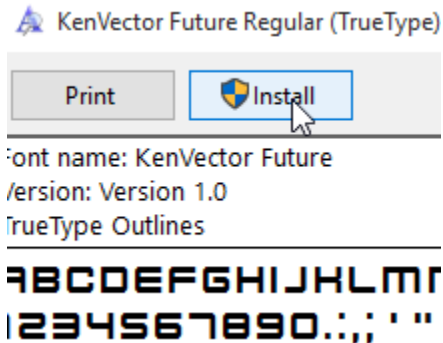


1 Space Rescue Guide

2 SETUP

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

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



Start a new Gamemaker project. Name it **SpaceRescue**. Make sure to pay attention to where you save the project – I would recommend you make a folder in your Documents folder for all your games.

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

NOTE2: TECHNICALLY, ROCKS FLOATING IN SPACE ARE ASTEROIDS. THEY ARE NOT METEORIDS UNTIL THEY ENTER AN ATMOSPHERE. IN THIS GAME, THEY ARE CALLED METEORS... IF THAT BOTHERS YOU, FEEL FREE TO CHANGE METEOR TO ASTEROID ANYWHERE YOU SEE IT.

3 BACKGROUND AND ROOMS

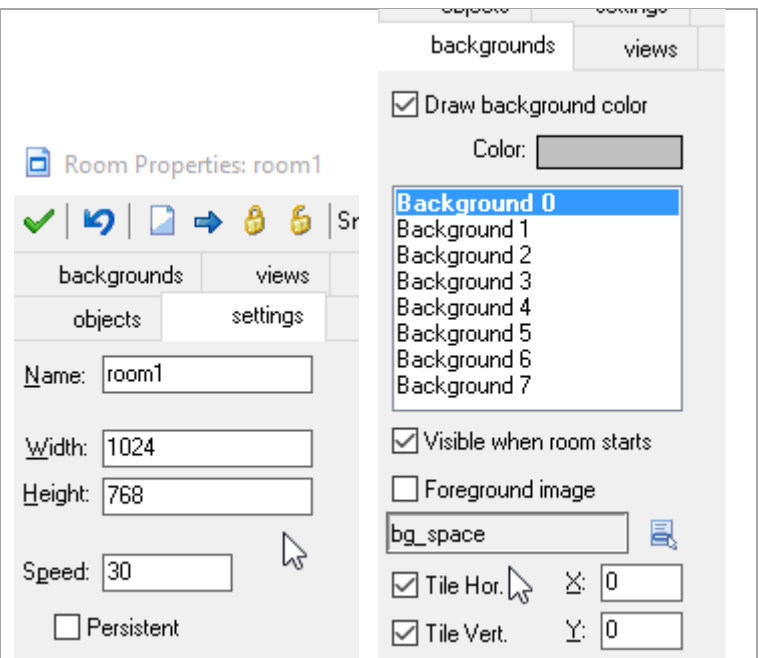
Load the image **bg_space.png** as a **Background**.

Make a new room called **room1** and set it to use that background. Make sure it is Tiled both horizontally and vertically.

Duplicate it and call the room **room2**

Duplicate it again to make **room3**

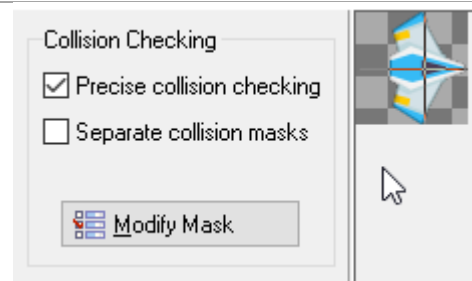
Duplicate it once more to make **roomEnd**



4 SPRITES

Load the image **playerShip_0.png** as a **Sprite** called **sprite_playerShip**

Set Precise collisions and move the **Origin** of the sprite to the middle of the ship (not center of the sprite)



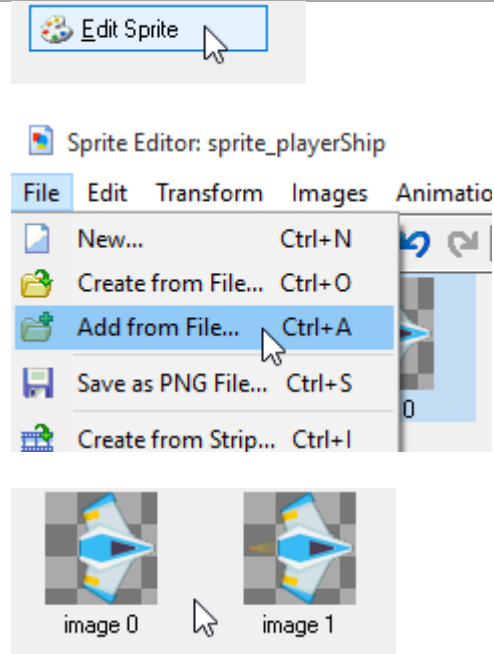
Now, still in the sprite properties, click the **Edit Sprite** button.

Then do **File → Add from File**

Navigate to the SpaceRescueFiles folder and select **playerShip_1.png**

You should end up with two *subimages*. The second is basically the same as the first but has some flames coming out the back of the ship.
When the ship is stationary we will show image 0.
When the ship is accelerating, we will show image 1.

Close the sprite editor and the sprite properties windows.



Make a **sprite_meteor** from **meteorBig_0.png**

Then edit the sprite and File → Add from File to add **meteorBig_1.png**, **meteorBig_2.png** and **meteorBig_3.png**

We will use these four different meteor images to make meteors in the game look different – we will pick a random one of the images for each meteor.

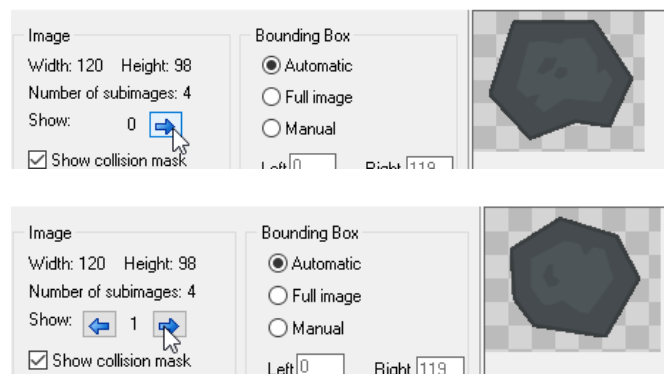
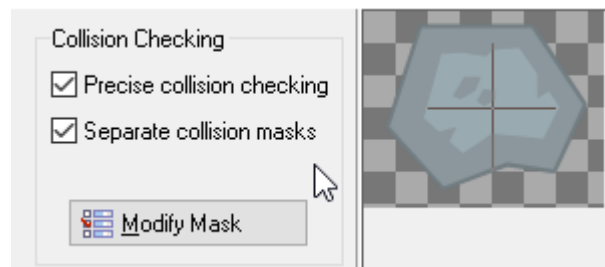
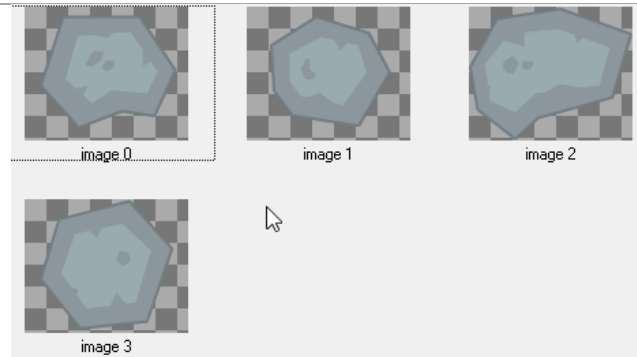
Back in the sprite properties, center the **Origin** and use **Precise Collisions**.

Also check **Separate Collision Masks**

This makes it so each subimage of the sprite (each meteor) uses a different area to count as colliding.

To see the masks, click **Modify Mask** button. Then you can use the arrow buttons to cycle through the images. Each should have a gray area indicating its collision mask that matches the image.

To compare, you can uncheck **Separate collision masks** back in the properties window and then look at the masks – you will see one big blob that does not match any of the images. Using that, you could collide even though your ship isn't touching the image you see.



Do the same steps to make **sprite_meteorSmall** from the **meteorSmall_0.png** and **meteorSmall_1.png** files



Do the same steps to make **sprite_explosion** from the **explosion_0.png** up to **explosion_8.png** files.

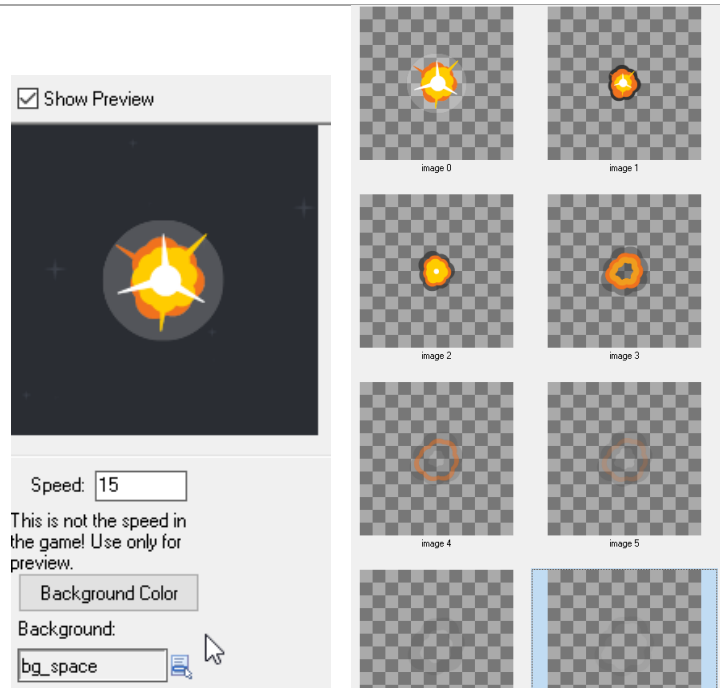
These 9 frames will play as an animation when the player crashes

You do **NOT** need **Precise collision** checking, it won't collide with anything.
Do center the **Origin**.

Check **Show Preview** to see it play

Below the preview, you can set the background and change the speed to see it play faster/slower and in front of different backdrops.

I like it at speed 15. That is half the normal speed (30).
Later we will make sure that this animation plays at half normal speed to match what we see here.

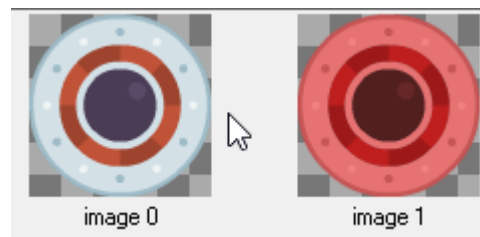


Make **sprite_beacon** from the two beacon .png images.

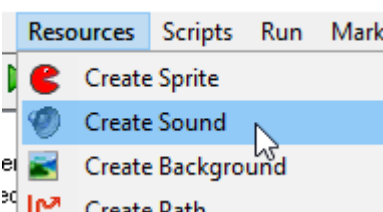
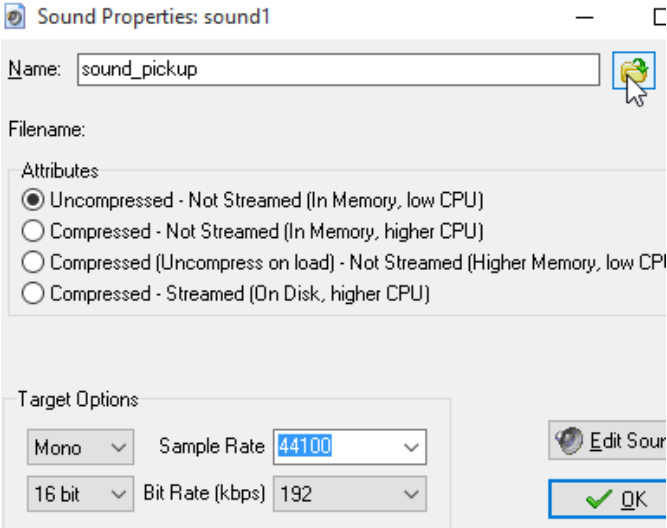
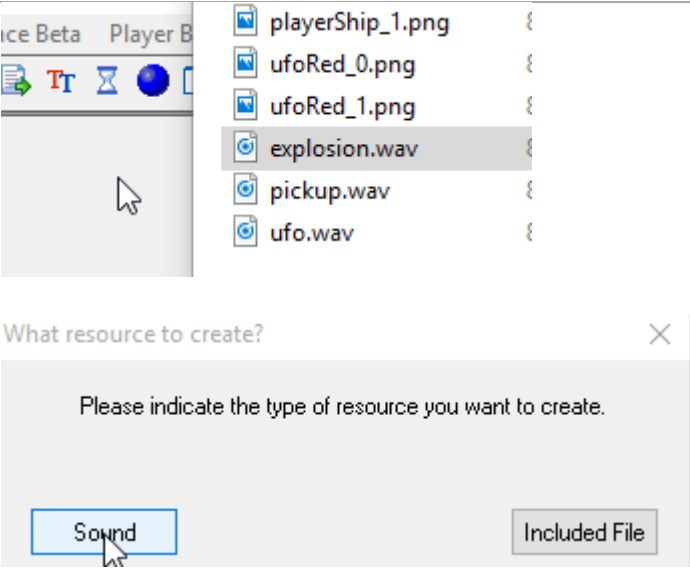


Make **sprite_ufoRed** from the two ufoRed .png images.

Both new sprites should have **Precise collisions** and centered **Origin**.



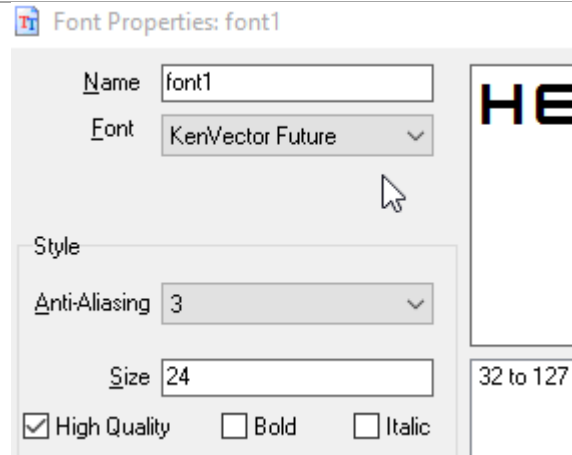
5 SOUNDS

| | |
|---|---|
| <p>From the Resources menu, do Create Sound</p> <p>Name the sound sound_pickup and then click the open file button. Navigate to the pickup.ogg sound in the SpaceRescueFiles folder and select it.</p> <p>For our purposes, uncompressed sounds are probably the best option.</p> |   |
| <p>You can also drag sound files into Gamemaker. Drag in explosion.wav and then select Sound as the type name it sound_explosion</p> <p>Do the same to bring in ufo.wav as sound_ufo</p> |  |

6 FONT

Add a new font – you can just call it **font1**

I use **KenVectorFuture** with size of **24** points, but anything approximately the same size will work



7 PLAYER

Add a new object, **object_player** and give it the **sprite_playerShip**

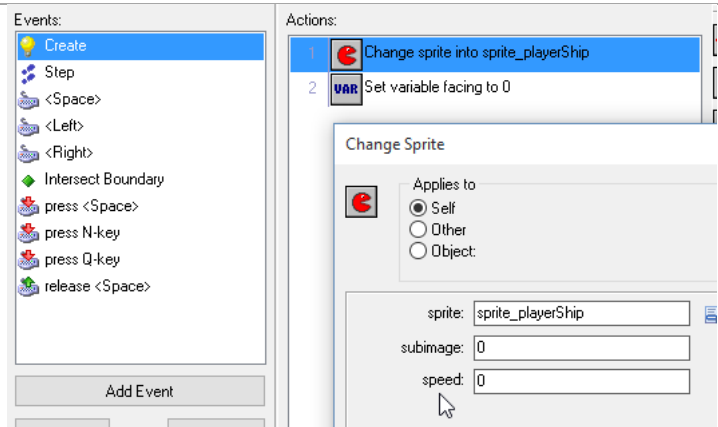
Add a **Create** event.

In it put a **Change Sprite** action. We are going to select the same sprite the object is already using, but we will set the **speed** to 0 and **subimage** to 0. This prevents the ship from switching automatically between the two subimages and makes sure it is showing the first image (number 0).

Then also in the Create event add a Set Variable and set **facing** to be **0**.

We will use “facing” to store the direction the ship is pointing in as the player steers it. We are using 0 to start because the sprite is pointing to the right and right is direction 0.

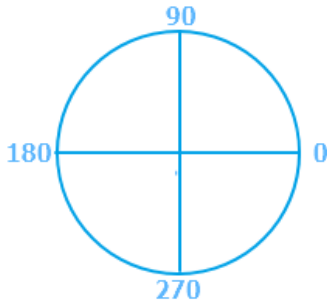
While Gamemaker already keeps track of the direction every object is moving using the built in variable *direction*, we want the ship to be able to point in a direction it is not moving. So we need to make our own variable to do so.



Add a **Keyboard <Left>** event. Make sure to use keyboard, not key press... we want this event to keep firing while the key is held down.

Add the action **Set Variable** and enter **facing** to **5 relative**. This will add 5 to the current value of "facing". Adding to an angle turns it counter-clockwise:

DIRECTIONS REMINDER:



Events:

- Create
- Step
- <Space>
- <Left>
- <Right>
- Intersect Boundary
- press <Space>
- press N-key
- press Q-key
- release <Space>

Add Event

Delete

Change

Actions:

1 **VAR** Set variable facing to 5

Set Variable

Applies to
☒ Self
☐ Other
☐ Object

variable: facing

value: 5

☒ Relative

Add a **Keyboard <Right>** event.

Add the action **Set Variable** and set **facing** to **-5 relative**. Subtracting 5 from an angle turns it clockwise.

IF YOU GO ABOVE 360, THE ANGLE ESSENTIALLY WRAPS BACK TO 0. IF YOU GO BELOW 0 THE ANGLE WILL WRAP AROUND TO 360.

To show that the player is spinning the ship, we need to rotate the sprite. This needs to happen every step to show the current direction of the ship.

Add a **Step** event

To it, add a **Transform Sprite** action. Leave the x and y scale at 1 (100%) and do not mirror it. For the angle, type **facing**

This tells Gamemaker to use the value currently stored in the variable facing as the angle to rotate the sprite. If the player has changed facing to 25 by holding the left key, we will rotate the sprite 25 degrees.

Events:

- Create
- Step
- <Space>
- <Left>
- <Right>
- Intersect Boundary
- press <Space>
- press N-key
- press Q-key
- release <Space>

Add Event

Actions:

1 Transform the sprite

Transform Sprite

Applies to
☒ Self
☐ Other
☐ Object

xscale: 1

yscale: 1

angle: facing

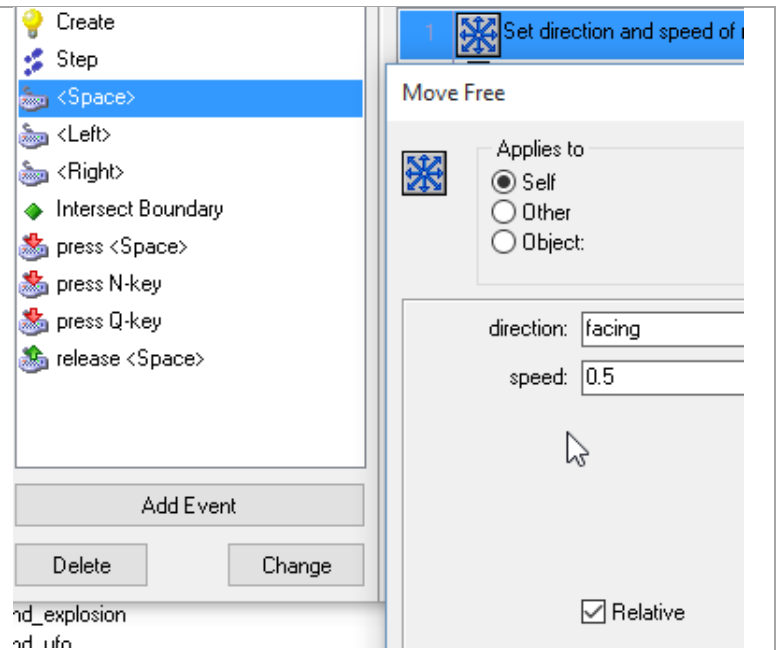
mirror: no mirroring

Place an **object_player** in room1. Test to make sure you can turn smoothly in both directions.

Add a **Keyboard <Space>** event.

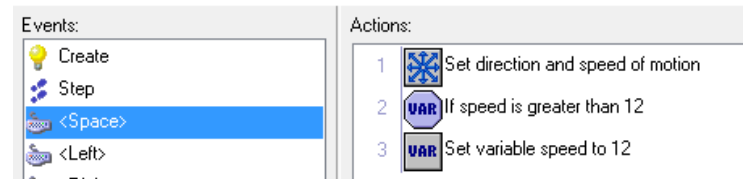
Add the action **Move Free** and set **direction** to **facing** and speed to **0.5 relative**.

This takes the current speed (relative checkbox) and adds to it 0.5 in the direction given by the variable "facing".



Then add the action **Test Variable** and check if **speed is greater than 12**

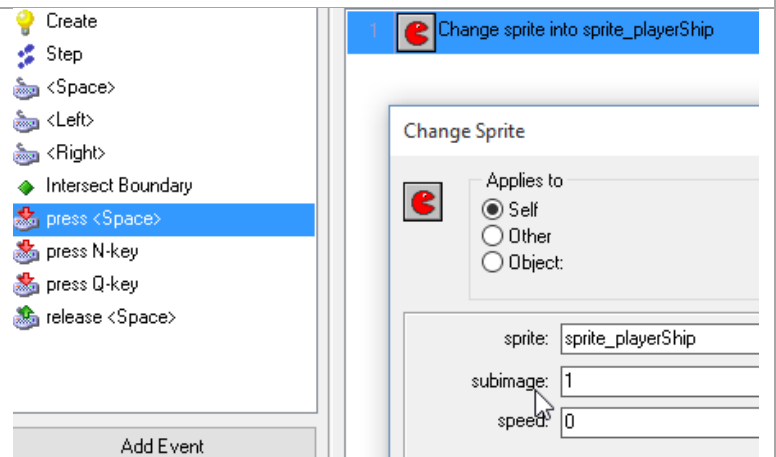
After it, place a **Set Variable** and set **speed** to **12**. Together, these say "If our speed is now above 12, change it to 12". This code is not needed, but keeps the ship from going too fast to control.



To make it clear that the ship is accelerating, we want to change the sprite of the ship.

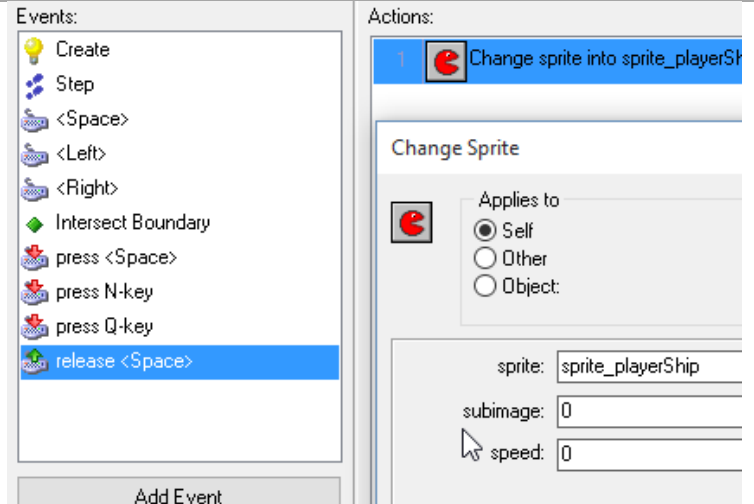
Add a **Key Press <Space>** event (NOT Keyboard, this only should happen once per press)

Use a **Change Sprite** action to change the **subimage** to **1** but keep the same sprite and speed of 0. We want to change which subimage we are showing to the second one (image 1) but we still do not want them to automatically switch.



Add a **Key Release <Space>** event (NOT Keyboard, this only should happen once when the key is released)

Use a **Change Sprite** action to change the **subimage** to **0** to go back to showing the ship without flames

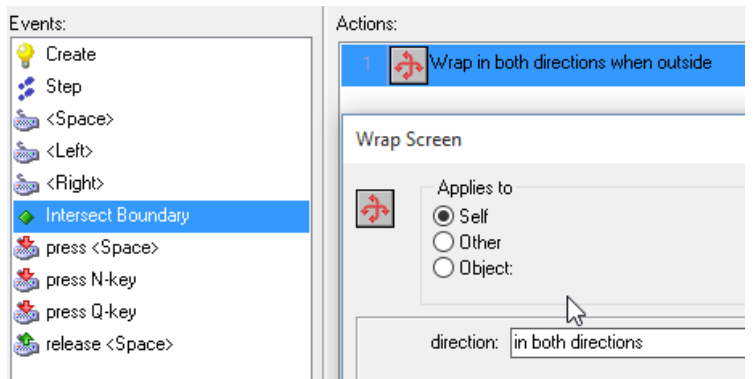
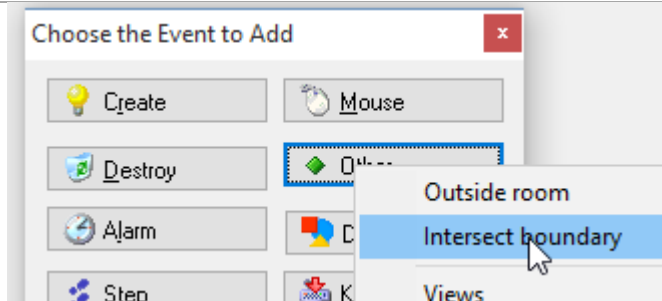


Now we want to make it so the player wraps around the screen.

Add **Other → Intersect Boundary** event

In it put **Wrap Screen** (from the move tab) and make sure to select **both directions**

Wrap Screen only works in an intersect boundary event – it automatically makes the object wrap around to the opposite side of the screen.



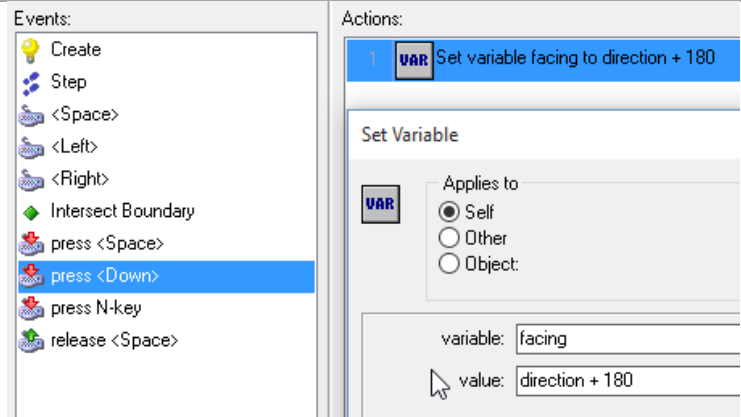
Test to make sure you can fly around and that you wrap around the screen when you go off it.

You may have noticed that coming to a stop is hard. To make it easier, we will add a key that turns the ship opposite its present course so you can decelerate.

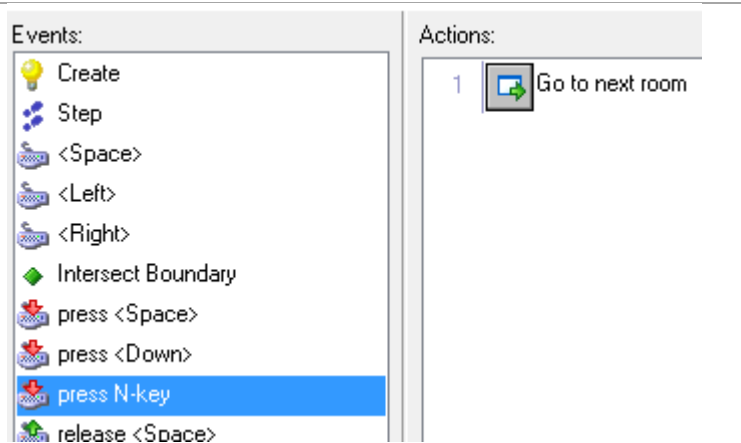
Add a **Key Press <Down>** event

Use a **Set Variable** action to change **facing** to **direction + 180**

“direction” is the name of the built in variable Gamemaker uses to keep track of the angle an object is moving. Adding 180 to an angle flips it around. So “direction + 180” says “take the current direction we are going and get the opposite angle” we set the variable “facing” which we use to control the angle of the ship to that value.



Lastly, add an event and action to make the N key move to the next room. This will come in handy while play testing your game once we add more rooms.



Test out the new functionality and make sure your ship works.

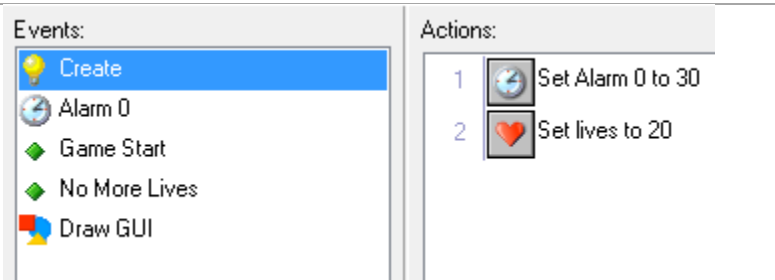
8 CONTROLLER 1

We will use a controller object to control the flow of the game in each room.

Make an object **object_controller1**

In a **Create** event, **Set Lives to 20** and **Set Alarm 0 to 30**

We will use lives to keep track of how much time the player has left in this room. The alarm will be used as our timer. (Remember 30 steps is one second).



| | |
|--|---|
| <p>Add an event for Alarm 0</p> <p>In it, Set Alarm 0 to go off again in 30 steps. Then Set Lives to -1 Relative.</p> <p>Together this will subtract one life (second from our timer) and set the alarm to go off again after another second so we can do it again.</p> | <div> <div>Events:</div> <div> Create Alarm 0 Game Start No More Lives </div> </div> <div> <div>Actions:</div> <div> 1 Set Alarm 0 to 30 2 Set lives relative to -1 </div> </div> |
|--|---|

Finally, add a **Draw GUI** event to show the time (lives) and score

Set Color to white (make sure to actually pick white, do not use the default “white”)

Set Font to **font1** and align **right**

Draw Score at position x: **1019**, y: **5** v with caption **Score**:

This will be in the upper right corner of the screen with text that grows from right to left.

Set Font to **font1** and align **left**

We will draw lives on the left side stretching from left to right.

Test Lives if **smaller** than **10**

Set Color to **red**

This will make the time left turn red if we are low on time.










Draw Lives at position x: **5**, y: **5** (upper left corner) with caption **Time Left**:

Place a controller in room 1 and make sure you see your score and time left. The time should count to 0 and then restart the room.

Events:

- Create
- Alarm 0
- Game Start
- No More Lives
- Draw GUI**

Actions:

- 1  Set the color
- 2  Set font to font1
- 3  Draw the value of score
- 4  Set font to font1
- 5  If lives are smaller than 10
- 6  Start of a block
- 7  Set the color
- 8  End of a block
- 9  Draw the number of lives

9 BEACONS

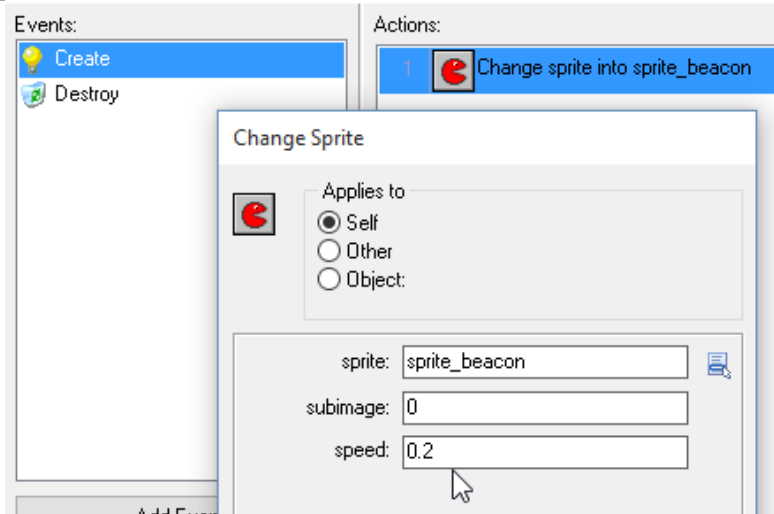
The objective of the game will be to rescue all the beacons in each room.

Make an object **object_beacon** using the **sprite_beacon**

In a **Create** event, **Set Sprite** to the same sprite but set the **speed** to 0.2

This will cause each subimage to be shown for 5 steps ($1 / 0.2 = 5$) before switching to the next subimage.

THE NORMAL SPEED OF A SPRITE IS 1. IT CHANGES BY ONE SUBIMAGE PER STEP. A SPEED HIGHER THAN 1 SPEEDS UP THE ANIMATION (SPEED 2 MEANS CHANGE BY 2 FRAMES PER STEP – IT WILL SKIP EVERY OTHER FRAME). A SPEED LOWER THAN 1 SLOWS DOWN THE ANIMATION. A NEGATIVE SPEED WILL PLAY THE ANIMATION BACKWARDS.

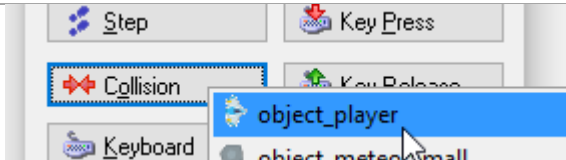


When a beacon is picked up it will be destroyed.

Add a **Collision** with **object_player** event

In it, **Destroy** the beacon

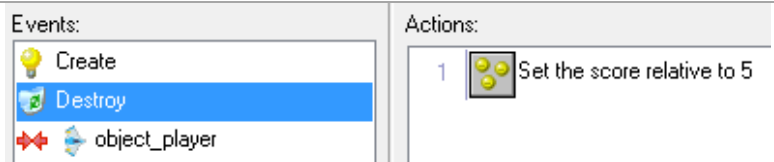
Then **Play Sound** (main1 tab) **sound_pickup**.



Add a **Destroy** event

Set Score relative 5 to add 5 points

DESTROY HAPPENS WHENEVER AN OBJECT IS DESTROYED. THE DESTROY EVENT ACTIONS ARE RUN JUST BEFORE THE OBJECT DISAPPEARS. IT IS A GOOD PLACE TO PUT CODE ASSOCIATED WITH AN OBJECT DISAPPEARING – THAT WAY WE KNOW IT WILL HAPPEN NO MATTER WHERE THE OBJECT IS DESTROYED FROM. IF YOU DECIDED YOU WANTED BEACONS TO BE DESTROYABLE IN BAD WAYS (NO POINTS), YOU PROBABLY WOULD PUT THIS CODE DIRECTLY IN THE COLLISION OBJECT_PLAYER INSTEAD OF IN THE DESTROY.



Now we want to see if this is the last beacon – if so, we should go to the next room.

Add a **Test Instance Count** (control tab) select **object_beacon**, **equal to**, and enter **1** for the number.

This says “If the number of object_beacons in the room is 1 (the one currently being destroyed), do the next instructions”



Use a **Start Block** and **End Block** to group the next two actions to both depend on the there being only 1 beacon. In the block, place:

Set Score relative to lives

This says change the existing score (relative) by the amount of lives we currently have. Since lives are being used to store the time left, this takes the time left in the room and adds it as a bonus to the player’s score.

Also add a **Go To Next Room**

Actions:

- 1  Set the score relative to 5
- 2  If the number of instances is a value

Test Instance Count









object:

number:

operation:

Actions:

- 1  Set the score relative to 5
- 2  If the number of instances is a value
- 3  Start of a block
- 4  Set the score relative to lives
- 5  Go to next room
- 6  End of a block

Set Score



new score:

☒ Relative

Add a few beacons to the first room.

Make sure you can collect beacons and earn points and that you go to the next (empty) room when you get the last one.

10EXPLOSION AND METEORS

When the player hits an obstacle, we want them to realize what happened – we don't want the level to just restart with no warning. We need to make an explosion appear for a few seconds before restarting the level.

Make an object **object_explosion** using the **sprite_explosion**

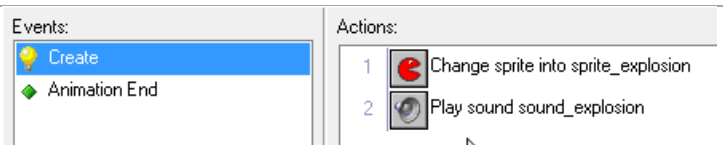
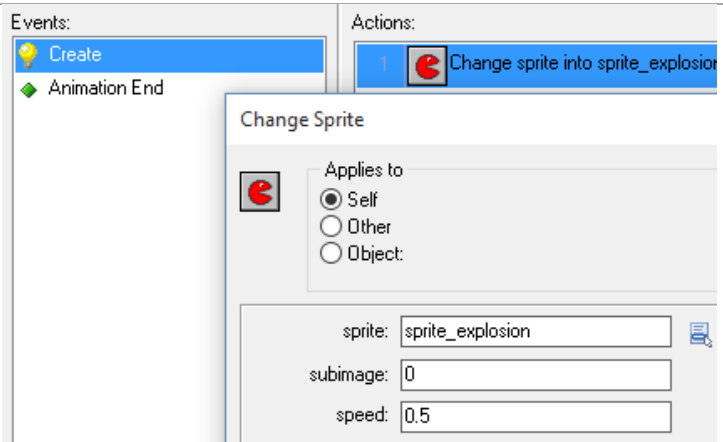
Set its **Depth** to **-10** this will force it to be drawn on top of other objects. We want to make sure it is visible.

OBJECTS ARE DRAWN IN ORDER ACCORDING TO DEPTH. THE BIGGER THE DEPTH, THE SOONER THE OBJECT IS DRAWN: AN OBJECT AT DEPTH 500 IS DRAWN BEFORE SOMETHING AT DEPTH 200 WHICH IS DRAWN BEFORE SOMETHING AT DEPTH 0. OBJECTS DRAWN LATER COVER UP EARLIER ONES. WHEN TWO OBJECTS AT THE SAME DEPTH OVERLAP YOU CANNOT PREDICT WHICH WILL SHOW UP ON TOP OF THE OTHER. THE DEFAULT DEPTH IS 0, SO OUR -10 MAKES SURE THIS APPEARS ON TOP OF THE REST OF THE OBJECTS IN THE GAME.

In the **Create** event use a **Set Sprite** to force the speed to play at speed **0.5**

Remember that while setting up the sprite, I liked how it played at half speed (speed 15 instead of 30). This is how we make it play at half speed in game.

Then add a **Play Sound** to play **sound_explosion**



Add an **Other** → **Animation End** event


This happens when the sprite is done displaying its last subimage.

Put in a **Restart Room** to start the play over after they see the explosion.

Events:

- Create
- Animation End**

Actions:

- 1  Restart the current room

You can do a quick test by putting an explosion in room1 and making sure you see it play and then the room restarts. Then take it back out of the room (CTRL-Right Click to remove).

Now make an **object_meteor**

We could make four different meteor sprites and four different meteor objects, but that seems like a lot of work just to make the meteors all look different. Instead we will use the one meteor sprite with 4 frames and pick a random one for each meteor to show.

Add a **Create** event


In it do **Set Sprite** to **sprite_meteor** speed **0** subimage **random(4)**

“random(4)” says “pick a number from 0-3”. Since the four subimages of the meteor sprite are numbered 0, 1, 2 and 3, this picks one at random. Speed 0 makes sure we stick with that subimage instead of constantly switching.

Object Properties: object_r

Name:


Sprite


Events:

- Create**

Actions:

1  **Change sprite into sprite_meteor**

Change Sprite

 Applies to

☒ Self

☐ Other

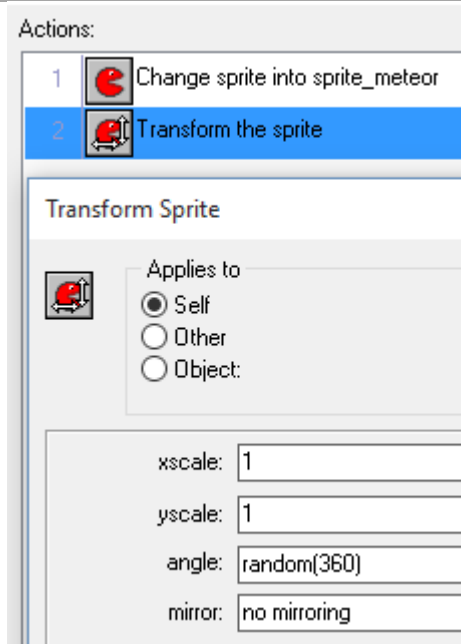
☐ Object:

sprite:

subimage:

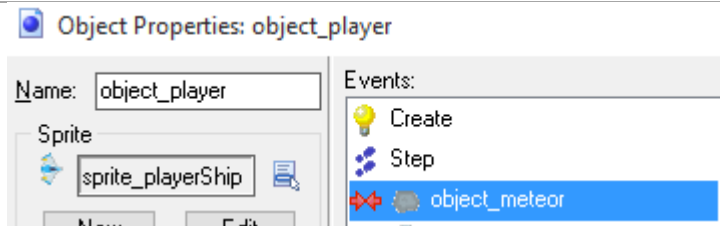
speed:

Then add a **Transform Sprite** and set the angle to **random(360)**
This will spin the sprite to a random angle (0 to 359 degrees).



Put a few meteors in room1 to make sure they look right. You can't hit them yet but you should see them.

Go back to **object_player** and add an event **Collision** with **object_meteor** event

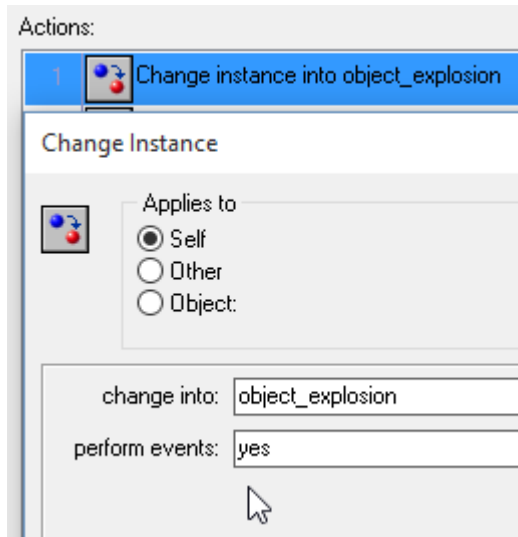


In it place a **Change Instance** action.
This turns the current object into a new type of thing.

Select **object_explosion** and specify **yes** to perform events.

WHEN AN OBJECT CHANGES INTO ANOTHER THING IT KEEPS ALL OF ITS CURRENT INFORMATION (X AND Y POSITION, SPEED, DIRECTION, ANY EXTRA VARIABLES YOU MADE). PERFORM EVENTS YES MEANS DO THE DESTROY EVENT OF THE OLD OBJECT AND CREATE EVENT OF THE NEW OBJECT. PERFORM EVENTS NO MEANS DO NOT DO THE DESTROY AND CREATE LOGIC.

We need to perform the create event as we turn into the explosion object because it sets the animation speed.



Then add the rest of the shown actions:







Set Score relative to -20

Test if **score** is **smaller** than **0**

Set Score to 0

This will subtract 20 points but not let the player go below 0.

Actions:

- 1  Change instance into object_explosion
- 2  Set the score relative to -20
- 3  If score is smaller than 0
- 4  Start of a block
- 5  Set the score to 0
- 6  End of a block

Make sure you can explode when you ram the meteors.

Make an **object_meteorSmall** using the appropriate sprite.

Click the drop down menu next to the **Parent** button and select **object_meteor**


This will do two things

1. Make object_meteorSmall count as an object_meteor... thus the player's ship will explode when it hits a small meteor.
2. Make object_meteorSmall behave just like the object_meteor. The small meteor will automatically run the same code as the normal one.

SELECTING A PARENT IS HOW WE SAY "THIS NEW THING IS JUST A SPECIAL TYPE OF THIS OTHER KIND OF THING".

ANY EVENTS AND ACTIONS YOU HAVE IN THE PARENT OBJECT AUTOMATICALLY APPLY TO THIS NEW OBJECT TYPE (THE CHILD TYPE) – YOU DO NOT HAVE TO COPY AND PASTE THEM TO THE NEW OBJECT. YOU WILL NOT SEE THE CODE IN THE CHILD OBJECT TYPE, BUT THEY WILL FUNCTION AS IF THEY WERE THERE. THIS IS CALLED INHERITING A BEHAVIOR.

ALSO, ANYTHING THAT INTERACTS WITH THE PARENT TYPE WILL INTERACT IN THE SAME WAY WITH THE CHILD TYPE WITHOUT ANY EXTRA WORK.

 **Object Properties: object_meteorSmall**

Name:

Events:

Sprite



New

Edit

☒ Visible ☐ Solid
☐ Persistent ☐ Uses Physics

Depth:

Parent:

Mask:

Children:

<no parent>
object_player
object_meteorSmall
object_meteor

If you try putting meteorSmall's in the room and running the game, you will notice that they appear to turn into the large meteors. That is because they are running the **Create** event of the **object_meteor** because **object_meteor** is their parent. It turns their sprite into the regular meteor one.

To fix this, add a Create event to **object_meteorSmall**

In it **Set Sprite** to the small meteor sprite, with subimage **random(2)** and speed **0**. This will make sure we use the right sprite and pick one of the two subimages at random.


Finally **Transform Sprite** – leave everything normal except for angle which should be **random(360)** to spin the sprite a random direction.

ADDING YOUR OWN VERSION OF AN EVENT REPLACES THE INHERITED VERSION OF THE EVENT THIS OBJECT WOULD OTHERWISE DO. IT SAYS SOMETHING LIKE, "I KNOW MY PARENT DOES IT SOME OTHER WAY... BUT THIS IS HOW I WANT TO DO THE CREATE EVENT"


Test the small meteors and make sure they show up with their own sprite but otherwise behave like normal meteors (you can crash into them).


Object Properties: object_meteorSmall

Name:



Sprite: 

☒ Visible ☐ Solid

Events:  Create

Events:  Create

Actions:

- 1  Change sprite into sprite_meteorSmall
- 2  Transform the sprite

11ROOM 2 & MOVING METEORS


In room 2 we will introduce some new challenges. We also want to give the player a little more time. To do so, we will need to make a new controller object to handle room 2.

Make an **object_controller2** and set its parent to **object_controller1**

If we do nothing else, the controller2 will behave just like controller1.

Object Properties: object_cont


Name:


Sprite: 

☒ Visible ☐ Solid

☐ Persistent ☐ Uses Physics

Depth:

Parent: 

Mask: 

To make controller2 give more time, we need to adjust the **Create Event**. That is where we set the lives, which represent time left.

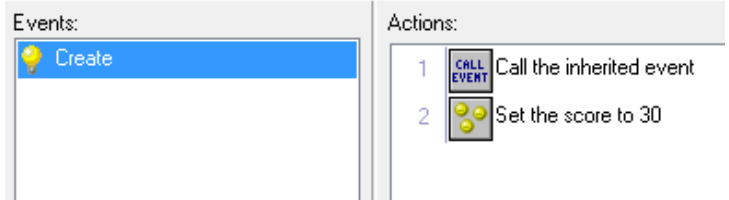
Add a **Create Event** to override the parent version with our own.

In it add a **Call Parent Action** (control tab).

This says to go do the parent version of this event. controller1 uses create to set the lives and set an alarm to start counting, so that is what will happen here.

After that place a **Set Score** to 30. The parent code will have set the score (time left) to 20. This will override that.

YOU CAN PUT THE CALL PARENT EVENT ACTION AT THE START, END OR MIDDLE OF THE ACTION LIST. BUT IF YOU WANT TO CHANGE SOMETHING THE PARENT CODE DOES, YOU NEED TO CALL THE PARENT FIRST, THEN DO YOUR OWN CODE.



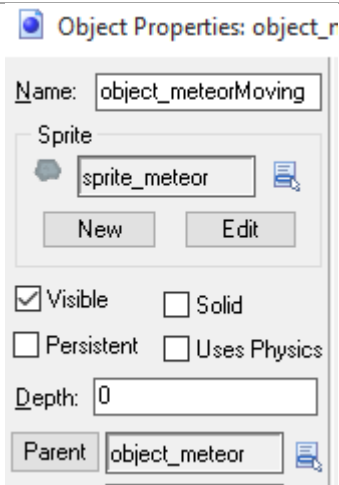
The screenshot shows the 'Events' panel on the left with a single event named 'Create' highlighted. The 'Actions' panel on the right contains a list of two actions: 1. 'CALL EVENT' with the label 'Call the inherited event', and 2. 'Set the score to 30' with a score icon.

There is nothing else to do... all the other code from the parent should work just fine.

Place a controller2 in room2 and make sure you get 30 seconds in that room. Don't forget you should be able to use the N key to skip to the next room.

Now lets add some moving meteors.

Make an **object_meteorMoving** set its parent to be **object_Meteor**



The screenshot shows the 'Object Properties' window for 'object_r'. The 'Name' field is 'object_meteorMoving'. The 'Sprite' section shows 'sprite_meteor' selected. Below the sprite are 'New' and 'Edit' buttons. There are checkboxes for 'Visible' (checked), 'Solid', 'Persistent', and 'Uses Physics' (unchecked). The 'Depth' field is set to '0'. The 'Parent' dropdown is set to 'object_meteor'.

We need to change the creation behavior to start it moving. Add a **Create** event

It should do the parent's sprite setup work so first add a **Call Parent Event**

Then to make it move in a random direction at a random speed, use a **Move Free** with direction **random(360)** and speed **random(4) + 1**

This should make each meteor go in a random direction and at a random speed between 1 and 4.99999

RANDOM ALWAYS GIVES YOU A NUMBER BETWEEN 0 AND NOT QUITE THE NUMBER YOU SPECIFY. RANDOM(4) ACTUALLY GIVES US 0-3.99999 BUT WHEN GAMEDMAKER NEEDS A WHOLE NUMBER, LIKE WHEN YOU ARE SETTING A SUBIMAGE, IT IGNORES THE DECIMAL PART AND WOULD TREAT THE NUMBER AS 0-3.

The screenshot shows the Event Editor interface. In the 'Events' list, 'Create' (lightning bolt icon) and 'Intersect Boundary' (diamond icon) are listed. In the 'Actions' list, '1 Call the inherited event' (CALL EVENT icon) and '2 Set direction and speed of motion' (Move Free icon) are listed. The 'Move Free' action is expanded, showing 'Applies to' set to 'Self', 'direction' set to 'random(360)', and 'speed' set to 'random(4) + 1'.

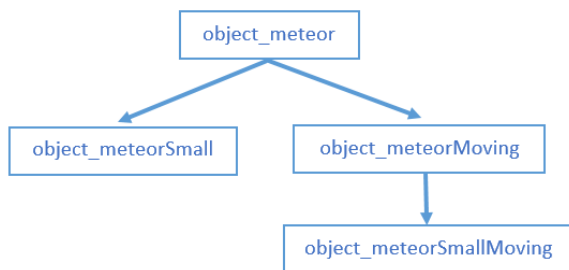
To keep the meteor from disappearing, add an **Other → Intersect Boundary** event

Just like the player ship, we want to **Wrap Screen** in both directions

The screenshot shows the Event Editor interface. In the 'Events' list, 'Create' and 'Intersect Boundary' are listed. In the 'Actions' list, '1 Wrap in both directions when outside' (Wrap icon) is listed.

Now make an **object_meteorSmallMoving** and set its Parent to **object_meteorMoving**

This means the meteorSmallMoving is a special type of meteorMoving. And since meteorMoving is a special type of meteor, so is the SmallMoving one. A SmallMoving meteor gets all the code from meteor and from meteorMoving.



AN OBJECT TYPE CAN ONLY HAVE ONE DIRECT PARENT. BUT IF THE PARENT HAS A PARENT, IT GETS CODE FROM ITS GRAND PARENT AS WELL.

Note that we could have made SmallMoving be a child of Small. But there is more code from the Moving that we want to inherit than there is in the Small. We have to pick one or the other so we are choosing the one that gets us the most “free” code.

The screenshot shows the 'Object Properties: object_m' window. The 'Name' field is 'ect_meteorSmallMoving'. The 'Sprite' field is 'sprite_meteorSma'. The 'Parent' field is set to 'object_meteorMov'. Other fields include 'Visible' (checked), 'Solid' (unchecked), 'Persistent' (unchecked), 'Uses Physics' (unchecked), 'Depth' (0), and 'Mask' (same as sprite).

The one thing we have to customize is the sprite that gets used.

Add a **Create** event and a **Call Parent** action
This will run the object_Moving version of Create

Then **Change Sprite** to the **sprite_meteorSmall** with a **random(2)** subimage and speed **0**
This will change the sprite back to a random small one.

The screenshot shows the Scratch IDE's 'Events' and 'Actions' panels. In the 'Events' panel, the 'Create' event is selected. In the 'Actions' panel, the 'Change sprite into sprite_meteorSmall' action is selected. Below this, the 'Change Sprite' sub-panel is visible, showing 'Applies to' set to 'Self', 'sprite' set to 'sprite_meteorSmall', 'subimage' set to 'random(2)', and 'speed' set to '0'.

Place 1-2 Moving and SmallMoving meteors in room 2 and give it a try.

12Room 3 & UFOs

Add an **object_UFO** using the matching sprite

In a **Create Event**, use **Change Sprite** to set the animation speed to **0.5**

The screenshot shows the Scratch IDE's 'Object Properties' and 'Events/Actions' panels. The 'Object Properties' panel shows the object named 'object_UFO' with the 'sprite_ufoRed' sprite assigned. Below this, the 'Events' and 'Actions' panels are shown. In the 'Events' panel, the 'Create' event is selected. In the 'Actions' panel, the 'Change sprite into sprite_ufoRed' action is selected. Below this, the 'Change Sprite' sub-panel is visible, showing 'Applies to' set to 'Self', 'sprite' set to 'sprite_ufoRed', 'subimage' set to '0', and 'speed' set to '.5'.

UFOs will seek out the player.

Add a **Step Event**

Every step we will change the UFO's course to head towards the player

In it use **Move Towards**

It allows you to pick an x and y location for the object to move to.

Specify x as **object_player.x** and y as **object_player.y** and speed as **2**

WHEN YOU SAY OBJECT_PLAYER.X IT MEANS "THE X (HORIZONTAL LOCATION) OF THE OBJECT_PLAYER". YOU COULD ALSO SAY SOMETHING LIKE OBJECT_PLAYER.SPEED TO MEAN "THE SPEED OF OBJECT_PLAYER".

IF YOU REFER TO SOMETHING THAT THERE ARE MULTIPLE OF, LIKE "OBJECT_METEOR.X", GEMMAKER WILL PICK ONE OF THEM TO BE THE ONE THAT IS USED.

IF YOU REFER TO SOMETHING THAT IS NOT IN THE ROOM, LIKE "OBJECT_EXPLOSION.X" BEFORE THERE IS AN EXPLOSION, YOU WILL GET AN ERROR AND YOUR GAME WILL STOP.

Events:

Create
Step

Actions:

Move towards point (object_player.x,object_player.y)

Move Towards

Applies to
☒ Self
☐ Other
☐ Object:

x: object_player.x

y: object_player.y

speed: 2

As mentioned, trying to do object_player.x if there are no object_players in the room will make your game crash.

You can test this by putting a UFO in room 1 and crashing into a meteor. As soon as the player turns into the explosion, there is no more object_player to get an x or y from.

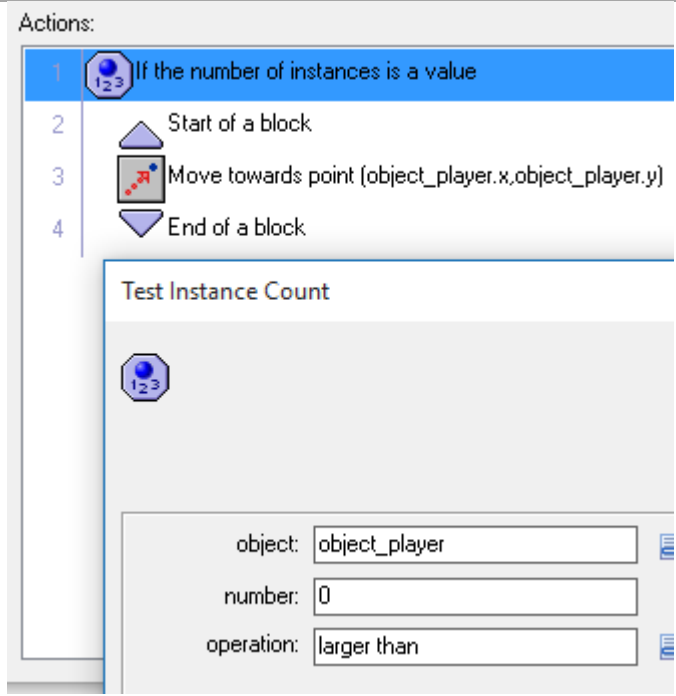
The error message you get will look like the picture to the right. Notice it gives you hints where the problem is: **"action number 1 of the Step Event of objectUFO"** and **"Unable to find instance for object named object_player"**

IF YOU GET AN ERROR MESSAGE LIKE THIS YOU ARE NOT EXPECTING, EXAMINE IT TO SEE WHERE THE PROBLEM IS AND IF YOU CAN GET ANY HINTS ABOUT WHAT THE PROBLEM IS.

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

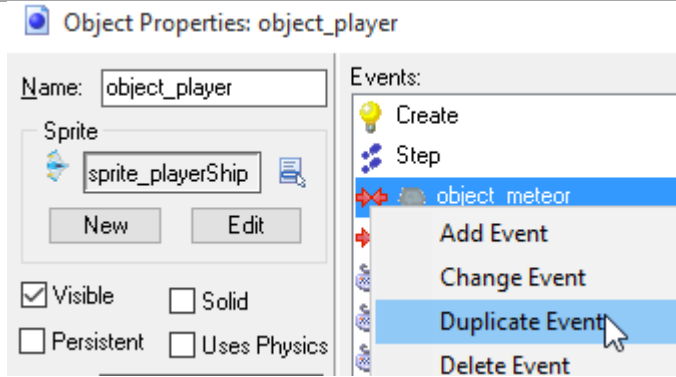
```
Unable to find any instance for object index '0' name 'object_player'  
at gml_Object_object_UFO_StepNormalEvent_1 (line 1) - action_move_point( object_pla  
#####
```

To fix the problem, put a **Test Instance Count** action in before the **Move Towards** to verify that the number of `object_player`'s is larger than 0. This says "If there are more than 0 player's, move towards one of them". If there are 0 players, it will not do anything.



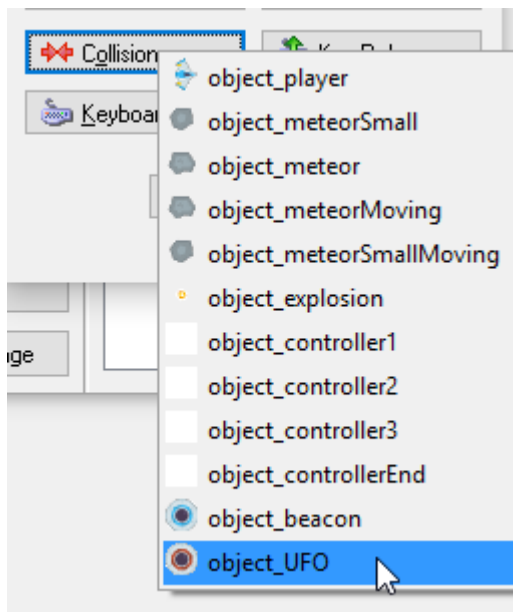
To make the player crash into the UFO, let's go back to the **object_player**

The logic for hitting a UFO will be the same as hitting a meteor. So right click **Collision** with **object_meteor** and choose **Duplicate**



Pick **Collision** with **object_UFO**

This should make a new event just like the old one but for hitting the UFO.




To control the third room and make a UFO, Create a new object, **object_controller3**

Have it use **object_controller1** as its parent.



Object Properties: object_c

Name:

Sprite
 





☒ Visible ☐ Solid
☐ Persistent ☐ Uses Physics

Depth:

Parent: 
Mask: 




In the **Create, Call Parent Code**, then override the lives to 25 (25 seconds) with a **Set Lives**
Finally, set **Alarm1** for **150** steps (5 seconds)

Note: we can't use Alarm0 – that is being used by the parent code to count off the seconds. If we tried using Alarm0 here, it would override that behavior and mess up the counter!

| Events: | Actions: |
|--|--|
|  Create | 1  Call the inherited event |
| | 2  Set lives to 25 |
| | 3  Set Alarm 1 to 150 |

Add an **Alarm 1** event to handle the alarm going off

In it **Create Object object_UFO** at 0, 0 not relative (upper left corner of the screen)

| Events: | Actions: |
|---|--|
|  Create | 1  Create instance of object objec |
|  Alarm 1 | |

Create Instance

Applies to
☒ Self
☐ Other
☐ Object:

object:

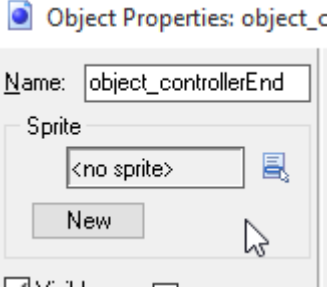
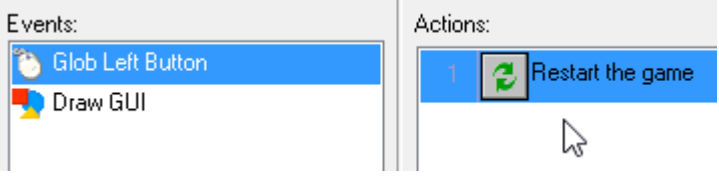
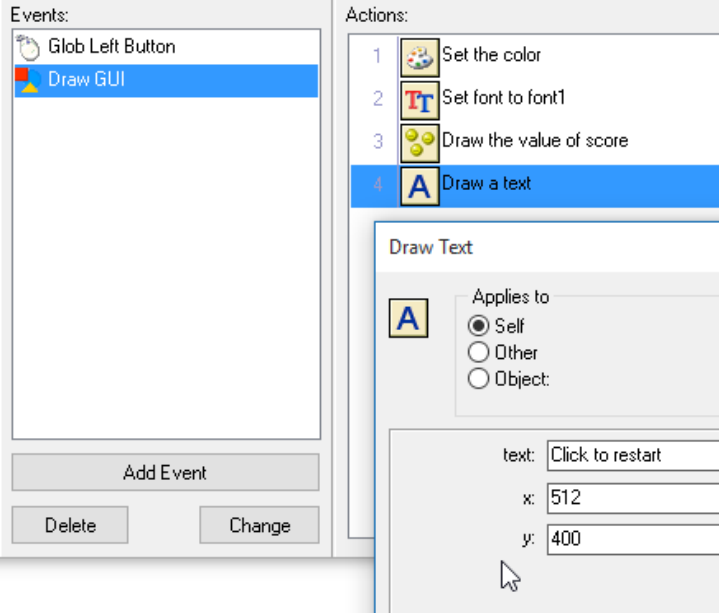
x:

y:

☐ Relative

Place a controller3 in the third room and make sure it spawns a UFO after 5 seconds.

13ENDING

| | |
|--|--|
| <p>Add an object_controllerEnd</p> <p>Don't set a parent for it</p> |   |
| <p>Add a Draw GUI event</p> <p>In it:</p> <ul style="list-style-type: none">• Set Color to White• Set Font to font1 and align center• Draw Score at position (512, 350) with caption Score:• Draw Text at position (512,400) text: Click to restart |  |
| <p>Test the final room to make sure it shows your score.</p> | |

14EXTRAS

Time to improve on the game.

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

Focus on adding new features (objects and behaviors) not on the looks of the game or just adding a bunch of new rooms using the same objects.

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

IDEAS:

You do not have to do all of these – you do not even have to do any of them if you have your own ideas. As you add features, try to think about how they affect the balance of the game. Making the space bar shoot bullets in all directions to destroy all the meteors might be fun to play with, but is going to destroy the challenge of the game.

- Moving beacons that must also be collected
- An emergency “warp” button for the player that teleports them to a random location
- When large meteors collide, they turn into multiple moving small ones
- A gun that creates bullets moving in the player’s direction (`object_player.direction`)
- A powerup that makes the player accelerate faster. One approach is to in player create, set variable `SpeedUp` to 0. Then in player collide with powerup, set variable `SpeedUp` to 1. In the spacebar event, test variable `SpeedUp` to see if it equals 1. If so, accelerate by 0.7 instead of the normal 0.5.
- A shield activated by the mouse that when active follows the player and destroys meteors. Use the technique we used with the crosshair in `ShootingGallery` but jump to `object_player.x` and `object_player.y` instead of the mouse’s position.

Also, watch out for the fact that even if the shield is bigger than the player, a meteor can collide with the shield and ship in the same step. You might need to use a **Check Mouse** action (control tab) in the Collision with Meteor to make it say something like “If the player is not holding down the left mouse button, then do all this stuff. Otherwise don’t do anything.”

- A black hole or other object that attracts the player. To control attraction, you should use **Set Gravity** in the player’s **Step**. To calculate the direction to a black hole you can use something like this for direction:
`point_direction(x, y, object_hole.x, object_hole.y)`

It calculates the angle from the point given by `x, y` to the point given by `object_hole’s x and y`. Unless every room has a hole, you should “protect” the Set Gravity with an instance count like we did with the UFO’s `Move Towards` so you do not try to calculate the direction to something that is not there.