Mat: Starting the week of the 10th I began working on some map updates. This carried over into the next week where I implemented an obstacle array. This array is initialized as all 0s. When an object model is placed, the index is marked with a 1. This helps to ensure that no models will spawn in that location. Next I set up a system to prevent any one quadrant of the map from becoming too populated with objects.

The next task I undertook was reworking our highlight generation. Now two layers of highlights are created by making two planeBufferGeometries and setting the heights of their corners based on the cell they cover. The highlights are placed using a modified and exported method which will accept a y and x coordinate as well as the object and the number of vertices in the map. The method (placeObject located in gameBoard.js) will automatically convert the array coordinates to cartesian plane coordinates (e.g. obstacles[15][15] is equivalent to map[8][8] while obstacles[0][0] is equivalent to map [-8][-8]). These highlight layers are colored blue (to indicate that a space is in movement range) and red (to indicate a viable target). The visibility of these highlights is easy to switch as each highlight has a unique name.

Over the course of these weeks, I began refactoring some code to simplify files. To that end, I have moved the layer1 method which creates the natural obstacles for the map to its own file as well as the highlight creation process. From this, I found some snippets that were being used regularly and refactored them to export and reuse. One such method is the previously mentioned placeObject method. This refactoring was helpful in my most recent completed task, actor spawn locations.

For actor spawning locations, I set up a nested loop. The outer loop controls whether the player party or enemy party is being placed. The inner loop places 5 random models in a randomly chosen section of the map. The parties will spawn opposite of eachother in a randomly chosen, 3x6 block of the map. Each block is 2 spaces from the edge of the map. The parties face each other upon spawning and can be oriented North/South or East/West. For the time being, I have the place where the model spawns set its corresponding obstacle array index to 2. When checking if a space is occupied, the method checks for values that are not 0 to determine if a space is filled. So with actors being represented by a 2, we can easily determine if a model is an actor or terrain object.

Emily: Throughout the hectic last couple weeks, I worked on merging mainly me and Carson's code together as he decided to not be a part of the group anymore. My process involved looking over all of his code and determining the contributions that would benefit me and Mat. I found a couple pieces of code that we will be altering and decided to merge them with my existing to edit later. I also pulled down what Mat had put up in the master branch of the new repo, so I merged everything into that code at the same time.

The next thing I wanted to focus on was implementing a way for the player to click an enemy to select them as a target. To do this, I decided to use a Raycaster. While I managed to get the Raycaster to select the appropriate model, it was selecting only portions of the material. I looked up ways to combat this problem and found that this occurs when using glTF models as they are made up of different meshes. I came across two options: picking and bounding boxes. Picking involves creating our own mesh to place on models. I decided to go with the bounding box option and have been in the process of giving our models bounding boxes to make the implementation of Raycaster easier and more reliable.