



Computer Games Development Software Functional Specification

Year IV

Donal Howe
C00249662
Suprvisor:
Oisin Cawley
26/04/2023


DECLARATION

Work submitted for assessment which does not include this declaration will not be assessed.

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) Donal Howe

Student Number(s): C00249662

Signature(s): 

Date: 24/04/2023

Please note:

- a) * Individual declaration is required by each student for joint projects.
- b) Where projects are submitted electronically, students are required to submit their student number.
- c) The Institute regulations on plagiarism are set out in Section 10 of Examination and Assessment Regulations published each year in the Student Handbook.

Table Of Contents

1	Introduction.....	1
2	Brief description of the chosen algorithms	2
2.1	Description of D star Lite.....	2
2.2	Description of A star	2
2.3	Description of Dijkstras algorithm.....	2
2.4	Description of lifelong planning Astar.....	4
2.5	Description of depth first search	4
3	Functional Specification	6
3.1	Algorithm tables and grid sizes.....	6
3.2	Console visualisation.....	7
3.3	Code visualisation	Error! Bookmark not defined.
3.4	Visualisation of paths using the different algorithms available	9
3.5	Data collection visualisation	11
4	Design and describe how the application will be used	18
5	References.....	19

List Of Tables

Table 3-1 Description Of User Interface Overview.....	7
Table 3-2 Description Of Enum Visualisation.....	8
Table 3-4 Description Of Generated Paths	11
Table 3-5 Description Of Data Visualisation.....	17

1 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.

2 Brief description of the chosen algorithms

2.1 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.

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.

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 "inconsistent" 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.

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.

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.

2.2 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.

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."

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.

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.

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.

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.

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.

2.3 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.

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.

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.

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.

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.

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.

2.4 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.

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.

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.

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.

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.

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.

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.

2.5 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.

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.

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.

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.

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.


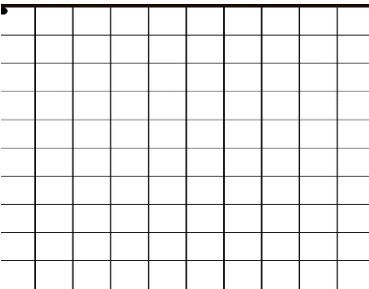
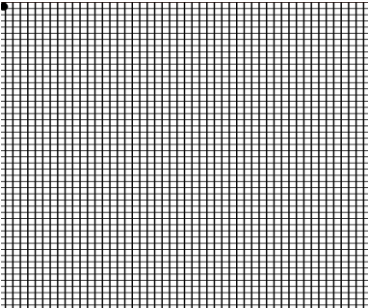
Overall, DFS is a non-heuristic guided search algorithm that explores a graph or tree by traversing as far as possible along each branch before backtracking. It uses a stack data structure to keep track of the visited nodes and is useful for solving a variety of problems

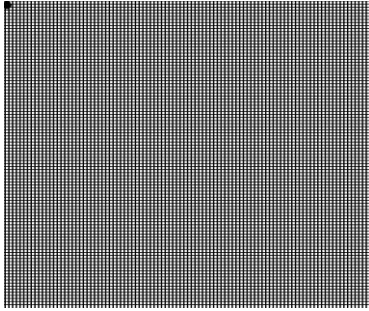
.

3 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.

3.1 Algorithm tables and grid sizes

User Interface Images	Description
 <p>The menu consists of several buttons: 'SMALL', 'MEDIUM', 'LARGE' (all red); 'ASTAR', 'DSTAR LITE', 'DIJKSTRAS' (all cyan); 'LPA STAR', 'DEPTH (DFS)', 'JUMP POINT SEARCH' (all green); 'I WANT TO RACE THE ALGORITHMS', 'I DON'T WANT TO RACE THE ALGORITHMS' (both pink); and 'I WANT DEBUG ON', 'I WANT DEBUG OFF' (both blue).</p>	<p>The Menu:</p> <p>The user can select grid size and the algorithm they wish to use.</p>
 <p>A 10x10 grid with 100 cells.</p>	<p>Grid size “Small”:</p> <p>The small grid with 100 cells and row of columns of 10 each</p>
 <p>A 50x50 grid with 2500 cells.</p>	<p>Grid size “Medium”:</p> <p>The medium grid with 2500 cells and row of columns of 50 each</p>

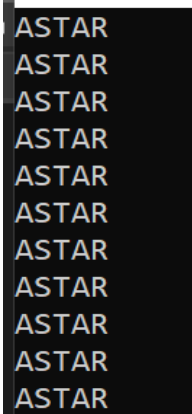
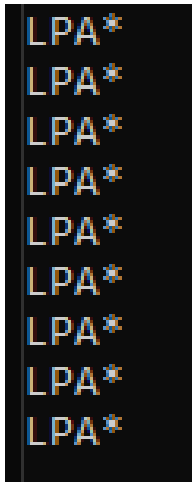


Grid size “Large”:

The large grid with 10,000 cells and row of columns of 100 each

Table 3-1 Description Of User Interface Overview

3.2 Console visualisation

Console Images	Description
	The Astar Search algorithm being ran and displayed in the console there will be a visual representation in the application.
	The Lifelong Planning Astar (LPA*) algorithm being ran and displayed in the console there will be a visual representation in the application.

```
jump Point  
jump Point  
jump Point  
jump Point  
jump Point  
jump Point  
jump Point  
jump Point  
jump Point  
jump Point
```

The Jump Point Search algorithm being ran and displayed in the console there will be a visual representation in the application.

```
DSTAR  
DSTAR  
DSTAR  
DSTAR  
DSTAR  
DSTAR  
DSTAR  
DSTAR  
DSTAR  
DSTAR
```

The Dstar Lite Search algorithm being ran and displayed in the console there will be a visual representation in the application.

```
DepthFirstS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS  
DIKSTRAS
```

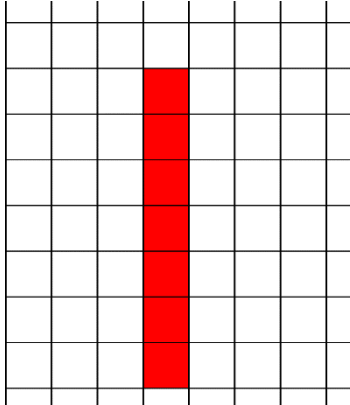
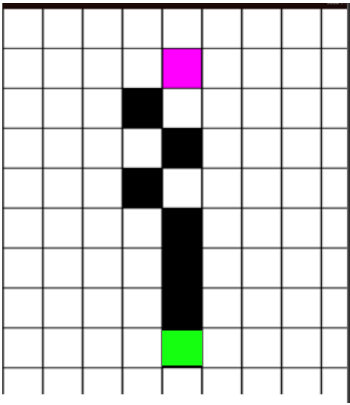
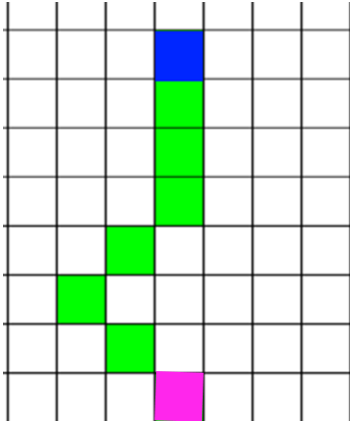
The Dijkstra's Search algorithm being ran and displayed in the console there will be a visual representation in the application.

```
DepthFirstSearch  
DepthFirstSearch  
DepthFirstSearch  
DepthFirstSearch  
DepthFirstSearch  
DepthFirstSearch  
DepthFirstSearch
```

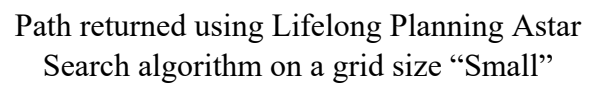
The Depth First Search algorithm being ran and displayed in the console there will be a visual representation in the application.

Table 3-2 Description Of Enum Visualisation

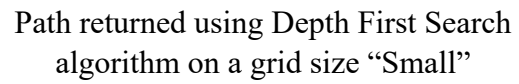
3.3 Visualisation of paths using the different algorithms available

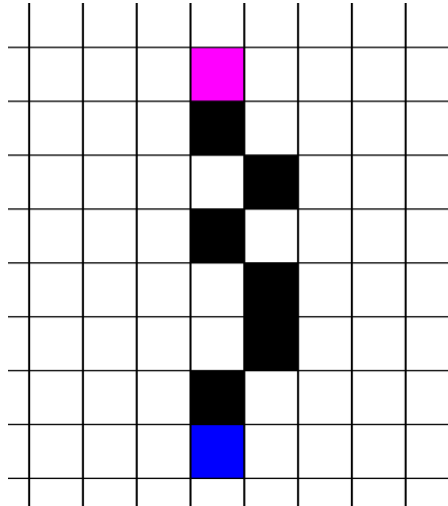
Generated Path images	Description
	An example of basic walls. Red Nodes are the walls placed on the grid. The grid size in question is the small grid of size 100 nodes.
	Path returned using Astar Search algorithm on a grid size “Small” Green Node = Start Node Magenta Node = Goal Node
	Path returned using Dstar Lite Search algorithm on a grid size “Small” Magenta Node = Start Node Blue Node = Goal Node

Dstar Lite with Debug on second Screen



Magenta Node = Goal Node





Path returned using Dijkstra’s Search algorithm on a grid size “Small”

Table 3-3 Description Of Generated Paths

3.4 Data collection visualisation

Data Collection Images	Description
<div> <div>small0.090104</div> <div>small0.090104</div> <div>small0.090104</div> <div>small0.090104</div> <div>small0.090104</div> <div>small0.035049</div> <div>small0.035049</div> <div>small0.035049</div> <div>small0.035049</div> <div>small0.035049</div> </div>	Dstar lite Small Grid Data
<div> <div>medium0.607782</div> <div>medium0.607782</div> <div>medium0.587208</div> <div>medium0.587208</div> <div>medium0.587208</div> <div>medium0.587208</div> <div>medium0.587208</div> <div>medium0.587208</div> <div>medium0.587208</div> </div>	Dstar lite Medium Grid Data

4	large30.132551
5	large30.132551
6	large30.132551
7	large30.132551
8	large30.132551
9	large30.132551
0	large15.718164
1	large15.718164

Dstar lite Large Grid Data

0	0.138869
1	0.135377
2	0.137532
3	0.138712
4	0.137757
5	0.13468
6	0.13875
7	0.134387
8	0.139464
9	0.14005
0	0.135846
1	0.13409
2	0.137532

Astar Small Grid Data

0	0.054926
1	0.052687
2	0.05154
3	0.053191
4	0.053235
5	0.053147
6	0.052287
7	0.057563
8	0.06216
9	0.053713
0	0.05319
1	0.05194

Astar Medium Grid Data

0.214963
0.215844
0.211203
0.227789
0.210902
0.2214
0.208909
0.213074
0.208984
0.212795
0.209617
0.215853
0.218203

Astar Large Grid Data

small0.004657
small0.004215
small0.004081
small0.004760
small0.004344
small0.004281
small0.004179
small0.004119
small0.004166
small0.004177
small0.004104

Dijkstra's Small Grid Data

medium0.106067
medium0.101982
medium0.100599
medium0.099943
medium0.103269
medium0.104548
medium0.101578
medium0.107218
medium0.103903

Dijkstra's Medium Grid Data

large0.306492
large0.314421
large0.310135
large0.305354
large0.310717
large0.311183
large0.320436
large0.301858
large0.311105
large0.302993
large0.310332
large0.302764
large0.313947
large0.303118
large0.304529
large0.314564
large0.306293
large0.310978
large0.394907
large0.415434

Dijkstra's Large Grid Data

1	0.060988
2	0.053446
3	0.052542
4	0.048405
5	0.05359
6	0.056054
7	0.060459
8	0.049954
9	0.016173
10	0.031826
11	0.043666
12	0.019829
13	0.019031
14	0.057758
15	0.019782
16	0.014882
17	0.01341
18	0.019583
19	0.044412
20	0.067002
21	0.058747
22	0.054505
23	0.056759
24	0.014936
25	
26	0.000001
27	0.044518
28	0.064991
29	0.057503
30	0.043987
31	0.052066
32	0.056835
33	0.043385
34	0.03708
35	0.04824
36	0.049523
37	0.047292
38	0.065324
39	0.041056
40	0.05734
41	0.012315
42	0.01315
43	0.052136
44	0.032886
45	0.039039
46	0.047111
47	0.044316
48	0.047052
49	SMALL0.047077
50	SMALL0.012580

Lifelong Planning Astar Small Grid Data

1	6.85173
2	0.734882
3	1.834106
4	0.138732
5	0.129428
6	0.124472
7	0.128328
8	0.125412
9	0.128645
10	0.125299
11	0.126269
12	0.141289
13	0.135602
14	0.127624
15	0.133236
16	0.128248
17	0.127459
18	0.127251
19	0.134204
20	0.125799
21	0.13666
22	0.128267
23	0.125909
24	0.122678
25	0.124547
26	0.12436
27	0.127235
28	0.128062
29	0.129842
30	0.126661
31	0.124545
32	0.128332
33	0.126248
34	0.121886
35	0.122602
36	0.121884
37	0.129145
38	0.125402
39	0.12579
40	0.125083
41	0.121427
42	0.122467
43	0.127048
44	0.12334
45	0.123312
46	0.124714
47	0.124113
48	0.127719
49	0.123168
50	0.122851

Lifelong Planning Astar Medium Grid Data

1	31.07115
2	52.64384
3	33.74767
4	40.86055
5	47.72051
6	45.53032
7	53.24764
8	52.95221
9	3.57054
10	4.780712
11	45.7002
12	6.730712
13	0.751454
14	6.681502
15	0.541438
16	0.738923
17	2.816107
18	0.142126
19	0.123806
20	0.127686
21	0.127129
22	0.122066
23	0.129806
24	0.124657
25	0.124153
26	0.125186
27	0.140876
28	0.122897
29	0.125032
30	0.12083
31	0.123917
32	0.121894
33	0.120957
34	0.131863
35	0.127367
36	0.125948
37	0.123203
38	0.122842
39	0.125081
40	0.122601
41	0.1238
42	0.126579
43	0.126039
44	0.125063
45	0.120336
46	0.123575
47	0.124412
48	0.122735
49	0.124408
50	1.169286

Lifelong Planning Astar Large Grid Data

	A
1	0.0002
2	1.2824
3	0.9671
4	0.4174
5	0.3823
6	0.2182
7	0.1984
8	0.1839
9	0.1837
10	0.1655
11	0.1837
12	0.2006
13	0.2837
14	0.2487
15	0.1837
16	0.1859
17	0.1802
18	0.3681
19	0.1997
20	0.1998
21	0.1841
22	0.3998
23	0.1993
24	0.3675
25	0.2316
26	0.3511
27	0.199
28	0.1837
29	0.2168
30	2.018
31	0.2998
32	0.0002
33	0.717
34	0.3501
35	0.3163
36	0.251
37	0.216
38	0.4843
39	0.3024
40	0.2962
41	0.2845
42	0.0017
43	0.6325
44	0.1836
45	0.2004
46	0.2178
47	0.1982
48	0.2004
49	0.1998
50	0.2178

Depth First Search Small Grid Data

	A
1	0.028
2	0.0548
3	0.0405
4	0.0066
5	0.8858
6	0.1629
7	0.2033
8	0.1751
9	0.2105
10	0.1621
11	0.2193
12	0.1796
13	0.1959
14	0.1734
15	0.1806
16	0.1702
17	0.2087
18	0.2022
19	0.3668
20	0.2042
21	0.2143
22	0.2462
23	0.2321
24	0.2183
25	0.1975
26	0.2247
27	0.2275
28	0.1964
29	0.2224
30	0.2186
31	0.2013
32	0.1922
33	0.2051
34	0.1864
35	0.1995
36	0.1948
37	0.1894
38	0.1923
39	0.1924
40	0.1993
41	0.1951
42	0.2174
43	0.2105
44	0.1902
45	0.1841
46	0.1658
47	0.2024
48	0.1913
49	0.1833
50	0.19

Depth First Search Medium Grid Data

id	val
1	0.1844
2	6.4758
3	0.6363
4	0.364
5	0.2801
6	0.2544
7	0.2033
8	0.3693
9	0.2056
10	0.3639
11	0.161
12	0.3703
13	0.2023
14	0.2413
15	0.1671
16	0.2047
17	0.3656
18	0.2105
19	0.1632
20	0.4405
21	0.1999
22	0.1664
23	0.5239
24	0.376
25	0.2293
26	0.2012
27	0.1633
28	0.201
29	0.2019
30	0.1985
31	0.2486
32	0.2426
33	0.2025
34	0.201
35	0.1652
36	0.4071
37	0.1648
38	0.2082
39	0.2074
40	0.2105
41	0.2108
42	0.6469
43	0.2846
44	0.2537
45	0.2423
46	0.2429
47	0.2056
48	0.2133
49	0.2443
50	0.2485

Depth First Search Large Grid Data

Table 3-4 Description Of Data Visualisation

4 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 initially is black and once the 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 your algorithm to find the paths time into an excel file in seconds so the user can then see the time for the algorithm to finish to better see the 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.

5 References

- Swift, N. (2020) *Easy A* (star) pathfinding*, Medium. Available at: <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2> (Accessed: April 24, 2023).
- Likhachev, M. *et al.* (2005) *Anytime D* - CMU school of computer science, Carnegie Mellon University School of Computer Science*. DARPA's MARS program, NSF Graduate Research Fellowship. Available at: <https://www.cs.cmu.edu/~ggordon/likhachev-et.al.anytime-dstar.pdf> (Accessed: April 24, 2023).
- Stentz, A. (1994) *The D* Algorithm for real-time planning of optimal traverses*. Available at: https://www.ri.cmu.edu/pub_files/pub3/stentz_anthony_tony_1994_2/stentz_anthony_tony_1994_2.pdf (Accessed: April 24, 2023).
- Lu, J. *et al.* (2022) *Jump point search algorithm*, Encyclopedia. Jiakai Lu. Available at: <https://encyclopedia.pub/entry/24246> (Accessed: April 24, 2023).
- Raheem, F.A. and Hameed, U.I. (2018) *Path planning algorithm using D* heuristic method based on PSO in ...*, American Academic Scientific Research Journal for Engineering, Technology, and Sciences. Available at: <https://core.ac.uk/download/pdf/235050716.pdf> (Accessed: April 24, 2023).
- Harabor, D. and Grastien, A. (2011) *(PDF) improving jump point search - researchgate*. Available at: https://www.researchgate.net/publication/287338108_Improving_jump_point_search (Accessed: April 24, 2023).
- Rachmawati, D. and Gustin, L. (2020) *Analysis of Dijkstra's algorithm and a* algorithm in ...* - iopscience. IOP Publishing Ltd. Available at: <https://iopscience.iop.org/article/10.1088/1742-6596/1566/1/012061> (Accessed: April 24, 2023).
- 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> (Accessed: April 24, 2023).
- Pathak, M.J., Rami, S.P. and Patel, R.L. (2018) *Comparative analysis of search algorithms - ijcaonline.org*. International Journal of Computer Applications. Available at: <https://www.ijcaonline.org/archives/volume179/number50/pathak-2018-ijca-917358.pdf> (Accessed: April 24, 2023).
- Pandey, K.K. and Kumar, N. (2017) *A comparison and selection on basic type of searching algorithm in data ...*, researchgate. Available at: https://www.researchgate.net/publication/308119139_A_Comparison_and_Selection_on_Basic_Type_of_Searching_Algorithm_in_Data_Structure (Accessed: April 24, 2023).
- Foad, D. *et al.* (2021) *A systematic literature review of a* pathfinding*, Procedia Computer Science. Elsevier. Available at:

<https://www.sciencedirect.com/science/article/pii/S1877050921000399> (Accessed: April 24, 2023).

Koenig, S., Likhachev, M. and Furcy, D. (2004) *Lifelong planning a**, *Artificial Intelligence*. Elsevier. Available at: <https://www.sciencedirect.com/science/article/pii/S000437020300225X> (Accessed: April 24, 2023).

Koenig, S. and Likhachev, M. (2002) *D* lite* - *idm-lab.org*. Available at: <http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf> (Accessed: April 24, 2023).