COS300019 Introduction to Artificial Intelligence

Report Assignment 1

Chrishan Christesious Aloysious

101268207

Table of Contents

[Introduction 3](#_Toc7990987)

[Search Algorithms 4](#_Toc7990988)

[Uniformed search algorithms 4](#_Toc7990989)

[Depth-first search 4](#_Toc7990990)

[Breadth-first search 5](#_Toc7990991)

[Iterative deepening depth first search 5](#_Toc7990992)

[Informed search algorithms 6](#_Toc7990993)

[Greedy best-first search 6](#_Toc7990994)

[A\* 6](#_Toc7990995)

[Iterative deepening A\* 7](#_Toc7990996)

[Implementation 7](#_Toc7990997)

[Features 9](#_Toc7990998)

[Conclusion 10](#_Toc7990999)

[Acknowledgements/Resources 11](#_Toc7991000)

[References 11](#_Toc7991001)

# Introduction

WallE is a Robot Navigation program that would navigate in a cell-based environment. The environment would consist number of cells occupied by walls and the robot will not be able to go through the wall. The robot is placed initially in one of the cells and informed about number of goal cells. It is asked to find a path to one of the goal cells.

The robot will only be able to move parallel to the X and Y axis of the environment. This restricts the maximum number of actions the robot can take to four; Up, Left, Down and Right and doesn’t have the option of moving diagonally.

The following is one possible environment. Here the wall cells are grey in color, the green cells are the goal cells and the red cell would be the initial position of the robot. The cells are identified by their coordinates and the origin of this coordinate system would be the leftmost top corner.

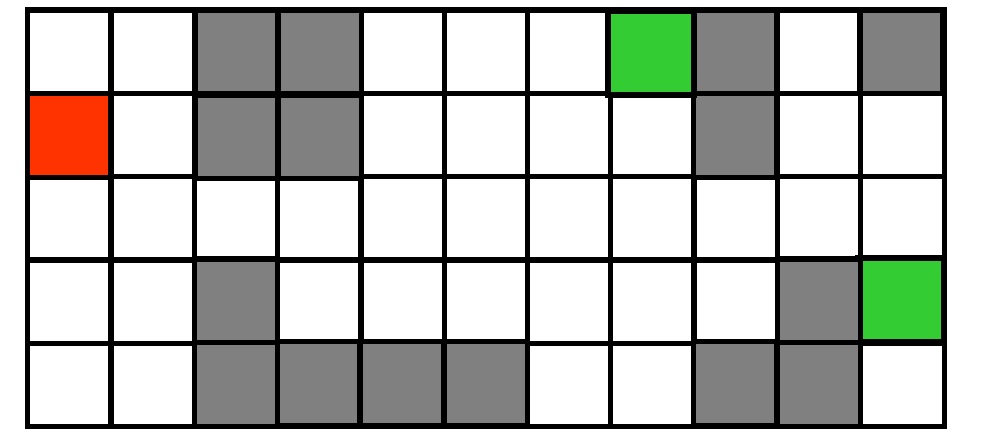


Figure : Example of a grid

When more than one goal is provided WallE will find the path to the goal with the lowest Manhattan distance from the initial position. To find the path to the goal state we use a Tree search. A tree is a special case of graph.

A graph is collection of finite number of vertices/nodes and finite number of edges. A tree is finite set of one or more nodes such that there is a designated node called root.

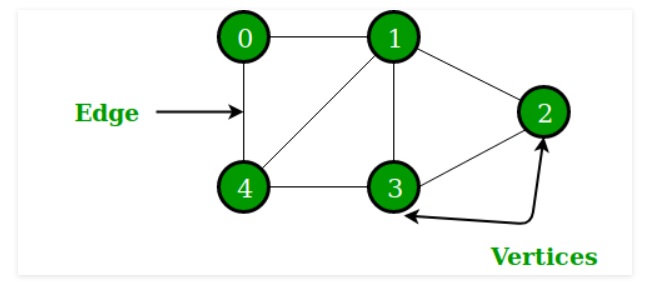


Figure : A graph

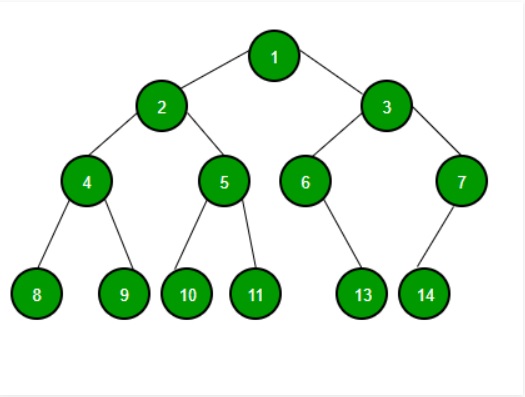


Figure : A Tree

# Search Algorithms

There are six different search algorithms WallE can use to find a path to one of the goals from its initial position. These algorithms can be divided into two main types.

## Uniformed search algorithms

Uninformed search algorithms have access only to the problem definition and doesn’t have any knowledge to find the solution. This strategy is also called a blind search as it totally ignores the destination it is heading towards in the path until it discovers the goal.

In our robot navigation we used three uninformed search strategies. They are Depth-First Search, Breadth-first search and Iterative deepening depth first search.

### Depth-first search

Depth-first search uses a stack which implements a ‘LIFO’ (Last In First Out) to add and remove nodes to and from the frontier. By implementing the stack, the search is proceeded in depth first manner. The algorithm chooses the first alternative at each node then backtracks to another alternative until it has traversed all paths from first selection.

The memory required being linear with respect to the search graph is one of the main advantages of depth-first search. And the time complexity of this algorithm to depth d is O(b^d) making it a time-limited search than a space-limited search.

However, being a blind search method, it is not guaranteed to find the solution and there is no assurance to find a minimal solution, if more than one solution exists.

### Breadth-first search

Breadth-first search explores the search tree to find the goal layerwise. To implement this strategy a queue is used to add and remove nodes from the frontier. A queue follows a ‘FIFO’ (First In First Out) approach.

The major advantage of breath-first search is it will never get trapped exploring the useless path forever and if a solution exists it is guaranteed to find it. And if there is more than one solution Breadth first search will find the solution that requires a smaller number of steps.

Due to it’s layerwise exploration, breath first search will consume lot of time if the solution is farther away from the root. And as each level of the tree must be saved in order to generate the next level the space complexity of Breadth First Search is O(b^d), making it a space bound search than a time bound search.

### Iterative deepening depth first search

Iterative Deepening Depth First Search or IDDFS is the uninformed custom search method implemented in WallE. As mentioned above, Depth First Search is time limited and Breadth First Search is time bound. IDDFS combines Depth First Search’s space efficiency and Breadth First Search’s fast search for nodes closer to root.

IDDFS does a Depth First Search for different depths starting from an initial value. However here, the search is restricted to a depth limit. If one of the goal nodes is not found the search is initiated again from the start with the depth limit being increased to one more level.

Basically, a Depth First Search carried out in a Breadth First Search fashion. So the frontier is a stack that is used in Depth First Search.

This search strategy is best suited for a complete infinite tree and in situations where slower performance is accepted and/or memory usage is limited i.e. BFS + DFS. In summary we are trading off time for memory.

IDDFS can also be viewed as a Depth Limited Search but with an increasing depth limit till a goal node is found if it exists.

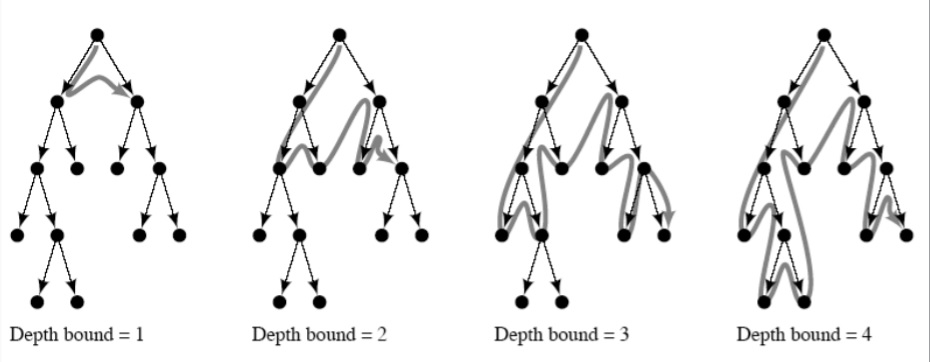


Figure : IDDFS

## Informed search algorithms

The inefficiency of uninformed search algorithms is unsurprisingly due to its lack of knowledge about features of the search space. Informed search algorithms have some indication of which nodes are better which would lead the search agent towards more promising paths. The heuristic value is the key idea behind informed search algorithms. It is specific for a particular search problem. For a routing problem, Euclidean distance can be used as a heuristic. In our case, we use the Manhattan distance between the goal and the robot.

There are three informed search algorithms used in WallE and they are Greedy best-first search, A\* and Iterative deepening A\*.

### Greedy best-first search

Greedy best-first search uses the heuristic function which is the Manhattan distance between the goal and the robot and evaluates nodes to expand. The node that is closes to the goal will be given priority during expansion.

Most of the time Greedy best-first search will provide a non-optimal solution.

### A\*

A\* algorithm combines features of uniform-cost search and pure heuristic search to efficiently compute optimal solutions. It is one of the best and popular strategy used in path-finding problems. The algorithm uses an evaluation function when evaluating nodes to expand and this evaluation function considers two main factors.

* The movement cost to move from the starting point to the current node
* The heuristic which would be the Manhattan distance between the goal cell and the current cell of the robot. This can be seen as a smart guess.

A\* is a complete and optimal strategy if the heuristic function is admissible, meaning it never overestimates actual cost. The main disadvantage of the algorithm is its memory requirement. As the entire open list must be saved it is severely space-limited.

### Iterative deepening A\*

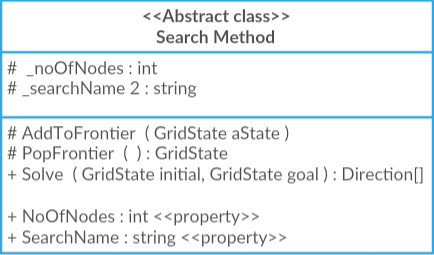
Iterative deepening A\* is the informed custom search method implemented in WallE. Recall how Iterative Deepening Depth First Search eliminated the memory constrain of Breadth First Search. Iterative deepening A\* does the same with A\* without sacrificing optimality.

Iterative deepening A\* or IDAS is very similar to IDDFS. Like IDDFS it performs a series of depth-first searches, and as soon as a node is generated whose evaluation value exceeds a ‘threshold’ for that iteration, its path is cut off. Every time an iteration fails to find a path to the goal the threshold is increased. The algorithm terminates when the gal state is found whose total cost doesn’t exceed the current threshold.

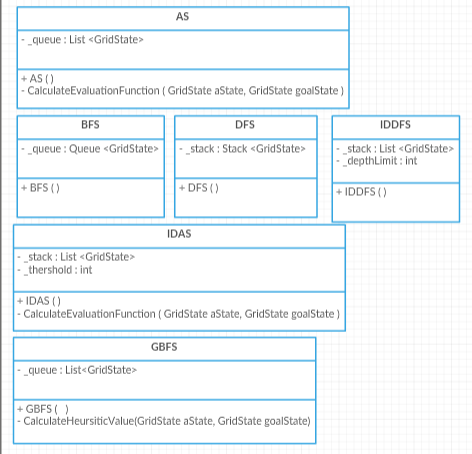
The memory requirement is linear with respect to the maximum search depth. IDA\* is asymptotically optimal in time and space over all heuristic search algorithms that find optimal solutions. Comparatively it’s much easier to implement and runs faster that A\*.

# Implementation

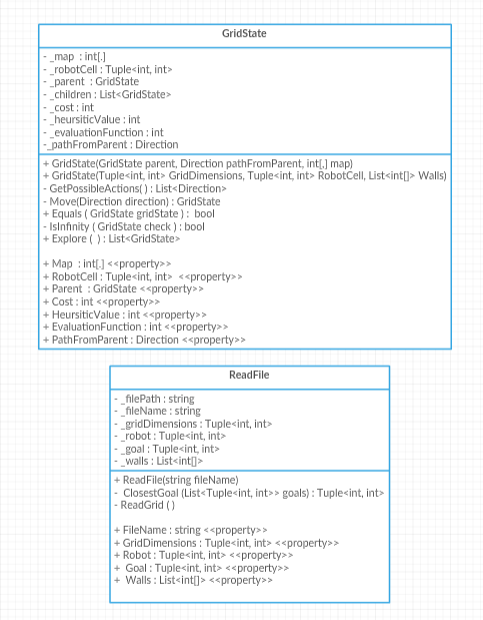
The UML class diagrams are as follows. All search classes inherit an abstract class SearchMethod.



The search algorithms obviously overrides the abstract methods and have their own methods as well.



The GridState class is used to create and represent nodes and the ReadFile class is used to read the input text file.



There’s also an enum class for Direction which consists of Directions Up, Down, Left Right and None.

# Features

WallE has a few features included and have a few features missing.

There’s a couple of instances where code optimization results in a smaller number of nodes created. A\* and GBFS priorities exploring the node with the lowest evaluation value and heuristic value respectively in the frontier. But when two or more nodes have the same values, depending on which one gets explored first the number of nodes created change.

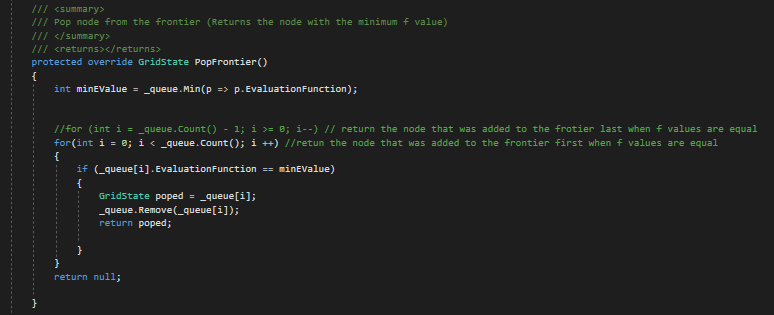


Figure : Code optimizaton A\*

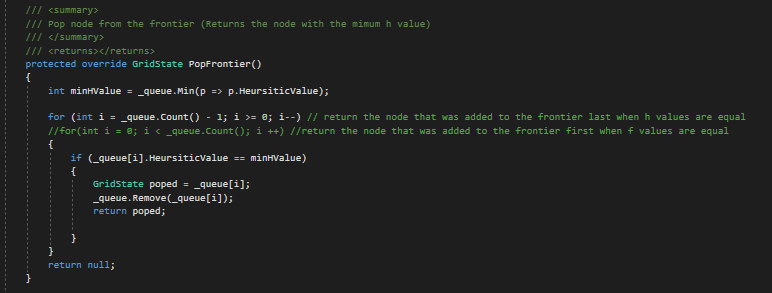


Figure : Code Optimization GBFS

The program does not deal with an invalid search method input when it’s called and only responds to the abbreviations of the search names. For example, it would not respond to an invalid search method name such as “Uniform cost search” or the expanded search name of DFS, “Depth First Search”.

The input file has to be placed in the Debug folder of the program under the name ‘test.txt’. Any other directory and/or text file name would not work.

The GridState class has a map of the grid implemented through a two-dimensional array which is something not necessary. Though something that is not useful and slows down the program this can be useful if the grid state needs to be printed out.

The program doesn’t also deal with the situation if the closest goal is the same as initial cell of the robot.

# Conclusion

In conclusion, each algorithm has their own advantages and disadvantages. Some are space limited and some are time limited. Optimality is a factor that needs to be considered and the possibility of finding a solution if it exists is an obvious factor that needs to be considered.

In previous section the pros and cons of these algorithms have been analyzed in depth. But obviously an informed search is a better choice than uninformed search. And out of the three informed search methods that has been implemented, IDAS stands out as the best choice for our case due to its ‘smart’ guessing, optimality and it is guaranteed to find the solution if one exists.

However it is to be noted all search methods trades something off for another. So depending on which one we can afford to loose the ideal search method would vary in different situations.

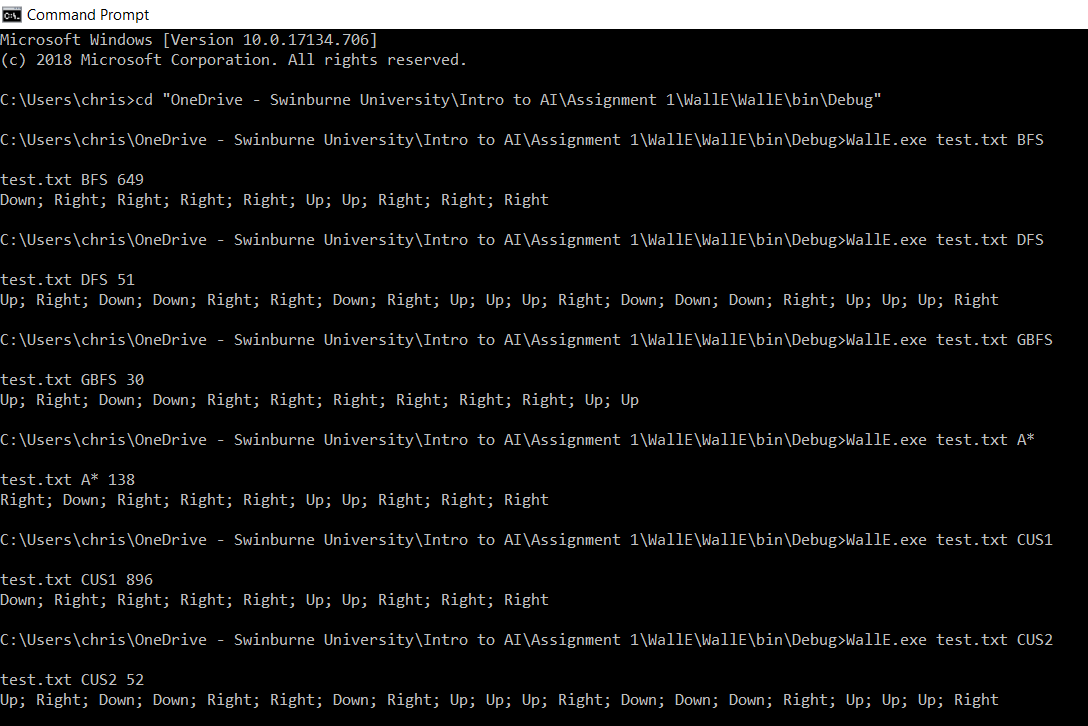


Figure : Command Prompt Output

# Acknowledgements/Resources

Theory reference: Lecture slides

UML class diagrams created in: <https://creately.com/>

Integrated Development Environment used: Microsoft Visual Studio 2017

# References

The following sites were used to understand the theory related to the search algorithms in depth.

* Trees and Graphs : <https://freefeast.info/difference-between/difference-between-trees-and-graphs-trees-vs-graphs/>
* Trees and Graphs : <https://www.geeksforgeeks.org/difference-between-graph-and-tree/>
* IDDFS : <https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/>
* IDDFS : <https://ai.stackexchange.com/questions/8735/when-should-the-iterative-deepening-search-and-the-depth-limited-search-be-used>
* Informed and uninformed search : <https://techdifferences.com/difference-between-informed-and-uninformed-search.html>
* DFS : <http://intelligence.worldofcomputing.net/ai-search/depth-first-search.html#.XM6Lr44zZPY>
* BFS : <https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/practice-problems/>
* BFS : <http://intelligence.worldofcomputing.net/ai-search/breadth-first-search.html#.XM6QKI4zZPY>
* IDDFS, GBFS : <http://www.sci.brooklyn.cuny.edu/~chipp/cis32/lectures/Lecture5.pdf>
* Informed search :<https://kartikkukreja.wordpress.com/2015/06/07/informed-search-algorithms/>
* IDAS : <http://intelligence.worldofcomputing.net/ai-search/iterative-deepening-a-star.html#.XM69bI4zZPY>