Week 1 Presentation

Carson, Emily, & Mat

Game ideas:

- Simple
- Turn based
- Tactical RPG combat
- Roguelike

Our idea: A simple, turn based, tactical roguelike RPG

Or as we like to call it

Tacti*cool* RPG

Tactical RPGs

- Descended from table-top role playing games and wargames
  - Specifically, the combat rules of TTRPGs
    - Dungeons & Dragons, Warhammer
- More complicated Chess
  - Battles take place on a map, usually overlaid with a grid.
  - Characters are organized into teams
    - Player controlled team
    - Enemy A.I. team
- Turn based
  - One character acts at a time
  - On a character's turn they have a variety of possible actions
    - Usually limited to a "Move" action and another combat action
  - Turn order typically determined by character stats plus some random element

Tactical RPGs

- Isometric/top down view
  - Gives aerial view of the combat
  - Provides meta information about the current battle
- Characters have specific jobs or classes
  - Jobs tend to be grouped by combat role
    - Tanking, healing, buffing/debuffing, casting, etc...
  - Each job tends to have strategic strengths and weaknesses
- Characters typically grow in strength and ability
  - Experience points are given for completing actions successfully
    - Usually per turn
  - Sometimes, “character” level is tracked separately from “job/class” level
    - Base stats (Attack, Defense, Health) grow with character level
    - Increasing job levels grant new abilities

Tactical RPGs

Top left to bottom right:

- Final Fantasy Tactics
- Fire Emblem: Three Houses
- XCOM: Enemy Unknown
- Mario vs. Rabbids: Kingdom Battle

Roguelikes

- Descended from *Rogue*, a 1980 computer game
  - Players were presented with a procedurally generated dungeon
  - When a player died, all progress would be lost
  - The game was popular and inspired several other titles with similar mechanics
  - Eventually, the term “Roguelike” was applied as a genre
- Stages are randomly generated
  - Usually pieced together algorithmically from pre designed rooms
  - Enemies, hazards, rewards and exits are also randomized
- “Permadeath”
  - Character death is permanent on each playthrough
  - This is intended to give the player a short, fresh experience with each playthrough

Roguelikes

- Evolved further into “Rogue-lites” in the 2000’s
  - Tends to refer to games of other genres with roguelike mechanics
    - Action Platformer, Shooter, Adventure
  - Sometimes incorporate various persistent rewards
    - Better starting conditions
    - New weapons/equipment/abilities
- Today, those terms tend to be used interchangeably, but they always include:
  - Randomization
  - Permadeath

Roguelikes

Top Left to Bottom Right

- Enter the Gungeon
- Dead Cells
- 20XX
- The Binding of Isaac

The Blend

- At the start of a new run, you will get a randomized squad of five
  - Each “Actor” will have one of nine unique jobs
  - Jobs fall into three categories
    - Melee, Ranged and Defender
  - If a member of your squad falls in combat, that member is gone forever
- Your party will grow in strength throughout the run
  - Actors will get gain experience levels throughout the run
    - Builds overall strength of your party
- One member of your squad is the commander
  - The commander will be a bit more powerful
  - Any persistent rewards would be applied to the commander
  - If the commander falls in battle, game over

The Blend

- Your squad will face a randomized set of strategic battles
  - Battles will increase in difficulty
  - The player will receive some sort of reward after each one
- Battles will take place on procedurally generated boards
  - Shape, size and obstacles will be randomly chosen
- All Actors have strengths and weaknesses
  - The base strengths and weaknesses of each category or job are static
    - Melee > Ranged > Defender (rock paper scissors)
    - Specific jobs will have unique s&w
  - Actors may gain elemental s&w as the run progresses
    - Easily distinguishable

Our Process

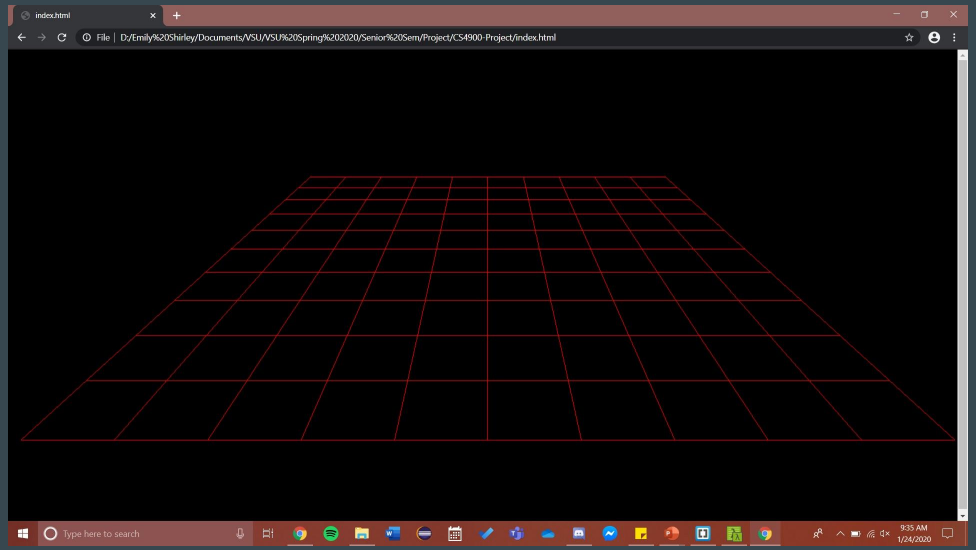
- I began by:
  - Creating a floor
  - Adding a Texture to the floor
  - Loading a cube object on top of the floor
  - Locking a camera on said cube
  - Using Q and E to rotate the camera around the cube
  - Adding more cubes
  - Using the Z key to swap which cube the camera is focused on
  - Being able to rotate the camera around the cube it is currently set to

Our Process

- Issues I Faced:
  - The only issue I faced was changing my camera rotation from a formula within our animate function to using OrbitControls.

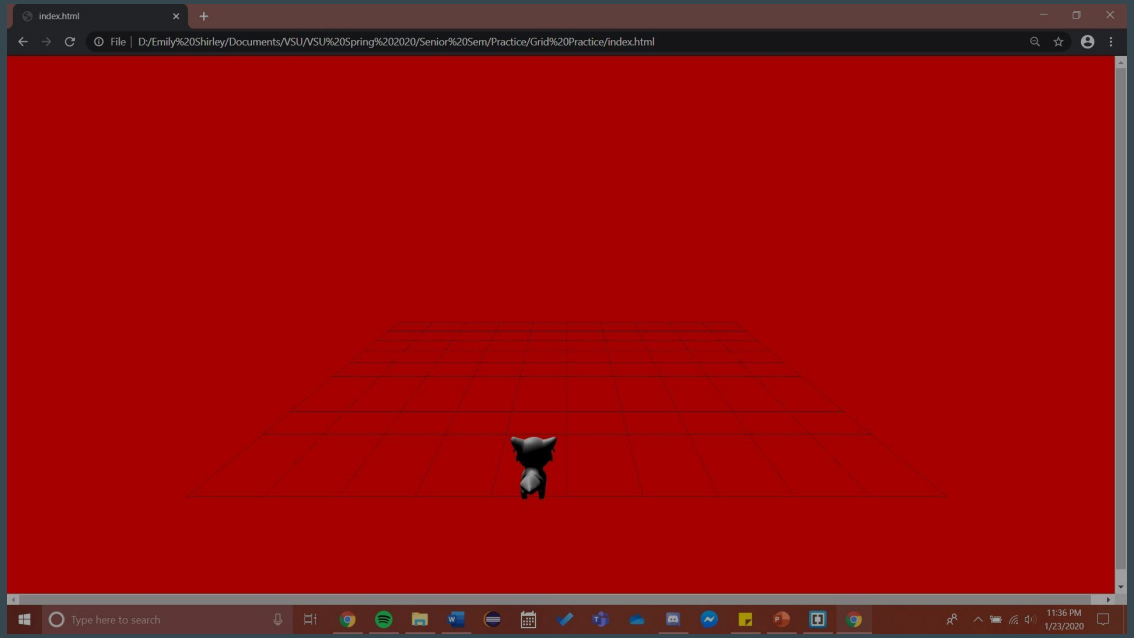
Our Process

- I discovered GridHelper
  - There are numerous other helper classes that can be easily found on threejs.org
- Kept editing the camera position so that it is at a good angle with respect to the grid



Our Process

- Next, I imported a free cat model on top of the grid and added lighting so that it is visible.

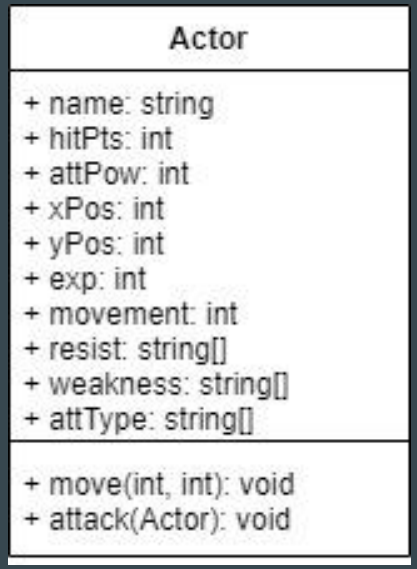


Our Process

- The last thing I did was to change the cat's position based on a key press using the w, a, s, and d keys.
- Each key on the keyboard has a key code that can easily be used to control what key presses do by using *event.key*
- Tips and obstacles:
  - Imports are IMPORTant
  - You may have loaded something correctly, but it just isn't in view
    - The lighting may also play a factor in visibility
  - Trial and error process

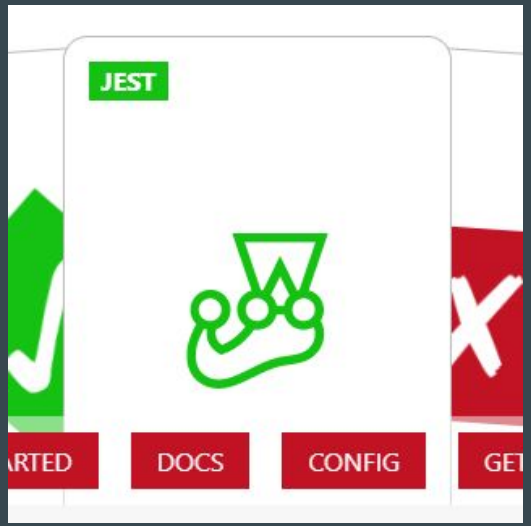
Object Planning and Obstacles

- Started by considering the characters and their basic functions
  - Need to move, attack, keep track of stats
  - Realized I wasn't familiar with OOP in JavaScript, freshened up
- Started the Actor class
  - Simple for now
  - move() was not totally necessary yet, but simple to implement
  - attack() has to make some checks, so when actors.js was finished I needed to begin testing



Jest: “a delightful JavaScript Testing Framework”

- I wanted to begin familiarizing myself with other libraries and frameworks
  - Searched for JavaScript testing frameworks.
    - There was a lot to sort through
- Jest is Facebook’s js testing framework
  - Often used for testing node & angular apps
  - Has loads of functionality
  - Greatly exceeds our needs for now
    - Can be used for simple test cases
  - jestjs.io/en/



To use Jest

- In order to properly use Jest, you will also need node.js
  - Node.js can be found at nodejs.org/en/
  - Node comes with a package manager called npm
    - Npm will ensure that your directory has all necessary dependencies
    - In the directory, run npm --save-dev jest
- With Jest installed, you can begin writing test cases
  - I wrote my tests to check specific values after specific actions using “Matchers”
    - Matchers like toBe() check a value explicitly
    - Matchers like contain() check a collection for the specified value

```
test('Ranged values check', () => {
    let m = new Ranged('Rick');

    expect(m.hitPts).toBe(10);
    expect(m.weakness).toContain('Melee');
    expect(m.attType).toContain('Ranged');
})
```

```
  √ Base actor attack test (4ms)
  √ Melee values check (1ms)
  √ Defender values check (1ms)
  √ Ranged values check
  √ Modified attacked test (1ms)
  √ Modified resistance test

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        4.475s
Ran all test suites.
PS C:\Users\Mathieu Davidson\Desktop\CS4900>
```

Back to Objects

- Lastly, once I had a class to test and test cases written, I ran into another hiccup
  - Jest was not a fan of the way I was attempting to import my custom class
    - Hit the JavaScript books again
  - As it turns out, this was due to my lack of an export statement followed by export statements with bad syntax, followed by more bad syntax in my import
    - After some research and experimentation, I found a way that worked for me

```
73  module.exports.Actor = Actor;
74  module.exports.Melee = Melee;
75  module.exports.Defender = Defender;
76  module.exports.Ranged = Ranged;
```

```
1  Actor = require('./actors.js').Actor;
2  Melee = require('./actors.js').Melee;
3  Defender = require('./actors.js').Defender;
4  Ranged = require('./actors').Ranged;
```

This week's useful findings

- GridHelper
- *event.key*
- *scene.getObjectByName("name")*
- OrbitControls

Our next steps

- Carson:
  - Have our character model always look in the position of the cursor on screen
- Emily:
  - Contain model within a grid
  - Add texture to model
- Mat:
  - Implement range considerations for Actors

...and any additional ideas we come up with!