**Final Project – Pathfinding Algorithms**

*Michael Graziano & Miguel Mark*

**I. PROJECT OVERVIEW:**

*i.**Problem Description:*

In many video games, computer controlled game characters are required to traverse complex terrain for many reasons such as when a player is requesting their virtual army to go attack the enemy, or when a player is being escorted by a helpful character to a secret path to the next goal. To do this, game designers use pathfinding algorithms to find the shortest path from a starting location to a target location. These pathfinding algorithms include algorithms we have studied in class such as Dijkstra’s, but also other algorithms such as Greedy Best-First Search and A\*. The question then becomes which one does this shortest path calculation most efficiently so as to not have any noticeable delays for the player?

In this project, we set out to study the performance of these three algorithms by having each of them solve a randomly generated N by N maze. This N by N maze, generated by an online web service[[1]](#footnote-2) and saved as a .svg file, acts as our input for the problem. The .svg file is first decomposed into two representations: a matrix representation of the .svg maze and an adjacency list representation of the path through the maze. Once this is done, the time required to solve the maze is recorded for all three mazes and reported for graphing. Finally, we save the paths of these algorithms as modified .svg files to verify the correct solution.

*ii. Design Specifics:*

a. Maze Representation

b. Path Finding Algorithms

Dijkstra’s Algorithm, the Greedy Best-First Search and the A\* Algorithm are all implemented using a binary min-heap; specifically we used the structure that was provided to us for our Dijkstra implementation in Homework 5. Although this particular priority queue implementation does not provide us with the best overall performance (the Fibonacci heap would provide us with better time complexity), its implementation is much simpler for the time constraints of the project. The purpose of the heap is to allow us to easily sort our nodes in increasing order, starting with the node with the smallest or “best” priority in the queue. When we studied Dijkstra’s Algorithm, the “best” priority would be the node closest to the start node. Greedy Best-First Search and A\* Algorithm change this priority value to provide better performance.

Greedy Best-First Search is similar in operation to Breadth First Search and Depth First Search but unlike these blind exploration algorithms Greedy Best-First Search tries to make its decision based on the following criteria: the next node to be visited should be closer to the end node.[[2]](#footnote-3)2 In order to determine what the distance of a node was from the end node, we assigned the sum of the absolute values of delta x & delta y of the node in the maze relative to the end node. This approximation is called the “Manhattan Distance” and is applicable to our problem because we only allow orthogonal movement through the maze.[[3]](#footnote-4)3 This distance is used in place of the growing path length that is used in Dijkstra’s algorithm to determine priority in the queue. Although this algorithm helps to force exploration toward the end node and therefore reach it sooner, it does not take into account the weight of arcs, which in a video game could represent the difficulty of the obstacles along that particular path.

**II. CODE INSTRUCTIONS:**

**III. SUPPORTING INFORMATION**

**IV. WORK BREAKDOWN**

|  |  |
| --- | --- |
| * Maze cell to graph node identification * Pathfinding algorithm implementation * Documentation   X  *Michael Graziano* | * Interpretation of .svg file into maze matrix * Adjacency list interpretation of maze   X  *Miguel Mark* |

1. http://www.mazegenerator.net/ [↑](#footnote-ref-2)
2. https://www.geeksforgeeks.org/best-first-search-informed-search/ [↑](#footnote-ref-3)
3. https://www.geeksforgeeks.org/a-search-algorithm/ [↑](#footnote-ref-4)