# Simulations – Assignment03

Loops, conditional statements and functions can all be used together to create simulations. Simulations are representations of events in the real world. But, before we can create a meaningful simulation of any kind, we need to first become accustomed to using all those language elements together in the same assignment.

In this assignment, you will simulate the roll of dice and then use those dice values to generate rectangular shapes of different sizes and colors. This, of course, has limited value in any practical sense. But this assignment is meant to be a learning exercise, to give you some programming experience before you try to do anything more ambitious!

|  |
| --- |
| **Watch out!** We’ll need to use a little CSS and jQuery along without JavaScript to make it all work. Be sure to read the gray boxes like this one. |

## Instructions

1. Download assignment03.zip and put your assignment03 folder into your MIS2402 workspace.
2. Start by editing simulation1.html – this page will simulate the roll of a single die.

## Scenario 1 – Roll a single, six-sided die

|  |  |
| --- | --- |
| There are many kinds of dice. The most common die has six sides.  A roll of a six-sided die should return a random integer between 1 and 6.  This first scenario is really very simple. ***There is no loop in this scenario.*** |  |

1. In simulation1.html observe that a function called getRandomInt() has been defined for you. This function will return a random number between 1 and x. So getRandomInt(3) might return 1, 2, or 3.
2. End the existing click event handler for btn\_1. You will use this to call the getRandomInt function, put the result of that function into variable ( you can call the variable answer if that is helpful). Then, put the answer into the inner html of the existing tag textDisplayed1 with a command similar to the following:

$("#textDisplayed1").html(answer);

|  |
| --- |
| **That’s jQuery.** The statement means, find the HTML tag with the id of textDisplayed1 and put the value of answer into the inner html of that tag. |

1. Test your work. You should see random numbers between 1 and 6 when you click the “Calculate” button. (That was easy, right?)

## Scenario 2 – Simulate the roll two dice at the same time

|  |  |
| --- | --- |
| When you play a board game, you often roll two dice at the same time. In other games, you might roll three or more dice. In this scenario we will extend the work we did in scenario one to simulate a roll of two dice.  Think carefully here. The role of two dice is not a uniformly random number between 1 and 12. Neither is it a uniformly random number between 2 and 12. |  |

Seven is the most likely number when you roll two six-sided dice. This is common knowledge. So, you need to stop here and think how exactly your code will simulate this.

1. Edit simulation2.html – this page will simulate the roll of 2 dice. Complete the roll2Dice()function. This function should use the existing getRandomInt() function and return the sum of the two dice.
2. Now, complete the statCheck() function. The purpose of the function is to determine the average outcome of roll2Dice()function. This will require a loop. Use a loop to get the sum of 1000 rolls of 2 dice. Then divide by 1000. This will be the average value of a 1000 rolls. Return this value.
3. **Test your work.** Because your result depends on random values, there’s no guarantee that you will always get exactly the same result. But, if you have done everything right, your answer should be *approximately* seven.

Here are some sample outcomes when you click the “Stat Check” button.

A screenshot of a game

Description automatically generated A screenshot of a game

Description automatically generated A screenshot of a game

Description automatically generated

|  |
| --- |
| **Now get ready for part 3.**  Here’s where we need to know some CSS. The appearance of an HTML tag (it’s color, its size, etc.) can be specified with CSS attributes. You can apply CSS by specifying the **class** attribute of a tag, or by using the **style** attribute of a tag… or both.  Additional directions will be specified in the instructions that follow.  **Curious about CSS?** If you want to get more information about CSS, you can read about it here:  <https://www.w3schools.com/css/> |

## Scenario 3 – Vary colors

Now we will use our random numbers to vary the color of HTML elements on our page.

1. Edit simulation3.html. In this file you will need to copy/paste in the isNaturalNumber() function that your created earlier in the semester. You should also bring in (that is copy/paste) the roll2Dice() function you created in scenario 2.
2. Take note of the function, getRandomBox() already written for you. It gets a random number that represents a roll of 2 dice. Then it uses that number to return a little bit of HTML.   
     
   The HTML looks like this:

|  |  |
| --- | --- |
| <div **class**="red" **style**="width:2em"></div> | Notice that in our little scenario here, we are using the class attribute to control the color, and the style attribute to control the width. |

1. You job now is to complete the renderBoxes() function. This function should do a few things. If its input ( a variable named n ) is not a natural number, it should place the text “Bad data. Try again.” into the textDisplayed1 html tag. The jQuery command to do that looks like this:

$("#textDisplayed1").html("Bad data. Try again.");

1. If the value of n is good, then the function should execute a loop that iterates n times. With each iteration, you should append the output of getRandomBox into the textDisplayed1tag. The jQuery command to do that looks like this:  
     
   $("#textDisplayed1").append(getRandomBox());

CONTINUED

1. **Test your work.** Your error trapping should work. Your output will, of course, vary randomly. And the number of boxes you see should match the number of boxes you requested.

A screenshot of a cell phone

Description automatically generated

## Scenario 4 – Vary **both** the size and color

1. Now we will use our random numbers to vary the size of HTML elements on our page. You will notice that that start file for simulation4.html is completely empty. You are expected to copy all of simulation3.html and paste it into simulation4.html
   1. EDIT the contents of the <title> tag so that it reads: Simulation - Part 4
   2. EDIT the contents of the <h1> tag so that it reads: Simulation - Part 4
2. Using your knowledge of string concatenation and functions, edit the getRandomBox() function. Your edit should be such that the width of the box varies independently of the color. The width may be 2em, 3em, 4em … all way up to 12em.  
     
   To help you out, lets take a second look at that HTML sample we saw before

|  |  |
| --- | --- |
| <div class="red" style="width:**2em**"></div> | If the value 2em (in bold) was, for example, 8em, we would get a wider box. |

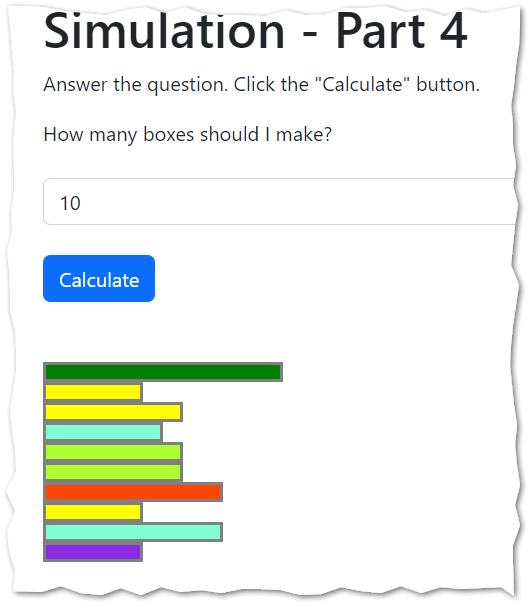
Each “em” is roughly one character in size”. You can read a little bit about “em” as a unit of measure here: <https://www.w3.org/Style/LieBos3e/em.en.html>

**HINT #1:** If, after looking at the above reference, you are still not clear on what **2em** is doing in the code sample above, *maybe you should try it.* Edit the **2em** and change it to **4**em, or **10**em … see what happens.

**HINT #2:** You need to call the roll2Dice() function once to determine the box color, and call it a second time to determine the width of the div tag. Use concatenation to generate the string you want (**2**em, **3**em, **4**em … etc.)

CONTINUED

1. Test your work. Your color and width values will vary randomly.  
     
   You are not expected to rigorously prove this, but a quick casual inspection of your work should suggest that greenish colors are the most common, as are boxes of width seven (because of, you know, math and stuff.)



When you are done…

1. Be sure to upload both all four html files to assignment 3 on canvas.

How will this assignment be graded?

Scenario 1 work is worth 25 points. Scenario 1 needs to run. If it does not run at all you lose all 25 points.

Scenario 2 work is worth 25 points. Scenario 2 needs to run. If it does not run at all you lose all 25 points.

Scenario 3 work is worth 25 points. Scenario 3 needs to run. If it does not run at all you lose all 25 points.

Scenario 4 work is worth 25 points. Scenario 4 needs to run. If it does not run at all you lose all 25 points.

Assuming your work runs – any other point deductions will be assigned in 5-point increments.

Things you could lose points for:

* Missing / incomplete error trapping in scenarios 3 and 4.
* Solving the problem without loops / or with unexpected language elements not taught in class
* Renaming functions
* Miscalculations
* Any bug / error that causes the program return incorrect results