

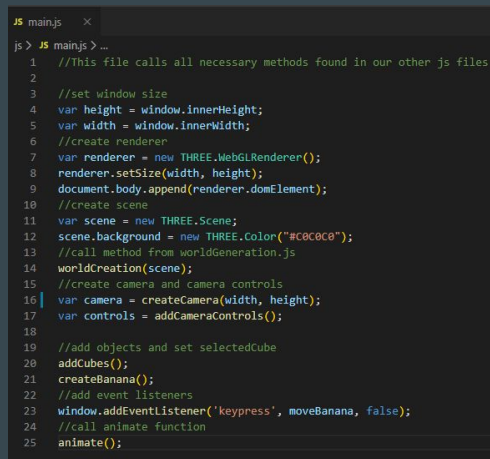
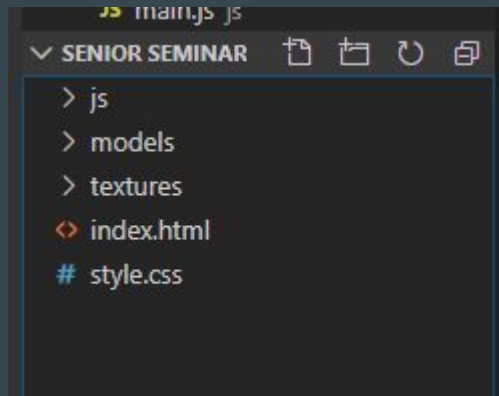
Week 2 Presentation



Carson, Emily, & Mat

Reworking our file structure

- We were tired of having all of our code being written only in the main.js class
 - Made it difficult to push to Github
 - Generally messy to look at
- One Key Goal:
 - Modularize our code into smaller, more specific files that serve only a few purposes
- Our current structure:
 - Index.html
 - Style.css
 - /JS
 - /Models
 - /Textures



Modularization process

As the project continued, it became clear that our *main.js* file would become too difficult to manage.

We had the idea of splitting it into four files instead:

main.js

worldGeneration.js

objectGeneration.js

camera.js

Modularization process

- Along with the *three.js* files, each file is linked to *index.html* using the script tag and set as a source (src) attribute.

```
4
5 ▼ <head>
6     <link rel="stylesheet" href="./style.css">
7     |
8     <script src="./js/three.js"></script>
9     <script src="./js/OrbitControls.js"></script>
10    <script src="./js/TrackballControls.js"></script>
11    <script src="./js/OBJLoader.js"></script>
12    <script src="./js/worldGeneration.js"></script>
13    <script src="./js/camera.js"></script>
14    <script src="./js/objectGeneration.js"></script>
15
16    <title>Senior Seminar Project</title>
17 </head>
18
19 ▼ <body>
20     <script src="./js/main.js"></script>
21 </body>
```

Modularization process

Next, each separate file contains functions that are then called in *main.js*. These functions need to be called in a certain order for the game to be set up.

These steps made our code easier to handle. Additionally, the debugging process will be easier, it is more readable, and merging into a github branch won't cause as many conflicts.

```
//This file calls all necessary methods found in our other js files

//set window size
var height = window.innerHeight;
var width = window.innerWidth;
//create renderer
var renderer = new THREE.WebGLRenderer();
renderer.setSize(width, height);
document.body.append(renderer.domElement);
//create scene
var scene = new THREE.Scene;
scene.background = new THREE.Color("#C0C0C0");
//call method from worldGeneration.js
worldCreation(scene);
//create camera and camera controls
var camera = createCamera(width, height, renderer, scene);
var controls = addCameraControls();

//add objects and set selectedCube
var selectedCube = addCubes();
createBanana();
//add event listeners
window.addEventListener('keypress', cameraRotation, false);
window.addEventListener('keypress', moveBanana, false);
//call animate function
animate();
```

Implementing range into Actors

We began by thinking about range in a strictly grid-based system

- We decided that an actor should be able to hit any of the surrounding eight spaces with a range of 1
- To visualize the problem, I made the chart. After thinking about the problem for a while, I designed a range scan algorithm.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 5 | | | | | |
| | | | | 5 | 4 | 5 | | | | |
| | | | 5 | 4 | 3 | 4 | 5 | | | |
| | | 5 | 4 | 3 | 2 | 3 | 4 | 5 | | |
| | 5 | 4 | 3 | 1 | 1 | 1 | 3 | 4 | 5 | |
| 5 | 4 | 3 | 2 | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| | 5 | 4 | 3 | 1 | 1 | 1 | 3 | 4 | 5 | |
| | | 5 | 4 | 3 | 2 | 3 | 4 | 5 | | |
| | | | 5 | 4 | 3 | 4 | 5 | | | |
| | | | | 5 | 4 | 5 | | | | |
| | | | | | 5 | | | | | |

Range Scan Algorithm

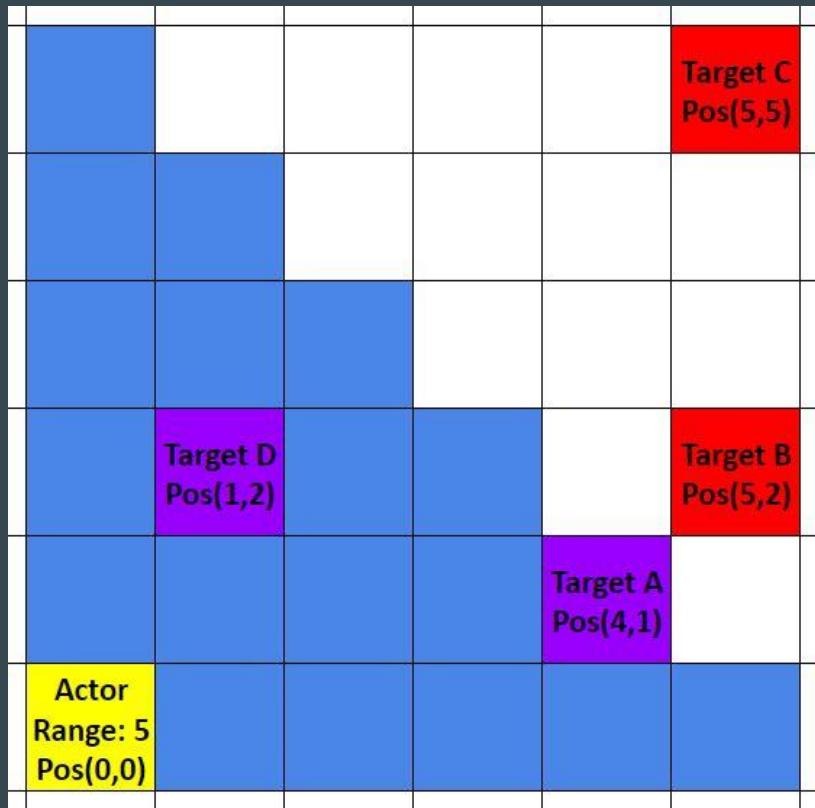
| | | | | | | |
|--|--------------------------|----------------------------|----------------------------|----------------------------|---------------------------|-----------------------|
| Let r = the actor's range (5), let i = zero | $(x + i, y + r)$ | $\leftarrow r=5, i=0$ | | | | |
| Use the actor's range as a starting point and decrement as you loop. | $(x, y + 4) \rightarrow$ | $(x + 1, y+4)$ | $\leftarrow r=4, i=1$ | | | |
| In the inner loop, check the cells to the right i times. | $(x, y + 3) \rightarrow$ | $(x + 1, y+3) \rightarrow$ | $(x + 2, y+3)$ | $\leftarrow r=3, i=2$ | | |
| After the inner loop, before the end of the outer, increment i . | $(x, y + 2) \rightarrow$ | $(x + 1, y+2) \rightarrow$ | $(x + 2, y+2) \rightarrow$ | $(x + 3, y+2)$ | $\leftarrow r=2, i=3$ | |
| With each iteration, check to see if the target's x,y match the scan x,y | $(x, y + 1)$ | $(x + 1, y+1) \rightarrow$ | $(x + 2, y+1) \rightarrow$ | $(x + 3, y+1) \rightarrow$ | $(x + 4, y+1)$ | $\leftarrow r=1, i=4$ |
| Perform this operation for each corner (double check your signs) | Actor $(xPos, yPos)$ | $\leftarrow (x + 1, y+1)$ | $\leftarrow (x + 2, y+1)$ | $\leftarrow (x + 3, y+1)$ | $\leftarrow (x + 4, y+1)$ | $(x + r, y + i)$ |
| | | | | | | |

Way more complicated than necessary (but not useless)

Range Check

Take the difference between the actor's position and the target's position (xPos and yPos).

- If the sum of the differences is greater than your actor's range, the target is out of range.
 - A: $(4-0) + (1-0) = 5 = \text{in range}$
 - B: $(5-0) + (2-0) = 7 = \text{out of range}$
 - C: $(5-0) + (5-0) = 10 = \text{out of range}$
 - D: $(1-0) + (2-0) = 3 = \text{in range}$



Range Check

```
//Check to see if an actor is in attack range
inRange(actor){
    var xDiff = this.getDiff(this.xPos, actor.xPos);
    var yDiff = this.getDiff(this.yPos, actor.yPos);

    if(xDiff + yDiff > this.range)
        return false;
    else
        return true;
}

//Get the difference between two ints
getDiff(int1, int2){
    if(int2 > int1)
        return int2 - int1;
    else
        return int1 - int2;
}
```

Range Scan

```
[0,0,0,0,0,0,0,0,0,0,0,0]],
this.move(5,5); //position the
var x = this.xPos;
var y = this.yPos;
var lessRange = 0; //setup variab

for(var r = this.range; r > 0; r--){
    for(var i = 0; i <= lessRange; i++){ //
        arr[x-i][y+r] = 1; //
        arr[x+i][y-r] = 1;
        arr[x+r][y+i] = 1;
        arr[x-r][y-i] = 1;
    }
    lessRange++;
}

return arr;
}
```

Other uses for range scan

- Can be used for highlighting terrain
 - Show how far an actor can move or the range of tiles it can affect.
- Non-meta A.I. operations
 - If we don't allow the A.I. to know all the information about the battlefield the way the player does, it can be useful for enemies to find their targets and plan their turns.

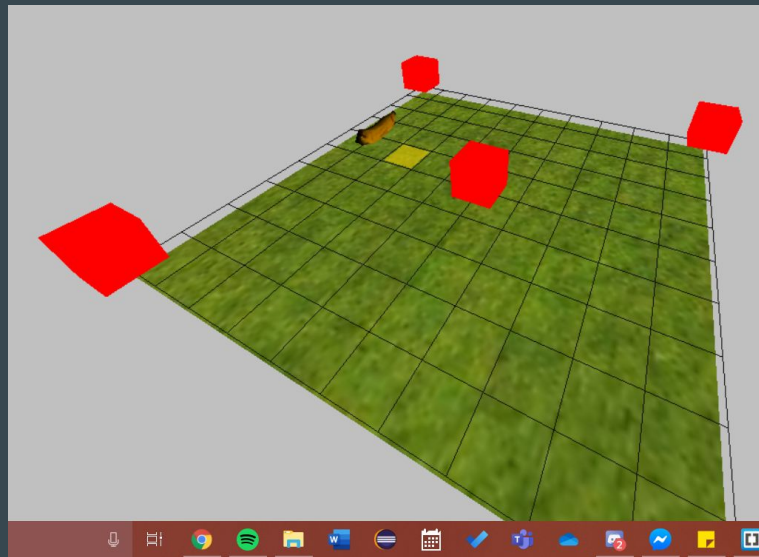


Showing range in our game

We knew we needed a way for the player to visibly see the range, so I had the idea of using small, yellow planes that are placed on the grid.

These are noticeable while not being too distracting. I accomplished this by setting the opacity to 0.5.

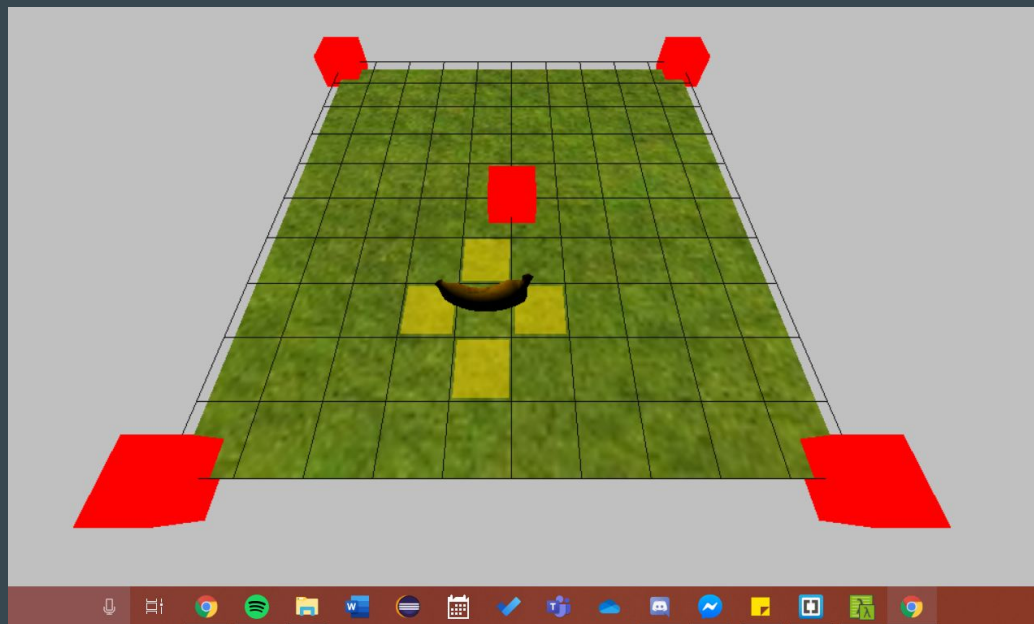
```
function createHighlight(){  
    //adding plane geometry to  
    var highlightPlane = new  
    var highlightMaterial = new  
        color: "#FFD700",  
        transparent: true,  
        opacity: 0.5  
    });
```



Showing range in our game

Once the panel was made, I changed the position of it relative to the banana within the *moveBanana(event)* event handler.

After getting that to work, I added three more panels surrounding the banana in order to more accurately reflect a range.



Potential AI Library

- Yuka
 - Built specifically to be used in THREE.js
 - <https://discourse.threejs.org/t/yuka-a-javascript-library-for-developing-game-ai/5298>
 - What we could potentially use it for:
 - Game Saves
 - Line of Sight
 - Short-Term Memory System

Character Models following the cursor


- Since we want our character to attack in the direction they are looking, we were looking to implement a way to have models follow the cursor.
- No code has been written for this yet, but here is some concept of how it would work
- <https://discourse.threejs.org/t/3d-model-look-mouse-cursor/5455/8>

Having the camera follow a model

- Before: We had the camera focus on various, stationary cubes in the environment
- Now: Have the camera follow the movements of our banana model, no matter what the current rotation of the camera is
- Issues: You cannot properly rotate around the object with Q and E, so we need to reimplement that.

This week's useful findings

- Textures can be tricky!
 - We ended up finding a more basic model from turbo squid (hence the banana)
 - The texture is a .png rather than a .rar file
- Setting opacity
 - In order to set the opacity of a material, you **need** to set the *transparent* value to true.
- Positioning
 - Use `console.log()` to output a `Vector3` value in order to view a model's exact placement on the map



```
▶ Vector3 objectGeneration.js:106
▶ Vector3 {x: -0.5, y: 0.25, z: -2.5} objectGeneration.js:106
▶ Vector3 {x: 0.5, y: 0.25, z: -2.5} objectGeneration.js:86
▶ Vector3 {x: 0.5, y: 0.25, z: -1.5} objectGeneration.js:86
▶ Vector3 {x: 0.5, y: 0.25, z: -0.5} objectGeneration.js:86
>
```


Our next steps

- Carson
 - Continue to fix up the camera, and begin to research procedural generation
- Emily
 - Make highlight blocks transparent when off of map
 - Create appropriate number and design of highlight blocks based on character
- Mat
 - Collaborate with Emily on Highlights
 - Work on tying models and actors together