GD 19
Eye of the Cyclops, Part 2

We continue to learn more about classes and objects. Remember that a class is a generic template while objects are copies of the generic template. We use a class definition to define private attributes and private functions to create custom game objects.

Eye of the Cyclops. Take down the eye by shooting lasers at it. Match the parachute with the slogan to receive more lasers. A match that is off will cause the parachutes to turn into wheel spikes and become hazards. Use the left and right arrow keys to move your character left and right. Click on the left button of the mouse to shoot lasers.

1. class definition 2. Constructor 3. "**this.**" versus object name

---

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.**

```
/* ========================================
* Player & Cyclopes
* ======================================== */
var playerObj = new Player( 350, 450, "#0079D1", 0, 0, 'j', 50, 50 );
playerObj.load_name_and_image( "Vuong", "../images/dot_spider.png" );
playerArr.push( playerObj );

playerObj = new Player( 350, 150, "#0079D1", 0, 0, 'j', 100, 100 );
playerObj.load_name_and_image( "Cyclops", "../images/cyclops_eye.png" );
playerArr.push( playerObj );
```

Explanation

The code above uses the **"new"** operator to create an object of class Player. Remember that a class is a generic template and is used as an original copy. If we want to make multiple copies of a class, we use the **"new"** operator and each copy is called an object.

Next, notice that the function **load_name_and_image()** is a **private function**. How can we tell? **The member access operator**, which is the period, **shows ownership**. To the **left** of the member access operator is the **owner** while to the **right** of the member access operator is **what is being owned.**

This would mean that playerObj is the owner and **playerObj owns load_name_and_image();**

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.**

```
/* ======================================
*     Create QA Object
* ====================================== */
var qa = new QA( "Impossible is nothing", "Adidas" );
allQAArr.push( qa );

qa = new QA( "Stronger than dirt", "Ajax" );
allQAArr.push( qa );

qa = new QA( "Belong anywhere", "Airbnb" );
allQAArr.push( qa );

qa = new QA( "I can't believe I ate the whole thing", "Alka Seltzer" );
allQAArr.push( qa );

qa = new QA( "You're in good hands", "Allstate" );
allQAArr.push( qa );
```

Explanation

We can use the "**new**" operator on different classes to create different types of objects. In step 2, we use the "**new**" operator on class Player to create multiple player objects.

In the code above, we are using the "**new**" operator on class QA to create multiple question and answer objects. We are customizing each QA object by putting input data in between the open and closing parenthesis.

Finally, we add the new question and answer object into an array by using the private function .push() of the array.

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.**

```
/* ================================================
* Clear and Redraw
* ================================================ */
cContext.clearRect(0, 0, cObj.width, cObj.height);
sampleCount++;
cContext.drawImage(img, clipX, clipY, clipW, clipH, imgX, imgY, imgW, imgH);
```

Explanation

Remember that our game has a refresh rate of 60 hz. This means that every 16.67 milliseconds, the content on the screen is cleared and then we have to redraw the game objects in their new position. The end result is animation.

The code above clears the screen, increases the sampleCount by 1, and then draws the background images.

JS Challenge 1. Look at the code in red and answer the following questions

1. What is the member access operator?
2. What is the name of the owner?
3. What is being owned?

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.**

```
/* ========================================
* User Laser
* ======================================== */
cContext.beginPath()
cContext.strokeStyle = "#ffffff";
cContext.lineWidth   = "3";

var pObj = new Path2D( "M " + playerArr[0].pXPos + " " + playerArr[0].pYPos + " L " + mouseXEnd + " " + mouseYEnd );

cContext.stroke( pObj );
```

Explanation

In order to bring down a parachute and prevent it from being a spiked wheel, the user must be able to control the direction of the laser bring down the parachute. The code above creates a new path2D object by using the "**new**" operator. We then **customize the object** by putting input data inside the open and closing parenthesis.

The laser's starting position is the x and y position of our game character. This means that the starting position will always be the x and y position of the game character.

The ending position of the laser will change based on where we position the mouse cursor. This is important because putting the starting position at a fixed location prevents the laser will floating in the game and it helps us aim the laser at the parachute. The variables **`mouseXEnd`** and **`mouseYEnd`** hold the x and y position of the mouse.

Since we are creating a custom path, we need to use a custom draw function. The **`.stroke()`** is the custom draw function.

NOTE: we are simply drawing a laser but we haven't released the laser yet. The next JS Challenge will release the laser.

JS Challenge 2. Write the code on JS Fiddle

```
/* ================================================
* JS Challenge 2 - Make A Laser
* ================================================ */
```

1. create a variable called **zLObj**
2. use the "**new**" operator to create an object of type **class ZipLine**
3. the inputs to the class definition to customize the object are
    - **3.1. playerArr[0].pXPos**
    - **3.2. playerArr[0].pYPos**
    - **3.3. mouseXEnd**
    - **3.4. mouseYEnd**
    - **3.5. 3**

JS Challenge 3. Write the code on JS Fiddle. Write "for" loop of user's laser. The array has name zipLineArr

```
/* ========================================
* Check if user's laser touched eye
* ======================================== */
```

1. Declare a digital key named **z** and load the data **0** into it
2. Check if **z is less than the length of the array**
3. The variable **z** should jump by 1 position

5. 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.**

```
/* ================================================
* Bounce Effect
* ================================================ */
if( playerArr[1].pXPos <= 0 || playerArr[1].pXPos >= 900 )
{
      motionXDir *= -1;
}
```

Explanation

The Cyclop's eye must move around to prevent being touched by your laser. The Cyclop's eye will change positions randomly and this makes the eye look like it is dancing. We must also prevent the eye from going through the left and right wall of the game (that just looks weird).

The code above **check's the x position of the eye and if it is less then 0,** then the eye is outside the left boundary. To prevent the eye from going through the wall, the code changes the direction by multiplying by -1 (multiplying by 1 causes the eye to go in the opposite direction).

It is important to remember that the change in direction does not happen all the time. Why? Because the "**if**" statement is our guard and **it guards the code inside the curly graces**. The code inside the curly braces only executes **IF** the condition is true.

The code above also uses the **logical OR operator (||)** to change direction when the eye is outside the left **OR** right boundary. The logical **OR** operator means that if one of the conditions is true, then we execute the code inside the curly braces.

JS Challenge 3 – Prevent going through floor and ceiling

```
/* ================================================
* JS Challenge 3
* ================================================ */
```

1. Check if the eye's **y position** is less than 10 **OR** greater than 200.
2. If one **OR** the other condition is true, make the eye change its **Y direction**