Authored by: George Engel, Justin Keeling, & Cory Johns

**Report Checklist**

**GOAL:** Get them to understand our code with out having to read it

-interesting aspects of solution should be the focus (any non-obvious stuff) (NO CODE)

-Language: Python

-Parameter settings: ???

-why they where chosen: ???

Generic solution explanation:

Explanation of why we need solution X: (Node list for algorithms to take in?)

Pseudo code for solution X:

-Explain Sudo Code for solution X.

What we considered & why we did/didn’t use it:

-Node trees

-Node list

-generic priority queues

## Overview:

This project was written in python since it is easy to use and allowed us to focus more on the concepts of the project rather than the implementation. The assignment was broken up into parts with each author contributing approximately an equal amount to the project.

Cory covered the MazeNode class, reading in the maze text files, making the node tree, the top-level search function, and printing the results. Justin handled the Depth First and Breadth First search algorithms. George handled the Greedy and A\* search algorithms. All of us extensively planned and communicated each step of the project to each other.

# Main Parts:

Read in maze from text file into 2D array

Before the maze could be searched it first had to be converted into a form that would make the search algorithms easy to make. This was accomplished by first reading in the maze from the text file and directly putting the characters into a 2D array such that each element in the array contained only one character.

Convert 2D array to node tree

To facilitate the search algorithms, the 2D array was converted into a node tree where each node represented a state in the search. Each node contained a list of all adjacent nodes that were traversable and flags to indicate if the node had been visited already.

This node tree was made in the following way:

* Starting from the 2D array of the last step
* Start at element 0, 0
* Make a new node at current location if the maze square is not a wall and the location does not already have a node.
* Then add and connect nodes at (row+1, col) and (row, col+1) if the location is in the maze and the square is not a wall or a node has not been made there already
* Repeat this process at (row, col+1) and wrap to (row+1, 0) if the end of the row has been reached
* This process allows us to hit all the nodes without checking the same node twice, and ensures that the new nodes being made have not been made previously
* Finally, the program will return the node that contains the start point, this will act as the root of the tree and will consequently lose all islands not reachable from the root, as these where not useful anyway.

Run each search algorithm with each maze type

Each search algorithm was then ran from the top level search function that called a parameter function that took the root node as an input. The function then returned a list of the visited nodes which was then printed to the output file.

Depth first search

Breadth first search

Greedy search

A\* search