Project 1.3 Tutorial
EMDA 203

Instructions

1. Download the `Project_1_3_Tutorial.zip` archive from the Moodle site's Project_1_3 assignment section, and unzip the archive.
2. Open your `Users` folder (your P drive) and find your `EMDA_203_Starterkit` folder. Once there, open up the `builds` folder, and delete the old `Project_1_3` folder.
3. Copy the entire unzipped `Project_1_3_Tutorial` folder you just unzipped into your `builds` folder.
4. To avoid confusion, delete the zip archive and the unzipped copy of the new folder from your `Downloads` folder (or wherever you downloaded them).
5. Now, open up `Project_1_3_Tutorial_Code_2020.html` in Chrome and your text editor (Visual Studio Code, NotePad++, etc.). As you work through the project, answer the questions below.

Using the Console Call

Uncomment the "console.log" call at about line 48 and then reload the page in chrome.

- What does this put in the console?
- Why would this be a useful place to put a console.log command in your code?
- What is the overall importance of "console.log" calls in programming in general?

The Importance of Registration

Uncomment the `hero.regX = 64;` and `hero.regY = 64;` lines of code (around line 62).

- What do these lines of code mean?
- Why is the registration property important?

Generating Targets – Overview

Using single lines of code to generate a few targets is fine, but what if we want to generate 100 targets, or 1000 targets? Programming line by line for a large number of targets quickly becomes impractical, so we team up *for loops* and *array* objects. The *for loop* lets us repeat a process a set number of times, and an *array* object lets us store and reference the results of this process.

Generating Targets I

Let's start generating some targets… lots of targets. At about line 55, uncomment the function call, `makeTargets(numberOfTargets);` and reload the page.

- What is the *argument* passed in this function call?
- Knowing this, how can we change the number of targets the function generates?
- How do we change the scale of the targets?
- Why is it a good idea to use variables to control these properties?

Generating Targets II—Parameters and for loops
Go to the definition, `function makeTargets(howManyTargets){ … }` around line 104. What is the *parameter* in this function definition?

On line 106, we use our first *for loop*. A *for loop* lets us repeat a process a set number of times. Often, we will use a *for loop* to *iterate/step through* an array, but more on that later…

The general syntax of a for loop is:
for(*however many times we want something done…* **){** *…execute this block statement* **}**

On line 106, our *for loop* reads:
`for( var i = 0; i < howManyTargets;  i++ ) { … }`

- What does our parameter, `howManyTargets` do here?

Generating Targets III – Getting random
Line 107 has been good clean fun, but you'll have noticed, our targets are all the same. To get some more variety, we'll add some randomness to our lives by using conditionals and the JavaScript Math library. The Math library has two important methods in it: Math.random() and Math.floor(). Math.random() generates a random number between 0 and .99999999999999999, while Math.floor() cuts any decimals off of a number, converting it to an integer. Comment out our old friend, line 107 and *un-comment* lines 108 – 115. Save and re-run. Yay.

- What does the variable whichTarget do? How many different target types are possible here?
- What is the variable stageMargin used for?
- Where is the scale of each target set?
- What is the rollRange() function doing for us here? Where is it defined?
- What is line 132 doing for us? How is the targetsArray used in this line?
- Can you figure out how to change the speed of the targets?

Seeing arrays in action via the console
Un-comment line 133 and reload the page. Look at the console. This shows how the array is populated as we iterate through the for loop.

Comment-out line 133 again, and un-comment line 134. Reload the page and look at the console. This shows us how the array is populated each time through the loop.

Let's move these targets!
In Project 1_2, we wrote a move function that had to be called separately for each target we wanted to move: `moveTarget(target01); moveTarget(target02);` etc… We've replaced that tedious business with a new function, `moveTargets()` defined around line 141. This new and

improved function uses a *for loop* to walk through the `targetsArray` and adjust each target's `.x` and `.y` property based on that particular target's `.speedX` and `.speedY` properties.

- Where is the `moveTargets()` function call? *Un-comment* it, and watch the targets move.
- On line 142, what is the `targetsArray.length` property doing for us?

Understanding the Boundary Checks

Go to the `function targetBoundaryCheck()` definition around line 148. Hey, there's our trusty for loop again, iterating through the ol' `targetsArray`. Good times. Notice that `targetBoundaryCheck()` contains two conditionals that check if a particular target in the array is moving off stage. If it is, it reverses the target's `.speedX` or `.speedY` property to make it "bounce" off of the stage boundary. Now, go check out the `heroBoundaryCheck()` function around line 162. . .

- How is `heroBoundaryCheck()` different from `targetBoundaryCheck()`?
- Where should the function calls to these functions be coded? Go there now and *un-comment* the function calls to see 'em in action!

One last piece of randomness…

Now that everything is bouncing and moving around, let's go back to our `makeTargets()` function definition. Comment-out the four lines beginning with `var coinTossX` around line 127. Save and run the code to see what effect this has on the targets.

- What are the potential values contained in coinTossX and coinTossY?
- How are they used in the two conditional statements?