Ayoub Mabkhout

Instructor: Dr. Tajjeeddine Rachidi

CSC 4301 Intro. to Artificial Intelligence

February 13, 2022

Introduction to Artificial Intelligence Project #1 Report: Experiment Comparing the Performance of Several Graph Search Algorithms in a Pathfinding Problem

Table Of Contents

Introduction

The principal objective of this first project was to try different approaches to pathfinding algorithms, such as Depth-First Search (DFS) which is implemented using a stack fringe and expands the deepest node until it finds a path, Breadth-First Search (BFS) which expands the shallowest node using a queue, and Uniform-Cost Search (UCS) which is a modification of BFS using a priority queue to find the least costly path, and finally A* which uses a heuristics function to conduct a better, more informed, search for an optimal path. We then compared each approach in terms of performance and completeness.

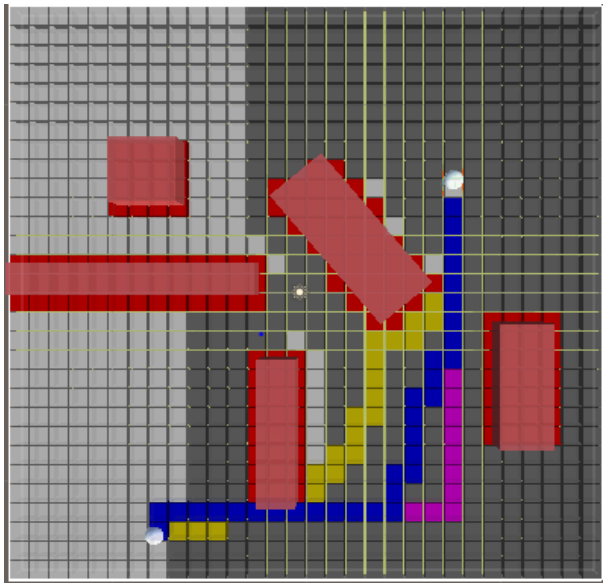
In order to run these tests and comparisons, we used Unity 3D with a top-down 2D view and VS Code to program our simulations in C#.

We built this experiment on top of Sebastian Lague's amazing work on pathfinding in Unity, the original project files before the experiment was built can be found on Github using the following link: https://github.com/SebLague/Pathfinding/tree/master/Episode%2003%20-%20astar

Different Paths

Due to weird unsolved bugs, we were forced to disable the seeker's ability to travel diagonally because the paths generated otherwise were inconsistent. In our version, the seeker can only move horizontally or vertically and the cost to move from one node to the next is always 10.

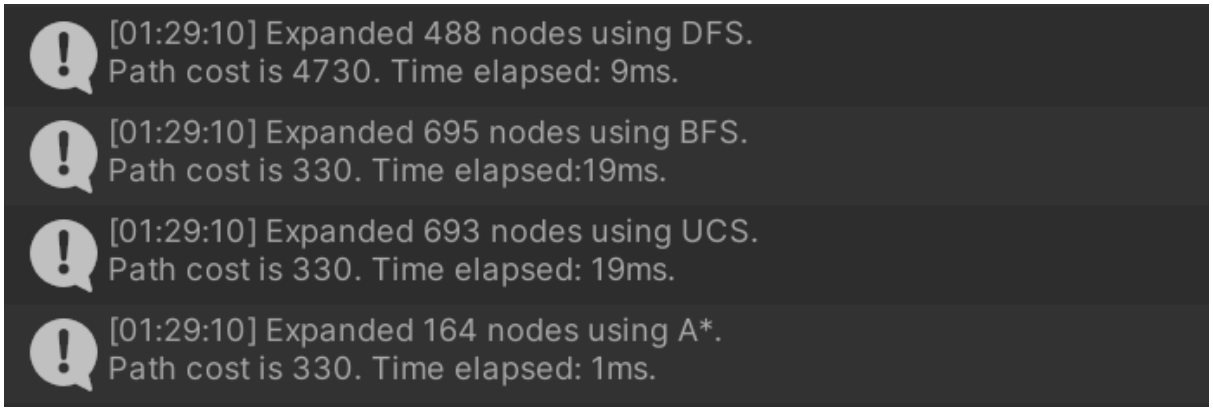
Below is a screenshot showing all four algorithms running at the same time with their respective paths, along with a description of the color codes:



- White: Default, nodes that are not part of any path
- Red: Obstacle
- Gray: DFS Path
- Magenta: BFS Path
- Yellow: UCS Path
- Blue: A* Path

Performance Comparison

After running the experiment, the above picture in the previous section represents what Unity has drawn on the Gizmos API. The picture below shows what was printed to the Debug Console. We used the Debug Console to take note of the performance of each algorithm.



Let us now analyze our performance results. To do so, we are recording 4 main metrics which are: the number of nodes expanded, the cost of the path, the time elapsed to run the algorithm, and finally, the memory used. Concerning the memory, we will be using the task manager to record the change in memory usage by the Unity Editor process. We noticed that, on idle, the unity editor uses an average of 250 MB of RAM, with an error of plus or minus 40 Mb. We will subtract 250 MB from the memory usage of the process when running each of the algorithms individually in order to measure and compare the memory usage of the algorithms.

DFS:

DFS did not prove to find the shortest path of the bunch, as it would take note of the first path it finds in its depth-based expansion of the nodes and their neighbours. The resulting path may look silly and have higher space cost overall, but the algorithm performed okay in terms of time.

- Expanded Nodes: 488
- Path Cost: 4730
- Time Elapsed: 9ms
- Memory cost: 507 MB – 250 MB = 257 MB

BFS:

BFS keeps expanding nodes in every direction until it finds the target node, it then returns the first path that found that node. If the cost to travel from one node to another is constant, then the solution is always optimal.

- Expanded Nodes: 695
- Path Cost: 330
- Time Elapsed: 19ms
- Memory cost: 542 MB – 250 MB = 292 MB

UCS:

UCS also expands in all directions but not simultaneously. UCS uses Djikstra's shortest path algorithm, which expands the lowest path so far. Given that the cost to move from one node to another is constant in our example, the algorithm expands nodes in every direction in a very similar fashion to BFS. The difference between the two algorithms would be very significant if the cost of traveling was variable.

- Expanded Nodes: 693
- Path Cost: 330
- Time Elapsed: 19ms
- Memory cost: 512 MB – 250 MB = 262 MB

A*:

A* uses the heuristics function, which in our case is the GetDistance function wrote back Sebastian Lague, in order to guide itself more directly to the goal. As a result, A* does not expand nearly as many nodes as the previous algorithms

- Expanded Nodes: 164
- Path Cost: 330
- Time Elapsed: 1ms (incredibly fast)
- Memory cost: 515 MB – 250 MB = 265 MB

Performance Table:

Algorithm	Expanded Nodes	Path Cost	Time Elapsed	Memory Cost
DFS	488	4730	9ms	257 MB
BFS	695	330	19ms	292 MB

UCS	693	330	19ms	262 MB
A*	164	330	1ms	265 MB

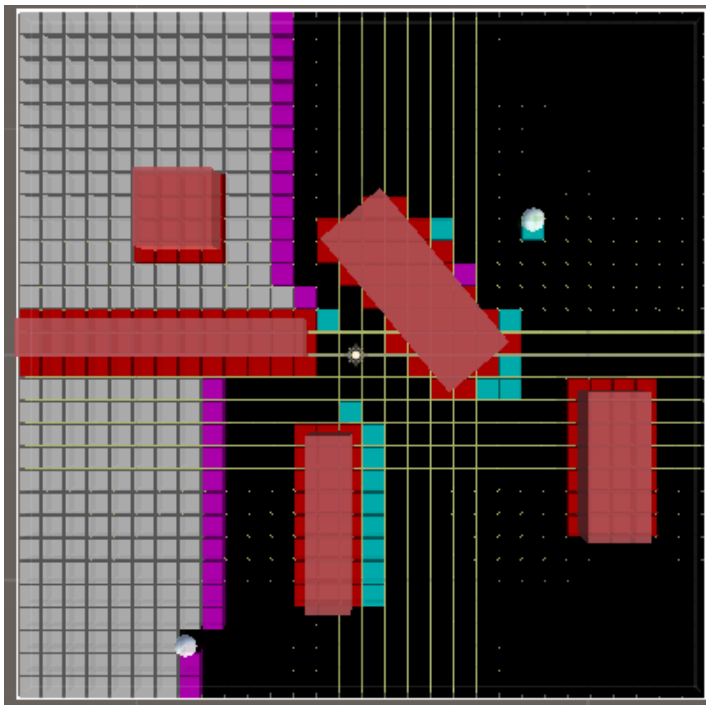
Visualizing Node Expansion

In this section, instead of showing the paths found by all four algorithms simultaneously, we zoom on each algorithm individually in order to better visualize the expansion of nodes. In each of the following screenshots, the nodes are represented with the following color code:

- White: Default
- Red: Obstacle
- Magenta: Discovered nodes
- Cyan: Expanded nodes
- Black: Path

Note that magenta nodes will always come at the edge of the cyan area because the nodes colored in magenta are only the nodes that have been discovered by their neighbours but haven't been expanded into.

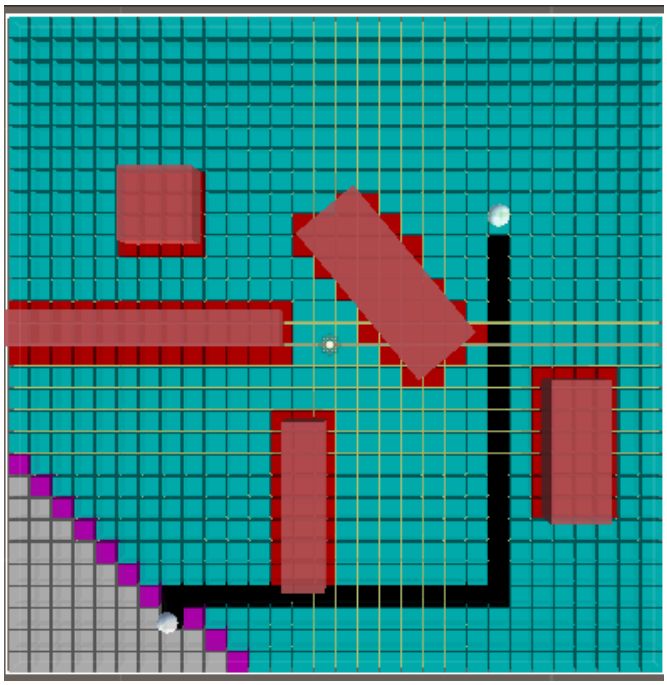
DFS:



Here is a visual representation of DFS. It is the only algorithm that produces a non-optimal path in our tests. Here, the seeker keeps following one path, blindly, only

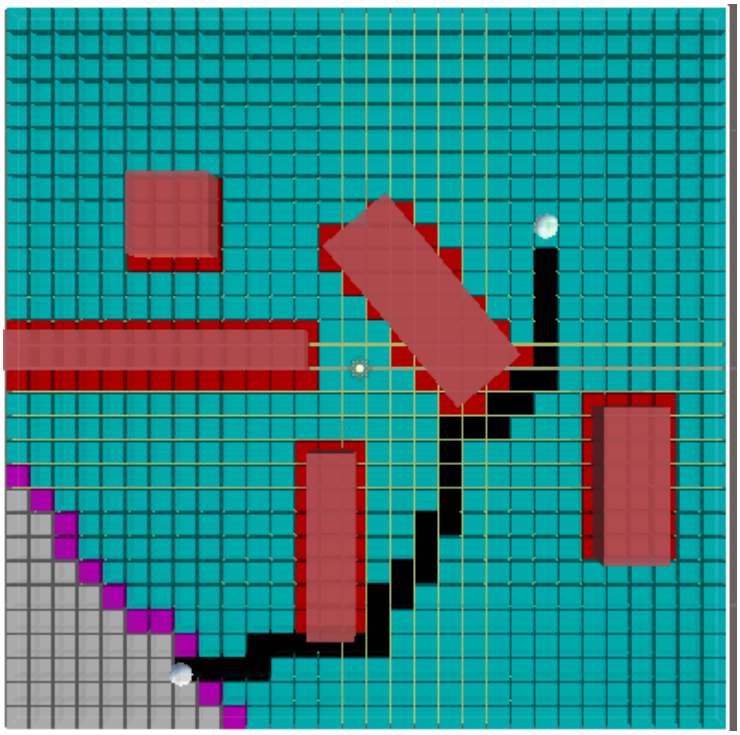
backtracking when it hits a dead end. Meaning that most expanded nodes are also part of the path.

BFS:



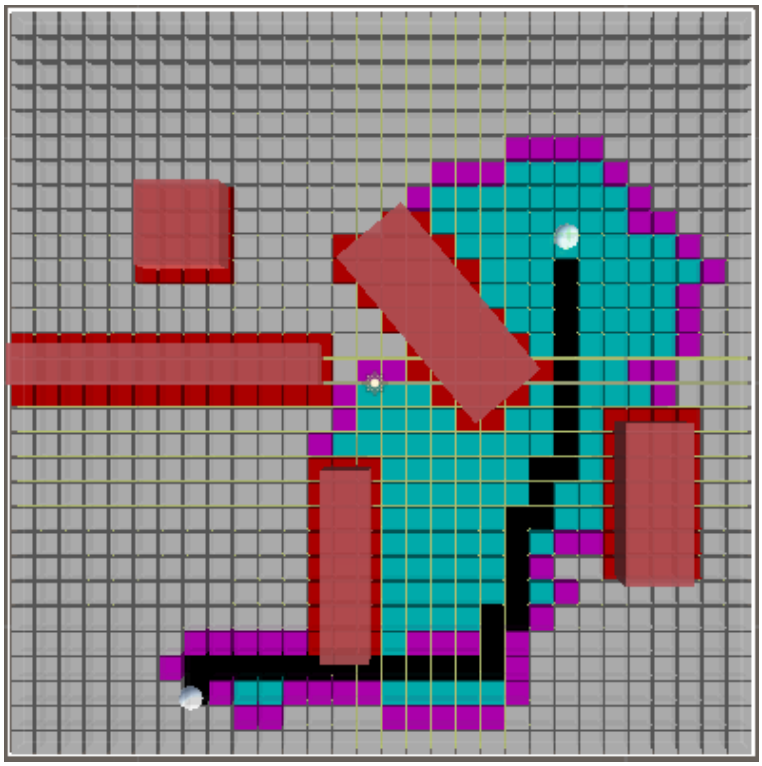
Here is a visual representation of DFS. It is clear from this screenshot that the algorithm was expanding nodes uniformly in all directions, the lower left corner shows that DFS was expanding nodes right until the moment it found the target.

UCS:



As stated earlier, UCS expands nodes in a very similar way to BFS. The path chosen is a little bit different but is still equal in terms of cost. However, I should note that, while the cost of UCS in this example might look similar to that of BFS in terms of the number of nodes that have been expanded, UCS is, in reality, far more expensive than BFS in terms of CPU performance (time complexity) because the implementation of a Priority Queue fringe is more costly than that of a regular Queue. As a matter of fact, our Priority Queue implementation multiplies the original time complexity by a factor of O(log(n)). This is the difference in time complexity between a linear search and a sorting algorithm. The main suspected reason why UCS does not appear to be more costly than BFS in our experiment is that, since the gCost of nodes is directly proportional to their depth in the search tree, whenever a new node is being Enqueued, it will just go to the last index of the queue and not sorting will occur.

A*:



As we can see, thanks to its use of the heuristics function, the A* algorithm can be more direct/informed in its search, saving it from expanding nodes unnecessarily. In terms of number of nodes expanded and memory usage, the A* algorithm is by far the most efficient in our experiment. However, similarly to the UCS algorithm, the A* algorithm suffers from the same performance issues that UCS does because of the Priority Queue fringe implementation. This makes A* almost always better than UCS, but its performance relative

to BFS will depend on the branching factor proper to the nature of the environment and the placement of the obstacles. If the branching factor is too high, then BFS will perform better, but if the search is straight-forward, A* would perform better.

Conclusion

As a result of this experiment, we can conclude that the type of graph search algorithm or, depending on the perspective, the type of fringe implementation has an enormous effect of the performance of a pathfinding program. From the results obtained, we can confidently rule out the DFS approach as the worst approach, as it generates a very unreliable and suboptimal path. Conversely, we can easily deduce that the A* algorithm is the best approach by far in terms of performance in almost all settings, the only exception would be settings where the algorithm would have a very high branching factor.

Special Credits for Sebastian Lague for building the unity environment/project on top of which this experiment was conducted.