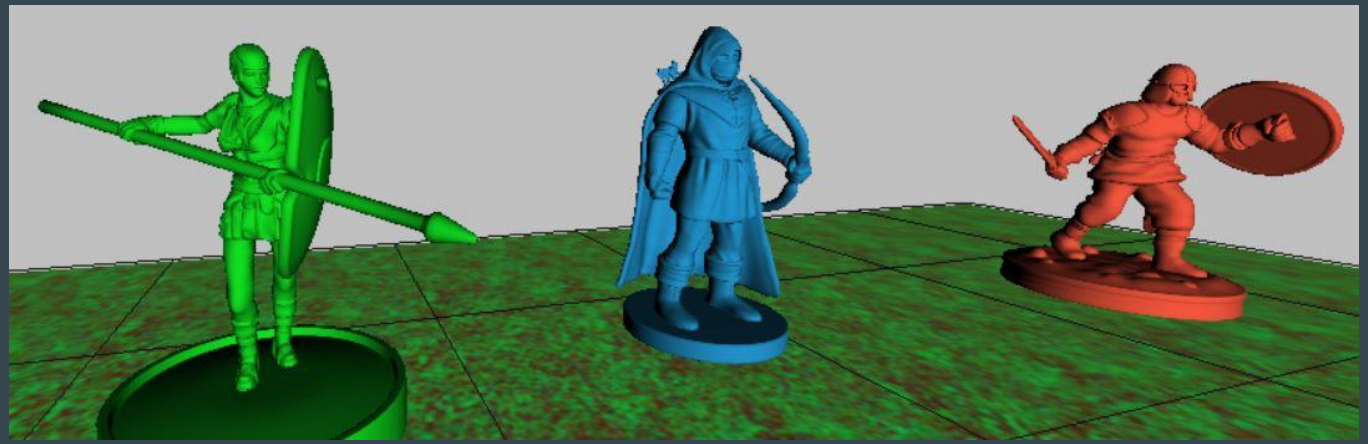
Week 3 Presentation

Carson, Emily, & Mat

Models and Actors

- I started out by closely examining our code
  - I wasn't very familiar with the graphics
- I devised a way to load models generically
  - Mostly by hacking code together
  - Worked by passing strings to a function that loaded the model
  - Needed new models



Things got weird

- As we were warned, I ran into loading problems
  - When trying to manipulate a new model (hard coded), its position couldn't be accessed
  - I hadn't considered the size of the models
    - Meant for 3D printing... so, yeah...
      - Gotta fix that
  - The code to access the model's position was reached before the model finished loading

Enter LoadingManager

- Part of Three.js
- Provides a way to keep track of assets as they load
  - onLoad, onProgress, onError
- https://threejsfundamentals.org/threejs/lessons/threejs-game.html
- https://blackthread.io/blog/progress-bar/
- As for me...
  - I have the LM working by itself, but the onLoad function is being triggered early for some reason I haven't pinpointed yet

```
const manager = new THREE.LoadingManager();
manager.onLoad = init;
const models = {
  pig:    { url: 'resources/models/animals/Pig.gltf' },
  cow:    { url: 'resources/models/animals/Cow.gltf' },
  llama:  { url: 'resources/models/animals/Llama.gltf' },
  pug:    { url: 'resources/models/animals/Pug.gltf' },
  sheep:  { url: 'resources/models/animals/Sheep.gltf' },
  zebra:  { url: 'resources/models/animals/Zebra.gltf' },
  horse:  { url: 'resources/models/animals/Horse.gltf' },
  knight: { url: 'resources/models/knight/KnightCharacter.gltf' },
};
{
  const gltfLoader = new GLTFLoader(manager);
  for (const model of Object.values(models)) {
    gltfLoader.load(model.url, (gltf) => {
      model.gltf = gltf;
    });
  }
}

function init() {
  // TBD
}
```

Simple Week

- For this week's progress, I focused on three main things:
  - Highlight visibility
  - Limit movement when a key is held down
  - Changing models to glTF file format
    - Minimum file size
    - Efficient to use within apps
    - *This is in progress as the models and actors get merged*

Highlight Changes

- I ended up putting the highlight mesh objects into an array
- The visibility changes based on the banana's position
- Used the *visible* attribute along with my map boundary constants

```
//set highlight visibility
if(banana.position.z === (mapTopZ)){
    highlights[0].visible = false;
}else
    highlights[0].visible = true;
if(banana.position.x === (mapLeftX)){
    highlights[3].visible = false;
}else
    highlights[3].visible = true;
if(banana.position.z === (mapBottomZ)){
    highlights[2].visible = false;
}else
    highlights[2].visible = true;
if(banana.position.x === (mapRightX)){
    highlights[1].visible = false;
}else
    highlights[1].visible = true;
```

Limit Movement

- Used a simple boolean value (*down*) to prevent a key from firing off the event handler multiple times (when it is held down)
- It is initialized within the *main.js* file and utilized within the moveBanana() function in *objectGeneration.js*
- A simple bananaUp() function is called with the keyUp event and it sets *down* back to false

```
function moveBanana(event){
    //boolean to prevent event from firing again
    if(down)
        return;
    down = true;
    //used to reference the created object
    var banana = scene.getObjectByName("banana");
```

My Primary Focus

- For this week, I did not do much programming. Instead, I decided to dive head first into researching Procedural Generation.
- In the end, I found out what I think would be our best approach for our game, but I want to discuss a major issue I found myself having.

Major Issue I was Facing

- Whenever I had an idea for what I wanted to do, I would look for a way it was already done in Three.js
- If I couldn't find anything, I would become more and more discouraged and try to think of a new way to solve the problem.
- Eventually, I sat down and thought of some way to help with this problem.

How I fixed the problem

- Instead of looking for something someone else had done in Three.js, I began to look at some games in the same genre as ours in general.
- This thought process led me back to think of one of my personal favorite games, XCOM 2, and how it has procedural generation in its levels.
- Eventually, I found a powerpoint from the lead level designer for XCOM 2 from an event called Games Developers Conference (GDC). In it, he fully describes how procedural generation works in their game.

How we want to potentially implement procedural generation

- Plot and Parcel
  - Create a static map
  - Fill the static map with holes
  - Each hole can then be filled with any object designed to fit within that kind of space
  - Direction of these objects can also be changed
- Issues
  - Would need to design a static map and objects to fill the spaces
  - Still need “some” logic for how to fill the holes

Example of Plot and Parcel

Example Continued

This week's useful findings

- glTF file format
  - Easier to use and will probably help in the future when we have a lot of models loading at once!
- The *visible* attribute
  - Came in handy when fixing the highlights
- Plot and Parcel
  - A "simple" way to implement procedural generation

Our next steps

- Emily
  - Keep working with Mat to fully merge the Actor code with the model code
  - Refactor code further to allow for future improvements
  - Find better glTF models (or make our own!)
- Carson
  - Begin designing a static map for our game
  - Come up with objects or obstacles to add to our map
- Mat
  - Troubleshoot onLoad function wackyness
  - Continue working with Emily on model and actor implementations