## horizontal line



Intro to A. I. Project #1

2/13/2022

**/─**

Aniss Aoudjehane

Al Akhawayn University in Ifrane

Introduction to Artificial Intelligence

School of Science and Engineering

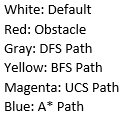
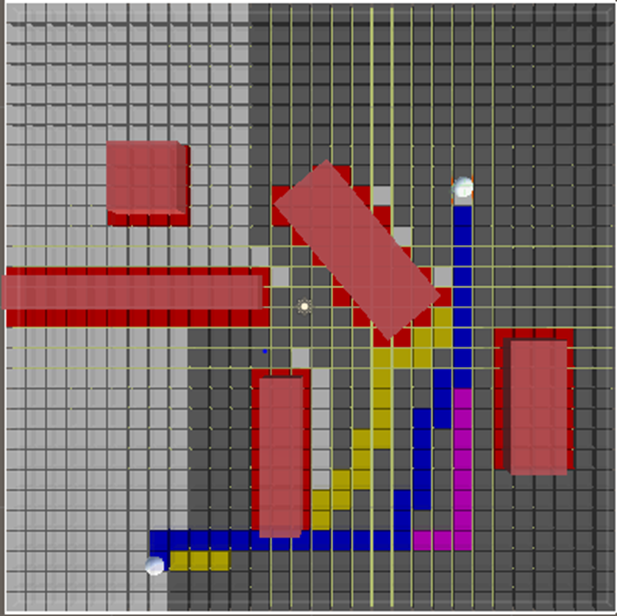
# Overview

During this first project, we built a simulation coded in C# and ran in Unity 3D with a top-down 2D view, in which we made a seeker node find a path to an objective node through a grid using different pathfinding algorithms, such as BFS and DFS.

# Goals

1. Compare the efficiency of different pathfinding algorithms, in terms of time required, as well as memory taken up by the algorithm.
2. Find the best alternative for each different case scenario, if applicable.
3. Get familiar with C# and the use of Unity.

# Specifications

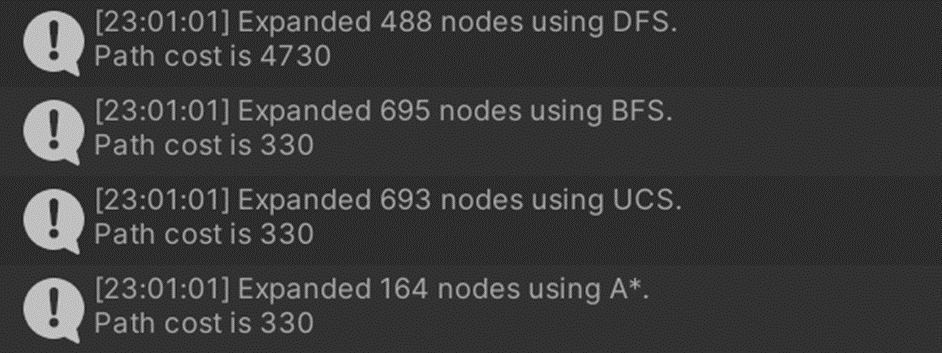


Above is the graphical representation of our simulation. Unfortunately we were not able to make our seeker node travel in a diagonal direction, and instead it can only move vertically or horizontally. We assume the cost of moving in by one node in any direction is 10.

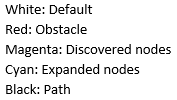
The principal objective of this first project was to try different approaches to pathfinding algorithms, such as Depth-First Search (DFS) which is implemented using a stack fringe and expands the deepest node until it finds a path, Breadth-First Search (BFS) which expands the shallowest node using a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

In order to run these tests and comparisons, we used Unity 2D and VS Code to program our simulations in C#.

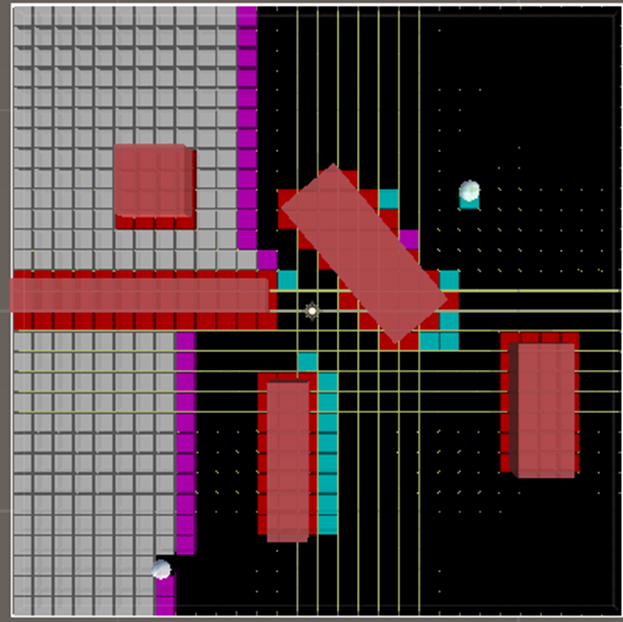
# Performance comparison



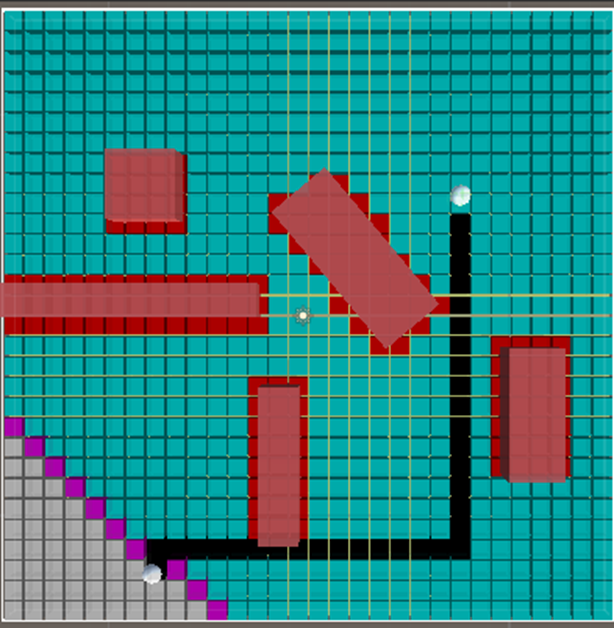
Above is a direct comparison of the memory cost of each of the algorithms.

For the upcoming section, the color coding for the screenshots of our simulation is as follows:  
  
We will be looking at the node expansion pattern for each of the four algorithms, as well as the path discovered.

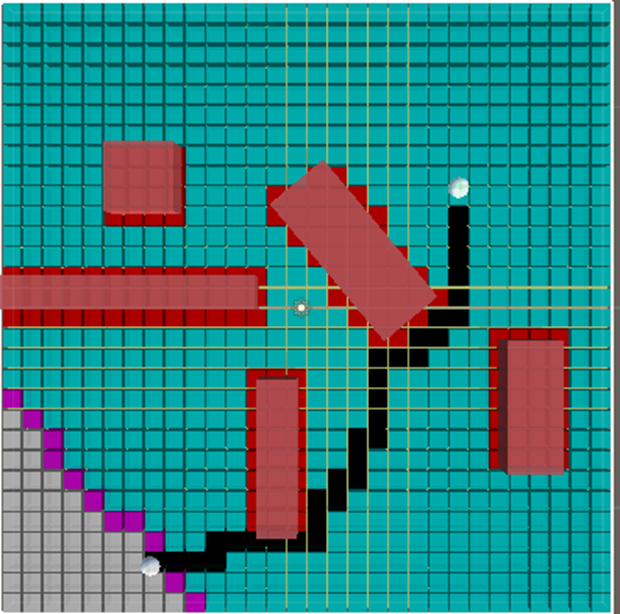
## DFS

  
  
The node expansion of DFS expands the deepest node until it finds a path, and then keeps note of that first path.  
  
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 neighbors. The resulting path may look silly and have higher space cost overall, but the algorithm performed well in terms of time.

## BFS

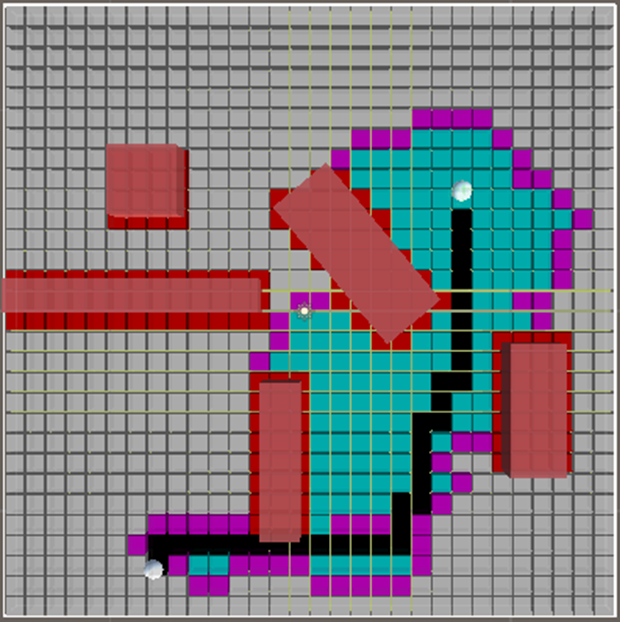
  
  
This screenshots displays that BFS was expanding nodes uniformly until it found the goal node. The path obtained is a lot more optimal than DFS, but the algorithm is more expensive.

## UCS



As one might notice by studying the node expansion pattern from UCS and BFS that the two algorithms operate similarly in their expansion. They expanded a similar number of nodes, and the path cost is the same. However, UCS is a lot more expensive than BFS in terms of time, due to its use of a priority queue. This would be useful if the cost of moving from a node to another was different, but since the costs are consistent, there is no benefit to using UCS over BFS here.

## A\*



Thanks to A\*’s use of different heuristics, the algorithm doesn’t expand unnecessary nodes like its standard brethren. The algorithm has a better idea of where it should search for the goal node, which makes it the most efficient in terms of memory cost. Unfortunately A\* also suffers from the same high time complexity issues as UCS, also due to the priority queue implementation.

This A\* algorithm uses the GetDistance function written by Sebastian Lague, which is the main reason it is so efficient.

In terms of time, the algorithm could perform better than BFS if the branching factor is low.