**Technical Document**

**Demonstrating Pathfinding Using A\* and Greedy Approaches**

**\_\_\_\_\_ Something Something Alpha Beta Pruning**

**March 14, 2018**

**Authors: Garett MacGowan, Heni Virág, Sean Tippett**

**Pathfinding Problem**

Pathfinding is a very useful tool in computing. Generally, pathfinding is used to find the shortest path between a start node and a goal node on a graph. This paper will discuss the implementation of two different pathfinding methods, a Greedy approach, and an approach using A\*.

**Solutions Using Greedy and A\* Approaches**

It is important to note that the Greedy approach is complete but not optimal. The Greedy approach can be tricked into following very costly paths since it only considers the estimated number of steps to the goal, not the actual cost of the path it is considering. On the other hand, A\* is complete and optimal. A\* optimality assumes the use of an admissible heuristic, which is the case in this example.

**Considerations**

Although not utilized in this example, the execution speed of the A\* algorithm can be improved by relaxing the admissibility of the heuristic. This is done by essentially adding to the number generated by the heuristic function by some small margin, which makes the number of node expansions required to get around obstacles far fewer. This is essentially because the heuristic is given more influence relative to the cost of a traversal. However, since the heuristic is no longer admissible, it is not guaranteed to find an optimal solution (though it may find a solution that is very close to optimal).

**Solution (Greedy and A\*)**

***main():***Called By  
Program start  
Calls  
gridRead(), solutionHandler(), writeSolutions()  
DescriptionThis function handles reading input boards from gridRead() which it then passes to solutionHandler() for the paths to be determined. Once the paths are known, the board representation is passed to writeSolutions() where the solved boards are drawn into files “pathfinding\_a\_out.txt” and “pathfinding\_b\_out.txt”. Each file contains the solutions to the input problems where 4 and 8 directions of movement respectively are permitted.

***gridRead():***Called By  
main()  
Calls  
None  
Description  
 This function reads lines from files “pathfinding\_a.txt” and “pathfinding\_b.txt” located in the working directory of the script. The files should contain the problems represented as rectangular grids where the grid elements are made of either ‘\_’, ‘X’, ‘S’, or ‘G’ characters. The ‘\_’ character represents an open space, the ‘X’ character represents an obstacle, the ‘S’ character represents a starting point, and the ‘G’ character represents the goal. For each file, the problem grids present are loaded into two three dimensional lists, pathProblemsA and pathProblemsB, with the start and goal nodes marked in two other lists, pathStartEndA and pathStartEndB. All four lists are returned to be used in the solutionHandler() function.

***retrieveNeighbors():***Called By  
greedy(), aStar()  
Calls  
None  
Description  
 This function, depending on the problem type, builds and returns the list of the neighbors of a node on the grid. If the problem is of type “A”, the function checks for neighboring nodes that are in positions that are either vertical or horizontal relative to the node. If the problem is of type “B”, the function checks for neighboring nodes that are in all positions surrounding the node. So long as the neighboring nodes do not contain “X”, they are added to the list of neighbors. This list of neighbors represents the possible positions for the next node in the path.

***greedy():***Called By  
solutionHandler()  
Calls  
retrieveNeighbors(), mannHeuristic(), chebHeuristic()  
Description  
 This function implements a Greedy approach to pathfinding. It works to generate the path by estimating the number of steps required to reach the goal node for each path option within one step of the current node. This is done using a priority queue and a heuristic function, as well as a hash map to help rebuild the path once it’s found. For each proceeding option on the current path, the node that is estimated to have the least number of steps to the goal is selected to be part of the path. The priority queue ensures that the approach is always expanding its most promising path, and the process continues until the goal node is found. Once the goal node is found, the path is added to a list by retracing the hash map “cameFrom” to the starting node. Once the list of nodes on the path is built, the result is returned.

***aStar():***Called By  
solutionHandler()  
Calls  
retrieveNeighbors(), mannHeuristic(), chebHeuristic()  
Description  
A\* works to generate the path by estimating the number of steps required to reach the goal node for each path option within one step of the current node, while also taking into consideration the cumulative cost of the path being considered. This is accomplished using a priority queue, a heuristic function, and a hash map which keeps track of path costs from visited nodes. Just like in the Greedy approach, another hash map is used to help rebuild the path once it is found. For each proceeding option on the current path, a cost is determined by taking the sum of the heuristic estimate and the cost of the path so far. The node with the lowest cost is selected to be part of the most viable path. The priority queue ensures that the most promising path is always being expanded, and the process continues until the goal node is found. Once the goal node is found, the path is added to a list by retracing the hash map “cameFrom” to the starting node. Once the list of nodes on the path is built, the result is returned.

***mannHeuristic():***Called By  
greedy(), aStar()  
Calls  
None  
Description  
 This function calculates “Manhattan distance”, the distance defined by the sum of the difference between two point’s x coordinates and the difference between the two point’s y coordinates.

***chebHeuristic():***Called By  
greedy(), aStar()  
Calls  
None  
Description  
 This function calculates “Chebyshev distance”, the distance defined by the greater distance of either the x coordinate distance or the y coordinate distance between two points. This heuristic is used because it is a well-informed heuristic while also being admissible. Euclidian distance is not admissible in a grid scenario allowing 8 directions of movement. This is because the hypotenuse can overestimate the number of steps needed to reach the goal when the goal is immediately adjacent on a diagonal.

***drawSolutionPath():***Called By  
solutionHandler()  
Calls  
None  
Description  
 This function occupies grid positions with a “P” where the positions are part of the solution path. It then returns the grid list.

***solutionHandler():***Called By  
main()  
Calls  
greedy(), aStar(), drawSolutionPath()  
Description  
 This function handles generating both a Greedy solution and an A\* solution for each grid that is passed to it. It stores the results in a two-dimensional list to make the file writing process cleaner. Each inner list holds the Greedy and A\* solutions to a grid.

***writeSolutions():***Called By  
main()  
Calls  
None  
Description  
This function writes the solved paths to a file in a readable manner. Solutions are drawn in the same order they were read from the input files with tags added to show which algorithm was responsible for which solution. Spaces are used to indicate when a new problem is being shown.

***visualizer():***Called By  
Optional  
Calls  
None  
Description  
Prints a grid to the screen. Used for debugging.