**Draft Chapter 1: The Canvas Grid**

( Note to self: Using link: <https://codepen.io/dwbento/pen/JjExVLg> )

**Getting Setup**

Before we get started on creating game objects and game behaviors, let’s spend some time setting up our game canvas and organizing our code for use in the Codepen environment. Codepen is a nice, convenient online platform that simplifies the publishing of our HTML/CSS/JavaScript code. Codepen takes care of some basic, unattractive utility work so that we can focus on being creative. Once we are finished with our asteroids game using Codepen, we won’t be able to instantaneously transfer our code into web-ready format. But, our final product won’t be too far from publishing on the web.

As we start organizing our code, we will generally split our code into HTML, CSS (style), and JavaScript categories. Normally, we use the <style> tag in the <head> section of the HTML to instruct our HMTL editor to import our style code (CSS code). Likewise, we would normally use the <script> tag to import the JavaScript code into the <head> section. But, we can take advantage of one of Codepen’s settings to do this for us. For this reason, we will exclude the <style> tag from the head of our HTML code. We will still use the <script> tag in the <head> section to import multiple pieces of JavaScript code into our game. We will store our style code and our script code in separate “pens”, where a Codepen “pen” is just another word for “file”.

So, our first pen, titled “***AsteroidsGame***”, will contain our HTML code and some basic JavaScript code. Here is a listing for that HMTL code:

***Listing XX***. Basic HMTL Game Code

<!doctype html>

<html>

<head>

<title>Asteroids Game</title>

<script>

// Generally, this is where you place the address of your JS code file

</script>

<style>

// Generally, this is where you place the address of your CSS code file

</style>

</head>

<body>

<h1>Basic Canvas</h1>

<canvas id="asteroids" width="500" height="500"></canvas>

</body>

</html>

We will keep our style (CSS) code in a separate pen (file). To start our first style pen, we will create a pen named “***style\_asteroids***” and use the following code.

***Listing XX***. Very Basic Stylesheet

body {

font-family: sans-serif;

text-align: center;

}

canvas {

background-color: black;   
}

Now, turning our attention to our first JavaScript code for our game. We will organize most of our JavaScript code into logical structures called “classes”, which you have already studied. If you have not, then now is a good time to take a break from creating this game and spend some time studying the topics of “classes” and “objects”. Please don’t continue with this game project until you are comfortable with classes and objects.

Let’s create a pen named “***class\_grid***”. In this separate pen, we will place the new code for our class in the JS windowpane. This class will maintain all the attributes for our background grid and also define the behavior for drawing the background grid.

***Listing XX***: Class Grid

class Grid {

constructor(w, h, incr) {

this.w = w;

this.h = h;

this.incr = incr;

}

drawGrid(c) {

//stepping from left to right in increments of “incr” pixels

for (var x = 0; x < this.w; x += this.incr) {

c.beginPath();

c.moveTo(x, 0);

c.lineTo(x, this.h);

c.stroke();

}

//stepping from top to bottom in increments of “incr” pixels

for (var y = 0; y < this.h; y += this.incr) {

c.beginPath();

c.moveTo(0, y);

c.lineTo(this.w, y);

c.stroke();

}

}

}

Since you already know some things about classes, the constructor portion of the above code should look familiar. We can think of the “class” as the blueprints for an “object”. In order to invoke this Grid class to create a “real” grid object, we use code like this:

let gridObject = new Grid(canvas.width, canvas.height, 10);

Now that we created an instance of our class (instantiated an object), we can command that the object draw itself:

gridObject.drawGrid(context);

All the code needed to instantiate the grid object and then draw the grid is as follows:

***Listing XX***: Instantiating the Grid Object

// Need to obtain and store the “context”

var canvas = document.getElementById("asteroids");

var context = canvas.getContext("2d");

// instantiate the grid object

let gridObject = new Grid(canvas.width, canvas.height, 10, "green", 0.3);

// command the object to draw itself

gridObject.drawGrid(context);

We will place this code in the JS windowpane of the “AsteroidsGame” pen, not include it in any of the classes. Notice that our grid object needed access to the canvas object that lives in the HTML document. The *document.getElementById*( ) method literally reaches into the parent document and looks for an element that has been tagged with a particular ID. And if you look inside the parent HTML, you will see that the canvas element was given an ID of “asteroids”. The *canvas.getContext*( ) method imports a toolchest of drawing attributes and drawing methods that are associated with our particular canvas element.

For the method *drawGrid*( ), we need to pass in the canvas’ context. As mentioned earlier, the “context” can be thought of as all the drawing tools and the canvas attributes; all contained in a large package. We need to pass this to the *drawGrid*( ) method so it can draw on the canvas. Think of a teacher handing over the dry marker and eraser to a student so they can draw on the whiteboard.

You can see from the listing above, we will utilize various ***attributes*** (***characteristics***) from the “context” (“drawing toolchest”) that we just imported. We will use:

canvas.width:

canvas.height:

context.strokeStyle:

context.lineWidth:

We will use the following ***methods*** (***behaviors***) that also come along with the “context” that we imported:

context.moveTo( )

context.lineTo( )

context.stroke( )

context.beginPath( )

***Assignment XX***: Your job is to look up each of the forementioned attributes and methods and write a short 1-sentence definition for each. Here is a authoritative source to help you get started: [*https://www.w3schools.com/JSREF/DEFAULT.ASP*](https://www.w3schools.com/JSREF/DEFAULT.ASP)

***Assignment XX***: Before we move on to the next phase, it is imperative that you fully understand how the ubiquitous For-Loop works. If you have not invested time studying this universal component of computer science, then take a break from creating this game. Go and study this topic until you are very comfortable with using For-Loops. There will be numerous For-Loops in this game. There is a plethora of awesome resources on the internet that can guide you safely through this topic. Likewise, there are a multitude of resources on the net that are unorganized or tough to understand. I recommend that you include [*www.w3schools.com*](http://www.w3schools.com) as one of your tutorials.

***Listing XX***: Updated HTML Code

<html>

<head>

<title>Asteroids Game</title>

<style>

/\* This is normally where you import the stylesheet, ***if*** we weren't using Codepen

</style>

<script src="URL for filename.js"></script>

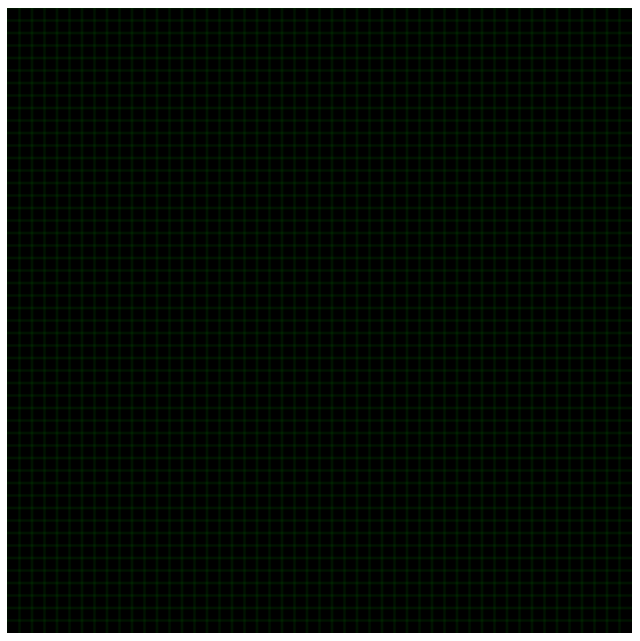
</head>

<body>

<h1>Basic Canvas</h1>

<canvas id="asteroids" width="500" height="500"></canvas>

</body>

</html>

Here is a screenshot of our grid thus far.

***Figure XX***. Basic Grid

**Improve the Grid**

Congratulations on getting your grid to work! Even more, congratulations on setting up your first class; the class we called “Grid”. But, to be honest, our grid is not very impressive. One improvement would be incorporating major- and minor-axis. Now that we have a class that’s functional, let’s improve it. Assuming that our basic grid already has a spacing of 10 pixels, we could change the line’s thickness every 50 or 100 pixels to create the major-axis. Let’s put an If-Statement inside each of our For-Loops.

***Listing XX***: Major/Minor Grid

drawGrid(c) {

for (var x = 0; x < this.w; x += this.incr) {

c.beginPath();

c.moveTo(x, 0);

c.lineTo(x, this.h);

if (x % 50 == 0) {

c.lineWidth = 0.6; // if x is a multiple of 50

} else {

c.lineWidth = 0.3; // otherwise, no change

}

c.stroke();

}

//stepping from top to bottom in increments of 10 pixels

for (var y = 0; y < this.h; y += this.incr) {

c.beginPath();

c.moveTo(0, y);

c.lineTo(this.w, y);

if (y % 50 == 0) {

c.lineWidth = 0.6; // if y is a multiple of 50

} else {

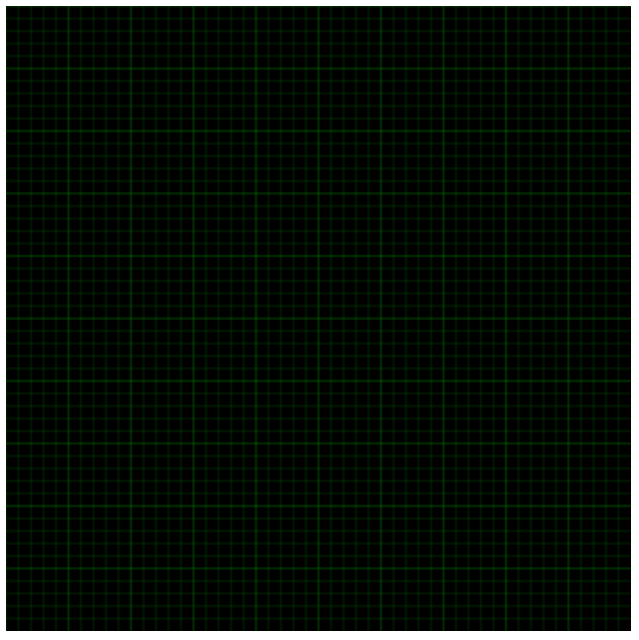
c.lineWidth = 0.3; // otherwise, no change

}

c.stroke();

}

}

Our If-Statement is checking to see if the ***x***- or ***y***-variable is a multiple of 50. If ‘yes’, then the linewidth is doubled. Also notice that we only changed code inside of the Grid class, inside the *drawGrid*( ) method.

***Figure XX***: Grid with Major- and Minor-axis