GD 7
Math Clean Up, Part I

In GD 6, we learned about class and objects. In GD 4 and 5, we learned about side scrolling games and how a **VIEW FRAME** is used to deceive the player. In this lesson, we bring everything together to create an **infinity scroller**.

The main character, which is a boat, stays in the same position but the background scrolls left infinitely to give us the perception of always going to the right.

Math Clean Up – match the answer with the questions. Control your ship's canon using the computer mouse. Think quick or else the ocean debris will get close to your ship and jam the engines.

1. Class and objects 2. Infinity scroller

---

0. Review Class & Objects.

The following review questions can be answered by looking back at GD 6.

0.1. What is a class?

0.2. What is an object?

0.3. How do we create a clone of a class?

0.4. How do we initialize an object?

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.**

```
/* ================================
* QA Class Definition
*
* ================================ */
class QA
{
        constructor( question, answer)
        {
            this.quest = question;
            this.an    = answer;
        }

        check_if_matching( uAnswer )
        {
            return (this.an == uAnswer) ? 1 : 0;

        }

        get_qa_pair()
        {
          return { q: this.quest, a: this.an };

        }

}
```

Explanation

The code above is the class definition of QA, which holds our question and answer. Notice that the reserve word "`class`" is used to show that it is a **special way to combine attributes and functions (actions) together**.

Remember that a **class is a generic model** while an **object is a clone of a class**. Once we make a clone, the constructor is used to initialize the clone and the clone can be customized by giving the constructor parameters.

The class definition has two private attributes and they are called `this.quest` and `this.an`. Notice the reserve word "`this`" means that it is private property and this means that only the class has access to it.

The answers that are displayed onto the screen while falling with the asteroid come from the private property `this.an`.

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( "2 + 2", "4" );
allQAArr.push( qa );

qa = new QA( "6 - 1", "5" );
allQAArr.push ( qa );

qa = new QA ( "9 + 2", "11" );
allQAArr.push ( qa );

qa = new QA ( "3 + 2", "5" );
allQAArr.push ( qa );

qa = new QA ( "13 - 3", "10" );
allQAArr.push ( qa );
```

Explanation

The code above uses the reserve word "`new`" to create multiple clones of the `QA` class. Once a clone is created by the word "`new`", the constructor is called to initialize and optionally customize the clone. Finally, we then load the clone into a variable named "`qa`".

Notice that we give the constructor two arguments to match our constructor definition from above. Since objects can be customized, we can add different data to each clone.

Finally, we put the clone into an array using the `.push()` method. The items in the array will be randomly picked and then displayed onto the screen.

CONTINUE TO THE NEXT PAGE

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.**

```
/* ========================================
* monster timer interval - 3 seconds before debris move right
*
*
* ======================================== */
var
monTI             = 3,
monCurrSec        = 0,
monXPos           = 00,
monYPos           = 250,
monWidth          = 100,
monHeight         = 0,
monXOffset        = 25,
monYOffset        = 1;
```

Explanation

We have an added hazard to the game and it is debris (I called it monster but forgot to change it).

We are using **variables to describe** the (x,y) position, width, height, and speed of the debris. Because we are describing the game object, **this means that we are working on level 1.**

The code in **orange** means that the debris will move closer to the ship every 3 seconds.

The code in **blue** means that the debris's starting position is between the ship and the left boundary.

The code in **grey** means that the debris will advance 25 pixels every 3 seconds.

CONTINUE TO THE NEXT PAGE

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.**

```
/* ===================================
* Game Clock
*
*
* =================================== */
currCount--;
if( currCount == 0 )
{
  totalGameTime--;
  monTI--;

  if( monTI == 0 )
  {
    monXPos += monXOffset;
    monTI = 3;
  }

  currCount = oneSecond;

}
```

Explanation

The game clock variables from above controls the movement of the debris and the run time of the game. When the game starts, the player has 60 seconds to play the game.

The code in **blue** waits for 3 seconds to elapse and then advances the debris by adding the value stored inside `monXOffset` to the x position of the debris. We then restart the 3 second timer by load the value of 3 into the variable `monTI`.

How many samples make up 1 second? The equation is 67 samples make up 1 second. The code in **gold** at the bottom loads 67 into the variable `currCount` and the code in **gold** at the top subtracts 1 from the `currCount`. When the `currCount` reaches 0, then 1 second has elapsed and we subtract the `totalGameTime` by 1.

CONTINUE TO THE NEXT PAGE

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.**

```
/* ============================================
 * Infinity Scroll
 *
 *
 * ============================================ */
clipX -= p1Score;

if( clipX <= 00 )
{
  clipX = 600;
}


// -- first image
ctx.drawImage(img, clipX, clipY );

// -- second image
ctx.drawImage(img, clipX-(canvas.width), clipY );
```

Explanation

The previous GD projects, 4 – 6, used a very large image to perform side scrolling. There is a better way and this new approach still deceives the player by using 2 images.

When the game starts, the **first image** is already within the **VIEW FRAME** while the **second image** is outside and to the right of the **VIEW FRAME**. When the player gets a correct answer, the **first image** scrolls left and the **second image** will scroll left and ***also follow behind***.

When the **first image** is at the left boundary, the **second image** is right behind it and this gives the impression that the image continues.

Immediately, the **first image** is reset to the right side and out of view of the **VIEW FRAME**. When the **second image** is at the left boundary, the **first image,** which was recently reset to the right side, is scrolling along with the **second image** and this gives the impression that the image continues.

The process from above repeats until the game ends. By drawing the **same image back to back** and have each image scroll left, it gives the impression that the image scroll infinitely.

JS Challenge 1 → Write the answer inside chat of Zoom

1. What is a digital key?
2. What is the first index of an array?
3. What is the last index of an array?
4. How do we select on position of an array?

JS Challenge 2 → Write the answer inside chat of Zoom

1. Create an array named "**nums**" and make it an empty array
2. Create a digital key named **c** and load the data 3 into it
3. Select one position using the digital key and load data "**hi**"
4. Load the data 5 into the digital key name **c**
5. Select one position using the digital key and load data "**yes**"