Zakk Loveall
Austin Barner
Vihan Garg
12/08/2023
Final Planning Document

1. **Overview:**
    a. The final project we've settled on is our terrain traversal AI idea. Essentially the idea is to use Austin's code that he wrote for generating 2D terrain (pseudo code and images below), and transfer it over from Java to C# in the GoDot engine. What this code will do is use a perlin noise library (provided by GoDot) and create a perlin noise map. We then give the terrain generation algorithm values that will generate the terrain based on the perlin noise map. These values range from 0.1 to 1 and are cutoff values for the gray scale perlin noise map where darker areas will be associated with certain terrain tiles (land/sand) and lighter areas will be associated with empty tiles (water). These darkness/lightness of a pixel can be determined through a value from 0.1 to 1, which is what our cutoff will be based on. After our terrain is generated, our agent, possibly represented by some basic 2D pixel art, will use an A* algorithm along with a heuristic we come up with to navigate this terrain. There will be a clearly marked start and end tile in the terrain map that the agent will navigate to and from. Austin's code is tile based, which works perfectly for our purposes. Reference our intermediate document for more details.
2. **Bios:**
    a. **Austin:** Working on integrating his terrain generation algorithm into GoDot (C#) along with any UI based code.
    b. **Zakk:** Working on implementing the A* algorithm with the heuristic that will navigate the terrain. Also helping Austin with UI code if needed.
    c. **Vihan:** Working on implementing the A* algorithm with the heuristic as well that will navigate the terrain. Also doing all the pixel art needed for the project.
3. **Functional Requirements (MVP):**
    a. An executable file that the user can open.
    b. A UI that displays the generated terrain and an agent.
    c. A visible start and end location.
    d. An agent that can move throughout the terrain.
    e. This agent's moves will be based on a working heuristic.
4. **High-level Overview:**
    a. Working terrain generation in C# (pseudo code below).
    b. Working display functions for the UI.
    c. A function for checking to see if the agent can get to the end.
    d. A heuristic function that works with the terrain.
    e. A working A* algorithm (pseudo code below).
5. **Top Goals:**
    a. Terrain generation has been successfully moved from Java to GoDot.

b. The agent can be seen on the screen and makes moves based on a heuristic.
c. A finished heuristic function that allows the agent to make the right movements.
d. We'll know when we've hit the first goal because we will be able to run the program and see tile-based 2D terrain that generates differently each time it's opened. We'll know when we've hit the second one once the agent can be seen on the screen and is making movements (this doesn't mean correct movements). We'll know we've hit our third goal once our agent is able to successfully navigate from the starting position in the terrain to the ending position.

6. **Steps:**
    a. **Christmas Break:** Getting familiar with GoDot.
    b. **January:** Austin, Vihan, and Zakk will all work on getting Austin's code transferred to GoDot/C#. This is most likely going to be the most tedious part.
    c. **February:** Finish transferring code, get some pixel art on the screen that can move around (maybe just via keyboard input at first). Also get collision with terrain working. Vihan and Zakk will get the agent on the screen and moving, Austin will focus on collision.
    d. **March:** Get appropriate heuristic function written that will successfully move the agent from it's starting point to it's ending point. Vihan and Zakk will do this. Austin will help with getting the terrain to account for the heuristic movement logic.
    e. **April:** Polish up project to be ready to present. Potentially get a stretch goal if time is there. We will all work on polishing the project.
7. **Stretch Goals:**
    a. When the agent reaches the end of maze, automatically start a new simulation.
    b. Add genome-based system that agent will operate off of instead of heuristic.
    c. Add other agents that our agent can interact with.
    d. Breed with other agents

2D terrain generation pseudo code:

https://docs.godotengine.org/en/3.5/classes/class_opensimplexnoise.html
<Include a library called OpenSimplex2S which gives access to the method OpenSimplex2S.noise3_ImproveXY(seed, x, y, z)

```
enum TileType {
   BARRIER,
   EMPTY
}

private Tile[][] grid
private ArrayList<Tile> allEmptyTiles = new ArrayList<Tile>();
private Tile start;
private Tile end;
```

```
function generateTerrain(seed, width, height, terrainScalar, barrierThreshold):

   If seed is < 0:
     throw InvalidSeedException();
     return null;

   instantiate the 2D array ‘grid’ to an empty 2D array of dimensions  height x width

   for int i = 0; i < height; i++ {

         for int j = 0; j < width; j++ {

            double noiseVal = OpenSimplex2S.noise3_ImproveXY(
               seed,
               j * terrainScalar,
               i* terrainScalar,
               0.0
            );

            if noiseVal >= barrierThreshold :
               grid[i][j].tileType = TileType.BARRIER
            else:
               grid[i][j].tileType = TileType.EMPTY
               allEmptyTiles.add(grid[i][j]);
         }
   }

 return the grid

function setStartAndEnd(minDistance, maxAttempts) {

         if minDistance >= grid.length / 2 || minDistance >= grid[0].length / 2 :

                  throw new MinDistanceTooLargeException();
                  return null;

         int attempts = 0
         while true:

                  start_candidate = random Tile from allEmptyTiles

                  end_candidate = random Tile from allEmptyTiles that is not
start_candidate
```

```
                if distance between start_candidate and end_candidate is >= minDistance
&&
                   start_candidate and end_candidate have an open path to eachother

                        start = start_candidate
                        end = end_candidate
                        return
                attempts++

                if attempts == maxAttempts :
                        throw new MaxAttemptsReachedException()
                        return


}
```

A* Algorithm for Terrain Navigation:

```
function aStar(start, goal, terrain):
   // we will assume cells are tiles
   // assume empty cells are blue tiles/have blue on the tile
   // non empty tiles are other colors.
   openSet = Priority Queue with start cell (initially empty)
   cameFrom = empty map
   gScore = map with default value of infinity for all cells
   gScore[start] = 0
   fScore = map with default value of infinity for all cells
   fScore[start] = heuristic(start, goal)

   while openSet is not empty:
      current = cell in openSet with the lowest fScore
      if current == goal:
         return reconstructPath(cameFrom, current)

      remove current from openSet

      for each neighbor of current:
         tentativeGScore = gScore[current] + distance(current, neighbor)
         if tentativeGScore < gScore[neighbor]:
            cameFrom[neighbor] = current
            gScore[neighbor] = tentativeGScore
```

```
                fScore[neighbor] = gScore[neighbor] + heuristic(neighbor, goal)
                if neighbor not in openSet:
                    add neighbor to openSet

    return failure (no path found)

function reconstructPath(cameFrom, current):
    totalPath = [current]
    while current in cameFrom:
        current = cameFrom[current]
        add current to totalPath
    return reverse totalPath

function heuristic(cell, goal):
    // Challenge will be figuring out how to implement this function.
    return distance between cell and goal
```

Actual terrain generated by the code Austin wrote: