GD 18
Eye of the Cyclops, Part 1

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.**

```
/* ================================================
* Color code arr
* ================================================ */
var
colorVArr = [ "#006400", "#fa99ff", "#d69D70", "#3d6f87", "#06a578", "#f8a894",
"#bcbcbcb", "#1f3a56", "#c2a6ba" ];
```

Explanation

The code above makes colorVArr an array because it uses the square brackets. Remember that the starting position of an array is position 0 and inside **position 0** is the data "**#006400**";

JS Challenge 1 – you can write the answer inside the Zoom Chat

1. What data is inside position 4?
2. What data is inside position 2?
3. What data is inside the last position?

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.**

```
    /* ============================================
    * Dot class
    * ============================================ */
    class Dot
    {
      constructor( xStart = 0, yStart = 0, xO = 1, yO = 1, sRad = 5, fillC =
"#ffd700", strokeC = "#c0c0c0" )
        {
            this.xPos     = xStart;
            this.yPos     = yStart;
            this.xOffset  = xO;
            this.yOffset  = yO;
            this.rad      = sRad;       // -- serves as point amount and id
            this.fillC    = fillC;      // -- serves as point amount and id
            this.strokeC  = strokeC;
            this.stableCount = 201;
```

Explanation

The code above is the class definition of Dot, which are the parachutes that start at the top and come down.

The construct is important because it is used to initialize the objects (ie. the clones) that are created from the class (ie. the model). The constructor has many inputs and these inputs give us the capability to customize the objects. The inputs that are used to customize the object are positioned in between the open and closing parenthesis.

Inside the body of the constructor, we are creating many **private variables**. For example, `this.xPos, this.yPos, this.xOffset,` and etc. We know that these are private variables because of the "`this.`" in front of the name of the variable.

Remember that the **period** is called the **member access operator** and it shows **ownership**. To the **left** of the period is the **owner** and to the **right** of the period is **what is being owned**.

This would mean that "**Dot**" is the owner because "**this.**" is a shortcut for Dot.

Next, Dot owns **xPos, yPos, xOffset**, and etc because they are on the right side of the assignment operator.

We then load the input into the private variables using the assignment operator.

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.**

```
/* ========================================
* Drop One Offset
* ======================================== */
drop_one_offset()
{
	this.yPos += this.yOffset;


}
```

Explanation

The function definition above is a private function (ie. private action) that belongs to class Dot. We know that it is private because the function definition is **inside the curly** braces of class Dot.

Inside the body of the private function definition, we are increasing the private variable yPos by the amount stored inside the private variable yOffset. The effect of doing this is that the parachute will drop one step and gets closer to the ground.

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 hit box
* ======================================== */
check_hb( oXPos, oYPos )
{

	// -- calculate hit box for horizontal
	var hBox = ((this.xPos-this.rad) < oXPos) && (oXPos < (this.xPos + this.rad));

	// -- calculate hit box fo vertical
	var vBox = ((this.yPos-this.rad) < oYPos) && (oYPos < (this.yPos + this.rad))

	// --
	if( hBox && vBox )
	{
		console.log( " {{{{{{{ you got the hitbox }}}}}}} " );
		return 1;

	}
	else
	{
		return 0;
	}

}
```

Explanation

The player should be rewarded with a point when the laser hits the correct slogan. In our style of writing code, we must manually check that there is a collision between two or more game objects. Each game object has a hit box and when two game objects enter each other's hit box, then we have collision. The code above checks if the laser has entered the hit box of a single parachute. The inputs into the functions are in between the open and closing curly braces and they are oXPos and oYPos.

We use these two inputs to check if for a collision. The long if statement does two things

1. checks if the x position of the laser is in between the x position of the parachute.
2. Checks if the y position of the laser is in between the y position of the parachute.

If the laser has entered the vertical **AND** horizontal hitbox of the parachute, then the laser has collided with the parachute. We notify the game by giving back a 1 to indicate "hit". In coding, we use the word "**return**" to give back data.

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.**

```
/* ===================================
* Create Dot
* =================================== */
var dot_1 = new Dot( 50, -40, 1, 1, 50 );
dotArr.push( dot_1 );

var dot_2 = new Dot( 310, -40, 1, 1, 50 );
dotArr.push ( dot_2 );

var dot_3 = new Dot( 150, -40, 1, 1, 50 );
dotArr.push ( dot_3 );

var dot_4 = new Dot( 450, -40, 1, 1, 50 );
dotArr.push ( dot_4 );
```

Explanation

In step 2, we created the class definition of Dot. **A class is the original copy of a game object.** What if we wanted to create multiple copies of that same game object? **These copies are called objects** and we use the "new" operator to create copies.

In the code above, we are creating 4 objects of the Dot class and then customizing each one by putting inputs inside the open and closing parenthesis. Next, we use the .push() to put each object into an array named dotArr.

6. 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.**

```
/* ======================================
* Dot loop
* ====================================== */
for( var b = 0; b < dotArr.length; b++ )
{
    cContext.strokeStyle = dotArr[b].strokeC;
    cContext.fillStyle   = dotArr[b].fillC;
    cContext.beginPath();

    cContext.drawImage( dotArr[b].costumeImage, dotArr[b].xPos, dotArr[b].yPos, dotArr[b].rad, dotArr[b].rad );

    cContext.stroke();
    cContext.fill();


}
```

Explanation

In step 5, we are creating 4 objects of the class Dot and they become our parachutes that start at the top and drop one step until touching the ground.

The code above uses a for loop to select each one of the parachutes and then uses the private variables to draw the parachutes at their starting position at the top of the screen.

Remember, we are now **outside** of the class definition. So, we can't use "**this.**" anymore. When we are on the **outside** of the class definition, we **say the name of the object** in front of the member access operator.

JS Challenge 2 – You can answer the questions inside Zoom Chat

1. What is a class?
2. Why is the constructor needed?
3. What is the period called? What does it do?
4. What does "this." mean?
5. What is an object?
6. How is an object different from a class?
7. What operator is used to make an object?

JS Challenge 3 – You can answer the questions inside Zoom Chat

Assume the array exists, `var letterArr = [ 'v', 'z', 'y', r', 'o', t', 'm', 'c' ];`

1. What data is in position 3?
2. What data is in position 5?
3. What is the length of the array?
4. Load the data 'j' into position 4
5. Load the data 'h' into position 5
6. Declare a digital key named u and load the data 3 into it
    a. Load the data 'z' into position of u
    b. Increase the variable u by 2
    c. Load the data 'L' into position u