GD 16
Feed The Fish, Part 2

In part 1, we learned that a class is a generic description of something. Inside the constructor of the class definition, we use the **member access operator,** which is the period, to show ownership of private variables and actions. Remember that to the **left** of the member access operator is the **owner** and to the **right** of the member access operator is **what is being owned**.

```
// -- object at position 0 owns the action drop_one_offset();
dotArr[0].drop_one_offset()
```

What if we wanted to make multiple copies of the class? We can use the class as the original and make copies of the class by using the "**new**" operator. These copies of the original class are called **objects.**

In this part 2, we will combine array manipulation with classes and objects to loop through and update the private variables of each object. This will allows us to make the game objects move.

Feed The Fish is a game where the players must route the food to the correct fish. The food is of different color and the challenge is to route the correct food to the same colored fish. We feed the fish by drawing ramps that must slope in the downward direction. These ramps can be combined together to create a maze that will route the food to the same colored fish.

CONTINUE TO THE NEXT PAGE

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.**

```
/* =======================================
* New Ramp
* ======================================= */
if( newLF )
{
    newLF = 0;

    // -- starting position is down below in the middle
    var zLObj = new ZipLine( mouseXStart, mouseYStart, mouseXEnd, mouseYEnd, 3 );

    zipLineArr.push( zLObj );


}
```

Explanation

The code above creates an object by using the "**new**" operator. The class name is **ZipLine** and the object name is **zLObj**.

The input arguments to create the object are `mouseXStart, mouseYStart, mouseXEnd, mouseYEnd, 3`.

a. The first two input arguments are the starting x and y position of the ramp
b. The next two input arguments are the ending x and y position of the ramp
c. The last input argument, 3, is the thickness of the ramp

The paragraph above creates a clone (ie. creates an object) and the input arguments make the object unique.

Finally, we put the ramp into the array named zipLinArr by using **.push().**

CONTINUE TO THE NEXT PAGE

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.**

```
/* ================================
* Draw Ramp
* ================================= */
for( var z = 0; z < zipLineArr.length; z++ )
{
        cContext.beginPath();
        cContext.lineWidth   = "5";
        cContext.strokeStyle = "#006500";

       // -- the code below should be on a single line
        var pObj = new Path2D( "M " + zipLineArr[z].xStPos + " " +
zipLineArr[z].yStPos + " L " + zipLineArr[z].xEnPos + " " + zipLineArr[z].yEnPos
);

       // -- the code below should be on a single line
        cContext.drawImage( cursorImg, zipLineArr[z].xStPos,
zipLineArr[z].yStPos, 50, 50 );

        cContext.stroke( pObj );
      }
```

Explanation

In step 1, we created a ramp object by using the “new” operator. Remember that in modern video game development, we also have to draw or render the game object onto the canvas.

If the game player wanted to route a specific food color to a food, the game player will use many ramps to route the food. Everyone of these different ramps are stored into the array named zipLineArr. We will loop through the array to draw the ramp onto the canvas.

The code in **blue** will first connect the starting (x,y) point with the ending (x,y) point to create a line. The next code in **blue** will then draw the icon to show the starting point of the ramp.

Finally, the code in **green** will draw the ramp. For each item in the array, the code will draw a ramp for that item. So, if the player created 10 ramps, then the **.length** is equal to 10 and the loop will run 10x times.

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.**

```
/* ========================================
* Collision/interference detection
* ======================================== */
for( var b = 0; b < dotArr.length; b++ )
{
     for( var z = 0; z < zipLineArr.length; z++ )
     {
            // -- downward, right direction
            if( dotArr[b].check_hb( zipLineArr[z].xStPos, zipLineArr[z].yStPos,
zipLineArr[z].xEnPos, zipLineArr[z].yEnPos ) )
            {
                dotArr[b].load_laser_profile( zipLineArr[z].return_profile() );

            }

            // -- downward, left direction
            if( dotArr[b].check_hb_2( zipLineArr[z].xStPos, zipLineArr[z].yStPos,
zipLineArr[z].xEnPos, zipLineArr[z].yEnPos ) )
            {
                dotArr[b].load_laser_profile( zipLineArr[z].return_profile() );

            }


      }

}
```

Explanation

The code above has a loop inside a loop. Notice that the green banner has the word "`Collision/interference detection` ". When the game starts, the dots (**the dots are the colored fish food**) are at the top of the screen and fall down.

What if any of the dots touch a ramp? **It could be any dot that touches the fish food**. In order for us to find out which dot touched a ramp, we would have to loop through the entire dotArr by starting at position 0.

Next, we have to check if the dot at position 0 has touched any of the ramps → **the dot at position 0 could have touched any ramp**. So, in order to find out if the dot at position 0 has touched any of the ramps, we have to loop through the entire **zipLineArr** by starting at position 0.

If we find that a dot has touched a ramp, then the next step is to determine the direction of the ramp. There are two directions, downward right or downward left, and that is why two "if" statements are needed.

a. If a dot touched a ramp that has direction downward right, then the first "if" statement is true and the dot travels in the downward right direction.

b. If a dot touched a ramp that has direction downward left, then the second “if” statement is true that the dot travels in the downward left direction.

JS Challenge 1 – Array Review

This JS Challenge can be completed by writing the code on the website https://www.w3schools.com/code/tryit.asp?filename=GD76D4FQPT8Q

1. Above the function main(), declare an array named **myLetter**. The right side of the array has the data [a, a, b, z, h, m, e, w, p, y, u, q, w, n]

2. Inside the function main(), do the following

   a. Make a digital key named “**u**” and load the data 0 into it. The digital key will be used to write the “for” loop
   b. Write a “for” loop starting at position 0, ending at the length of the array, and jump by 2

3. Inside the **body of the “for” loop**, do the following
   a. declare a variable named **currLetter**.

   b. Next, use square brackets and the digital key named “**u**” to select one position of the array. Load this data into the variable **currLetter**

   c. Send an alert with **currLetter** inside the parenthesis of the alert

JS Challenge 2 – Calculate Average

This JS Challenge can be completed by writing the code on the website https://www.w3schools.com/code/tryit.asp?filename=GD76D4FQPT8Q

1. Above the function main(), declare an array named **classGrades**. The right side of the array has the data [5, 3, 6, 9, 10, 2, 0, -5, 6, 3]

2. Inside the function main(), do the following

   a. Declare a variable named **sum** and load the data 0 into it
   b. Make a digital key named “**c**” and load the data 0 into it. The digital key will be used to write the “for” loop
   c. Write a “for” loop starting at position 0, ending at the length of the array, and jump by 1

3. Inside the **body of the “for” loop**, do the following

   a. Use square brackets and the digital key named “**c**” to select one position of the array. Next, the data at the position should be added to the variable **sum**

4. Outside of the “for” loop, calculate the average. In order to calculate average, the equation is

   Average = **sum** / **amount of items in the array classGrades**

   a. You already have **sum**.
   b. The **amount of items in the array classGrades** is the length of the array

5. Send an alert of the average