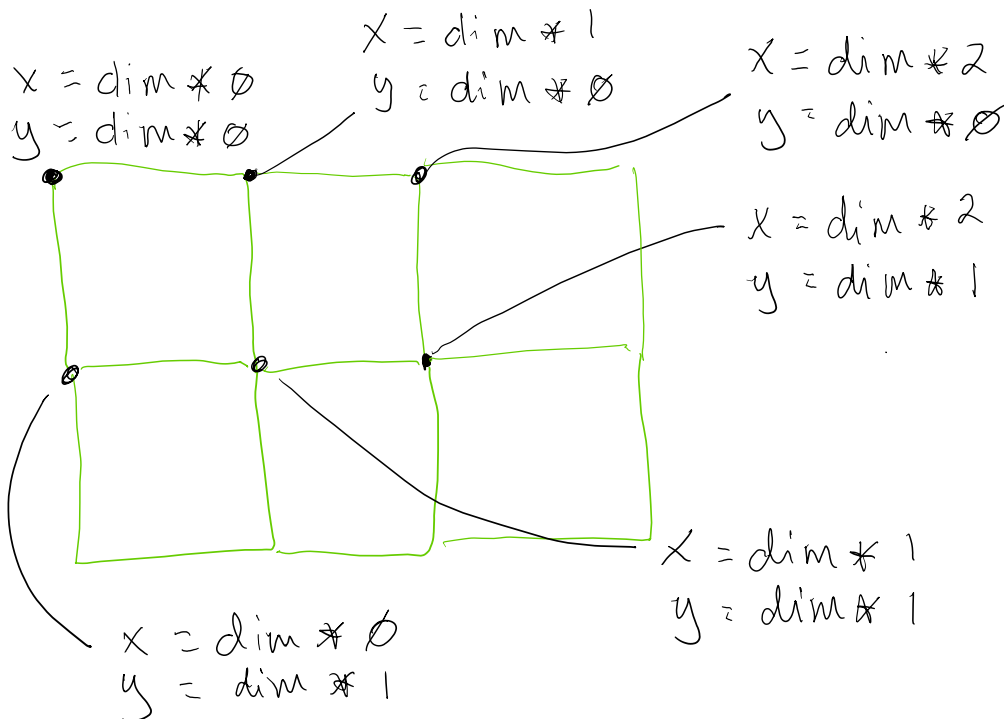# Whack a Monster

Your Tasks (Mark these off as you go)
☐   Write code to create an interactive 2D grid of buttons
☐   Write code to randomly move a monster on your grid
☐   Write code to keep track of whether the monster is clicked
☐   Write code to keep track of the game score
☐   End the game and alert the user of their score
☐   Receive credit for the group portion of this lab

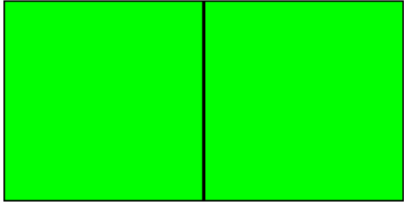## ☐   Write code to create an interactive 2D grid of buttons

In a previous lab, you wrote code to create a 2D grid of buttons. Below we will review how we did this.

First, we created a function that accepted 3 parameters which represented the dimension, x position, and y position of each button. We then declared a variable (dim) to represent the dimension of each button. This new variable allowed us to create many buttons in terms of this dimension. The placement of each button in terms of the dim is illustrated below,

$$x = dim * 1$$
$$y = dim * 0$$

$$x = dim * 0$$
$$y = dim * 0$$

$$x = dim * 2$$
$$y = dim * 0$$

$$x = dim * 2$$
$$y = dim * 1$$

$$x = dim * 0$$
$$y = dim * 1$$

$$x = dim * 1$$
$$y = dim * 1$$

To make our grid interactive required that we add action listeners to each of the buttons. To tell us which button is clicked, we also added an id.
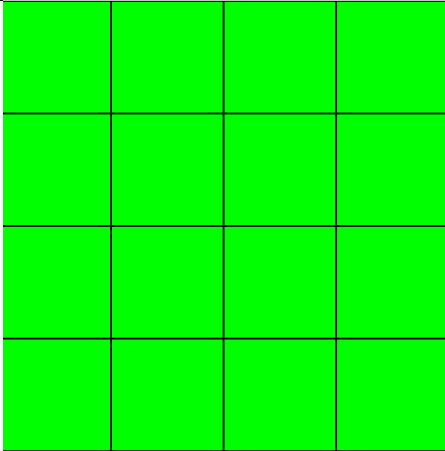
The example below illustrates how to create two interactive buttons using our function. $miss$ is the function called when a button is clicked.

Code	Output

<pre>var dim = 100;
 	
function makeButton(d, xPos, yPos, id){
    b = document.createElement("button");
    b.style.border = "black solid thin";
    b.style.width = d+"px";
    b.style.height = d+"px";
    b.style.backgroundColor = "lime";
    b.style.position = "absolute";
    b.style.left = xPos+"px";
    b.style.top = yPos+"px";
    b.id = id;
    b.addEventListener("click", miss);
    document.body.append(b);
    return b;
}
var b0 = makeButton(dim, dim*0, dim*0, 0);
var b1 = makeButton(dim, dim*1, dim*0, 1);</pre>	

The $miss$ function called when a button is clicked is defined below,

```
function miss(){
    console.log('MISSED');
}
```

Refer to the $makeButton$ function above. Write code that calls the function to create the grid shown below. The position of each button should be defined in terms of dim. The id of each button should also be specified.

Code	Output

☐ Write code to randomly move a monster on your grid

Now that we have a grid of clickable buttons, we can randomly assign (and reassign different buttons to our monster).

The code below creates a timer which executes a function called *moveMonster* every second. The timer has also been assigned to a variable *moveTimer*. Assigning our timer to a variable will allow us to turn it off at a later time. Each time *moveMonster* is called we will want to generate a random location for our monster and color the button red.

```
moveTimer = setInterval(moveMonster, 1000);
```

(a) In the body of the *moveMonster* function below assign a random integer 0 thru 16 (16 not inclusive) to *currentLoc.*
(b) Style the backgroundColor of the button whose id corresponds to the random number to red.

```
var currentLoc;
function moveMonster(){




}
```

So far, our code generates random numbers and changes the background colors of buttons with those numbers. Below is an example of a sequence of timed events that could be used to generate the grid shown.

	time	Pseudocode	Random number
	1	buttonId = Random(0, 16) buttonId.color = "red"	2
	2	buttonId = Random(0, 16) buttonId.color = "red"	9
	3	buttonId = Random(0, 16) buttonId.color = "red"	4
	4	buttonId = Random(0, 16) buttonId.color = "red"	7
	5	buttonId = Random(0, 16) buttonId.color = "red"	14

But how do we make them turn off once they have been turned on? If we know the previous button that was selected, we can turn it off before we color the next button.

Creating a new variable and assigning a random number to this variable will enable us to establish our previous button.

```
var previousLoc = Math.floor(Math.random()*16);
document.getElementById(previousLoc).style.backgroundColor = "red";
```

Now that we have the previous location defined, we have a way of turning it off once a new location has been selected.

In the previous problem, you wrote code to create a random integer 0 thru 16 (16 not inclusive) and used this number to style the backgroundColor of the button whose id corresponded to this number to red.

(a) In the body of the *moveMonster* function below, write code to change the background color of the *previousLoc* button to "lime".

(b) Reassign the variable *previousLoc* to the *currentLoc*.

```
function moveMonster(){
//Add code for part (a) below




//code to assign a random number to currentLoc (completed previously)
//code to style the currentLoc red (completed previously)

//Add code for part (b) below




}
```

☐ Write code to keep track of whether the monster is clicked

During the button creation process we added an action listener to each button.

```
b.addEventListener("click", miss);
```

To keep track of when the monster is clicked we will need to add an action listener to the button that represents the monster. But, this creates a problem. Doing so will cause two competing action listeners to be assigned to the same button (a miss and a hit). We will need to do the following:

1. At the onset of the game, we will need to
   a. register a hit action listener to the *previousLoc* and
   b. remove the miss action listener from the *previousLoc*
2. In the *moveMonster* function, we will need to
   a. Remove the hit action listener from the *previousLoc*
   b. Re-register the miss action listener to the *previousLoc*
   c. Register a hit action listener to the *currentLoc*
   d. Remove the miss action listener to the *currentLoc*

The above logic is illustrated below,
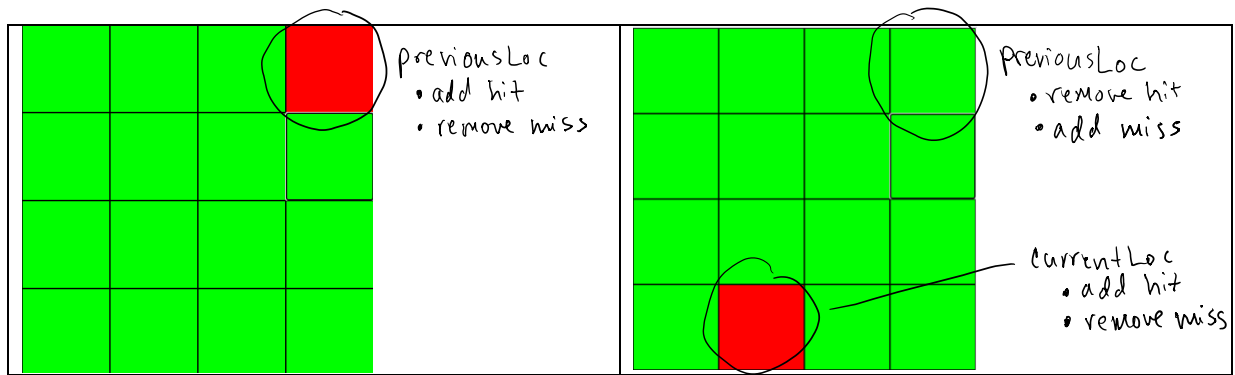
The code below registers a hit action listener to the *previousLoc* at the onset of the game. It also removes the corresponding miss action listener,

```
document.getElementById(previousLoc).addEventListener("click", hit);
document.getElementById(previousLoc).removeEventListener("click", miss);
```

The hit function below is called if the *previousLoc* is clicked.

```
function hit(){
    console.log('HIT');
}
```

Below you will add the necessary code to keep track of whether the monster was clicked,

(a) Remove the hit event listener from the *previousLoc* and add the miss event listener to the *previousLoc*
(b) Remove the miss event listener from the *currentLoc* and add the hit event listener to the *currentLoc*

```
function moveMonster(){

//code to style previousLoc lime (completed previously)

//add code for part (a) below




//code to assign a random number to currentLoc (completed previously)

//code to style the currentLoc red (completed previously)

//add code for part (b) below





//code to reassign the previousLoc to the currentLoc (completed previously)

}
```

□ Write code to keep track of the game score

Now that we know whether the monster is clicked we can keep track of the game score. A global variable *score* will be assigned at the top of our program.

```
var score = 0;
```

When the monster is clicked, we will increment *score* by 1. When the monster is missed, we will decrement *score* by 1.

In the *hit* function below, write code that will increment *score* by 1 when the monster is clicked.

In the *miss* function below, write code that will decrease *score* by 1 when the monster is missed

```
function hit(){
    //add code for part (a) below

}

function miss(){
    //add code for part (b) below


}
```

□ End the game and alert the user of their score

At some point we will need to end our game and alert the user of their final score. To do this will require two things,

1. A timeout timer
2. A function that will
   a. Alert the user of their final score
   b. Turn off the interval timer

Recall that a timeout timer delays the execution of a function. A timeout function will therefore allow us to control when we want our game to end.

The code below illustrates how to create the timeout timer. Below we have defined a global variable, *gameTime*, which represents the length of the game in seconds. Next, we created a timeout timer. The timeout timer accepts two parameters. The first parameter represents the function we want to call after a specified amount of time. The second parameter is the specified amount of time in milliseconds. To convert our *gameTime* to milliseconds we multiply by 1000.

```
var gameTime = 10;
gameTimer = setTimeout(gameOver, 1000*gameTime);
```

To end our game you will need to do the following
   (a) In the *gameOver* function write code to turn off the *moveTimer* – refer to your notes on how to
       stop a timer event
   (b) Alert the user of the final score

```
function gameOver(){



}
```

□ Receive Credit for this lab guide

Submit this portion of the lab to Pluska to receive credit for the lab guide. Once received, your completed code
challenges will also be graded and will count towards your final lab grade.