Armaan Esfahani

Wednesday, February 22, 2017

CS 1301

Pseudo Code/Algorithm Explanation

The core of the algorithm will be based on the following equation:

f = g + h

Where g is the distance moved from the start of the (total steps)

Where f is the distance (directly) towards the end.

The idea of this algorithm is that it will continue to path towards the solution, but only if the move is the most efficient possibility. So, while it may run into a wall and attempt to path around it, if it sees another path has surpassed its efficiency it will stop the current search and continue along the other one.

To find my g, I will simply add +1 to the parent point’s current position (which start at 0 from the start).

There will be two key parts to this:

* Point Object
* Maze Solve Algorithm (may be split across many methods)

The rest is simply going to display the information the above will provide

The pseudo code will look as follows:

Point Class {

int x, y, f, g, h;

Point Parent; the point that connected to this one, there may be several paths to the same point—but ultimately the final path can be traced back by continually calling it’s parents

Point (int x, int y); This will be a constructor that is used for endpoint and startpoint as startpoint has no parent, and endpoint will not have a parent until the path is structured

Point (int x, int y, Point parent); Most used constructor. Used to chain points

Getters and setters for valid methods

}

Pseudo Code for algorithm:

ArrayList open; - array list of possible move

ArrayList closed; - array list of successful moves (finished paths)

Add starting position to open array (this is the default place to start looking for paths)

While loops starts, this will make sure open is not empty

Find the most efficient (lowest f) in open list, set it as current point (p)

Remove p from the open list

Generate arraylist of all valid moves stemming from p (not wall or out of bounds)

For each loop based on valid moves (pp, or possible move)

If this node is the end, stop and return the path based on parents of pp

Determine g for pp (parent.g + 1)

Determine h for pp (distance formula with end coordinates)

Determine f for pp (add g + h)

If there is a more efficient f with same coordinates in open skip this point

If there is a more efficient f with same coordinates in closed skip this point

If the above two were false, add this node to open

End for Each

Add p to closed list

End While loop

Throw exception if this area is reached, as this means the maze has no solution