Aaron Gunther

Lee’s Algorithm

We decided to go with a grid based game in our project, because of this the method we decided to implement for path finding was Lee’s Algorithm. The algorithm initially virtualizes that in game grid, in order to perform all calculations on the virtualized grid before implementing them on the in-game unity grid made up of game objects. The grid is virtualized by running through the children of the game object “GridLayout” which are all “tiles” on the grid, by running through these children in order the algorithm is able to label each tile with its corresponding (x,y) point as shown:



(6,6)

(5,6)

(4,6)

(3,6)

(2,6)

(1,6)

(0,6)

(6,5)

(5,5)

(4,5)

(3,5)

(2,5)

(1,5)

(0,5)

(6,4)

(5,4)

(4,4)

(3,4)

(2,4)

(1,4)

(0,4)

(6,3)

(5,3)

(4,3)

(3,3)

(2,3)

(1,3)

(0,3)

(0,2)

(1,2)

(2,2)

(3,2)

(4,2)

(5,2)

(6,2)

(6,1)

(5,1)

(4,1)

(3,1)

(2,1)

(1,1)

(0,1)

(6,0)

(5,0)

(4,0)

(3,0)

(2,0)

(1,0)

(0,0)

At this point a start point for our AI to start at and a goal point for our AI to find are generated randomly by the algorithm. At the same time the tiles containing rocks are labeled with a score of -1 in order to ensure that they are ignored by the pathing part of the algorithm. Next the pathing part of the algorithm takes place, starting from the AI start point each tile around the AI is labeled with a number corresponding to the distance away from the start point. Then a neighboring tile next to the AI is selected at random and all of its neighboring tiles are also labeled, this is repeated until the finish point is found as shown:



5(stop)

5

4

3

4

2

-1

2

-1

2

2

2

-1

2

-1

1

1

1

1

1

-1

1

Start Point (0)

Once this section of the pathfinding stops the AI works backwards from the finish point, finding the first lower value, then the next, and the next, until the start point is found as shown:



0

1

2

3

4

5

These nodes are then returned as a list which is iterated through in order to transition the AI’s sprite throughout the tiles in order to make the AI appear as if it is moving. Once the AI reaches the finish point the finish point is again randomly generated in order for the program to continue functioning.