GD 2
Event Loop

So far, we have used a loop to redraw every game object in their updated (x,y) position. **In this lesson, we will attach an event listener to our website so that it detects a click of the left button of the computer mouse.** We then add this code into our event loop so that the loop redraws game objects in their updated (x,y) position AND can also detect and handle user events.

Game Falling Dots – a single dot falls from the sky and the user must click on the dots to save it. Every time the user saves the dot, the dot will change color, re-spawn in a random x position and then fall faster.

1. Event loop 2. Event listener

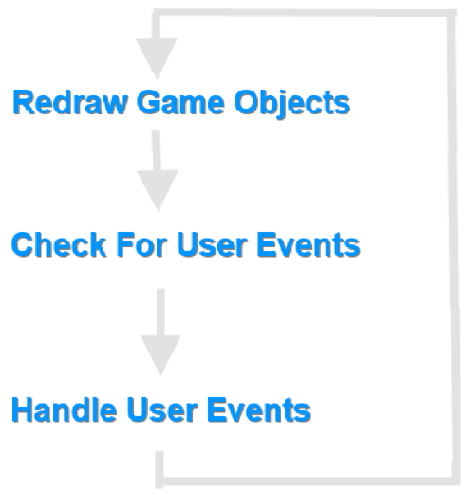
0. EVENT LOOP

So far, we have done animation by redrawing every game object in their updated (x,y) position. Notice that the same function calls itself in order to perform the redrawing.

What if the user pressed a button the keyboard while the computer is redrawing every game object in their updated (x,y) position? We also need to write code to account for user events in addition to drawing events.

Our event loops now contains code that does the following

1. Redraws every game object in their updated (x,y) position
2. Check for user events (like mouse click or keypress on the keyboard)
3. React to user events from step 2



The event loop does this continuously because if it stops then we might miss a redrawing event or a user event. This is why we must follow a simple rule → keep the code short to prevent hanging up the even loop

1. Write the code below in between <script> </script>. The **large, green banner** is your landmark. Go to the coding website and look for it. Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* =========================
* falling rain
*
*
============================ */
cYPos += cYStep;
```

Explanation

Remember that there are two coordinates that describe the position of a game object and they are the **x position (side to side)** and the **y position (up and down).** We want our circle to fall from the sky so we add onto the current y position.

The variable `cYStep` holds how fast the dot will fall from the sky.

2. Write the code below in between <script> </script>. The **large, green banner** is your landmark. Go to the coding website and look for it. Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* =======================
* mouse click
*
*
========================= */
window.addEventListener( "click", check_coord );
```

Explanation

Remember in JS 4 where we learned about events and event handler. We wrote the following

```
<p onclick = "goDown()" > Down </p>
```

The event was **onclick** and the event handler (ie. the reaction) was the JS function definition **"goDown()".**

1. **The code above was HTML**, we attached an event **onclick** to a paragraph tag.

JS can do event and event handler. The code in **blue** says the entire website will **listen for a mouse click.** When the user clicks on the left button of the mouse, the reaction is to call the function `check_coord();`

3. Write the code below in between <script> </script>. The **large, green banner** is your landmark. Go to the coding website and look for it. Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ====================
* check coord
*
*
* ====================== */
function check_coord( eData )
{
      mouseClckFlag = 1;
      mouseX = eData.offsetX;
      mouseY = eData.offsetY;

      console.log( "mouseX: " + mouseX + ", mouseY: " + mouseY );

}
```

Explanation

The function definition above is the **event handler (ie. reaction)** and is called when the user "**clicks**" on the left button of the mouse.

The parameter **eData** has information about the keyboard and mouse button. We load the x and y coordinate of the mouse onto the variable **mouseX and mouseY.**

CONTINUE TO THE NEXT PAGE

4. Write the code below in between <script> </script>. The **large, green banner** is your landmark. Go to the coding website and look for it. Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ============================
* check if user clicked on mouse
*
*
* ==============================*/
if( mouseClckFlag )
{
        mouseClckFlag = 0;

if( cXPos-cRad <= mouseX && mouseX <= cXPos+cRad && cYPos-cRad <= mouseY && mouseY <= cYPos+cRad )
        {
                console.log( " ----------- split the atom ----------- " );
                cColor = "#" + Math.floor( Math.random() * 656655 );

                cXPos = Math.floor( Math.random() * 500 );
                //cYPos = Math.floor( Math.random() * 500 );
                cYPos = Math.floor( Math.random() * 10 );
                cYStep++;
        }
}
```

Explanation

In the game falling dots, **we use the computer mouse to click on the dots before they touch the ground.** Every time we click on the exact x and y coordinate of the falling dot, the dot will reset at the top and fall in a random position.

In order to save the falling dot, the x and y position of the computer mouse must be within the hit box of our falling dot and the long "**if**" statement does the hit box check.

If the x and y position of our computer mouse is within the hit box of the falling dot, then we do the following:

1. Add one point to the player's score
2. change its color to a random color
3. reset the falling dot to the top and then position it at a random position on the x axis
4. increase the step size on the y axis to increase the speed of the falling dot

HTML Challenge

Look for the green banner

```
<!-- ===================================
* Player score HTML
* ===================================== -->
```

Create a **paragraph tag** and give it an id of "**p1Score**"

JS Challenge 1

1. Look for the green banner

```
/* ============================
* Player score
*
*
* ============================ */
```

and put code underneath it. Create a variable called **playerScore**. Use the assignment operator to load the data 0 into it.

JS Challenge 2

When the user clicks on a falling dot, we want to give the player 1 point and make the dot smaller. In order to do this, we must find the code that takes care of the hit box. Find that code and do the following

1. link JS code with the HTML element whose id is "**p1Score**"
2. user **.innerHTML** to show the updated score onto the website
3. decrease the radius of the circle by subtracting 1 from the variable **cRad**