Academic reference

The algorithm we are proposing to implement is the A* search algorithm (https://ieeexplore.ieee.org/abstract/document/4082128)

To help us understand the algorithm's functionality and implementation, we are using the Red Blob Games page on A* as a reference. We also use the Pseudo Code presented in the page as a basis for our implementation of A* search. (https://www.redblobgames.com/pathfinding/a-star/introduction.html)

Algorithm Summary

A* search is a path-finding algorithm that finds the most optimal path from a start node to and an end node in a graph. We can define optimality in different ways such as least distance or least time taken.

Essentially A* seeks to minimize $f(n) = g(n) + h(n)$, where $g(n)$ is the smallest cost from the node n to the start node and $h(n)$, heuristic function, is the smallest cost from the current node n to the end node.

In our implementation, we seek to find the most optimal path with respect to distance. For minimizing distance, there are several heuristic approaches like manhattan distance and diagonal distance, but we will be using euclidean distance.

Function I/O

```
void converttogrid(std::string infile);
```

```
@param infile -- the absolute path to the preprocessed text file
@return --  Does not return anything, but creates a 2 dimensional grid in the class
```

```
Template <typename Heuristic>
vector<pair<int, int>> aStar(pair<int, int> start, pair<int,int> target, vector<vector<int>> grid, Heuristic h);
```

```
@param start -- the coordinates of the start point in the grid
@param target -- the coordinates of the end point in the grid
@param grid -- the two dimension processed grid from the text file
@param h -- the heuristic function used to find the cost to compare
@return --  the coordinates of the points that form the most optimal path
```

```
Template <typename Heuristic>
vector<pair<int, int>> getNeighbor(pair<int, int> curr, vector<vector<int>> grid);
```

```
@param curr -- the coordinates of the current point in the grid
@param grid -- the two dimension processed grid from the text file
@return -- the neighbor point of the current point that have not been processed yet
```

Data Description

For creating our testing dataset, we chose to write codes to generate multiple random grids and use BFS to verify if a path from start to end exists and find the shortest path for each grid. If a path exists, the grid generated can be used as the testing data. The grid generating algorithm was written in c++. We also generated multiple random grids that do not have the shortest path between source and destination to make sure the algorithm can handle both success and failure cases. You can find the algorithm used in the data folder. As for the test data, we divided the test data into small (5X5 to 10X10), medium(10X10 to 50X50), and large(50X50 to 200X200), and generated multiple test data for the same width and height. The name of the file can imply the information of the test grid. The file name is “test_grid_widthn_heightm(v)” where n is width, m is height, and v is different versions. In the test data text file, the first line is the width and height of the grid followed by the minimum distance between source and destination. The grid starts from the following line.