

Assignment 1 Report

COS30019 – Introduction to AI

The Son Ngo

Student ID: 103827804

[I.Instructions 2](#_Toc164482895)

[II. Introductions 4](#_Toc164482896)

[Glossary 4](#_Toc164482897)

[III. Search algorithms 5](#_Toc164482898)

[IV. Implementation 8](#_Toc164482899)

[V. Testing 9](#_Toc164482900)

[VI. Features and additional information 13](#_Toc164482901)

[VII. Research 13](#_Toc164482902)

[1) Research problem 13](#_Toc164482903)

[2) Literature review 14](#_Toc164482904)

[3) Objectives 14](#_Toc164482905)

[4) Methodology 15](#_Toc164482906)

[VIII. Conclusion 17](#_Toc164482907)

[IX. Resources 18](#_Toc164482908)

[X. References 18](#_Toc164482909)

# Instructions

Below are the steps that the reader can use to open the program and operate it:

* Unzip the folder and open a Terminal (for MacOS) or Command Prompt (for WinOS)
* Make sure you have installed Python and pygame on your computer. These are the pre-requisite tools to run this program. If you do not have either one of these tools, do the following:
  + If there is no Python: you can download it via this website

<https://www.python.org>

Check whether Python has been installed successfully by typing “python –version” on the Terminal. If the output is the version of your newly installed Python, you have installed successfully

* + If there is no “pygame”: with Python ready, run “pip install pygame” on the Terminal. Check the successful installation by typing “pip show pygame”. If the output shows the version together with other information about “pygame", you have installed it successfully
* Run “python search.py <filename> <method>”. Where <filename> is the directory of the data file you want to test the program on, and <method> is the search algorithm you want to implement. Make sure to type the command in the correct format and order.
* The result will be shown on the Terminal. It includes the coordinates of the goal found, the total number of states expanded by the program and the path from the starting point to the goal in a list of directions.
* In addition to the Terminal outputs, a GUI made by “pygame” will pop up. Figure 1 shows the outlook of the GUI

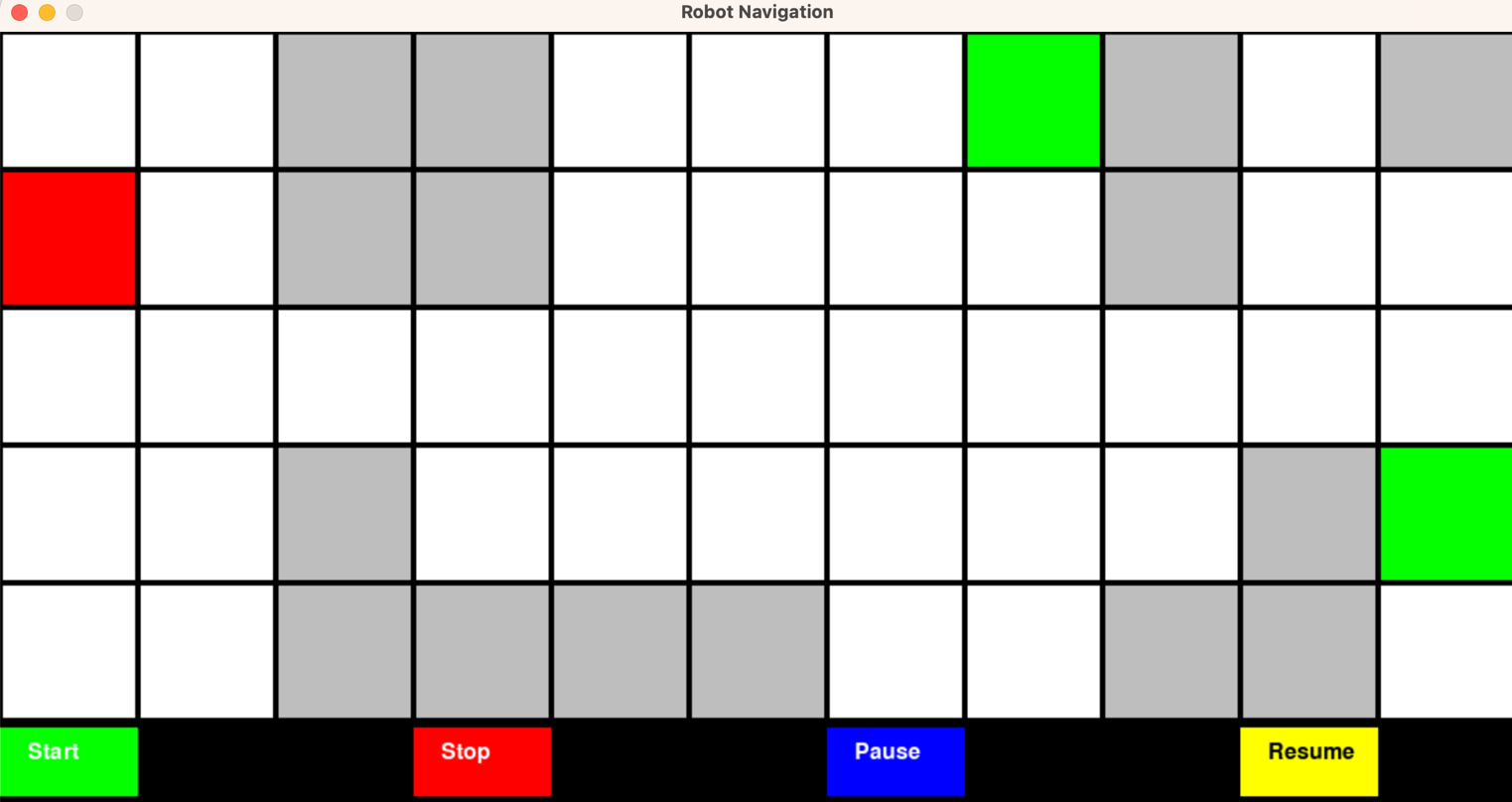


Fig 1. Graphic User Interface

* To use the GUI, you have to control it using the four buttons at the bottom. To start the search, press the green button.
* If you want to reset the search while it is running, press the red button. If you want to pause the program to see the recent states expanded in detail, press the blue button. Subsequently, you can press the yellow button to continue the program.
* If you want to escape the UI, simply press the X button in the top-left corner of the window. Alternatively, press Ctrl+C on the terminal to escape.

The GUI will be explained in more detail in the Research section.

# Introductions

Search algorithms are prevalent in other environments in which the agent is required to explore and seek solutions, thus maximising its performance evaluated by its expected outcomes. Take the Mars Rover as an example of a search **agent**, which has the **goal** of collecting dirt samples on this red planet. Mars is an extreme **environment**, with a plethora of disasters that can hinder the process of collecting dirt, such as sand storms, cosmic rays, heatwaves, etc. To make the rover overcome these challenges and maximise its **performance**, which is based on how much dirt it can collect, NASA engineers need to work on integrating search algorithms for the rover. The result is a Rover that understands its surroundings and is capable of navigating through its environment to find places with fewer disasters.

Search algorithms are extremely powerful in navigation problems. Whether the task involves finding the shortest path in a road network, navigating the complexities of a game environment, or even managing network traffic, these problems can universally be modelled using s**earch trees** and **graphs**. Due to the ability to represent any environment and systematical ways, search trees and graphs brought flexibility and scalability to any programs made by humans.

In the context of this project, we have an environment represented in square grids. The environment consists of a starting state, many goal states, walls and normal states. This program aims to visualise how search algorithms work on computer programs, specifically in the context of navigation problems. The outcome of the program is the prototype for more complex navigation applications, such as GPS or bots in video games.

## Glossary

* Agent: any object that is capable of automatically doing tasks given by users or other programs.
* Goal: the state in which the agent is expected to reach.
* Node: Another way to call a state.
* Completeness: an algorithm is completed when it always finds out the path if one exists.
* Optimality: an algorithm is optimal if it always finds out the shortest path to the goal.
* Time complexity: how much time the algorithm takes.
* Space complexity: how much memory the algorithm takes.
* Environment: the surrounding conditions of the world in which the agent is operating.
* Performance: the performance of an agent is evaluated based on its expected outcomes rather than the rationality of its actions.
* Search tree: a tree-based data structure allowing dynamic and efficient operations of data. It aims to reduce time complexity for data handling. Each node of the tree contains keys and values to its child node.
* Graph: a non-linear data structure consisting of nodes (or vertices) and edges connecting pairs of nodes.

# Search algorithms

In this project, the program is required to implement a total of 6 algorithms, 3 of them are informed search and the other is uninformed search. The section below will explain in detail what these algorithms are.

1. Breadth-First Search

Explanation: Breadth-first search is an algorithm that explores the graph layer-by-layer. Starting from the root node, it explores all neighbouring nodes at the present depth before moving on to nodes at the next depth level. It uses a Queue data structure to handle the node explored at its frontier, according to the First In First Out rule (FIFO). This means the first node to be explored will be the first to pop out, given that it is not the goal.

Discussion: BFS is complete, meaning it will find a solution if one exists, and it is optimal for finding the shortest path in an unweighted graph. However, it requires significant memory since it stores all nodes of the current depth level before moving to the next level. For example, in the first test (Table 1), the BFS algorithm expanded about 33 nodes, while DFS only expanded, more than doubled that of IDDFS (14 nodes).

1. Depth-First Search

Explanation: Depth-First Search explores as far as possible along each branch before backtracking. Starting from a chosen root node, it explores each branch before backtracking to previous nodes to explore new branches. It uses a Stack data structure to store the frontier nodes. Stack works based on the LIFO rule, so the last node inserted will be the first one to be expanded further.

Discussion: DFS is less memory-intensive than BFS because it stores only a single path from the root node along the branches being explored. However, it is neither necessarily optimal nor complete, as it can get trapped in loops or infinite branches in graphs without careful implementation (like using recursion with termination checks). This is demonstrated by the node count between DFS and BFS in the first test: 28 and 33 respectively. However, DFS reach a further goal than BFS ((10,3) instead of (7,0)), indicating its inoptimality.

1. Greedy Best First Search

Explanation: This algorithm selects which node to explore next according to a specific heuristic that estimates how close the end of the path is to the current node. It prioritizes nodes that appear to be closest to the goal, based purely on the heuristic.

Discussion: Greedy Best First Search can be faster than BFS or DFS because it uses a heuristic to guide the search towards the goal. However, it is not guaranteed to find the shortest path and can be incomplete, particularly in complex graphs with many obstacles. To prove this, we can look at the result of test 1. Greedy Best First Search ends the search early, although there is a path to the goal. The reason for this is that GBFS does not take into account the node with a higher cost which is capable of leading the agent to the goals.

1. A-star search

Explanation: A\* is a refinement of both BFS and Greedy Best First Search, utilizing both a heuristic function (like Greedy Best First) and a cost function from the start node (like BFS). The algorithm evaluates nodes by combining the cost to reach the node and the estimated cost from the node to the goal.

Discussion: A\* is both complete and optimal, as long as the heuristic function is admissible (never overestimates the true cost). It is widely used in pathfinding and graph traversal problems due to its effectiveness and efficiency in practice. If we look at the results of test 1, we can see that AS can find the closest goal, which is (7,0). The node count for it is also really low: 25, only higher than that of IDDFS

1. Iterative Deepening Depth-First Search

Explanation: IDDFS combines the depth-first search's space-efficiency and breadth-first search's completeness. It performs successive runs of depth-first search, each time increasing the depth limit until it finds the goal.

Discussion: IDDFS is useful when the depth of the solution is unknown. It combines the benefits of both DFS and BFS and guarantees to find the shortest path in an unweighted graph. Each iteration expands the memory footprint incrementally and only requires as much memory as DFS. This is demonstrated by its lowest node count of 14, and the capability of finding the closest (7,0) goal.

1. Beam search

Explanation: Beam Search is a heuristic search algorithm that explores a graph by expanding the most promising nodes in a limited set. It is similar to BFS but uses a heuristic function to limit the number of nodes stored at each level to a fixed small number, known as the beam width.

Discussion: Beam Search is useful in finding a good enough solution quickly with limited memory. However, it is not complete or optimal because it only explores a fraction of the graph. The quality of Beam Search highly depends on the number of nodes it takes at each set. This approach is often used in areas like natural language processing where quick, reasonably good solutions are preferable. The data from the test shows that beam search has the second-highest node count, just fewer than that of BFS.

# Implementation

The program has a total of 22 classes, which have a singleton design pattern for the GUI. The image below shows how these classes are related to each other.

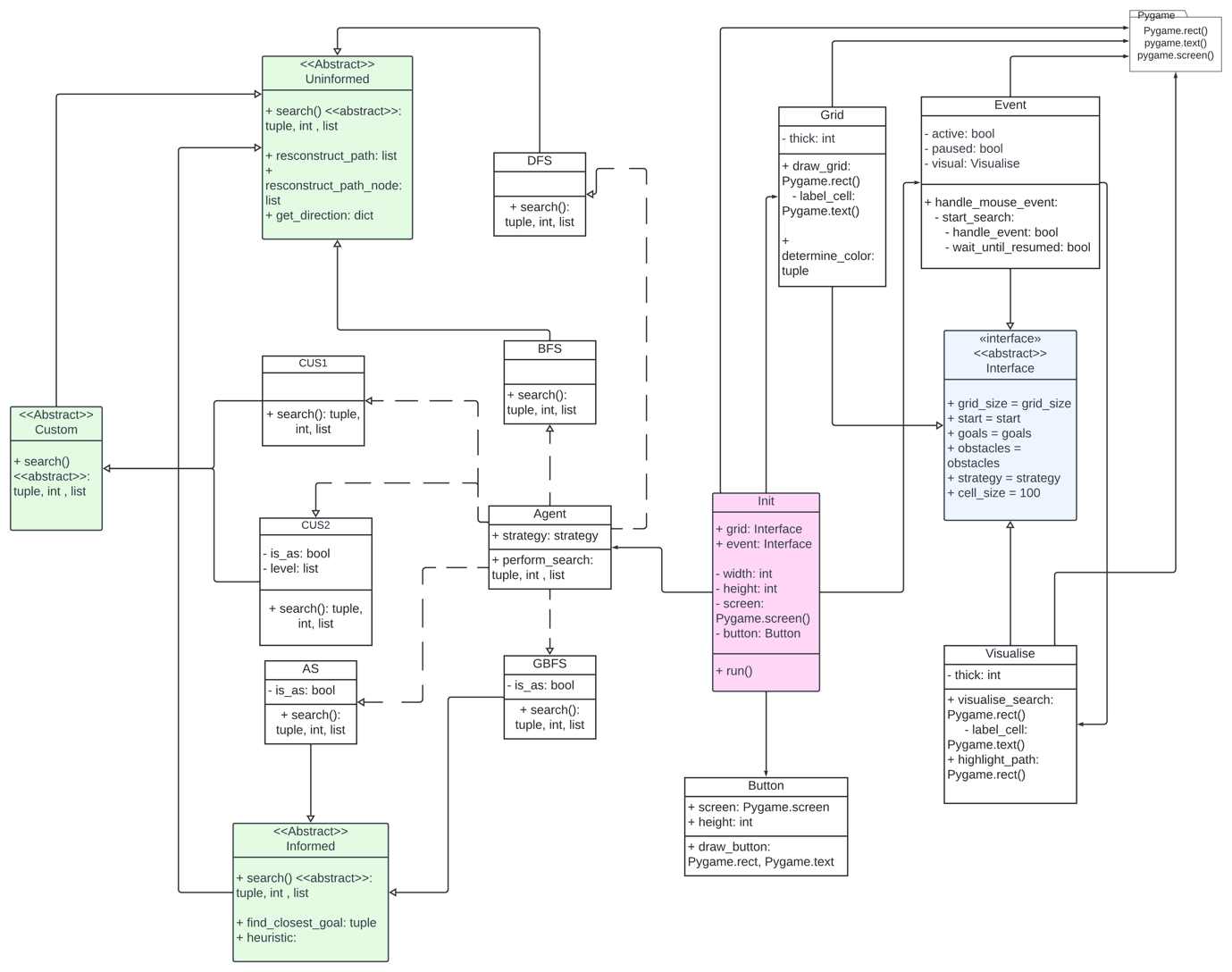


Fig 2. UML diagram of program

The Init class is the Singleton instance which will be declared in the main method. This instance initialises the main window and assembles other components (Button, Grid, Event) to create the main GUI. The role of Pygame library is extremely important in this section, as it provides the necessary pre-defined methods and objects to create the GUI (Rectangle, window screen, etc). It is also important to note the use of abstract class Interface. It acts as the placeholder, with a defined blue-print for other GUI components, thus enable scalability and cohesion.

On the left-hand size of the UML diagram, the Agent class demonstrates polymorphism by defining a property called strategy. This property will then be handled in main method and overridden by 6 search algorithms. To maintain a low coupling for the program, 3 abstract classes are introduced, each inherited by 2 algorithms. Because of having many methods of constructing path and map directions, which are required by every algorithm, Uninformed class is inherited by both 2 other abstract classes. In terms of Informed class, it has shown demonstration of encapsulation, with the 2 concrete methods only applied for the informed search methods.

To see the diagram clearer, click this link: <https://lucid.app/lucidchart/12e17975-787b-4916-943f-9180d0e3359f/edit?viewport_loc=-3121%2C-1045%2C10111%2C5732%2CHWEp-vi-RSFO&invitationId=inv_41fb11dd-3b0a-49bd-8fc8-787a2df24823>

# Testing

|  |  |
| --- | --- |
| Test | Result |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table 1. Test results

The above table shows 10 tests that I have made for the program. All of the environment can be processed by the program and give the correct outputs. If you do not understand the result of the 6th test, the starting point lies in the goal. In terms of the 7th test, the starting point is outside of the grid. For the 8th test, it is a transposed version of the first test. About the last test, I intentionally make the file in the wrong format. Instead of raising an error, the program will tell the user to check the format again.

# Features and additional information

Generally speaking, the program has the following features:

* Read the environment text file
* Implement the specified method in the command prompt
* Output the results of the search on both the command prompt and the GUI
* The GUI can play background music.
* The GUI has the capability of visualizing the internal workflow of algorithms.
* After finishing the search, the GUI highlights the path found to the goal in orange and closes the window automatically after 5 seconds.
* Users can also interact with it, such as playing, pausing, resetting or resuming the visualisation.

The following features are missing:

* Separated application for GUI with icon
* Detailed visualization for beam search (the way it chooses promising nodes)
* Menu stage to automate the environment testing

Please note that this program does not have a separate application to launch the GUI. Alternatively, the user will see the GUI launched after typing the “python search.py <filename> <method>”. A detailed explanation will be discussed shortly.

# Research

## Research problem

The search algorithm is an essential field of Computer Science, thus every Software Engineer, System Engineer, Data/ML scientist or anyone who works with computers is required to have a solid understanding of search patterns and related data structures. However, for novices or Computer Science students with little experience with programming, interpreting complex theories about data structures and algorithms, especially those regarding searching, has never been easy. For this reason, universities and colleges should change their pedagogy and re-design the courses’ structure, thus stemming the enthusiasm in students by helping them absorb abstract and complex theories.

## Literature review

Fortunately, learning about DSA through visualisation has been prevalent in the recent years. Many pieces of research showed that interpreting DSA through interactive video games or web-based animations leads to better learning capabilities and experience. According to the research (Foteini Grivokostopoulou and Ioannis Hatzilygeroudis, 2013), the web-based educational system with interactive examples, exercises and feedback about AI gathers positive results and feedback from students. In another research made by Foteini Grivokostopoulou, Isidoros Perikos and Ioannis Hatzilygeroudis (2016), results show that arcade video games like Pacman also effectively enhance students' engagement and understanding of AI search algorithms in an entertaining, interactive, and motivating way. Visual-based learning is also suitable for understanding advanced mathematical concepts. The Liu Ying-hua (2012)‘s research introduces a GUI to teach mathematical experiments to students with specific examples. This GUI is based on Matlab – a programming language for complex numeric computing and algorithms.

## Objectives

Realising the importance of visual-based implementation for search algorithms, this project aims to create a program that does not stop at command line control and output. Alternatively, this program expands the User Interface by utilizing “pygame” – a Python library that specialises in making interactive 2D video games. “pygame” itself has many pre-defined methods and objects to create a visually appealing video game, so it is a suitable library to create an interactive GUI for DSA learning. Having this upgraded UI in hand, the user will see the beauty in the search algorithms, as well as their power in automating the lives of humans. By using different colours for states and decorative buttons, the UI might stem the enthusiasm of users and urge them to delve into the realm of Computer Science.

## Methodology

As shown in Figure 1, before starting the visualization of the chosen algorithm, the user has the option to observe the environment first. The buttons are drawn with different colours so that they look more visually appealing. These buttons are put at the bottom intentionally because otherwise, it might interrupt the representation of other objects in GUI. This collection of 4 buttons is adequate for the user to gain control over the GUI and opt to slow down the visualization.

To visualize informed search algorithms, each state in UI is tagged with heuristic values. For A-start search, each state is noted with both the cost to reach the goal from the recent state and the cost to reach the recent state from the start point. On the other hand, other informed search algorithms have their states tagged with only one heuristic value, which is the cost from the recent state to the goal. By providing additional information about the heuristic values, users can interpret more precisely the mechanisms behind each algorithm.

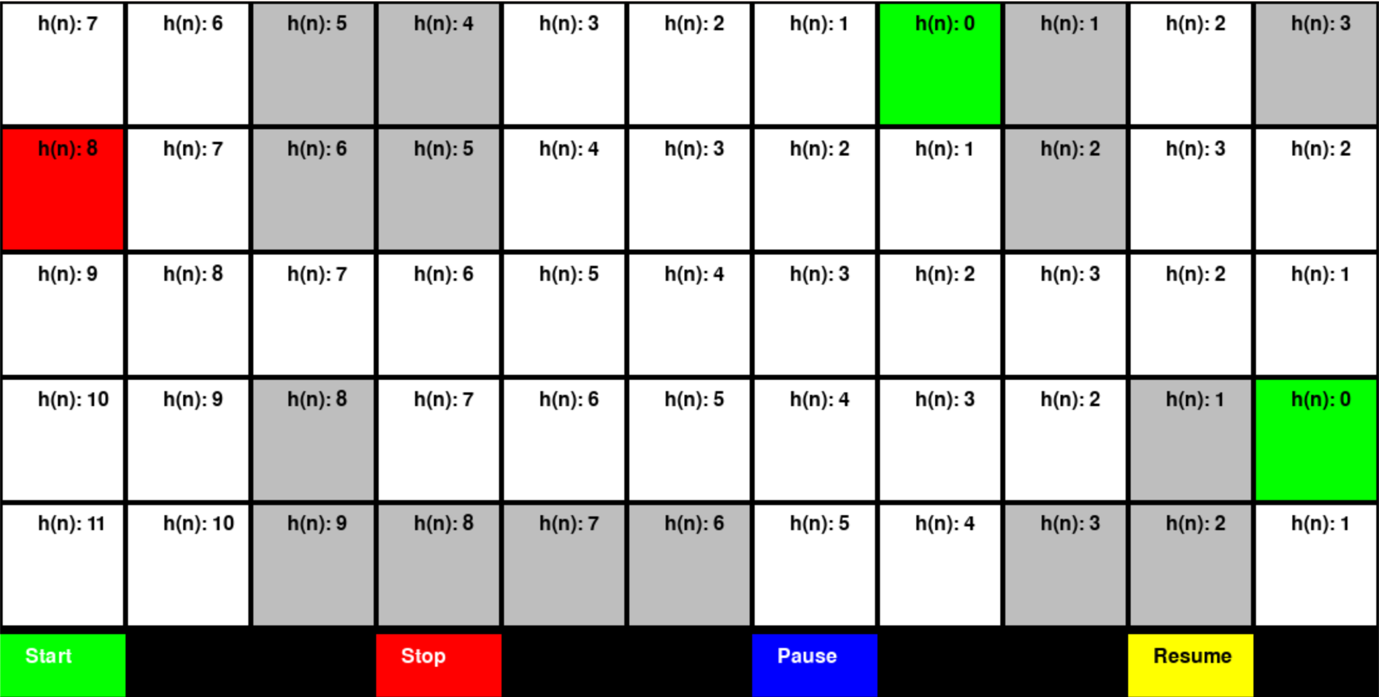


Fig 3. Map with h(n)

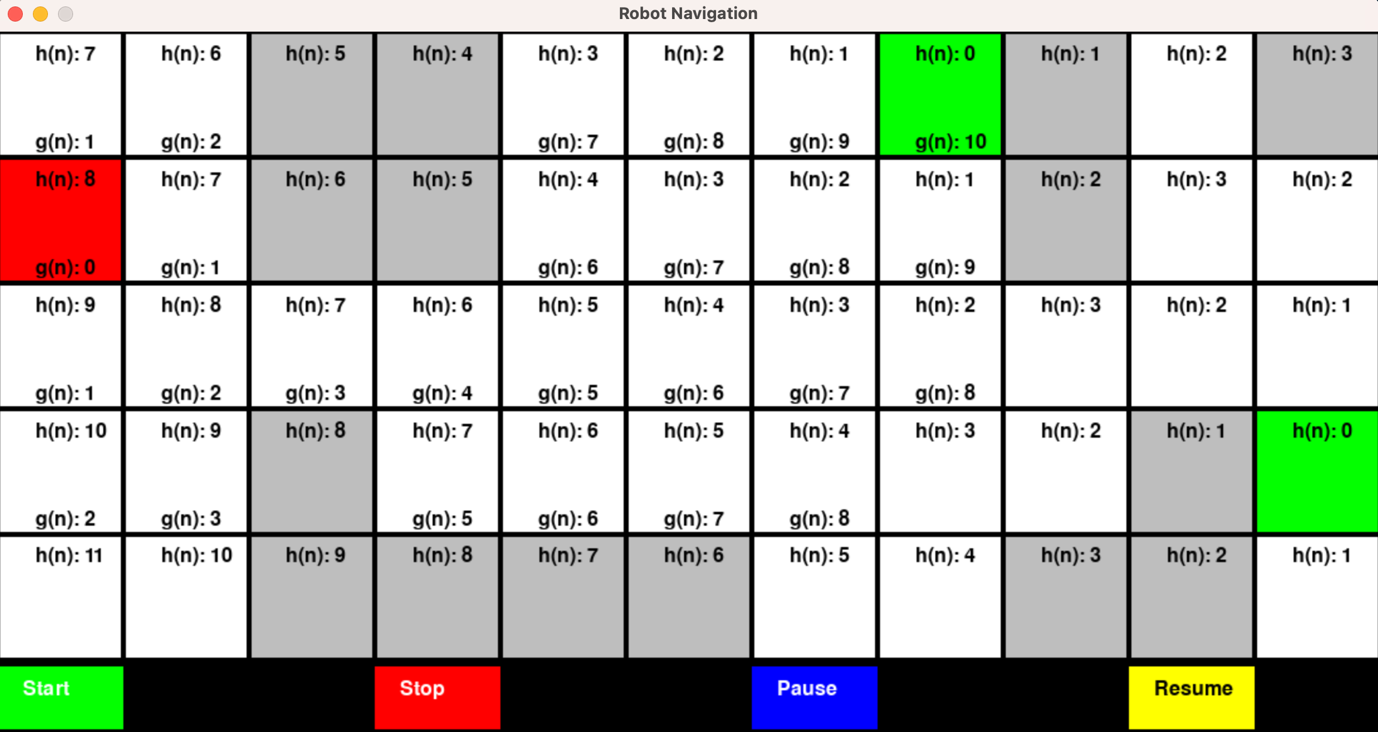


Fig 4. Map with h(n) and g(n)

It is essential to note how the GUI visualize the expansion of nodes before reaching one of the goals. Right after the user presses the Start button, the exploration begins, and any expanded node will be highlighted in yellow.

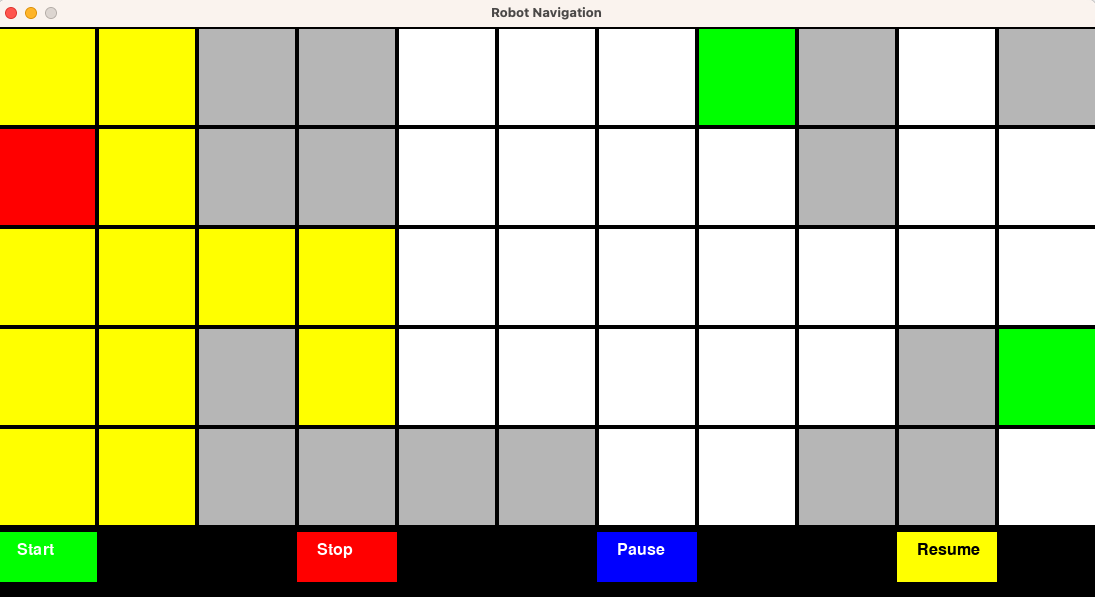


Fig 5. Node expansion

Last but not least, the UI will highlight the path in orange after finishing a search algorithm. This feature is important, as it notify users when the algorithm is completed. Knowing the path found is also necessary for the user to evaluate the completeness and optimality of the algorithm. Eventually, the UI is closed after 5 seconds. This is intended for time-saving and avoiding resource corruption.



Fig 5. Highlighted path

# Conclusion

This report has gone through all 6 search algorithms and how to implement them in the context of navigation problems. In addition to the command line prompt, the program also have a decorative and interactive GUI for users to learn about the algorithms.

If I have to choose the best algorithms for this navigation problem, I would like to choose A-star and IDDFS. According to the results, these algorithms are both complete and optimal. They also have relatively low space complexity, which is a huge advantage if one wants to deploy a resource-efficient algorithm.

To improve the perfection of this program, it might need to have a 3D-based GUI and more advanced features of the GUI, such as a separate application or a menu stage, so that users can choose from a collection of pre-built environments. This might be done using more advanced libraries such as the Unreal Engine. In general, they will be considered in the near future.

# Resources

* VisualGo.net - <https://visualgo.net/en/dfsbfs>: This website provide interactive animations of a variety of DSA, including the graph and graph traversal algorithms. BFS and DFS have been visualized by it with wonderful animations.
* GeeksforGeeks - <https://www.geeksforgeeks.org/a-search-algorithm> and <https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/> : This website discusses a wide variety of topics regarding Computer Science, from basic programming for beginners to manual of implementing advances algorithms. In these links, suggest the Python implementations of A-start and IDDFS. I have consulted their code to build my algorithm.
* Analytics Vidhya - <https://www.analyticsvidhya.com/blog/2021/02/uninformed-search-algorithms-in-ai/> : This website gives an overview of all the uninformed search algorithms and explains clearly their mechanism. It helps me get solid understanding of uninformed searches, before deploying them.
* Builtin - [https://builtin.com/software-engineering-perspectives/beam-search#](https://builtin.com/software-engineering-perspectives/beam-search): This website gives deep explanation of beam search. It also explained how beam-width can be implemented in navigation.
* Home – Resonance: A good Lofi song I have taken as the background music for the GUI.

# References

Belwariar, R. (2018). *A\* Search Algorithm - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/a-search-algorithm.

GeeksForGeeks (2016). *Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS) - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/.

GeeksForGeeks (2024). *Backtracking Algorithms*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/backtracking-algorithms/.

Home (2014). *Resonance*. [Odyssey] Midwest Collective. Available at: https://www.youtube.com/watch?v=8GW6sLrK40k [Accessed 19 Apr. 2024].

Sriniketh, J. (2021). *Uninformed Search Algorithms in AI | Search Algorithms in AI*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/02/uninformed-search-algorithms-in-ai/.

Topper, N. (2023). *Introduction to the Beam Search Algorithm | Built In*. [online] builtin.com. Available at: https://builtin.com/software-engineering-perspectives/beam-search# [Accessed 7 Apr. 2024].

Grivokostopoulou, F., & Hatzilygeroudis, I., 2013. TEACHING AI SEARCH ALGORITHMS IN A WEB-BASED EDUCATIONAL SYSTEM. *International Association for Development of the Information Society*.

Grivokostopoulou, F., Perikos, I., & Hatzilygeroudis, I. (2016). An Educational Game for Teaching Search Algorithms. , 129-136. https://doi.org/10.5220/0005864601290136.

Ying-hua, L. (2012). Teaching System of Advanced Mathematics Experiment Based on GUI. *Journal of Chengde Petroleum College*.