CS 461 Artificial Intelligence

Project Report

Group 12

Fahad Waseem Butt - 21801356

Ogulcan Pirim - 21702280

Agil Aliyev - 21701367

Berdan Akyurek - 21600904

Gokberk Boz - 21602558

Spring 2022

TABLE OF CONTENTS

1.0 INTRODUCTION

The goal of this project was to implement the various algorithms learnt in the Artificial Intelligence course. This was to be attempted on a brand new environment, the Unity 3D interface in C#. For this purpose, the game Quoridor was chosen as it is a deterministic game and also because searching methods can be reasonably implemented to facilitate pawn movement. Up until this phase, the goal was to achieve the implementation of Search Algorithms and Adversarial Search Algorithms, with a possibility to attempt further algorithms if time permitted such. The main achievements were progress in implementing the game in the Unity environment and the implementation of the Search Algorithms, with further implementations not being feasible to implement due to a time constraint.

To solve the problem of finding a path to the goal state, Searching Algorithms may be implemented to accomplish this. In this project, all the algorithms discussed in class, pertaining to what was set out to be achieved, were implemented. The Depth First Search Algorithm starts at the initial node and expands deeper until a node with no children is reached and then repeats. The Breadth First Search Algorithm starts at the initial node and expands neighbouring nodes first before traversing deeper. The Uniform Cost Search Algorithm starts at the initial node and finds the lowest cumulative cost to reach the goal, which is in addition to its working following the Breadth First Search Method. When solving the problem of finding the shortest path to the goal with the lowest cost, the A Star Search Algorithm is usually the best graph traversal method to use. Details of the algorithms implemented will be elaborated on further following this section. When observing the algorithms in action when the game runs, it is observed that the pawns move according to the briefly aforementioned expectations.

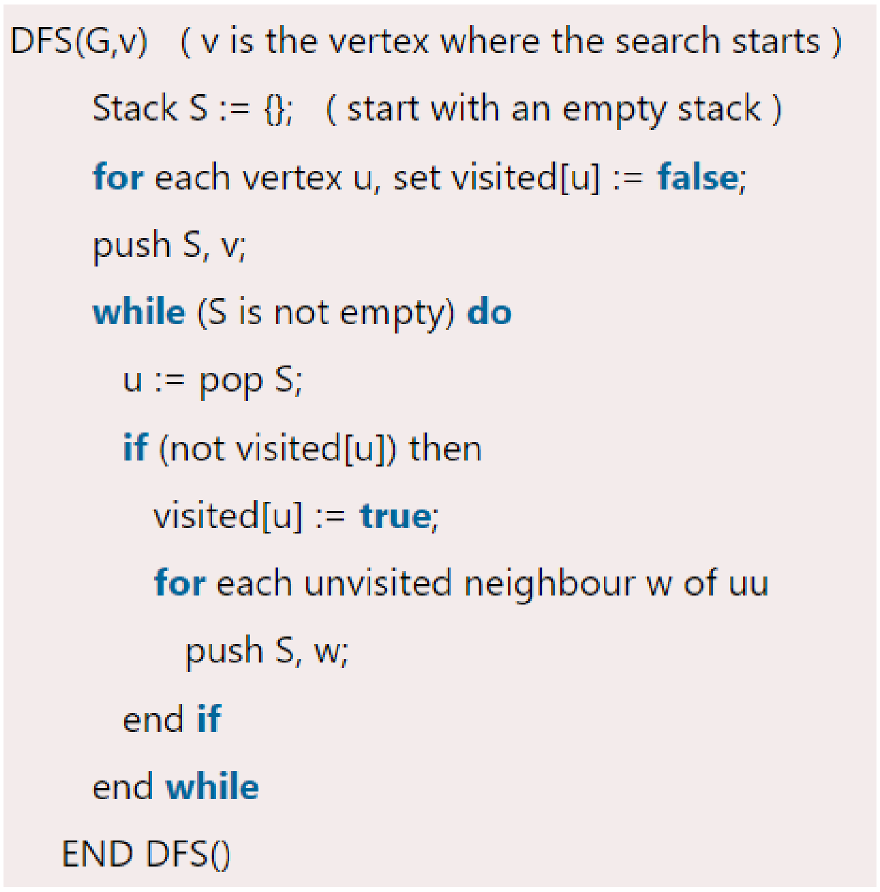
2.0 BACKGROUND

2.1 Problem Domain

Quoridor is a strategy game that has a significant level of complexity despite having fairly simple rules. The goal of the players is to move their pawns to the opposite side of the board in order to win. At each turn, a player may move their pawn by one space or place a wall (vertically or horizontally). The placed walls may make the path of the opponent more difficult, but blocking the path to the goal is not allowed. [1]. This game has been chosen as it is a deterministic game, wherein players take turns to make their moves and there is a clear outcome after a finite number of iterations that one player wins or one player can make no further moves [2]. Hence such a game fits the requirements to explore the possibilities for Artificial Intelligence algorithms that suit our goals.

2.2 Depth First Search Algorithm

The Depth First Search Algorithm is a recursive search method which goes from the starting node of a graph or tree and then goes deeper down the children nodes until no more are found, and then proceeds down adjacent branches. In more technical terms, a starting node is chosen, with all adjacent nodes pushed to a stack. To visit the next node, a node is popped from the stack, with all the new adjacent nodes also being pushed into a stack. This process is repeated until the stack is depleted. To avoid the case of revisiting already visited nodes, such nodes are marked and collected in another data structure [3]. The pseudocode for the Depth First Search Algorithm is as in Figure 1.

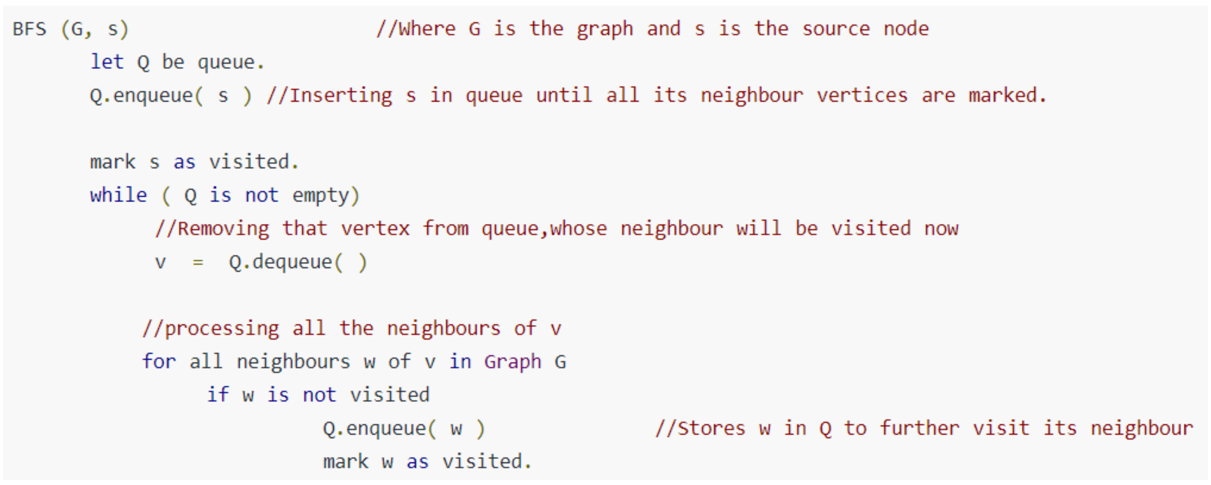


```
DFS(G,v)   ( v is the vertex where the search starts )
      Stack S := {};   ( start with an empty stack )
      for each vertex u, set visited[u] := false;
      push S, v;
      while (S is not empty) do
        u := pop S;
        if (not visited[u]) then
          visited[u] := true;
          for each unvisited neighbour w of uu
            push S, w;
        end if
      end while
    END DFS()
```

Figure 1: Depth First Search Algorithm Pseudocode [4]

2.3 Breadth First Search Algorithm

As opposed to Depth First Search, the Breadth First Search Algorithm first visits all adjacent nodes in a tree or graph, if any exist, before going deeper. In more technical terms, the Breadth First Search Method initiates by starting at, for instance, a vertex of some graph and placing it at the back of a queue. Next, the front node of the queue is taken and placed into a data structure, for instance a list, containing visited nodes. A list is made that contains the nodes adjacent to the vertex, and any nodes that have not been visited are added, however, to the back of the queue. This process is repeated until a goal state is reached or the queue is exhausted [5]. The pseudocode for the Depth First Search Algorithm is as in Figure 2.



```
BFS (G, s)                   //Where G is the graph and s is the source node
      let Q be queue.
      Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.

      mark s as visited.
      while ( Q is not empty)
           //Removing that vertex from queue,whose neighbour will be visited now
           v  =  Q.dequeue( )

          //processing all the neighbours of v
          for all neighbours w of v in Graph G
               if w is not visited
                        Q.enqueue( w )             //Stores w in Q to further visit its neighbour
                        mark w as visited.
```

Figure 2: Breadth First Search Algorithm Pseudocode [6]

2.4 Uniform Cost Search Algorithm

The Uniform Cost Search Algorithm is similar to the Breadth First Search Method, however this time, movement costs are considered and actions with lower costs gain priority. Even in more technical terms the similarity can be observed. This time, the starting node is placed inside a priority queue. Elements with the largest priority are removed, with the children being placed in the priority queue in a similar fashion, in that their cost determines the priority, with visited nodes being placed in a separate data structure to keep track of them. This process is repeated until reaching the goal node or if the queue is exhausted [7]. The pseudocode for the Uniform Cost Search Algorithm is as in Figure 3.

```
$g(s) \leftarrow 0$
// $g(node)$ is the cost of the path from $s$ to the node.
$frontier \leftarrow$ a min-priority queue ordered by $g$, containing only $s$
$expanded \leftarrow$ an empty set
while $true$ do
    if $frontier$ is empty then
        return $failure$
    end
    $v \leftarrow$ pop the lowest-cost node from $frontier$
    if $goal(v) = true$ then
        return $v$
    end
    $expanded \leftarrow expanded \cup \{v.state\}$
    for $w \in successors(v)$ do
        if $w.state \notin expanded$ and no frontier node represents
         $w.state$ then
            $g(w) \leftarrow g(v) + c(v, w)$
            Insert $w$ in $frontier$
        else if there's a frontier node $u$ such that $w.state = u.state$
         and $g(v) + c(v, w) < g(u)$ then
            Remove $u$ from $frontier$
            Insert $w$ in $frontier$
    end
end
```

Figure 2: Breadth First Search Algorithm Pseudocode [8]

2.5 A Star Search Algorithm

When compared with some other informed search algorithms, the A Star Search is good when the goal is to reach the goal with the minimum cost, that is to say, choosing the path which has the smallest distance from start to goal while also achieving this in the least time. As with most searching algorithms, the A Star Search starts at a starting position and expands to other points until it reaches a goal state. It makes use of a sum (represented by f(n)) between the cost of the current move (represented by g(n)) and the shortest cost remaining until reaching the goal state (represented by h(n)), and has been shown in the equation below [9].

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

Further, the pseudocode for this algorithm is shown in Figure 4.

The goal node is denoted by node_goal and the source node is denoted by node_start

We maintain two lists: **OPEN** and **CLOSE**:

OPEN consists on nodes that have been visited but not expanded (meaning that sucessors have not been explored yet). This is the list of pending tasks.

CLOSE consists on nodes that have been visited *and* expanded (sucessors have been explored already and included in the open list, if this was the case).

```
Put node_start in the OPEN list with $f$(node_start) = $h$(node_start) (initialization)
while the OPEN list is not empty {
  Take from the open list the node node_current with the lowest
      $f$(node_current) = $g$(node_current) + $h$(node_current)
  if node_current is node_goal we have found the solution; break
  Generate each state node_successor that come after node_current
  for each node_successor of node_current {
    Set successor_current_cost = $g$(node_current) + $w$(node_current, node_successor)
    if node_successor is in the OPEN list {
      if $g$(node_successor) $\leq$ successor_current_cost continue (to line 20)
    } else if node_successor is in the CLOSED list {
      if $g$(node_successor) $\leq$ successor_current_cost continue (to line 20)
      Move node_successor from the CLOSED list to the OPEN list
    } else {
      Add node_successor to the OPEN list
      Set $h$(node_successor) to be the heuristic distance to node_goal
    }
    Set $g$(node_successor) = successor_current_cost
    Set the parent of node_successor to node_current
  }
  Add node_current to the CLOSED list
}
if(node_current != node_goal) exit with error (the OPEN list is empty)
```

Figure 4: A Star Algorithm Pseudocode [10]

3.0 RESULTS

The Quoridor game was implemented from scratch on Unity and the board is as in Figure 5.

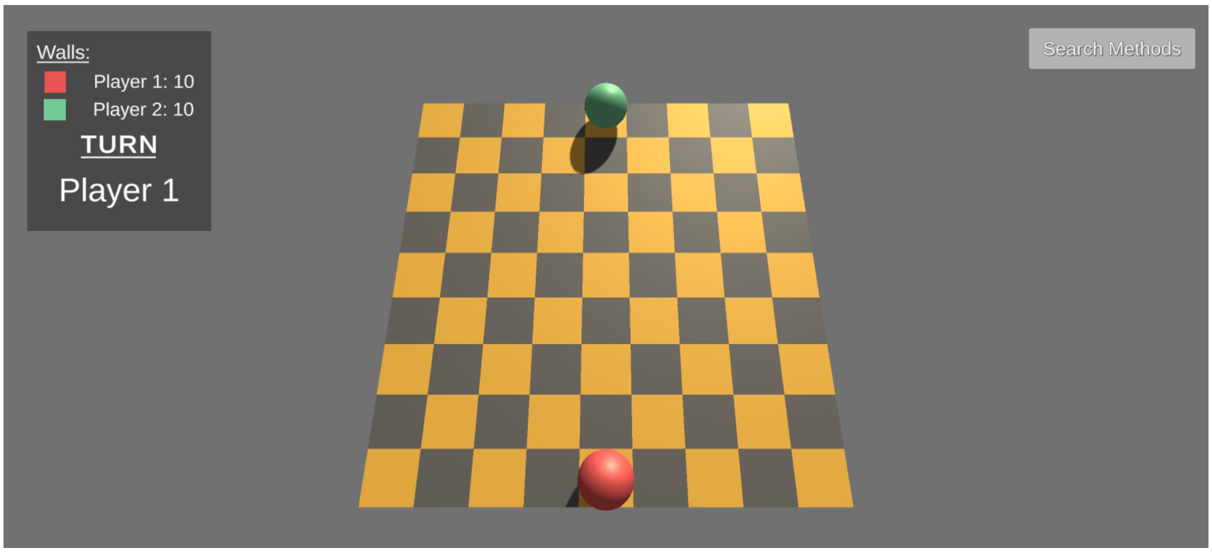


Figure 5: The Quoridor Game Board on Unity

3.1 Results for Searching Algorithms

The Searching Algorithms were evaluated by taking the number of nodes expanded. The possible movements were Up (U), Down (D), Right (R) and Left (L). To test the algorithms, four puzzles were made, and each Searching Method was run over all four puzzles, with the number of nodes (count of U, D, R, L) taken in tabular form as data to be evaluated.

3.1.a Depth First Search Results

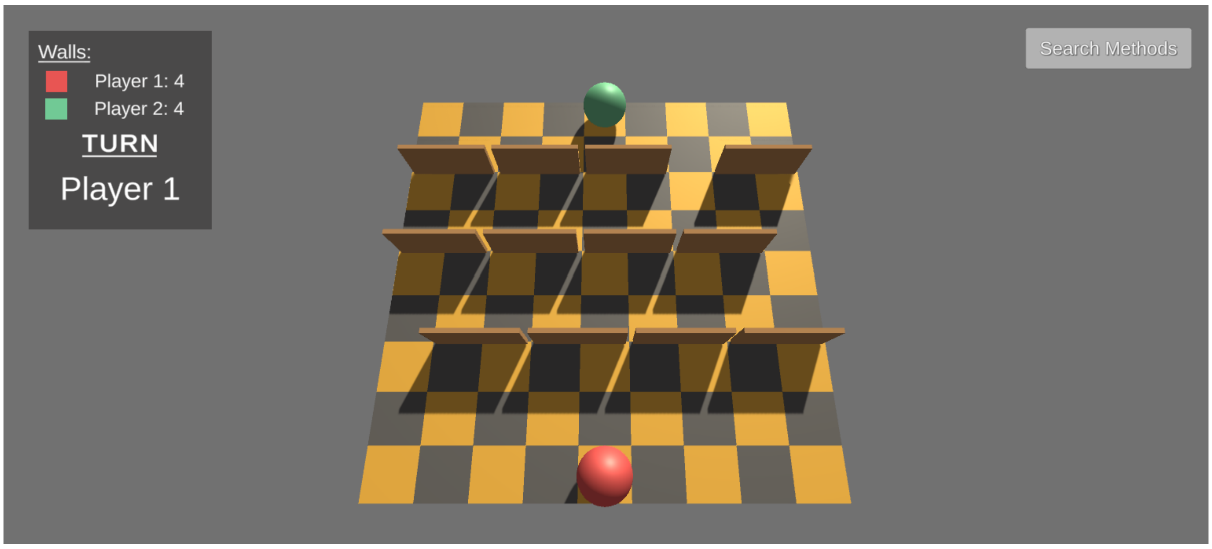


Figure 6: Puzzle #1 Given to Agent (Red)

Puzzle No.	Path Taken By Agent	Number of Nodes Expanded
1	UULDDLUULDDLUUUURDRURDRURD RURDRUUULDLUUU	40
2	URDRURUUULDDLUULDDLUUUURDR URURRUU	33
3	UULLDLUURRULUURDRUULLU	22
4	URDRURDRUUULDLULDLULDLUULD DDDLUUUUUURDRURDDRUURDDRRR ULLUUU	58

Table 1: Depth First Search Results Enumerated

3.1.b Breadth First Search Results

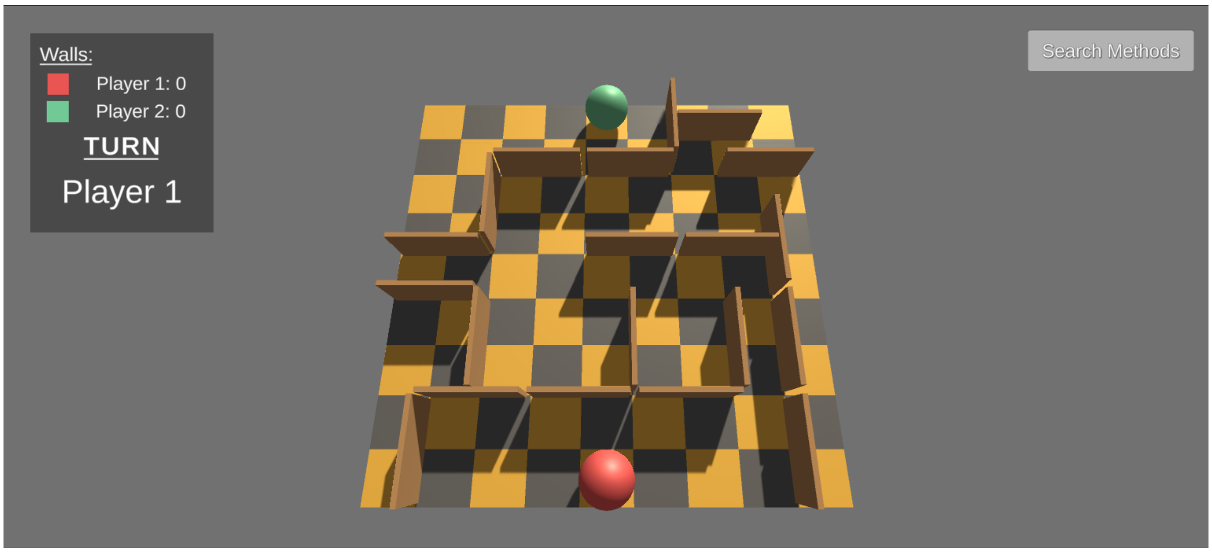


Figure 7: Puzzle #2 Given to Agent (Red)

Puzzle No.	Path Taken By Agent	Number of Nodes Expanded
1	LLLLUUURRRRRRRRUULLUUU	22
2	RRRUUUULLLLURRRUURRUU	21
3	UURRULLURULUURU	15
4	RRRUULLLLLUULLURRURRRDDRRRU LLUUU	32

Table 2: Breadth First Search & Uniform Cost Search Results Enumerated

3.1.c Uniform Cost Search Results

In this implementation, the movement cost for each move is the same. As such, the Uniform Cost Search Algorithm acts exactly like the Breadth First Search Algorithm. Hence the results for this section are the same as those gathered in Table 2.

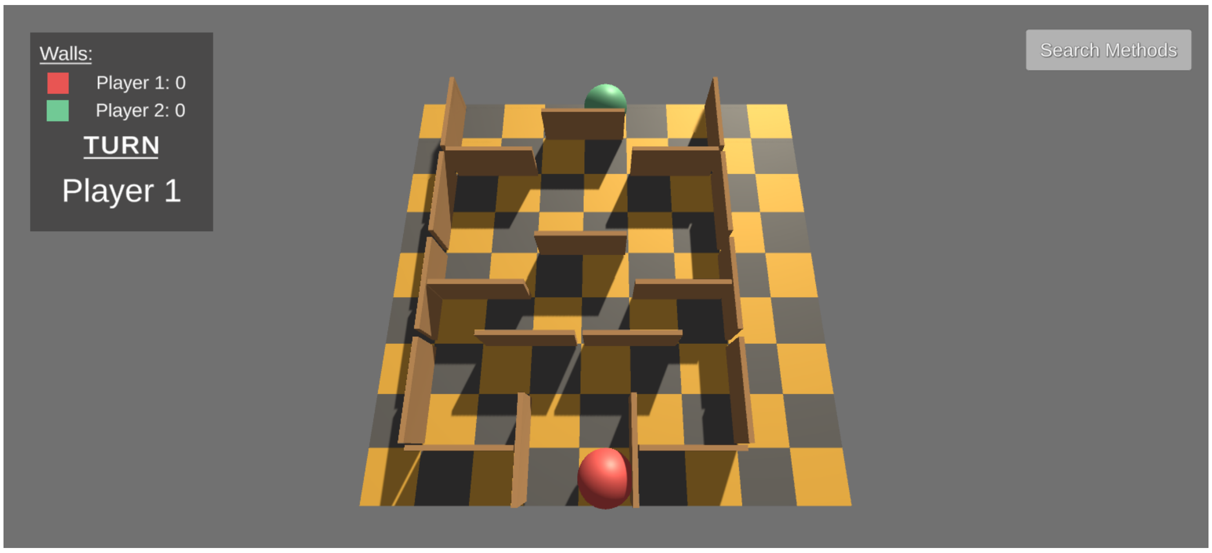


Figure 8: Puzzle #3 Given to Agent (Red)

3.1.d A Star Search Results

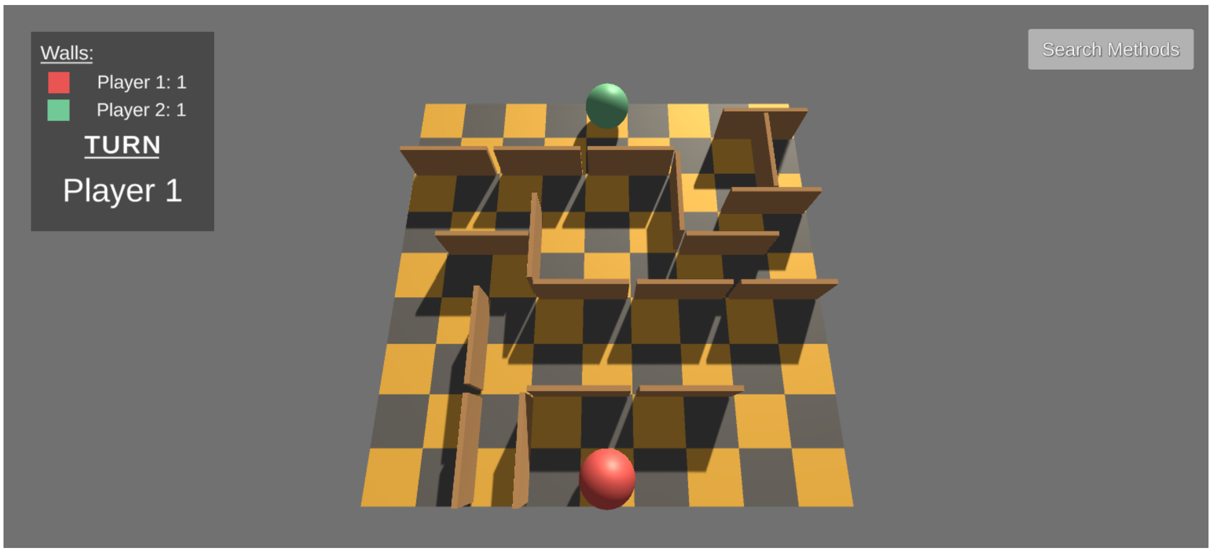


Figure 9: Puzzle #4 Given to Agent (Red)

The Heuristic chosen for this implementation of the A Star Search Method is the straight line distance from the Agent to the nearest goal tile, which is on the opposing side of the player. As there are 9 goal states, on this 9x9 board, this is the only reasonable heuristic that may be applied, as otherwise, the Agent possibly may not be able to choose between more than one goal state to calculate the Heuristic from.

Puzzle #	Path Taken By Agent	Number of Nodes Expanded
1	RULLLLDLUUURRRRRRRUULLUUU	26
2	LURRRDRUUUULLLLURRRUURRUU	25
3	LURURDRUULLURULUURU	19
4	LURRRDRUULLLLLUULLURRURRRDD RRRULLUUU	36

Table 3: A Star Search Results Enumerated

4.0 DISCUSSION

4.1 Results

4.1.a Searching Algorithms

For testing the Searching Algorithms, four sample puzzles were made for the Search Methods to solve and the Nodes expanded to solve them were recorded so that they may be compared. It was noted that the Artificial Intelligence Agents following each method made the same choices in each game, when given the same puzzle. This is expected as the path to the goal stays the same in each case since the Searching Methods expand the same nodes each time.

The Depth First Search Algorithm was expected to go through each possible node depth-wise, such that the path chosen would end up covering all nodes leading to the goal rather than the shorter path. When looking at the path taken by the Depth First Search Agent, it can be seen that it unnecessarily moved up and down tiles before moving adjacent (Right or Left), and hence expanding the nodes through unnecessary routes.

As has been stated prior, the Uniform Cost Search Algorithm works just like the Breadth First Search Algorithm because the movement cost in each direction, everywhere on the board is the same. Hence, looking at the results for these algorithms, it can be noted that the Agent takes the best path to the goal state, i.e. the shortest path. This becomes more prominent when comparing with the result for the Depth First Search in that the number of nodes expanded in the Breadth First Search Method is considerably smaller.

When looking at the results for the A Star Search Algorithm, the observations gathered are a little unexpected. The number of nodes expanded and the path observed were both significantly shorter than those of the Depth First Search Algorithm, however, there were some unnecessary movements made by the Agent. This can be seen numerically, as the number of nodes expanded in the A Star Search Method are consistently 4 nodes more than those in the fully efficient results for the Breadth First Search Method. When looking at the nodes expanded more closely (U, D, R, L), it is seen that the extra 4 moves occur at the start. The A Star Search Agent mistakenly moves Right or Left in the first move, and soon makes a mistaken movement Up, which makes up the first 2 extra moves. Then the Agent moves in order to undo the incorrect movement by moving Left or Right, and Down, which makes up the remaining 2 extra moves. This unnecessary movement at the start is caused by the heuristic. When at the initial state, considering moving left or right, the heuristic chosen gives the same value for both movements, left or right. As such, this makes the Agent move irrationally at the start. But as it gets closer to the goal state, and traverses the puzzle, it can be seen that the results for the A Star Search Method converge back to the Breadth First Search (or Uniform Cost Search) Method.

Overall, it can be said that all the algorithms were successfully implemented, and everything that was needed to evaluate the Searching Algorithms has been observed and discussed.

4.2 Alternatives

Considering that this is a deterministic game, and that the achieved results, as well as the proposed alternative method of Adversarial Search, are essentially searching methods, there may not be a way to directly compare these with other Algorithms. However, had the Adversarial Search Algorithms, Minimax and Expectimax, been implemented, the game could be played in more ways, with possibilities open to make the Adversarial Search Methods against the Search Methods. Alternatives to this can be the implementation of more Artificial Intelligence Algorithms, such as Reinforcement Learning and testing how the trained Agent would interact with the ones already present. And considering how an Adversarial Search Algorithm might play the game, as well as how a human player playing against the Artificial Intelligence Agent, in such conditions may make the results differ. If there were more time to continue working on the project, or perhaps if it had started earlier, with the current knowledge, work would be faster, and more effort would be put in trying out further implementations of Artificial Intelligence Algorithms and making them play against each other.

4.3 Insights

There have been insights gained in working in the Unity environment, as well as understanding how to take algorithms that worked on previously in Python and then making them work properly in C#. Due to a lack of time to be able to make all the components of this project work properly, there has not yet been an exploration into some methods to see how they would perform. A big cause for losing time was to work with an already developed game found online, and then making the algorithms work on it. However, it was found that the repository found was not clear enough so as to facilitate making all the required algorithms work. As such, more time needed to be spent in order to make the Quoridor game on Unity from scratch, so that the program could be written in a way that allowed for making the algorithms more feasible. A way to extend researching how the Searching Algorithms work would be to extend the problem domain to include a human player playing against an Artificial Intelligence Agent, which would provide more interesting data, with more variety, to make further comparisons.

5.0 REFERENCES

[1] M. Marchesi. “Quoridor”. BoardGameGeek. https://boardgamegeek.com/boardgame/624/quoridor

[2] “Deterministic Games”. University of Washington Department of Mathematics. April 2014. https://sites.math.washington.edu/~mathcircle/circle/2013-14/second/games.pdf

[3] P. Garg. “Depth First Search Tutorials & Notes”. Hackerearth. https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/

[4] “DFS Algorithm”. Javatpoint. https://www.javatpoint.com/depth-first-search-algorithm

[5] S. Bhadaniya. “Breadth First Search in Python (with Code) | BFS Algorithm”. FavTutor. December 2020. https://favtutor.com/blogs/breadth-first-search-python

[6] P. Garg. “Breadth First Search Tutorials & Notes”. HackerEarth. https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

[7] F. Hasan. “What is uniform-cost search?”. Educative. https://www.educative.io/edpresso/what-is-uniform-cost-search

[8] M. Simic. “Uniform-Cost Search vs. Best-First Search”. Baeldung. October 2021. https://www.baeldung.com/cs/uniform-cost-search-vs-best-first-search

[9] Edpresso Team. “What is the A* algorithm?”. Educative. https://www.educative.io/edpresso/what-is-the-a-star-algorithm

[10] L. Alseda. “A* Algorithm pseudocode”. Universitat Autonoma De Barcelona. https://mat.uab.cat/~alseda/MasterOpt/AStar-Algorithm.pdf

6.0 APPENDIX:

6.1 Link to GitHub Repository

The link to the GitHub Repository is as follows. Note that the branch to use should be FSum.

https://github.com/ogulcanpirim/AI-Project/tree/FSum

6.2 Division of Work

Ogulcan Pirim has worked on developing the Quoridor game in the Unity environment, with support from Fahad Waseem Butt on some parts. The A Star Search Method was worked on by Fahad and Ogulcan. Berdan Akyurek and Agil Aliyev did work on the Breadth First Search and Depth First Search Methods. The contributions mentioned are the key areas each member has worked on, however, all the mentioned group members have worked such that the implementation accomplished is a cumulative result of the group's efforts.