

Computer Games Development

Software Functional Specification

Year IV

Donal Howe

C00249662

Suprivisor:

Oisin Cawley

26/04/2023

**DECLARATION**

Text, table

Description automatically generated

**Table Of Contents**

[1 Introduction 1](#_Toc133255805)

[2 Brief description of the chosen algorithms 2](#_Toc133255806)

[2.1 Description of D star Lite 2](#_Toc133255807)

[2.2 Description of A star 2](#_Toc133255808)

[2.3 Description of Dijkstras algorithm 2](#_Toc133255809)

[2.4 Description of lifelong planning Astar 2](#_Toc133255810)

[2.5 Description of depth first search 2](#_Toc133255811)

[3 Functional Specification 3](#_Toc133255812)

[3.1 Algorithm tables and grid sizes 3](#_Toc133255813)

[3.2 Console visualisation 4](#_Toc133255814)

[3.3 Code visualisation 6](#_Toc133255815)

[3.4 Visualisation of paths using the different algorithms available 11](#_Toc133255816)

[3.5 Data collection visualisation 13](#_Toc133255817)

[4 Design and describe how the application will be used 17](#_Toc133255818)

[5 References 18](#_Toc133255819)

###### List Of Tables

[Table 3‑1 Description Of User Interface Overview 4](#_Toc133328423)

[Table 3‑2 Description Of Enum Visualisation 5](#_Toc133328424)

[Table 3‑4 Description Of Generated Paths 8](#_Toc133328425)

[Table 3‑5 Description Of Data Visualisation 11](#_Toc133328426)

# Introduction

The objective of this project was to put together a comprehensive comparison of guided and non-guided based pathfinding algorithms to the incremental dynamic pathfinding algorithm known as Dstar Lite under a game’s development context. So one can decide based of the information shown in this document whether to or not implement Dstar lite into their project or perhaps to implement another algorithm such as lifelong planning A star or, A star itself.

# Brief description of the chosen algorithms

## Description of D star Lite

## Dstar Lite is a search algorithm that works by finding the shortest path between two nodes in a graph. It is similar to the A\* algorithm in that it uses heuristics to guide the search, but it differs in the way it handles changes to the graph. (Koenig, S. & Likhachev, M., 2005)

## When the algorithm starts, it calculates the heuristic cost for each node in the graph based on its distance from the starting node and its estimated distance to the goal node. However, instead of storing the heuristic cost for each node, it stores the cost-to-come, which is the actual cost of reaching a node from the starting node. (Koenig, S. & Likhachev, M., 2005)

## As the algorithm searches the graph, it keeps track of the nodes that have been visited and their costs. If a non-traversable obstacle is placed on the path, the algorithm marks the affected nodes as "inconsistant" i.e its rhs and gcost values are not equal and updates their costs to reflect the change. It then starts the search again, but this time only looks at the dirty nodes and their neighbors, instead of recalculating the entire path. (Koenig, S. & Likhachev, M., 2005)

## Dstar Lite uses a technique called "lazy update" to avoid the expensive re-calculation of the path. When a dirty node is visited, the algorithm checks if its cost has changed. If it has, the algorithm updates the node's cost and adds it to the list of dirty nodes. This process continues until the goal node is reached, or until there are no more dirty nodes to explore. (Koenig, S. & Likhachev, M., 2005)

## Overall, Dstar Lite is a more efficient algorithm than A\* when dealing with dynamically changing environments or maps. It reduces the amount of computation needed by only updating the parts of the graph that have changed, rather than recalculating the entire path. (Koenig, S. & Likhachev, M., 2005)

## Description of A star

## A\* (pronounced "A-star") is a popular heuristic algorithm used to solve shortest path problems. It is widely used in robotics, gaming, and other fields that require pathfinding. The algorithm works by exploring a graph to find the shortest path between a starting node and a goal node. (Swift, N. 2020)

## The A\* algorithm is informed, meaning that it has some knowledge about the problem it is solving. Specifically, it uses a heuristic function to estimate the distance from the current node to the goal node. This heuristic is represented by the "Hcost value." (Swift, N. 2020)

## To find the shortest path, A\* also keeps track of the distance from the starting node to the current node, represented by the letter "g," called the "G-cost." Together, the H-cost and G-cost make up the "F-cost," which is used to compare nodes and determine which node to explore next. (Swift, N. 2020)

## The algorithm starts by adding the starting node to a priority queue, which sorts nodes by their F-cost. It then removes the node with the lowest F-cost from the queue and explores its neighbors. For each neighbor, A\* calculates its F-cost and adds it to the queue. (Swift, N. 2020)

## If a neighbor has already been explored, A\* compares its new F-cost to its old F-cost. If the new F-cost is lower, the neighbor is updated and added back to the queue. If the new F-cost is higher, the neighbor is left alone. (Swift, N. 2020)

## If an obstacle is encountered during the search, A\* will have to recalculate the path to avoid the obstacle. This is done by marking the affected nodes as "dirty" and adding them back to the priority queue. The algorithm continues until the goal node is reached or there are no more nodes in the queue. (Swift, N. 2020)

## A\* is widely used because it is both complete (i.e., it is guaranteed to find a solution if one exists) and optimal (i.e., it finds the shortest path). However, the quality of the solution depends on the quality of the heuristic function used. A good heuristic function can greatly reduce the time and memory required to find the shortest path. (Swift, N. 2020)

## Description of Dijkstras algorithm

Dijkstra's search algorithm is a classic algorithm used to find the shortest path in a weighted graph. It works by iteratively exploring the graph from the start node to find the shortest path to every other node.( Rachmawati, D. and Gustin, L. 2019)

At the start of the algorithm, the distance from the start node to every other node is set to infinity, except for the start node itself, which has a distance of zero. The algorithm then visits the start node and examines all its neighbors. For each neighbor, it calculates the distance from the start node to the neighbor and updates the distance if it is shorter than the current distance.( Rachmawati, D. and Gustin, L. 2019)

The algorithm then selects the unvisited node with the smallest distance and visits its neighbors. This process continues until the algorithm has visited all nodes or until the goal node is reached. .( Rachmawati, D. and Gustin, L. 2019)

Unlike A\* search, Dijkstra's algorithm does not use any heuristic to guide the search. Instead, it uses the distances between nodes as weights to guide the search. In other words, it considers only the cost of the path traveled so far and does not take into account the estimated distance to the goal node.( Rachmawati, D. and Gustin, L. 2019)

Dijkstra's algorithm is guaranteed to find the shortest path in a weighted graph, provided that there are no negative edge weights. However, it can be computationally expensive if the graph is large or if there are many edges. Additionally, it does not work well in situations where the graph is dynamic and changes frequently, as it requires recomputing the entire graph when a change occurs.( Rachmawati, D. and Gustin, L. 2019)

Overall, Dijkstra's algorithm is a powerful tool for finding the shortest path in a weighted graph, but its performance can be limited by the size and structure of the graph.( Rachmawati, D. and Gustin, L. 2019)

## Description of lifelong planning Astar

Lifelong Planning A\* (LPA\*) is a heuristic algorithm used for incremental pathfinding, which means that it updates the shortest path from the start node to every other node as the search progresses. Unlike traditional pathfinding algorithms, LPA\* does not require a complete graph or a precomputed heuristic function. (Koenig, S., Furcy, D. and Likhachev , M. 2004)

The algorithm starts with the start node and calculates the shortest path to all its neighbors. It then adds these neighbors to a priority queue based on their g-value, which is the length of the path from the start node to the current node. The priority queue is ordered by the sum of the g-value and a heuristic estimate of the remaining distance to the goal node, called the h-value. (Koenig, S., Furcy, D. and Likhachev , M. 2004)

As the search progresses, LPA\* updates the g-values of the nodes to reflect the shortest path found so far. Whenever a node is updated, its neighbors are added back to the priority queue to check if a shorter path can be found through them. (Koenig, S., Furcy, D. and Likhachev , M. 2004)

LPA\* continues until the goal node is reached or until the shortest path is found to all nodes. If the graph changes, LPA\* can update the shortest paths by only considering the affected nodes, rather than recomputing the entire graph. (Koenig, S., Furcy, D. and Likhachev , M. 2004)

One advantage of LPA\* over other pathfinding algorithms is its ability to handle changing environments. It can efficiently update the shortest paths when the graph changes, rather than computing them from scratch. LPA\* is also guaranteed to find the shortest path if the heuristic is consistent. (Koenig, S., Furcy, D. and Likhachev , M. 2004)

However, LPA\* can be slower than other algorithms because it needs to update the shortest paths of all nodes each time a node is changed. Additionally, it requires more memory to store the g-values and the priority queue. (Koenig, S., Furcy, D. and Likhachev , M. 2004)

In summary, Lifelong Planning A\* is an incremental pathfinding algorithm that efficiently updates the shortest paths as the graph changes. It uses a priority queue based on the g-values and h-values to guide the search, and it is guaranteed to find the shortest path if the heuristic is consistent. (Koenig, S., Furcy, D. and Likhachev , M. 2004)

## Description of depth first search

Depth First Search (DFS) is an algorithm used for traversing or searching through a tree or graph data structure. It starts at the root node or a selected node and explores as far as possible along each branch before backtracking. (kaur, N. and Garg, D. 2012)

The algorithm works by visiting the node and marking it as visited, then recursively visiting all its unvisited neighbors. It continues this process until it reaches a dead end, i.e., a node with no unvisited neighbors. It then backtracks to the previous node and continues exploring any remaining unvisited neighbors. (kaur, N. and Garg, D. 2012)

DFS uses a stack data structure to keep track of the visited nodes and the nodes that still need to be visited. Whenever it visits a new node, it adds it to the top of the stack, and whenever it backtracks, it removes the top node from the stack. (kaur, N. and Garg, D. 2012)

DFS can be used to solve a variety of problems, including finding a path between two nodes, detecting cycles in a graph, and generating mazes. It is also used as a building block for more complex algorithms such as topological sorting and Tarjan's algorithm. (kaur, N. and Garg, D. 2012)

One advantage of DFS is that it requires less memory than breadth-first search (BFS) since it only needs to store the nodes on the current path. However, it may not find the shortest path to a goal node since it explores one path at a time and may get stuck in a local maximum. (kaur, N. and Garg, D. 2012)

Overall, DFS is a non-heuristic guided search algorithm that explores a graph or tree by traversing as far as possible along each branch before backtracking. It uses a stack data structure to keep track of the visited nodes and is useful for solving a variety of problems(kaur, N. and Garg, D. 2012)

.

# Functional Specification

The software will in essence function as a visual pathfinding application. So the user will run the application and see a basic grid they can then adjust the size of the grid to three specified sizes “Small” being a 10x10 grid, “Medium” being a 50x50 grid and “Large” being a 100x100 size grid. They can then choose from a variety of pathfinding algorithms them being Astar, Dstar Lite, Dijkstra’s algorithm, lifelong planning Astar , jump point search and the only no heuristic pathfinding algorithm depth first search. The user can also place down obstacles during process of the algorithms search and before the algorithm has been ran if they perhaps are looking for a specific path, onto the grid which will have the pathfinding algorithms react to them and find a corresponding path.

## Algorithm tables and grid sizes

|  |  |
| --- | --- |
| User Interface Images | Description |
| Chart  Description automatically generated with low confidence | The Menu:  The user can select grid size and the algorithm they wish to use. |
| A picture containing shoji, building  Description automatically generated | Grid size “Small”:  The small grid with 100 cells and row of columns of 10 each |
| Background pattern  Description automatically generated | Grid size “Medium”:  The medium grid with 2500 cells and row of columns of 50 each |
| Background pattern  Description automatically generated | Grid size “Large”:  The large grid with 10,000 cells and row of columns of 100 each |

Table 3‑1 Description Of User Interface Overview

## Console visualisation

|  |  |
| --- | --- |
| Console Images | Description |
| Text  Description automatically generated | The Astar Search algorithm being ran and displayed in the console there will be a visual representation in the application. |
| Text  Description automatically generated | The Lifelong Planning Astar (LPA\*) algorithm being ran and displayed in the console there will be a visual representation in the application. |
| A picture containing text  Description automatically generated | The Jump Point Search algorithm being ran and displayed in the console there will be a visual representation in the application. |
| Text  Description automatically generated with medium confidence | The Dstar Lite Search algorithm being ran and displayed in the console there will be a visual representation in the application. |
| Text  Description automatically generated | The Dijkstra’s Search algorithm being ran and displayed in the console there will be a visual representation in the application. |
| Text, chat or text message  Description automatically generated | The Depth First Search algorithm being ran and displayed in the console there will be a visual representation in the application. |

Table 3‑2 Description Of Enum Visualisation

## Visualisation of paths using the different algorithms available

|  |  |
| --- | --- |
| Generated Path images | Description |
| A picture containing shoji, building, silhouette  Description automatically generated | An example of basic walls. Red Nodes are the walls placed on the grid. The grid size in question is the small grid of size 100 nodes. |
| A picture containing shoji, crossword puzzle, building, clipart  Description automatically generated | Path returned using Astar Search algorithm on a grid size “Small”  Green Node = Start Node  Magenta Node = Goal Node |
|  | Path returned using Dstar Lite Search algorithm on a grid size “Small”  Magenta Node = Start Node  Blue Node = Goal Node |
|  | Dstar Lite with Debug on second Screen |
| Chart  Description automatically generated | Path returned using Lifelong Planning Astar Search algorithm on a grid size “Small”  Green Node = Start Node  Magenta Node = Goal Node |
| Table  Description automatically generated with medium confidence | Path returned using Depth First Search algorithm on a grid size “Small” |
| A picture containing shoji, crossword puzzle, building  Description automatically generated | Path returned using Dijkstra’s Search algorithm on a grid size “Small” |

Table 3‑4 Description Of Generated Paths

## Data collection visualisation

|  |  |
| --- | --- |
| Data Collection Images | Description |
| Text  Description automatically generated | Dstar lite Small Grid Data |
| Text  Description automatically generated | Dstar lite Medium Grid Data |
| Text, table  Description automatically generated | Dstar lite Large Grid Data |
| Table  Description automatically generated | Astar Small Grid Data |
|  | Astar Medium Grid Data |
| Table  Description automatically generated | Astar Large Grid Data |
| Text  Description automatically generated | Dijkstra’s Small Grid Data |
|  | Dijkstra’s Medium Grid Data |
| Text  Description automatically generated | Dijkstra’s Large Grid Data |
|  | Lifelong Planning Astar Small Grid Data |
|  | Lifelong Planning Astar Medium Grid Data |
|  | Lifelong Planning Astar Large Grid Data |
|  | Depth First Search Small Grid Data |
|  | Depth First Search Medium Grid Data |
|  | Depth First Search Large Grid Data |

Table 3‑5 Description Of Data Visualisation

# Design and describe how the application will be used

**How Will the programme run?**

When the user first opens the programme they will see three separate screens, with all having different functionality.The first screen the user will see is the “Menu” this is where the user will be able to control type of algorithm they want to run, the size of the grid that they want to run the algorithm on, whether they want to run their algorithm of choice against “Dstar Lite” and finally whether they want to run Dstar Lite in debug mode.The user can select six search algorithms from “dstar lite”,”astar”,”lifelong planning astar”,”depth first search”, “dijkstra’s search” and the “jump point search” algorithms.

The second grid is the editable grid. What the user can do on this grid is choose the start and end points of their algorithm.They can also place down as many walls as they want on the grid if they want a specific path.

The third and final screen intilly is black and once they user selects that they want to race the algorithms, on this screen a grid will appear to match the one on the second screen.This will then match any input you place on the second screen such as walls and the end and start points.Then once the start and endpoints are selected on the second screen Dstar Lite only will run on this screen. If the user selects to have debug mode on then this screen will run dstar lite in debug mode.This will have the three variables “rhs cost” ,”g cost” and their “key value” of each cell appear so the user can see how the algorithm effects each cell in real time.

**What is the purpose of the application?**

The purpose of the application is for the user to see the differences in paths in real time. So that they can potentially decide which algorithm best suits their grid. They can see how each algorithm reacts to changes in the path on a dynamic grid of varying sizes and they can also see the speed of each algorithm on a static grid where nothing changes if they so choose.

The programme also stores the time it took for you algorithm to find the paths time into an excel file in seconds so th user can then see the time for the algorithm to finish to better see the a difference between the algorithms. They can see all of the data in the three separate excel files for each of the algorithms for example the times for astar on the small grid they can see in “AstarTime.csv”

That is the functionality and purpose of the application.

# References

Foead, D, Ghifari A, Kusuma B M, Hanafiah N,Gunawan E.(2021) “A Systematic Literature Review of A\* Pathfinding,” *Procedia Computer Science*, 179, pp. 507–514.

Harabor, D. and Grastien, A. (2014) “Improving Jump Point Search,” *Proceedings of the International Conference on Automated Planning and Scheduling*, 24(1), pp. 128–135.

kaur, N. and Garg, D. (2012) *Analysis of the depth first search algorithms - gdeepak.com*. Available at: https://www.gdeepak.com/pubs/Analysis%20of%20the%20Depth%20First%20Search%20Algorithms.pdf

Koenig, S. & Likhachev, M., (2005). Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3), pp.354–363.

Koenig, S., Furcy, D. and Likhachev , M. (2004) “ Lifelong Planning A\*,” *Artificial Intelligence Journal*, 155(1-2), pp. 93–146.

Likhachev, M,Ferguson I D,Gordon J G, Stentz A, Thrun S. (2005) Anytime dynamic A\*: An anytime, planning algorithm., ICAPS, 5, pp. 262–271.

Luo, Y., Jiakai L, Qiong Q, Yanyu L*.* (2022) “Improved JPS path optimization for mobile robots based on angle-propagation theta\* algorithm,” *Algorithms*, 15(6), p. 198. Available at: https://doi.org/10.3390/a15060198.

Patel (2019) *Heuristics from Amit’s Thoughts on Pathfinding*, *Heuristics*. Available at: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html (Accessed: April 25, 2023).

Patel, R.L., Pathak, M.J. and Rami, S.P. (2018) *International Journal of Computer Applications (0975 – 8887)*, 179(50).

Rachmawati, D. and Gustin, L. (2019) “Analysis of Dijkstra's Algorithm and A\* Algorithm in Shortest Path Problem,” *Journal of Physics: Conference Series*, 1566, pp. 26–27.

Raheem, F.A. and Hameed, U.I. (2018) “Path Planning Algorithm using D\* Heuristic Method Based on PSO in Dynamic Environment,” *American Academic Scientific Research Journal for Engineering, Technology, and Sciences* [Preprint].

Stentz, A. (1994) “The D\* Algorithm for Real-Time Planning of Optimal Traverses.”

Swift, N. (2020) *Easy A\* (star) pathfinding*, *Medium*. Medium. Available at: https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2 .