University of Puerto Rico
Mayagüez Campus
Mayagüez, Puerto Rico

Department of Electrical and Computer Engineering

Search Algorithms

by:

Manuel Alejandro Umaña Rodriguez
Dahyna Gabrielle Martínez Pérez
Jan Luis Pérez de Jesús
Christopher Hans Mayens Matías

For: Profesor José Fernando Vega Riveros
Course: ICOM 5015, Section 001D
Date: March 19,2025

Abstract:

How is the performance of a search agent measured? Throughout the following report the objective is to establish how an environment affects the efficiency of different search algorithms. To determine this two different environments were studied and discussed. The first environment was a plane with differently shaped obstacles with a start and goal coordinates. The objective of the search agent was to traverse from start to goal utilizing the shortest path considering the obstacles in the way. For this particular environment two search algorithms were used: Greedy Best-First Search and A* Search. These two algorithms utilized heuristic functions to traverse the plane, as these algorithms are both informed search they utilized the estimated cost from the current position to the goal and take paths that minimize cost. The results showed that A* Search is a superior algorithm for this environment, because it considers the cost from the start to the current position, creating a more efficient path to achieve the desired goal. The additional information allows it to make fewer mistakes as it moves across the plane keeping in memory where it's been to better traverse its environment. The second environment studied was the classic Missionaries and Cannibals problem. This problem consists of three missionaries and three cannibals on one side of the river. A boat is stationed on the left side and its objective is to move everyone to the right side but several limitations are in place. The boat can hold a minimum of one passenger and a maximum of two passengers and the number of missionaries can't be outnumbered by the cannibals in one area otherwise the cannibals eat the missionaries in that area. For the environment two search algorithms were used: Breadth First Search and Depth First Search. Between these two algorithms Breadth First Search fared better in comparison. The reason for this is that Depth First Search traverses all the reachable vertices of an adjacent vertice. This means that it continuously visits each vertex and may even visit an invalid state. When this happens the algorithm needs to loop a few steps back to return to a valid state which consumes performance. Breadth First Search was able to find the most optimal and quickest path due to the fact that it searched through layers and not the entire depth of the branch. Having an algorithm that constantly goes through unnecessary paths in order to cover more ground could be useful in other cases but for this particular one it created more room for error and ultimately affected the performance of the DFS.

Table of Contents:

Group Collaboration

The tasks amongst each team member were divided by each member's experience utilizing python, as follows:

- Christopher Mayens and Manuel Umaña were in charge of exercise 3.7 and 3.9 respectively within the python code and its segments within the report.

-Jan Pérez and Dahyna Martínez were in charge of the development of the report and presentation. In addition to the creation of the presentation with the help of Christopher Mayens and Manuel Umaña.

-All members of the group were included and participated in the video. Video editing was made by Jan Pérez.

I. Introduction

Artificial Intelligent agents are created to be able to perceive and interact with their environments. For the purpose of the class, rational agents are to be studied and designed. The reason for the implementation of these agents is so they may solve specific problems with the best performance possible. This entails having the agent interact with a problem/ environment that may contain obstacles or constraints.

To be able to select an optimum agent for a specific problem one needs to first be able to completely construct and identify the problem to be solved. This is done by identifying the initial state of the problem, a description of the possible actions of the problem, a description of what each action does (transition model), a goal test, and a path cost. With these parameters defined one can narrow down the search algorithm to reach the goal state. This is to say there can be more than one algorithm that can solve the same problem.

Two experiments were analyzed. The first one consisted of having an agent find the shortest path between two points, on a plane that consisted of convex polygons. This exercise involved a heuristic function that is the estimated cost of the current position to the goal. For this reason an informed search was necessary. The two informed searches that were implemented, analyzed, and compared were the A* Search and the Greedy Best-First Search. Their performances were compared to measure which algorithm was most optimal. Other features that were analyzed in the environment were the size of the state space, the most optimal path, and the number of states and paths towards the goal.

For the second experiment the Missionaries and Cannibals problem was analyzed. This problem consisted of transporting three missionaries and three cannibals to the other size of a river bank. The problem has two constraints: one or two people at maximum can be on the boat, the boat could not be empty traversing the river and there could never be more cannibals than missionaries in any given location. Due to the problem not only having any prior knowledge or heuristic beyond the problem definition, the search algorithms that were used in the search for the most optimum and shortened path were uninformed searches. There were a variety of algorithms that we could have used, but due to the definition of the problem and the constraints involved the algorithms used were the Breadth First Search and the Depth First Search. These algorithm's performances were analyzed and compared to determine the most optimal solution. The design of the environment was also studied as the base code could be improved with the verification of repeated states.

II. Related Work

For the environment of exercises 3.7, it was determined before conducting the simulations to be partially observable, discrete, deterministic, static, single agent, and the agent represents a search agent with two different search algorithms tested. The two algorithms analyzed for this were A* Search algorithm and Greedy Best-First Search.

A* Search algorithm considers each step that it takes according to a value of total estimated cost of reaching the goal which consists of the sum of the movement cost to move from the starting point to a given position and the estimated cost to move from the current position to the final position [1]. The formula that describes this algorithm is:

$$f(n) = g(n) + h(n)$$

where:
$f(n)$ is the value of total estimated cost of reaching the goal
$g(n)$ is the cost from start to the current position
$h(n)$ is the estimated cost from current position to the goal (heuristic function)
n is the current position/node

The Greedy Best-First Search algorithm is an informed search algorithm where the evaluation function is strictly equal to the heuristic function [2]. The formula that describes this algorithm is:

$$f(n) = h(n)$$

where:

$f(n)$ is the value of total estimated cost of reaching the goal
$h(n)$ is the estimated cost from current position to the goal (heuristic function)
n is the current position/node

The performance measure for this exercise was simply described as the amount of steps taken to achieve this goal. The following formula was used to describe this behaviour:

$$Performance = Amount\ of\ steps\ taken\ to\ achieve\ the\ end\ state$$

For the environment of problem 3.9 (Missionary and Cannibal Problem) it is described as fully observable, deterministic, static, sequential, single agent, and discrete. The agent was tested with different uninformed algorithms, those being Breadth First Search and Depth First Search. These were utilized as the problem did not involve any heuristic formula for finding the solution. Other uninformed search algorithms were considered such as, Uniform Cost Depth-Limited Search, Iterative Deepening Search, and Bidirectional BFS. However they were discarded due to the different constraints involved in the definition of the problem making them unattainable or suboptimal in their performance.

The performance measurement for this exercise was the number of steps taken to reach the goal state.

$$Performance = Amount\ of\ steps\ taken\ to\ achieve\ the\ end\ state$$

III. Performance Approach and Results

A. Performance Analysis to Obtain the Shortest Path Between Two Points With Obstacles

The performance of an agent is an essential step to identify if it's capable of solving a problem in the most optimum way. For this reason in particular it's necessary to measure the performance on different scenarios and conditions. Exercise 3.7 established an environment that was partially observable, single agent, static, discrete, deterministic, and sequential.

For this exercise a case where an agent had to traverse from point A to point B was studied. A premade environment was created utilizing a schematic similar to the one in Figure 1 from the book Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach. 3rd Edition) [3]. The base code to work on this exercise was taken from the aima code specifically from the search4e.ipynb file [4]. The functions that described the algorithm for straight line distance and heuristic were used and written in a separate file called geometry.py. The agent in this simulation is utilizing a pathfinding algorithm that searches for a route from the start node to the goal node. The shortest path from one polygon vertex to another consists of straight line segments with some vertices. In a continuous plane the shortest path between two points is a straight line; however this exercise encourages the user to create obstacles to see how the algorithms react to these changes. For these cases the shortest path is defined by the best way to traverse from two points by moving efficiently through the obstacles. These parameters can be chosen to study different scenarios at different locations. If we consider all possible positions for x and y in the plane, the number of states for this exercise is infinite. However the designed code had a grid size of 20 by 20 meaning there were 400 discrete states. A good state for this particular exercise consists of the start position, the goal position and the vertices of the obstacles. Code was implemented to allow the user to click on a coordinate to create an obstacle. For the agent to get from point A to point B it can go up,down, left or right to explore adjacent cells. The agent considers a wall as an obstacle and doesn't go through it. Once a solution is found the program displays the shortest path of the algorithm selected. Three cases were studied to compare the performance between algorithms A* Search and Greedy Best-First Search.

Case 1: The start and end goal utilizing the coordinates of Figure 2. To make the display have better visibility of the obstacles the number of the coordinates was disabled leaving the following cases displaying the environment as Figure 3. This environment had a decent amount of obstacles between start and goal meaning both algorithms would have to traverse the environment differently. For the A* Search algorithm took 38 steps to achieve its goal and for Greedy Best-First Search it took 44 steps. The results of this case were displayed in Figure 4 and 5.

Case 2: The start goal and the end goal were among a straight line across the middle of the plane as seen in Figure 6. This environment was built to test both algorithms in a semi-ideal scenario in which minimal obstacles affected the path between the start and goal. For this particular case both A* Search algorithm and Greedy Best-First Search algorithm took 23 steps to achieve its goal. The results for this case were displayed in Figure 7 and 8.

Case 3: The start was set across the middle of the plane on top of a pyramid and the goal was hidden around a corner of the rhombus in the bottom right corner as displayed in Figure 9. This environment served to test the algorithms in tight corners where making a mistake would add multiple unnecessary steps. For this case A* Search algorithm took 22 steps and the Greedy Best-First algorithm took 24 steps. The results for this case were displayed in Figure 10 and 11.

After studying the cases carefully it was found that the A* Search algorithm was much more efficient at traveling from start to goal. This is due to the fact that A* Search evaluates nodes based on actual cost from the start node to the current node and the heuristic estimated cost from the current node. The sum of these values determine the total estimated cost of reaching the goal. For performance, A* Search picks the option that will result in a smaller sum [1]. Meanwhile the Greedy Best-First Search algorithm only evaluates this function considering the heuristic function, disregarding the edge weights in a weighted graph [2]. This means that it only considers the estimated cost to move from the current position to the goal. The A* Search algorithm method is much more effective because it has more information stored and can correctly prepare for the next step without risk of making critical mistakes.

B.Problem Formulation and Algorithm Performance for AI Algorithms

AI agents are created to be able to perceive and act within an environment with the goal of solving specific problems. Since the purpose of the course is to learn about and to design agents that act rationally, a well established problem and constraints are in order to maximize the goal achievement with the provided information. In addition to this, more than one AI algorithm will be able to solve the same problem. To be able to determine which is the most appropriate algorithm, comparisons of their performance metrics are studied.

To be able to design a problem one needs to establish the initial state of the agent, a description of the agent's possible actions, a description of what each action does (transitional model), a goal test, and a path cost. For the experiment of problem 3.9 from the book Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach. 3rd Edition [3], the provided problem was the Missionaries and Cannibals problem. The base code of the problem that was used to work on this exercise was taken from the aimacode specifically from the search.py file [4].

The problem is described as follows. There are three missionaries (M) and three cannibals (C) on one side of a river. There is a boat on this side of the river that can hold at minimum one person and at maximum two people. The agent is required to get everyone to the right side of the river without leaving a group of missionaries in one place outnumbered by the cannibals in that place. From this description the environment is described as fully observable, deterministic, static, sequential, single agent, and discrete.

In the search.py file a class called MissionariesCannibals inherits from Problem. With this the initial states of all three cannibals, three missionaries, boat position, and riverside populations are established as (3,3,1,0,0) where 3-missionaries in left bank ,3-cannibals in left bank , 1 boat on the left bank, 0 missionaries in right bank, and 0 cannibals in right bank. All possible and valid actions of the boat movement are calculated. A valid condition is defined as never having a missionary be outnumbered by the cannibals no matter if they are on the left side of the river, the right side of the river or within the boat and the boat can carry at most two people. The state space for all the valid moves is shown in Figure 12. The path cost was designed to be the number of steps it took the search algorithm to reach the goal state. The goal state was defined as having all the missionaries and cannibals on the right side of the river. If an algorithm was unable to solve the problem, a step limit was created.

Since there was no provided information to estimate the cost to reach a state and the algorithm would have to search the entire space systematically uninformed searches were utilized. Only Breadth First Search and Depth First Search were utilized and compared; this was due to the characteristics of the other uniformed searches [5] as explained below.

- Uniform Cost: Not complete since all path costs in the base code are the same values.
- Depth-Limited Search: Choosing a practical depth limit is complicated due to limited information and constraints.
- Iterative Deepening Search: Repeated expansions of nodes to increase depth limit to find solution, not shortest path.
- Bidirectional BFS: Complicated to define the reverse search due to limited information and constraints.

Initially the Breadth First Search (BFS) algorithm was used. From the actions function it checks if a certain action is valid. The nodes of the graphs were the number of missionaries and cannibals on each side of the river and the boat position. The edges of the graph were the valid moves. Every action that the algorithm took was saved in a tuple. These decisions were used to create a GUI. In this GUI one could observe the amounts of cannibals, missionaries, the boat's direction, the action taken, the amount of steps taken. The BFS solved the problem in 11 steps.

The BFS algorithm was switched with the Depth First Search (DFS) algorithm. For this case the nodes of the graphs were the number of missionaries and cannibals on each side of the river and the boat position. The edges of the graph were the valid moves.This algorithm tried to solve the problem by going as deep as it could to one branch before backtracking. This experiment was executed using two different methods. The first involved an unlimited amount of steps to execute and the second had a limited amount of steps.

When the algorithm had no step limit it stated that the problem was unsolvable, yet when provided a limit the algorithm could solve the problem. It is to be noted that the minimum limit to solve the problem was 11 steps. When the limit was raised the algorithm would loop between steps and reach the goal in the established limit amount.

The two algorithms were able to solve the missionary and cannibal problem. The BFS fared better than the DFS. This was due to the fact that DFS searches for the traverses all the reachable vertices of an adjacent vertex [6]. The algorithm can try to visit an invalid state and loop a few steps until it can find a valid state or it may get infidelity stuck similar to when no step limit was established. Second, since it only traverses through a whole branch first, it will not find the solution by the shortest path as seen when tested with different step limits. The BFS was able to find the most optimal and quickest solution due to the fact that it searched through the layers instead of the entire depth of the branch; it can find the shortest path first [5]. It is preferred to have been able to have a verification of repeated states because in the event that a valid move repeats itself the program does not loop back to a previous state preventing it from finding the goal state.

Given the simple environment of the problem, it still requires a lot of effort and time for a person to solve it. In Saul Amarel's work, he discussed the intricacies of this problem, highlighting how different representations can impact the ease of finding a solution. He noted that the problem's complexity is influenced by the way states and moves are conceptualized, which affects the solver's ability to navigate the numerous possible actions efficiently [7]. Another parameter that impacts a person's availability to solve the problem is the constraints of the problem having the boat only able to carry two people and never having more cannibals than people in any location. This entails the person to try to consider every possible valid combination that results in a permutation of 5 elements, of the initial state tuple, which is a 120 possible results for combination of actions with only a fraction of those being valid moves given the constraints already mentioned.

C. Figures and Tables

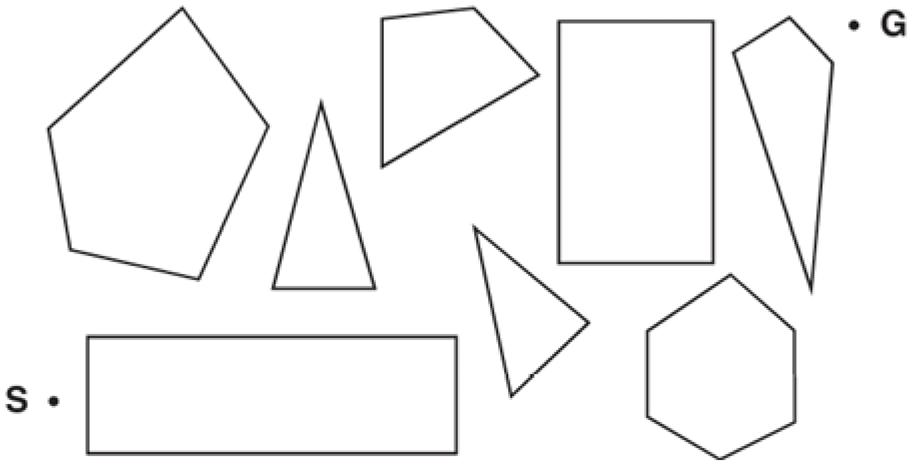
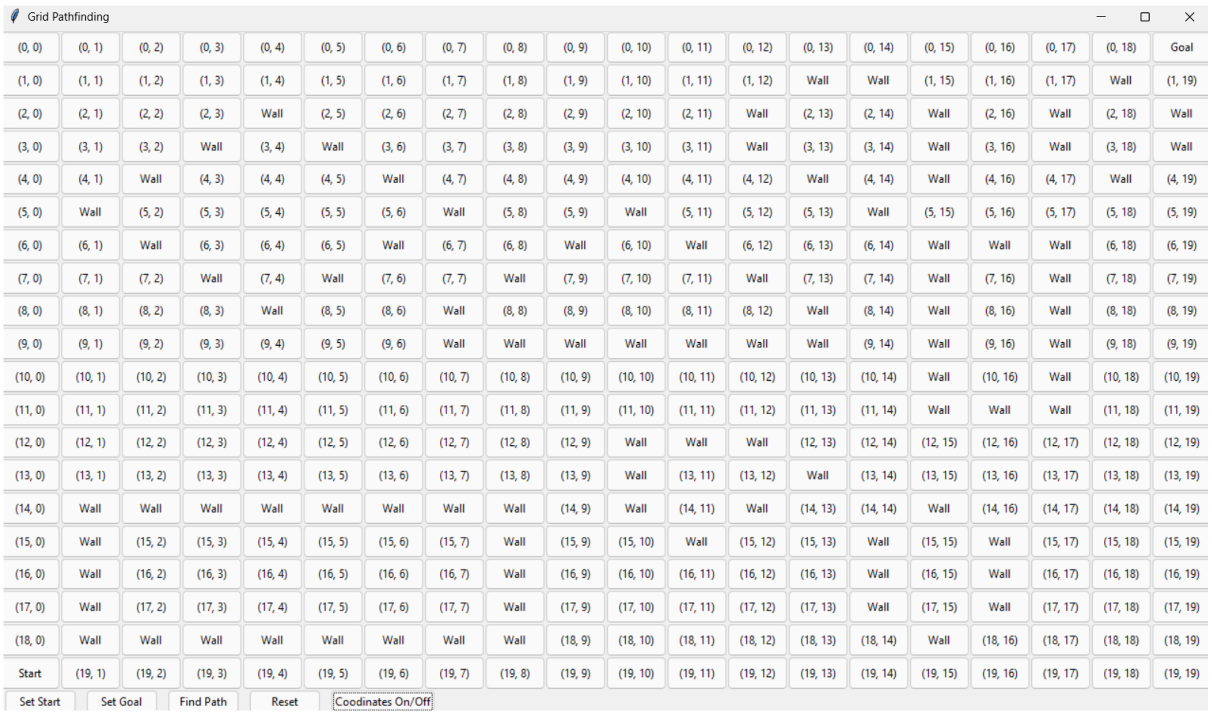
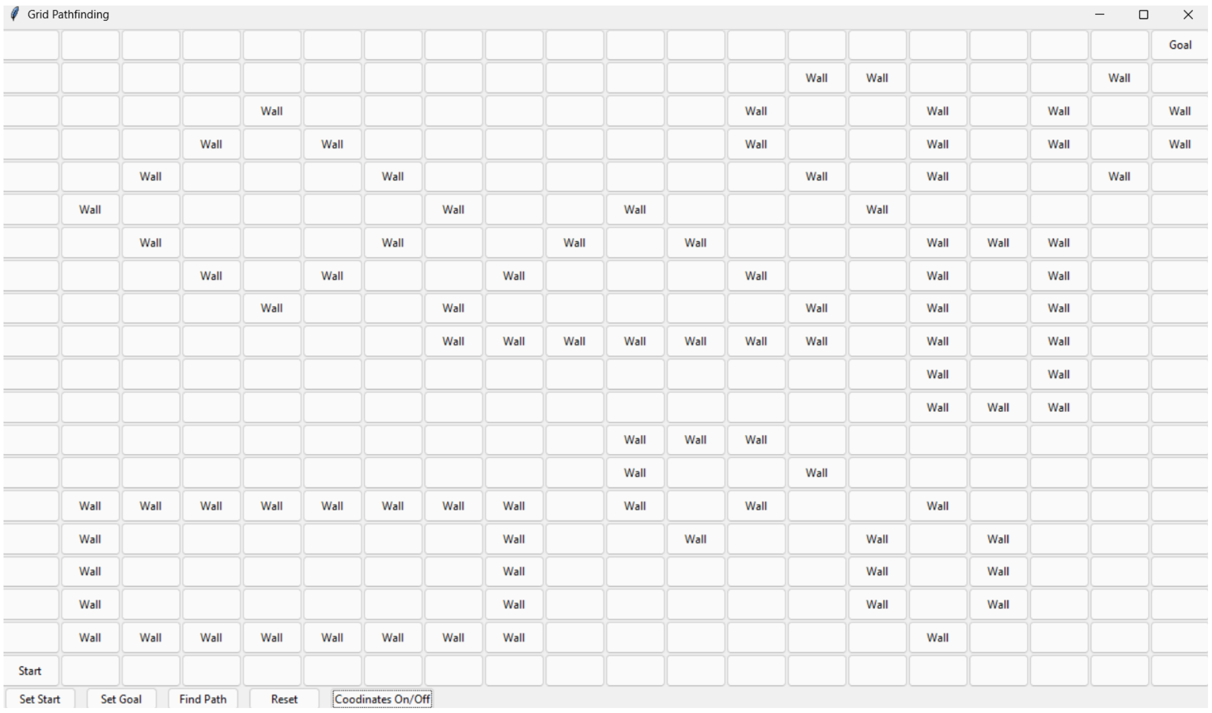


Figure 1. Figure 3.13 from “A scene with polygonal obstacles. S and G are the start and goal states”.



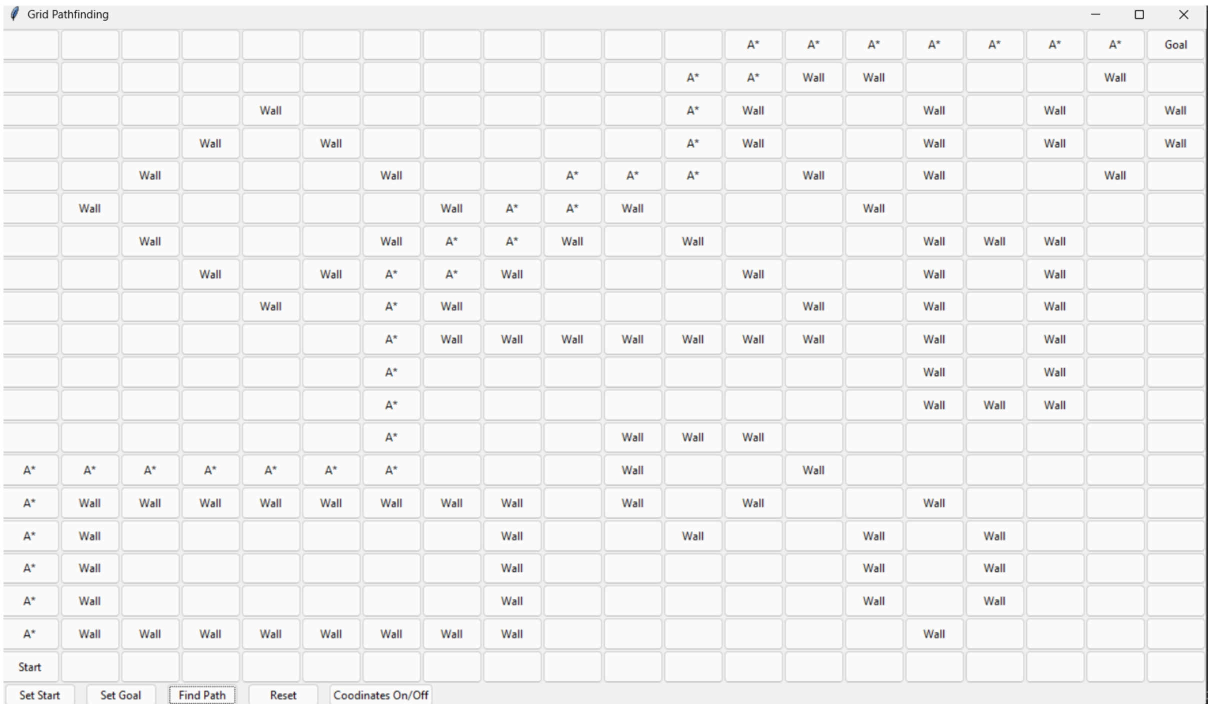
(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)	(0, 7)	(0, 8)	(0, 9)	(0, 10)	(0, 11)	(0, 12)	(0, 13)	(0, 14)	(0, 15)	(0, 16)	(0, 17)	(0, 18)	Goal
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)	(1, 7)	(1, 8)	(1, 9)	(1, 10)	(1, 11)	(1, 12)	Wall	Wall	(1, 15)	(1, 16)	(1, 17)	Wall	(1, 19)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	Wall	(2, 5)	(2, 6)	(2, 7)	(2, 8)	(2, 9)	(2, 10)	(2, 11)	Wall	(2, 13)	(2, 14)	Wall	(2, 16)	Wall	(2, 18)	Wall
(3, 0)	(3, 1)	(3, 2)	Wall	(3, 4)	Wall	(3, 6)	(3, 7)	(3, 8)	(3, 9)	(3, 10)	(3, 11)	Wall	(3, 13)	(3, 14)	Wall	(3, 16)	Wall	(3, 18)	Wall
(4, 0)	(4, 1)	Wall	(4, 3)	(4, 4)	(4, 5)	Wall	(4, 7)	(4, 8)	(4, 9)	(4, 10)	(4, 11)	(4, 12)	Wall	(4, 14)	Wall	(4, 16)	(4, 17)	Wall	(4, 19)
(5, 0)	Wall	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)	Wall	(5, 8)	(5, 9)	Wall	(5, 11)	(5, 12)	(5, 13)	Wall	(5, 15)	(5, 16)	(5, 17)	(5, 18)	(5, 19)
(6, 0)	(6, 1)	Wall	(6, 3)	(6, 4)	(6, 5)	Wall	(6, 7)	(6, 8)	Wall	(6, 10)	Wall	(6, 12)	(6, 13)	(6, 14)	Wall	Wall	Wall	(6, 18)	(6, 19)
(7, 0)	(7, 1)	(7, 2)	Wall	(7, 4)	Wall	(7, 6)	(7, 7)	Wall	(7, 9)	(7, 10)	(7, 11)	Wall	(7, 13)	(7, 14)	Wall	(7, 16)	Wall	(7, 18)	(7, 19)
(8, 0)	(8, 1)	(8, 2)	(8, 3)	Wall	(8, 5)	(8, 6)	Wall	(8, 8)	(8, 9)	(8, 10)	(8, 11)	(8, 12)	Wall	(8, 14)	Wall	(8, 16)	Wall	(8, 18)	(8, 19)
(9, 0)	(9, 1)	(9, 2)	(9, 3)	(9, 4)	(9, 5)	(9, 6)	Wall	Wall	Wall	Wall	Wall	Wall	Wall	(9, 14)	Wall	(9, 16)	Wall	(9, 18)	(9, 19)
(10, 0)	(10, 1)	(10, 2)	(10, 3)	(10, 4)	(10, 5)	(10, 6)	(10, 7)	(10, 8)	(10, 9)	(10, 10)	(10, 11)	(10, 12)	(10, 13)	(10, 14)	Wall	(10, 16)	Wall	(10, 18)	(10, 19)
(11, 0)	(11, 1)	(11, 2)	(11, 3)	(11, 4)	(11, 5)	(11, 6)	(11, 7)	(11, 8)	(11, 9)	(11, 10)	(11, 11)	(11, 12)	(11, 13)	(11, 14)	Wall	Wall	Wall	(11, 18)	(11, 19)
(12, 0)	(12, 1)	(12, 2)	(12, 3)	(12, 4)	(12, 5)	(12, 6)	(12, 7)	(12, 8)	(12, 9)	Wall	Wall	Wall	(12, 13)	(12, 14)	(12, 15)	(12, 16)	(12, 17)	(12, 18)	(12, 19)
(13, 0)	(13, 1)	(13, 2)	(13, 3)	(13, 4)	(13, 5)	(13, 6)	(13, 7)	(13, 8)	(13, 9)	Wall	(13, 11)	(13, 12)	Wall	(13, 14)	(13, 15)	(13, 16)	(13, 17)	(13, 18)	(13, 19)
(14, 0)	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	(14, 9)	Wall	(14, 11)	Wall	(14, 13)	(14, 14)	Wall	(14, 16)	(14, 17)	(14, 18)	(14, 19)
(15, 0)	Wall	(15, 2)	(15, 3)	(15, 4)	(15, 5)	(15, 6)	(15, 7)	Wall	(15, 9)	(15, 10)	Wall	(15, 12)	(15, 13)	Wall	(15, 15)	Wall	(15, 17)	(15, 18)	(15, 19)
(16, 0)	Wall	(16, 2)	(16, 3)	(16, 4)	(16, 5)	(16, 6)	(16, 7)	Wall	(16, 9)	(16, 10)	(16, 11)	(16, 12)	(16, 13)	Wall	(16, 15)	Wall	(16, 17)	(16, 18)	(16, 19)
(17, 0)	Wall	(17, 2)	(17, 3)	(17, 4)	(17, 5)	(17, 6)	(17, 7)	Wall	(17, 9)	(17, 10)	(17, 11)	(17, 12)	(17, 13)	Wall	(17, 15)	Wall	(17, 17)	(17, 18)	(17, 19)
(18, 0)	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	(18, 9)	(18, 10)	(18, 11)	(18, 12)	(18, 13)	(18, 14)	Wall	(18, 16)	(18, 17)	(18, 18)	(18, 19)
Start	(19, 1)	(19, 2)	(19, 3)	(19, 4)	(19, 5)	(19, 6)	(19, 7)	(19, 8)	(19, 9)	(19, 10)	(19, 11)	(19, 12)	(19, 13)	(19, 14)	(19, 15)	(19, 16)	(19, 17)	(19, 18)	(19, 19)

Figure 2. Coordinates recreating the environment in Figure 1



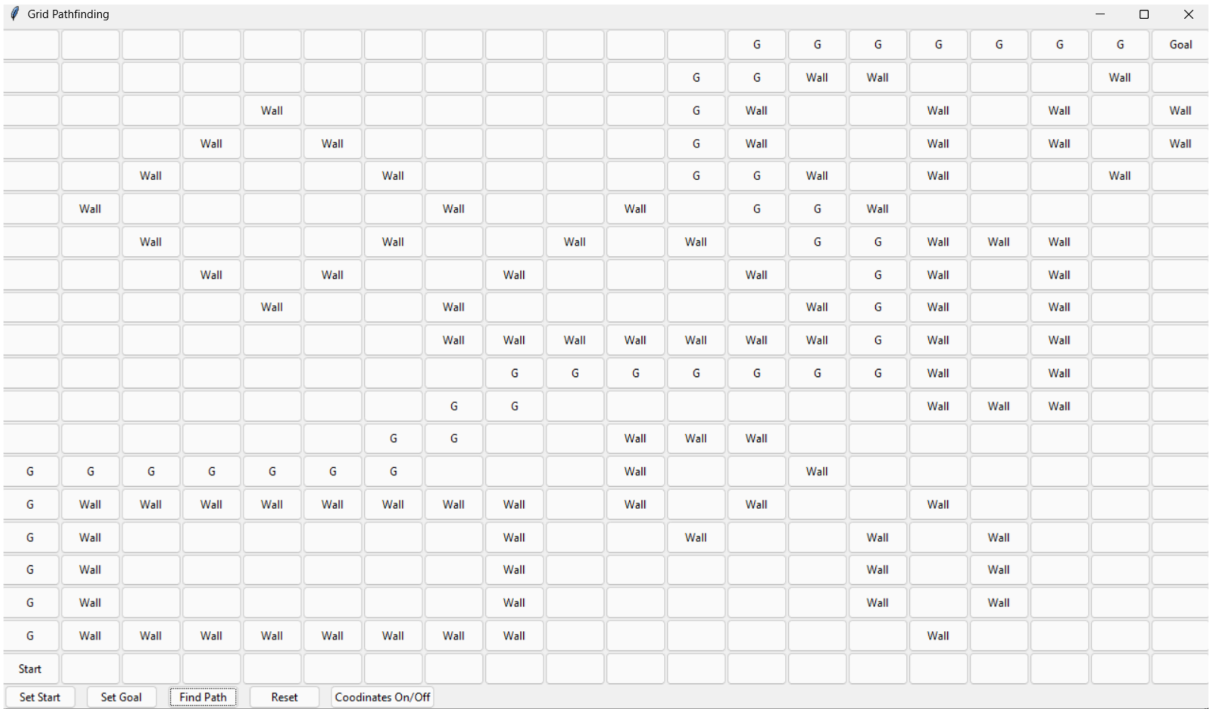
																			Goal
													Wall	Wall				Wall	
				Wall								Wall			Wall		Wall		Wall
			Wall		Wall							Wall			Wall		Wall		Wall
		Wall				Wall							Wall		Wall			Wall	
	Wall						Wall			Wall				Wall					
		Wall				Wall			Wall		Wall				Wall	Wall	Wall		
			Wall		Wall			Wall				Wall			Wall		Wall		
				Wall			Wall						Wall		Wall		Wall		
							Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall		
															Wall		Wall		
															Wall	Wall	Wall		
										Wall	Wall	Wall							
										Wall			Wall						
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall			Wall				
	Wall							Wall			Wall			Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall							Wall				
Start																			

Figure 3. Case 1, coordinates recreating the environment in Figure 1 without numbers



												A*	A*	A*	A*	A*	A*	A*	Goal
											A*	A*	Wall	Wall				Wall	
				Wall							A*	Wall			Wall		Wall		Wall
			Wall		Wall						A*	Wall			Wall		Wall		Wall
		Wall				Wall			A*	A*	A*		Wall		Wall			Wall	
	Wall						Wall	A*	A*	Wall				Wall					
		Wall				Wall	A*	A*	Wall		Wall				Wall	Wall	Wall		
			Wall		Wall	A*	A*	Wall				Wall			Wall		Wall		
				Wall		A*	Wall						Wall		Wall		Wall		
						A*	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall		
						A*									Wall		Wall		
						A*									Wall	Wall	Wall		
						A*				Wall	Wall	Wall							
A*	A*	A*	A*	A*	A*	A*				Wall			Wall						
A*	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall			Wall				
A*	Wall							Wall			Wall			Wall		Wall			
A*	Wall							Wall						Wall		Wall			
A*	Wall							Wall						Wall		Wall			
A*	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall							Wall				
Start																			

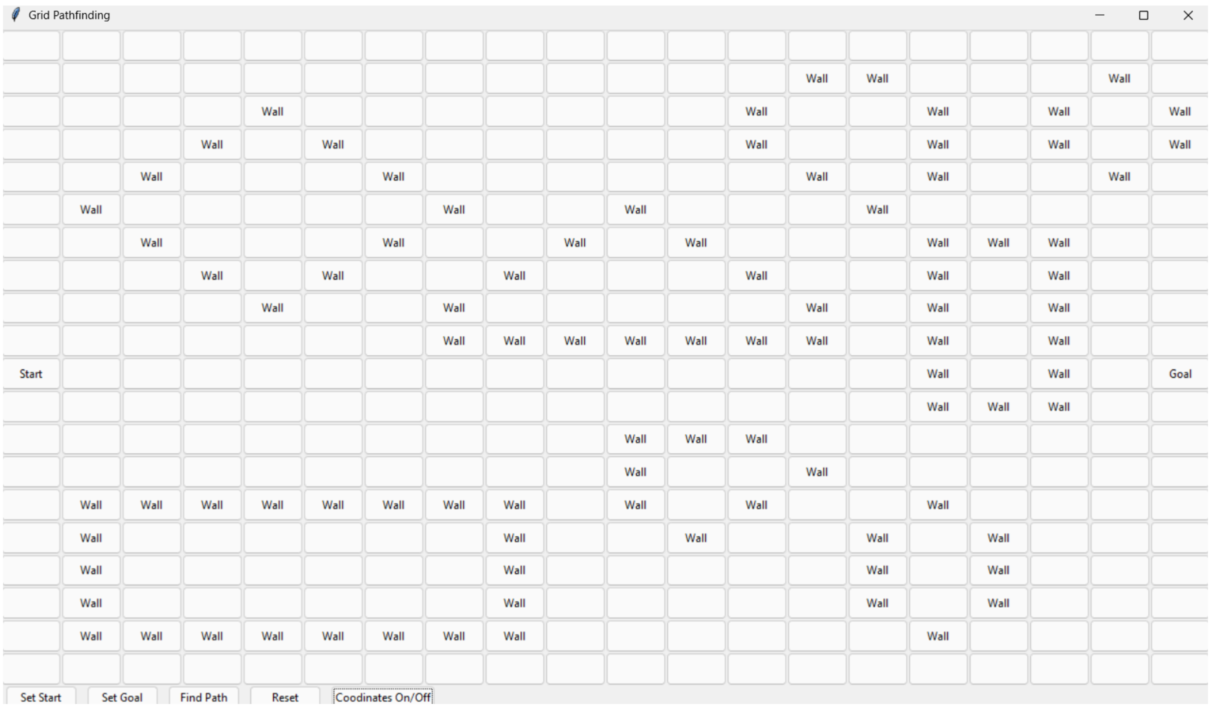
Figure 4. Solution of shortest path between start and goal utilizing A* search algorithm for case 1


Grid Pathfinding

												G	G	G	G	G	G	G	Goal
											G	G	Wall	Wall				Wall	
				Wall							G	Wall			Wall		Wall		Wall
			Wall		Wall						G	Wall			Wall		Wall		Wall
		Wall				Wall					G	G	Wall		Wall			Wall	
	Wall						Wall			Wall		G	G	Wall					
		Wall				Wall			Wall		Wall		G	G	Wall	Wall	Wall		
			Wall		Wall			Wall				Wall		G	Wall		Wall		
				Wall			Wall						Wall	G	Wall		Wall		
							Wall	Wall	Wall	Wall	Wall	Wall	Wall	G	Wall		Wall		
								G	G	G	G	G	G	G	Wall		Wall		
							G	G							Wall	Wall	Wall		
						G	G			Wall	Wall	Wall							
G	G	G	G	G	G	G				Wall			Wall						
G	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall			Wall				
G	Wall							Wall			Wall			Wall		Wall			
G	Wall							Wall						Wall		Wall			
G	Wall							Wall						Wall		Wall			
G	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall							Wall				
Start																			

Set Start | Set Goal | Find Path | Reset | Coodinates On/Off

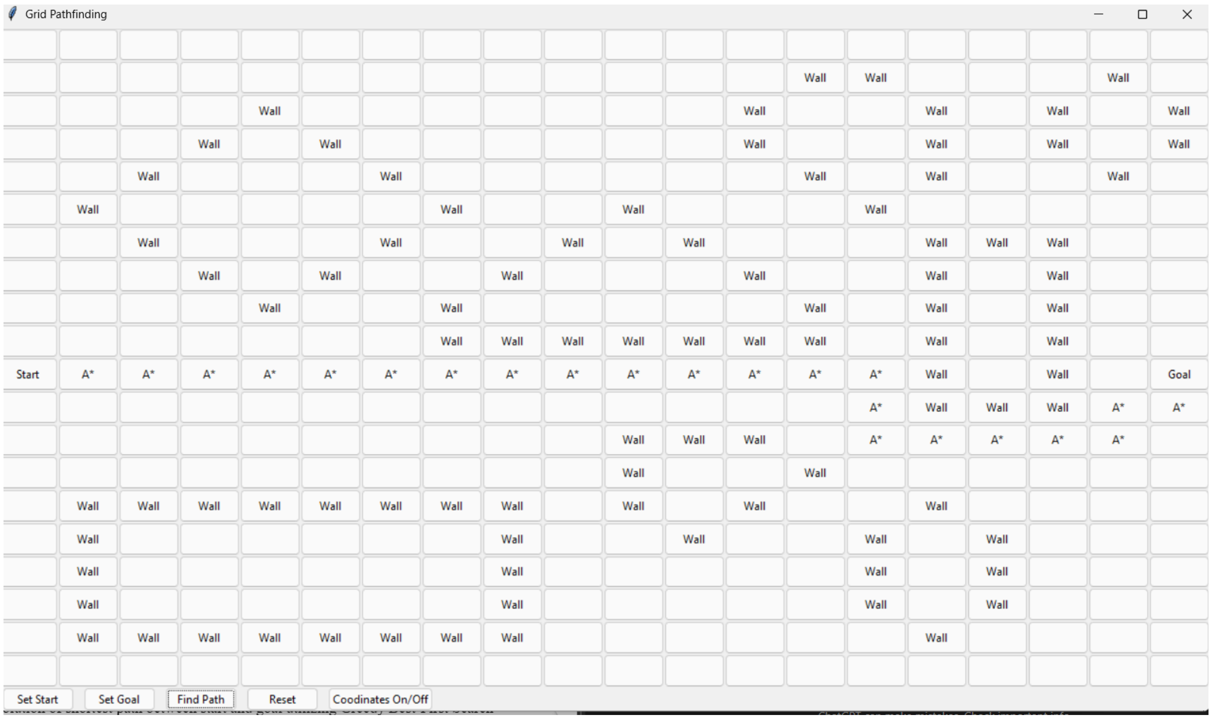
Figure 5. Solution of shortest path between start and goal utilizing Greedy Best-First Search algorithm for case 1


Grid Pathfinding

													Wall	Wall				Wall	
				Wall								Wall			Wall		Wall		Wall
			Wall		Wall							Wall			Wall		Wall		Wall
		Wall				Wall							Wall		Wall			Wall	
	Wall						Wall			Wall				Wall					
		Wall				Wall			Wall		Wall				Wall	Wall	Wall		
			Wall		Wall			Wall				Wall			Wall		Wall		
				Wall			Wall						Wall		Wall		Wall		
							Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall		
Start															Wall		Wall		Goal
															Wall	Wall	Wall		
										Wall	Wall	Wall							
										Wall			Wall						
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall			Wall				
	Wall							Wall			Wall			Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall							Wall				

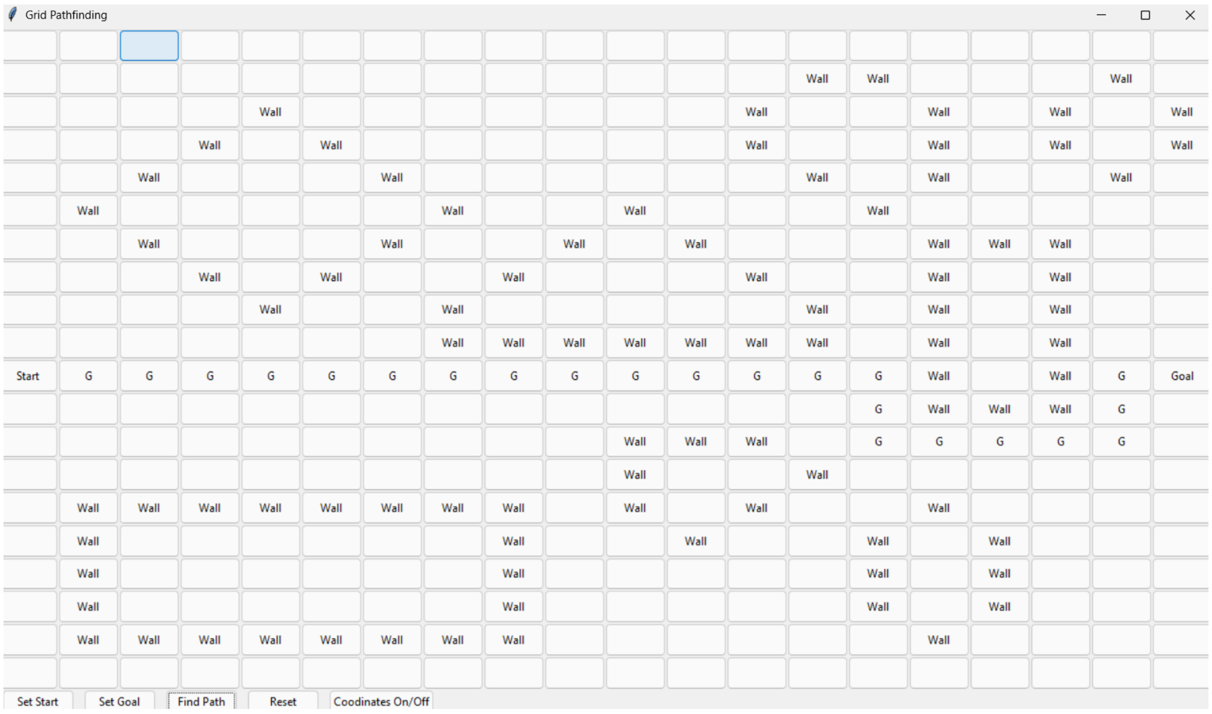
Set Start | Set Goal | Find Path | Reset | Coodinates On/Off

Figure 6. Case 2, coordinates recreating a somewhat ideal environment without numbers



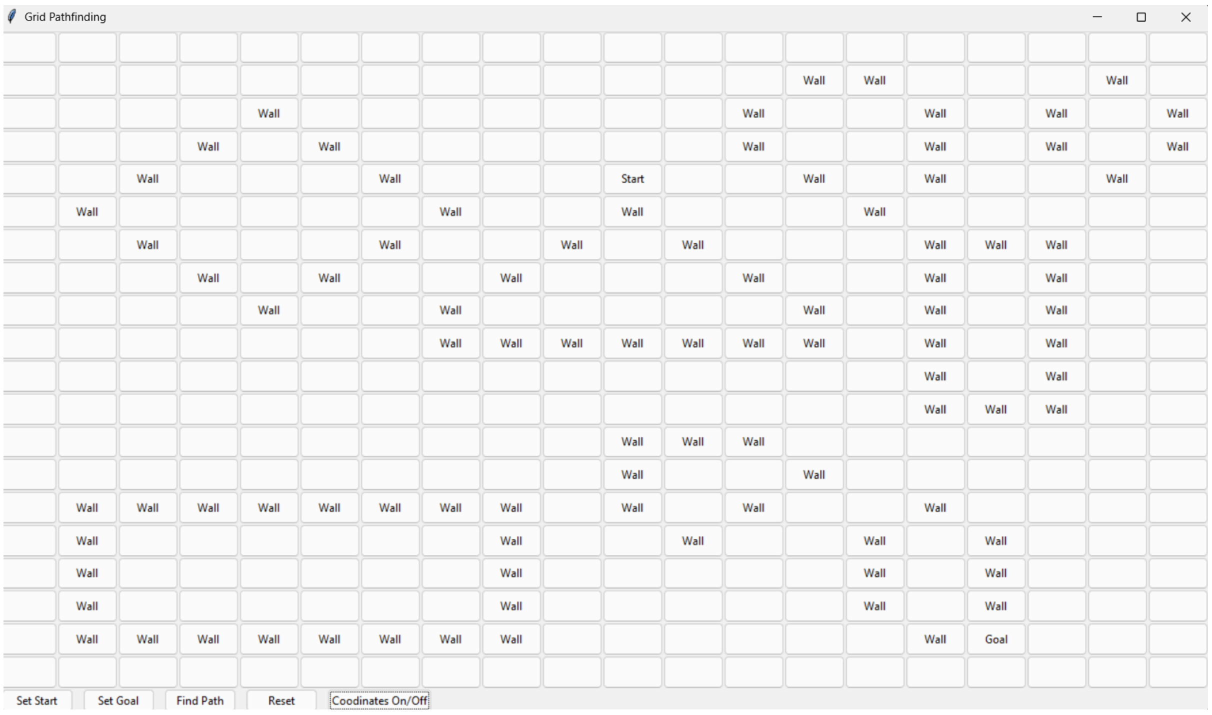
													Wall	Wall				Wall	
				Wall								Wall			Wall		Wall		Wall
			Wall		Wall							Wall			Wall		Wall		Wall
		Wall				Wall							Wall		Wall			Wall	
	Wall						Wall			Wall				Wall					
		Wall				Wall			Wall		Wall				Wall	Wall	Wall		
			Wall		Wall			Wall				Wall			Wall		Wall		
				Wall			Wall						Wall		Wall		Wall		
							Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall		
Start	A*	A*	A*	A*	A*	A*	A*	A*	A*	A*	A*	A*	A*	A*	Wall		Wall		Goal
														A*	Wall	Wall	Wall	A*	A*
										Wall	Wall	Wall		A*	A*	A*	A*	A*	
										Wall			Wall						
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall			Wall				
	Wall							Wall			Wall			Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall							Wall				

Figure 7. Solution of shortest path between start and goal utilizing A* search algorithm for case 2



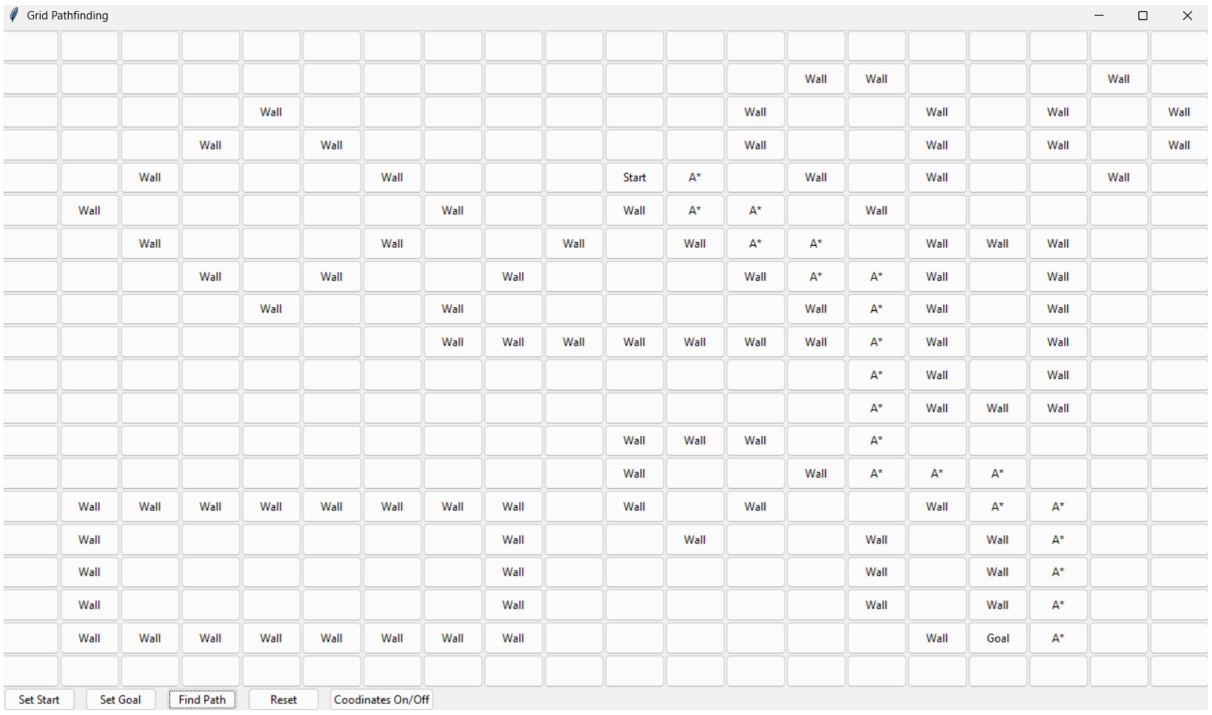
													Wall	Wall				Wall	
				Wall								Wall			Wall		Wall		Wall
			Wall		Wall							Wall			Wall		Wall		Wall
		Wall				Wall							Wall		Wall			Wall	
	Wall						Wall			Wall				Wall					
		Wall				Wall			Wall		Wall				Wall	Wall	Wall		
			Wall		Wall			Wall				Wall			Wall		Wall		
				Wall			Wall						Wall		Wall		Wall		
							Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall		
Start	G	G	G	G	G	G	G	G	G	G	G	G	G	G	Wall		Wall	G	Goal
														G	Wall	Wall	Wall	G	
										Wall	Wall	Wall		G	G	G	G	G	
										Wall			Wall						
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall			Wall				
	Wall							Wall			Wall			Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall							Wall				

Figure 8. Solution of shortest path between start and goal utilizing Greedy Best-First Search algorithm for case 2



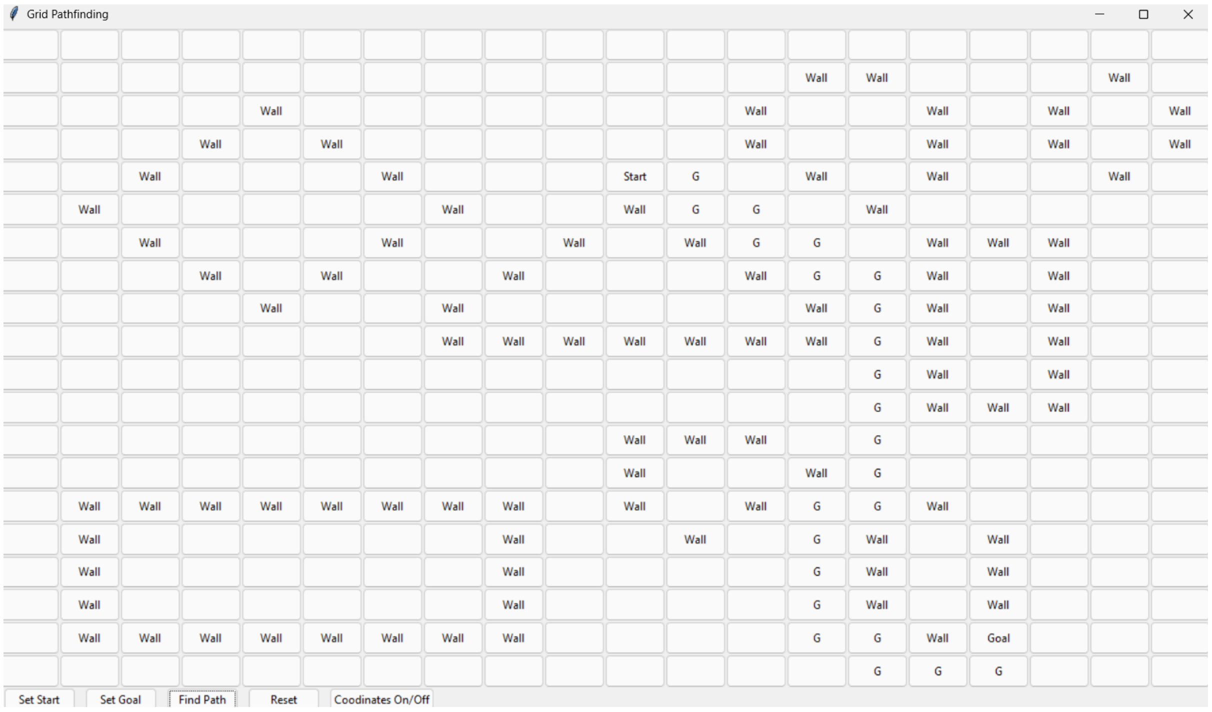
													Wall	Wall				Wall	
				Wall								Wall			Wall		Wall		Wall
			Wall		Wall							Wall			Wall		Wall		Wall
		Wall				Wall				Start			Wall		Wall			Wall	
	Wall						Wall			Wall				Wall					
		Wall				Wall			Wall		Wall				Wall	Wall	Wall		
			Wall		Wall			Wall				Wall			Wall		Wall		
				Wall			Wall						Wall		Wall		Wall		
							Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall		
															Wall		Wall		
															Wall	Wall	Wall		
										Wall	Wall	Wall							
										Wall			Wall						
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall			Wall				
	Wall							Wall			Wall			Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall							Wall						Wall		Wall			
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall							Wall	Goal			

Figure 9. Case 3, Coordinates recreating a tight corner environment without numbers



													Wall	Wall				Wall	
				Wall								Wall			Wall		Wall		Wall
			Wall		Wall							Wall			Wall		Wall		Wall
		Wall				Wall				Start	A*		Wall		Wall			Wall	
	Wall						Wall			Wall	A*	A*		Wall					
		Wall				Wall			Wall		Wall	A*	A*		Wall	Wall	Wall		
			Wall		Wall			Wall				Wall	A*	A*	Wall		Wall		
				Wall			Wall						Wall	A*	Wall		Wall		
							Wall	Wall	Wall	Wall	Wall	Wall	Wall	A*	Wall		Wall		
														A*	Wall		Wall		
														A*	Wall	Wall	Wall		
										Wall	Wall	Wall		A*					
										Wall			Wall	A*	A*	A*			
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall			Wall	A*	A*		
	Wall							Wall			Wall			Wall		Wall	A*		
	Wall							Wall						Wall		Wall	A*		
	Wall							Wall						Wall		Wall	A*		
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall							Wall	Goal	A*		

Figure 10. Solution of shortest path between start and goal utilizing A* search algorithm for case 3



													Wall	Wall				Wall	
				Wall								Wall			Wall		Wall		Wall
			Wall		Wall							Wall			Wall		Wall		Wall
		Wall				Wall				Start	G		Wall		Wall			Wall	
	Wall						Wall			Wall	G	G		Wall					
		Wall				Wall			Wall		Wall	G	G		Wall	Wall	Wall		
			Wall		Wall			Wall				Wall	G	G	Wall		Wall		
				Wall			Wall						Wall	G	Wall		Wall		
							Wall	Wall	Wall	Wall	Wall	Wall	Wall	G	Wall		Wall		
														G	Wall		Wall		
														G	Wall	Wall	Wall		
										Wall	Wall	Wall		G					
										Wall			Wall	G					
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall		Wall		Wall	G	G	Wall				
	Wall							Wall			Wall		G	Wall		Wall			
	Wall							Wall					G	Wall		Wall			
	Wall							Wall					G	Wall		Wall			
	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall					G	G	Wall	Goal			
														G	G	G			

Figure 11. Solution of shortest path between start and goal utilizing Greedy Best-First Search algorithm for case 3

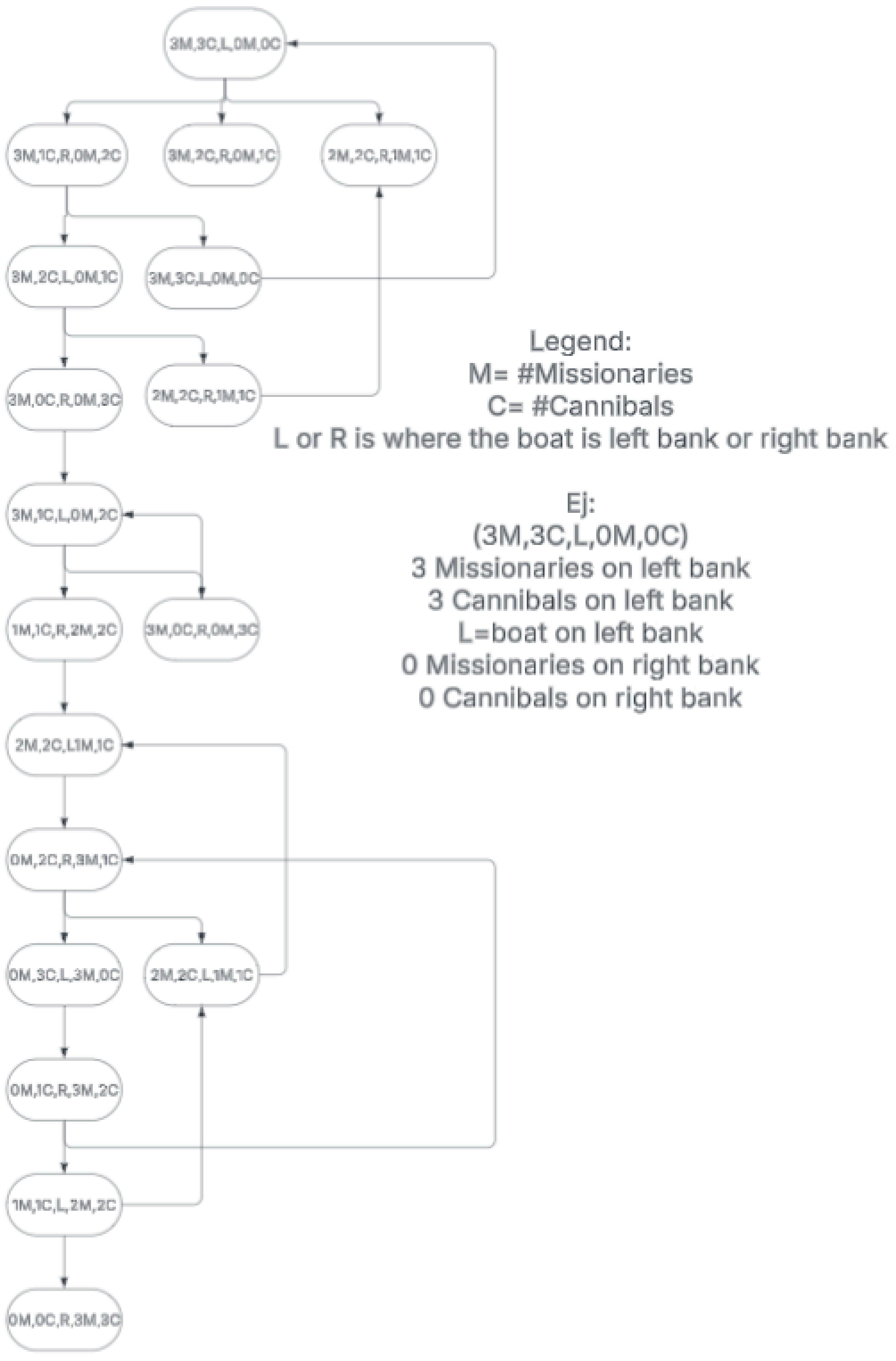


Figure 12: Complete state space for valid moves in missionaries and cannibals problem

IV. Conclusion:

For the first part of the experiment an agent had to travel from point A to point B utilizing only four actions: Left, Right, Up and Down. This agent was studied utilizing two different search algorithms to determine which one was the most optimal. Between Greedy Best First Search algorithm and A* Search algorithm, the most efficient was A* Search. This was due to the fact that A* Search considers multiple conditions to achieve the goal state. While the Greedy Best-First Search algorithm strictly utilizes the heuristic value (the cost from the current position to the goal), A* Search also considers the actual cost from the start node to the end node. The combination of these two values makes A* Search more optimal for this environment. On two out of the three cases studied the A* Search managed to traverse from start to end the fastest. The singular case where Greedy Best-First Search matched A* Search was the one where minimal obstacles were in the way between points. Search algorithms become more and more effective as they decrease the level of uncertainty. More knowledge helps the algorithm make less mistakes and obtain the desired results faster. However, this can also be a downside as more context alters the speed of the algorithm. A* Search requires more information to make a decision on what to do, this is less of a problem for Greed Best-First Search that only considers one thing. Speed and Optimality is something important to consider depending on the environment of the study.

For the second part of the experiment, different search algorithms were used to solve the Missionary and Cannibal problem. This problem formulation takes an analytical approach by defining the agent's initial state (missionaries, cannibals, and boat locations) and describing the actions, where the boat can move left or right with one or two passengers on board. The goal state is reached when everyone has successfully crossed the river without violating the pre-established constraints. We established the path cost by assigning one step for everytime time the algorithm made its way to the goal state. After fully defining the problem, an optimal algorithm was selected without the need to test multiple options. The algorithms utilized and compared were the Breadth First Search and the Depth First Search. The Breadth First Search proved to be the more efficient and shortest path solution. This was due to its ability to search per layer rather than a whole branch segment. This allowed the program to be able to find the shallowest solution quicker by being able to cover more ground. The verification of repeated states would have improved the DFS by preventing it from entering infinite loops. For problems that require searching for the quickest solution in an unweighted environment, these prove to be the best algorithms. The reason this algorithm performed better than any person is due to the number of possibilities and constraints one would need to consider when solving the problem.

References:

[1] R. Belwarier, “A* Search Algorithm,” *geeksforgeeks.org*, Jul 30, 2024. [Online]. Available: A* Search Algorithm - GeeksforGeeks .[Accessed Mar 18, 2025]

[2] C.Yang, S. Kumar, and M. Wardhani. “Greedy Best-First Search,” *codeacademy.com*, Dec 4, 2024. [Online]. Available: AI | Search Algorithms | Greedy Best-First Search | Codecademy . [Accessed Mar 18, 2025]

[3] S. Russell, and P. Norvig, “Artificial Intelligence: A Modern Approach,” *aima.cs.berkely.edu*, Aug 22, 2022. [Online]. Available : https://aima.cs.berkeley.edu . [Accessed Mar 18, 2025]

[4] Aimacode, “aimpa-python,” *github.com*, Feb. 1, 2016. [Online]. Available:https://github.com/aimacode/aima-python . [Accessed Mar. 17, 2025].

[5] Sonuk, “Uninformed Search Algorithms,” *allinformativepost.com*, Jan. 19,2024 [Online]. Available : https://www.allinformativepost.com/uninformed-search-algorithms/ .[Accessed Mar 18, 2025]

[6] G. Tejashree, “Depth First Search (DFS) for Artificial Intelligence” *geeksforgeeks.org*, May 16, 2024. [Online]. Available:https://www.geeksforgeeks.org/depth-first-search-dfs-for-artificial-intelligence/ . [Accessed Mar. 18, 2025]

[7] S. Amarel, “On the Presentation of Problems of Reasoning about Actions,” *RCA Laboratories Princenton ,N.J.*, Mar. 19, 2025. [Online]. Available:https://cse-robotics.engr.tamu.edu/dshell/cs625/MI3-Ch.10-Amarel.pdf . [Accessed Marc. 18, 2025]

Code Repository:

https://github.com/Dahyna-Martinez/Agents-and-Task-Management