*Nada Bounajma 77807 - worked along with Nour ELHouda Touti*

*Spring 2022 - Artificial Intelligence Project*

**Search Algorithms Analysis**

*All credit goes to Sebastian Lague*

1. **Overview of the Gaming Environment**

The gaming environment *A\_start* *copy* shown below was created using Unity (version 2020.3.27f1). The green surface represents the plane which holds 3000 nodes, cubes for obstacles, and white capsules for the Seeker and Target.

**The files accompanying the gaming environment are :**

* The Scene *A\_star* which displays all gaming Objects(found inside the Scenes Folder)
* C# files, for searching algorithms to allow the Seeker find the Target
* Material files, which were used to color the ground and cubes.

Graphical user interface, application

Description automatically generated

1. **Searching Algorithms Visualization**

**A screenshot of a computer

Description automatically generated with medium confidence**

We have a total of 5 Searching Algorithms represented in different colors by modifying code in *Grid.cs* to enable that option :

* A\* in **Black**
* A\* with a different Heuristic in **Yellow**
* BFS in **Cyan**
* DFS in **Blue**
* UCS in **Green**

Some paths are not shown properly because of overlapping. We will therefore show them separately and analyze each one of them along the way. For reference, all their respective codes are found inside *PathFinding.cs*

1. **Search Algorithms Analysis**

It is safe to say that not all the Fringe nodes get to be explored in any Search Algorithms, but only the ones that are prioritized by their respective data structure. For instance, suppose we added the 8 neighbors of a current node, the next time we refer to a new current node because it got prioritized, it might happen to be the Target Node, as a result, 7 fringe nodes will remain stored and unexplored. For that matter we have chosen the Fringe Nodes to represent our Space complexity. With regards to Time complexity, we have created an instance of type *StopWatch* which will be our Timer. We start the Timer after variables and data structures declaration to remove any overhead needed for the algorithm's functioning. Then we stopped the Timer when the current node is the target node.

1. **A\* Search Algorithm**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Algorithm Description**

Prioritizes nodes with the least fCost(past cost + future cost) and smaller hCost by removing them, checking if they are the Target Node, if not, we proceed to expand them by adding their Neighbors to the Priority Queue. It keeps repeating the same process until the next node it prioritizes happens to be the Target Node.

**Time and Space Complexity**

Algorithm stored 145 nodes, explored 86 of them, and took 1 millisecond to find the Target. The small number of nodes stored shows how efficient and optimal this searching algorithm is compared to the others.

1. **A\* With Different Heuristic**

**A screenshot of a computer

Description automatically generated**

**Algorithm Description**

Uses the same logic as A\*, the only difference is in the heuristic function which is essential for determining priority. It chooses instead the node that is the closest to the Start Node (refer to *Node.cs*)

**Time and Space Complexity**

The algorithm stored 605 nodes, explored 576 of them, and took 2 millisecond to find the Target. Unlike A\*, this algorithm is given an fCost2 that is not as efficient in finding the best node to prioritize. We made fCost2 function return gCost instead. Therefore, whenever the algorithm tries to expand the node that is the closest to the Target, it will instead expand the node that is the closest to the Seeker. This results in visiting more nodes than needed in hopes of finding the Target.

1. **BFS Search Algorithm**

A screenshot of a computer

Description automatically generated with medium confidence

**Algorithm Description**

BFS uses FIFO logic in which the node that is added first to the queue is the node that is first dequeued. Whenever a node is dequeued we check whether it is the Target Node, if not, we add its neighbors to the queue and repeat the same process until the goal is found.

**Time and Space Complexity**

The Algorithm stored 2058 nodes, explored 1959, and took only 2 milliseconds.

Considering the positioning of the capsules, the BFS searching algorithm will not perform better than the DFS. Because in this case, the BFS will have to go approximately over 70% of the grid starting from the bottom to the top. That explains the huge fringe number 2058. As for why nearly all of them got explored was because the BFS just like its name, is a Breath first search algorithm, it explores nodes before allowing itself to visit other nodes.

1. **DFS Search Algorithm**

A screenshot of a computer

Description automatically generated

**Algorithm Description**

DFS uses LIFO logic in which the node that is pushed last to the stack is the node that is popped first. Whenever a node is popped, we check whether it is the Target Node, if not, we add its neighbors to the stack and repeat the same process until the goal is found.

**Time and Space Complexity**

The Algorithm stored 1200 nodes, explored 335, and took 1 millisecond which was faster than BFS. In this case, the DFS will perform better than the BFS as the algorithm only explored nearly half of the grid. The nodes explored is significantly smaller compared to that of the BFS because it happened to find a node that if explored vertically, it will become a path to the goal.

1. **UCS Search Algorithm**

**A screenshot of a computer

Description automatically generated**

**Algorithm Description**

Uses the same concept as A\* Search but when it comes to prioritizing, UCS prioritizes the node with the smallest cost (gCost) since the start node.

**Time and Space Complexity**

The Algorithm stored 2261 nodes, explored 589, and took 2 milliseconds.

Instead of choosing the node with the smallest fCost, UCS chooses the node that has the smallest cost since the start node which explains why it is not as efficient as A\* Search.