. Then, please, put your attention in the "events" form (the smaller one). It's a list of events that this object is able to react to. It's empty at the moment, we click in the "Add Event" button below, to add a new Event. We chose the "Key Pressed" item and the "Down" element from the pop up menu. You've just selected an event that will be triggered when the player presses the arrow down key.

When an event is added, Game Maker understands that you wish to set an action for it. Actions are specified by writing some small program (or "code"). That's what will be executed if the event occurs. Game maker pops up another form in the workspace whose content is a text editor where it expects that you write the code of the action. It also shows a line that links the event to the editor, making it clear that you are ready to write some small program to that precise event.

The following image summarizes most of the process so far

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



You can move the forms in the workspace by clicking with the middle button of the mouse and moving it. Center the workspace visible area in the code editor for the new event. You could find some lines starting with a double slash (//) in the editor. These are comments lines that have no effect on the program and help the developer review and understand what is written. Write the following two lines with every character including ending semicolons.

```
direction=270;
speed=1;
```

This is enough to make a simple movement when the key is pressed. The effect of those two lines are explained as following: Every object has some properties, like position, speed, size, visibility, etc. When specifying actions, we can set new values for any of these properties and the game engine will apply the change to the object. For example, if we set a value to the property "x", the horizontal position of the object will change to the new value, i.e. It will move abruptly to that position.

In the example above, though, we're using a different way of moving objects. The first line is changing the direction of the object. By specifying a value of 270 degrees we are heading it to the down direction. And by setting the speed to 1, we are effectively setting a motion or movement to the object. The combination of both is the action "move down".

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

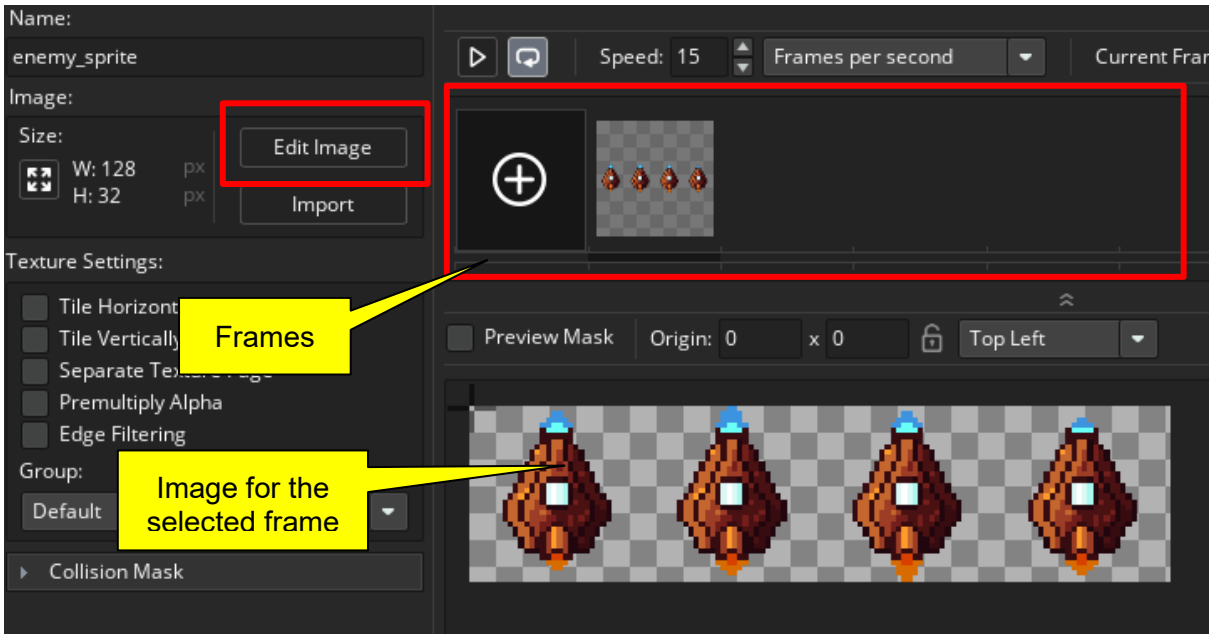
It's time to test the game as explained above. At this very point, you may not like that the ship doesn't stop moving or only moves down, but that's just because we've not set all the events. Moving it up, left and right is only a matter of repeating what has been done with this event, selecting the related keys and giving the corresponding direction (0 right, 90 up and 180 left).

Now, We are going to create enemy ships. For the most part it's just the same process: find an image, use it to create a sprite, create an object, give it some behaviour and place it in the room. For the image you can use one of the images located in opengameart.org. We'll be using one of the images found here https://opengameart.org/sites/default/files/Ships_1.zip .

Inside this package you can find several image files. Every image is actually a row of images of a ship with small differences. You can search and find other similar images in many websites.

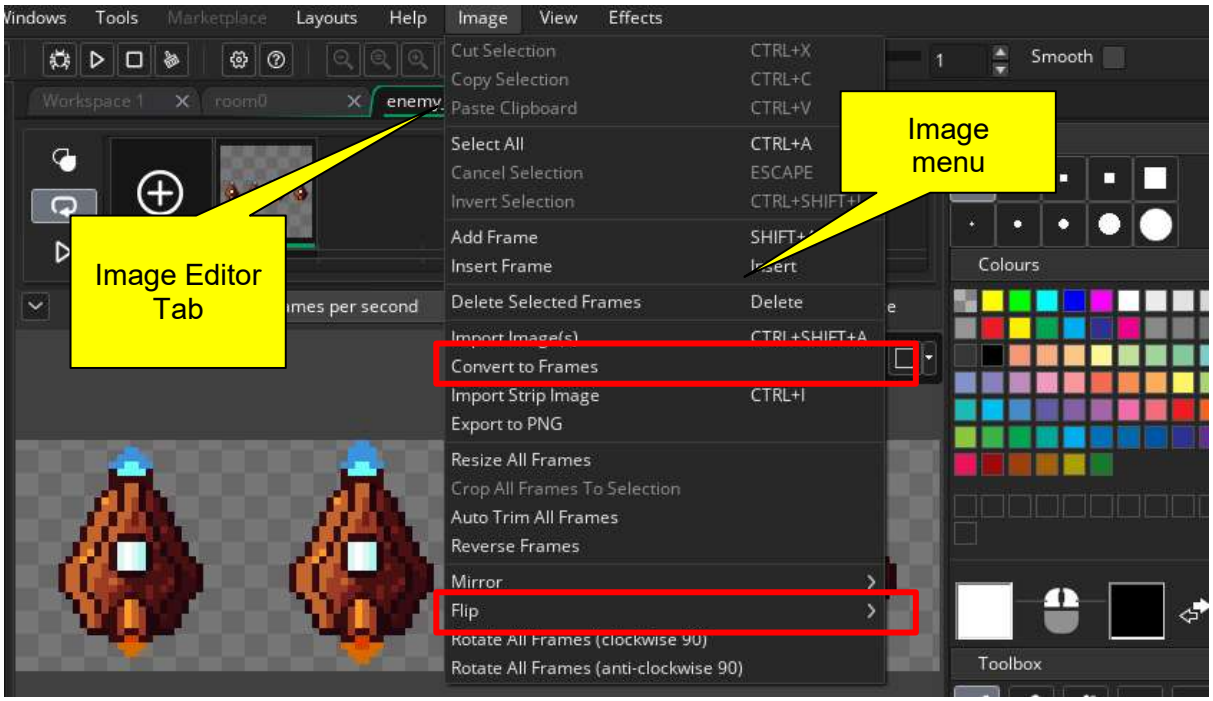
This is usually called "a strip" of images. They are provided this way to simplify the process to create an animated image. By splitting the image into four different images, and sequentially showing each one, the game can have animated elements. All the four images will be handled in one sprite which will automatically do the animation.

Start by repeating the same process done in the ship sprite. When you've got your sprite, you'll see it only has one frame and its image is the whole strip. We need to edit our image and split it into four different frames. This is done in the Image Editor

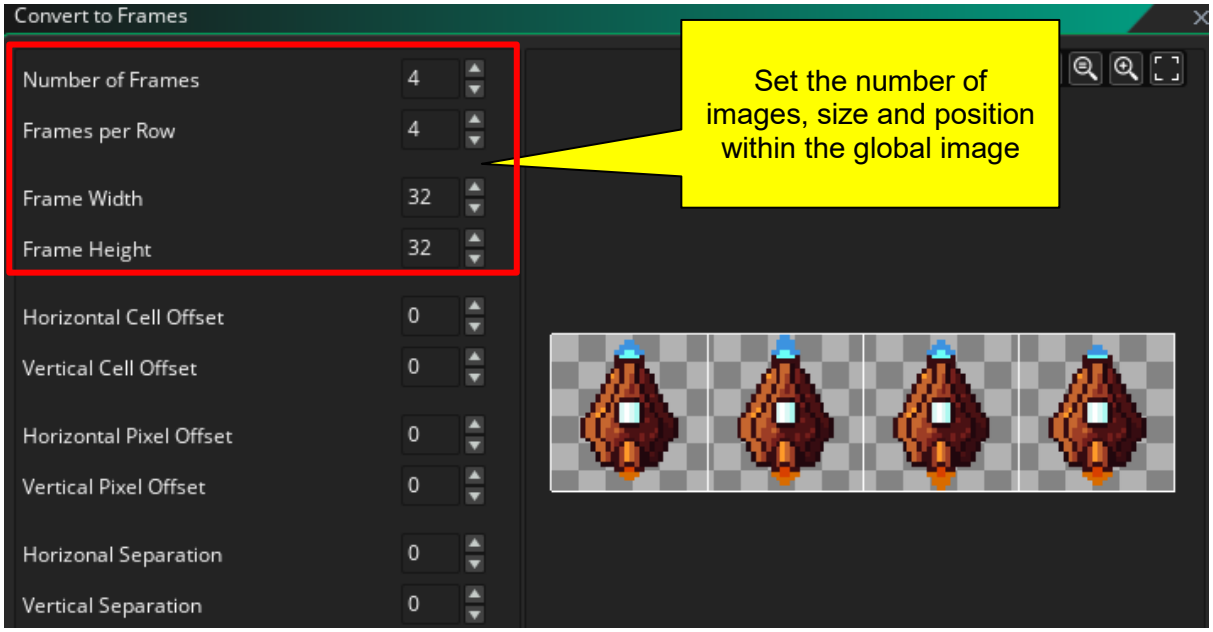


When you click the "Edit Image" button, a new tab in the center area of the IDE appears. This is a image editor that you can use to draw or modify images of your sprites.

When displaying the image editor the menú changes with some new submenus. We are going to use two tools highlighted in the following screenshot:



With the "Convert to frames" option, we will automatically split the image into four images.



And we will flip all the images upside down because our enemy ships will face our ship from top to bottom of the room. When we're done with the sprite, go to create the object for the enemy ship and assign the new sprite. You can test you object

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

To move the enemy ships we are taking a more standard and universal way of doing things. It might be helpful to learn how computer graphics are conceived in the following link. But the movement in the following example is itself so easy that most of the times, no extra learning is required. Enemy ships will move down at a constant speed on its own. Besides, unlike the player ship, we don't have to expect any special event like a keypress. So the idea is to set the speed and direction just from the beginning and let the object move down continually. We could do this mainly in two different ways:

- Using the "Create" event, which is the special event that happens once in the life of an object. So everytime a enemy is created, this event is triggered prior to anything else in that object goes on.
- Using the "Step" event, which happens every single frame.

Using the create event requires the speed and direction properties to be set and then we can forget about the object movement (unless we wish to maneuver the enemy ships). While that's also true for the step event, we are going to make use of another way of moving things. We are going to change the position, not the speed and direction, and by doing so continuously we're actually moving it down. Summarizing: Use the step event and increase the "y" property of the object for each frame with the following code:

```
y = y + 1;
```

If you test the game (placing one enemy object at least in the room), you can move the player ship to collide with the enemy ship, but nothing will happen since we've implemented nothing on this. Before we continue, let's solve something easy right now. Enemy ships will move down the room and eventually they'll fall outside and that object no longer has any involvement in the game. It won't hurt if we do nothing and let the object move down continuously, but it's not elegant, nor efficient. We have to destroy the object as soon as it reaches out. The destruction of an object is generally performed by the `instance_destroy()` function. In the step event that repeatedly changes the position of the enemy object, we can add a check condition to check if the object has to be destroyed. The condition is "location is below the bottom of the room" and the corresponding action "destroy this object" are specified this way:

```
if ( y  > room_height ) instance_destroy();
```

The above is a more complex sentence which is read as follows: *If the question - has the variable "y" a higher value than the variable "room_height" has a affirmative answer, then execute the instance_destroy() command*. The game engine will execute the above sentence by comparing the actual value of y (which is the height of the ship) to the value of "room_height" which is the vertical dimension of the game window. Remember, this comparison will take place during the actual game, not as we develop the game.

When running the game, unfortunately, it's not easy to verify that this works because the ship will disappear anyway. But you may change the condition to set the disappearing point at any height. For example:

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
if ( y  > room_height/2 ) instance_destroy();
```

would destroy any enemy ships reaching mid room.

Now, we are going to make the game create enemy ships randomly, they will appear from the top of the room and at any horizontal position. To do so, we'll rely on several functions and events:

- Use alarms to set the moment a new enemy is created
- Set random delays for the alarm
- Set random x coordinate position when creating a new enemy
- Set a -1 (or less) y coordinate to place the enemy above the room

Game maker allows the developer to use up to 12 alarms, they are named with numbers from 0 to 11. At any moment, the developer can set an alarm by calling the function

```
alarm_set(index, value);
```

Where index is the alarm number and value is the delay measured in number of frames. If you need to use time units (like seconds) you have to use the variable "room_speed" to get the frames of a given time. When an alarm goes off an "Alarm Event" is triggered. That's how you can react to the alarm expiration. Each object has its own alarms, so you can not set an alarm in one object and place the event in another.

The question then arises, Where do we set the alarm? If we ponder for a while we can transform the question, "What object does always exist?". Because, there, it's where we can set alarms and react to them. And the object is the player ship object. It may be a bit confusing that the player object controls the creation of enemy objects, but we have no other option as there is no other object alive forever (as long as the game runs). So we will do the following:

1. Set an Alarm as soon as the player ship is created, This will be done by using the "Create Event" of player object. We may choose now a constant time with this alarm

   ```
   alarm_set(0, 100);
   ```

2. Add a "Alarm Event" for the player object. This way we can react to the alarm.
3. The code for this event will have to create a new enemy object as specified some paragraphs above. So far we are just creating an enemy 100 frames after the game starts.
4. Still within the "alarm Event" we set again this same alarm to prepare the creation of another enemy ship sometime in the future. Te alarm sets itself over and over. The value for the alarm will be set taking the value from the random_range() function, which takes the lower and upper bounds of a random value;

Co-funded by the Erasmus+ Programme of the European Union

```
delay=random_range(100,400);
```

The resulting code for the action linked to the Alarm Event is:

```
enemy_x = random_range(0,room_width);
instance_create_layer(enemy_x, -1,layer,enemy_object);

random_delay = random_range(200,400);
alarm_set(0,random_delay);
```

The instance_create_layer() function takes 4 arguments:

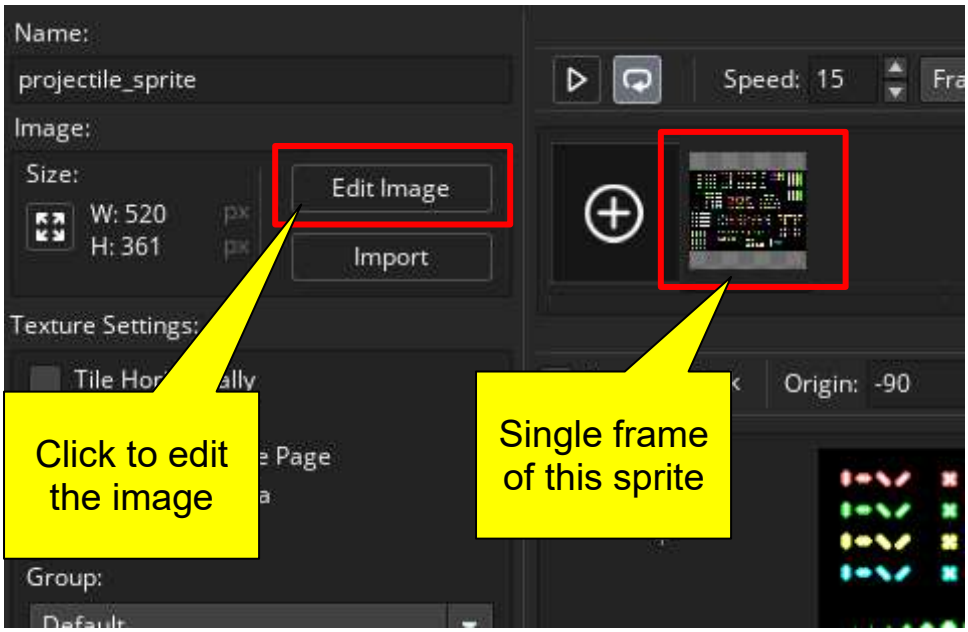
- X and Y coordinates where to create the object
- "layer" is a constant keyword. Game maker can arrange objects into layers one above others to simplify the correct overlapping of objects (for example when drawing planes above terrain objects, we want planes to actually hidden what is below). We do not have layers at this point so the target layer will be the one as the player object layer, hence the "layer" keyword.
- "enemy_object" is the object class from where an instance is to be created.

If you run the game, you'll see how enemy ships start appearing from top of the room. But they do not collide, nor shoot. We are going to add the ability to shoot bullets or projectiles to enemy ships. The shots will be objects and they will be visible as they have a sprite assigned to them. We first need to find or draw the projectile. For this tutorial we can get a set of projectiles got from : https://opengameart.org/content/bullet-collection-1-m484 . The downloaded image has several kinds of projectiles and laser beams in many directions. We just need to choose one and make a separate file or picture for it. You can do it in the GameMaker built-in image editor or with your favourite image editing software.

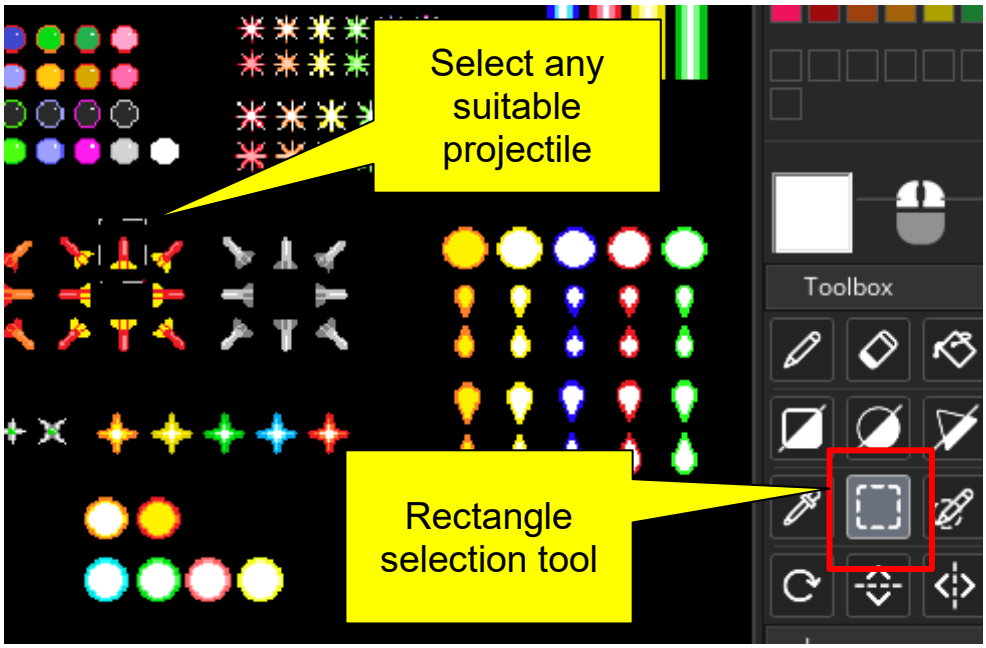
If you use GameMaker, once you create the sprite and load the image the sprite will be a large frame containing all the items. We do want just one of them, so we will

1. Open the image editor by clicking the "Edit Image" button. A new tab will appear

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



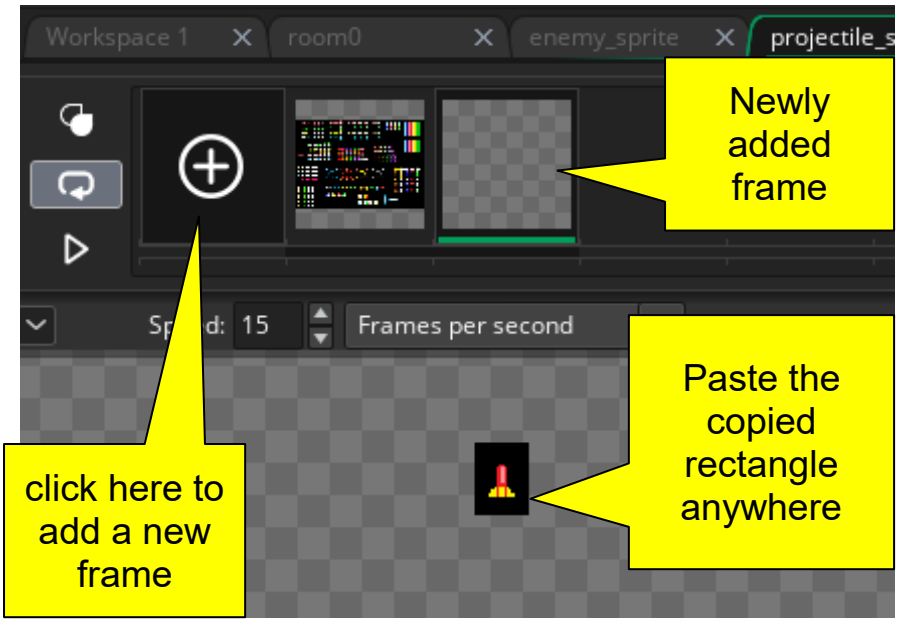
2. Cut the chosen projectile using the "rectangle select tool" and type ctrl+c to copy it.



3. Create a new frame for the sprite

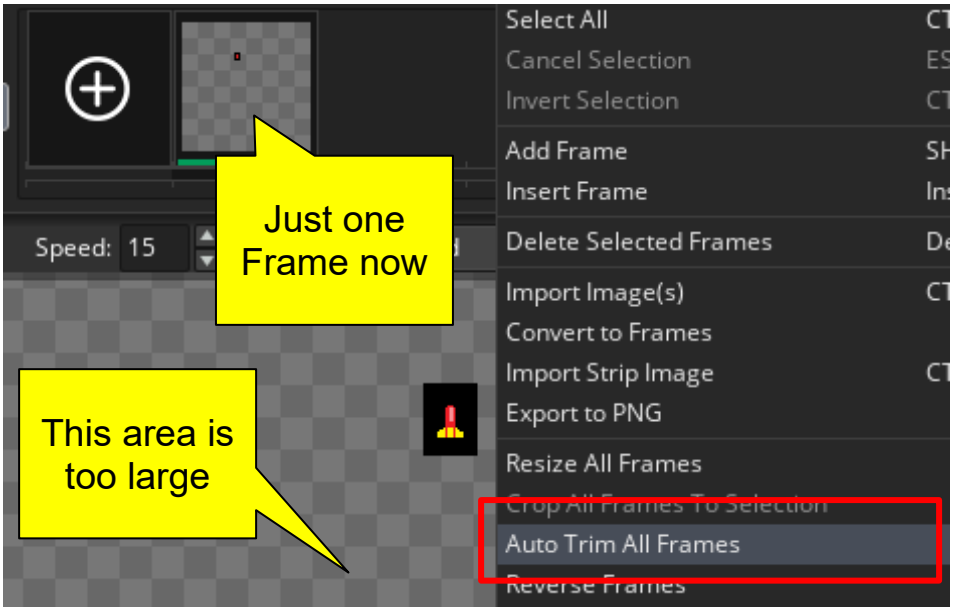
4. Paste or draw the selected projectile in the new Frame

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



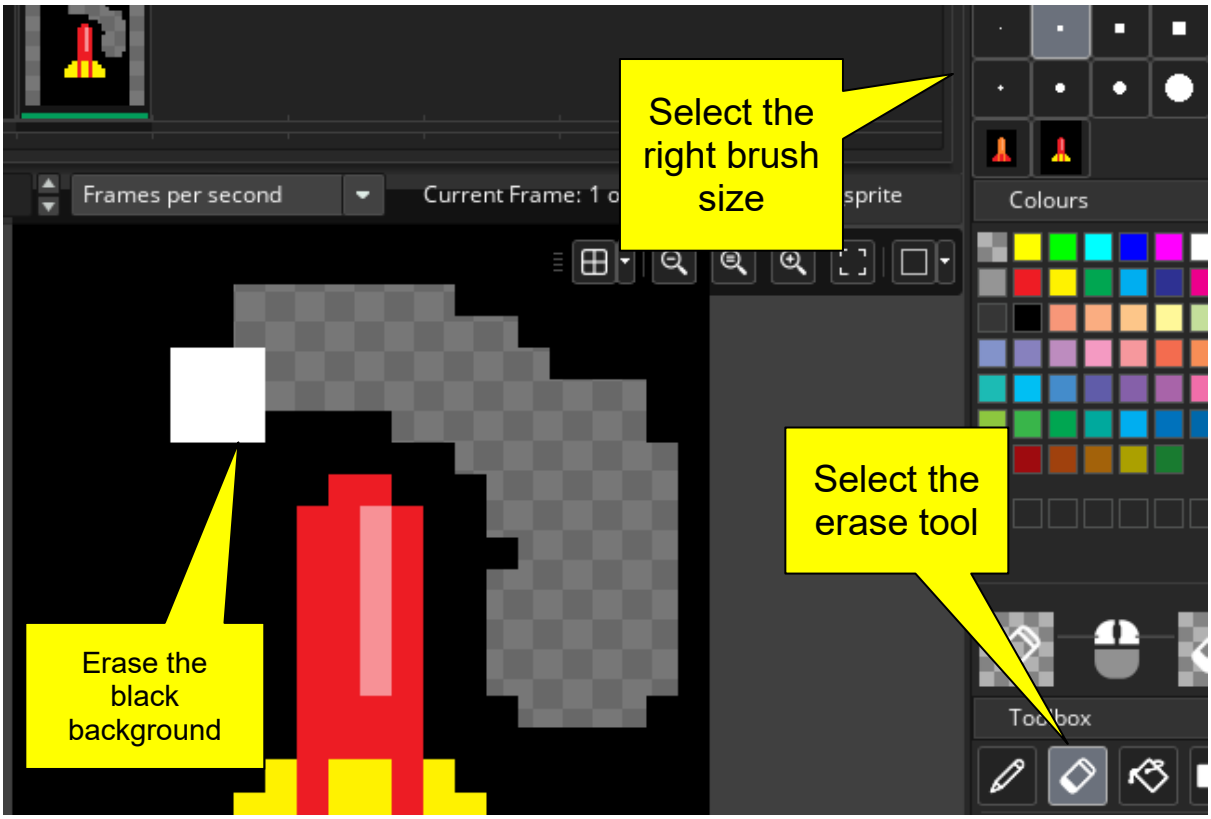
5. Delete the first image with all the items as it is no longer needed

6. "Autotrim all Frames" to adjust the size to the projectile used. This will shrink the canvas to fit the size of the projectile (together with the unwanted black background)



7. Delete or erase the black background to make it transparent or invisible so we will only see the projectile. This can be achieved by selecting the "Erase tool" and an appropiate brush size:

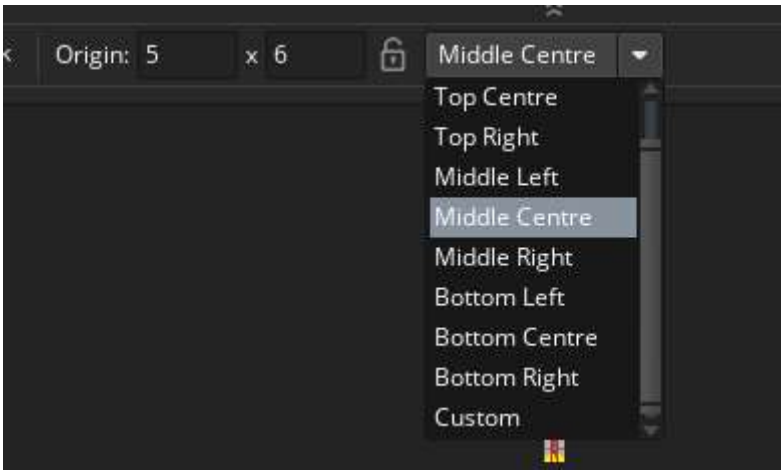
Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



8. In this case, you could alternatively select all the black pixels wit the "Magic Wand tool" and erase them by pressing "Del" key or the "Cut Selection" in the image menu.

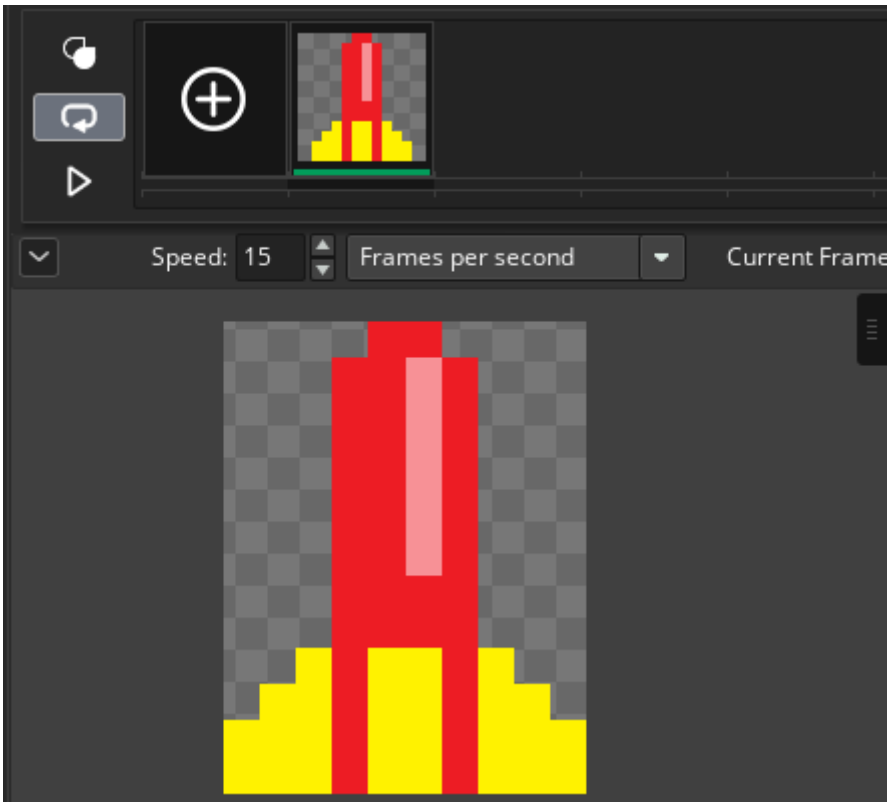
9. If the size of the canvas is still larger than the projectile you can apply again the "AutoTrim all frames" .

10. Finally, don't forget to properly set the center of the object for that, you can select the "Middle Centre" option in the Sprite Tab



The final result will look like:

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

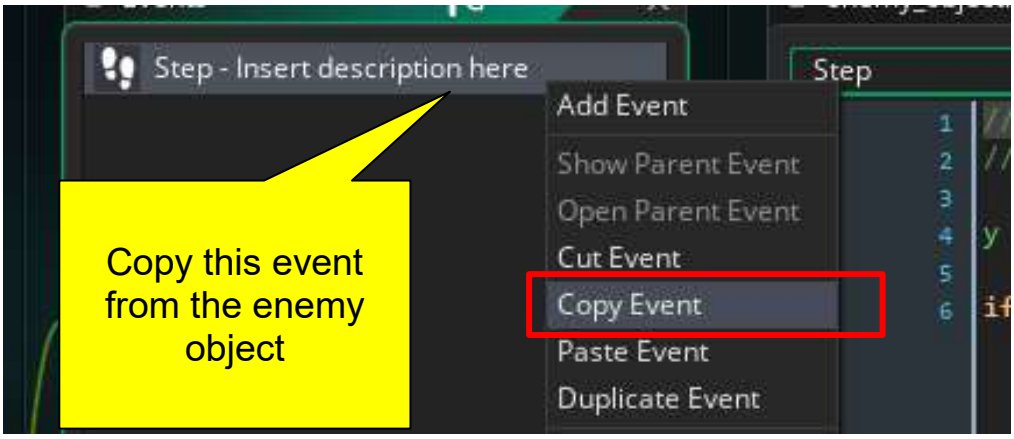


Now, we create the "projectile_object". Notice that some behaviour is very similar to that of the enemy ships.

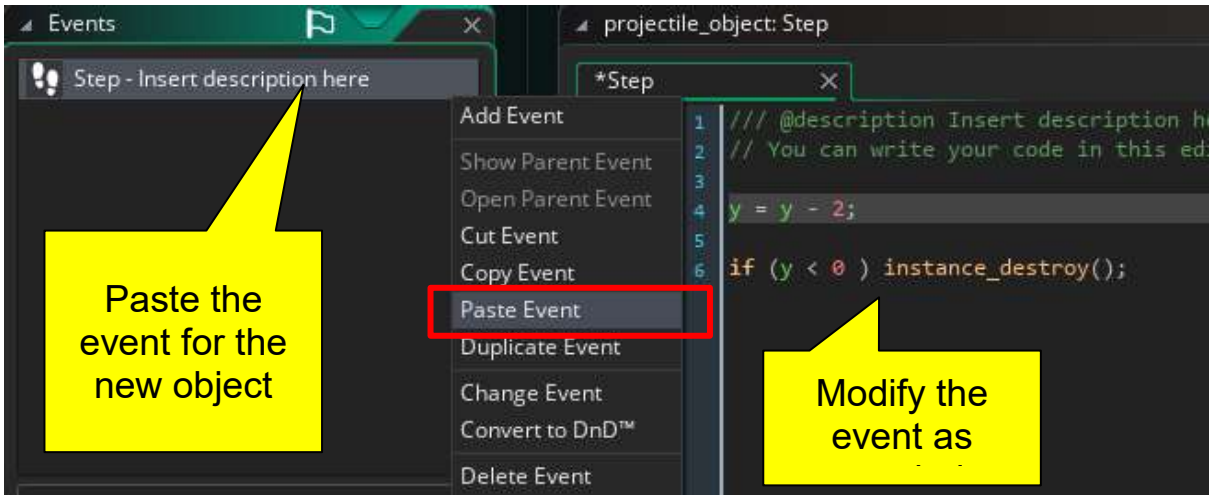
- They only move in vertical
- They are created by some other object, albeit not randomly in this case.
- They must be destroyed as soon as they fall out of the room
- They will collide with other objects killing them but also themselves.

For the most part we can take things already done from the enemy ship and replicate them in this object. For example, the "Step Event" which handles the movement of enemy ships is replicated here with minor changes (the projectile moves up and at a faster speed). Game Maker allows "Copy & Paste" of events between different objects which comes in handy here.

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Once copied, go to the projectile object and paste the event, then modify the code to make it travel up the room, at a faster speed (y = y + 2 for each frame).



To shoot we need to go back to our player ship object which is responsible in taking the keys pressed and create other objects. Two things arise now:

- Choose the key which will make the shot, typically the Space key, and set the KeyDown Event to create a new projectile. Notice that we are not using the "Key Pressed" event, just because it to behave a bit differently, the projectile will be shot as soon as we hit the key without needing to release it to make a complete keypress.d

  The action for this event is pretty simple and has been done already with enemy ships:

  ```
  instance_create_layer(x, y,layer,projectile_object);
  ```

  This is, we create a new instance of a projectile in the same position as the player ship is (hence the "x,y" coordinates given to the instance_create_function)

Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- The "cooldown" period. In most games there is a minimum time between shots, so the game doesn't become too easy for the player as he can shoot indiscriminately at everything.

  To get this done we can use Alarms again. Everytime a projectile is created, a variable ("can_shoot") is set to false. This prevents other projectiles to be created if the key is pressed again, additionally, an alarm is also set. When the alarm goes off, the variable changes the value and shooting becomes possible again. The "can_shoot" variable needs to be set to true upon object creation. Here's the code for the "KeyDown Event":

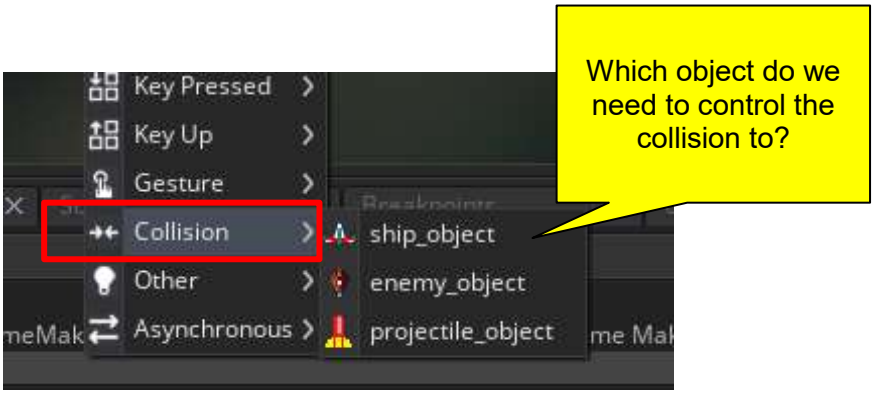
```
if (can_shoot)  {
    instance_create_layer(x, y,layer,projectile_object);
    can_shoot = false;
    alarm(1,30);
}
```

  The chosen Alarm must not be the same used before to create enemy ships. That's why we chose the "Alarm 1" this time. Now If you take a look, you'll see that just one shot can be made, because "can_shoot" is set to false and no longer the "if " condition will succeed. To complete this part, we need to set an Alarm 1 Event with the following simple code (which also has to be included in the Create event):

```
can_shoot = true;
```

The game is almost complete but it lacks the most interesting part: the collisions. Basically collisions will produce deaths or object destruction. These can be set up in some different ways, either on where to implement them and which object takes up the duty of destroying the objects. In the following chapter "Platform Game" we will implement collisions in a more detailed and precise way. Here we will use the "Collision Event" and to simplify things, we may put all the collision stuff into the Enemy object, because the player ship and the projectiles won't collide, but enemy ships can collide with both. Therefore the rule is "enemy ships destroy every other object" (and themselves).

Go to the enemy ship object, select "Add Event", go to the "Collision" Section and notice that we can set collision events for this object to any type of object.



Co-funded by the Erasmus+ Programme of the European Union
The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

It doesn't matter which one we do first, The resulting code will be the same:

```
instance_destroy(other);
instance_destroy();
```

You may think that something is missing in the collision between the enemy ship and the player ship: The "Game Over" condition. At this point you'd be right, but if the game gets more complex we may agree that the player ship could die from other game events (for example, running out of fuel). Therefore to set the game finalization It would be advisable to use the "Destroy Event" of the Player object, which will trigger whenever it gets to be destroyed for any reason.

Before we finish with this section, please notice that you've gone through some different concepts that can be used together to add many functionalities to the game, either new ones or changing the way the current game is played. For example, you can change the speed simply increasing the position change in every step. Or you can move the ship differently if you change the events used to detect keypresses and the actions taken.

But this is not enough to gather a general perception of Game Maker. In the next section we will discover how to make platform games and we'll use different elements to expand possibilities with Game maker.

Co-funded by the Erasmus+ Programme of the European Union