GD 15
Feed The Fish, Part 1

So far, we have organize game objects into an array and this allows us to use a “**for**” loop to update the x,y position of the game objects. Each array that is used can have different lengths and the specific length of an array can be found by writing **.length** at the end.

For example, the length of dotArr can be found by writing dotArr**.length.** The next game combines array manipulation with classes and creating objects.

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.

1. Class definition 2. **new** operator 3. Creating objects

---

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 = [ "#006500", "#0056ff", "#ffd700", "#fa5500" ];
```

Explanation

The food for the fish starts at the top of the screen and then drops down. Each food is of different color and the inside color is the color that we should focus on. The code above uses **square brackets on the right side** of the assignment operator, which means that it is an array.

The first position of an array is always 0 and the data at position 0 is #006500. What is the data at position 1, position 2, and position 3?

You are free to customize the colors so that the food coloring is different. Use the website below to pick 4 colors and use them. https://en.wikipedia.org/wiki/Web_colors

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.**

```
/* ====================================
* Initialize priviate variables
* ==================================== */
this.xPos     = 300;
this.yPos     = index * -55;
this.xOffset  = 0;
this.yOffset  = .5;
this.rad      = 7;
```

Explanation

The code above is inside the constructor of the class definition "class Dot". The constructor is used to initialize the private variables of the class Dot. We know that the variables are private because of the world "this" followed by the period " . " For example,

```
this.xPos = 300;
```

Remember that the **period is called the member access operator** and it shows ownership.
1. The **left side** of the period is the owner
2. The **right side** of the period shows what is being owned

In our case, this means that "**Dot**" is the owner and Dot owns **xPos**. The code above sets our Dot's x position to be 300 and radius to be 7.

JS Challenge - Add onto the class definition and add the following private variable. Remember that private variable use "`this.`"

Look for the green banner below and put the code under the banner

```
/* ====================================
* JS Challenge 1
* ==================================== */
```

1. private variable named **description** and use the assignment operator to load the data "solid food"

   For example,

   ```
   this.description = "solid food";
   ```

2. private variable named **type** and use the assignment operator to load the data "fish food"
3. private variable named **cost** and use the assignment operator to load the data "3.50"
4. private variable named **breed** and use the assignment operator to load the data "fresh fish"

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.**

```
/* ==============================================
* Go Down On Step
* ============================================== */
drop_one_offset()
{

  // -- fly to top right
  if( this.opcode == 1 )
  {
      // -- move line, start point
      this.xPos   += this.stepSize;
      this.yPos   += this.stepSize * this.tr;

  }
  else if( this.opcode == 0 )
  {
    // -- move line, start point
      this.xPos   -= this.stepSize;
      this.yPos   -= this.stepSize * this.tr;

  }
  else
  {
      this.yPos += this.yOffset;
      this.xPos += this.xOffset;
  }


}
```

Explanation

The code above is a **private function**. We know that it is private because it is **inside the class definition of Dot**. Another way we know that it is a private function is because the code is **in between the curly braces** of Dot.

This is important because it means that only Dot has an action named `drop_one_offset()` ➔ the class Dot owns an action called `drop_one_offset()`

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.**

```
/* ==========================================
* Create object of Dot using "new" operator
* ========================================== */
var dot_1 = new Dot( 50, 50, 0, .11, 50, 0 );
dotArr.push( dot_1 );

var dot_2 = new Dot( 310, 50, 0, .1, 10, 1 );
dotArr.push( dot_2 );

var dot_3 = new Dot( 150, 310, 0, .31, 20, 2 );
dotArr.push( dot_3 );

var dot_4 = new Dot( 450, 0, 0, .51, 7, 3 );
dotArr.push( dot_4 );

var dot_5 = new Dot( 300, 0, 0, .7, 50, 4 );
dotArr.push( dot_5 );
```

Explanation

Remember that a **class is a generic template** and we only have **ONE class**.

In most video games, there are multiple copies of the same enemy. **How do we make multiple copies of the same enemy?** We use the word "**new**" to make copies and each copy is called an object. The code above does not make enemies but instead makes 5 copies of Dot ➔ it makes 5 copies and each copy is a different color food.

Since a class is a generic template, we can use it over and over again to make more copies. This is why we use the "**new**" operator 5 times and this results in 5 copies (ie. objects) being created.

If we wanted to customize the food, then we put values inside the open and closed parenthesis to customize the food's shape, color, speed, and so on.

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.**

```
/* ==========================================
* Top To Bottom
* ========================================== */
dotArr[b].drop_one_offset();
dotArr[b].check_sat();
```

Explanation

The code above combines step 3 and 4 together. In step 3, we created the private function `.drop_one_offset()` In step 4, we used the "**new**" operator to create a copy of a Dot (ie. we are making clones of Dot) and then we put each copy into the array using `dotArr.push( dot_1 );`

The code above uses the **member access operator, which is the period**, to show ownership and also to use the private function `.drop_one_offset()` and `.check_sat()`.

1. Remember that the private functions `.drop_one_offset()` and `.check_sat()` were **inside the curly braces** of **class Dot** and so both private functions belong to Dot.

Each clone is calling the private function `.drop_one_offset()` and this causes each clone to start at the top and fall down to the bottom one step at a time.

The fishes on the left and right side stay in the same place. Why? Because the fishes **DO NOT** own `.drop_one_offset()` and a lack of ownership means that they can't use the private action.

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.**

```
/* ==========================================
* Keyboard Control
* ========================================== */
function key_con( e )
{
      console.log( "inside key_con() }}}}}}}}}}}}}} - e.key: " + e.key );

      // -- move left
      switch( e.key )
      {
            // --
            case 'c':
            case 'C': console.log( "----------- clear the lasers -----------" );
                        clearLaserFlag = 1;
                        break;

            /* ========================
            * JS Challenge 2
            * ========================== */


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



            // --
            default: console.log( "nothing happened" );

      }

}
```

Explanation

From the code that has been written, we notice that controlling a game character or changing the game requires having to write the actual action. For example, if I wanted the food at **position 0** to drop one step, I would have to manually write

```
dotArr[0].drop_one_offset();
```

This is good but is there a way to call the private function by using single letter command? There is and the solution is to use a switch statement. A switch statement is used to link a letter to an action (it can be public or private action).

The code in **blue** is a variable that has the keyboard button that was pressed on. The switch statement uses the word **case** to link up a keyboard button with an action. So, when we press down on the c button (upper or lower) of the keyboard, the code will set `clearLaserFlag = 1;` and this will cause the ramps to be cleared.

When the action is complete, the word **"break;"** is used to exit the switch statement. Remember to put **"break;"** or else the next line of code is executed.

JS Challenge 2 – Button control of dotArr[0]

Look for the green banner below and put the code under the banner

```
/* ====================================
* JS Challenge 2
* ====================================== */
```

1. Modify the switch statement so that `dotArr[0].drop_one_offset();` is called when the '**d**' button on the keyboard is pressed down.

2. Start by writing the word **case** and then write the letter.

3. Next, remember to write the colon (:) as a separator

4. Next, write `dotArr[0].drop_one_offset();` on the right side of the colon

5. Finally, remember to write **"break;"**

JS Challenge 3 – Button control of dotArr[1]

Look for the green banner below and put the code under the banner

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

1. Modify the switch statement so that `dotArr[1].drop_one_offset();` is called when the '**f**' button on the keyboard is pressed down.

2. Start by writing the word case and then write the letter.

3. Next, remember to write the colon (:) as a separator

4. Next, write `dotArr[1].drop_one_offset();` on the right side of the colon

5. Finally, remember to write **"break;"**