Zakk Loveall
Vihan Garg
Austin Barner
11/3/23
Senior Design 1
Intermediate Planning Assignment

Updates:

- Our scope for our project has changed drastically because we realized our previous scope was simply not achievable in a reasonable time period. Our new scope involves simply using an AI algorithm like A* to train an agent to navigate a pseudo-randomly generated terrain, consisting of empty spaces and barriers. There will be two main parts to this project, writing code to generate the terrain and writing code for the agent to navigate said terrain. We still plan on using GoDot to simplify the process of creating a UI for our simulation. We will use OpenSimplexNoise (an open-source perlin noise generator) and a tile map to generate our terrain. OpenSimplexNoise is a library provided by GoDot, which will generate the noise map that will be used to generate our 2D terrain. Once the terrain has been generated, a start tile and end tile will be picked. There will be a minimum distance, n, that the start tile and the end tile will be from each other. Furthermore, the terrain generator will work with both a pre-set seed or a random seed, as well as various other presets. This flexibility in world generation will allow for our agent to be trained in different “environments” and potentially even scenarios. OpenSimplexNoise makes the entire process incredibly easy to implement. Our agent, which will be represented as some simple pixel art (subject to change) will use a heuristic to determine the best move it will need to take to navigate the terrain. It will then make a move based on this heuristic. Once it reaches the end tile of the terrain, the simulation will end. One of our stretch goals is to generate another terrain map for the agent to navigate after it reaches the end of the first one. We wrote some pseudo code located below in the document to hopefully help better illustrate a potential A* algorithm that our agent can use to navigate the terrain. Alongside this pseudo code is actual code that Austin wrote in Java that will generate the terrain using the exact tile-based system that we will need. This is the main part of the code that explains our logic the best, if needed, Austin can send you the entire Java project for confirmation. Our biggest two challenges that have yet to be solved in a meaningful sense are determining a good heuristic function for the agent to operate off of, and integrating the terrain generation algorithm into GoDot. We will also have to adjust a couple of the values in the noise map to ensure that there is always a path from the start tile to the end tile. Austin has written most of the code for generating terrain that will work with our implementation of A* (located below) so the hard part here is done. It will be tedious to transfer it from Java to C#, which is what GoDot uses, but thankfully GoDot makes this process nearly seamless by providing the same libraries that Java does.
- Our focus is to simply create a simulation where terrain is randomly generated and an agent is trained to navigate this randomly generated terrain. A couple of our new stretch goals now include, generating another terrain map for the agent to navigate after it

finishes its current terrain map, adding a genome structure that our agent can utilize to 'evolve' (reinforcement learning maybe?) over time as it navigates through these terrain maps, and changing from basic terrain navigation to world navigation.
- Our plan/timeline has remained the same, but it's been adjusted to account for our new scope.

What we will deliver:

- We can promise to deliver an executable file that, when opened, will have a UI with pseudo-randomly generated terrain and an agent in a starting position, and the user can see this agent attempt to navigate the terrain and make its way to the end as it learns which move is the best to make. There will be a clearly marked start and end spot that the user can identify (maybe via color or text on the tile). More in-depth descriptions of these processes above, along with pseudo code below.
- Below is the terrain generated by the code that Austin wrote. The space that the agent can't move into is the green/yellow space, and the agent that the agent can move into is the blue. The white spot would be the starting location, the black spot is the end location. We also have pseudo code written that will hopefully give a better idea of how our project will work.
- By changing the scope of our project, our roles have become more clear. We now realize that Austin will focus on the terrain generation, Vihan and Zakk will focus on integrating an A* algorithm (or something similar) to generate the best move that the agent will make. Vihan will focus on any pixelart that might be needed for our UI, and Zakk will help Austin with the terrain generation if needed.

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: