Week 5: Group 1

Alan Bettis, Ryan Trull, Merritt Hancock, Kenda Blair

Slime Puzzle Game

Weekly Tasks

Flood Fill Implementation

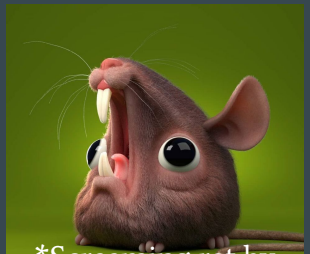
A* Implementation

Camera Refactoring

Model Design/Implementation

Enemies

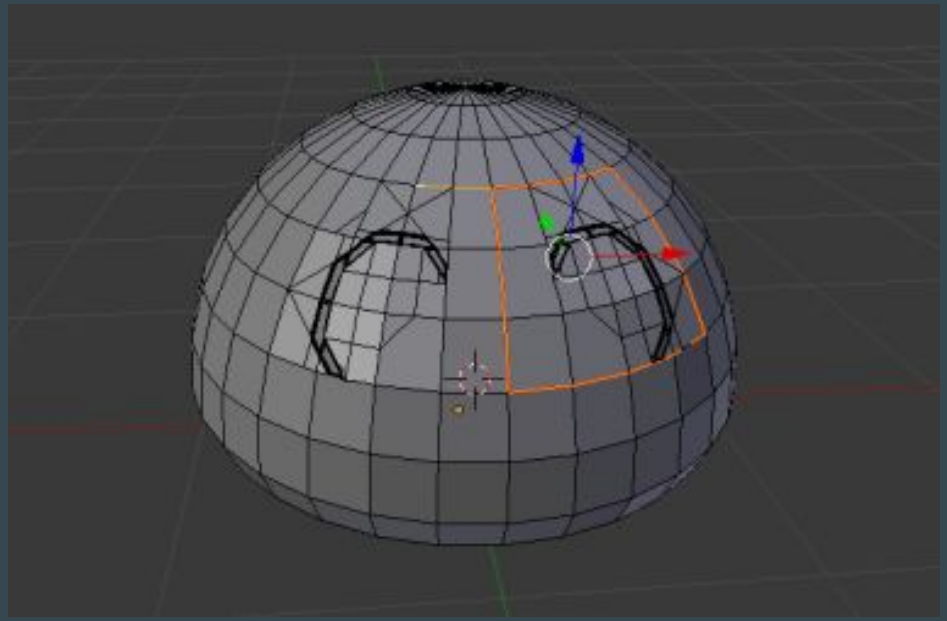
- We discussed and drafted out what kind of enemies we want for the game.
- For the beginning levels, we came up with three base enemies:
  - Verm: Your basic enemy fodder (No ability)
  - Milcap Soldier: Mushroom enemy who carries a stick and shield (No ability)
  - Pinpod: Cocoon type enemy with retractable spikes (Spike ability)



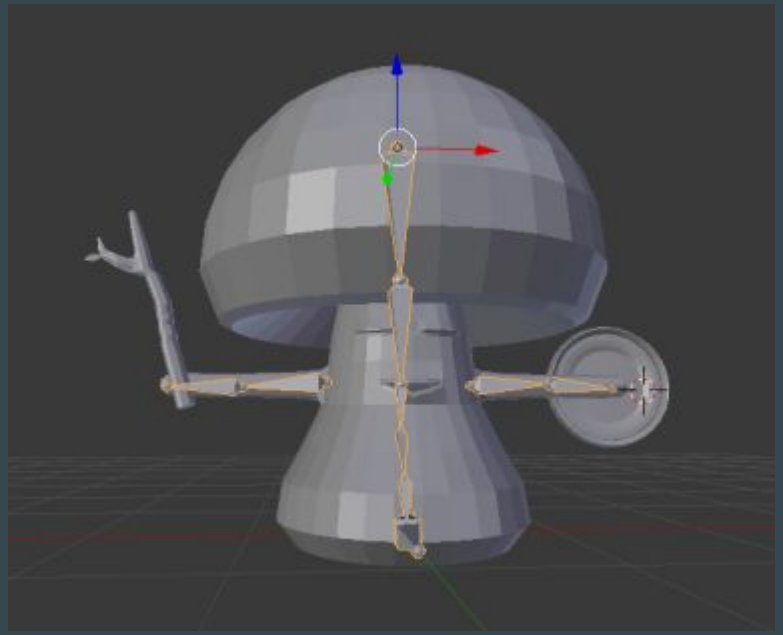
*Screaming rat by Sebastian Montecinos

*Zoomer from Metroid Prime

Models



Slime Main Character



Milcap Soldier

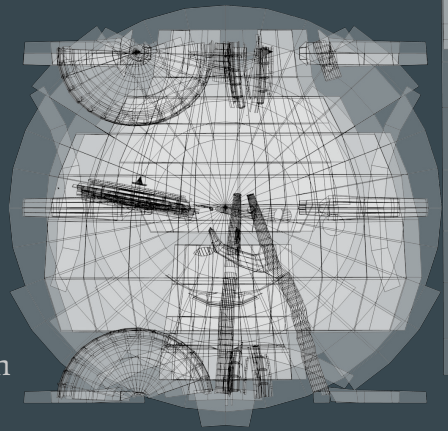
UV Texturing Methods Pros/Cons

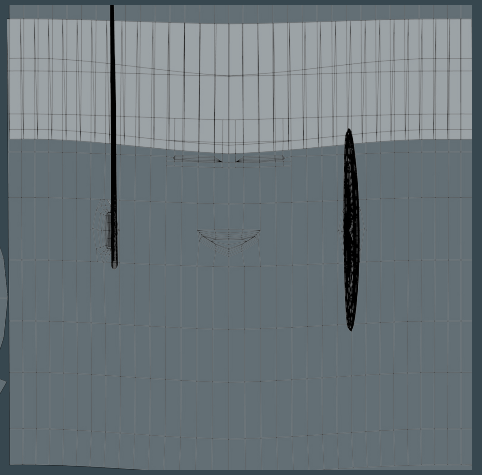
Cube Projection
Pros: simple show of top bottom and sides
Cons: Tends to overlap a lot

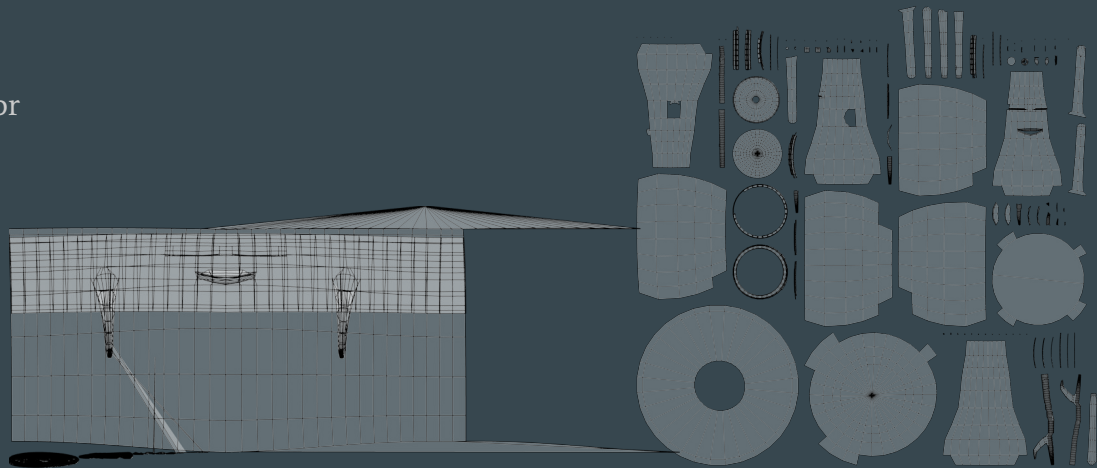
Cylindrical Projection
Pros: One of the best projections for being easily read
Cons: If there are multiple vertical sections they overlap the main mesh

Sphere Projection
Pros:similar benefits to Cylinder mapping
Cons: Stretches more than Cylindrical mapping at equator

Smart UV Projection
Pros: no overlapping faces
Cons: Texture maker needs to know where edges line up



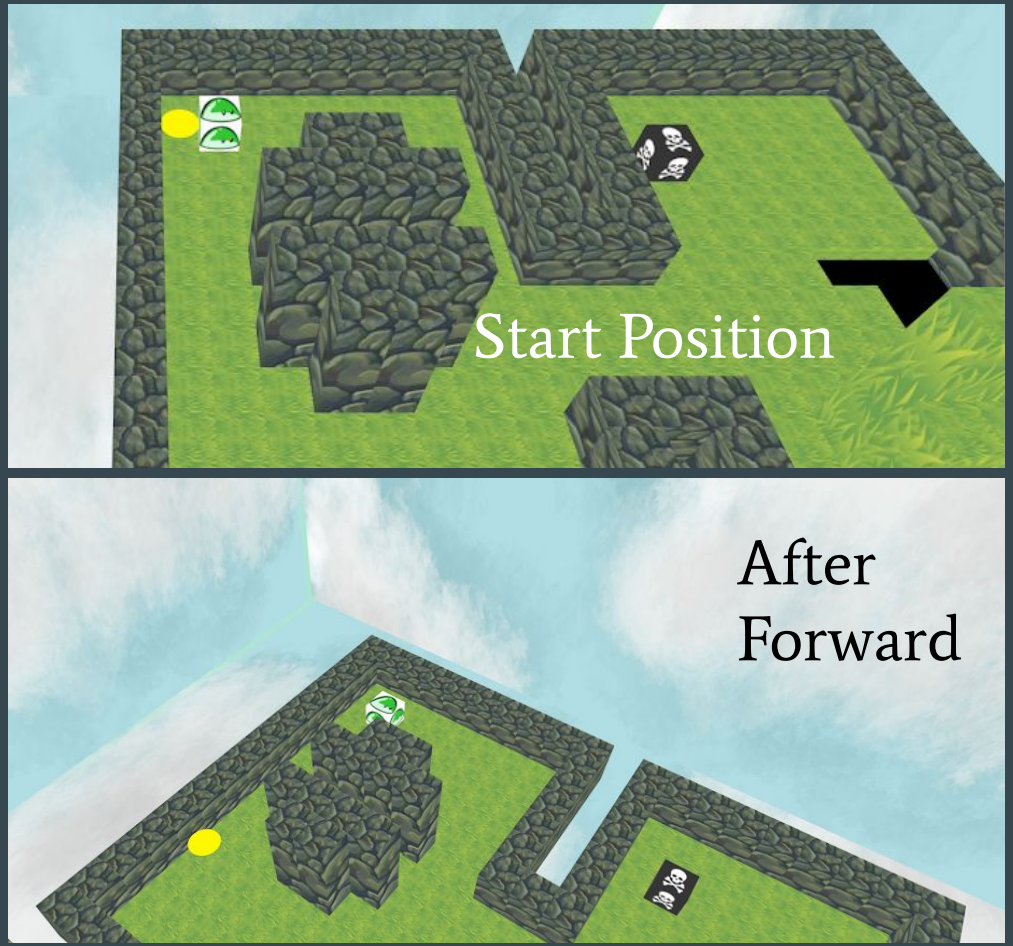




Camera

- Modularization
  - Separate file
  - Limited Viewing Angles
- Panning Problems
  - Increment camera position
  - Solutions?

```
case “right”:
    camera.position.x += 1;
    break;
```



Flood Fill Overlay

- Our original, base implementation of the flood fill algorithm did not take into account heights and walls, and also caused some lag upon generation.
- Through some refactoring and modification, we changed how the algorithm functions.

Flood Fill (cont.)

Suggestions: increase the speed of Flood Fill implementation when its complete?

Visibility-based instead of object-based:

- On generation of the board, every tile is given an overlay tile which is invisible
- When cursor hovers over a player (and now an enemy), it performs the same algorithm, but instead of creating new objects, it will merely change the visibility.
- When cursor is not on an entity, it will change visibility back to false.

Suggestions: Showing off the code for some of the algorithms would be great.

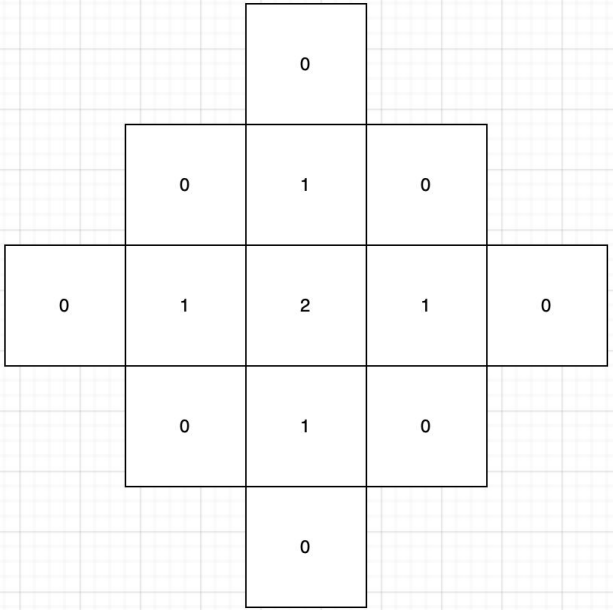
```
//Checks neighboring tiles. To be used by both Flood Fill and A*
function checkNeighbor(entity, sourceTile, destinationTile){
    var maxHeight = entity.jumpHeight;
    var heightDifference = Math.abs(sourceTile.height - destinationTile.height);
    //This variable needs to be gotten from the entity later, but for now we can just
    //use the basic traversability (No water/void/gap spaces)
    var traversableTerrain = [0, 1, 4, 8];

    //Make sure destinationTile exists
    if(destinationTile == null){
        return false;
    }
    //Make sure the destination tile is within the movement range
    //(this is taken care of in flood fill, but not in A*)
    var xDistance = Math.abs(destinationTile.position[0] - entity.position[0]);
    var zDistance = Math.abs(destinationTile.position[2] - entity.position[2]);
    if(xDistance + zDistance > entity.movementRange){
        return false;
    }
    //Make sure maxHeight exceeds the height difference between the tiles
    if(maxHeight < heightDifference){
        return false;
    }
    //Make sure the destination tile is on the list of traversable terrains
    if(!traversableTerrain.includes(destinationTile.type)){
        return false;
    }
    //Make sure the destination tile isn't occupied
    if(destinationTile.occupant != null){
        return false;
    }
    return true;
}
```

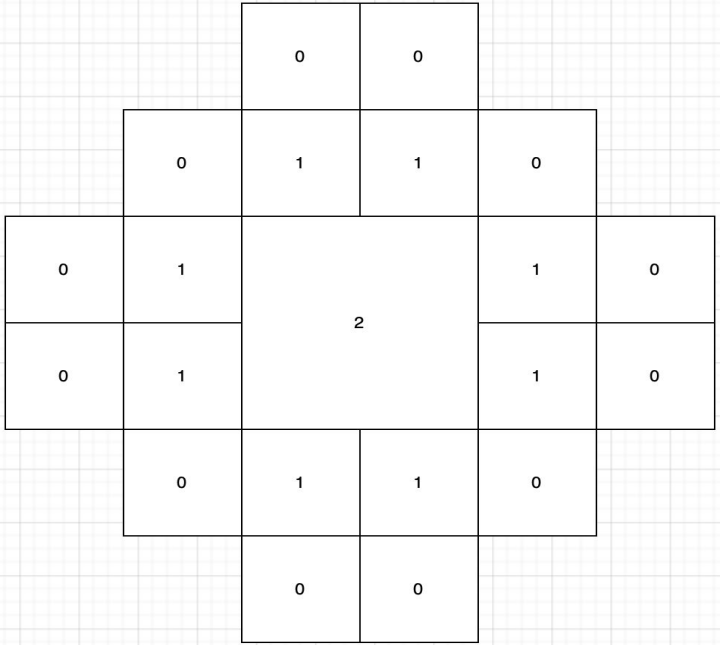
```
function movementOverlay(x, z, range, board, entity){//uses the flood fill algorithm to create overlay
    if(range>=0 && x >= 0 && x < board.overlayMap.length && z >=0 && z < board.overlayMap[x].length){
        //Do not render an overlay tile that has an entity in it
        if(board.tileArray[x][z].occupant == null){
            board.overlayMap[x][z].overlay.material.visible = true;
        }
        //recursive call for surrounding spaces
        if(checkNeighbor(entity, board.tileArray[x][z], board.tileArray[x+1][z])){
            movementOverlay(x+1, z, range-1, board, entity);
        }
        if(checkNeighbor(entity, board.tileArray[x][z], board.tileArray[x][z+1])){
            movementOverlay(x, z+1, range-1, board, entity);
        }
        if(checkNeighbor(entity, board.tileArray[x][z], board.tileArray[x-1][z])){
            movementOverlay(x-1, z, range-1, board, entity);
        }
        if(checkNeighbor(entity, board.tileArray[x][z], board.tileArray[x][z-1])){
            movementOverlay(x, z-1, range-1, board, entity);
        }
    }
}
```

Moving forward with Flood Fill

Player Size 1 Flood Fill



Player Size 2 Flood Fill



A*

- A* is now implemented... But at what cost?
- Pathfinding.js uses Node.js syntax: const circle = require('./circle.js');
  - We and many others use ES6: import {Circle} from “circle.js”;
- Took modified chunks of relevant code from Pathfinder and used ES6 import/export statements

Future Tasks

Loading Screen

Working Test Enemy

Continue Model Development