GD 9
Capital Of The Universe, Part I

How do we gain or lose points? Games that reward players based on avoidance or lack of avoidance promote control and strategy.

In this lesson, we create Capital Of The Universe. This first lesson writing code to gradually move the rings from right to left and and drawing text onto the screen.

Capital Of The Universe – Balance the ball with the laser while matching the city with the state capital.
1. Level 1 & 2 2. Dynamic game objects 3. Drawing text onto the screen

---

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.**

```
/* ===========================================
* Position the rings
*
* =========================================== */
put_in_position_x( origin, offset, spacing )
{
     this.cXPos = origin + (offset * spacing);
}

put_in_position_y( origin, offset, spacing )
{
     this.cYPos = origin + (offset * spacing);
}
```

Explanation

The private functions above belong to the class Circle. How do we know that the class Circle is the owner of these two private functions? We know for sure because the code above is in between the open and closing **curly braces that belong to class Circle (scroll up to see)**. Remember that curly braces are used to group together lines of code. So, anything in between the open and closing curly braces belong to the same group. In our case, the group name is "Circle".

The two private functions above position the rings on the far right side and then space them vertically. Remember that the x is side to side position while the y is the vertical position.

Finally, we have the word "`this.`" Remember that "`this.`" means private variable. So, the Circle owns the private variable `cXPos` and `cYPos`.

Verbal Challenge
Compare the two variable `this.cXPos` with `cXPos` and see that they both have the same name of `cXPos`. However, which one is public and which one is private? How can you tell?

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.**

```
/* ======================================
* Check hit box side boundary
*
* ====================================== */
check_sb()
{
    // -- bounce effect x-axis
     if( this.cXPos <= 5 )
    {
       this.reset_to_right_side();
       return 1;
    }
}
```

Explanation

The game Capital of the Universe is based on rounds. Each round starts with the rings appearing on the middle right side. The round ends when the rings "**touch**" the far left side.

The code in **orange** checks if a ring has "**touched**" the left side of the screen. Every time you hear the word **collision or touch,** think hit box. In our case, the hit box is the entire left side. **How do we implement a hit box and check for collision detection?** We use can "**if**" statement.

Since the entire left side is the hit box, we check "**if the x position of a ring is less than 0**". If so, we start a new round by resetting the rings to the right side.

Next, the function definition gives data back to the caller by using the "**return**" statement. We **return 1** to indicate that a ring has touched the hit box of the left side.

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.**

```
/* ====================================
* Level 1 - updating the game object
* Right to left movement of the rings
* ==================================== */
move_one_step()
{
  this.cXPos += this.cXStep * this.cXDir;

}
```

Explanation

Modern video game development is based on two levels. The first level is updating the position of a game object. For us, we use variables to store the x and y position of our rings. **We update the x and y position by performing math operations and then loading the new data into our variable.**

By default, the game sets `this.cXDir` as -1 to indicate going to the left side. The code then adds 1 to the private variable `this.cXPos`. This translates into → move 1 step to the left side. The ring has been moved one step to the left and will be redrawn in its new position.

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.**

```
/* ==========================================
* Level 2 - Draw the ring
*
*
* ========================================== */
ctx.beginPath();
console.log( " ---------------------------- " );
console.log( "cXPos: " + cArr[c].cXPos );
var index = indexObjArr[c].costumeIndex;
console.log( " ---------------------------- " );
console.log( " << costumeIndex: " +  indexObjArr[c].costumeIndex );
ctx.drawImage( asteroidImgSrcArr[index], cArr[c].cXPos, cArr[c].cYPos, cArr[c].cRad, cArr[c].cRad );
```

Explanation

Step 3 was Level 1, where we update the ring by moving its x position on step to the left.

This step is Level 2, where we redraw the ring in its updated (x,y) position. The code above uses `ctx.drawImage()` to tell the website to redraw the ring. Since Step 3 moved the ring one step to the left, the ring will look like it is moving from right to left.

Remember that we have to wipe the screen every 16.67 milliseconds and then redraw the game objects (in this case, the ring) in their update (x,y) position. This creates animation because we have redrawn the ring in a different position than the last time.

If we don't update the (x,y) position of the ring, then we will redraw the ring in the same position and it looks like no progress was made.

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 QA Object
*
* ========================================= */
var qa = new QA( "Alabama", "Montgomery" );
allQAArr.push( qa );

qa = new QA( "Alaska", "Juneau" );
allQAArr.push ( qa );

qa = new QA( "Arizona", "Phoenix" );
allQAArr.push ( qa );

qa = new QA( "Arkansas", "Little Rock" );
allQAArr.push ( qa );

qa = new QA( "Cali", "Sacramento" );
allQAArr.push ( qa );
```

Explanation

The code above is the question and answer pair that will be displayed onto the screen. Notice that we are using an array to store all question and answer pairs. We are also using **.push()** to put the next question and answer pair at the end of the array. This way, we build the array similar to building a line → we enter the line at the end instead of at the front.

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.**

```
/* ===============================
* Show Timer Text
*
*
* =============================== */
ctx.fillStyle = "#ffffff";
ctx.beginPath();

ctx.fillText( "Timer: ", 550, 45 );
ctx.fillText( totalGameTime, 650, 45 );
ctx.fill();
```

Explanation

There are premade drawing functions for drawing text. The code in **blue** uses the `.fillText()` to tell the website to draw the text "**Timer:** " and then position the text at x position 550 and y position 45.

If we want to change the color of the text, there is also a premade drawing function for coloring the text. The code in **orange** uses the `ctx.fillStyle` to set the color as "`#ffffff`", which is white.

Classes & Objects Review

Have you every driven on PCH from southern California all the way up to northern California? You will see many different types of building, tress, and cars.

Remember that a class is a generic model of something. For example, a car is a class. Saying that "**I see a car driving down PCH**" is a very generic way of describing a car.

We don't usually describe a car in generic vocabulary. Instead, when we see a car driving on PCH, we usually describe the car in a **very specific way**. For example, "**I see a green, 2001 Chevy Malibu driving down PCH**".

So, we can customize a generic model by creating a copy (or a clone) of a class and then giving that copy unique values (an object).

So, a class is a generic model while an object is making a copy of the generic model and then customizing the object with unique values.

```
CLASS Definition
class Car
{
      constructor( model, year, color )
      {
            this.carModel = model;

            this.carYear  = year;

            this.carColor = color;
      }

      print_color()
      {
            console.log( “color of car is: ” + this.carColor );
      }


}
```

The `constructor` is used to create a copy and to also customize the copy with **unique values.** The code above has **3 attributes** for Car and they are `carModel, carYear, and carColor`. The inputs to the constructor are `model, year, color.`

While driving down PCH, we see a boat. Do we describe a boat by saying “**The boat has a carModel of Malibu**?”. We don’t describe a boat in that way because a boat **does not** have attributes of `carModel, carYear, and carColor`. Instead, only a car has those 3 attributes.

The code uses `the period` to shows ownership and the period is called the “**member access operator**”. This means that `carModel, carYear, and carColor` belong to Car and nothing else. Nothing else has those 3 attributes except car.

JS Challenge 1

1. What is a class?
2. What is an object?
3. What operator is used to show ownership?

JS Challenge 2

1. Create a class definition for **SpaceShip**
2. Write the constructor to initialize the class definition
    a. The inputs to the constructor are **rocket, shape, speed**
    b. The private attributes inside the constructor are **ssRocket, ssShape, ssSpeed**.