PATHFINDING WITH HEURISTICS IN 2D SPACE

Implementation using 2D Grid World Representation

FINAL PROJECT REPORT

November 28, 2023

Austin Jetrin Maddison 6481268

ICMA 395 HEURISTIC OPTIMIZATION - INSTRUCTOR: TIPALUCK KRITYAKIERNE

Mahidol University
International College

PATHFINDING WITH HEURISTICS IN 2D SPACE

Implementation using 2D Grid World Representation

FINAL PROJECT REPORT
December 10, 2023
Austin Jetrin Maddison 6481268

ICMA 395 HEURISTIC OPTIMIZATION - INSTRUCTOR: TIPALUCK KRITYAKIERNE

Mahidol University
International College

ACKNOWLEDGMENTS

I want to express my gratitude to Ajarn Tipaluck Krityakierne for her exceptional role in making the Applied Mathematics course an engaging and enriching experience. Before joining, I had a basic interest in the practical side of math but lacked confidence in my own mathematical skills. Opting for this course on a whim turned out to be a wise decision.

Ajarn Tipaluck has introduced me to the world of advanced mathematics in a way that is both accessible and enjoyable. As a first-timer in this domain, her teaching style has been a perfect fit, making the learning process smooth and rewarding.

Despite the inevitable challenges, each week brought a sense of accomplishment as I discovered new facets of the subject. What stands out about Ajarn Tipaluck is her dedication—she puts effort into crafting engaging lessons and shares valuable insights from her experiences. This commitment naturally motivates my peers and I to reciprocate with our best efforts.

Thanks to this course, I not only feel more confident in my own abilities but also genuinely excited to explore mathematics further and apply these skills to areas that interest me. I appreciate Ajarn Tipaluck for making this learning journey both enjoyable and enlightening.

CONTENTS

PROJECT PROPOSAL

I proposed to do a teaching orientated project where I will be covering the topic of pathfinding algorithms and their implementation with the focus on the A* algorithm and a 2D grid world representation. To demonstrate my understanding I would also develop an interactive visualization tool which will be used for testing and teaching the pathfinding algorithms.

PROJECT OVERVIEW

Report

The report will offer an overview of efficient pathfinding using heuristics, covering its motivations, spatial constraints, and exploration of key algorithms within the context of a 2D grid world representation. The report will use the custom interactive application functions as a practical tool, aiding in the visualization and comprehension of the performance and characteristics of these pathfinding algorithms.

Presentation

The final presentation will be an educational talk on pathfinding algorithms and how their implementation works. Then I will follow up the talk with an interactive demo application for everyone to engage with. Why include an interactive demo? Well, I'm a believer that people grasp concepts more effectively when they can actively engage with them. So, after the talk, I've prepared a hands-on demo for everyone to participate in and experience the A* pathfinding algorithm in action. Moreover, this project offers a valuable addition to my personal portfolio. Given i aspire to work in the domain of interactive computer graphics, i see it as an opportunity to elegantly merge this aspect in my final project.

Interactive Visualization Tool

This application will enable users to run and visualize various pathfinding algorithms in real-time. Importantly runtime statistics will be generated to observe varying algorithms performance and behavioral characteristics under different circumstances.

RELATED PUBLICATIONS

The A* algorithm was invented by American scientists Peter Hart, Nils Nilsson, and Bertram Raphael[1]. Their paper "A Formal Basis for the Heuristic Determination of Minimum Cost Paths" in 1968 extends the Dijkstra's algorithm to find the optimal path more path more efficiently. Now the A* algorithm is the defacto for finding the shortest path between positions in space.

Third year student project by Ruoqi He, Chia-Man Hung "Pathfinding In 3d Space - A*, Theta*, Lazy Theta* In Octree Structure"[2] inspired me to create my own real-time pathfinding tool to supplement this teaching orientated project. They used the free Unity realtime engine to implement their 3D pathfinding algorithm. Thus, I would use the Unity as well to implement my tool. Lastly, for reference for each algorithms implementation

I used Hybesis H.urna article "Pathfinding algorithms: the four Pillars."[3] which provides details on efficient data structures and procedures of each algorithm.

[1] *P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.*

[2] *Ruoqi He, Chia-Man Hung, "Pathfinding In 3d Space - A*, Theta*, Lazy Theta* In Octree Structure", https://ascane.github.io/projects/07_pathfinding3d/report.pdf*

[3] *Hybesis H.urna, "Pathfinding algorithms: the four Pillars.", https://medium.com/@urna.hybesis/pathfinding-algorithms-the-four-pillars-1ebad85d4c6b*

INTRODUCTION

PATHFINDING, the task of determining the most cost-effective route between points in a spatial environment, poses a significant challenge when considering various constraints. These constraints may include unwalkable areas or regions with varying traversal costs. While a naive approach, such as Breadth-First Search, exists, it becomes computationally expensive for large search spaces. Fortunately, individuals have developed clever ways to leverage heuristics for efficient pathfinding.

This report provides a overview of efficient pathfinding with heuristics, from its motivations and spatial constraints to a exploration of prominent algorithms in the context of a 2D grid world representation using a custom interactive pathfinding tool. The custom interactive application serves as a practical tool for visualizing and understanding these algorithms performance and pathfinding characteristics.

REPORT OVERVIEW

Motivations For Efficient Pathfinding

Before delving into algorithmic details, it is crucial to explore the motivations behind efficient pathfinding. Applications from robotics and autonomous vehicles navigating dynamic environments to video games traversal in virtual worlds. Understanding these motivations sets the stage for appreciating the significance of performant pathfinding algorithms.

World Representation

The foundation of pathfinding lies in defining the space and spatial constraints. Whether it involves describing unwalkable zones or assigning varying traversal costs to different areas, knowing how to define a world is essential for programming pathfinding algorithms.

Pathfinding Algorithms

This report explores various pathfinding algorithms, including Breadth-First Search, Dijkstra's, Greedy-Best-First Search, and the spotlighted A*.

Testing and Scenario Exploration & Results

Each algorithm's operation within a 2D grid world representation is discussed, shedding light on their strengths and limitations. Special attention is given to the A* algorithm, known for its exceptional performance due to its strategic use of heuristics which aligns with the interests of this course and final project.

Interactive Application

The outcome of this project's exploration is a custom interactive pathfinding application designed to generate and visualize findings. This tool serves as a practical demonstration of algorithms discuss and allows users to interact with and observe the behavior of different algorithms in real-time.

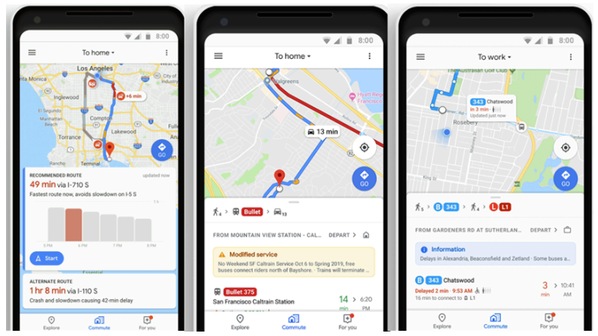
MOTIVATIONS FOR EFFICIENT PATHFINDING

You might be surprised to discover how frequently pathfinding is used in our daily lives, impacting our navigation, logistics, robotics, and even video games.

EXAMPLES

Maps and Navigation

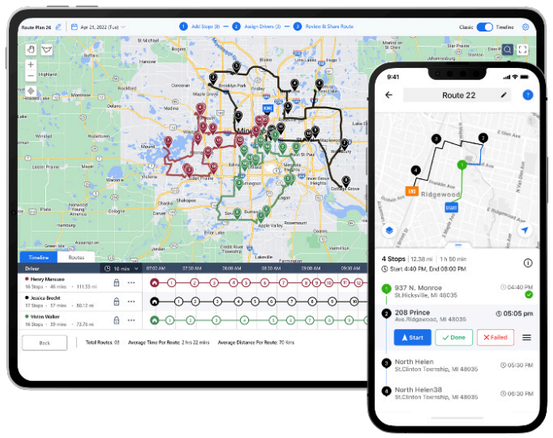
Many of us rely on navigation apps like Google Maps to find the best routes, considering factors such as traffic and mode of transport. Trusting these suggestions relies on efficient pathfinding.



Google Maps

Logistics

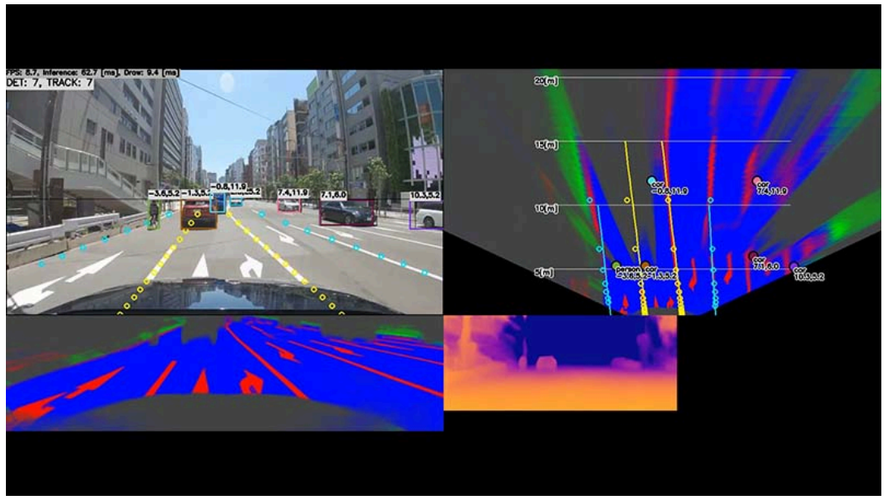
In logistics, pathfinding is vital for optimizing transportation routes and ensuring efficient delivery schedules. The ability to adapt to changing conditions is crucial for effective logistics planning.



Upper, route planning and optimization platform.

Robotics

In the realm of robotics, particularly in self-driving cars, pathfinding is essential for navigating ever-changing environments. Continuous rerouting is necessary as obstacles cannot always be anticipated in advance.



NVDIA DriveWorks

Video Games

Even in the virtual world of video games, pathfinding is important. Agents, like enemy characters, often need to traverse dynamic virtual landscapes. Fast and adaptive pathfinding ensures they reach their destinations efficiently, even as the virtual world or player's position changes.

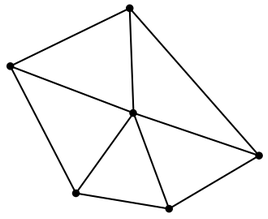
2021 GDC AI Summit - Death Stranding: An AI Postmortem

Conclusion

These examples show the significance of fast and adaptive pathfinding in various domains. Whether in daily navigation, logistics, robotics, or gaming, the ability to quickly find optimal paths is crucial for effectively responding to dynamic and ever-changing conditions.

WORLD REPRESENTATION

To understand space and its limitations, we use graphs. Graphs are like maps made up of spots (nodes) connected by paths (edges). They're handy because they help us show how we can move from one spot to another. In the graph, each spot is a node, and the paths between them can represent different things.



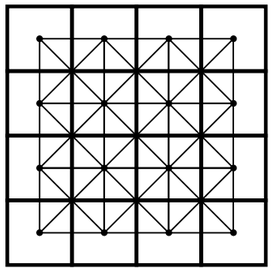
For instance, we can use edges to show where there are obstacles – spots you can't walk through. Or, we can make some paths cost more by giving them a weight. This weight is like saying it's a bit more expensive to go that way. Later, when we're figuring out the best route with pathfinding algorithms, these weights come into play in our calculations.

So, graphs are like a language that helps us describe spaces, their limitations, and so we can navigate them efficiently.

2-DIMENSIONAL

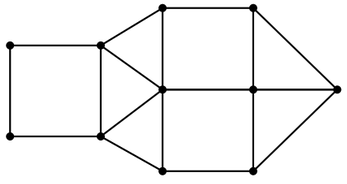
2D Cell Grid

Nodes: Each square, or cell, in the grid is a node.
Edges: The edges here represent the connections between neighboring cells. They define the relationships between adjacent spaces.



2D Polygonal Mesh (“Navigation Mesh”)

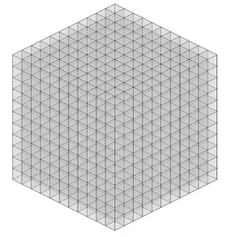
Nodes: In this case, nodes are the vertices of the shapes in the mesh, like the corners of a connect-the-dots drawing.
Edges: Edges connect the vertices of the shapes, forming the interconnected polygons. These polygon faces show where we can move.



3-DIMENSIONAL

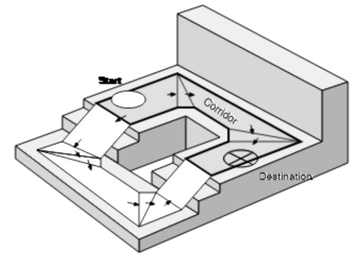
3D Voxel Grid

Nodes: Each cube, or voxel, in the 3D Voxel Grid is a node. They're like building blocks in a digital cube structure.
Edges: Edges here represent the connections between neighboring cubes. They define how the cubes are stacked and connected in the 3D space.



3D Polygonal Mesh (“Navigation Mesh”)

Nodes: Similar to the 2D case, nodes are the vertices of the polygons in the mesh, like the corners of a 3D shape.
Edges: Edges connect the vertices of the polygons, forming interconnected 3D shapes. These polygon faces define where we can navigate within the 3D space.



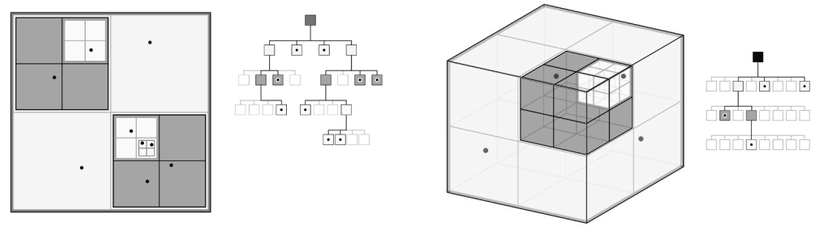
Unity Game Engine Manual, Inner Workings of the Navigation System.

ACCELERATION STRUCTURES

Quadtrees and Octrees for 2D and 3D Grid Representations

Despite this project will be using a 2D grid world representation it is worth mentioning that acceleration structures do exist for querying cells. To efficiently organize and navigate a 2D grid, quadtrees come into play. Quadtrees divide the grid into quadrants, providing a hierarchical structure that simplifies spatial queries and accelerates computational processes. This organization helps identify which cells are neighbors, making navigation smoother.

In the 3D counterpart, this hierarchical structure divides the space into octants, offering an efficient means to manage and analyze the volumetric data in three dimensions. Just like in the 2D case, octants define which cubes are neighbors, aiding in efficient navigation and analysis of the 3D space.



On the left is a quadtree and right is an octree. Apple Gameplay Kit, GKOctree.

PATHFINDING ALGORITHMS

The pathfinding algorithms discussed in this report can be classified into two main categories: informed and uninformed. An informed algorithm is one that systematically explores all relevant nodes in the search space to determine the global best path. On the other hand, an uninformed algorithm does not provide a guarantee of finding the global best path, as it may not explore alternative promising paths during the search process.

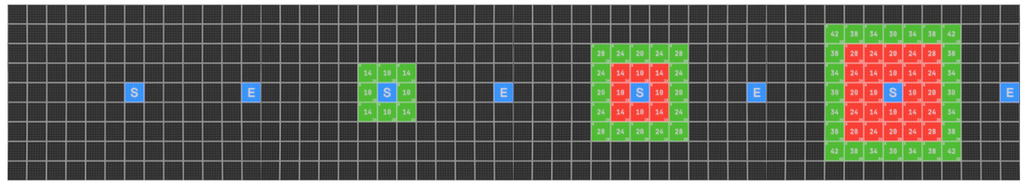
INFORMED
BREADTH-FIRST-SEARCH (BFS)

Finds all the nodes at the current frontier before moving on to nodes at the next frontier.

Psuedocode

```
function BFS(graph, start, end)
    queue := [start]
    visited := set()

    while queue is not empty
        current := dequeue(queue)

        if current is end
            return success

        for each neighbor of current
            if neighbor is not in visited
                enqueue(queue, neighbor)
                add neighbor to visited

    return failure
```

Search Behavior



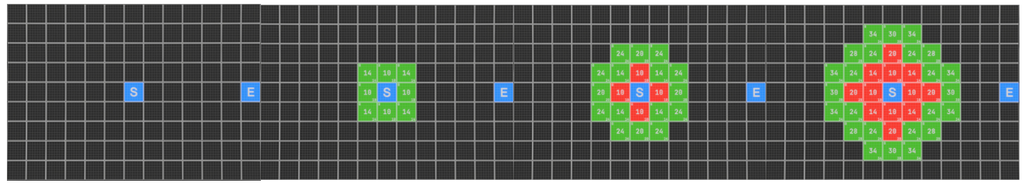
INFORMED
DIJKSTRA'S

Combines Dijkstra's algorithm with Greedy-Best-First-Search heuristic function.

Psuedocode

```
function Dijkstra(end, start)
    distance := map with all values set to infinity
    distance[start] := 0
    priorityQueue := {start}

    while priorityQueue is not empty
        current := node in priorityQueue with the smallest distance

        if current is end
            return reconstructPath(current)

        for each neighbor of current
            tentativeDistance := distance[current] + weight(current, neighbor)
            if tentativeDistance < distance[neighbor]
                distance[neighbor] := tentativeDistance
                add neighbor to priorityQueue

    return distance
```

Search Behavior



UNINFORMED + HEURISTIC
GREEDY-BEST-FIRST-SEARCH

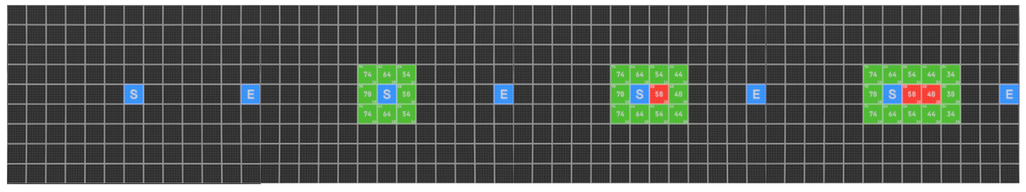
Always chooses the node smallest cost according to the heuristic function.

Psuedocode

```
function GreedyBestFirstSearch(start, end)
    priorityQueue := {start}
    visited := set()

    while priorityQueue is not empty
        current := node in priorityQueue with the lowest heuristic value

        if current is end
            return success

        remove current from priorityQueue
        add current to visited

        for each neighbor of current
            if neighbor is not in visited
                add neighbor to priorityQueue

    return failure
```

$f(n) = n's\ estimated\ distance\ from\ end\ node$

Search Behavior



INFORMED + HEURISTIC
A*

Combines Dijkstra's algorithm with Greedy-Best-First-Search heuristic function.

Psuedocode (Modified Greedy Best First Search)

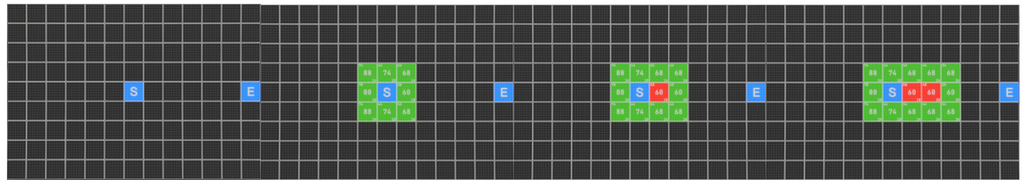
```
for each neighbor in neighbors(current)
    if neighbor is in closedSet
        continue

    tentative_gScore := gScore[current] + distance(current, neighbor)

    if neighbor is not in openSet or tentative_gScore < gScore[neighbor]
        gScore[neighbor] := tentative_gScore
        fScore[neighbor] := gScore[neighbor] + heuristic(neighbor, end)
        if neighbor is not in openSet
            add neighbor to openSet
            set cameFrom[neighbor] to current
```

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

$g(n) = n's\ distance\ from\ start\ node$

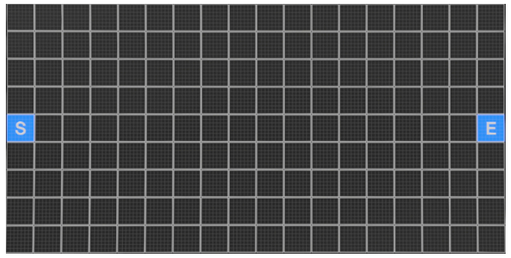
$h(n) = n's\ estimated\ distance\ from\ end\ node$

Search Behavior



TESTING AND SCENARIO EXPLORATION & RESULTS

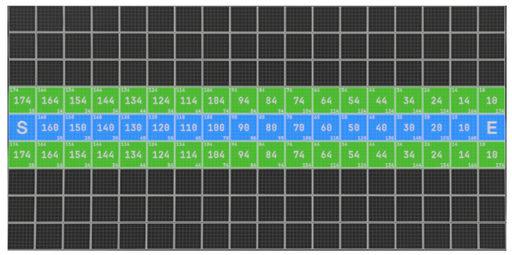
SCENARIO 1



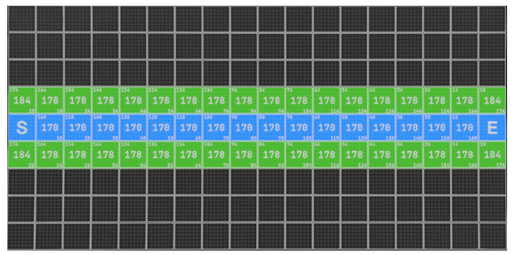
Starting Grid

Breadth-First-Search

Dijkstra's



Greedy-Best-First-Search



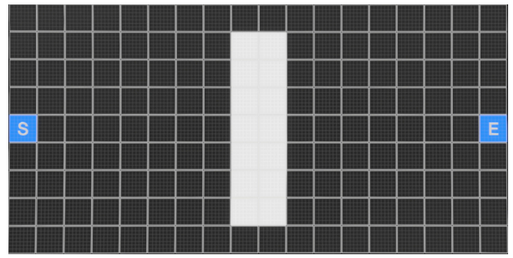
A*

Results

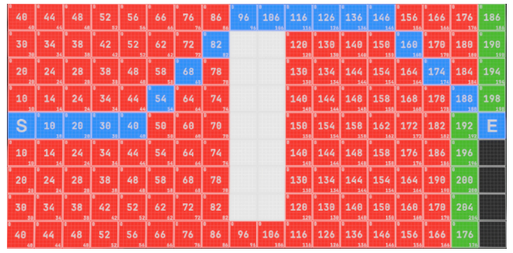
Algorithm	Iterations	Distances	Ratios
A*	18	170	0.32
Breadth First Search	145	170	0.99
Dijkstra's	141	170	0.93
Greedy Best First Search	18	170	0.32

All algorithms find the best possible path. It's clear that the algorithms that don't deploy any heuristics waste a lot of work on evaluating nodes that are not promising. While the algorithms that do you use heuristics cone directly and swiftly towards the end node.

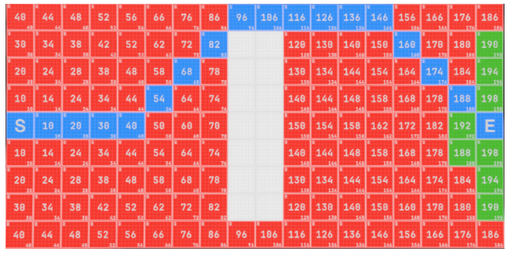
SCENARIO 2



Starting Grid



Breadth-First-Search



Dijkstra's

Greedy-Best-First-Search

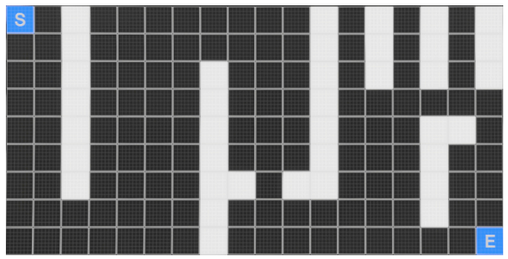
A*

Results

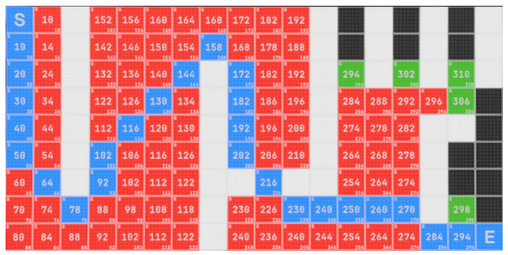
Algorithm	Iterations	Distances	Ratios
A*	67	202	0.62
Breadth First Search	134	202	0.94
Dijkstra's	139	202	0.9
Greedy Best First Search	23	214	0.4

In this situation, not all algorithms succeed in finding the optimal path. Greedy First Search lacks motivation to explore and tends to ignore potentially better paths. This behavior is noticeable as Greedy Best First Search seems to closely follow obstacles, eagerly trying to get as close as possible to the endpoint. In contrast, A* handles obstacles differently. Because of its informative nature, it considers alternative nodes as promising routes when approaching obstacles. When the endpoint is clearly visible, A* swiftly converges towards it. This approach not only results in discovering the best path but also avoids the excess work that BFS or Dijkstra's algorithms might undertake.

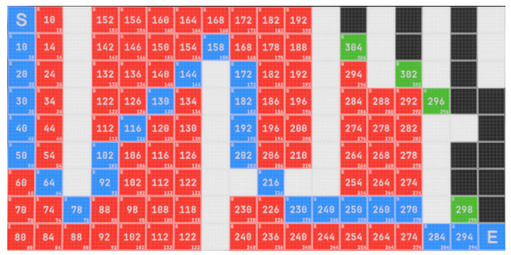
SCENARIO 3



Starting Grid



Breadth-First-Search



Dijkstra's

Greedy-Best-First-Search

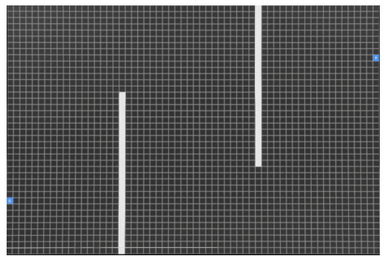
A*

Results

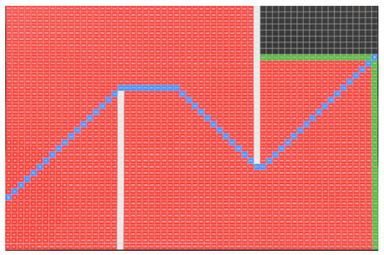
Algorithm	Iterations	Distances	Ratios
A*	79	304	0.57
Breadth First Search	106	304	0.72
Dijkstra's	106	304	0.67
Greedy Best First Search	35	344	0.48

In this instance, the presence of numerous convex and concave obstacles significantly influences the performance of heuristic algorithms. Once again, it's evident that, with the exception of Greedy Best First Search, all algorithms fall short of identifying the optimal path to the endpoint. This limitation stems from the previously discussed wall-hugging behavior. Despite its inability to find the best path, Greedy Best First Search manages to discover a viable path in less than half the iterations A* requires for optimal pathfinding. This observation might prompt the consideration of integrating Greedy Best First Search with other pathfinding algorithms to potentially expedite the initial phase of pathfinding. Indeed, in practical applications, a hybrid or "Frankenstein" pathfinding algorithm could be employed and strategically scheduled to enhance the efficiency of finding a feasible path, particularly in expansive search spaces.

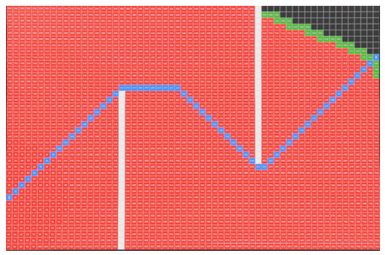
SCENARIO 4



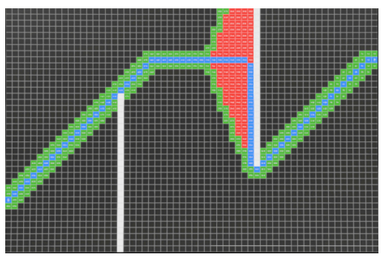
Starting Grid



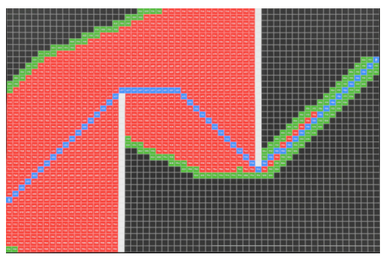
Breadth-First-Search



Dijkstra's



Greedy-Best-First-Search



A*

Results

Algorithm	Iterations	Distances	Ratios
A*	1135	786	0.53
Breadth First Search	2146	786	0.94
Dijkstra's	2238	786	0.94
Greedy Best First Search	168	922	0.16

In a broader search context, the efficacy of heuristic algorithms becomes more pronounced as the search space expands. This phenomenon is logical since the heuristic guides the search toward promising areas, and a path typically traverses only a small fraction of the entire search space. Consequently, many nodes can be presumed unnecessary for evaluation, resulting in time savings during the search process. Building upon the earlier observation, Greedy Best First Search exemplifies that, although it may not achieve the optimal path, it can derive a path in significantly fewer iterations compared to its counterparts.

CONCLUSION

To sum up, exploring different pathfinding algorithms in various situations helps us understand their strengths and weaknesses. Although Greedy Best First Search may not always find the absolute best path, its quick and efficient discovery of workable paths, especially in large areas, highlights its usefulness. A* demonstrates the advantage of heuristics by smartly focusing on promising areas, leading to significant time savings in navigating complex landscapes.

Moreover, the adaptability of algorithms, hinted at by the potential combination of Greedy Best First Search with other methods in a "Frankenstein" algorithm, shows how pathfinding solutions can be flexible. In my project presentation, I shared an example from the video game "Death Stranding," where developers used various tricks to make their pathfinding algorithms more effective while still ensuring a viable path. They combined A* with different ways to cut down on unnecessary searches.

As we explore pathfinding algorithms, finding the right balance between achieving the best outcome and doing so efficiently becomes a crucial consideration. This emphasizes the importance of tailoring approaches based on the specific characteristics of the problem at hand.

INTERACTIVE APPLICATION

OVERVIEW

I propose to do a teaching orientated project where I will be covering the topic of pathfinding algorithms and their implementation with the focus on A* pathfinding algorithm using a custom interactive pathfinding application.

FEATURES

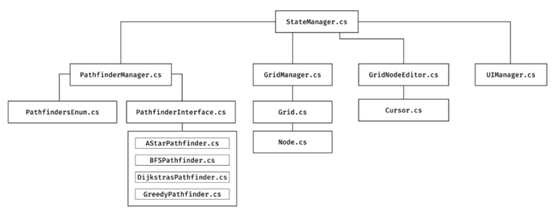
- Set start and end nodes.
- Paint obstacles that occlude the end node from the start node.
- Set grid size of 2D world grid.
- Be able to select different algorithms:
  - A* Algorithm
  - Dijkstra's Algorithm
  - Breadth First Search Algorithm
  - Greedy-Best-First-Search Algorithm
- Visualize cost objective and heuristic values that are being tracked for every node on
- the 2D world grid.
- Visualize global statistics of the algorithms progress:
  - Current iteration.
  - Current opened nodes count.
  - Current Closed nodes count.
  - Current best path distance.
  - Total visited nodes to total search space ratio.
  - Realtime table and or graphical plots.
- Incrementally step through iterations in the algorithm.
- Intuitive navigation using mouse and keyboard.

DEVELOPMENT ENVIRONMENT

The development environment used to develop this project's interactive application will be C# .NET with the Unity game engine.

APPLICATION CLASS STRUCTURE

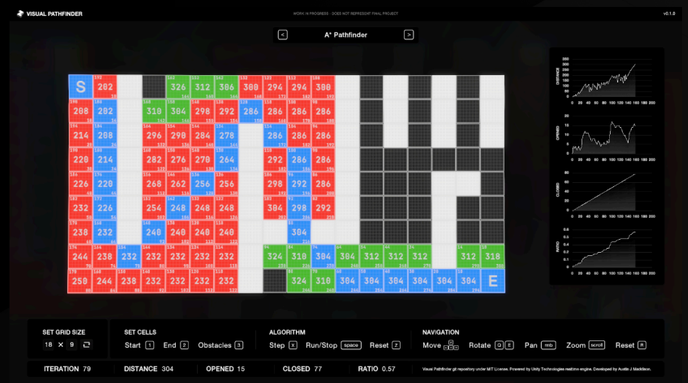
The composition of the program was daunting at first as I never written a program with this much moving parts. It was very important to keep the program very modular and reduce coupling as much as possible. This is important because it needs to be easy to extend different pathfinding algorithms. Even though the algorithms seem very simple in their implementation managing them in interactive loop was tricky due to the many edge cases the user can proceed in. The State Manager acts as the main controller for all the mechanisms below.



VISUAL PATHFINDER™

Visual Pathfinder is an interactive 2D pathfinding visualization tool. Enables users to author and run scenarios against common pathfinding algorithms *(A*, Dijkstra's, Greedy-Best-First Search, Breadth-First Search*). Progress data of pathfinding algorithms is generated in realtime.

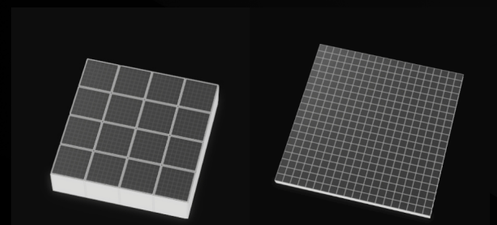
Screenshot



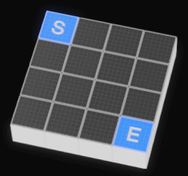
Features

World Grid Editor

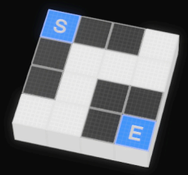
Resize Grid Size



Set Start and End



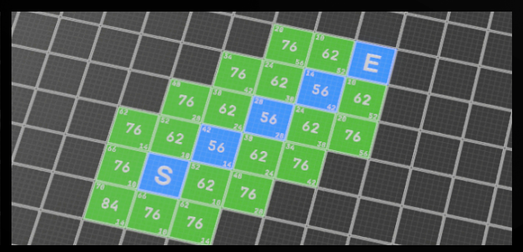
Paint Obstacles



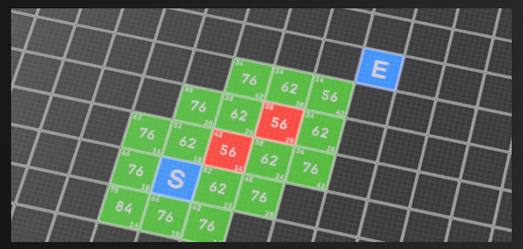
Running Pathfinder

Step

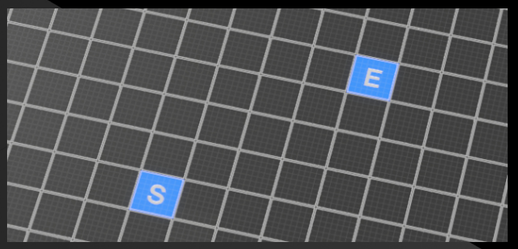
Run



Pause



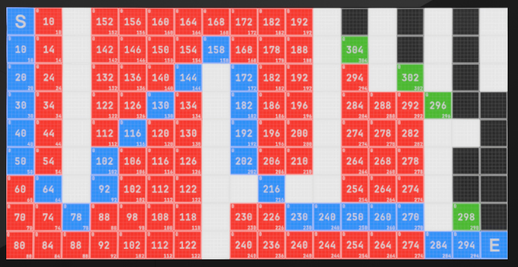
Reset



Algorithms

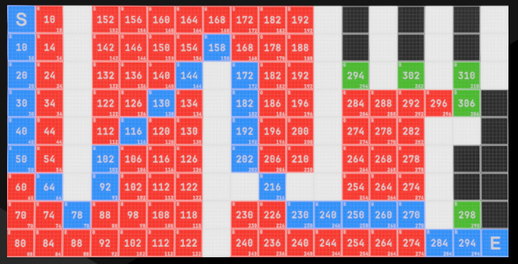
A*

Dijkstra's

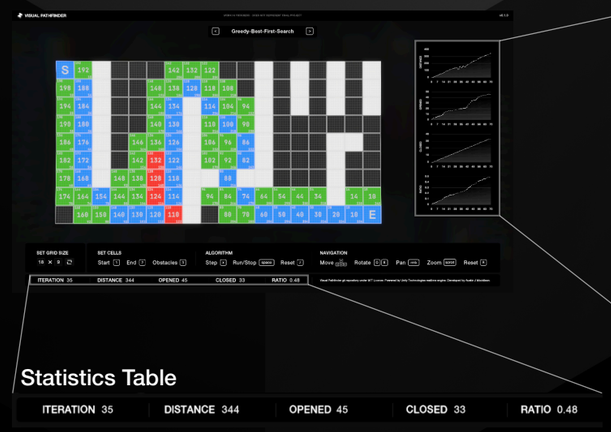


Greedy-Best-First Search

Breadth-First Search



Progress Data



Statistics Charts