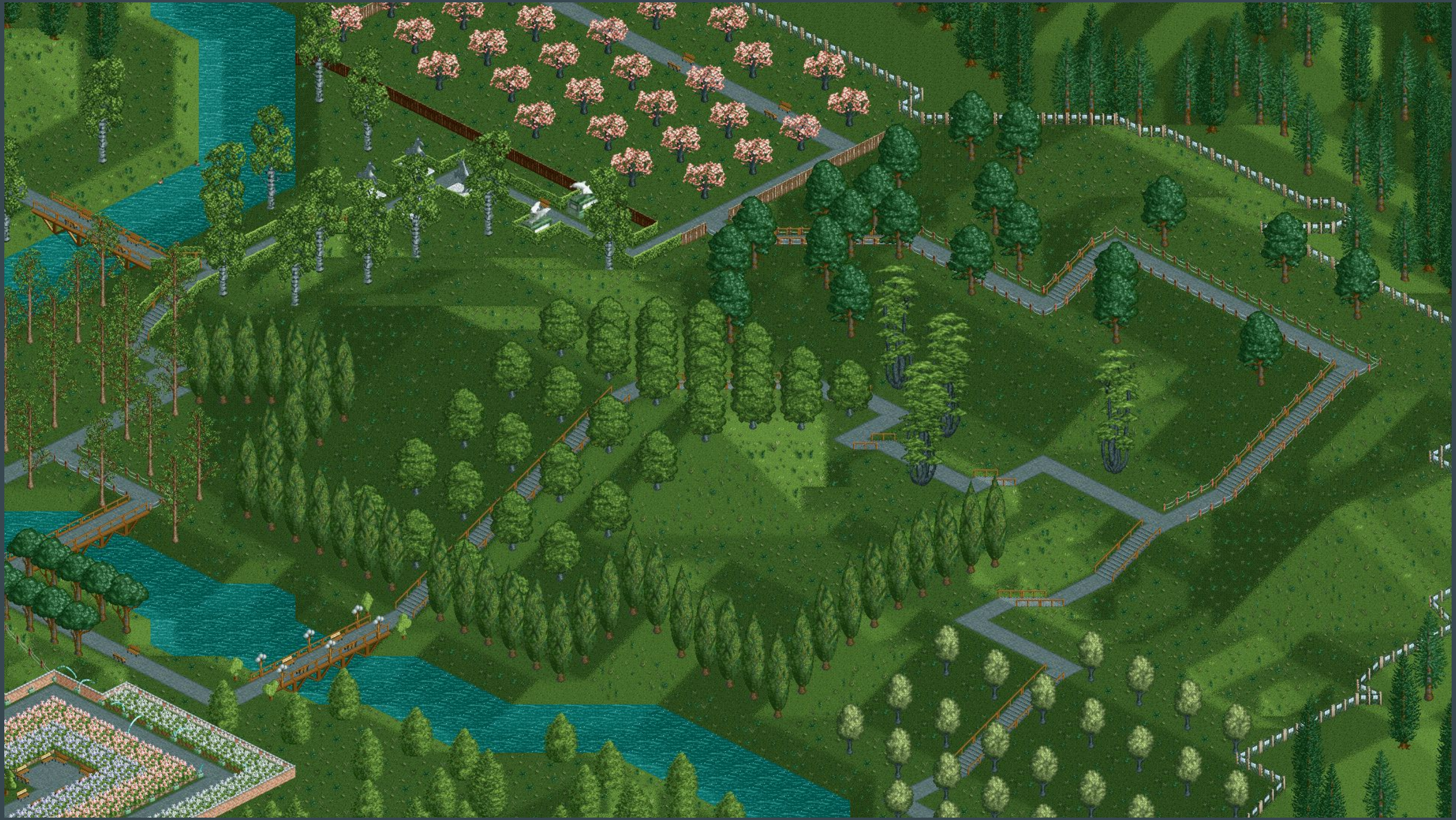
Week 5 Presentation Team 3

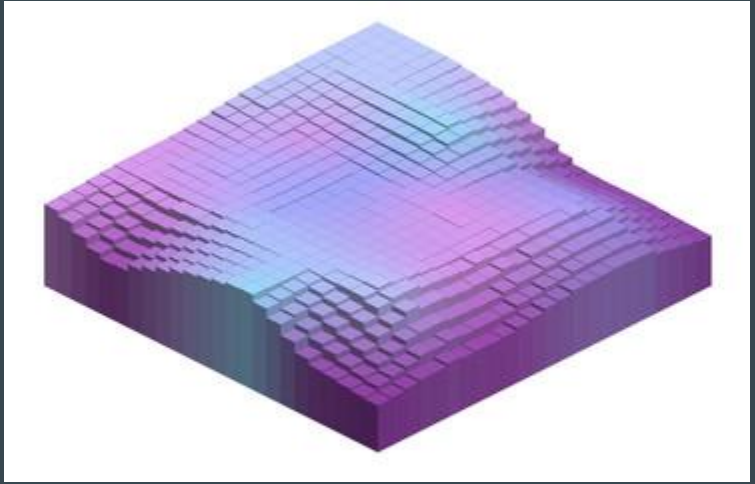
By Emily, Carson, & Mat

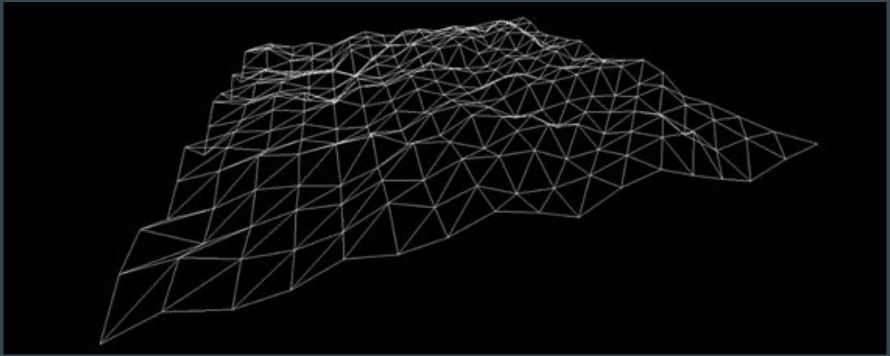
Terrain (some more)



What we are looking for in a map

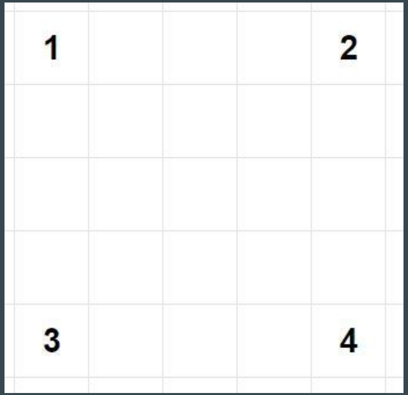
- Polygonal appearance
  - Squares and triangles
    - Few vertices in our height map
- We want to know what to expect
  - Easy to seed for "randomness"
  - We want to easily influence the shape of the map
    - Plains, valleys, basins, hills, etc...

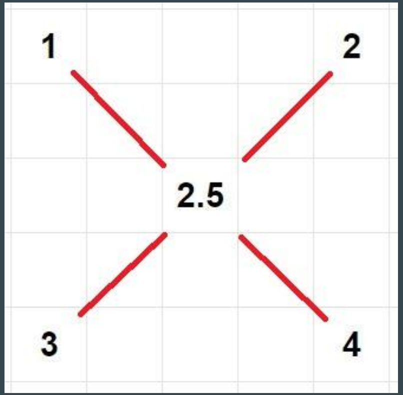


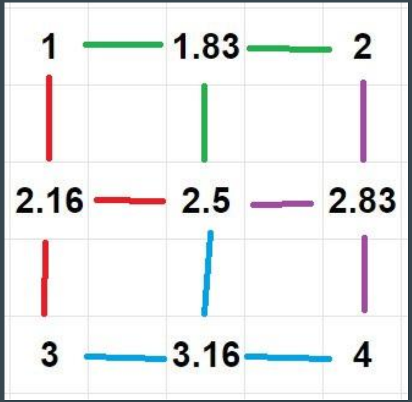


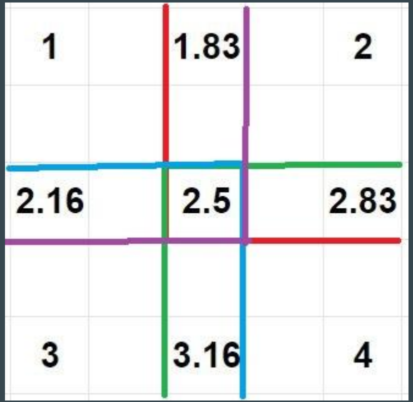
Diamond-Square Algorithm

- Fills a 2d array with midpoint displacement based on the initial corner values
- Array size is limited to 2^n+1
- Start by finding the value of the center space
  - Average of the corners (Square step)
- Next, find the center side values
  - Average of the points (Diamond step)
- Divide the array into four chunks
  - Rinse, repeat



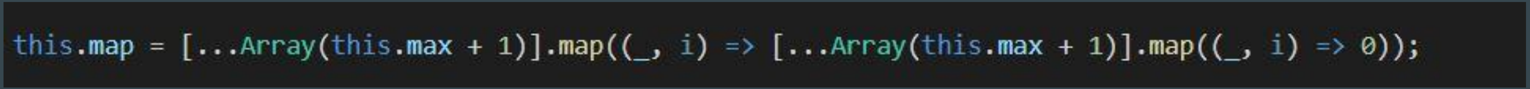






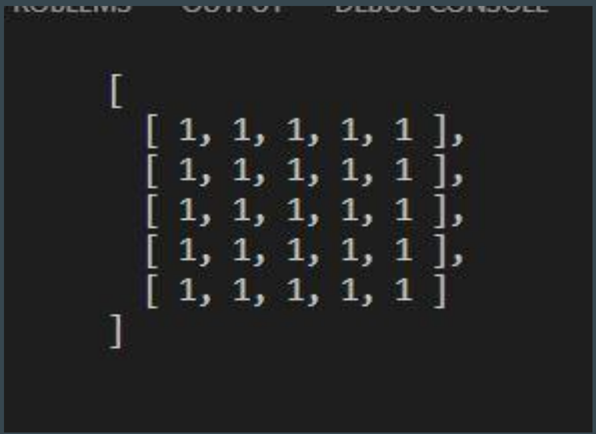
Problems

- Arrays...

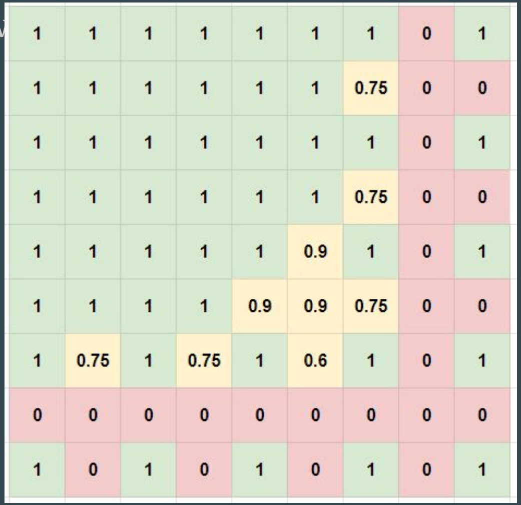


```
this.map = [...Array(this.max + 1)].map((_, i) => [...Array(this.max + 1)].map((_, i) => 0));
```

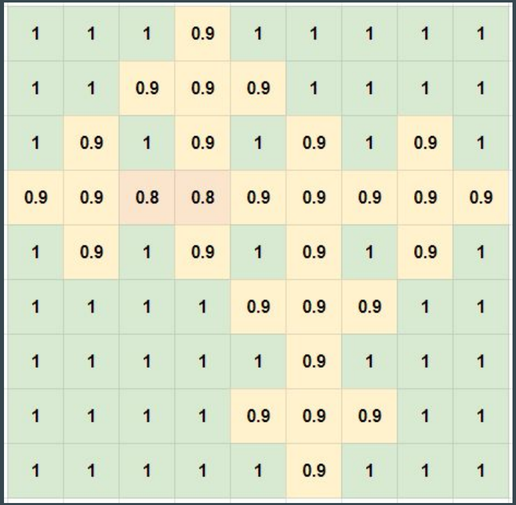
- Recursion...
- N greater than 2
    - When creating a 5x5 heightmap, it w
      perfectly



```
[
  [ 1, 1, 1, 1, 1 ],
  [ 1, 1, 1, 1, 1 ],
  [ 1, 1, 1, 1, 1 ],
  [ 1, 1, 1, 1, 1 ],
  [ 1, 1, 1, 1, 1 ]
]
```



1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	0.75	0	0
1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	0.75	0	0
1	1	1	1	1	0.9	1	0	1
1	1	1	1	0.9	0.9	0.75	0	0
1	0.75	1	0.75	1	0.6	1	0	1
0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1



1	1	1	0.9	1	1	1	1	1
1	1	0.9	0.9	0.9	1	1	1	1
1	0.9	1	0.9	1	0.9	1	0.9	1
0.9	0.9	0.8	0.8	0.9	0.9	0.9	0.9	0.9
1	0.9	1	0.9	1	0.9	1	0.9	1
1	1	1	1	0.9	0.9	0.9	1	1
1	1	1	1	1	0.9	1	1	1
1	1	1	1	0.9	0.9	0.9	1	1
1	1	1	1	1	0.9	1	1	1

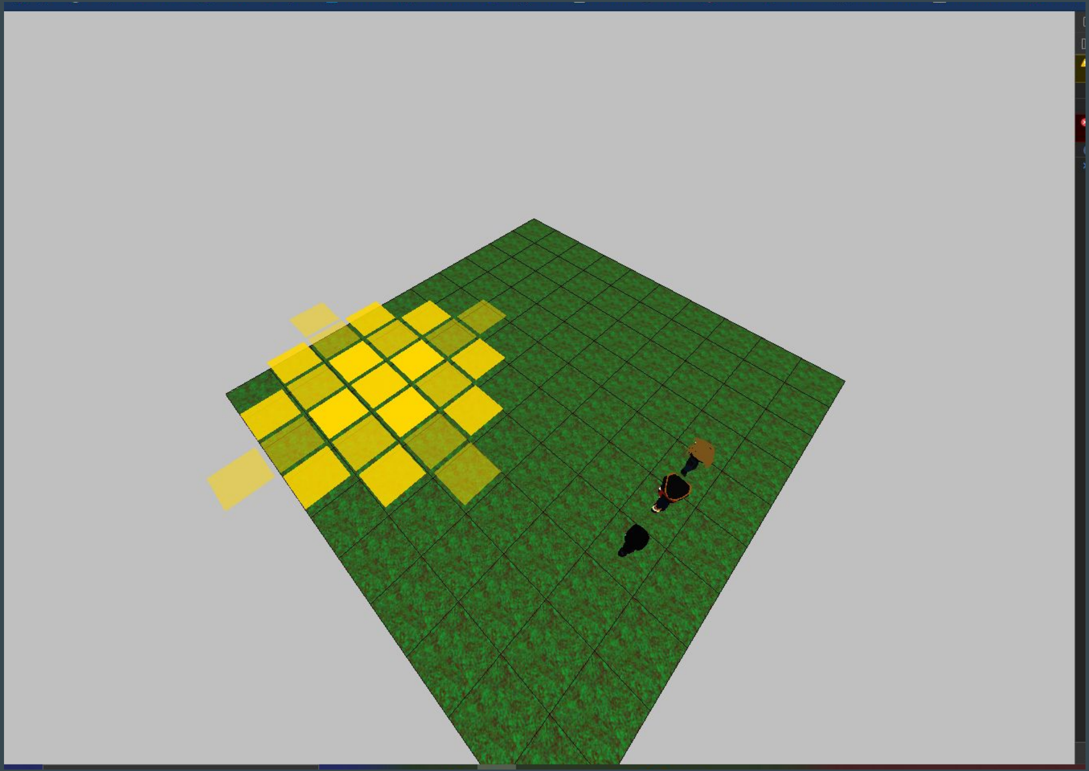
Our implementation of flood fill

- Because we are basing our movement on a grid system, we believed flood fill would work great for our move distance layout
- As we have it right now, all of our maps will be a consistent size (n x n) where n is equal to our terrain generator.
- Because of this, we load all highlights over the map and set their visibility to false.
- Our flood fill will then find all the spots within the range of the character and set their visibilities to true. We can then do the same process to set them all to false.

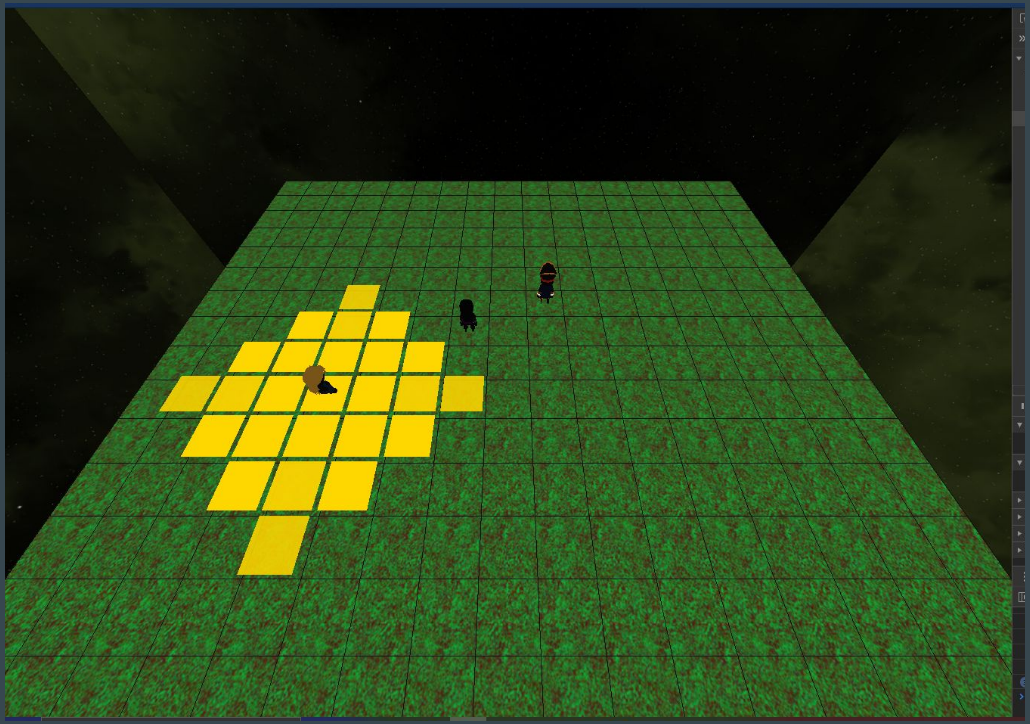
Issues we had with flood fill

- Could not easily despawn highlights
- At the time I made them, highlights could overlap with each other and cause major slowdowns

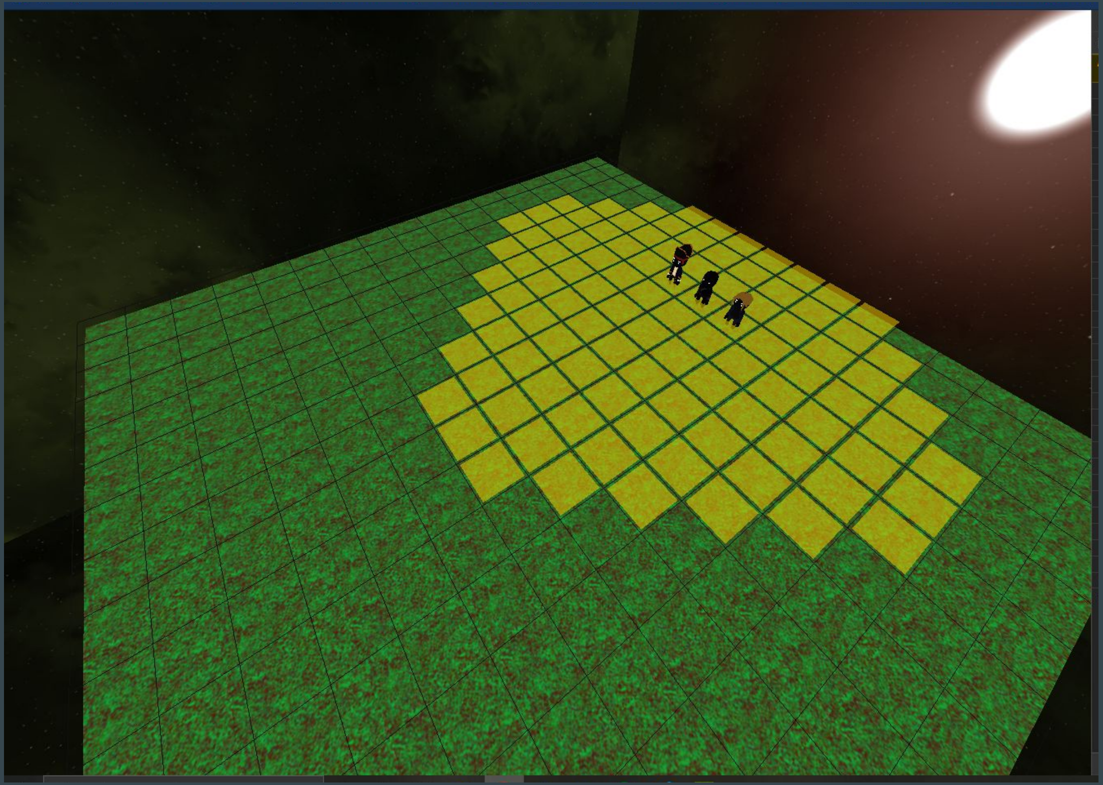
Initial Attempt



Second Attempt

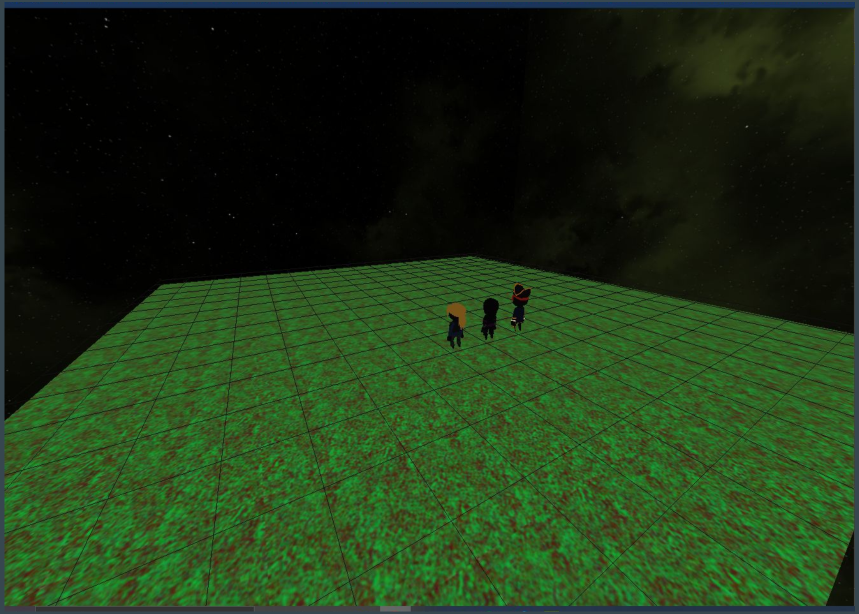


Current Attempt



Skybox

- Added a simple skybox using the tool that Group 5 used.
- Just temporary until we can find another skybox we want to use



LinkedList Implementation Attempt

- Was able to find a great Linked List implementation in javascript, so I used an edited form of that

https://www.geeksforgeeks.org/implementation-linkedlist-javascript/

- At first, I was creating a LinkedList, passing it to the *createModels()* function, then returning the created list for use
    - Doing it this way allowed my list to *appear* populated but only within the *createModels()* function.
    - If I tried to access it outside of its scope, it kept saying that my elements were undefined.

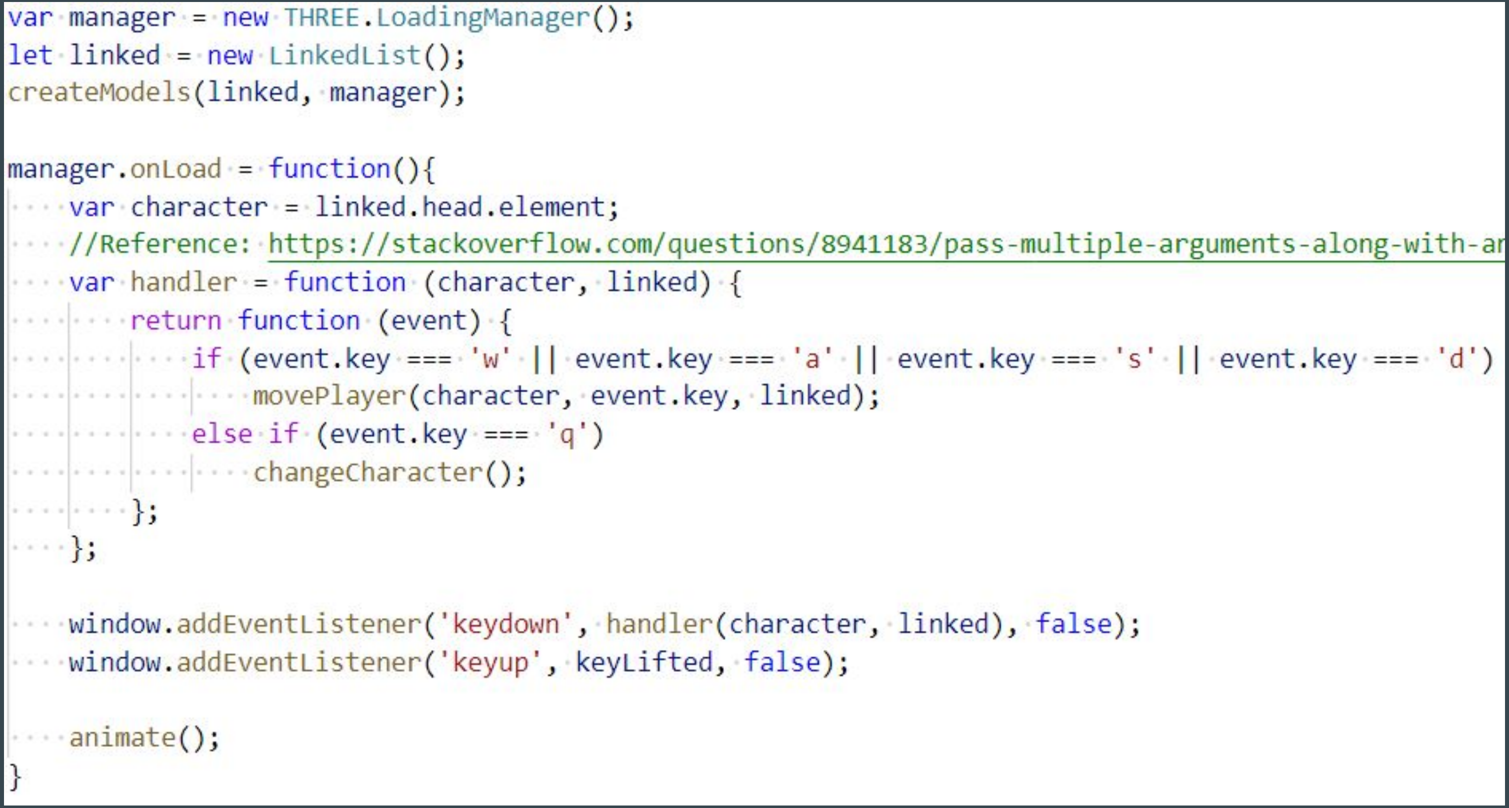
LinkedList Implementation Attempt

- I struggled for a while to figure out what was wrong since it was *partially* working

- I figured it was due to the models not loading completely and knew I needed to use the *onLoad()* function, but wasn't sure where

- I got some help (thanks, Wallace!), and came to this solution

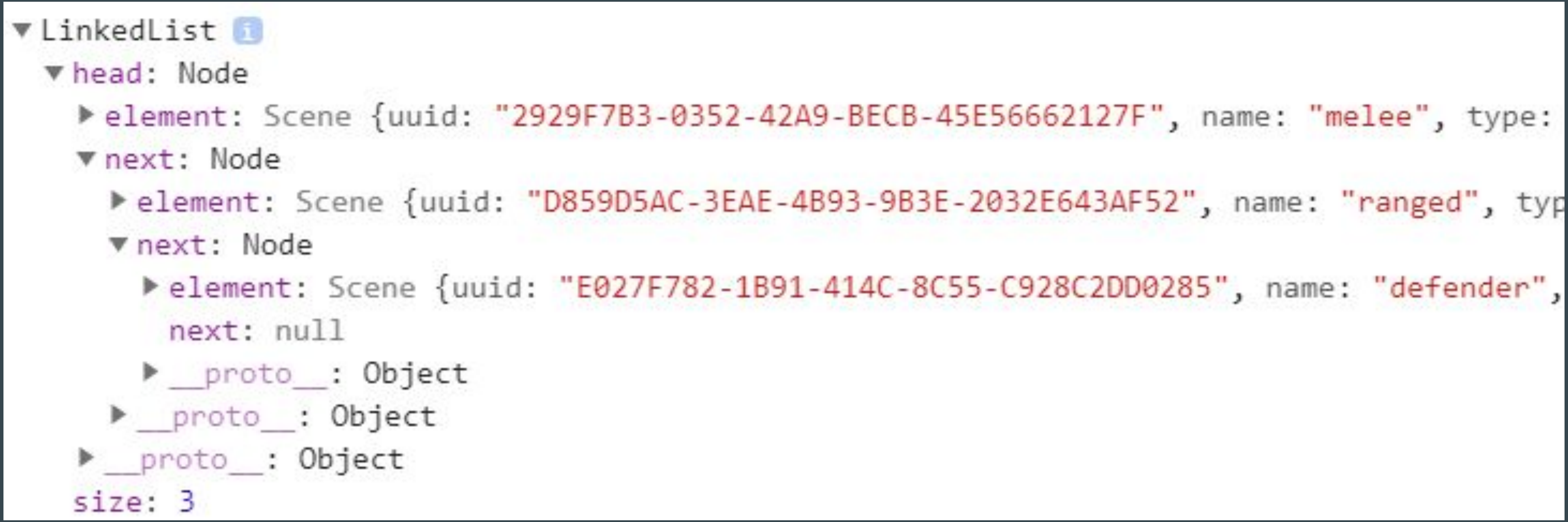
LinkedList Implementation Attempt



```
var manager = new THREE.LoadingManager();
let linked = new LinkedList();
createModels(linked, manager);

manager.onLoad = function(){
    var character = linked.head.element;
    //Reference: https://stackoverflow.com/questions/8941183/pass-multiple-arguments-along-with-ar
    var handler = function (character, linked) {
        return function (event) {
            if (event.key === 'w' || event.key === 'a' || event.key === 's' || event.key === 'd')
                movePlayer(character, event.key, linked);
            else if (event.key === 'q')
                changeCharacter();
        };
    };

    window.addEventListener('keydown', handler(character, linked), false);
    window.addEventListener('keyup', keyLifted, false);

    animate();
}
```

LinkedList Implementation Attempt

- This is what shows in the LinkedList currently



```
▼ LinkedList
  ▼ head: Node
    ▶ element: Scene {uuid: "2929F7B3-0352-42A9-BECB-45E56662127F", name: "melee", type:
    ▼ next: Node
      ▶ element: Scene {uuid: "D859D5AC-3EAE-4B93-9B3E-2032E643AF52", name: "ranged", typ
      ▼ next: Node
        ▶ element: Scene {uuid: "E027F782-1B91-414C-8C55-C928C2DD0285", name: "defender",
          next: null
        ▶ __proto__: Object
      ▶ __proto__: Object
    ▶ __proto__: Object
  size: 3
```

This week's useful findings...

- *var* vs *let*
  - *var* can be used outside of the scope (global)
  - *let* scope is limited to the block, statement, or expression in which it is used
- *manager.onLoad()*
  - Great way to only allow access once everything is loaded and ready

Our next steps

- Emily
  - Cycle between characters using a Linked List
  - Add enemies
  - Utilize *let*
- Carson
  - Finish implementing flood fill
  - Refactor my code
  - Assist the others in any way I can
- Mat
  - Finish debugging Diamond-Square algorithm
  - (try to) Add animation code into the model factory (AnimationMixer)