Jessica May

Joshua Linge

Assignment 1 Report

**Chosen Game Engine: Unity**

One of the reasons we chose Unity was because we have some prior experience with Unity and C#, and wanted to learn more about it. Unity also provides many features and libraries that other game engines would require us to create ourselves. This makes Unity an easier engine to work with, and also less time consuming. For our game we needed a 2D engine, and Unity just recently updated to provide necessary 2D framework. Unity has built in physics, collision detection, and ray-casting, which are all very important for developing a game. We also thought it was very important for our project to be cross-platform during development and for the end users. Unity provides this initially, where other game engines do not. We want our game engine to be powerful and robust, and Unity fits this goal.

**Report**

Background.cs

This script is small, but is the essential initializer of the game. It sets the size of our map, and initializes the Map and Grid in its StartUp() method. Inside Background's Update() method, the user can left-click in order to place and remove walls, and right-click to add Agents to the map. This is done by translating mouse coordinates to world coordinates.

Agent.cs

The Agent script is used to create Agents, and is the parent of the Player class. Agents have a heading, velocity, cell index, and radius. The Start() method initializes these. Agent's Move() method allows for Agent objects to move around the map by using the objects velocity, heading, and cell indexes of the grid. Only the Player object uses this method currently. The getLengthOfFeelers() method returns an array of vectors that gives the current length of each feeler given the number of feelers needed, the max length of a feeler, and the view angle of the Agent object. Based off of the number of feelers and a view angle, the angle for each feeler is calculated to ensure the angular distance between each feeler is equivalent. After the direction of each feeler is calculated, we use Unity's Physics library to cast a ray in that direction starting from the radius and going outwards. These rays are limited to a length of the given max feeler length value. If the feeler is not colliding with anything, it will have a magnitude of the max feeler length, otherwise the magnitude will equal the distance from the collision to the object. There are also two getter functions for cell index and radius.

Player.cs

Player extends Agent. The Start() method initializes the Player. Within the Update() method, the game takes in keyboard input. This includes movement of the Player agent and debug keys. This method also gets the current length of the feelers, and current nearby agents. OnGui() is called every frame to draw something onto the environment and is from the MonoBehaviours class.

Grid.cs

Grid is an organized container for all of the agents on the map. It consists of a 2D array of cells, and each cell can contain multiple agents. Agents are added to a cell based off of their coordinates in the world space. The constructor generates a new grid of a given width and height. Then follow two inBounds() methods, one taking in a vector of coordinates, and one taking in x and y coordinates. These methods check to see if the cell index of the given coordinates is within the bounds of the grid. The add methods will add a given agent to the grid. If the location given to these functions is not valid, they will return false. The remove methods will remove a given agent from the grid. They return false if a valid agent is not passed in. The move method is called when an agent moves from one grid cell to a new one. This updates the agent's location in the grid. The getNear method takes in an agent and a radius and returns a list of agents. This method is used when trying to locate adjacent agents. Based on the given agent's center and radius, it finds the range of grid cells that can be detected by the given agent. It then loops through each of these cells, and if any other agents are found, they are added to the list of agents to be returned.

Map.cs

Map is similar to Grid, however it deals with the placement of walls instead of agents, and each cell can only contain one wall. The constructor generates to start map for each run of the program, which consists of a background and a border of walls around the edges. The createBorder() method is the implementation of creating the border walls that is called in the constructor. The inBounds method verifies that a given wall's coordinates are in bounds of the map. GetCellIndex() returns the cell index the given wall is in. CellIndexToWorld() translates a given cell index to a world coordinate. AddWall() verifies the coordinates of a given wall and then adds that wall to the map by calling placeWall(). RemoveWall() removes a selected wall from the map. The user can no longer see this wall, and the Player agent can now move where there once was a wall.

**Lessons Learned**