

# BEGINNING JAVASCRIPT



# WELCOME TO BEGINNING JAVASCRIPT

In this course, ninjas will learn real JavaScript starting on day 1. They will progress through creating numerous JavaScript applications that can run in a browser or even on a server using node.js!

This course is not strictly game-based, so ninjas who take this course should be looking for a deeper understanding of real coding.

## FOREWORD

So you're getting ready for a Beginning Javascript camp, and you need to know what to do next! What you are about to read is a guide that will help you prep and plan for a full week of fun with your Code Senseis® and ninjas. Before we get started, please keep in mind that you can feel free to use it in its entirety, in pieces, or even not at all. This guide is here to give you a few tips and tricks based on what works for certain centers. In the end, you are going to know your center best. Want to change the times around? Sure! Want to add an early pick up? Yeah! Want to scrap the whole thing and do whatever you want? Be sure to get it approved by corporate first, but that's cool too!

Also, we are always here to answer any questions you might have based on this camp or just running camps in general. Please email [askhq@codeninjas.com](mailto:askhq@codeninjas.com) with any of your comments/questions/concerns!

# GENERAL SCHEDULE

TIME	ACTIVITY	DESCRIPTION
7:45 AM to 8:00 AM	Drop Offs	Parents drop kids off and sign check-in sheet.
8:00 AM to 8:15 AM	Ice Breakers	See suggested ice breaker list.
8:15 AM to 8:45 AM	Daily Kick Off	Describe what's going to happen today and get the kids excited for having fun! The dojo has rules...review. On day 1, introduce the senseis.
9:00 AM to 10:00 AM	Beginning JavaScript Lessons	Every day will have new JavaScript skills to learn.
10:00 AM to 10:30 AM	Brain Breakers	Select a STEM game from recommended list: rotate in groups of 10-15 based on the size of the camp.
10:30 AM to 12:00 PM	Beginning JavaScript Lessons Continued	Continue the day's lessons on JavaScript techniques.
12:00 PM	Half-Day Pickup	Sensei coordination: move students that are on half-day into the lobby or alternate dojo and have them wait for their parents to pick them up .
12:00 to 12:45 PM	Lunch	Lunch is in the dojo for full day students. All computers and equipment is put away. Drinks are limited to designated areas only.
12:45 PM to 1:00 PM	Lunch Clean Up	Students throw away trash and wipe down the area.
1:00 PM to 2:30 PM	STEM Enrichment	Students pick from any STEM based game or activity, including building their own games.
2:45 PM to 4:00 PM	Game Time!	Students can play Minecraft, Xbox, or build their own games!

# BEGINNING JAVASCRIPT DAY ONE

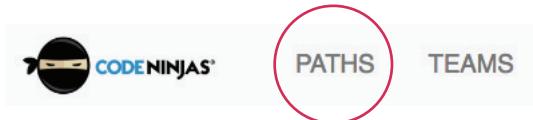
Over the next few days, you are going to learn the basics of making games using Javascript. Javascript is easy to learn and it is the primary programming language used for almost all of the websites around.

To make things convenient, we're going to be using Javascript in our GAME DEVELOPMENT PLATFORM (GDP). This platform is designed to simplify the process of creating and manipulating objects for our games so we can focus on building the games themselves. The next few pages explain how the Game Development Platform is used. Review them and then we'll start learning how to make a game in Javascript.



# CREATING A NEW PATH

- 1 Click on Paths in the Main Menu at the top of the page.



- 2 Create A New Private Path, if you don't already have one.

**CREATE A PRIVATE PATH**

- 3 Give your path a name like "My Basic Javascript".

NAME

- 4 Give your path a Description like "Games that I built at Code Ninjas".

DESCRIPTION

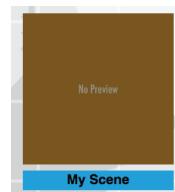
- 5 Click Create Path

**Create Path**

# CREATING A NEW SCENE

Every Path can contain one or more Scenes. Each Scene is a separate game that works by itself. Later, you will learn how to make multiple Scenes work together. For now, follow these instructions to create a new Scene.

- 1 Click the Path to add a new Scene from the paths screen.



- 2 Click "+ Scene"



- 3 Name Your Scene



HINT: Name it the same as the game you are building.

NAME	<input type="text" value="Your Game Name"/>
------	---

- 4 Describe Your Scene



HINT: You can say anything you want here.

DESCRIPTION	<input type="text"/>
-------------	----------------------

- 5 Click on "code"



HINT: All of the activities during this week will use code.

<input type="radio"/> BLOCKS
<input checked="" type="radio"/> CODE

- 6 Click Create Scene

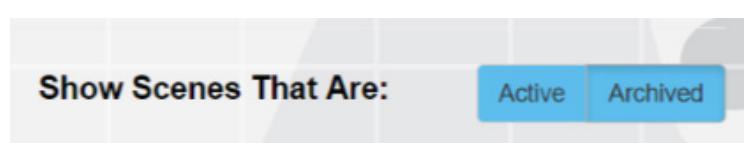
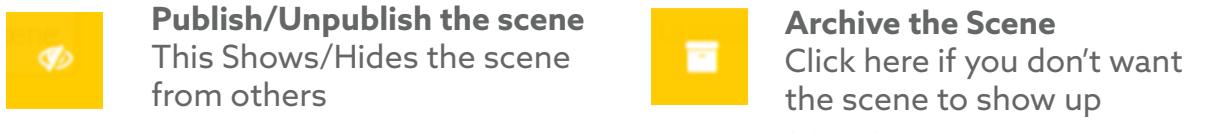


# EXISTING SCENES

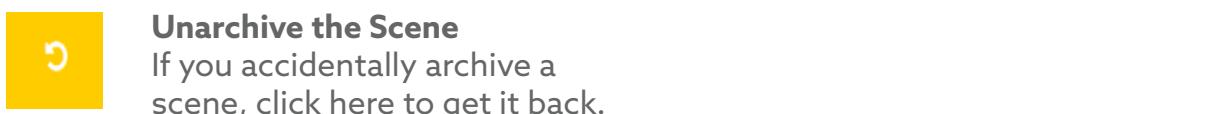


First, Click the Path that contains the scenes

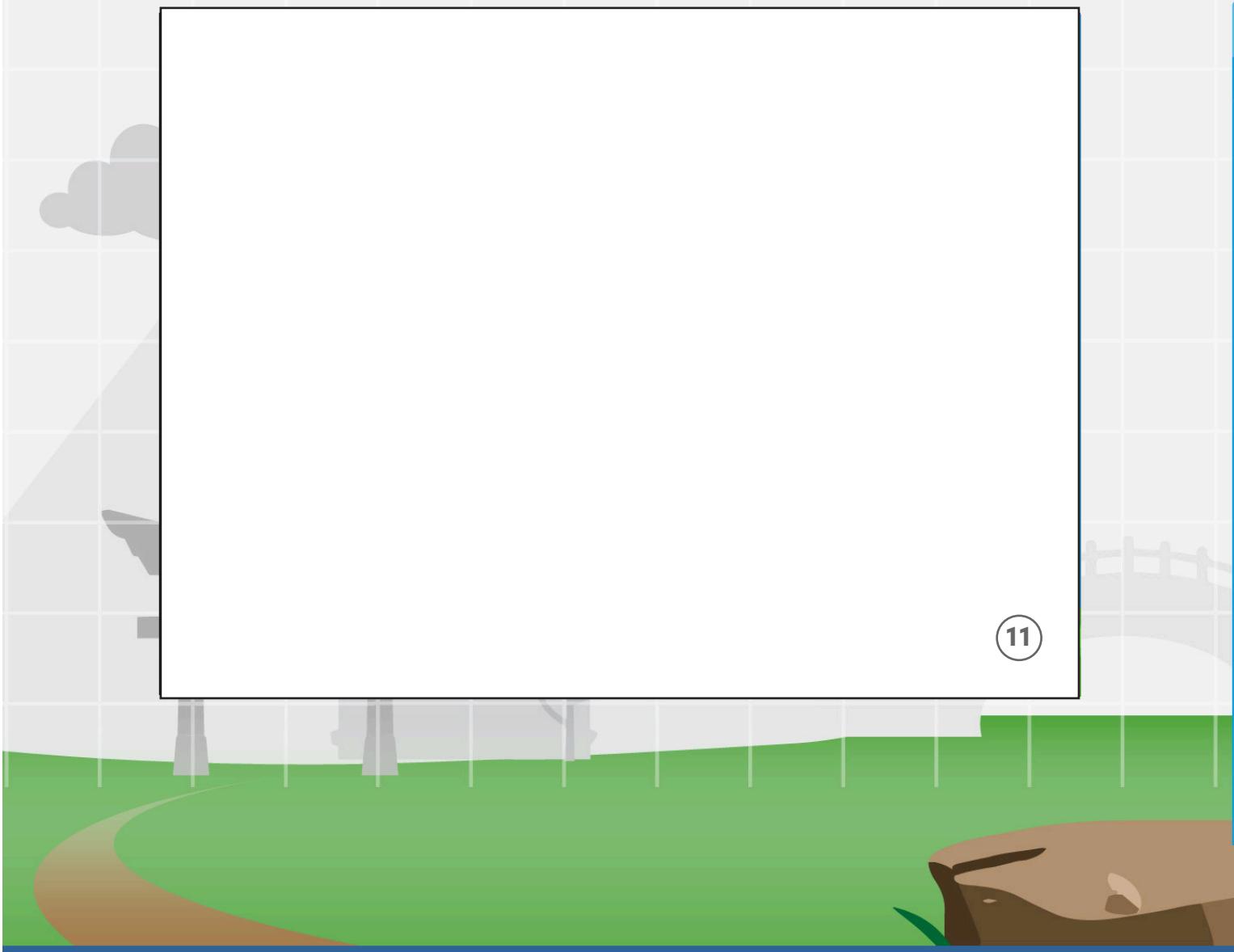
**Locate the Scene you want in the list of scenes**



If you want to see scenes that you have previously archived, click the Archived filter at the bottom of the scene list



## SCENE ONE



- 1 - Edit the selected Game Object's properties in the Properties Panel.
- 2- Always remember to Save Your Work!
- 3- Erase your progress and start over
- 4- Play the scene in a new window
- 5- Make a copy o f the scene
- 6- Edit Code for Selected Object

2

SAVE

3

CANCEL

4

PLAY

5

COPY

1

Properties

6

Events

General Info (UWPjoqzuOV)

NAME

Scene One

ROLE

DESCRIPTION

Scene One...

WIDTH

800

HEIGHT

600

SCALE X

1

SCALE Y

1

OPACITY: 1

 PLAYER CAN EDIT CODE PLAYER CAN EDIT PROPERTIES PLAYER CAN SEE IN TREE PLAYER EVENTS BUBBLE TO PARENT

7 -In this popup, you can select the different Game Objects that are in a Scene and see their properties and code on the right side panel.

8- Add Items to the Scene or to the selected Group

9- Runs the game in the editor to save time

10- Asset Search allows you to find assets that have been used in the past.

11- The Game Stage shows what your game looks like

## PROPERTIES PANEL



-  Breaks the connection to the asset so that it isn't updated automatically. You can't see this icon unless the selected Game Object is created from an asset.
-  Downloads the latest updates to the asset. You can't see this icon unless the selected Game Object is created from an asset.
-  Save the selected Game Object as an asset so that others can use it later.
-  Play the Animation on the selected Game Object. This is only visible for Sprites.
-  Change the visibility of the selected Game Object. You can always change the visibility in code as well.
-  Locked Game Objects are not draggable in the editor or by the player.
-  Removes the selected Game Object from the scene. You cannot undo this once it is done, so make sure this is what you want to do.

# SCENE EVENTS DESCRIPTIONS

Properties	Events
Intro to Game Development Platform	<b>Game Object Events</b>
1	Update Every Frame
	Initialize When Scene Starts
	<b>Mouse Events</b>

## Update Every Frame

This event will run every frame and allows the user to put code in the object's update cycle. This event only runs when the scene state is PLAY.

## Initialize When Scene Starts

This event runs only once, at the moment a game is started. This is the best place for declaring variables and other information that only needs to be acted on once.

# MOUSE EVENT DESCRIPTIONS

## Mouse Events

- Mouse Button Up
- Mouse Button Down
- Mouse Wheel
- Mouse Move
- Mouse Over
- Mouse Leave
- Mouse Enter
- Mouse Click
- Mouse Double Click

### Mouse Button Up

This event will run when the Left mouse button is released over the event's object.

### Mouse Button Down

This event will run when the Left mouse button is pushed down over the event's object.

### Mouse Wheel

This event will run when the mouse wheel is scrolled over the event's object.

### Mouse Move

This event will run when the mouse moves anywhere over the event's object.

### Mouse Over

This event will run when the Left mouse button is released over the event's object and is sensitive to child elements. So, if the mouse exits a child object and is still over this object, the Mouse Over event will fire again.

### Mouse Leave

This event will run when the mouse is no longer hovering the event's object.

### Mouse Enter

This event will run when the mouse hovers the event's object or any of its children for the first time. It will not fire again if hover is blocked by children, unlike Mouse Over.

### Mouse Click

This event will run when the Left mouse button is pushed down and up quickly over the event's object.

### Mouse Double Click

This event will run when the Left mouse button is pushed down and up two times quickly over the event's object. If this is detected, the Mouse Click event will not be run.

# DRAGGING EVENT DESCRIPTIONS

## Dragging Events

Drag Start

Drag Move

Drag End

### Drag Start

This event will run when the Left Mouse Button is pressed down over an object and the mouse is then moved without lifting the Left Mouse Button. It only runs one time per drag.

### Drag Move

This event will run while the mouse is moving and the Left Mouse Button is still held down. It only occurs after the Drag Start has executed and before the Drag End executes.

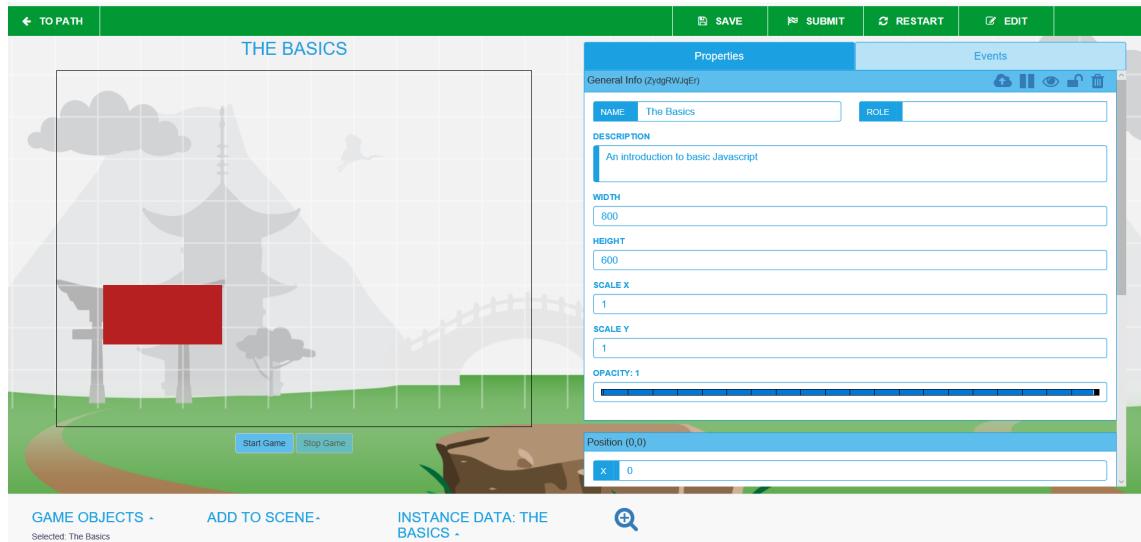
### Drag End

This event will run when the Left Mouse Button is released at the end of a drag. It is only executed one time.

## Activity 1

# The Basics

- 1** To begin, we will open a scene that has already been set up for us. Select the **Beginning Javascript** path. There are a few scenes available under the "Activities" header. Open the first scene, "The Basics," by clicking on the "play" button: 
- 2** In the scene, you will see something like this:

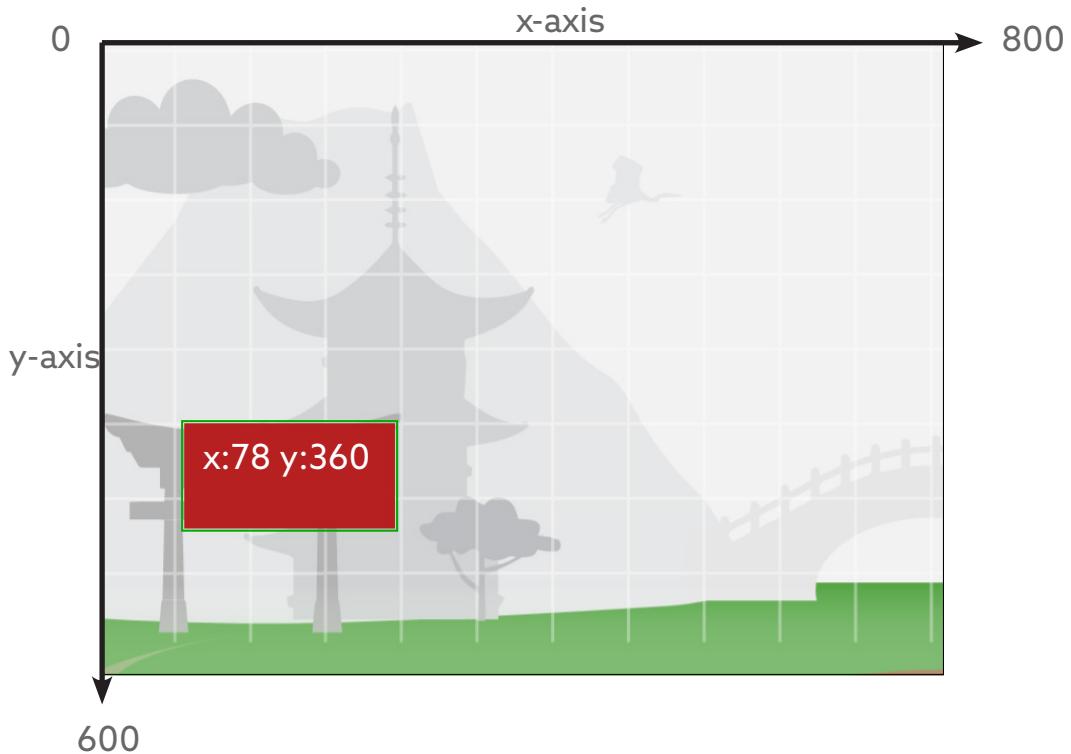


- 3** The large rectangle on the left is the game stage and everything you build in this scene appears there. At the moment there is a red rectangle in the stage. The rectangle is an object. An object can be a shape like a square or a circle, an image, text or even a group of objects. All objects have "properties" that can be defined using the menu on the right of the stage. The scene itself is also an object and can also be defined using properties.
- 4** Click on the red rectangle inside the stage. Now look at the properties for the rectangle. This rectangle has a name, ("rect"), a width, a height and many other properties.

- 5** You can change the object by changing its properties. Let's change the height of the rectangle. Click on the number underneath height and change the number from 100 to 150 and press "Enter."

NAME	rect
PARENT The Basics	
WIDTH	200
HEIGHT	150

- 6** You can also change the rectangle's position on the stage. Scroll down in properties until you see "Position." Currently it says the rectangle is at x:78 and y:360. What does that mean? The stage has two dimensions: left and right and up and down. These dimensions are represented by numbers, starting at 0 in the upper left corner. The farther you get from the corner, the larger the x and y numbers become. The x value is how far the rectangle is from the left edge and the y value is how far the rectangle is from the top edge.



- 7** The rectangle is currently at x:78. What number do you think it should be if you want it to be on the right side of the stage? We know the left side of the stage is 0. Let's add 400 to the x value. Click on the number of 78 and change it to 478 and hit enter.

Position (478,360)

X	478
Y	360

- 8** What happened? Now the rectangle is on the right side of the stage. What would you do if you wanted to move it higher on the stage?

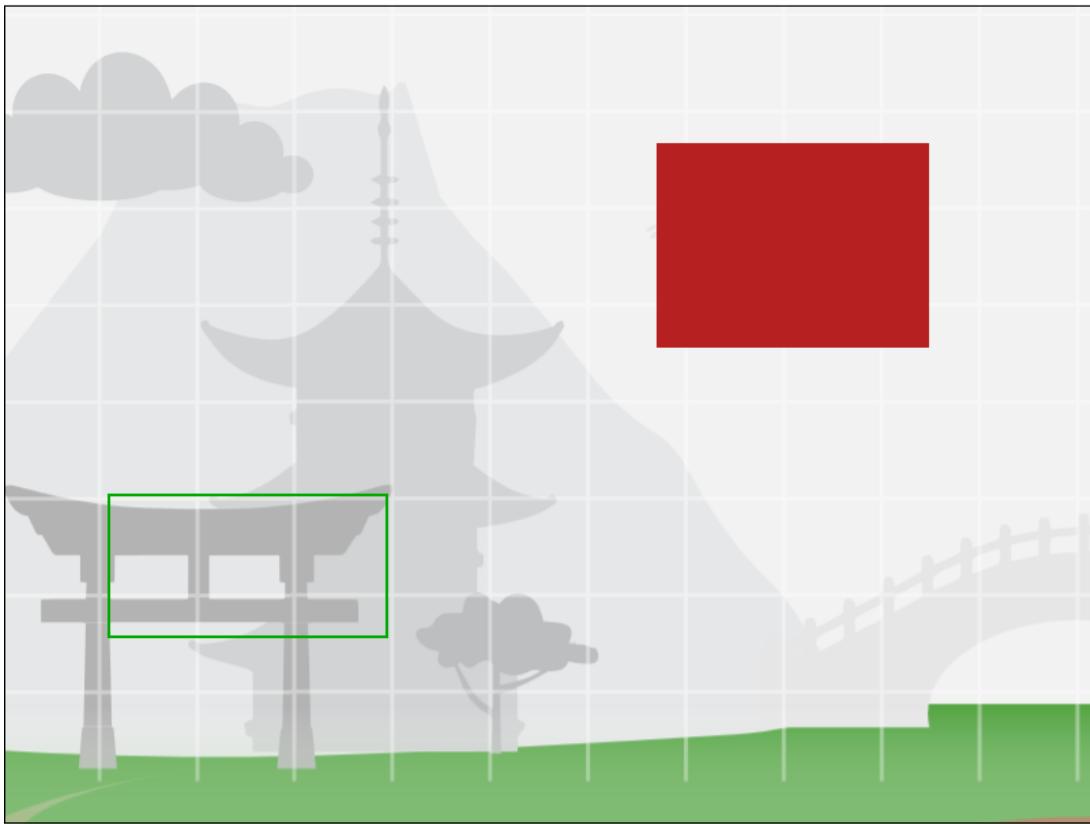
- 9** Remember, the top of the stage is y:0. The rectangle is currently at y:360, so changing the number from 360 to a smaller number should move it higher on the stage.

- 10** Change the y number from 360 to 100 and press enter.

Position (478,100)

X	478
Y	100

- 11** Now the rectangle is in the upper right of the stage at x:478, y:100.



- 12** So what does all of this have to do with javascript? By using javascript, we can change the properties of the objects on the stage. We can also send commands to the objects and get information from the objects as well. Using javascript and the objects in our scene, we can make a game!
- 13** See what changing the other properties for the rectangle does. Some of the properties like speed X and speed Y do not appear to do anything. All of the properties have a purpose as we shall soon see.

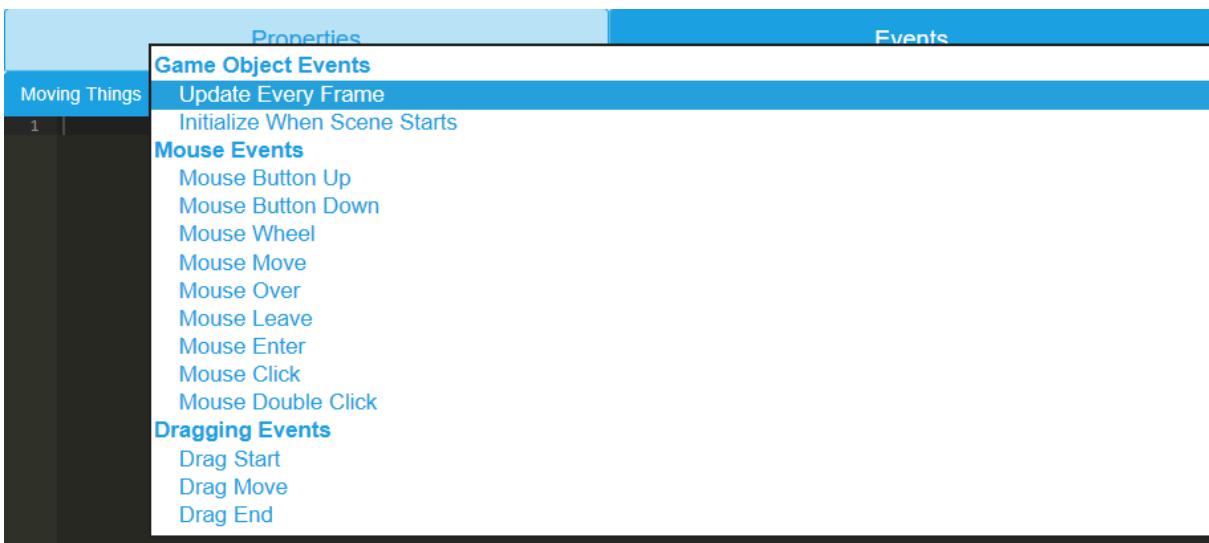
## Activity 2

# Moving Things

- 1** Let's open a new scene. Go back to the **Beginning Javascript** path. The next scene is called "Moving Things." Open it by clicking on the "play" button: 
- 2** This scene looks a lot like the scene we were using in The Basics. The rectangle now looks like a square because the width and height are the same. At the moment, this scene contains two objects: the rectangle and the scene itself.
- 3** So far, we've been modifying the objects using the properties menu. Now let's take a look at events. Make sure the scene object is selected by double clicking on the rectangle and click on the **Events** tab.



- 4** When the game is being played, the scene "listens" for certain events and when an event happens, it executes all of the code listed for that event. Click on "Update Every Frame." You will see a list of all of the events That you can use. Most of the time you will only use a couple of events for a game.



- 5** Every object has the same options for events and it is likely that as events happen, several objects will each be executing their own commands at the same time.
- 6** For now, make sure that **Initialize When Scene Starts** is selected. This event only happens once when the game is started. We're going to add code to move the rectangle to another location. Type in the text below and save your scene.

```
//this is a comment. Anything that follows 2 forward slashes is ignored  
//by the computer. Comments are useful for leaving notes without  
//disrupting the code.  
rect.x(50); //set the x value of the rectangle's location to 50
```

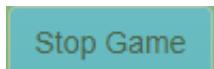
Moving Things	Initialize When Scene Starts </>
1 rect.x(50);	

"rect" is the name given to the rectangle when it was added to the scene. The very first thing we do is identify which object is getting a command. The period is there to separate the name from the properties we want to change on the object. In this case, we're changing the x position of the rectangle. The last part is known as the parameters. This is the value that we want to assign to the x property of the rectangle. The semicolon at the end is so that the program knows you're at the end of a line.

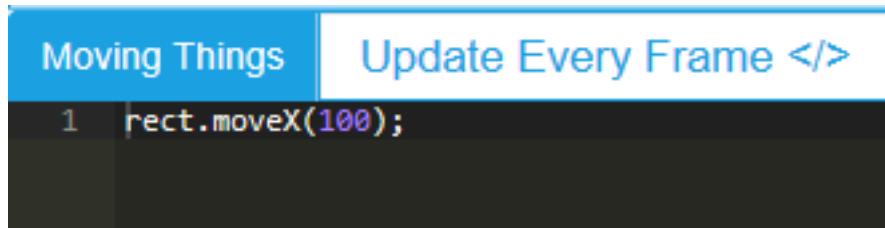
- 7** Click on **Start Game** and see what happens.
- 8** Click on **Stop Game**
- 9** We can change the y property of the rectangle as well. Click on **Events** and select **Initialize When Scene Starts** and add this to the code you've already written:

```
rect.y(250); //set the y value of the rectangle's location to 250
```

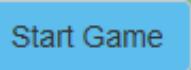
Moving Things	Initialize When Scene Starts </>
1 rect.x(50); 2 rect.y(250);	

- 10** Click on  and see what happens.
- 11** Click on 
- 12** While we can change the rectangle's location on the stage, it's not really moving. To do that, we need a different event.
- 13** Click on the **Events** tab, and select **Update Every Frame**. Add this code:

```
//move the rectangle along the x-axis at a speed of 100  
rect.moveTo(100);
```



If we had placed the code in Initialize When Scene Starts, it would have been executed exactly once. By placing it in the Update Every Frame event, the command is executed over and over again until the scene stops. As before, we are sending the command to the "rect" object. Instead of just giving it an X location, we are telling it to moveX, which is along the X axis. The number given in the parameters is the speed at which we want the rectangle to move. As we go from left to right, the x value is larger. The speed is telling the rectangle to move by 100 units to the right over and over again. If we wanted the rectangle to move to the left, we would use a negative number for the speed.

- 14** Click on  and see what happens.
- 15** Click on 
- 16** Click on the rectangle and click on properties. In the middle of the position properties is speed x and speed y. These properties are the default speed for the object and if we hadn't specified a number for the speed parameter, then the speed would have been 50.

SPEED X	50
SPEED Y	50

- 17** To test this, make sure the scene object is still selected. Click on **Events** and **Update Every Frame** and modify the code so that it looks like this:

```
//move the rectangle along the x-axis at the default speed
rect.moveX();
```

- 18** Click on **Start Game** and see what happens.

- 19** Click on **Stop Game**

- 20** Even though we had not specified a number for the speed in the parameters, the rectangle still moved, using the default speed of 50. Just like the other properties, we can change the speed x in code.

- 21** Click on **Events** and select **Initialize When Scene Starts**. Add this code:

```
//change the default speed of the rectangle to 200
rect.speedX(200);
```

We are telling the rectangle to change the speed x property from 50 to 200. What do you think will happen?

- 22** Click on **Start Game** and see what happens.

- 23** Click on **Stop Game**

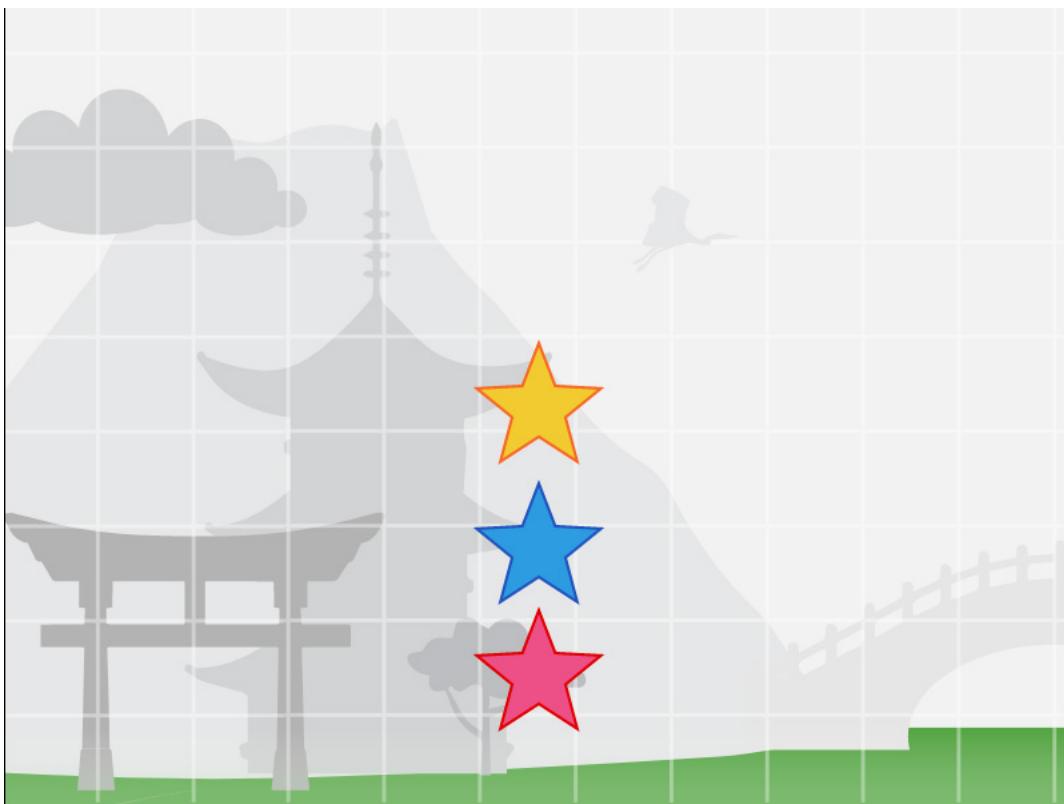
- 24** Go ahead and try different values for the x, y and speedX of the rectangle. Remember, if you want the rectangle to go from right to left, you need a negative number for your speed.

## Activity 3

# Round and Round

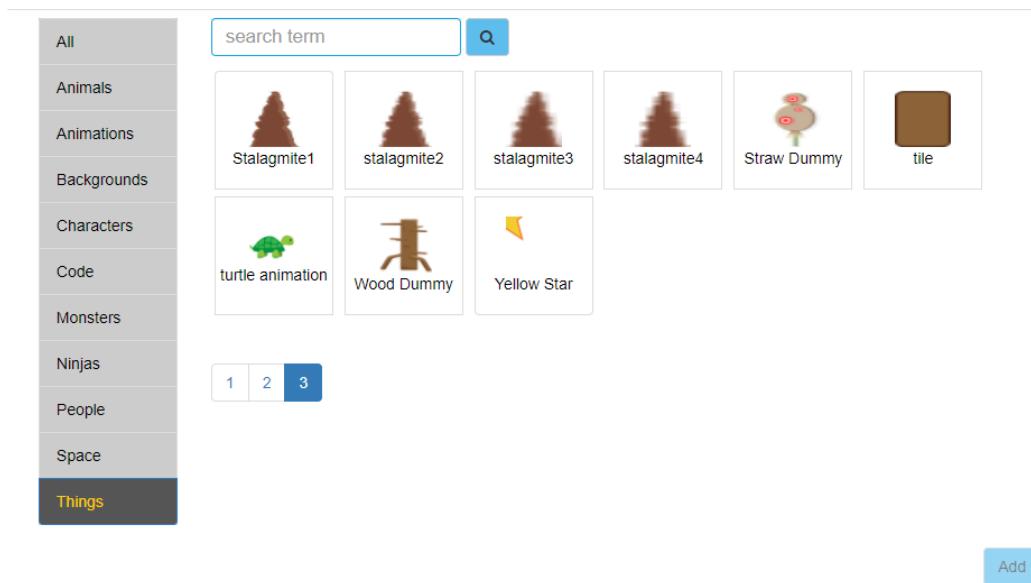
**1** Let's open a new scene. Go back to the **Beginning Javascript** path. The next scene is called "Round and round" Open it by clicking on the "play" button: 

**2** When we're done, your activity will look like the image below with three stars. We're going to make them spin in different ways.



**3** To make an object spin is very similar to making it move, except that the object is rotating around its origin point, which for the star, is its center.

- 4** Right now, the stage is empty. Let's add our first object. Click on the Search Assets menu  You will see something like this:

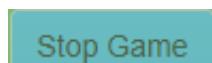


- 5** Select the "Things" menu on the left and search for the "Yellow Star" and click on "add" and close the Search Assets menu by clicking on the X in the upper right corner. The star is placed right where we want it, in the middle of the stage.
- 6** Click on the **yellow star** and select **Events** and **Update Every Frame**. Add this code:

```
$this.spin(40); //make this object rotate at a speed of 40
```

```
1 $this.spin(40);
```

The first word, "\$this," is another way of telling the code that we want to send the command to our selected object, which is the yellow star. We could also use the name of the star and the code would still work. We'll explain more about this later. Next is the word "spin", which is the command to have the star rotate around the star's origin point at the rate we specify. In this case, the rate is "40." Just like move, a positive number moves to the right (or clockwise) while a negative number moves to the left (or counter clockwise).

- 7** Click on  and see what happens.
- 8** Click on 
- 9** The yellow star spins. Let's move on to the next star.
- 10** Click on the Search Assets menu  Again, click on things and search for the blue star and add it to your scene and close the Search Assets menu.
- 11** The blue star appears below the yellow star.
- 12** Click on the **blue star** and select **Events** and **Update Every Frame**.  
Add this code:

```
1 $this.spin(-50); //make this object rotate at a speed of -50
```
- What do you think will happen?
- 13** Click on  and see what happens.
- 14** Click on 
- 15** What happened? The blue star was spinning, but it was spinning around the yellow star!

- 16** Let's take a closer look at the blue star. Select it and click on **Properties**. Scroll down and look at the property for OFFSET Y:

OFFSET Y -150

Remember, the object spins around its origin point and by changing the value of offset y, we've moved the origin point 150 pixels *above* the center of the star! Both the yellow star and the blue star are located at the same point on the stage (x:400, y:300), but because we have moved the origin point, the blue star is moving in a large circle around the yellow star.

- 17** Click on the Search Assets menu  Again, click on things and search for the **red star group** and add it to your scene and close the Search Assets menu.
- 18** Now let's look at the red star. The red star is part of a group called redStarGroup. When you double-click on the red star, the redStarGroup is selected.

NAME redStarGroup

PARENT Round and round

- 19** Click on the **redStarGroup** and select **Events** and **Update Every Frame**. Add this code:

```
$this.spin(70); //make this object rotate at a speed of 70
```

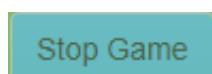
```
1 $this.spin(70);
```

- 20** It looks like the redStarGroup will rotate clockwise, but where is the origin point? Click on **Properties** and take a look:

OFFSET Y -200

**21** You can probably guess what the red star will do, right?

**22** Click on  and see what happens.

**23** Click on 

**24** The red star spins around the blue and yellow stars. Actually, since the red star is inside redStarGroup, it is the group that is spinning and the red star object is just going along for the ride. Can we give commands to objects inside groups? Yes we can!

**25** Make sure that you have the red star object selected - it will look like this:

NAME 

PARENT 

**26** With the **red star** selected, select **Events** and **Update Every Frame**. Add this code:

```
$this.spin(-70); //make this object rotate at a speed of -70
```

```
1 $this.spin(-70);
```

What do you think will happen?

**27** Click on **Start Game** and see what happens.

**28** Click on **Stop Game**

**29** It looks like the red star is staying still while it is moving around the yellow and blue stars. That's because the red star is spinning at exactly the opposite rate as redStarGroup. So everytime the group turns one way, the red star inside it makes the same turn in the opposite direction, making it look like it's not turning at all.

**30** Go ahead and try different numbers for the star's spinning and see what happens.



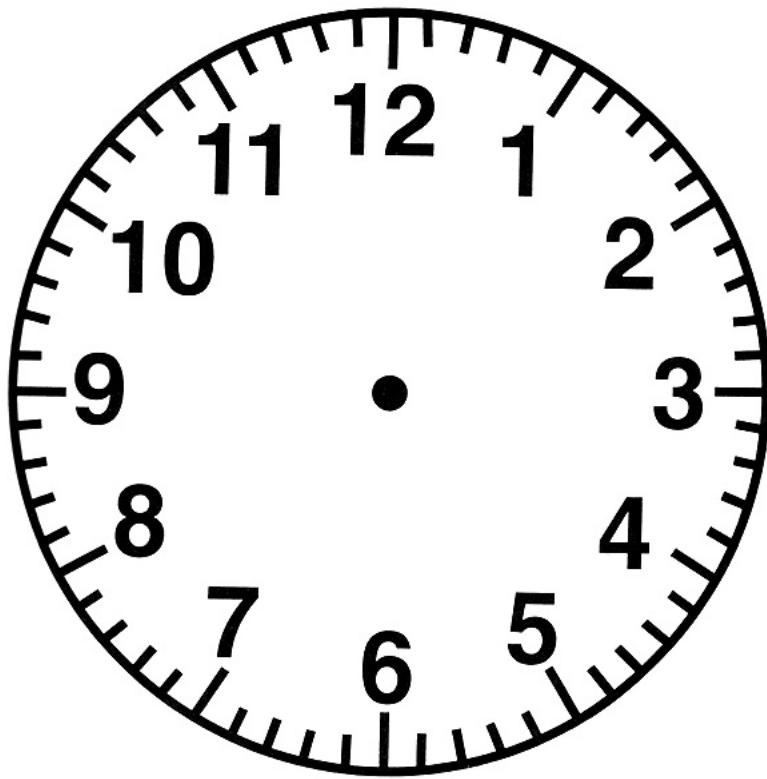
## Activity 4

# Clock

- 1 Let's open a new scene. Go back to the **Beginning Javascript** path. The next scene is called "Clock" Open it by clicking on the "play" button:

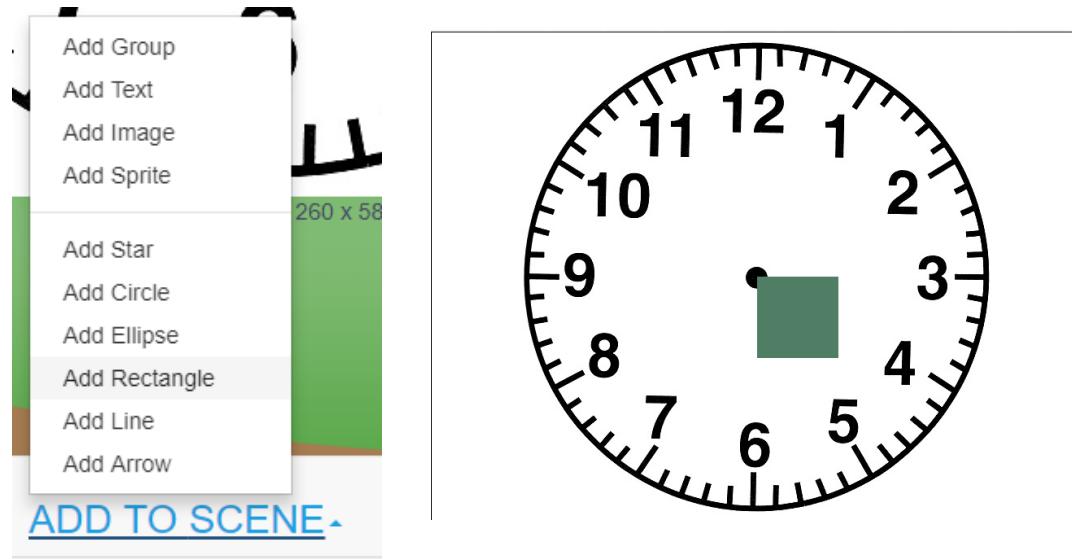


- 2 This new scene should look like the image below. We're going to make a working clock by changing the rotation of the clock hands.



- 3 The difference between spin and rotation is like the difference between moveX and x. One makes the object move while the other simply changes the object's position.

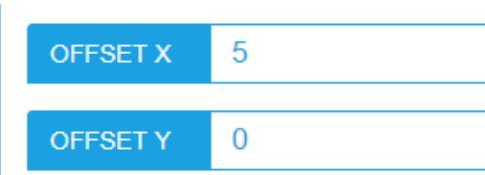
- 4** Our clock needs 3 hands for hour, minute and seconds. They will all start out as rectangle objects.
- 5** Make sure the scene is selected and click on ADD TO SCENE and click on "Add Rectangle." A randomly colored rectangle will be placed in the center of the stage.



- 6** Add 2 more rectangles to the scene. They will all be placed at the same position in the center, so you will only be able to see the one on top.
- 7** Select the last rectangle that you added (the one on top). This will be our second hand. Change the name of the rectangle to "sHand" and change its width to 10 and its height to 289.

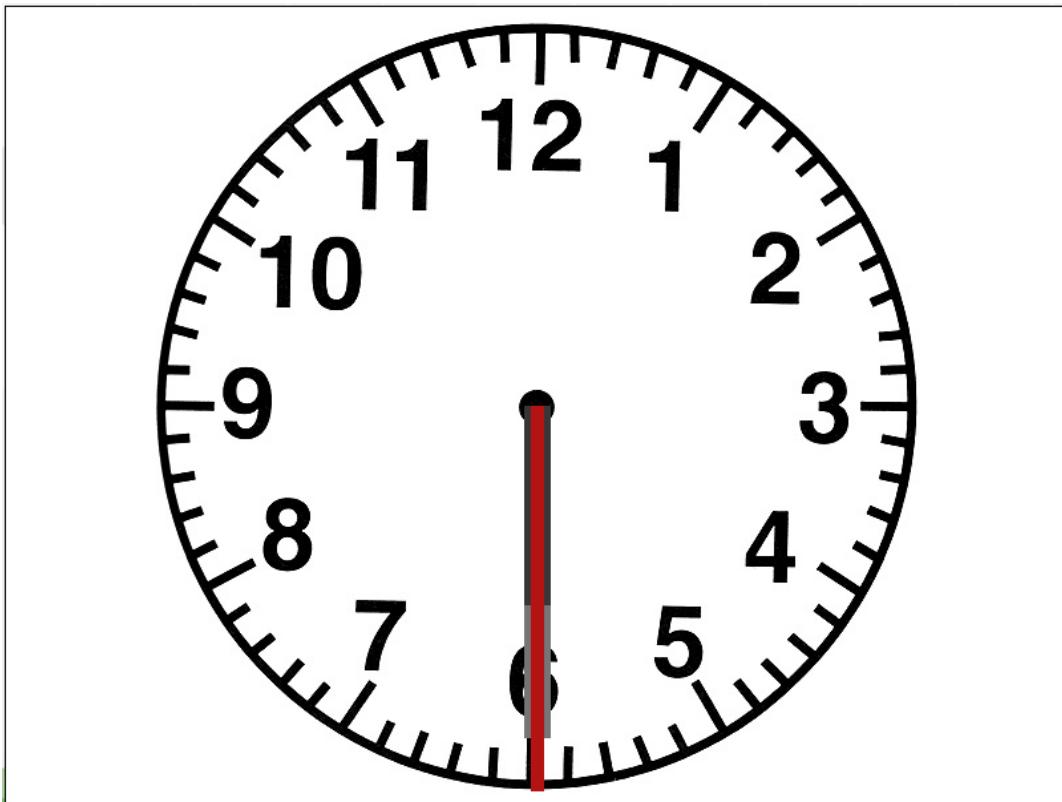


- 8** With "sHand" still selected, change the OFFSET X to 5. You can change the object's background color to any color you like.



Why did we change the x offset? The pivot of the hand was still on the upper left corner and we wanted to move it to the center of the top. Since the object is 10 pixels wide, an offset of 5 would put us right in the center.

- 9** Now that you have completed the second hand, it's time to move on to the hour hand. Select the next rectangle and change the name to "hHand," change the width to 20 and the height to 150.
- 10** We want to change the offset on this as well. Since this object is 20 pixels wide, the x offset should be half that, or 10.
- 11** You can change the background color of this object as well. Pick any color you like as long as it's different from the color of the second hand.
- 12** There should be one rectangle left, the minute hand. Select the last rectangle and change its name to "mHand" and change the width to 20 and the height to 250.
- 13** The minute hand has the same width as the hour hand, so it gets the same x offset of 10. Feel free to change the background color of this object to anything you like as long as it is not the same color as the minute or hour hands.
- 14** Now you should have something that looks like this:



**15** Now the clock has three hands - a short one for hours, a longer one for minutes and a long, skinny one for seconds. Each hand has an origin point at the top of the object so that when we rotate them, they will turn around the clock as if they were pinned to the center.

**16** Make sure the **Scene** object is selected and select **Events** and **Update Every Frame**. Add this code:

```
//use the value of $this.scene.second times 6 plus 180 to make the  
//rotation of the sHand object  
sHand.rotation($this.scene.second*6 + 180);
```

```
1 sHand.rotation($this.scene.second*6 + 180);  
2
```

We're using a special function to give us the current hours, minutes and seconds and they are turned into the variables, "\$this.scene.second," "\$this.scene.minute," and "\$this.scene.hour." So how do we make sure the second hand is in the right place?

Every time the second hand finishes a complete circle around the clock face it has taken 60 seconds. A circle has 360 degrees, so if we divide 360 by 60, we get 6, or the exact number of degrees for each position on the clock face.

The only problem is that the hand is pointing down instead of up (0 degrees in the Game Development Platform is straight down) so we have to add half of 360, or 180, to the result to get the actual rotation for the second hand.

Our code is the name of the object, "sHand" followed by the property "rotation" and the formula of seconds times 6 plus 180 for the rotation we need.

**17** Click on **Start Game** and see what happens.

**18** Click on **Stop Game**

**19** Every second, the second hand rotates around the clock face. We should be able to use the same formula for minutes as we used for seconds.

**20** Make sure the **Scene** object is selected and select **Events** and **Update Every Frame**. Add this code:

```
//use the value of $this.scene.minute times 6 plus 180 to make the  
//rotation of the mHand object  
mHand.rotation($this.scene.minute*6 + 180);
```

```
1 sHand.rotation($this.scene.second*6 + 180);  
2  
3 mHand.rotation($this.scene.minute*6 + 180);
```

mHand is the name of the minute hand and we use the variable \$this.scene.minute to find out what the current minute is. Everything else is exactly like the code we used for the second hand.

**21** Click on **Start Game** and see what happens.

**22** Click on **Stop Game**

**23** Our clock is almost finished! Let's take care of the hour hand.

**24** Make sure the **Scene** object is selected and select **Events** and **Update Every Frame**. Add this code:

```
//use the value of $this.scene.hour times 30 plus $this.scene.minute  
divided by 2.5 plus 180 to make the rotation of the hHand object  
hHand.rotation($this.scene.hour*30 + ($this.scene.minute/2.5) + 180);
```

```
1 sHand.rotation($this.scene.second*6 + 180);  
2  
3 mHand.rotation($this.scene.minute*6 + 180);  
4  
5 hHand.rotation($this.scene.hour*30 + ($this.scene.minute/2.5) + 180);
```

This formula is a little different from the minutes and seconds. There are only 12 hours on a clock face, so 360 divided by 12 gives us 30, which is the rotation for each number on the clock face. The next part of the formula isn't really necessary, but since the hour hand on an analog clock moves a little bit forward each minute, there is a formula to add that fraction to the rotation. Once again, we add 180 because the hour hand is pointing down instead of up.

**25** Click on **Start Game** and now you can see what time it is!

## BEGINNING JAVASCRIPT DAY TWO

On the first day, you were introduced to the Game Development Platform (GDP), how it contains objects, how those objects have properties and how the objects "listen" for events to execute instructions written in "code."

We learned about how the objects are on a stage and that the stage has an x-axis and a y-axis starting at 0 in the upper left corner. Objects that are placed on the stage are placed at x and y locations based on how far they are from the left edge and top of the stage. Finally, we learned that objects can be moved on the stage and that the rate of movement and direction is based on the speed assigned to that object.



## Activity 1

# Conditions

- 1** To begin, we will open a scene that has already been set up for us. Select the **Beginning Javascript** path. There are a few scenes available under the "Activities" header. Open the scene, "Conditions," by clicking on the "play" button: 
- 2** The scene looks familiar with a red rectangle right in the middle of the screen.
- 3** If you click on **Start Game**, the rectangle moves toward the right side of the screen, just like it did in "Moving Things," yesterday. Even though you can't see it, the rectangle is still going and won't stop until you stop the game.
- 4** **Stop the Game.**
- 5** Click on the rectangle and click on **Events** and **Update Every Frame**. This is what you will see:

```
//move this object along the x-axis at the default speed  
$this.moveX();
```

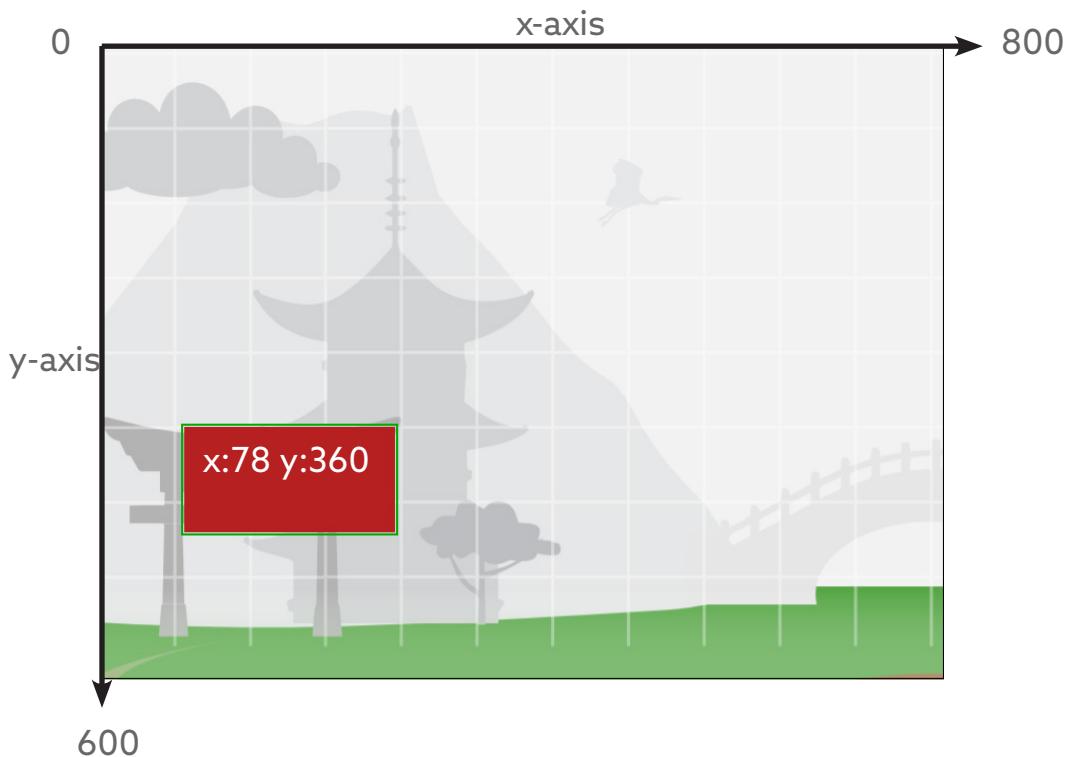


The first thing you will notice is that instead of the rectangle name, there is "\$this." We could have used the rectangle name and the rectangle would have moved just the same. However, by using "\$this," we are telling the event to apply the code to the object it is attached to, in this case, the rectangle. If you had code attached to a different object, such as the scene, then you would have to use the object name so that the program knows which object you want to send the command to. Later on we will see how the program can create objects with their own code attached and using "\$this" will be essential to making that work.

- 6** What would we do if we wanted the rectangle to change direction when it reached the edge of the screen? We would write a command that would tell the program "if the rectangle is at this location, then execute this other command." In javascript, this is known as a conditional.
- 7** Here's how it works. The statement contains an expression, such as if(water is wet) and if (and only if) that expression is true, then execute all of the code that is contained inside the brackets. If the expression is false, then all of the code in the brackets are ignored.

```
//if "water is wet" is true, do the following  
if(water is wet){  
    do this      //whatever command is here gets executed  
}
```

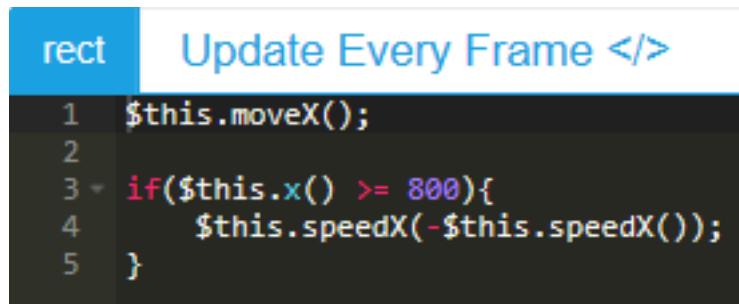
- 8** Most of the time, you will be using a mathematical equation for your expression, such as  $2 > 1$ ,  $3 == 3$ ,  $4 + 1 <= 5$ . With that in mind, how do we check to see if the rectangle is at the far right side of the stage?
- 9** Remember the chart from yesterday where we showed the x-axis and the y-axis? Looking at the chart, we can see that the far right of the screen is x:800.



**10** So if the x of the rectangle is higher than 800, it has reached the right side of the screen and we need to change the direction.

**11** Click on the rectangle, click on **Events** and select **Update Every Frame**. Add this code:

```
//if the x value of this object is greater than or equal to 800, do this
if($this.x() >= 800){
    //make the speedX of this object negative of what it was
    $this.speedX(-$this.speedX());
}
```



```
rect Update Every Frame </>
1 $this.moveX();
2
3 if($this.x() >= 800){
4     $this.speedX(-$this.speedX());
5 }
```

Our expression checks to see where the rectangle is on the stage. If it is greater than or equal to 800, then the rectangle's speed x is changed. Remember, a negative speed goes in the opposite direction. We're keeping the speed the same, just changing the direction.

**12 Start the Game.** What does the rectangle do when it reaches the right edge? Unfortunately, now the rectangle goes off the left edge.

**13 Stop the Game.**

**14** What would you do to check if the rectangle has reached the left side of the stage? Try it yourself before going to the next page.

**15** You would need another conditional, this time checking to see if the rectangle x value was less than or equal to 0.

**16** Click on the rectangle, click on **Events** and select **Update Every Frame**. Add this code:

```
//if the x value of this object is less than or equal to 0, do this
if($this.x() <= 0){
    //make the speedX of this object negative of what it was
    $this.speedX(-$this.speedX());
}
```

rect	<b>Update Every Frame &lt;/&gt;</b>
	<pre>1 \$this.moveX(); 2 3 if(\$this.x() &gt;= 800){ 4     \$this.speedX(-\$this.speedX()); 5 } 6 7 if(\$this.x() &lt;= 0){ 8     \$this.speedX(-\$this.speedX()); 9 }</pre>

Notice that we're using the exact same code in both conditionals to reverse the direction of the rectangle. That's because the opposite of a negative number is a positive number and the new speed of the rectangle would have it going to the right again.

**17 Start the Game.** Now the rectangle never goes off the stage, always changing directions when it reaches the edge.

**18 Stop the Game.**

**19** So far, we've been just moving the rectangle left and right on the x-axis. What would you do if you also wanted to move it up and down on the y-axis? Try it on your own before checking the answer on the next page.



**HINT:** The stage height is 600.

**20** The command to move the rectangle up and down is the same as the command to move it left and right, except that it is moveY instead of moveX.

**21** Click on the rectangle, click on **Events** and select **Update Every Frame**. Your code should look something like this:

```
//move this object along the x-axis at the default speed
$this.moveTo();
//if the x value of this object is greater than or equal to 800, do this
if($this.x() >= 800){
    //make the speedX of this object negative of what it was
    $this.speedX(-$this.speedX());
}
//if the x value of this object is less than or equal to 0, do this
if($this.x() <= 0){
    //make the speedX of this object negative of what it was
    $this.speedX(-$this.speedX());
}

//move this object along the y-axis at the default speed
$this.moveTo();
//if the y value of this object is greater than or equal to 600, do this
if($this.y() >= 600){
    //make the speedY of this object negative of what it was
    $this.speedY(-$this.speedY());
}
//if the y value of this object is less than or equal to 0, do this
if($this.y() <= 0){
    //make the speedY of this object negative of what it was
    $this.speedY(-$this.speedY());
}
```

**22** Start the Game.

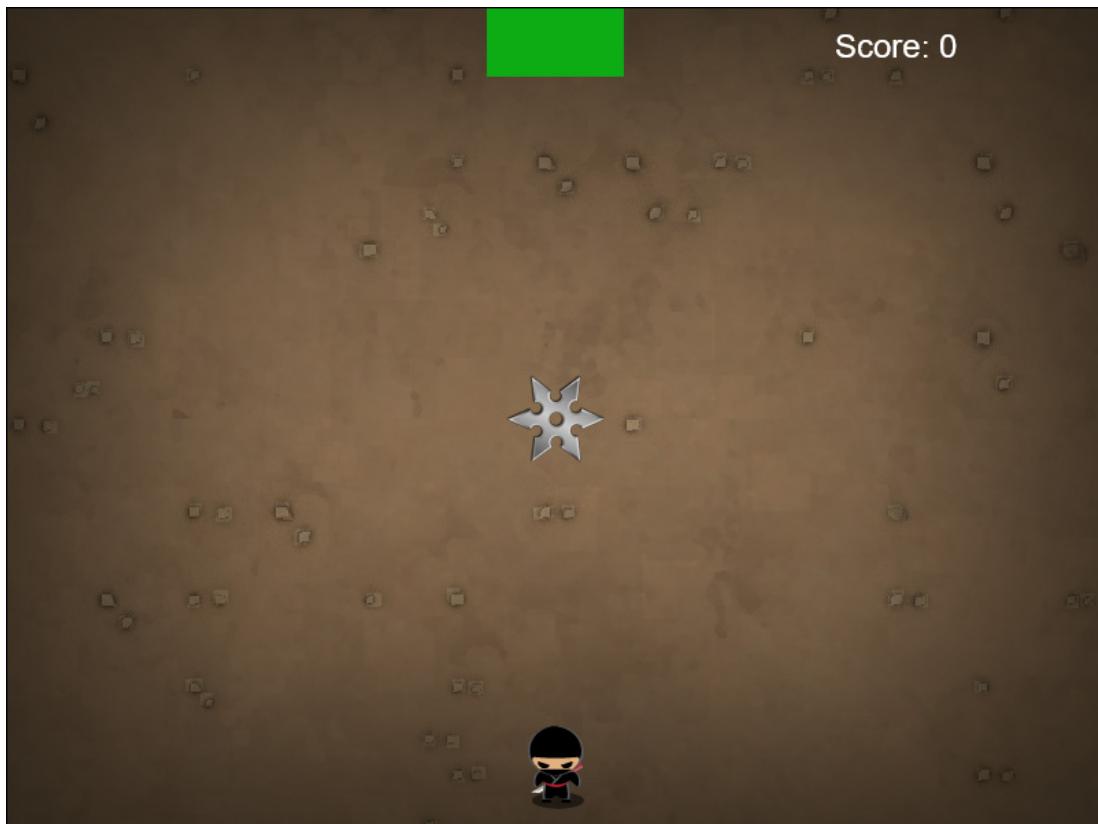
**24** Stop the game and try different values for the speedX and speedY of the rectangle. Remember, there are three ways that you can adjust the speed property of an object. See what different speeds does to the direction that the rectangle travels!

## Activity 2

# Dodge!

- 1 Now it's time to take what you've learned so far and make a game. Go back to the **Beginning Javascript** path. Open the scene, "Dodge," by clicking on the "play" button: 

- 2 Your scene will look like this:



- 3 The object of the game is to move the ninja from the bottom of the screen to the green rectangle at the top of the screen while avoiding the throwing star. If the ninja makes it to the top, he scores a point and starts again at the bottom. If he is hit by the throwing star, he is sent back to the bottom without getting any points.

**4** The ninja and the throwing star have already been programmed. **Start the Game** and move the ninja by using the AWSD keys on your keyboard. At the moment, nothing happens if the ninja touches the throwing star or the green rectangle.

**5 Stop the Game.**

**6** First of all, how do we detect if the ninja has been hit by the throwing star? We want to use a conditional, but what do we use for our expression?

**7** Fortunately, there is a game function in the Game Development Platform designed specifically for this purpose. It is called "isTouching" and it is used like this:

```
//anytime object1 is touching object2, do the following
if(object1.isTouching(object2){
    do this      //execute this command
}
```

**8** Click on the throwing star (shuriken) in the center of the stage and select **Events** and click on **Update Every Frame** and add this:

```
//if this object is touching the ninja object, do the following
if($this.isTouching(ninja)){
    ninja.x(400);      //place the ninja object at x:400
    ninja.y(555);      //place the ninja object at y:555
}
```

```
12 if($this.isTouching(ninja)){
13     ninja.x(400);
14     ninja.y(555);
15 }
```

Just as we are constantly checking to see if the throwing star has reached the edges of the stage, we now also check to see if the throwing star object has made contact with the ninja object. Every time that they touch, the ninja is set back to the starting position at the bottom of the screen.

**9** Start the Game and see what happens when the ninja is hit by the throwing star.

**10** Stop the Game.

**11** Click on the green rectangle at the top of the stage and select **Events** and click on **Update Every Frame** and add this:

```
//if this object is touching the ninja object, do the following
if($this.isTouching(ninja)){
    ninja.x(400);      //place the ninja object at x:400
    ninja.y(555);      //place the ninja object at y:555
}
```

**12** The game is almost complete, except for one thing. There isn't any way to keep track of the score.

**13** To make the score work, we need to create something to store what the current score is, and then we need to display that information on the stage.

**14** We will store the score in a variable. A variable is like a container for numbers, text or anything else you need to keep track of in your program. Variables can be changed often while the program is being run. When a command is given that includes a variable, whatever the variable is holding at that time is used.

**15** Select the green rectangle, click on **Events** and **Initialize When Scene Starts** and add this:

```
$this.gameScore = 0;      //the variable "gameScore" equals 0
```

We're creating a variable called gameScore and everytime the scene starts, we are setting its value to 0. Now we can use it in all events that are part of the green rectangle.

**16** Click on the green rectangle and select **Events** and click on **Update Every Frame** and change the code to look like this:

```
if($this.isTouching(ninja)){
    ninja.x(400);
    ninja.y(555);
    //gameScore is equal to the value of gameScore plus 1
    $this.gameScore = $this.gameScore+1;
    //the text value of the text object "score" is the current value of
    //gameScore
    score.text($this.gameScore);
}
```

goal      Update Every Frame </>

```
1 if($this.isTouching(ninja)){
2     ninja.x(400);
3     ninja.y(555);
4     $this.gameScore = $this.gameScore+1;
5     score.text($this.gameScore);
6 }
```

Now every time the ninja reaches the green rectangle, not only is the ninja being sent back to the bottom, but we are adding 1 to the gameScore variable. Then we send a command to the score object (a text object in the upper right of the screen) to change its text to be whatever is in the gameScore variable.

**17** Start the Game and see how quickly you can reach 10 points!

## Activity 3

# Padlock

- 1** Let's open a new scene. Go back to the **Beginning Javascript** path. The next scene is called "Padlock" Open it by clicking on the "play" button: 
- 2** This new scene should look like the image below. By pressing the left or right arrow keys, we're going to turn the wheel of the padlock and confirm the number that is shown on the wheel.



- 3** We have learned a little about spin and rotation from the Clock and Round and Round activities. We will use spin to turn the wheel of the padlock. But how do we know what number is at the top of the wheel? We can find that out by using the rotation property of the wheel.

- 4** Make sure the **Scene** object is selected and select **Events** and **Update Every Frame**. Add this code:

```
//if the left arrow key is pressed, then the variable leftPressed is true  
var leftPressed = isKeyPressed(Keys.leftArrow);  
//if the right arrow is pressed, then the variable rightPressed is true  
var rightPressed = isKeyPressed(Keys.rightArrow);  
  
1 var leftPressed = isKeyPressed(Keys.leftArrow);  
2 var rightPressed = isKeyPressed(Keys.rightArrow);
```

Here we have two variables, `leftPressed` and `rightPressed`. Instead of being given a number or true or false, the values of these two variables depend on the “`isKeyPressed`” function. This function checks to see if the indicated key on the keyboard is being pressed and if so, the variable is true. If the key isn’t being pressed, the variable is false.

- 5** What do we do with a variable that is true or false? We use it in a condition. While still in **Scene, Events, Update Every Frame**. add this code:

```
//if leftPressed is true, then do the following  
if(leftPressed){  
    plWheel.spin(-40);      //spin the object plWheel at a speed of -40  
}  
  
1 var leftPressed = isKeyPressed(Keys.leftArrow);  
2 var rightPressed = isKeyPressed(Keys.rightArrow);  
3  
4 if(leftPressed){  
5     plWheel.spin(-40);  
6 }  
7
```

As long as the left arrow key is being pressed, the condition is true and the padlock wheel (`plWheel`) spins counter clockwise.

- 6** **Start the Game** and turn the wheel by pressing the left arrow key.
- 7** **Stop the Game.**

- 8** We can use similar code for turning the wheel to the right. Make sure the **Scene** object is selected and select **Events** and **Update Every Frame**. Add this code:

```
//if rightPressed is true, then do the following
if(rightPressed){
    plWheel.spin(40);      //spin the object plWheel at a speed of 40
}

1  var leftPressed = isKeyPressed(Keys.leftArrow);
2  var rightPressed = isKeyPressed(Keys.rightArrow);
3
4  if(leftPressed){
5      plWheel.spin(-40);
6  }
7
8  if(rightPressed){
9      plWheel.spin(40);
10 }
11
```

Now the left arrow spins the wheel counter clockwise and the right arrow spins it clockwise.

- 9** The last thing we need to do is confirm which number is pointing up on the wheel. We will use rotation for that. While still in **Scene**, **Events**, **Update Every Frame**. add this code:

```
//give the variable "wheelNumber" the value of 360 minus the rotation
//value of plWheel divided by 9, rounded to the closest whole number
var wheelNumber = Math.round((360 - plWheel.rotation())/9);
```

```
1  var leftPressed = isKeyPressed(Keys.leftArrow);
2  var rightPressed = isKeyPressed(Keys.rightArrow);
3
4  if(leftPressed){
5      plWheel.spin(-40);
6  }
7
8  if(rightPressed){
9      plWheel.spin(40);
10 }
11
12 var wheelNumber = Math.round((360 - plWheel.rotation())/9);
13
```

The rotation of the wheel is a number between 0 and 360. Since there are 40 numbers on the wheel (0 and 40 are the same), each number on the wheel is 9 degrees apart (360 divided by 40 equals 9). When we turn the wheel to the right, it starts counting down from 40. When we turn the wheel to the left, it counts up from 0. In order to get the correct number from the wheel's rotation, we have to subtract the rotation from 360 and divide the result by 9.

We don't want any fractions, so we use `Math.round` to round the result to the closest whole number.

- 10** We still have to confirm that we have the right number. While still in **Scene, Events, Update Every Frame**. add this code:

```
//set the text value of the object "lockNumber" to the value of
//"/wheelNumber"
lockNumber.text(wheelNumber);
```

```
1  var leftPressed = isKeyPressed(Keys.leftArrow);
2  var rightPressed = isKeyPressed(Keys.rightArrow);
3
4  if(leftPressed){
5      plWheel.spin(-40);
6  }
7
8  if(rightPressed){
9      plWheel.spin(40);
10 }
11
12 var wheelNumber = Math.round((360 - plWheel.rotation()) / 9);
13
14 lockNumber.text(wheelNumber);
```

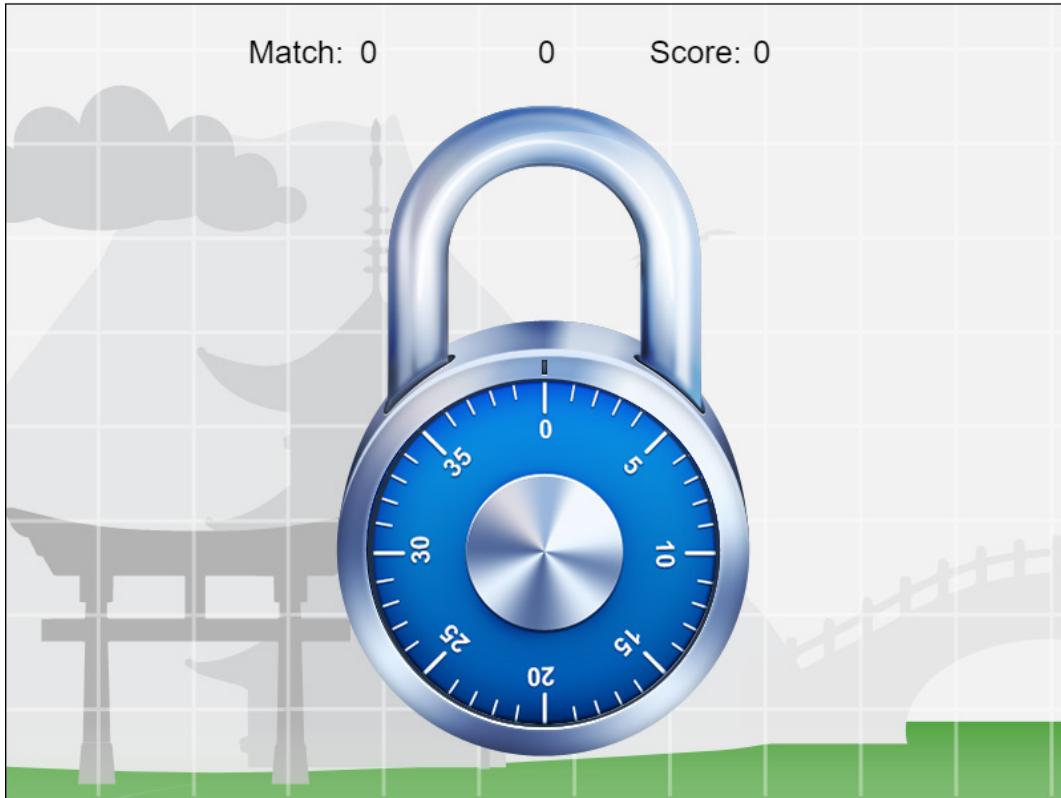
"lockNumber" is the text object at the top of the screen. When `wheelNumber` changes, that number is displayed in the `lockNumber` text object.

- 11** **Start the Game** and turn the wheel by pressing the left and right arrow keys. Do the numbers match up? This is not much of a game yet, but it has lots of possibilities.

## Activity 4

# Number Match

- 1** Let's open a new scene. Go back to the **Beginning Javascript** path. The next scene is called "Number Match" Open it by clicking on the "play" button: 
- 2** This new scene should look like the image below. We're going to take what we've built with the Padlock and make it into a game where you have to turn the wheel to match the number given on the left.



- 3** In this activity, we're going to expand on what we know about conditionals and also learn something about random numbers. Right now, the number on the wheel shows all the time. We could easily match the number just by holding down the left or right arrow key forever. Instead, we want the number to update only when neither key is being pressed. If the wheel is stopped on the same number as the match number, then the player scores a point and a new match number is chosen.

- 4** How do we tell if either key is being pressed? With a condition. Make sure the **Scene** object is selected and select **Events** and **Update Every Frame**. Add this code below the "add code here" comment:

```
//if either leftPressed OR rightPressed is true, do what follows  
if(leftPressed || rightPressed){
```

```
5  
6 //ADD CODE HERE  
7 if(leftPressed || rightPressed){  
8 }
```

Remember, a condition checks to see if an expression is true before executing the code within it. In this case, we have two expressions: leftPressed and rightPressed with the symbols "||" between them. The "||" means **OR**. So if either leftPressed is true or rightPressed is true, then the whole condition is true. If either button is pressed (or both buttons) then the code is executed. If neither button is pressed then the condition is false. Multiple expressions can be combined in a condition using the "||" (OR) operator or the "&&" (AND) operator. Refer to the chart below to see how to use these operators to combine expressions.

Left Statement	Operator	Right Statement	Result
true		false	true
true		true	true
false		true	true
false		false	false
true	&&	false	false
true	&&	true	true
false	&&	true	false
false	&&	false	false

- 5** You may have noticed that our condition is still open as we have not closed off the brackets. If we ran the program now, it would not run correctly because of this error.

- 6** Beneath the code you just entered are the two conditions for turning the wheel. So if either leftPressed or rightPressed is true, the program then executes the code for turning the wheel depending on which key is pressed. But what we really want to know is if neither key is being pressed. In that case, we expand our condition with an else statement.
- 7** Make sure the **Scene** object is selected and select **Events** and **Update Every Frame**. Add this code underneath the left and right wheel turning conditions:

```
//if neither leftPressed or rightPressed is true, do what follows
} else {
```

```
6 //ADD CODE HERE
7 if(leftPressed || rightPressed){
8
9 if(leftPressed){
10     plWheel.spin(-40);
11 }
12 if(rightPressed){
13     plWheel.spin(40);
14 }
15
16 } else {
17 }
```

Now we can execute one set of commands if either left or right arrow is pressed and another set if they are not.

- 8** Underneath the "} else {" statement is the code we used to calculate and display the wheel number. Below that, we need to add another bracket to properly close the condition. Your code should look like this:

```
if(leftPressed || rightPressed){
    if(leftPressed){
        plWheel.spin(-40);
    }
    if(rightPressed){
        plWheel.spin(40);
    }
} else {
    var wheelNumber = Math.round((360 - plWheel.rotation()) / 9);
    if(wheelNumber === 0){
        lockNumber.text("0");
    } else {
        lockNumber.text(wheelNumber);
    }
} //close off the bracket
```

```

15
16 } } else {
17
18     var wheelNumber = Math.round((360 - plWheel.rotation()) / 9);
19
20 if(wheelNumber === 0){
21     lockNumber.text("0");
22 } else {
23     lockNumber.text(wheelNumber);
24 }
25
26
27
28
29
30
31 }
32

```

The code for displaying the wheelNumber has changed a little bit. In JavaScript, the number 0 by itself is not displayed, so we have to trick it by telling it to display the character "0" instead. Again, we are using an "else" statement so that if wheelNumber equals 0, we will display "0" and if it equals anything else, we display that number.

**9 Start the Game.** Notice that the number for the wheel above the lock is only updated when the wheel is not being turned.

**10 Stop the Game.**

**11** We still need to check to see if the lockNumber matches the matchNumber. While still in **Scene, Events, Update Every Frame**. add this code before the last bracket of our condition statement:

```

//if the value of matchNumber equals the value of wheelNumber, do
//the following
if($this.scene.matchNumber == wheelNumber){
    //add 1 to the total value of gameScore
    $this.scene.gameScore = $this.scene.gameScore + 1;
    //update the scoreNumTxt object
    scoreNumTxt.text($this.scene.gameScore);
    //let matchNumber be a new random number between 0 and 39,
    //rounded to the closest whole number
    $this.scene.matchNumber = Math.round(random(39,0));
}

```

```
15
16 } else {
17
18     var wheelNumber = Math.round((360 - plWheel.rotation()) / 9);
19
20     if(wheelNumber === 0){
21         lockNumber.text("0");
22     } else {
23         lockNumber.text(wheelNumber);
24     }
25
26     if($this.scene.matchNumber == wheelNumber){
27         $this.scene.gameScore = $this.scene.gameScore + 1;
28         scoreNumTxt.text($this.scene.gameScore);
29         $this.scene.matchNumber = Math.round(random(39,0));
30     }
31 }
32 }
```

Every time there is a correct match, we add one to the score and display it. Then we need to select a new value for matchNumber. To do this, we use a function called **random** to automatically pick a number inside the range we specify for it. In this case we want a number between 0 and 39. In JavaScript, random numbers include fractions, so we use Math.round on the result to round it to the nearest whole number.

## 12 Start the Game.

See how quickly you can get a score of 10!

## BEGINNING JAVASCRIPT DAY THREE

So far we've learned how to move things on the stage, how to check when certain conditions are met and we built our first game.

Now that you know the basics, we will be introducing some new ways to make games with Javascript. Don't worry, all we are doing is building on what you already know, giving you more options to make the objects on the stage do what you want them to do.



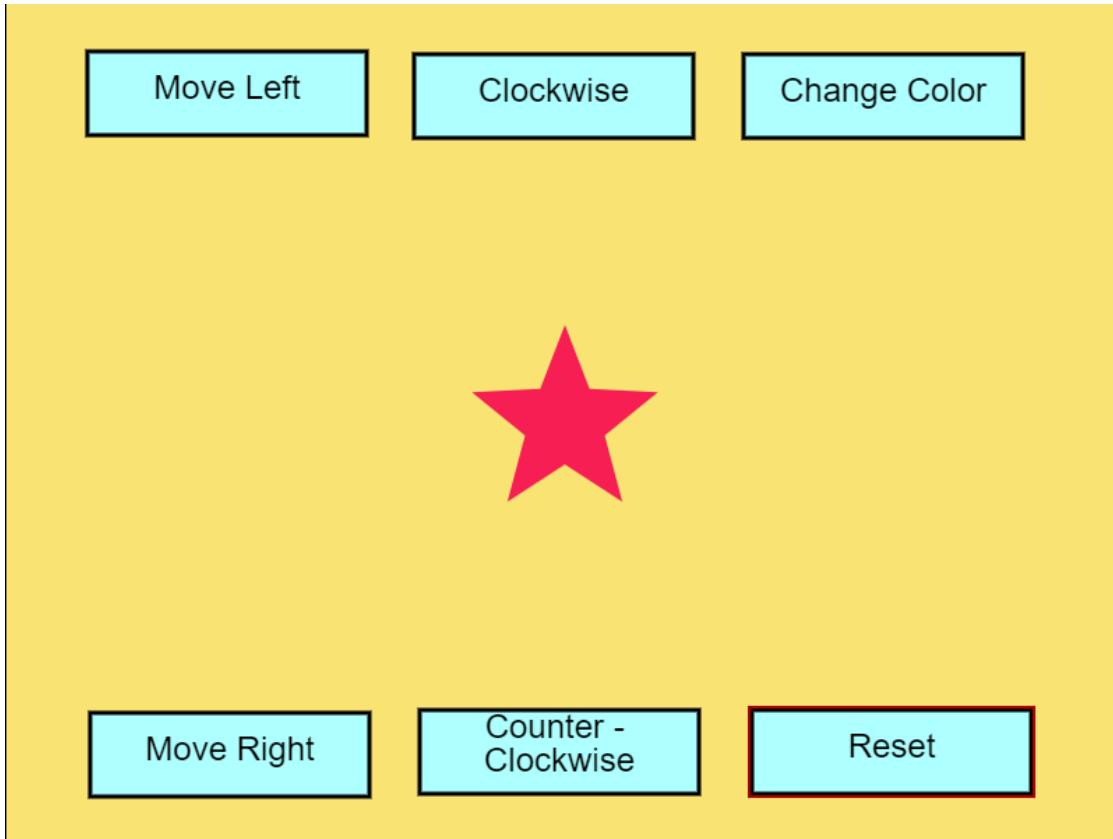
# Activity 1

## Buttons

- 1 Select the **Beginning Javascript** path. Open the scene, "Buttons" by clicking on the "play" button:



- 2 Your scene will look like the image below. When you click on any of the buttons, something happens to the star in the middle. The change color and reset buttons work, but the other buttons don't do anything.



- 3 Let's start by looking at the star. Click on it and click on **Events** and **Update Every Frame**. You should see this code:

```
//move this object along the x-axis at the rate of speedX  
$this.moveX($this.speedX());  
//spin this object at the rate of "spinSpeed"  
$this.spin($this.spinSpeed);
```

```
1 $this.moveX($this.speedX());  
2 $this.spin($this.spinSpeed);  
3
```

Every frame the star is told to move along the x-axis and spin.

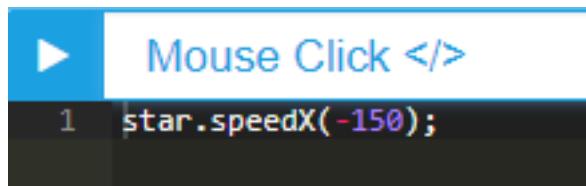
**4** Start the game. What happens?

**5** Stop the game.

**6** Why isn't the star moving? Remember, the object moves or spins at the rate it has been given and at the moment, that rate is 0. We can use the buttons to change that rate to whatever we want.

**7** Click on the "Move Left" button in the upper left corner. Make sure you have the object named "**Left**" selected - you may have to click on the button a second time to select it. Click on **Events** and choose the **Mouse Click** event from the pull down menu. Add this code:

```
//make the value of the star object's speedX -150  
star.speedX(-150);
```



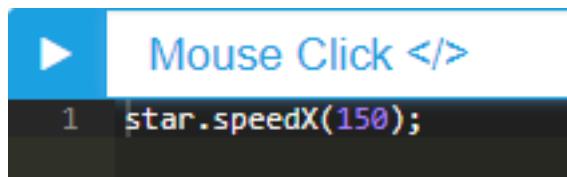
**8** Start the game and click on the Move Left button. Does the star move?

**9** Stop the game.

**10** Clicking on the Move Left button changes the star's speedX to -150 and it moves left. What about the Move Right button?

- 11** Click on the "Move Right button in the lower left corner. Make sure you have the object named "**Right**" selected - again, you may have to click on the button a second time to select it. Click on **Events** and choose the **Mouse Click** event from the pull down menu. Add this code:

```
//make the value of the star object's speedX 150  
star.speedX(150);
```



Now you should be able to make the star move left and right by clicking on the buttons. Let's try it out.

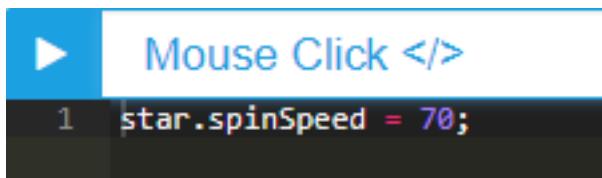
- 12** Start the game and click on either of the move buttons.

- 13** Stop the game.

- 14** Can you guess what you need to do to make the buttons that spin the star work? Remember that the star already has a spin command, but the variable is set to 0. Think about it and try it on your own before turning the page.

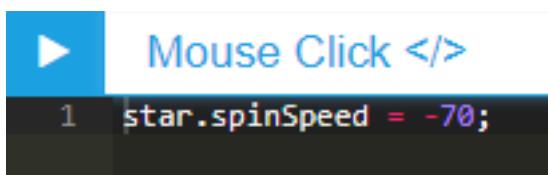
- 15** Click on the "Clockwise" button in the upper middle. Make sure you have the object named "**Clockwise**" selected - again, you may have to click on the button a second time to select it. Click on **Events** and choose the **Mouse Click** event from the pull down menu. Add this code:

```
//make the value of "star.spinSpeed" equal to 70  
star.spinSpeed = 70;
```



- 16** Let's finish this up by clicking on the "Counter-Clockwise" button in the lower middle. Make sure you have the object named "**counterClockwise**" selected - again, you may have to click on the button a second time to select it. Click on **Events** and choose the **Mouse Click** event from the pull down menu. Add this code:

```
//make the value of "star.spinSpeed" equal to -70  
star.spinSpeed = -70;
```



- 17** Start the game and try all of the buttons.

- 18** The change color button uses a special function to randomly select colors. Clicking on that button uses the function. Can you guess how the reset button works? After you stop the game, take a look at the Mouse Click event for that object and see what it does. Did you guess right?

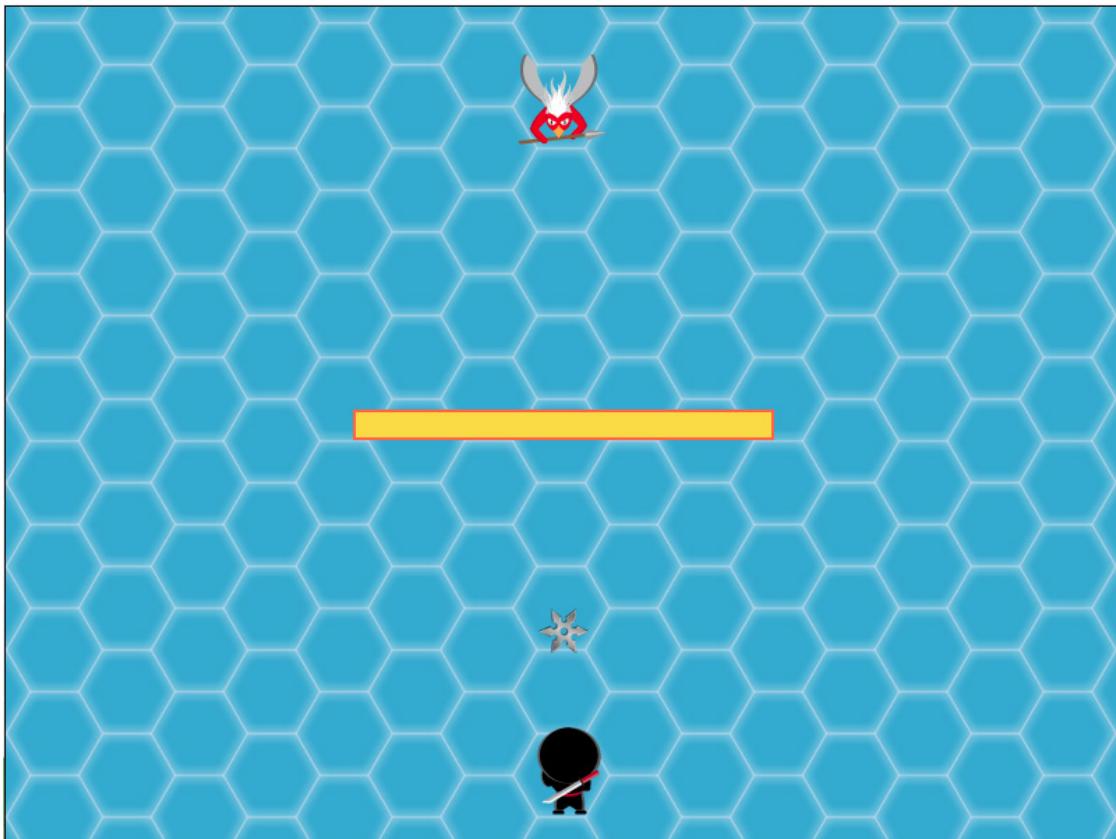
## Activity 2

# Ricochet

- 1 Select the **Beginning Javascript** path. Open the scene, "Ricochet," by clicking on the "play" button:



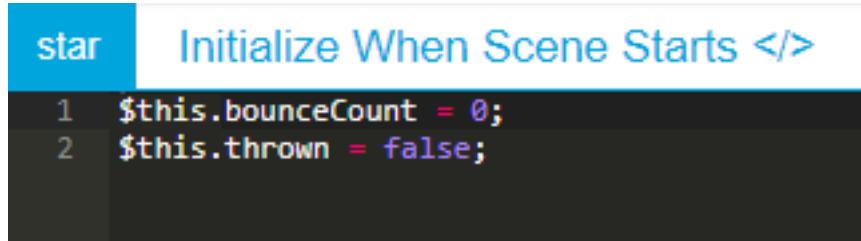
- 2 Your scene will look like the image below. The goal is for the ninja to hit the bug by throwing the throwing star. Since there's a wall in the way, the ninja will have to bounce it off the sides to reach the bug.



- 3 Most of the programming is already set up for moving the throwing star and for when the bug is hit by the throwing star. When the game is being played, the throwing star is positioned behind the ninja. But we have placed it above the ninja so you can see it and select it.

- 4** Select the throwing star and click on **Events** and **Initialize When Scene Starts**. Inside the event are two variables. The first is called "bounceCount" and it is used to keep track of how many walls the throwing star bounces off of. We are starting it at 0. When the star hits three walls, it is returned to the ninja.

```
$this.bounceCount = 0; //the variable "bounceCount" is equal to 0  
//this object has a variable called "thrown" with a value of false  
$this.thrown = false;
```



The second variable is a little different for two reasons. The first reason is that we are making it part of the star object by adding "\$this." to the variable name. This way, other objects in the scene can refer to the variable by looking at "star.thrown." By setting this variable to equal "false," we are limiting it to only two values - thrown can either be "true" or "false," nothing else.

- 5** Why did we make a variable that is only true or false? Remember that a conditional uses an expression that must be true or false. This way, any other object in the scene can immediately see if the star is thrown or not and execute code depending on the result.
- 6** What we want to do is click on the scene and send the star traveling in the direction of where we clicked, but ONLY if the star has not been thrown yet.
- 7** Make sure the scene is selected by double clicking on any object in the scene. Click on **Events** and **Mouse Click**. As you might have guessed, Mouse Click is an event that is triggered each time someone clicks on that object - in this case, the scene.

- 8** The first thing we want to do is check if the star has been thrown or not. So we will start with a conditional. Add this to the event:

```
//if the variable star.thrown is false, then this condition is true and any  
//code that follows is executed.  
if(star.thrown === false){
```

- 9** If star.thrown is false, then we find out where the player clicked and use that to set up the direction of the throwing star. Add this code:

```
//the x and y values of the pointer are given to the variable "pos"  
var pos = getPointerPos();
```

We're creating a variable called "pos" and using "getPointerPos()" to give the current x and y of the pointer to the variable.

- 10** The next step is to compare where the player clicked with the location of the ninja, which is where the star always starts. Remember, a negative speed goes up and to the left while a positive speed goes down and to the right. By subtracting the x and y of pos from the x and y of the ninja, we get the speedx and speedy for the star.

- 11** Underneath the code you just entered, add this:

```
//speedX of the object star is the x value of pos minus the x  
//value of the ninja object  
star.speedX(pos.x() - ninja.x());  
//speedY of the object star is the y value of pos minus the y  
//value of the ninja object  
star.speedY(pos.y() - ninja.y());  
//the variable called "thrown" in the star object is now true  
star.thrown = true;  
}
```

So if you click to the left of the ninja, the star's speedX is a negative number and the star uses that speed with a moveX command to move to the left. If you click to the right of the ninja, the star moves to the right. Same for clicking above the ninja with the y value. No matter where you click, the combined speedX and speedY of the star will send it in that direction. The last command changes the star. thrown variable to true, meaning that the star can now move.

**12** Here's what the completed code should look like:

```
if(star.thrown === false){  
    var pos = getPointerPos();  
    star.speedX(pos.x() - ninja.x());  
    star.speedY(pos.y() - ninja.y());  
    star.thrown = true;  
}
```

The image shows a Scratch script titled "Richochet" with a "Mouse Click" event. The script contains the following code:

```
1 if(star.thrown === false){  
2     var pos = getPointerPos();  
3     star.speedX(pos.x() - ninja.x());  
4     star.speedY(pos.y() - ninja.y());  
5     star.thrown = true;  
6 }
```

**13** Start the Game and click where you want the star to go. If it doesn't hit the bug after hitting three walls, you can try again.

**14** Stop the Game.

**15** You may have noticed that once you learn where to position the pointer to hit the bug, you can keep hitting the bug without moving the pointer again. We can make this a little more interesting by making the bug move to a different position each time it is hit.

**16** Select the bug and select **Events** and **Update Every Frame**. We have already programmed the bug to change a variable called "move" from false to true every time the bug is hit. Let's use that variable in a condition. Add this code:

```
//if "move" is true, execute the following code  
if($this.move){  
    //change the x value of this object to a random number from  
    //250 to 550  
    $this.x(random(550,250));  
    //change the y value of this object to a random number from  
    //50 to 200  
    $this.y(random(200,50));  
    $this.move = false;      //the variable "move" is now false  
}
```

```
1 if($this.move){  
2     $this.x(random(550,250));  
3     $this.y(random(200,50));  
4     $this.move = false;  
5 }
```

Every time the bug is hit, it changes the “move” variable to true, which generates random x and y positions for the bug. The random numbers are in a range to keep the bug mostly behind the yellow wall. Finally, the move variable is set to false until the next time the bug is hit.

- 17 **Start the Game** and see what happens when the bug is hit. Now the game should be a little bit more challenging.

## Activity 3

# Driving

- 1** So far we have been using moveX and moveY to move objects on the stage. But we can also steer an object and move it where it is pointing as you will see in this next game. Go back to the **Beginning Javascript** path. Open the scene, "Driving" by clicking on the "play" button: 
- 2** Your scene will look like this:



- 3** In this game, the coin moves to a random location in the stage. You must then steer the car by using the left and right arrow keys and make it move forward by pressing the up arrow key to pick up the coin. You score a point and the coin moves to another location.

- 4** Let's set up the arrow keys to steer the car. Click on the car, click on Events and Update Every Frame and add this code:

```
//when the left arrow is pressed, "leftPressed" is true  
var leftPressed = isKeyPressed(Keys.leftArrow);  
//when the right arrow is pressed, "rightPressed" is true  
var rightPressed = isKeyPressed(Keys.rightArrow);  
//when the up arrow is pressed, "upPressed" is true  
var upPressed = isKeyPressed(Keys.upArrow);
```

```
1 var leftPressed = isKeyPressed(Keys.leftArrow);  
2 var rightPressed = isKeyPressed(Keys.rightArrow);  
3 var upPressed = isKeyPressed(Keys.upArrow);  
4
```

Everytime you press one of those keys, the variable is given a value of true. Otherwise, it is false.

- 5** With that in mind, we can easily come up with a conditional for each key. Add this code:

```
//if "leftPressed" is true, do the following  
if(leftPressed){  
}  
  
//if "rightPressed" is true, do the following  
if(rightPressed){  
}  
  
//if "upPressed" is true, do the following  
if(upPressed){  
}
```

- 6** So how exactly do we get the car to turn? Every object has a rotation property that has a value from 0 to 360. A positive rotation turns the object clockwise, a negative rotation turns the object counter-clockwise. The center of the rotation is the object's origin point, which is the upper left corner of the object.. We have moved the car's origin point to the center of the car by adjusting the car's offset properties. To actively turn an object every frame, we must tell it to *spin*.

- 7** So when the right arrow is pressed, we want to spin the car clockwise and when the left arrow is pressed, we want to spin the car counter-clockwise. Add this to your code:

```
if(leftPressed){  
    //as long as leftPressed is true, spin this object at the rate of -40  
    $this.spin(-40);  
}  
  
if(rightPressed){  
    //as long as rightPressed is true, spin this object at the rate of  
    //40  
    $this.spin(40);  
}  
  
4  
5 ▾ if(leftPressed){  
6     $this.spin(-40);  
7 }  
8  
9 ▾ if(rightPressed){  
10    $this.spin(40);  
11 }  
12
```

The number tells the object how fast to spin. A larger number makes it spin faster.

- 8** **Start the Game** and use the left and right arrow keys to turn it left and right.
- 9** **Stop the Game.**

- 10** Now that we can steer the car, it is time to move it forward. There is a command called moveForwardByRotation that takes the current rotation of the object and moves it in that direction. Select the car, click on **Events** and **Update Every Frame** and add this code:

```
if(upPressed){  
    //as long as upPressed is true, move this object in the direction  
    //it is rotated  
    $this.moveForwardByRotation();  
}  
  
12  
13 - if(upPressed){  
14     $this.moveForwardByRotation();  
15 }
```

- 11** Start the Game and take the car for a spin! What happens when you drive off the edge?

**12** Stop the Game.

- 13** It's kind of hard to steer a car when you can't see it. Instead, if the car goes off the edge of the scene, let's have it reappear on the other side. Select the car, click on **Events** and **Update Every Frame** and add this code:

```
if($this.y()>600){ //if this object's y value is greater than 600  
    $this.y(0);      //place this object at y:0  
}  
  
if($this.y()<0){   //if this object's y value is less than 0  
    $this.y(600);    //place this object at y:600  
}  
  
if($this.x()>800){ //if this object's x value is greater than 800  
    $this.x(0);      //place this object at x:0  
}  
  
if($this.x()<0){  //if this object's x value is less than 0  
    $this.x(800);    //place this object at x:800  
}
```

```
16
17 if($this.y()>600){
18     $this.y(0);
19 }
20
21 if($this.y()<0){
22     $this.y(600);
23 }
24
25 if($this.x()>800){
26     $this.x(0);
27 }
28
29 if($this.x()<0){
30     $this.x(800);
31 }
```

**14** We still need to set up the coin to move to a new location and increase the score when the car touches it.

**15** Select the coin and select **Events** and **Update Every Frame**. Add this code:

```
//if this object is touching the carHit object, do the following
if($this.isTouching(carHit)){
    //place this object at a random x location between 0 and 750
    $this.x(random(750,0));
    //place this object at a random y location between 0 and 550
    $this.y(random(550,0));
    $this.score +=1;//add 1 to the total value of the "score" variable
    //update the text object "labelScore" with the value of "score"
    labelScore.text($this.score);
}
```

```
3 if($this.isTouching(carHit)){
4     $this.x(random(750,0));
5     $this.y(random(550,0));
6     $this.score +=1;
7     labelScore.text($this.score);
8 }
```

**16** Start the Game and see how long it takes you to collect 25 coins!

## BEGINNING JAVASCRIPT DAY FOUR

One thing you might not have noticed about the Game Development Platform is that everything happens as soon as it can. The code is executed for all objects and only waits if conditions for triggering an action have not yet been met. To force the program to wait, we must create a timer.

Also, in today's activities we will be re-using objects by making copies of them with clone. And finally, we let the computer do a little bit of the work for us by letting it create random numbers.

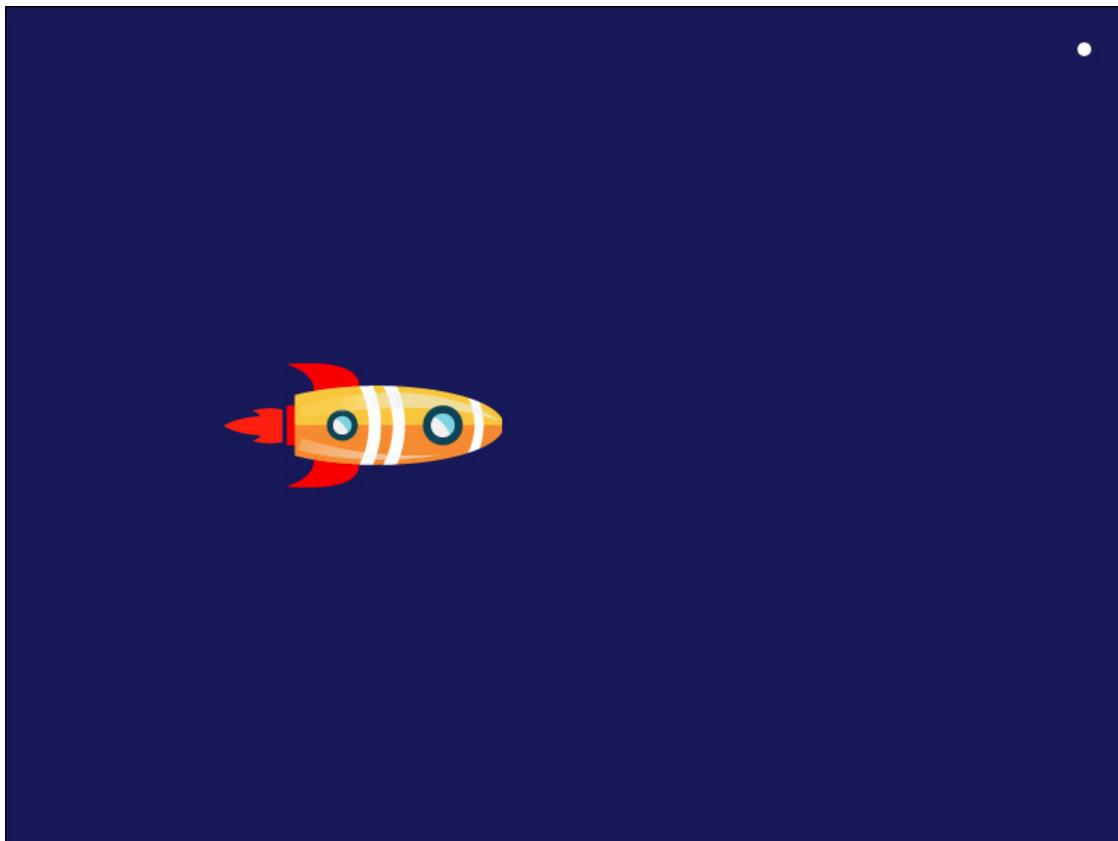


## Activity 1

# Space Flight

**1** Select the **Beginning Javascript** path. Open the scene, "Space Flight" by clicking on the "play" button: 

**2** Your scene will look like the image below. In this activity, we're going to have this spaceship fly through an endless field of stars and give you the ability to make it climb or dive.



**3** Notice the white dot in the upper right corner? We're going to tell the Game Development Platform to copy it over and over again to make a field of stars. To do so, we will be using the clone command.

- 4** With the scene selected, click on **Events** and **Initialize When Scene Starts**. You will see this code:

```
//Wait until the game has started before doing anything
if($this.scene.state() == "PLAY"){
    //the createTimer function executes the following code every
    //0.2 seconds (200 milliseconds)
    createTimer(200, function(){

});

}

1  if($this.scene.state() == "PLAY"){
2      createTimer(200, function(){
3
4
5
6
7
8
9      });
10 }
```

The first line makes sure that the code only runs when the game is actually being played and not while it is stopped. The second line is a timer that executes any code that follows every 0.2 seconds. We'll be explaining timers a little bit later. Right now, we're concerned with the code that the timer executes.

- 5** As we mentioned before, we want to make a clone of the star. Add this line right under the second line:

```
//the variable "nstar" represents a clone of the star object
var nstar = star.clone();
```

```
1 if($this.scene.state() == "PLAY"){
2     createTimer(200, function(){
3         var nstar = star.clone();
4
5
6
7
8
9     });
10 }
```

What this line does is create a clone of the object "star" and we're using the variable nstar to give properties to that clone. Every 0.2 seconds, a new clone will be made and each new clone will be assigned to the same variable. How does that work? The properties for each clone belong to that clone only, even after a new clone is made.

- 6 The only properties our cloned stars need are an x and y location, and a speed. Add this underneath the code you just entered:

```
//place the cloned object at x:800
nstar.x(800);
//place the cloned object at a random y location between 0 and 600
nstar.y(random(600,0));
//give the cloned object a random speedX between 100 and 700
nstar.speedX = random(700,100);
```

```
1 if($this.scene.state() == "PLAY"){
2     createTimer(200, function(){
3         var nstar = star.clone();
4         nstar.x(800);
5         nstar.y(random(600,0));
6         nstar.speedX = random(700,100);
7     });
8 }
```

As you can see, each cloned star begins on the right side of the stage at x: 800. However, the y and speedX values are a little different. Instead of giving a specific number, we are telling the program to pick a random number for us between the highest value and the lowest value. So the clone y will be somewhere between 0 and 600 and the clone speedX will be somewhere between 100 and 700.

**7** The star itself has instructions to move based on its speedX and every clone will have those instructions as well. **Start the Game**. Now the stars are moving from right to left and making the rocket look like it's flying.

**8 Stop the Game.**

**9** We want the rocket to dive and climb as it flies through space. Select the rocket, select **Events** and **Update Every Frame**. Add this code:

```
//assign the current x and y values of the pointer to the variable "pos"  
var pos = getPointerPos();
```

```
1 var pos = getPointerPos();  
2
```

Remember, getPointerPos gets the location in the scene of where ever the pointer is. However, if the pointer is outside of the scene, then there is no data for the function to get.

**10** If there is no data, the program will give us an error. To avoid that, add this line under what you just entered:

```
//if "pos" has a value, execute the following code  
if(pos){
```

```
1 var pos = getPointerPos();  
2  
3 if(pos){  
4 }
```

This way, the next part of code will only be executed if there actually is a value for pos.

- 11** If the pointer is above the rocket, we want to move it up. If it is below the rocket, we want to move it down. Let's start with moving it up. Add this beneath the code you just entered:

```
//if the y value of "pos" is less than the y value of this object, do this
if(pos.y < $this.y()){
    //move this object on the y axis at a rate of negative speedY
    $this.moveY(-$this.speedY());
    //the value of variable "rot" is the difference between pos.y and this
    //object's y, divided by 4
    var rot = (pos.y-$this.y())/4;
    //if the value of "rot" is less than -35, do this
    if(rot < -35){
        rot = -35;    //the value of "rot" is -35
    }
    $this.rotation(rot); //rotate this object by the value of "rot"
}
```

```
1  var pos = getPointerPos();
2
3  if(pos){
4      if(pos.y < $this.y()){
5          $this.moveY(-$this.speedY());
6          var rot = (pos.y-$this.y())/4;
7          if(rot < -35){
8              rot = -35;
9          }
10         $this.rotation(rot);
11     }
12
13
14
15
16 }
```

Remember, if we want to move something upwards, it needs a negative speedY. The remainder of the code rotates the rocket upwards based on how far the pointer is from the rocket. This makes the climbing of the rocket look more natural.

- 12** Now to move the rocket down. We'll be using the same code, but using a positive speedX instead. Add this code beneath the code you just entered and don't forget to close off your brackets!

```
//if the y value of "pos" is greater than the y value of this object, do
//do the following
if(pos.y > $this.y()){
    //move this object on the y axis at a rate of speedY
    $this.moveY($this.speedY());
    //the value of variable "rot" is the difference between pos.y
    //and this object's y, divided by 4
    var rot = (pos.y-$this.y())/4;
    //if the value of "rot" is greater than 35, do this
    if(rot > 35){
        rot = 35;      //the value of "rot" is 35
    }
    $this.rotation(rot); //rotate this object by the value of "rot"
}
}

1  var pos = getPointerPos();
2
3  if(pos){
4      if(pos.y < $this.y()){
5          $this.moveY(-$this.speedY());
6          var rot = (pos.y-$this.y())/4;
7          if(rot < -35){
8              rot = -35;
9          }
10         $this.rotation(rot);
11     }
12     if(pos.y > $this.y()){
13         $this.moveY($this.speedY());
14         var rot = (pos.y-$this.y())/4;
15         if(rot > 35){
16             rot = 35;
17         }
18         $this.rotation(rot);
19     }
20 }
21 }
```

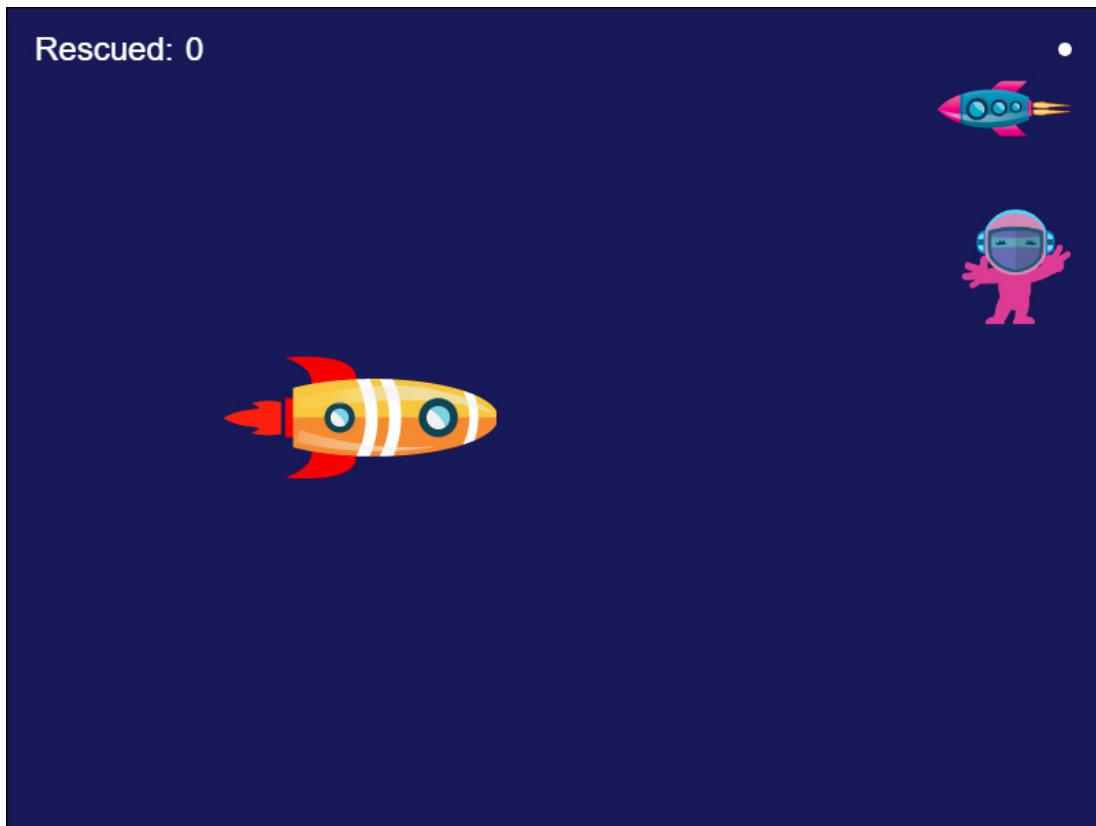
- 13** **Start the game.** Notice how the rocket climbs and dives as you move the pointer? But flying alone in space isn't very exciting. We'll make things more interesting in the next activity.

## Activity 2

# Space Rescue

- 1 Go back to the **Beginning Javascript** path. Open the scene, "Space Rescue" by clicking on the "play" button: 

- 2 Your scene will look like this:



- 3 In addition to what was added in the Space Flight game, there are now two new objects plus a score display. In this game, the goal is to rescue the astronauts by flying into them with your rocket while avoiding the blue rockets.

- 4** Make sure the scene ("Space Rescue") is selected either by double clicking anywhere in the scene or by selecting it from the **GAME OBJECTS** menu in the lower left corner. Just as we did with the stars in the Space Flight game, we're going to make clones of the bad ship and the astronaut. Let's start with the bad ship. With the scene selected, click on **Events** and **Initialize When Scene Starts** and add this below the code for the starTimer:

```
//the variable "shipTimer" is a timer that every two seconds does this  
shipTimer = createTimer(2000, function(){  
    //nship is a new clone of the badShip object  
    var nship = badShip.clone();  
    //place this clone at x:850  
    nship.x(850);  
    //place this clone at a random y value between 50 and 550  
    nship.y(random(550,50));  
    //set the speedX of this clone at -200  
    nship.speedX(-200);  
    //make this clone visible  
    nship.visible(true);  
});
```

- 5** Next is the astronaut. With the scene selected, click on **Events** and **Initialize When Scene Starts** and add this below the code for the shipTimer:

```
//the variable "astroTimer" is a timer that every 5 1/2 seconds does this  
astroTimer = createTimer(5500, function(){  
    //nastro is a new clone of the flyingNinja object  
    var nastro = flyingNinja.clone();  
    //place this clone at x:900  
    nastro.x(900);  
    //place this clone at a random y value between 100 and 500  
    nastro.y(random(500,100));  
    //set the speedX of this clone at -100  
    nastro.speedX(-100);  
    //make this clone visible  
    nastro.visible(true);  
});
```

## Space Rescue

## Initialize When Scene Starts </>

```
1 if($this.scene.state() == "PLAY"){
2     starTimer = createTimer(200, function(){
3         var nstar = star.clone();
4         nstar.x(800);
5         nstar.y(random(600,0));
6         nstar.speedX = random(700,100);
7     });
8
9     shipTimer = createTimer(2000, function(){
10        var nship = badShip.clone();
11        nship.x(850);
12        nship.y(random(550,50));
13        nship.speedX(-200);
14        nship.visible(true);
15    });
16
17     astroTimer = createTimer(5500, function(){
18        var nastro = flyingNinja.clone();
19        nastro.x(900);
20        nastro.y(random(500,100));
21        nastro.speedX(-100);
22        nastro.visible(true);
23    });
24 }
```

Now there are three createTimer commands, each with their own name so that they each act independently. The first timer should look familiar. It's the code to move the stars. The second timer clones a blue ship every 2 seconds (2000 milliseconds) and the last one clones a pink ninja every 5 and a half seconds. And each timer will be doing this again and again as long as the game is running.

- 6** Select the blue rocket, click on **Events** and **Update Every Frame** and add this:

```
//as long as this object is visible, do the following
if($this.visible()){
    $this.moveX();      //move this object on the x-axis at speedX
}

//if this object has an x value less than -50, do this
if($this.x() < -50){
    $this.remove();    //remove this object
}

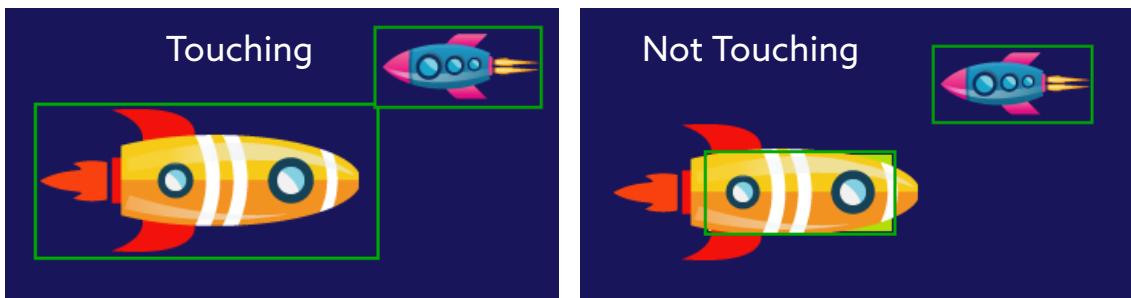
//if this object is touching the rocketHit object, do the following
if($this.isTouching(rocketHit)){
    $this.remove();    //remove this object
    rocket.remove();  //remove the rocket object
    //reset all of the running timers
    $this.scene.cleanupTimers();
}
```

badShip      Update Every Frame </>

```
1 if($this.visible()){
2     $this.moveX();
3 }
4
5 if($this.x() < -50){
6     $this.remove();
7 }
8
9 if($this.isTouching(rocketHit)){
10     $this.remove();
11     rocket.remove();
12     $this.scene.cleanupTimers();
13 }
```

In the first line, we check to see if the blue rocket is visible. Making the clone visible is the last thing we do when the clone is created. If it's visible, we move it based on its speedX. All of the stars have different speeds, but we're keeping this one constant. Then we check to see if the rocket has moved off the left side of the scene. If so, we use the remove command to destroy the clone - we don't need it anymore. Also, if the blue rocket is touching the hit area of the orange rocket (rocketHit), we remove them both, ending the game. Stay away from the blue rockets, okay?

**7** You might be wondering what is a hit area and how does the orange rocket have one? If we look at the orange rocket the way isTouching sees it, we would see a large rectangle that stretches out to contain all of the rocket, even the parts that are empty. And when the rocket rotates, that rectangle gets even larger. Behind the rocket is a rectangle called "rocketHit." It is smaller than the rocket so that other objects have to get closer to rocketHit before the isTouching function shows that they are in contact. Every frame, rocketHit is always in the same location as the rocket. You don't see it, but it helps make the game more fun.



**8** **Start the game.** Now there are blue rockets flying towards you. Use the mouse to move your ship out of the way.

**9** **Stop the game.**

**10** We need do the same thing for our astronaut. Click on it, click on **Events** and **Update Every Frame** and add this:

```
//as long as this object is visible, do the following
if($this.visible()){
    $this.moveToX();      //move this object on the x-axis at speedX
    $this.spin(40);        //spin this object at the rate of 40
}

//if this object has an x value less than -100, do this
if($this.x() < -100){
    $this.remove();        //remove this object
}

//if this object is touching the rocketHit object, do the following
if($this.isTouching(rocketHit)){
    $this.remove();        //remove this object
    //take score.text, convert it to a number, add 1 to that number
    //and have score.text display that new number.
    score.text(parseInt(score.text())+1);
}
```

```
1 if($this.visible()){
2     $this.moveTo();
3     $this.spin(40);
4 }
5
6 if($this.x() < -100){
7     $this.remove();
8 }
9
10 if($this.isTouching(rocketHit)){
11     $this.remove();
12     score.text(parseInt(score.text())+1);
13 }
```

This looks a lot like the code for the blue rocket with a couple of exceptions. First, we're telling the astronaut to spin in addition to moving. Spin makes the object rotate around it's origin point. A positive number spins it to the right (clockwise) and an negative number spins it to the left (counter-clockwise). We still remove the clone if it touches the rocket or leaves the edge of the scene. However, when it touches the rocket, we also increase the score by one. Since "score" is a text object, by giving it a text command, we can change what it displays. What we're doing is taking what is already in score, converting it to a number, and adding 1 to that number before displaying the new number as text.

**11 Start the game.** Try to rescue some astronauts. Does it look like they're disappearing before they touch the rocket?

**12 Stop the game.**

**13** When detecting if the astronaut is touching the rocket, the code is counting a lot of empty space outside of the astronaut as part of the astronaut. However, we know that the front end of the rocket is always at x:367. We can use that to make things a bit more precise.

**14** Click on the astronaut, click on **Events** and **Update Every Frame** and add this:

```
if($this.visible()){
    $this.moveX();
    $this.spin(40);
}

if($this.x() < -100){
    $this.remove();
}

//only do the following if this object's x value is less than 367 AND this
//object is touching the rocket object
if($this.x() < 367 && $this.isTouching(rocketHit)){
    $this.remove();
    score.text(parseInt(score.text())+1));
}
```

The `&&` tells the if statement not to do anything unless *both* arguments are true. So even if the astronaut is touching the rocket, nothing happens until it is both touching the rocket *and* past the nose of the rocket at x:367.

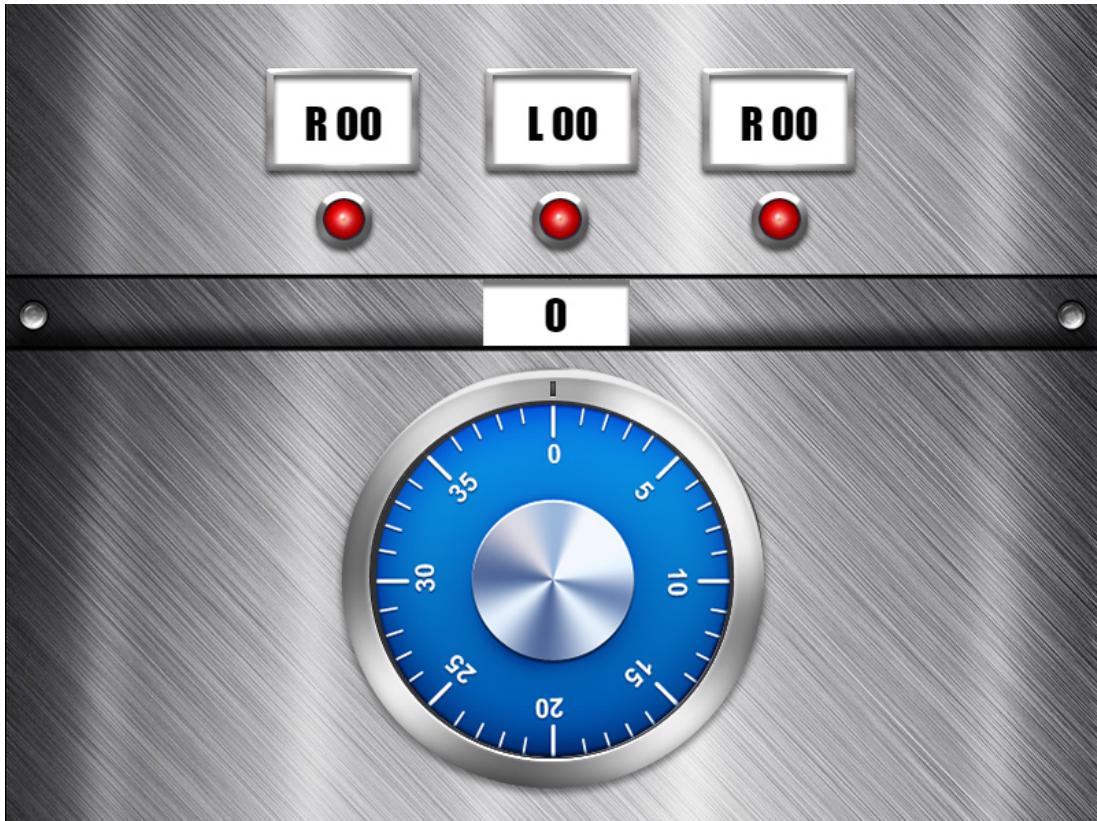
**15** Start the game. See how many astronauts you can rescue!

## Activity 3

# Combination

- 1 Go back to the **Beginning Javascript** path. Open the scene, "Combination" by clicking on the "play" button: 

- 2 Your scene will look like this:



- 3 This game is a bit of an expansion to the "Number Match" game. In this game, you have to spin the wheel in the proper direction (right or left) to match each of the three numbers and unlock the safe. If you get any of the three numbers wrong, you have to start over.
- 4 The first thing to do is randomly select 3 numbers for our combination at the start of the game.

- 5** Make sure the scene ("Combination") is selected either by double clicking anywhere in the scene or by selecting it from the **GAME OBJECTS** menu in the lower left corner. With the scene selected, click on **Events** and **Initialize When Scene Starts** and add this:

```
//make sure the game has started before executing the following code
if($this.scene.state() == "PLAY"){
    //choose a random number between 0 and 39 for
    //matchNumber1
    $this.matchNumber1 = Math.round(random(39,0));
    //choose a random number between 0 and 39 for
    // "matchNumber2"
    $this.matchNumber2 = Math.round(random(39,0));
    //choose a random number between 0 and 39 for
    //matchNumber3"
    $this.matchNumber3 = Math.round(random(39,0));

    //have the text object "match_1" display "R " and the value of
    //matchNumber1
    match_1.text("R "+$this.matchNumber1);
    //have the text object "match_2" display "L " and the value of
    //matchNumber2
    match_2.text("L "+$this.matchNumber2);
    //have the text object "match_3" display "R " and the value of
    //matchNumber3
    match_3.text("R "+$this.matchNumber3);
    //variable "numMatch" is equal to 1
    $this.numMatch = 1;
}
```

```
1 - if($this.scene.state() == "PLAY"){
2     $this.matchNumber1 = Math.round(random(39,0));
3     $this.matchNumber2 = Math.round(random(39,0));
4     $this.matchNumber3 = Math.round(random(39,0));
5     match_1.text("R "+$this.matchNumber1);
6     match_2.text("L "+$this.matchNumber2);
7     match_3.text("R "+$this.matchNumber3);
8     $this.numMatch = 1;
9 }
10 }
```

Creating the random numbers should look familiar from the "Number Match" game. But what about the next 3 lines? We are adding the letter "R" or "L" to our matchnumbers to tell the player which direction to turn the wheel. When we want to combine text and the value of a variable, we simply use a "+" to add them together. When doing this, remember that the program ignores any spaces unless they are inside quotes like this: " "

- 6** In the “Number Match” game, we only had one random number to check for a match and when that match was made, we simply forgot about it and made a new random number. However, we can’t do that in this game. We have to have some way of telling the computer which number the player is attempting to match. Also, we need to know if they are turning the wheel right or left.
- 7** With the scene still selected, click on **Events** and **Update Every Frame**. Here the code used in “Number Match” is still there, ready for us to modify it for the new game. Inside the “if(leftPressed || rightPressed)” statement, add this to the code:

```
//this part of the code is already written
if(leftPressed || rightPressed){

    if(leftPressed){
        plWheel.spin(-40);
        //add this code to set the value of variable "direction" to
        //"L"
        $this.direction = "L";
    }
    if(rightPressed){
        plWheel.spin(40);
        //add this code to set the value of variable "direction" to
        //"R"
        $this.direction = "R";
    }
    //add this code to set the value of variable "turning" to true
    $this.turning = true;

} else {
```

```
4  if(leftPressed || rightPressed){
5
6      if(leftPressed){
7          plWheel.spin(-40);
8          $this.direction = "L";
9      }
10     if(rightPressed){
11         plWheel.spin(40);
12         $this.direction = "R";
13     }
14     $this.turning = true;
15
16 } else {
17
```

When the player presses the left arrow key, the value of "direction" is "L" and when they press the right arrow key, the value of "direction" is "R." Variables can contain words just as easily as they contain numbers. Lastly, when the player presses either arrow key, the value of "turning" is true. Now we know if the player has turned the wheel and what direction they turned it.

**8**

Now we determine if the player has matched the first number. With the scene still selected, click on **Events** and **Update Every Frame**. Below the "} else {" statement is the code we used to find where the wheel was pointing. Beneath that, add this to the code:

```
//this part of the code is already written
} else {

    var wheelNumber = Math.round((360 - plWheel.rotation()) / 9);

    if(wheelNumber === 0){
        lockNumber.text("0");
    } else {
        lockNumber.text(wheelNumber);
    }

    //add this code
    //if the value of "numMatch" is equal to 1, do this
    if($this.numMatch == 1){
        //if the wheelNumber equals matchNumber1 AND the
        //value of "direction" is "R", do the following
        if(wheelNumber == $this.matchNumber1 && $this.direction == "R"){
            //have the light_1 sprite display frameIndex 1,
            //turning it from red to green.
            light_1.frameIndex(1);
            //numMatch is now equal to 2, setting it up for
            //the second number in the combination
            $this.numMatch = 2;
            //turning is now false because the player needs to
            //turn the wheel again
            $this.turning = false;
        }
        //if wheelNumber and matchNumber1 do not match OR
        //the value of "direction" is not "R", do this
    } else {
        //have the light_1 sprite display frameIndex 0,
        //making it red
        light_1.frameIndex(0);
    }
}
```

```

18     var wheelNumber = Math.round((360 - plWheel.rotation()) / 9);
19
20     if(wheelNumber === 0){
21         lockNumber.text("0");
22     } else {
23         lockNumber.text(wheelNumber);
24     }
25
26
27     if($this.numMatch == 1){
28         if(wheelNumber == $this.matchNumber1 && $this.direction == "R"){
29             light_1.frameIndex(1);
30             $this.numMatch = 2;
31             $this.turning = false;
32         } else {
33             light_1.frameIndex(0);
34         }

```

If the value of “numMatch” is 1, then check to see if the wheelNumber is equal to matchNumber1 AND if the value of “direction” is “R” then the player has successfully pressed the right arrow key and stopped it on the correct number. We show that they’ve made a match by turning the light from red to green and advance “numMatch” to 2 so the user can turn the wheel and match the next number.

**9**

We will be using very much the same code for matching the second and third numbers of the combination.

## 10

Beneath the code you just entered, add this:

```
//if the value of "numMatch" is equal to 2 AND "turning" is
//true, do this
if($this.numMatch == 2 && $this.turning){
    //if the wheelNumber equals matchNumber2 AND the
    //value of "direction" is "L", do the following
    if(wheelNumber == $this.matchNumber2 && $this.direction == "L")
    {
        //have the light_2 sprite display frameIndex 1,
        //turning it from red to green.
        light_2.frameIndex(1);
        //numMatch is now equal to 3, setting it up for
        //the third number in the combination
        $this.numMatch = 3;
        //turning is now false because the player needs to
        //turn the wheel again
        $this.turning = false;
        //if wheelNumber and matchNumber2 do not match OR
        //the value of "direction" is not "L", do this
    } else {
        //have the light_1 sprite display frameIndex 0,
        //making it red
        light_1.frameIndex(0);
        //have the light_2 sprite display frameIndex 0,
        //making it red
        light_2.frameIndex(0);
        //numMatch is now equal to 1, forcing the player
        //to start over
        $this.numMatch = 1;
    }
}
```

```
35      if($this.numMatch == 2 && $this.turning){
36          if(wheelNumber == $this.matchNumber2 && $this.direction == "L"){
37              light_2.frameIndex(1);
38              $this.numMatch = 3;
39              $this.turning = false;
40          } else {
41              light_1.frameIndex(0);
42              light_2.frameIndex(0);
43              $this.numMatch = 1;
44          }
45      }
```

## 11 Beneath the code you just entered, add this:

```
//if the value of "numMatch" is equal to 3 AND "turning" is
//true, do this
if($this.numMatch == 3 && $this.turning){
    //if the wheelNumber equals matchNumber3 AND the
    //value of "direction" is "R", do the following
    if(wheelNumber == $this.matchNumber3 && $this.direction == "R"){
        //have the light_3 sprite display frameIndex 1,
        //turning it from red to green.
        light_3.frameIndex(1);
    //if wheelNumber and matchNumber3 do not match OR
    //the value of "direction" is not "R", do this
} else {
    //have the light_1 sprite display frameIndex 0,
    //making it red
    light_1.frameIndex(0);
    //have the light_2 sprite display frameIndex 0,
    //making it red
    light_2.frameIndex(0);
    //have the light_3 sprite display frameIndex 0,
    //making it red
    light_3.frameIndex(0);
    //numMatch is now equal to 1, forcing the player
    //to start over
    $this.numMatch = 1;
}
}
```

```
46
47     if($this.numMatch == 3 && $this.turning){
48         if(wheelNumber == $this.matchNumber3 && $this.direction == "R"){
49             light_3.frameIndex(1);
50         } else {
51             light_1.frameIndex(0);
52             light_2.frameIndex(0);
53             light_3.frameIndex(0);
54             $this.numMatch = 1;
55         }
56     }
57
58 }
```

**12** Start the game. How quickly can you solve the combination?

## Bonus Step

Right now, there's nothing you can do when you match the last number of the combination. What if you could get a new 3 number combination by turning the wheel after completing a combination?

## BEGINNING JAVASCRIPT DAY FIVE

By this time you should have a good understanding of how you can use JavaScript to make all kinds of computer games. Keep in mind that even the most complicated game can be broken down to basic statements and conditions. A computer sees something as either true or false. There's nothing in between.

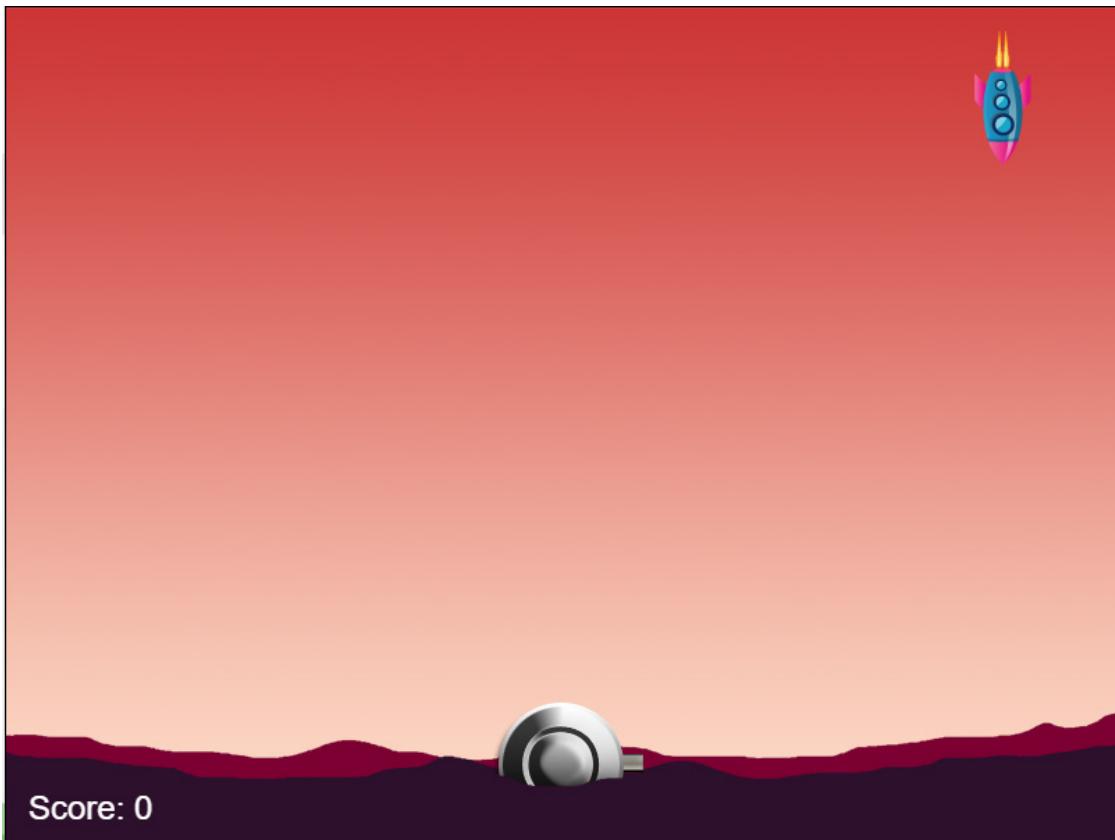
Today's activity has you defending a base against incoming missiles that get faster and faster. Let's get started.



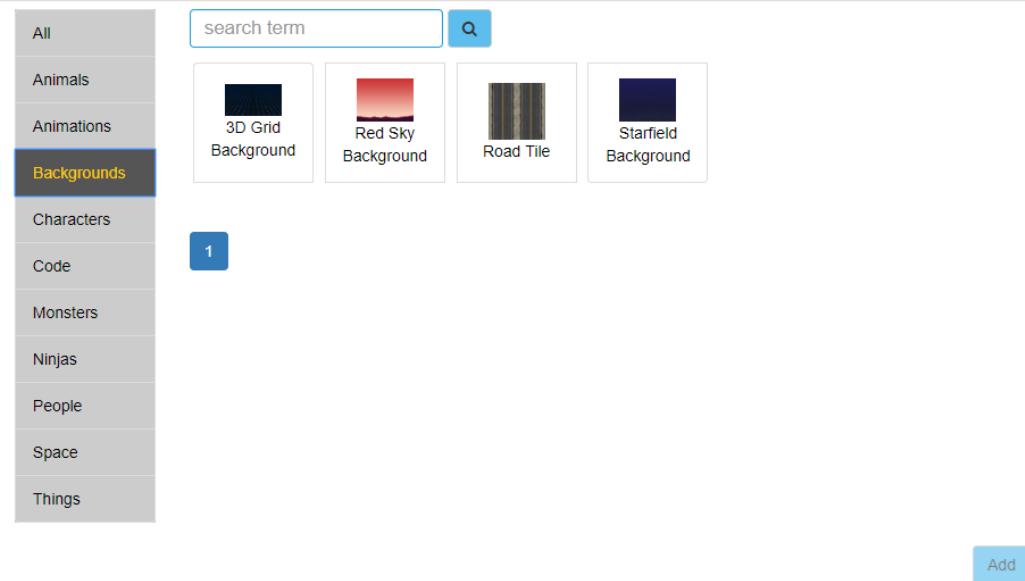
## Activity 1

# Missile Defense

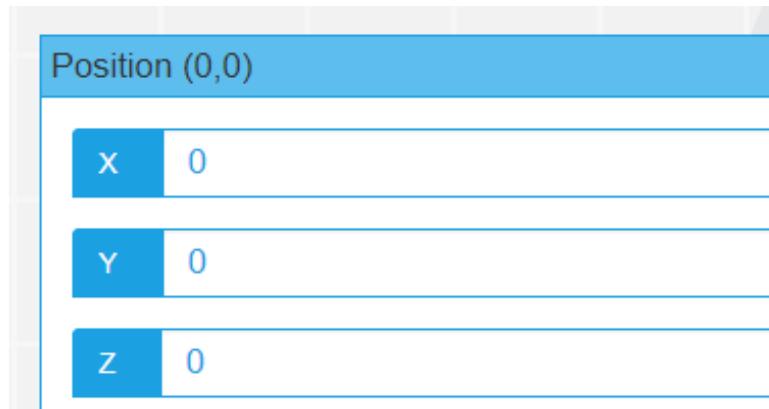
- 1** Select the **Beginning Javascript** path. Open the scene, "Missile Defense" by clicking on the "play" button: 
- 2** When we're done, your scene will look something like the image below. Your goal will be to use the mouse to aim the gun to shoot down the incoming missiles before they hit the ground. But right now, the scene is empty. Let's build our scene.



- 3** Right now, the stage is empty. Let's add our first object. Click on the Search Assets menu  You will see something like this:



- 4** Select the “Backgrounds” menu on the left and search for the “Red Sky Background” and click on “add” and close the Search Assets menu by clicking on the X in the upper right corner. The background image is placed in the middle of the stage.
- 5** The background isn’t meant to be in the middle of the stage. Select the background, select properties and change it so that it is at x:0, y:0, and z:0



By now you should have a good understanding of x and y, but what is z? Imagine that your computer monitor is z:0. Then start stacking things up so that there are more and more things between the computer monitor and your face. The things closest to your face are farthest from the monitor (which is z:0) so they have a higher z value. The background is as far back as we want to stack things so it is z:0. Everything that we want to be stacked in front of the background are z:1, z:2 and so on.

- 6** Next we want to add what we're going to be shooting at the missiles. Click on the Search Assets menu again and this time, choose "Things" from the menu on the left and search for "shot." Click on add to add it to the scene, but don't close the search assets menu yet. We have a couple more things to add.
- 7** Also in the "Things" menu is "Gun Barrel." Add that to your scene and find "missile" and add that to your scene. Finally, search for "Turret" and add that to your scene. Now you can close the Search Assets menu.
- 8** Now you have a bunch of assets in the middle of your stage. Time to move them where they belong. The Turret should be the last object you added, so select that and under properties, change the x to 349 and the y to 500. Leave the z as it is.

Position (349,500)

X 349

Y 500

- 9** The missile is going to be doing a lot of flying around, so where it goes isn't critical. Just drag it to the upper right of the screen so it's out of the way.
- 10** All that's left are the gun and the shot which will be placed behind the turret. Since you added them to the scene before the turret, they should already have a lower z than what the turret has.
- 11** Both the gun and the shot need to be at the same location on the stage. So click either one of them, set that object's properties to x: 400 and y: 543 and then do the same for the other object.

Position (400,543)

X 400

Y 543

**12** If the turret object is not in front of the gun and the shot, then select it and give it a higher value for z.

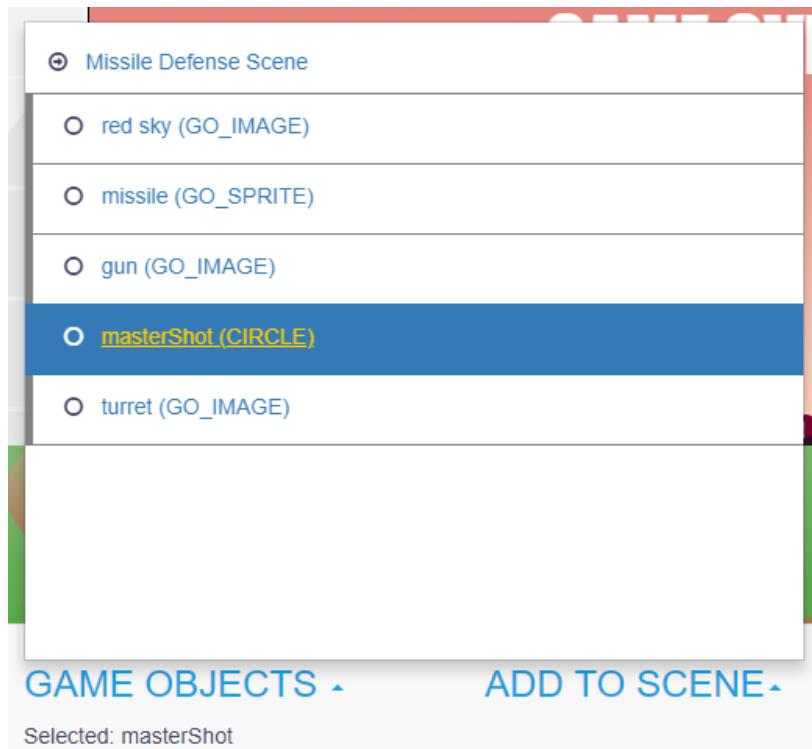
**13** Now it's time to make our game work. Let's start by selecting the gun object. If you have difficulty selecting it from the stage, use the GAME OBJECTS menu to select it. Click on **Events** and **Update Every Frame** and add this code:

```
//change the rotation of this object so it is pointing at the cursor  
$this.pointToCursor();
```

**14** What does that do? **Start the Game** and move the cursor around the stage. The gun barrel follows the cursor as long as it is on the stage.

**15 Stop the Game.**

**16** The gun fires the shot, but we haven't set the code up for that yet. And with the shot behind the turret, it's going to be hard to select. In cases like this, we can use the GAME OBJECTS menu in the lower left corner. Click on it and select "masterShot"



- 17** With masterShot selected, click on **Events** and **Initialize When Scene Starts** and add this code:

```
//hide this object by making it invisible  
$this.visible(false);  
//make sure that the object does not move  
$this.speedX(0);  
$this.speedY(0);  
  
1  $this.visible(false);  
2  $this.speedX(0);  
3  $this.speedY(0);
```

Even though the shot is hidden behind the turret, we are still setting it to hidden because it is the “master” that we are going to make the clones that we are shooting at the missiles from.

- 18** With masterShot still selected, click on **Events** and **Update Every Frame** and add this code:

```
//if this object has an x value greater than 800 OR less than 0 OR a y  
//value greater than 600 OR less than 0, do the following  
if($this.x()>800 || $this.x()<0 || $this.y()>600 || $this.y()<0){  
    $this.remove(); //remove this object  
}  
//if this object's visibility is true, do the following  
if($this.visible()){  
    //move this object forward in the rotation direction it is pointing  
    $this.moveForwardByRotation();  
}
```

```
1  if($this.x()>800 || $this.x()<0 || $this.y()>600 || $this.y()<0){  
2      $this.remove();  
3  }  
4  
5  if($this.visible()){  
6      $this.moveForwardByRotation();  
7  }
```

If this object ever goes beyond the edges of the stage, it is removed. Remember when we set this up to be invisible? When we make it visible it will always move in the direction it is pointing.

**19** We want the gun to fire a shot whenever the mouse button is clicked. Make sure the Scene is selected and under **Events**, choose **Mouse Button Down** and add this code:

```
//make sure the game has started before doing the following
if($this.scene.state() == "PLAY"){
    //the variable "shot" represents a clone of the masterShot object
    var shot = masterShot.clone();
    //use pointToCursor to set the rotation for this clone
    shot.pointToCursor();
    //modify the rotation by subtracting 90 so that the object is
    //actually pointing at the cursor
    shot.rotation(shot.rotation()-90);
    //make the clone visible so it will move
    shot.visible(true);
    //set the velocity of the clone
    shot.speedX(200);
    shot.speedY(200);
}
```

```
1 if($this.scene.state() == "PLAY"){
2     var shot = masterShot.clone();
3     shot.pointToCursor();
4     shot.rotation(shot.rotation()-90);
5     shot.visible(true);
6     shot.speedX(200);
7     shot.speedY(200);
8 }
```

Every time the mouse button is pressed, a clone of masterShot is made, aimed in the direction of the cursor and made visible so that it can shoot forward.

**20** Start the Game and click around the stage to fire the gun.

**21** Stop the Game.

**22** The next step is to set up the missile.

**23** Make sure the Scene is selected, and click on **Events** and **Initialize When Scene Starts** and add this code:

```
//make sure the game has started before doing the following
if($this.scene.state() == "PLAY"){
    //timer function that every 1 second does the following
    createTimer(1000, function(){
        //make a clone of missile and assign it to "newMsl"
        var newMsl = missile.clone();
        //place the clone at random x between 50 and 700
        newMsl.x(random(700,50));
        //place the clone at y:0
        newMsl.y(0);
        //make the clone visible
        newMsl.visible(true);
        //set the clone speedY at 50 plus the value of "score"
        newMsl.speedY(50+$this.scene.score);
        //make the variable "hit" equal to false
        newMsl.hit = false;
    });
    //attach the variable "score" to this scene and make it equal to 0
    $this.score = 0
}
```

```
1 if($this.scene.state() == "PLAY"){
2     createTimer(1000, function(){
3         var newMsl = missile.clone();
4         newMsl.x(random(700,50));
5         newMsl.y(0);
6         newMsl.visible(true);
7         newMsl.speedY(50+$this.scene.score);
8         newMsl.hit = false;
9     });
10
11     $this.score = 0;
12 }
```

When the game is started, createTimer makes a clone of the missile object every one second. The clone is placed at y:0 and between x:50 and x:700. The clone object is made visible and given a speedY of 50 PLUS the current score so that the more missiles are shot, the faster they will go. Each clone has a variable "hit" that is set to false. This is used so that each missile is not hit multiple times by the same shot. Finally, the score is initialized at 0.

**24** Select the missile object, and click on **Events** and **Initialize When Scene Starts** and add this code:

```
//the original missle that is being cloned is hidden  
$this.visible(false);
```

```
1 $this.visible(false);  
2
```

Just like the masterShot object, the original missile is hidden so that it doesn't move during the game.

**25** With the missile object still selected, click on **Events** and **Update Every Frame** and add this code:

```
//update the sprite animation every frame  
$this.incrementAnimation();  
//if this object is visible, do the following  
if($this.visible()){  
    //move this object at speedY (set when it was cloned)  
    $this.moveY();  
    //if this object reaches the bottom of the stage, do this  
    if($this.y()>600){  
        //remove the object  
        $this.remove();  
        //STOP the game  
        $this.scene.state("STOP");  
        //reset ALL timers to stop them  
        $this.scene.cleanupTimers();  
        gameOver.visible(true); //show the game over message  
    }  
}
```

```
1 $this.incrementAnimation();  
2  
3 if($this.visible()){  
4     $this.moveY();  
5     if($this.y()>600){  
6         $this.remove();  
7         $this.scene.state("STOP");  
8         $this.scene.cleanupTimers();  
9         gameOver.visible(true);  
10    }  
11 }
```

A sprite is like several images grouped together. However, the other images are never shown until the sprite gets a command. In this case, incrementAnimation has the sprite cycle through all of the images over and over again. Just like the shot, this object moves only if it is visible. In that case, it will move at the speedY rate set when it was cloned. The game ends if the object reaches the bottom of the stage, so we remove the object, stop the game and reset all of the timers.

**26** Start the Game and you should be able to see the missles fly down.  
Nothing happens when you shoot at them.

**27** Stop the Game.

**28** Select the missile object, and click on **Events** and **Update Every Frame** and add this code beneath the code you just entered:

```
//there are multiple shots on the stage, so we have to check all of  
//them all of the time and we don't know their names. Every shot has  
//a unique "role" of "projectile" so we find out how many are on the  
//stage and check each one  
for(shot=0;shot < $this.scene.findRoles('projectile').length; shot++){  
    //each projectile is assigned to the variable "bullet" and we check them  
    //one by one  
    var bullet = $this.scene.findRoles('projectile')[shot];  
    //only if there is actually a bullet object on the stage do the following  
    if(bullet){  
        //if this object is touching the bullet object AND this object is  
        //visible AND the "hit" value for this object is false, do this  
        if($this.isTouching(bullet) && $this.visible() && $this.hit === false){  
            //this is where we will destroy the missile  
        }  
    }  
}
```

**29** The remaining steps are what to do with the missile if it gets hit by a bullet. So in the part where it says "this is where we will destroy the missile above, add this code:

```
//change the sprite animation to "explode"  
$this.animation('explode');  
//add 1 to the total score  
$this.scene.score += 1;  
//have the scoreLabel text object display "Score: " plus the  
//value of the score variable  
scoreLabel.text("Score: "+$this.scene.score);  
//change this object's "hit" variable to true  
$this.hit = true;  
//remove the bullet object  
bullet.remove();  
//wait half a second for the explosion animation to play  
//and then do the following  
$this.delay = createTimer(500, function(){  
    $this.remove();      //remove this object  
    //this timer is set to false so it is removed immediately  
},false);
```

```

12
13  for(shot=0;shot < $this.scene.findRoles('projectile').length; shot++){
14
15      var bullet = $this.scene.findRoles('projectile')[shot];
16
17      if(bullet){
18          if($this.isTouching(bullet) && $this.visible() && $this.hit === false){
19              $this.animation('explode');
20              $this.scene.score += 1;
21              scoreLabel.text("Score: "+$this.scene.score);
22              $this.hit = true;
23              bullet.remove();
24              $this.delay = createTimer(500, function(){
25                  $this.remove();
26              },false);
27          }
28      }
29  }

```

When the gun is fired, it can make several clones of the shot object and we have no idea which is which. The shot object has the unique role of "projectile" so we know that every projectile on the stage is one of the shot clones. So we tell the computer to identify every instance of "projectile" on the stage and then we check them one by one.

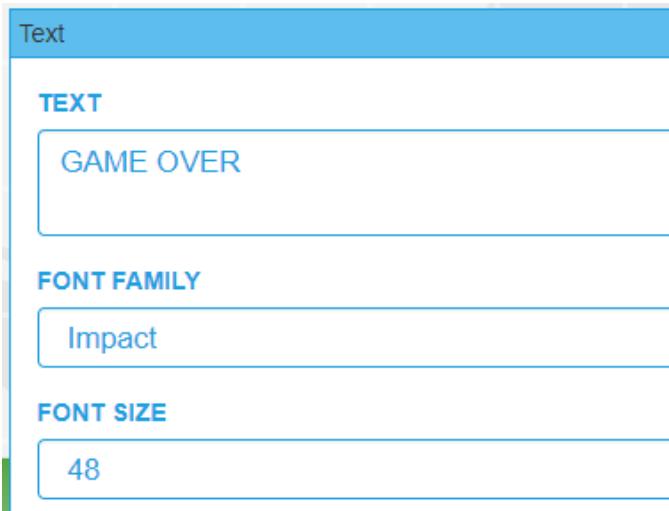
If one of the shot clones hits the missile object and the object is visible and the object has not yet been hit, then we have the missile play its explosion animation for half a second, add 1 to the score and mark the object as "hit."

- 30** You may have noticed that there are a couple of text objects referred to in the code that aren't yet in the scene. Let's fix that.
- 31** Make sure the Scene is selected and at the bottom of the menu, click on ADD TO SCENE and select "Add Text."
- 32** This places a new text object in the middle of the scene. Change the name of the object to "scoreLabel" and change the background color to white. Change the text of this to "Score: 0" and move it to the lower left corner of the stage.

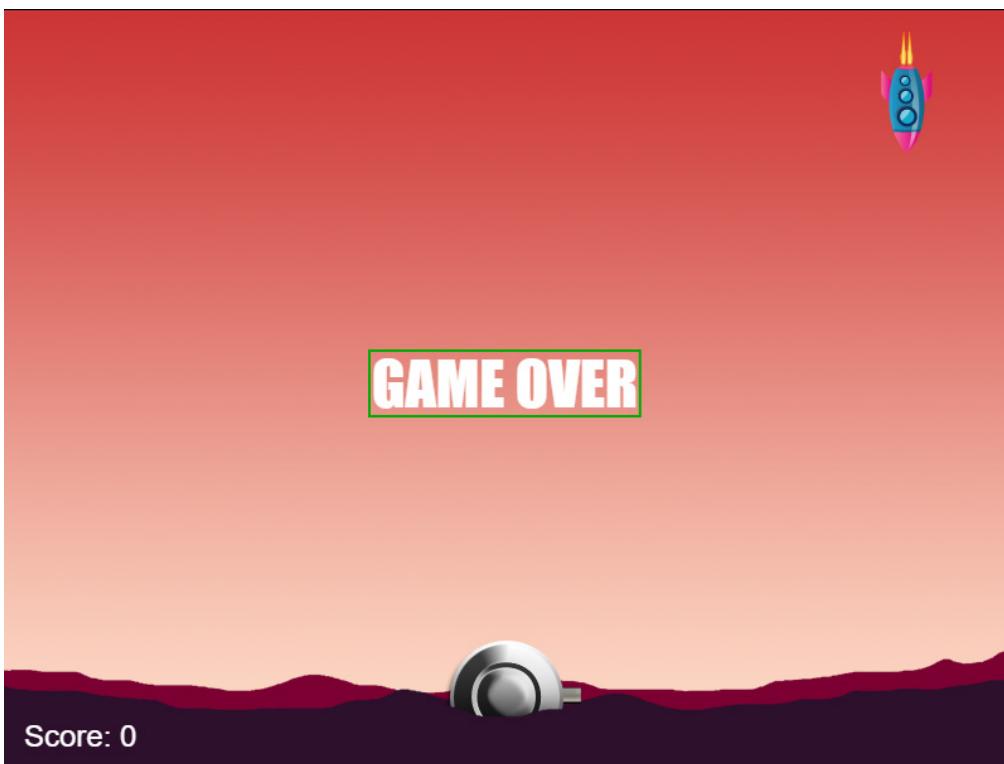
**33** Once again, make sure the Scene is selected and at the bottom of the menu, click on ADD TO SCENE and select "Add Text."

**34** Change the name of this object to "gameOver" and also change the background color to white.

**35** Change the text to "GAME OVER" and change the font family to Impact or any other font you like. Change the font size to 48 so it is big and easy to see.



**36** Move the gameOver object so it is in the middle of the stage.



- 37** Select the gameOver object, and click on **Events** and **Update Every Frame** and add this code:

```
//only do this when the game has started  
if($this.scene.state() == "PLAY"){  
    $this.visible(false);      //hide this object  
}
```

We don't want to display the game over message until the game has stopped.

- 38** Start the Game and defend your base from the incoming missiles. Can you make it to 50 points? To 100?