GD 13
Drawing App, Part 1

So far, we have used HTML 5 canvas to position and then draw game objects. We then animate the game objects by calling a function that calls itself.

Remember that HTML 5 canvas is just like a canvas that is used by artists to draw. We will take that last sentence literally because we will know take a detour towards building a small drawing app.

1. Single Frame 2. What is a shape?

---

0. TO REDRAW OR NOT REDRAW

In all previous projects, we created animation by updating the (x,y) position of the game objects and then redrawing the game objects in their new position.

For example, **Words In Space** has the asteroids start from the top and then slowly falling down. We updated the y position of each asteroid by added by 1 to the y position. When we redraw the asteroids in their new y position, it looks like the asteroids are moving.

In order to make the asteroids move continuously, we have to call the same redraw function over and over again. This meant that the redraw function calls itself. The code in **red** below calls itself over and over again.

```
// -- prevent infinite loop
if( sampleCount >= 10800 )
{
  window.cancelAnimationFrame(redrawCon)
}
else
{
   redrawCon = window.requestAnimationFrame(draw_game);
}
```

We have to erase everything on the screen so that we can start the next redraw cycle.

NOT THIS TIME!!!!

For the drawing app to work, we will turn off redraw and keep everything that was previously drawn by the user. **So, instead of wiping the screen and then redrawing, we are now keeping the previous work done and then adding onto the existing work.**

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.**

```
/* ================================================
* Pen Event & Event Handler
*
* ================================================ */
window.addEventListener("mousedown", path_start);

window.addEventListener("mouseup", path_end);

window.addEventListener("mousemove", new_point);
```

Explanation

The code above set up our event listener. Remember, an event listener tells the website to check for an event and if the event occurs, then react by calling the event handler. In the code, above, the words in double quotation are the events and the name after the comma is the event handler.

The **first event listener** tells the website to check **if the user has pressed down** on the left button of the computer mouse. If so, then call the JS function `path_start();`

The **second event listener** tells the website to check **if the user has released the left button** of the computer mouse. If so, then call the JS function `path_end();`

The **third event listener** tells the website to check **if the user has moved** the computer mouse. If so, then call the JS function `new_point();`

What happens if we don't set up event listeners? Then, our website will not detect the event. So, if we didn't write the code above, then when we click and move the mouse, then nothing will be drawn because the website was not set up to detect a mouse click or mouse move.

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.**

```
/* ============================================
* Create objects for paint panel
*
* ============================================ */
var cbObj = new ColorBox( "c", "#006500", 0, 0, 55 );
colorBoxArr.push( cbObj );

cbObj = new ColorBox( "c", "#ffd700", 1, 0, 55 );
colorBoxArr.push( cbObj );

cbObj = new ColorBox( "c", "#bb0000", 2, 0, 55 );
colorBoxArr.push( cbObj );

cbObj = new ColorBox( "c", "#00BFFF", 3, 0, 55 );
colorBoxArr.push( cbObj );

// -- shape
cbObj = new ColorBox( "s", "#606060", 0, 1, 55 );
colorBoxArr.push( cbObj );

cbObj = new ColorBox( "t", "#e0e0e0", 1, 1, 55 );
colorBoxArr.push( cbObj );

cbObj = new ColorBox( "z", "#ffa500", 2, 1, 55 );
colorBoxArr.push( cbObj );
```

Explanation

The code above sets up the color panel that is seen on the left side.

We are creating many objects of type **ColorBox**. Remember that **ColorBox** is a class and a class is a model. A class is very generic. We clone a class multiple times to create objects and we can then customize these objects by giving it input parameters. **The code in gold customizes the object by making the object a yellow paint object.**

The code in **orange** then puts the object at the back of the array named **colorBoxArr**.

3. WHAT IS A SHAPE?

A shape is an array of points. When we connect these points together with a line, the connections create an outline of the shape that we see.

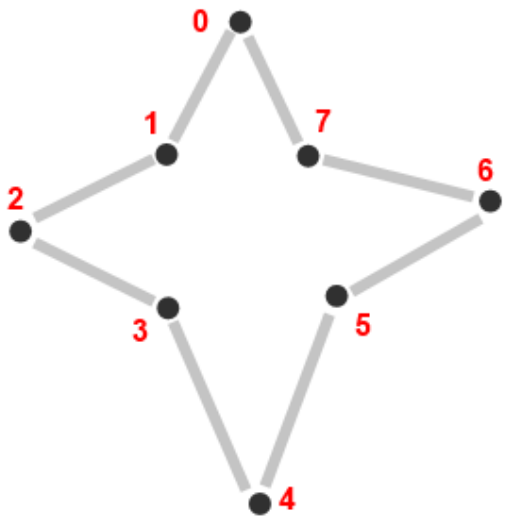
Canvas starts by asking us to pick an (x,y) point as the starting point and then we pick more points to following the starting point and so on.

So, we can think of canvas as a literal painting canvas and the predefined functions allow us to pick up our pen and place them down onto the "**canvas**" to mark the next (x,y) point. We then connect the points together to create a shape.

The code

```
ctx.fillStyle = "#c0c0c0";
ctx.fill();
```

connects the points with a line and then fills in the shape with a color that we choose, where `#c0c0c0` is grey.



When we connect all of the points together with a line, it creates a shape. **Remember that the first point, point 0, and last point, point 7, must connect with each other.** If they do no connect with each other, the line never ends and we don't have a shape.

In HTML 5 Canvas, we begin drawing by typing **ctx.beginPath()** and this allows us to start with **point 0**. We tell the website to **end and connect all points** with a line by typing **ctx.fill().**

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.**

```
/* =======================================================
* Using Pen
*
* ======================================================= */
if (newPathFlag && newXYFlag)
{
   if( sampleCount <= 0 )
   {
      // pathString = "M " + eventObj.offsetX + " " + eventObj.offsetY;

      prevMouseX = eventObj.offsetX;
      prevMouseY = eventObj.offsetY;

      // -- put into array
      pObj = new PointXY( eventObj.offsetX, eventObj.offsetY );
      pointArr[sampleCount] = pObj;

      sampleCount++;

   }
   else
   {
      pathString = "M " + prevMouseX + " " + prevMouseY + " L " + eventObj.offsetX + " " + eventObj.offsetY;

      prevMouseX = eventObj.offsetX;
      prevMouseY = eventObj.offsetY;

      // -- put into array
      pObj = new PointXY( eventObj.offsetX, eventObj.offsetY );
      pointArr[sampleCount] = pObj;

      sampleCount++;


      // -- draw
      var pathObj = new Path2D( pathString );
      ctx.beginPath();
      ctx.lineWidth   = "3";
      // ctx.strokeStyle = "#ffa500";
      ctx.stroke(pathObj);
    }

}
```

Explanation

Remember from Step 1 that we set event listener so that our website can detect when the user presses down on the left button of the mouse and when the user moves the mouse.

The code in **red** will active if the user clicks on the left button of the mouse **while** moving the mouse. The **logical AND** means that the user must do both or else the code won't execute. So,

if the user moves the mouse but does not press down on the left button of the mouse, the **logical AND is false** and so the code won't execute.

The code in **green** checks if the user has started drawing. If so, then we must save this first (x,y) point because it becomes the first point of the shape.

If the user has already started drawing (ie, precondition), then the code in **blue** means that the user has created the second point.

The code in **green** and **blue** work together to create the shape. The code in **green** is the first point while the code in **blue** is the second point. The code in **orange** then **connects the first point with the second point to create a line and this first line becomes the first connection that will make the outline of our shape.**

Finally, the **code in grey** puts the points into an array. Why? **Because a shape is an array of points.**

JS Challenge

As we have seen, a "`for`" loop can be used to solve many problems. In JS curriculum, we just a "`for`" loop to create a playlist and then used a "`for`" loop to create a list of calendar events. **For this JS challenge, you can refer to JS 10 as a reference.**

Go to the website https://www.w3schools.com/code/tryit.asp?filename=GD76D4FQPT8Q to write code for this JS challenge. Write the code inside the body of `function main()`. The code should do the following:

```
var points = [ 10, 30, 1, 5, 80, 1, 0 ]; // -- copy and paste into function main()
```

Put the code inside the body of function main(). Write a "`for`" loop to print out the data found at even-numbered positions.

1. Create a digital key variable named p and load the data 0 into it
2. Loop through until the end of the array
3. Inside the body of the "`for`" loop, select on position of the array
4. Send an alert to print the data at that position