

We will be creating a game called *Space Rocks!*, inspired by the classic 1980s arcade game *Asteroids*. In this game, the player controls a spaceship that turns left and right and using thrusters to move forward. The ship can shoot lasers which are used to destroy asteroids that you must avoid; colliding with an asteroid will destroy your spaceship!



For this project, we assume the reader has a basic background in Construct; in particular, the reader should be able to:

- change project/layout properties
- add sprites and behaviors and adjust their property values
- create events with given conditions and actions

Contents

| | |
|---|----|
| Step 0 – Setting up the project | 3 |
| Step 1 – Adding the Background sprite | 3 |
| Step 2 – Adding a Spaceship sprite..... | 3 |
| Step 3 – Adding custom keyboard controls | 4 |
| Step 4 – Adding Lasers | 8 |
| Step 5 – Adding Asteroids | 9 |
| Step 6 – Special Effect: Explosions | 12 |
| Step 7 – Special Effect: Rocket Thrusters | 13 |
| Step 8 – Special effect: Teleporting..... | 14 |
| Step 9 – Winning and Losing the Game | 16 |
| Step 10 – Review | 17 |
| Side Quest – Shields | 18 |
| Side Quest – Electromagnetic Pulse Item | 20 |
| Side Quest – Enemy UFO Spaceships | 22 |

Step 0 – Setting up the project

- Create a new project (new empty project)
- Click in the layout area, and change the layout size to 800, 600
- View the Project Properties, and change the window size to 800, 600.
You may also want to change the Name, Author, and ID properties.

Step 1 – Adding the Background sprite

- Right-click in the layout area and select *Insert New Object*
- Create a *Sprite* named **Background**
- In the image editor, use the file **spacey.jpg**
- Change the size and position of Background so that it covers the entire layout



Step 2 – Adding a Spaceship sprite

- Similar to the previous step, create a *Sprite* named **Spaceship** using the image file **spaceship.png**



Step 3 – Adding custom keyboard controls

Construct has many built-in behaviors for standard kinds of movement: 8-Direction, Platform, Car, etc. However, Asteroids has a unique control scheme that we will need to set up ourselves. We do this by enabling keyboard access (via the Keyboard object) and setting up some events.

- Right-click in the layout area and select *Insert New Object*
- Underneath the Input heading, select *Keyboard* (you don't need to re-name it)



Unlike when working with Sprites, nothing will appear in the layout area; instead you will see a brief message above the layout area that says the Keyboard object is available to the project.

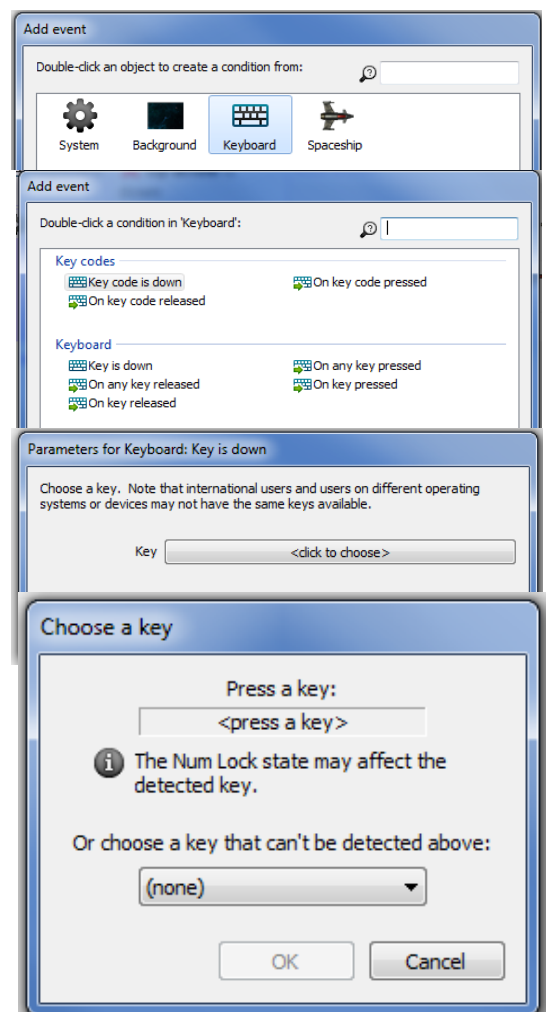
Next we'll add some events that allow us to move the spaceship using the keyboard. For each of these events, the conditions will involve the keyboard, and the action will involve the spaceship.

The first event will be:

“If the Left Arrow Key is held down, then turn the Spaceship counterclockwise 2 degrees.”

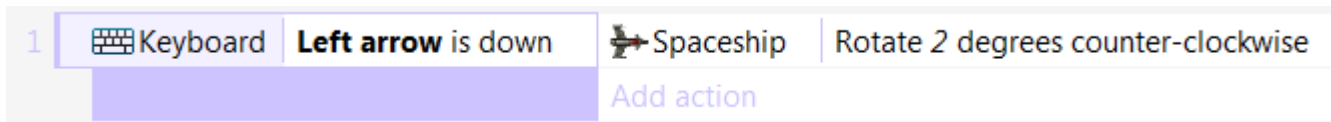
- Go to the event sheet and click “Add Event”
- Select the *Keyboard* object
- Select the condition “Key is Down”
- A window will appear; press the <click to choose> button
- Another window will appear, containing the message “Press a key:”. Press the **Left Arrow** key and the text “Left arrow” will appear. Then click on the *OK* button in this window, and the *Done* button in the other.

This completes the condition; next, we'll add the action.



- Next to the condition, click on “Add Action”
- Select the *Spaceship* object
- Underneath the *Angle* group, select the action “Rotate Counter-Clockwise”
- A window appears, asking you to enter the number of degrees; enter **2** and click the Done button.

The completed event should look like this:



Before we continue, there are two points that should be clarified and emphasized:

(1) There are two types of keyboard events that sound the same but are different:

- **Key Is Down:** as long as the key is being held down, this condition will be true, and the action will happen repeatedly; this is sometimes called a *continuous* event, and is used for actions such as movement.
- **On Key Pressed:** this condition will be true only when you first press the key, and will not be true again until you release the key and press it again; this is sometimes called a *discrete* event, and is used for actions such as jumping or shooting.

(2) Typically (whenever possible), games run at 60 FPS - frames per second. (Graphics-intensive or computation-intensive games may run at a slower rate.) This means that every event is checked and run, and every sprite is redrawn, 60 times per second. Therefore, in the event above, the unit of time measurement is *per frame*; 2 degrees per frame, and 60 frames per second, results in a total rotation of 120 degrees *per second* - that’s a full (360 degree) rotation every 3 seconds.

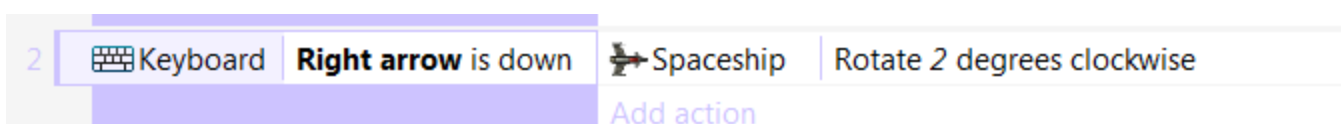
Now, let’s return to creating events.

The next event will be:

“If the Right Arrow Key is held down, then turn the Spaceship clockwise 2 degrees.”

It is almost identical to the previous event; the differences are underlined.

Follow the same process as before, making the necessary changes, and your event should look like:



Finally, we need an event that makes the spaceship move. There are a few ways to accomplish this.

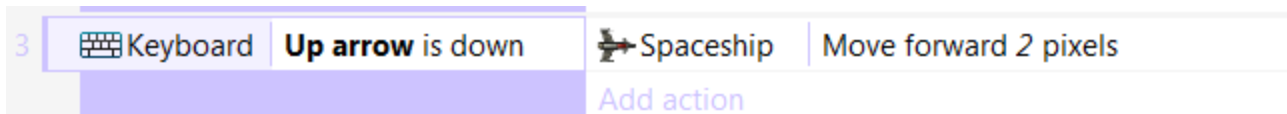
Option 1 (easy):

You could create the following event:

“If the Up Arrow Key is held down, then move the Spaceship forward 2 pixels.”

This causes the spaceship to move at a constant rate when (and only when) the button is down.

This can be accomplished similar to the previous events; the “Move Forward” action is located in the “Size & Position” group of actions in the Edit Action window. The completed event looks like:



This completes Option 1.

Option 2 (intermediate):

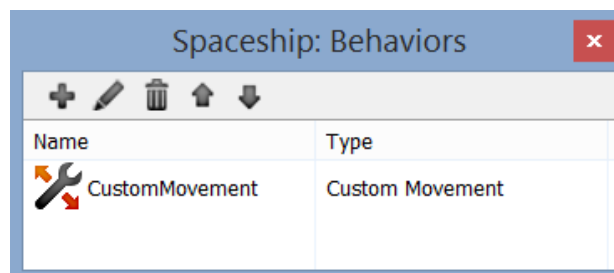
In the original Asteroids game, activating the rocket thrusters didn’t just *move* the spaceship, it caused it to *accelerate* - slowly increase its speed. Furthermore, when the button was released, the spaceship continued to drift at the same speed. This makes sense, since there are no opposing forces (such as friction) to slow down the spaceship. (The only way to reduce speed is to rotate the spaceship in the opposite direction and activate the thrusters to counteract the acceleration.)

To accomplish this, we can create the following event:

**“If the Up Arrow Key is held down,
then accelerate the Spaceship, at a rate of 100, in the direction of the Spaceship’s angle.”**

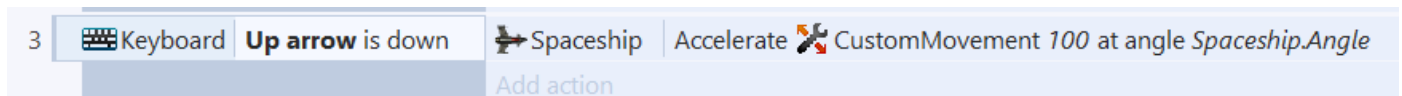
Acceleration is not a default property of a Sprite object, so to create this action, we will first need to add a behavior to the spaceship object.

- Select the spaceship object, and add the behavior *Custom Movement*



- In the event sheet, create a new event with the Keyboard condition “Up Arrow is down”.
- Add a Spaceship action; in the list of actions, in the group titled “CustomMovement: Velocity”, select the action “Accelerate toward angle”.
- A window will appear with textboxes where you can enter values.
 - Next to *Acceleration*, enter **100**. The unit of measurement is pixels per second, per second, which means that if you accelerate for 1 second, your velocity will increase by 100 pixels/second; if you continue to accelerate, your velocity will continue to increase.
 - Next to *Angle*, enter the text **Spaceship.Angle**. This refers to the direction (angle) the spaceship is currently facing; this is the direction we want to accelerate towards. (If the spaceship is already moving in a different direction, the forces will be combined and result in movement in an intermediate direction.)

The completed event looks like this:



This completes Option 2.

At this point, whether you have chosen Option 1 or Option 2 to move the spaceship, there is one more feature to add. In the original Asteroids game, when the spaceship flew past the right side of the screen, it reappeared on the left side, and vice versa; when it flew past the bottom edge, it reappeared on the top, and vice versa again. This is called “wraparound”, and can be accomplished in Construct by adding a behavior to objects which move in this way.

- Select the spaceship object, and add the behavior *Wrap*

Finally, you may want to add an event that caps the speed of the spaceship: perhaps the spaceship should only be able to travel at a maximum speed of 200/pixels per second. The event is:

“If the spaceship’s speed is greater than 200, set the spaceship’s speed to 200.”

- Create this event; the condition and action both belong to the Spaceship object, located in the list under *Custom Movement*. The final event should look like this:



Save the project, and test the keyboard controls for moving the spaceship; feel free to change the keys or values as desired.

Step 4 – Adding Lasers

In this step, we'll add a laser sprite that the spaceship can fire.

- Create a new object: a *Sprite*, named **laser**, using the image file **laser.png**.

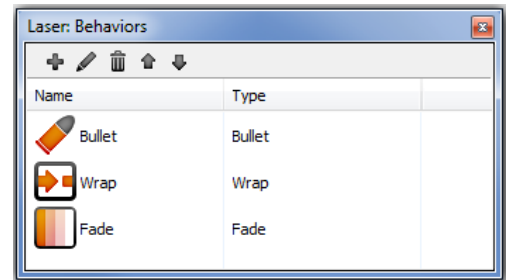


Adjust the size/proportions so that the laser is smaller than the spaceship.

- Add the following behaviors to the laser object:

- **Bullet**. This makes an object move in a straight line in whatever direction (angle) the object is facing. The default speed is 400 pixels/second; this can be changed in the properties panel.

- **Wrap**. We want the lasers to have wraparound capabilities, just like the spaceship.



- **Fade**. This behavior makes objects fade in or fade out, after an optional time delay, and can be set to automatically destroy objects after they have faded out. We want to use this behavior, because otherwise the lasers will last forever. In the properties panel, change the *Wait Time* to 1, change the *Fade Out Time* to 0.5, and leave *Destroy* set to **After Fade Out**.

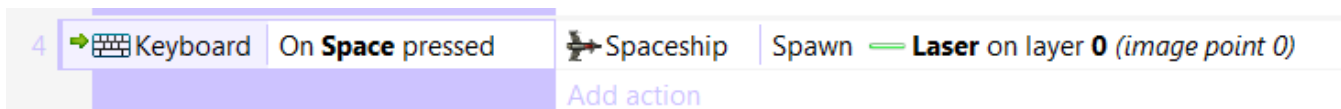
- In the event sheet, we'll add the event:

“If the Space key is pressed, then the Spaceship spawns (creates) a laser.”

As before, the condition is a *Keyboard* action; be sure to select the condition “On Key Pressed”. (If you select “Key Is Down”, then the spaceship will fire lasers continuously, which is entertaining to watch but a bit overpowered for this game.)

Also as before, the action is a *Spaceship* action. “Spawn” is a video game term meaning to create an object. The action “Spawn another object” is located in the *Misc.* group. Press the <click to choose> button and select the laser. When the object is spawned, by default it will be centered at the position and oriented at the same angle of the object spawning it.

The completed event looks like this:



Step 5 – Adding Asteroids

In this step, we'll add the space rocks (asteroids) to our project.

- Create a new object: a Sprite, named Asteroid.

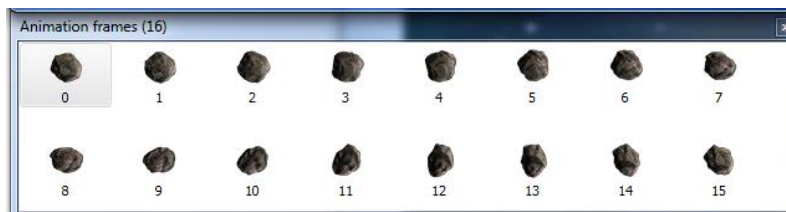
In this case, instead of a single image, we're going to use a sequence of images to create an asteroid that appears to be rotating in space.

- In the image editor, right-click on the Animation Frames window, and a menu will appear. We are going to add multiple frames (images used in an animation) at once, so select *Import Frames → From Sprite Strip...* . (A sprite strip is a single image that contains many animation frames for convenience. It is also possible to load frames from individual image files by selecting *From Files...*)

- Select the image file **asteroid-animation.png**. Notice that this file contains 16 images of an asteroid arranged in a 4-by-4 grid. When displayed in sequence, they look like a rotating 3D asteroid.



- A window titled *Import Sprite Strip* will appear, where we can enter information that tells the software how to divide the image. Enter 4 for the number of horizontal cells and also for the number of vertical cells. Also, check the box next to *Replace Existing Animation*; this will automatically delete the empty frame that is inserted by default, which we don't need. Then click the *OK* button, and the Animation Frames window will fill up with all the images for the individual frames, while the Edit Image window will show a picture of the currently selected frame.

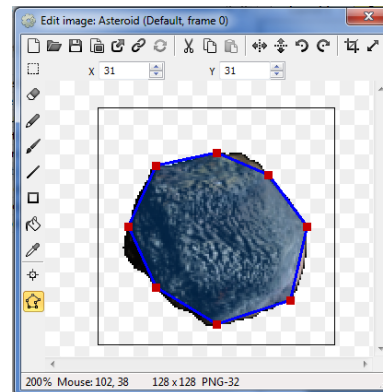
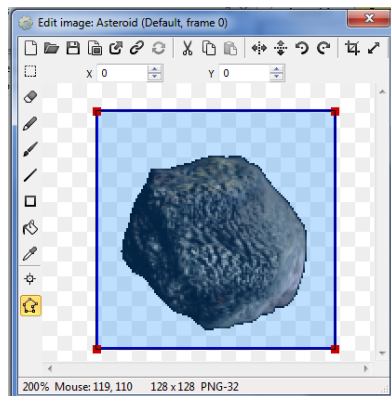


Because collisions will be very important in this game (lasers colliding with asteroids, or asteroids colliding the the spaceship), we need to adjust the Collision Polygon for the asteroids.

A Collision Polygon is the area of a sprite that result in collisions. (In general, images are not checked pixel-by-pixel because it requires too much time.) The Construct software estimates the collision polygon for you when you load a single image, and it usually doesn't need to be adjusted. When loading a sequence of images, however, this needs to be adjusted manually.

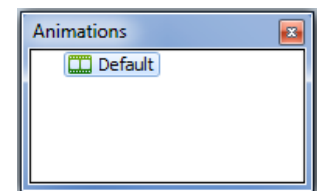
With Frame 0 selected in the *Animation Frames* window, go to the *Edit Image* window, and click the icon on the lower left to see that collision polygon for the current image. The little red boxes represent the corners, or *vertices*, of the collision polygon, and the blue lines represent the sides, or *edges*. You can right-click on a vertex to add additional vertices for greater precision, but adding too many will cause your game to lag.

You'll see that the current collision polygon is a square, which isn't very accurate. The easiest way to improve this is to right-click on the image, and in the pop-up menu, select *Guess Polygon Shape*. Since it would take a long time to repeat this process for every frame of the animation, right-click the image again and select *Apply to Whole Animation*.

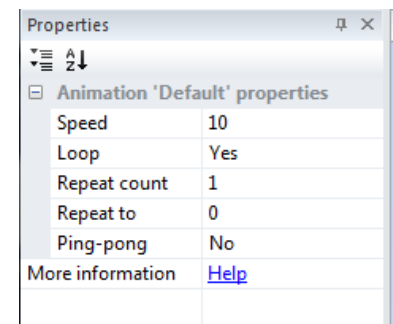


Finally, we need to adjust the properties of the animation.

- In the Animation List window, click on Default; the properties panel will then display the properties of this animation.
- *Speed* is the number of frames displayed per second.
Change the number to **10**.
- *Loop* controls whether the animation should repeat when finished;
Change this setting to **Yes**.

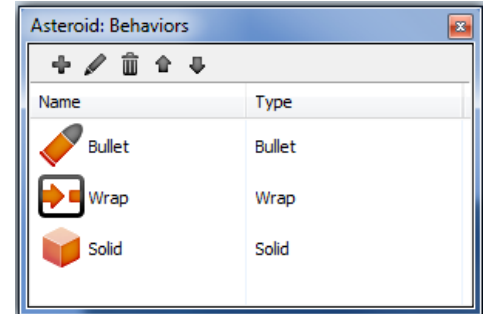


You can preview how the animation will look by right-clicking the name of the animation in the Animation List window, and select *Preview*. Adjust the settings as desired, then close the image editor windows.



Next, add the following behaviors to the asteroids:

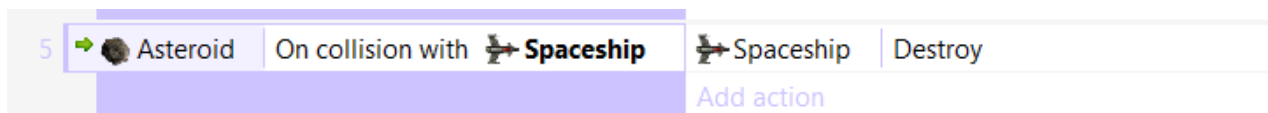
- *Bullet*. Similar to lasers, asteroids travel in a straight line. However, they move much more slowly than lasers, so in the properties panel, change the *Speed* to **100**. Also, we would like asteroids to bounce off each other, so set *Bounce Off Solids* to **Yes**.
- *Wrap*. Asteroids have wraparound movement like all other objects.
- *Solid*. The behavior will stop asteroids from appearing to pass “through” each other in the game.



Finally, we will add some events that govern how asteroids interact with other game sprites.

“If an asteroid collides with the spaceship, destroy the spaceship.”

The completed event looks like this:

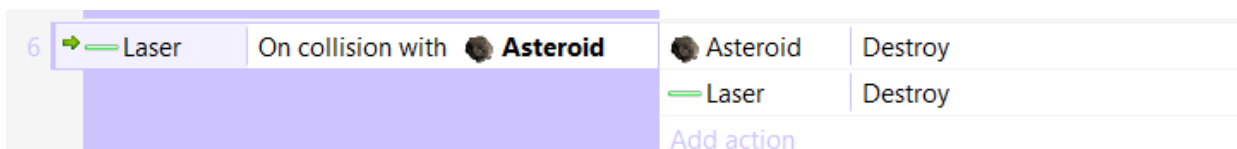


The next event to add is:

“If a laser collides with an asteroid, destroy the asteroid and destroy the laser.”

This event is slightly different because there are two actions that are performed when the condition is met. Clicking on “Add Action” underneath an action allows you to add more actions to a given event.

The completed event looks like this:



- On the layout, make a few duplicates of the asteroid object, but change their sizes and rotate them so that they look slightly different and start moving in different directions.

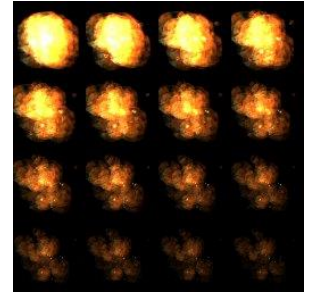
Save your game and check out your new space rocks!

Step 6 – Special Effect: Explosions

In this and the next few steps, we will add some special effects to the game.

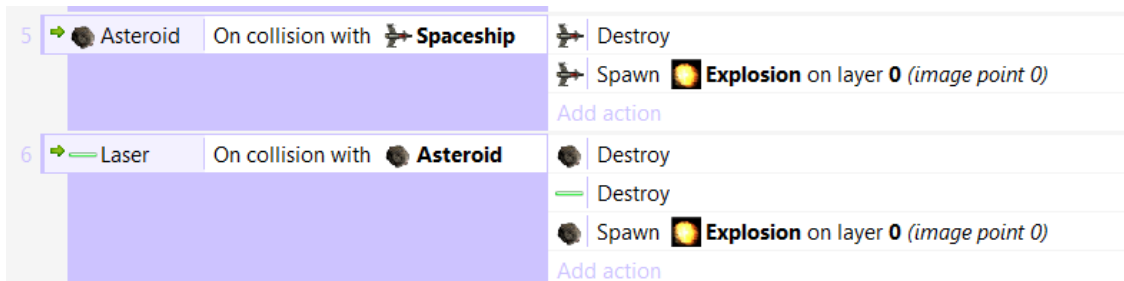
First, we'll create an animated explosion effect that appears when either an asteroid or the ship is destroyed.

- Add a new sprite, named **Explosion**. Similar to the process used in the previous step, when the image editor windows appear, right-click on the *Animation Frames* window, select *Import Frames* → *From Sprite Strip...* and select the image file **explosion.jpg**, which contains 16 images in a 4-by-4 grid. Click on *Default* in the *Animations* list window; change the *Speed* to **32**. (Leave *Loop* set at **No**, because explosions don't repeat.) Close the image editor windows.



- Select the explosion object, and in the properties panel, change *Blend Mode* to **Additive**. This is a useful setting for special effects based on light; instead of drawing the explosion on top of other sprites (also known as *Normal blending* - which also obscures the background objects), additive blending will combine the colors of the explosion with the sprites behind it. (It is a subtle difference from transparency, which combines the colors using a different formula - additive blending adds the RGB color values, while transparency averages these values.)

- Next, we add some actions that make explosions appear when certain sprites are destroyed; the sprite which is destroyed (asteroid or spaceship) will spawn an explosion sprite. The changed events will look like this:



- Also, when the explosion is finished, we would like to destroy (remove) the explosion from the layout. In the event sheet, add a new event; select the Explosion object, and the condition "On any animation finished". (There is a related condition called "On animation finished", but that requires you to type in the name of a specific animation, which is really only important for sprites that have multiple animations, so we won't use that here.) Then add the action Explosion - destroy.

The completed event will look like this:

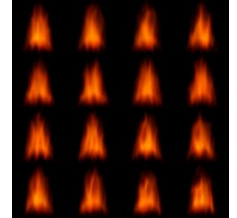


Save your project and test the new features!

Step 7 – Special Effect: Rocket Thrusters

Next, we'll add some animated rocket thrusters that appear whenever the spaceship is moving forward. This gives the player some visual feedback, which increased the quality of the gameplay experience.

- Add a new sprite, named **Flames**. As before, in the image editor, use the *Animation Frames* window to import frames from a sprite strip called **flame.jpg**; there are 16 images in a 4-by-4 grid. In the Animation properties panel, change the *Speed* to **16**, and change *Loop* set to **Yes**. Close the image editor windows. In the properties panel, change *Blend Mode* to **additive**.



- In the layout area, change the size, position, and rotation of the flames object so that it appears to be coming out of the spaceship, as illustrated in the picture to the right.



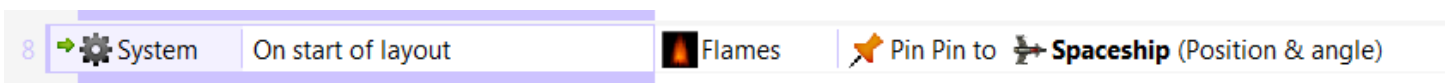
- Next, we'd like the flames to move and turn together with the spaceship; this can be accomplished using a behavior called Pin. Add this behavior to the flames object.

The Pin behavior can only be activated using an event. Furthermore, it only needs to have once, when the game first starts. We'll add the following event:

“When the layout first starts, the Flames object will pin itself to the Spaceship.”

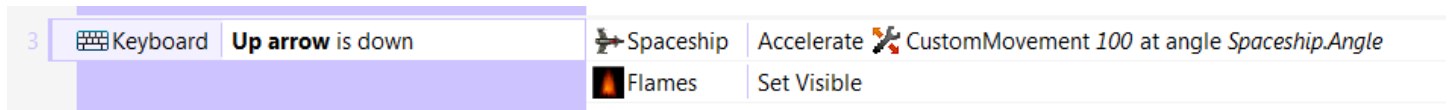
- Create a new event. The condition can be found under the System object, and it is called “On start of layout”. Add the action from the Flames object, select Pin To Object from the list; in the following window that appears, press <click to choose> and select the Spaceship object, and leave the mode set on Position & Angle.

The completed event will look like this:

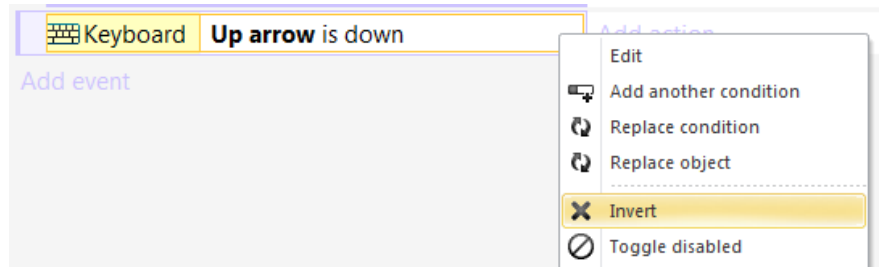


Next we'd like to add some events/actions that control when the flame special effect is visible. In particular, we would like the flames to be *visible* when the ship is accelerating (which happens when the Up Arrow key *is* held down), and we would like the flames to be *invisible* otherwise (when the Up Arrow key *is not* being held down).

The first of these goals can be achieved by adding another action to a preexisting event; the action belongs to the Flames object: *Set Visibility to Visible*. The modified event looks like this:

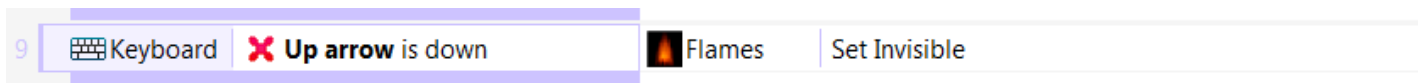


To make the flames invisible at the appropriate time, we need another event whose condition is when a key is NOT held down. You may have noticed there are no Keyboard conditions that refer to a key not being held down. The way that “negative” conditions are created in the Construct Game Engine is by creating the condition without the “not” part, and then in the event sheet, right-click on the condition, and then select “Invert” from the menu that appears. When you are finished, a red “X” appears in the condition, indicating that the event will occur only when the condition is false.



- Add the Flames object action: *Set Visibility to Invisible*.

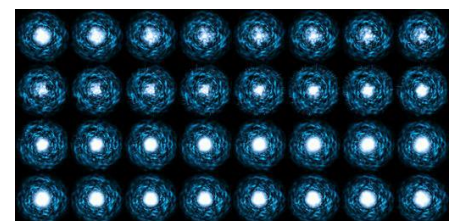
The completed event should look like this:



Step 8 – Special effect: Teleporting

Another feature that the original asteroids game had was teleporting. The idea is that, if a collision is about to happen and you can’t avoid it, you could press a button that would teleport you to a random place on the screen. However, there was always a change that you might teleport on top of an asteroid, so this power should be used sparingly. We will add this feature to our project, along with a wormhole-like animated effect that appears at our origin and our destination whenever we use this power.

- Add a new sprite, named **Warp**. As before, in the image editor, use the *Animation Frames* window to import frames from a sprite strip called **warp.jpg**; this time. The images are in a **8-by-4** grid. In the Animation properties panel, change the *Speed* to **16**, and change *Loop* set to **Yes**. Close the image editor windows.



- In the properties panel, change *Blend Mode* to **additive**.
- Change the size of the warp sprite so it is approximately the size of the spaceship sprite.
- Position the warp sprite outside of the layout.
- Add the behavior *Fade*; you can keep the default property settings and adjust later if desired.

Next, we will add the following event

“If the X Key is pressed, then create a warp sprite at the spaceship position, and then move the spaceship to a random position on the layout, and then create a warp sprite at this new position.”

- Create a new event, with the Keyboard condition “On Key Pressed” and select the “X” Key.
- Add the Spaceship action “Spawn Another Object”, and select the Warp object.
- Add another Spaceship action; in the action list, underneath the heading “Size & Position”, select “Set Position”

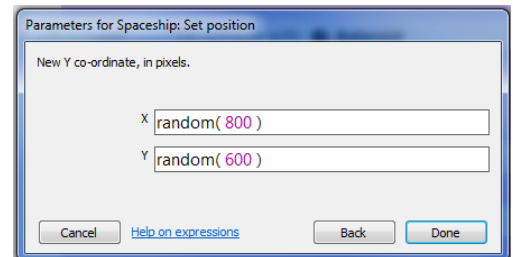
A window will appear where you can enter the new X and Y coordinates for the ship.

You can enter more than just a number in these boxes - you can enter any *expression*: either a number, or an arithmetic expression (such as **200 * 2 + 50**), or even a function. Unlike functions in your typical math class, that often have generic single-letter names like “*f*”, programming functions usually have descriptive names. For example, to find the absolute value of a number, there is a function called **abs**; you could enter **abs(-3)** and when the program starts, the computer would calculate this to be 3.

In this situation, we are going to use a function that outputs a random number. The name of this function, appropriately enough, is **random**. The **random** function takes one input: the largest random number that could be output. So, for example, to get a random number between 0 and 12, you could enter **random(12)**, and every time you’ll get a different output in this range.

Since we want the spaceship to appear at a random position on the screen, we want the computer to select a random number for the X (horizontal) coordinate, and a random number for the Y (vertical) coordinate.

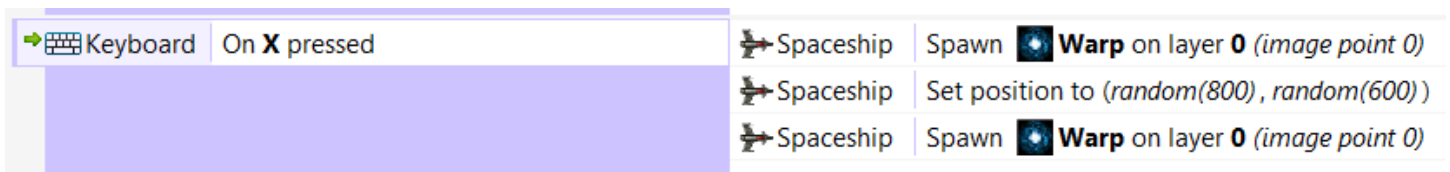
- The width of the screen is 800 pixels, so in the textbox next to X, enter **random(800)**
- The height of the screen is 600 pixels, so in the textbox next to Y, enter **random(600)**



Next, we want to create another warp sprite at this new position.

- Add another Spaceship action to spawn a warp object.

The completed event will look like this:



Step 9 – Winning and Losing the Game

Now we will add objects and events that show when the player has won the game (when there are no asteroids left) or when the player has lost the game (when the spaceship has been destroyed, or equivalently, when there are no spaceships left).

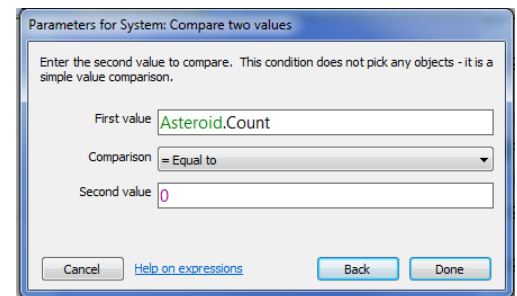
- Add a new sprite, called **TextWin**, and use the image **text-win.png**.
- Add a new sprite, called **TextLose**, and use the image **text-lose.png**.
- Position both the sprites so that they are centered on the screen.
- In the properties panel, set the *Initial Visibility* of both sprites to **Invisible**.
- Add the *Fade* behavior to each sprite, and change the fade properties as follows:
Active At Start: No. Fade In Time: 1. Fade Out Time: 0. Destroy After Fade Out: No.

Next we add an event for winning:

“If the number of asteroids is zero, then display the TextWin image.”

The condition is actually a *System* condition, called *Compare Two Values*. After selecting this condition, in the parameters window that appears, enter the following information:

- Next to “First value”, enter **Asteroid.Count**
- Leave the “Comparison” set as **= Equal to**
- Next to “Second value”, enter **0**



The action has two parts:

- the TextWin object needs to *Set Visibility to Visible*
- the TextWin object also needs to **Start Fade**

When the event is complete, it should look like this:



Similarly, we need to add an event for losing:

“If the number of spaceships is zero, then display the TextLose image.”

The steps are very similar to the previous event; in the condition, instead of **Asteroid.Count** you will use **Spaceship.Count**, and the actions are for the TextLose object.

The completed event looks like this:



Step 10 – Review

After this project, you should be familiar with:

- the Keyboard object
- the Behaviors: Bullet, Wrap, Fade, Pin, Custom Movement
- creating sprite animations
- using multiple actions in an event
- inverting a condition in an event
- using the random() function

If you're ready for an extra challenge, read on and try some of the *Space Rocks!* side quests.

Side Quests

(The following steps are optional and can be completed in any order.)

Side Quest – Shields

In this step, we'll add shields to the spaceship, which allow the ship to be hit a few times before it explodes, which serves to balance the gameplay.

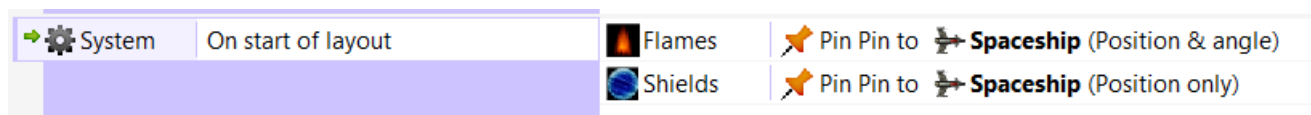
- Add a new sprite, named **Shields**, that uses the image **blue-sphere.jpg**
- Set the *Blend Mode* to **additive**.

Next, we'll add some subtle value-based animation effects to this object, to make the shields appear to be slowly rotating and pulsing.

- Add the behavior *Rotate*, and set the *Speed* to **5**.
- Add the behavior *Sine*, change *Movement* to **Size**, *Period* to **2**, and *Magnitude* to **5**.

We would like the shields to move together with the ship.

- Position the Shields object so that its center is aligned with the Spaceship object's center.
- Resize the Shields object so that it is slightly larger than the Spaceship.
- Add the behavior *Pin* to the Shields object.
- In the event with condition "On Start of Layout", add the Shields action Pin to Spaceship (Position Only).



Now, most importantly, we would like to set up how the shield interacts with the asteroids.

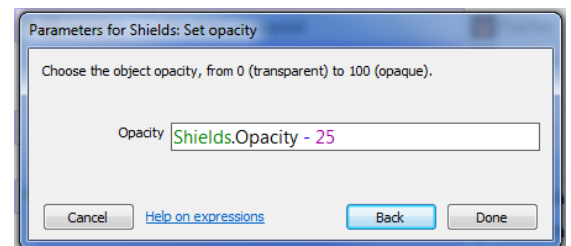
When there is a collision, we would like asteroids to bounce off of the shields, we can accomplish this with the Solid behavior, since asteroids are already configured to bounce off of solids. We also would like a collision to stop the ship from moving.

We can accomplish this with the following event:



We would also like some way to “damage” the shields; we need to keep track of the amount of damage and we’d also like to change the appearance of the shields in response. An easy way to handle both is by changing the opacity (transparency) setting; each collision decrease the opacity, and when it reaches zero, the shields are destroyed. We accomplish this in two parts:

- Add an action to the previous event that sets the new value of the shield’s opacity (`Shields.Opacity`) to the current value of the shield’s opacity, minus 25. To do this, select the “Set Opacity” action from the list, and enter **`Shields.Opacity - 25`**.



This means that the shields can withstand 4 collisions.

We also need to destroy the shields once their opacity reaches zero (otherwise, they will still be present, deflecting asteroids, while invisible). There are conditions that allow you to compare the values of a property for a single instance; *Compare Opacity* (listed under the Shield’s *Appearance* conditions) is one of them. Create the following event:

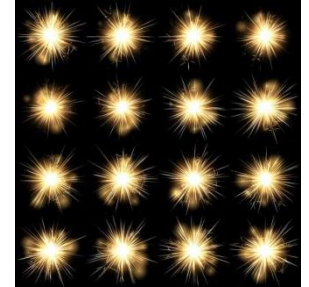


Your spaceship now has fully functioning shields!

Side Quest – Electromagnetic Pulse Item

Another feature that makes gameplay more interesting is the ability to collect and use special items. In this step, we'll add an item that appears occasionally, at random, when an asteroid is destroyed. When the item is collected (by the spaceship making contact with it), a large EMP (electromagnetic pulse) effect will appear for a few seconds, destroying any asteroid it comes in contact with.

- Add a new sprite, named **EMP_Effect**. In the animation frames window, use the spritesheet **spark-large.jpg**, which is a 4-by-4 grid of images. In the animation properties, change *Speed* to **24** and *Loop* to **Yes**.



- Change the *Size* to **300, 300**.

- Add the behavior *Fade*; change *Wait Time* to **1**.

- Add a new sprite, named **EMP_Item**. In the animation frames window, use the spritesheet **swirl-yellow.jpg**, which is part of a 8-by-4 grid of images. After the frames are loaded, delete the empty (black) frames. Change the animation *Speed* to **16** and *Loop* to **Yes**.

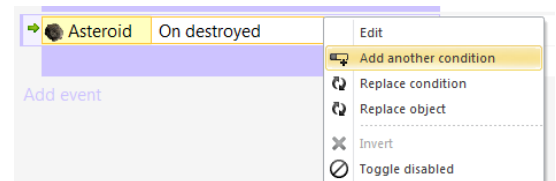
- For both sprites, change *Blend Mode* to **additive**.

- Move both sprites into the margins of the layout window (since they should not be visible at first).

Now we add some events that control the appearance and interaction with these items.

First: When an asteroid is destroyed, there should be a 20% chance of an EMP_Item spawning from the asteroid. Create an event with the condition Asteroid - *On Destroyed*. The “20% chance” can be created by adding a second condition: using the random function, create a random number between 0 and 100, and compare it to 20; if it is less than 20 (which will happen 20% of the time), then perform the action.

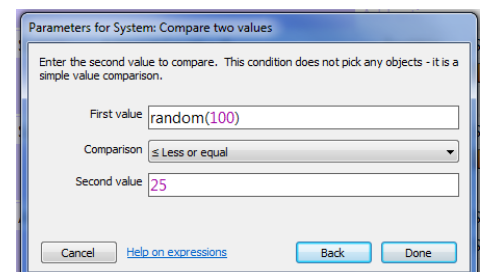
To add another condition to an event, right-click the condition that already exists, and from the menu that appears, select “Add Another Condition”.



When an event has two conditions, there is an implicit “and” between them; the associated actions will only happen when **both** of the conditions are true at the same time.





The second condition to add is the *System* condition **Compare Two Values**.

- For *First Value*, enter **random(100)**
- Change *Comparison* to **Less Or Equal**
- For *Second Value*, enter **20**








- Add the action *Asteroid - Spawn Another Object* (EMP_Item).






The completed event will look like this:

| | | | |
|--|------------------|--|--|
|  Asteroid | On destroyed |  Asteroid | Spawn  EMP_Item on layer 0 (<i>image point 0</i>) |
|  System | random(100) ≤ 25 | Add action | |

If the spaceship collides with the EMP_Item, then the EMP_Item should be destroyed and an EMP_Effect should be spawned at that position. Create this event; it should look like this:

| | | | |
|---|---|--|--|
|  Spaceship | On collision with  EMP_Item |  EMP_Item | Spawn  EMP_Effect on layer 0 (<i>image point 0</i>) |
| | |  EMP_Item | Destroy |

And finally, when the EMP_Effect overlaps with an asteroid, the asteroid should be destroyed. The completed event should look like this:

| | | | |
|--|--|---|---|
|  Asteroid | Is overlapping  EMP_Effect |  Asteroid | Spawn  Explosion on layer 0 (<i>image point 0</i>) |
| | |  Asteroid | Destroy |

This completes the EMP side quest. Save and test your project, and feel free to experiment!

Side Quest – Enemy UFO Spaceships

In this final side quest, we're going to add a (passive) enemy UFO spaceship.

When adding a new character to a game, there are many considerations to ponder:

- Where and when does the character appear?
- How does the character move?
- What are the character's abilities?
- How does the character interact with every other object in the game?
- How could the character affect the win/lose conditions?
- What happens to the character when the game is over?

Dealing with each of these considerations will make this side quest one of the longer ones.

First: the UFO will be spawned periodically at random positions to the left of the screen, and travel across the screen from the left to the right, moving up and down along the way (like a sine wave).

- Add a new sprite, called **UFO_Spawn**, that uses the image **UFO_Spawn.png**.
Position this sprite to the left of the layout.

- Add a new sprite, called **UFO**. Use the animation frames window to load the spritesheet **ufo.png**, which is a 12-by-1 grid. Change the animation *Speed* to **16** and *Loop* to **Yes**.




Next, we'll add an event that moves the spawn point to a random vertical (Y) location at regular intervals. We use the System condition *Every X Seconds*, and enter a number of seconds (like **10**).

Then add two actions:

- the UFO_Spawn action *Set Y*, and enter the expression **random(400) + 100**. This expression will produce a random number in the range from 100 to 500.
- the UFO_Spawn action *Spawn Another Object*; select **UFO**

The finished event should look like this:

| | | | |
|--------|-------------------------|-----------|--|
| System | Every 10 seconds | UFO_Spawn | Set Y to <i>random(400) + 100</i> |
| | | UFO_Spawn | Spawn  UFO on layer 0 (image point 0) |


















Next, we'll add some behaviors for UFO movement:

- Add the behavior *Bullet* to the UFO; set *Speed* to **125**.
- Add the behavior *Sine* to the UFO; set *Movement* to **Vertical**, *Period* to **1**, and *Magnitude* to **20**.

We also want the UFO to be destroyed if it passes beyond the right side of the screen, which happens when the UFO's X coordinate is greater than 800 (say, 900, because we want to make sure the entire sprite image, not just its center point, is off-screen). There is a UFO condition that allows us to compare its coordinate, *Compare X*. The finished event looks like this.











| | | | |
|--|---------|--|---------|
| <div>  UFO </div> | X > 900 | <div>  UFO </div> | Destroy |
|--|---------|--|---------|

Next, we need to figure out how the UFO interacts will the other objects. We'll assume that, like a spaceship, it is destroyed if it collides with an asteroid or a laser, and if the spaceship and asteroid collide, both are destroyed. And of course, we add explosions when most things are destroyed. The corresponding events are below:

| | | | |
|---|---|--|---|
| <div>  Asteroid </div> | On collision with <div> UFO</div> | <div>  UFO </div> | Destroy |
| | | <div>  UFO </div> | Spawn  Explosion on layer 0 (<i>image point 0</i>) |
| | | Add action | |
| <div>  Laser </div> | On collision with <div> UFO</div> | <div>  Laser </div> | Destroy |
| | | <div>  UFO </div> | Destroy |
| | | <div>  UFO </div> | Spawn  Explosion on layer 0 (<i>image point 0</i>) |
| | | Add action | |
| <div>  Spaceship </div> | On collision with <div> UFO</div> | <div>  Spaceship </div> | Destroy |
| | | <div>  Spaceship </div> | Spawn  Explosion on layer 0 (<i>image point 0</i>) |
| | | <div>  UFO </div> | Destroy |
| | | <div>  UFO </div> | Spawn  Explosion on layer 0 (<i>image point 0</i>) |
| | | Add action | |

- Of course, each of these could be subject to change; for example:
- A UFO colliding with an asteroid could destroy the asteroid also.
 - You could create a different animated explosion that is spawned by the UFO.
 - The UFO doesn't have to be destroyed when it collides with the player's spaceship.

If you have completed either of the other side quests for this project, you should also decide how the UFO interacts with the Shields and the EMP_Effect. For instance, you could create the events:

| | | | |
|---|--|--|---|
| <div>  Shields </div> | On collision with <div> UFO</div> | <div>  Shields </div> | Set opacity to <i>Shields.Opacity</i> - 25 |
| | | <div>  UFO </div> | Destroy |
| | | <div>  UFO </div> | Spawn  Explosion on layer 0 (<i>image point 0</i>) |
| | | Add action | |
| <div>  EMP_Effect </div> | On collision with <div> UFO</div> | <div>  UFO </div> | Destroy |
| | | <div>  UFO </div> | Spawn  Explosion on layer 0 (<i>image point 0</i>) |

Once again, you should feel free to change these events as you wish. For example:

- Perhaps the UFO isn't blocked by the Shields, or isn't affected by the EMP_Effect.
- The UFO could do a different amount of damage to the Shields, or destroy them entirely.
- Maybe the EMP_Item is always spawned when UFOs are destroyed by lasers, instead of being spawned by asteroids.

Finally, we need to consider how the UFOs affect (if at all) the game's ending. For example, perhaps when the game ends, we should stop spawning new UFO objects; this can be accomplished by destroying the UFO_Spawn object.















There is also a very, very subtle problem with our game logic that could occur. With our current set of events, it is possible to win and lose the game at the same time, causing both of the corresponding messages to appear. Can you see how this is possible?

The problem is that after the asteroids are all destroyed, there might also be a UFO remaining on the screen that causes the spaceship to be destroyed afterwards. There are multiple ways to fix this problem. We'll add a Fade behavior to the UFO that is triggered at the end of the game, and add a second condition to the spaceship-UFO collision event, which says the objects are destroyed only if the ship has not started to fade yet. The status of a fade effect can be checked by comparing the opacity of the object; if the opacity still 100, the object has not yet begun to fade.

All of these changes can be implemented by adding extra actions and conditions to the events that already exist.

- Add the behavior *Fade* to the UFO object; change *Active At Start* to *No*.

- Add the actions and *UFO - Start Fade* to the end of game events:

| | | | |
|--|---------------------|---|--|
|  System | Asteroid.Count = 0 |  TextWin | Set Visible |
| | |  TextWin |  Fade: start fade |
| | |  UFO_Spawn | Destroy |
| | |  UFO |  Fade: start fade |
| Add action | | | |
|  System | Spaceship.Count = 0 |  TextLose | Set Visible |
| | |  TextLose |  Fade: start fade |
| | |  UFO_Spawn | Destroy |
| | |  UFO |  Fade: start fade |

- Add the condition UFO - Compare Opacity (equal to 100) to the spaceship-UFO collision event:

| | | |
|-------------|-----------------------------|--|
| → Spaceship | On collision with Spaceship | Destroy |
| UFO | Opacity = 100 | Spawn Explosion on layer 0 (image point 0) |
| | | Destroy |
| | | Spawn Explosion on layer 0 (image point 0) |

Save and test your project, and feel free to experiment with different values and events!