# **159740 Studies in Intelligent Systems**

**Robot Path-Planning Algorithms**

**for**

**Fully Dynamic Shortest Path Problems**

**Bilesha Gunasekara**

**14096264**

**Table of contents**

**1.0 Introduction ............................................................................................................3**

**2.0 Algorithm Information……………….......................................................................3**

**3.0 Skeletal Code ..........................................................................................................4**

**4.0 SnapShots………………………………………….…………………………………….10**

**5.0 Tabulation of Results…………………………………………………………………..15**

**6.0 Discussion………………………………………………………………..……………...15**

**7.0 User Guide………………………………………………………………………….…....17**

**1.0 Introduction**

Intelligent artificial systems are utilised in many different technologies and industries to achieve specific duties. Machine vision, robots, machine learning, object identification are such areas where intelligent systems are widely used. It is important to understand such technologies to further refine and improve their capabilities for important situations.

This report will be specifically looking into the development and testing of intelligent algorithms that are capable of traversing a path with a start and end goal which may or may not have obstacles on the way. Lifelong Planning and D\* Lite were the two algorithms that were chosen and developed for this assignment.

**2.0 Algorithm Information**

Most of the data structures utilized in this assignment were already in the skeletal framework. I did not add new data structures to my algorithms. The main default data structure that was present and utilized was the vector structure. The vector structure was capable of handling the information within the cells. This includes the start and goal cells, successors and predecessors, the g and rhs values etc. The other data structure that was used is the heap type which is used for the priority queue.

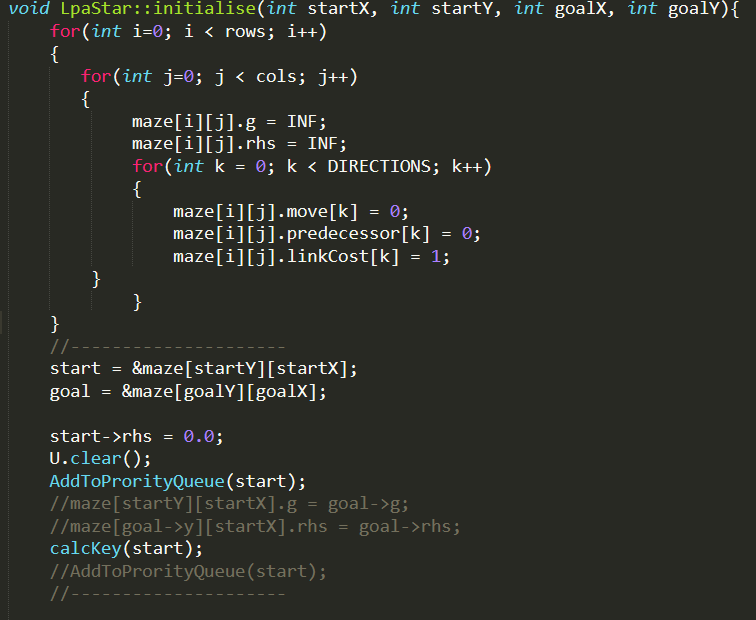
The logic that was displayed by the pseudocode from the lectures was followed and implemented step by step. The initialise function was edited from the original to make it more flexible. On top of the i and j values from the maze, the move, predecessor and link cost were initialized for reference.

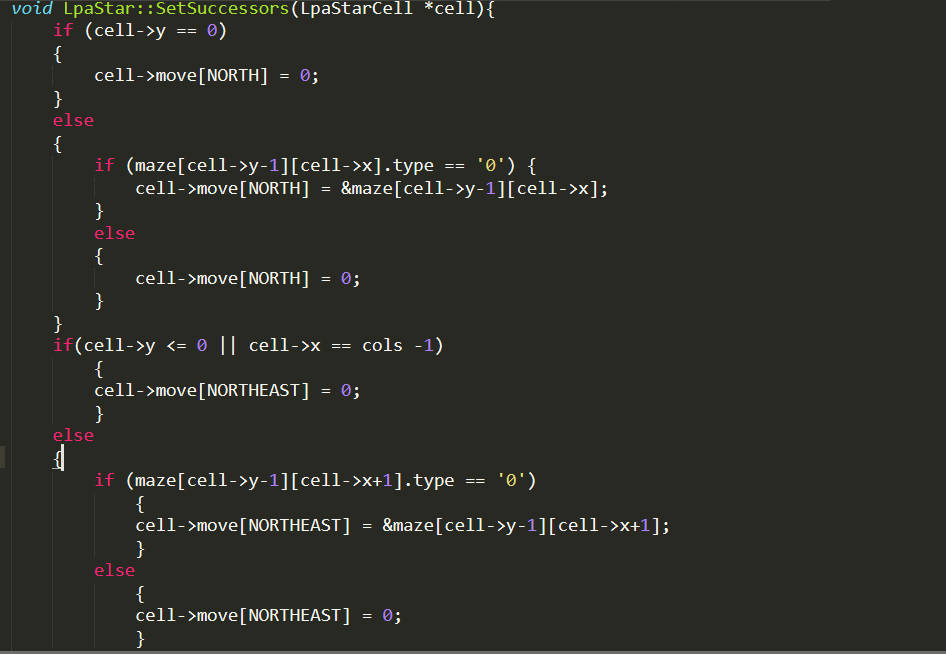
Functions were created to traverse through the cells in the grid mathematically for both the successors and predecessors. This was done using if and else statements to compare the x,y and type values of each cell and using the rows and columns as boundaries. A function for identifying the predecessors of each cell was also computed with similar logic as the D\* lite requires the information of the predecessors for the algorithm to run.

The main function has keys enabled for F7. F8, F11, F12 D and A keys to enable the respectful functions. A timer function is within the main function to record the total time it took for the algorithm to reach the end destination. Int keys were also created in the class to record the values for state space, vertex count and path distance.

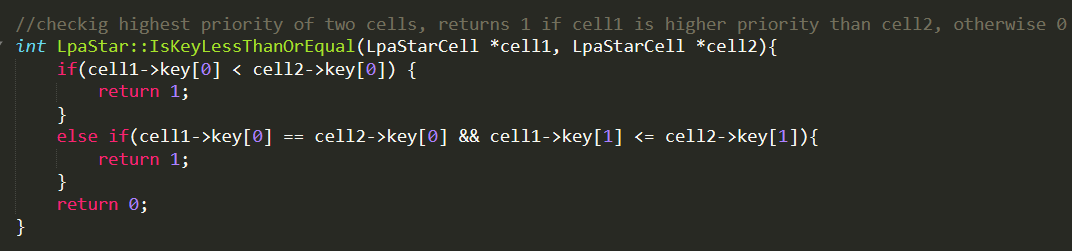
The rest of the code followed the step by step logic that was already in the pseudo code which was the update vertex, finding the shortest path and main functions. The display path function was edited for both LPA and D\*Lite.

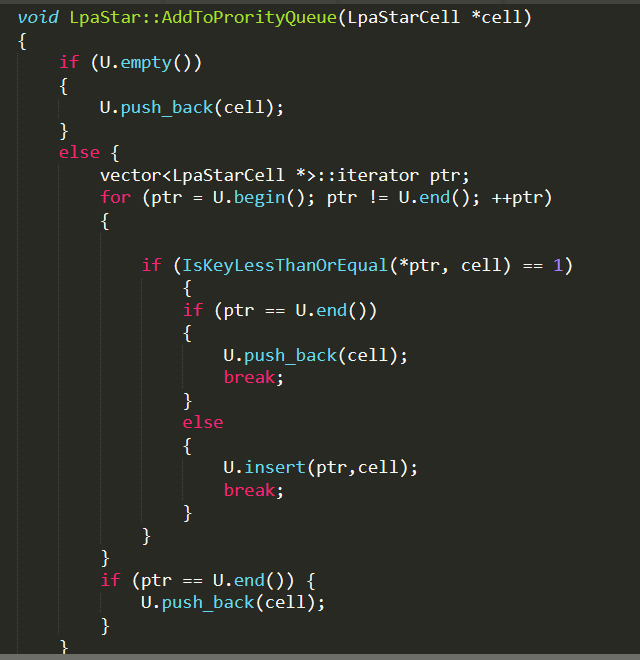
**3.0 Skeletal Code**

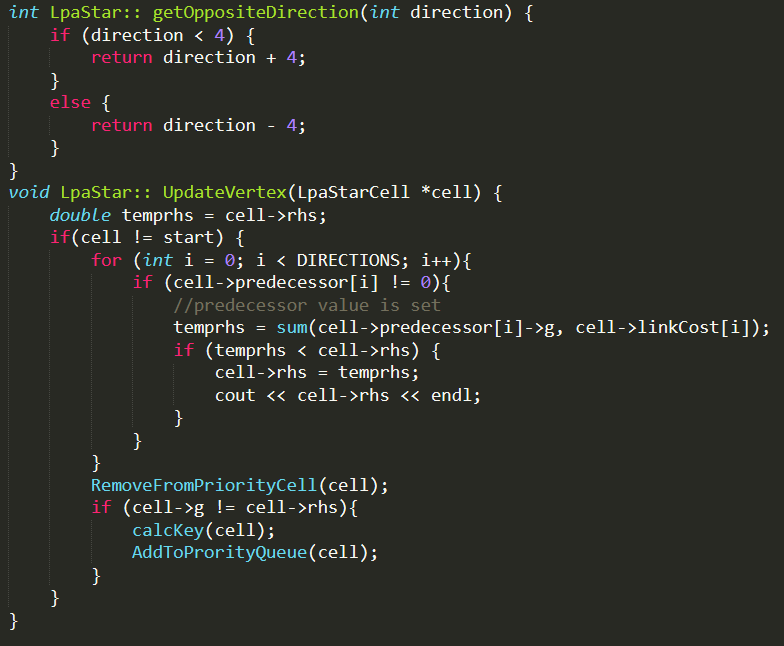
****

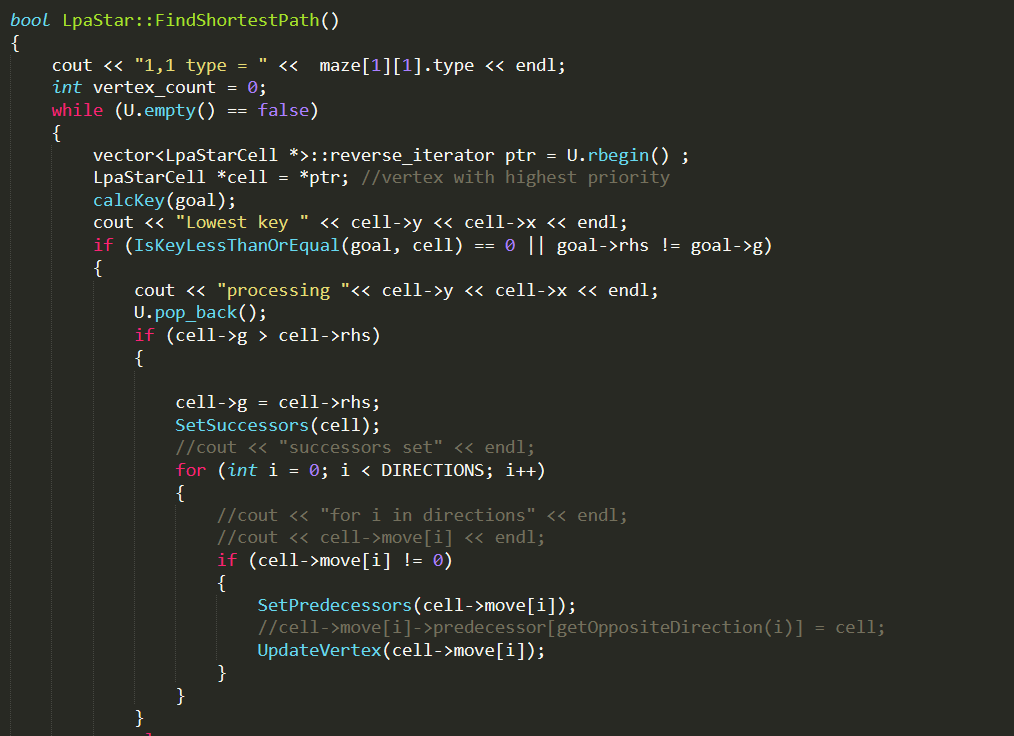
****

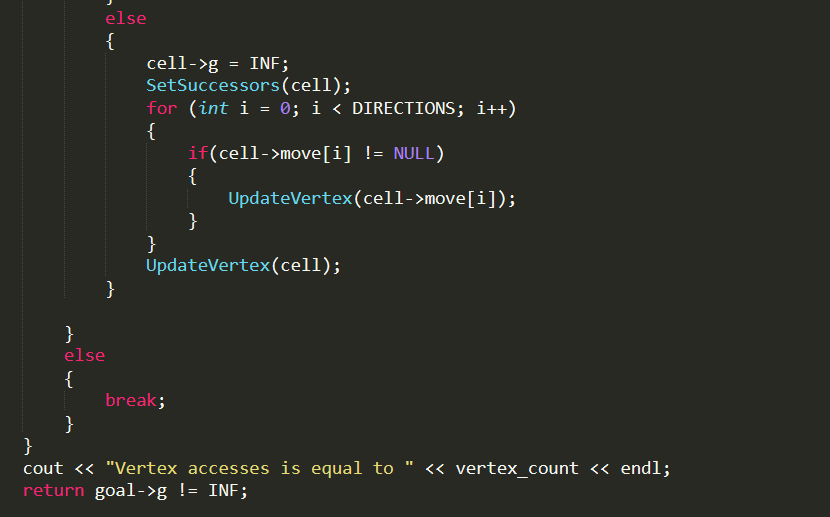
This code continues this cycle for all 8 directions. It will take too much space to post the entire function as it basically repeats the logic for each respective direction.

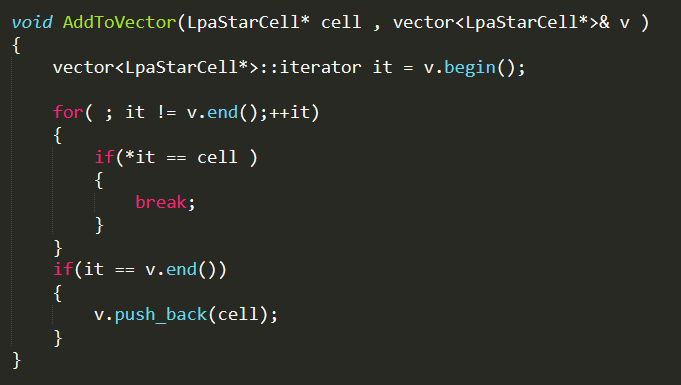




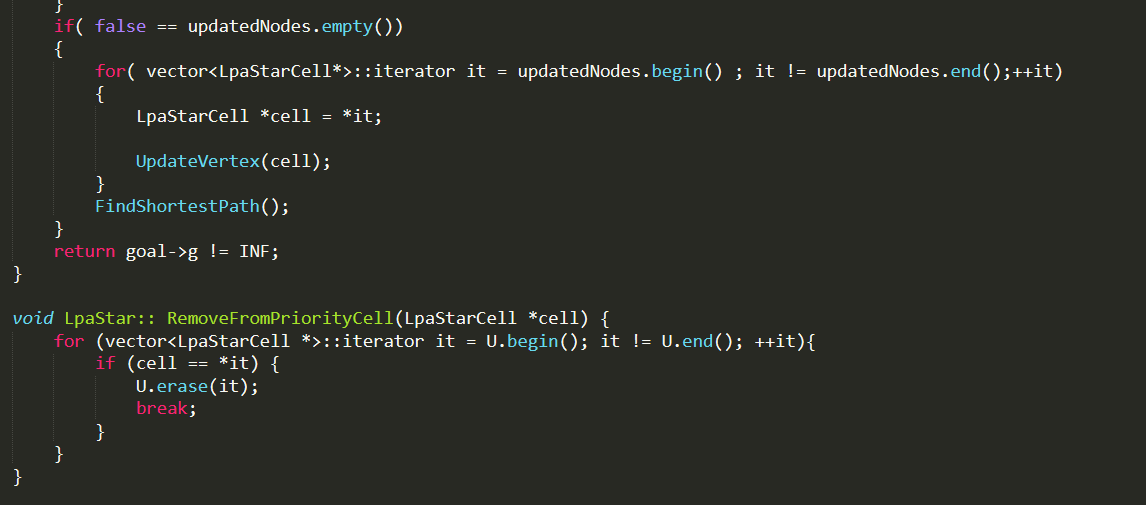






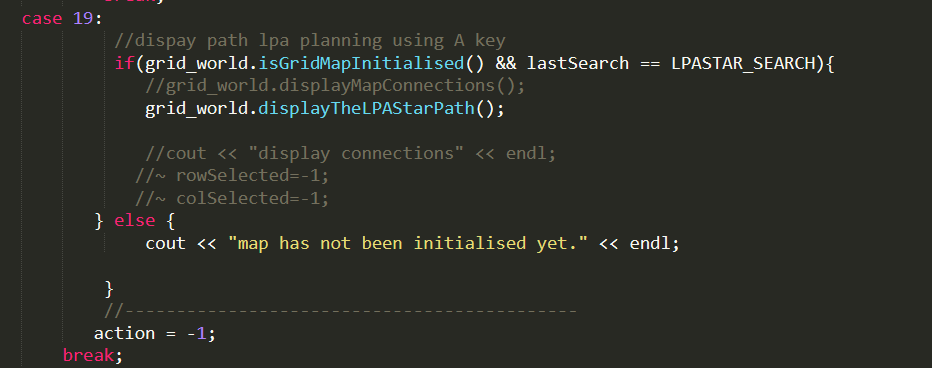




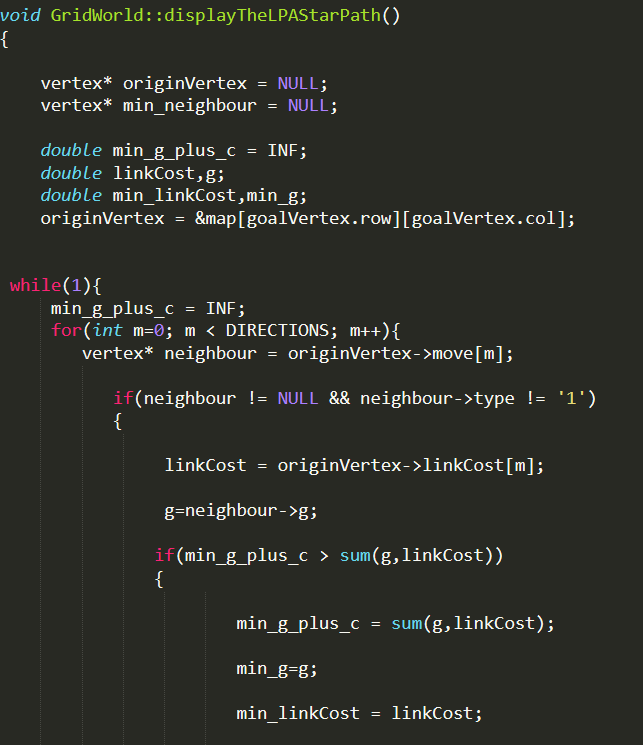


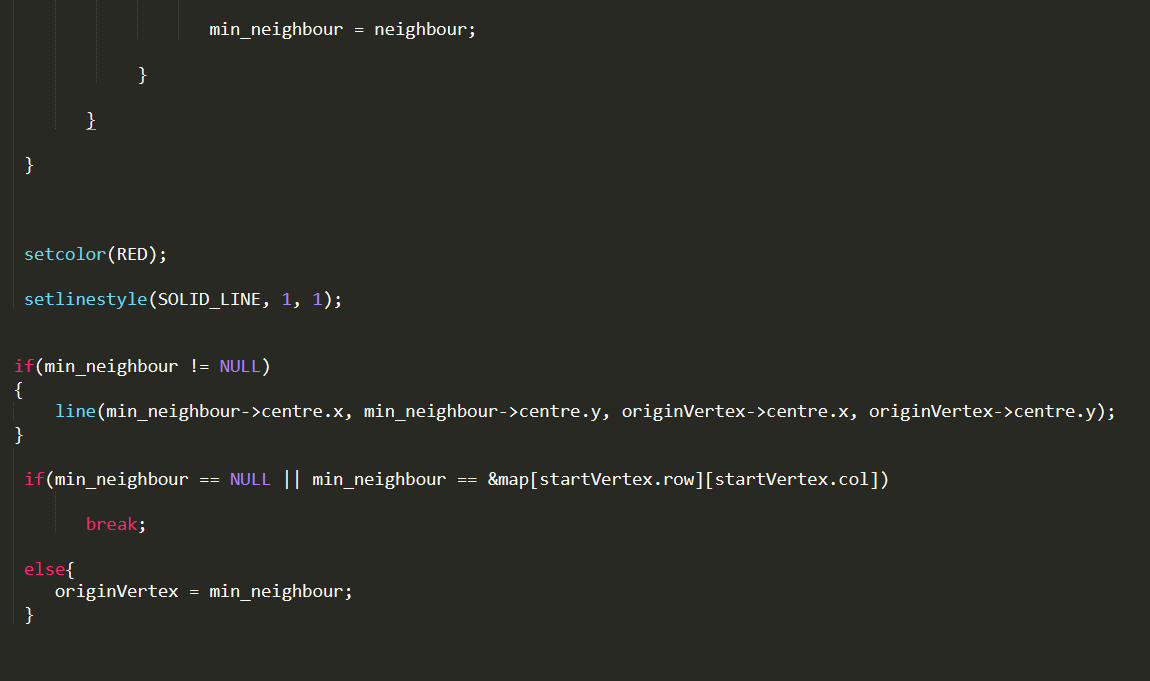
Main function calling the LPA star algorithm.





Find the shortest path function for LPAStar.

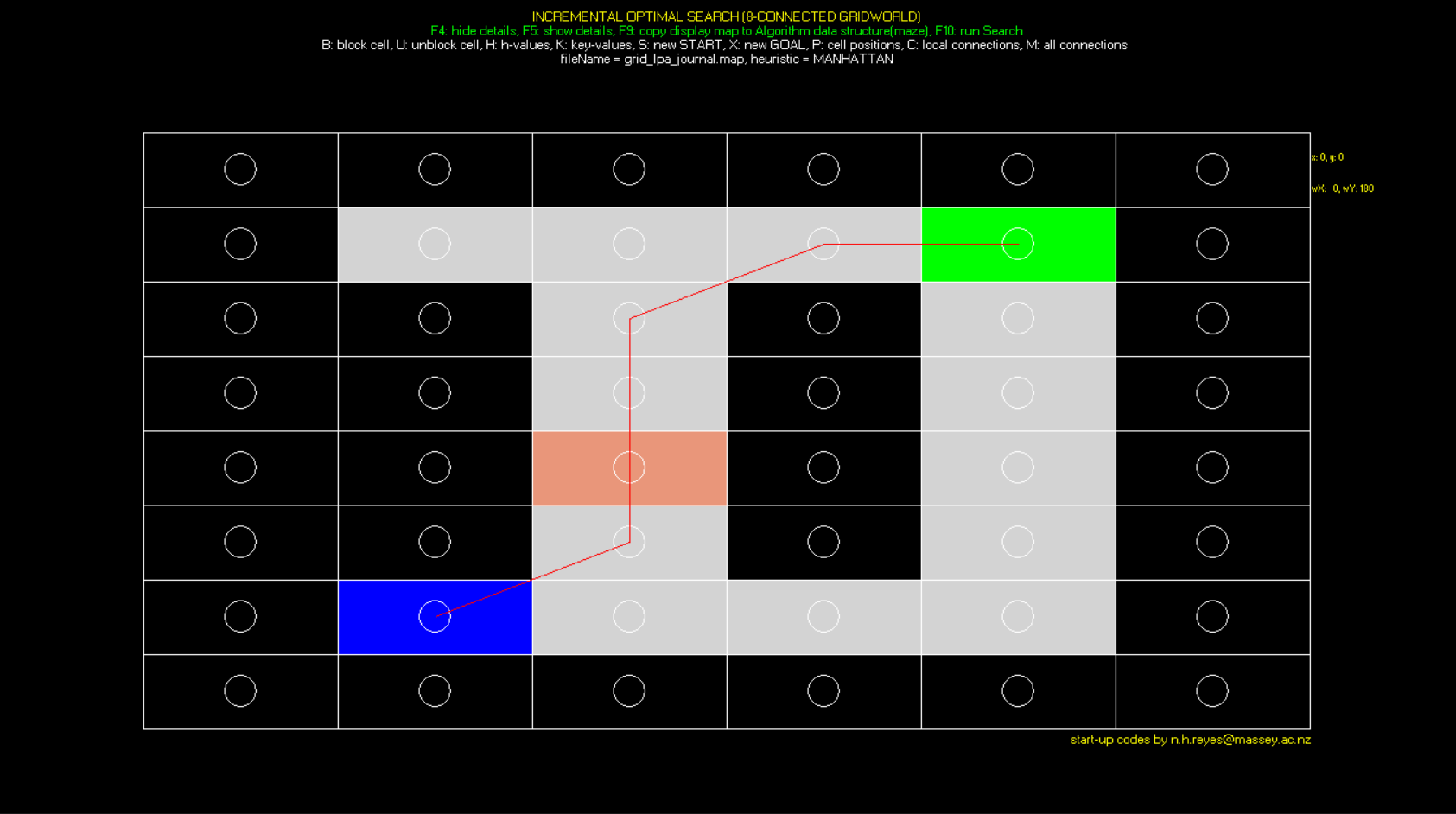


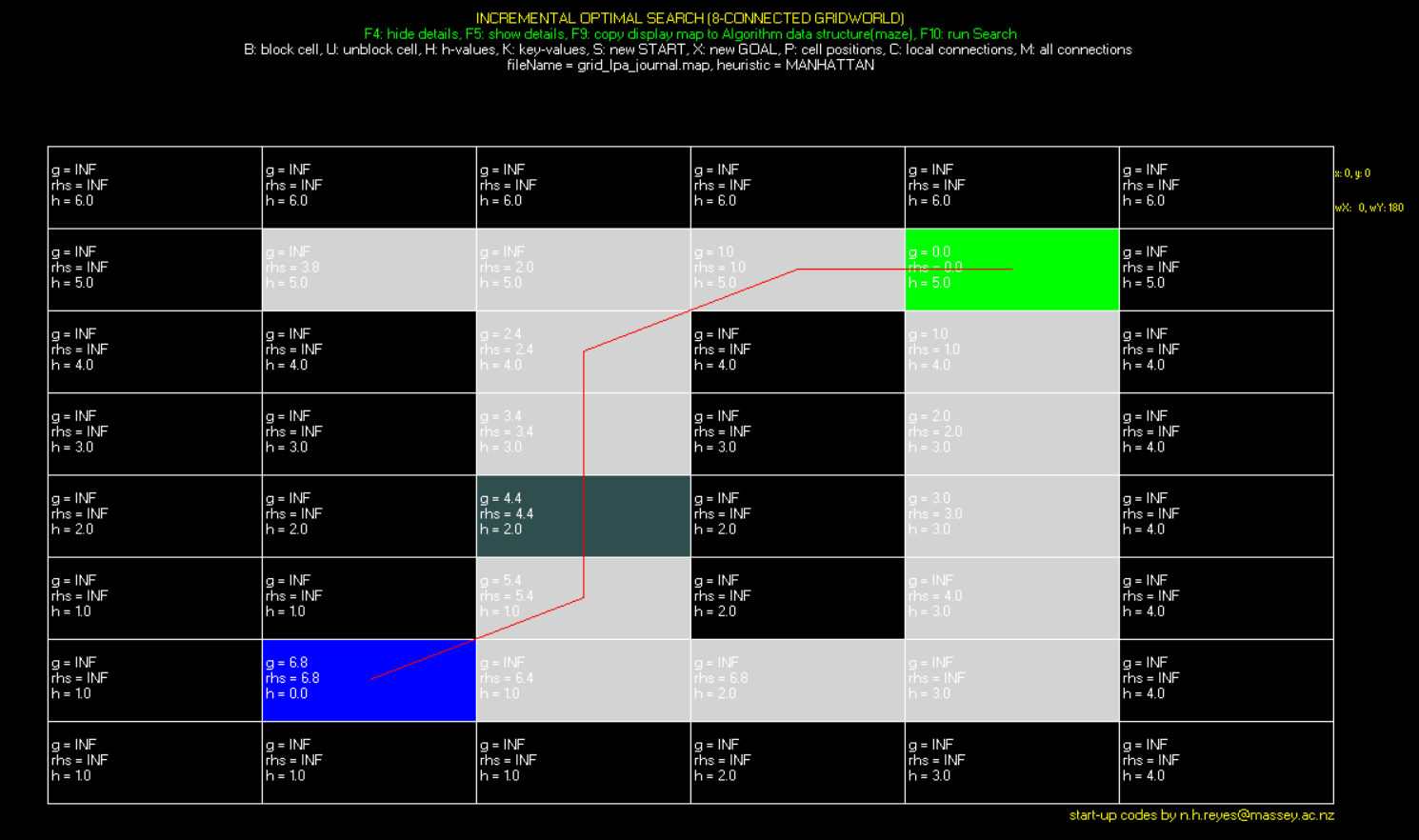


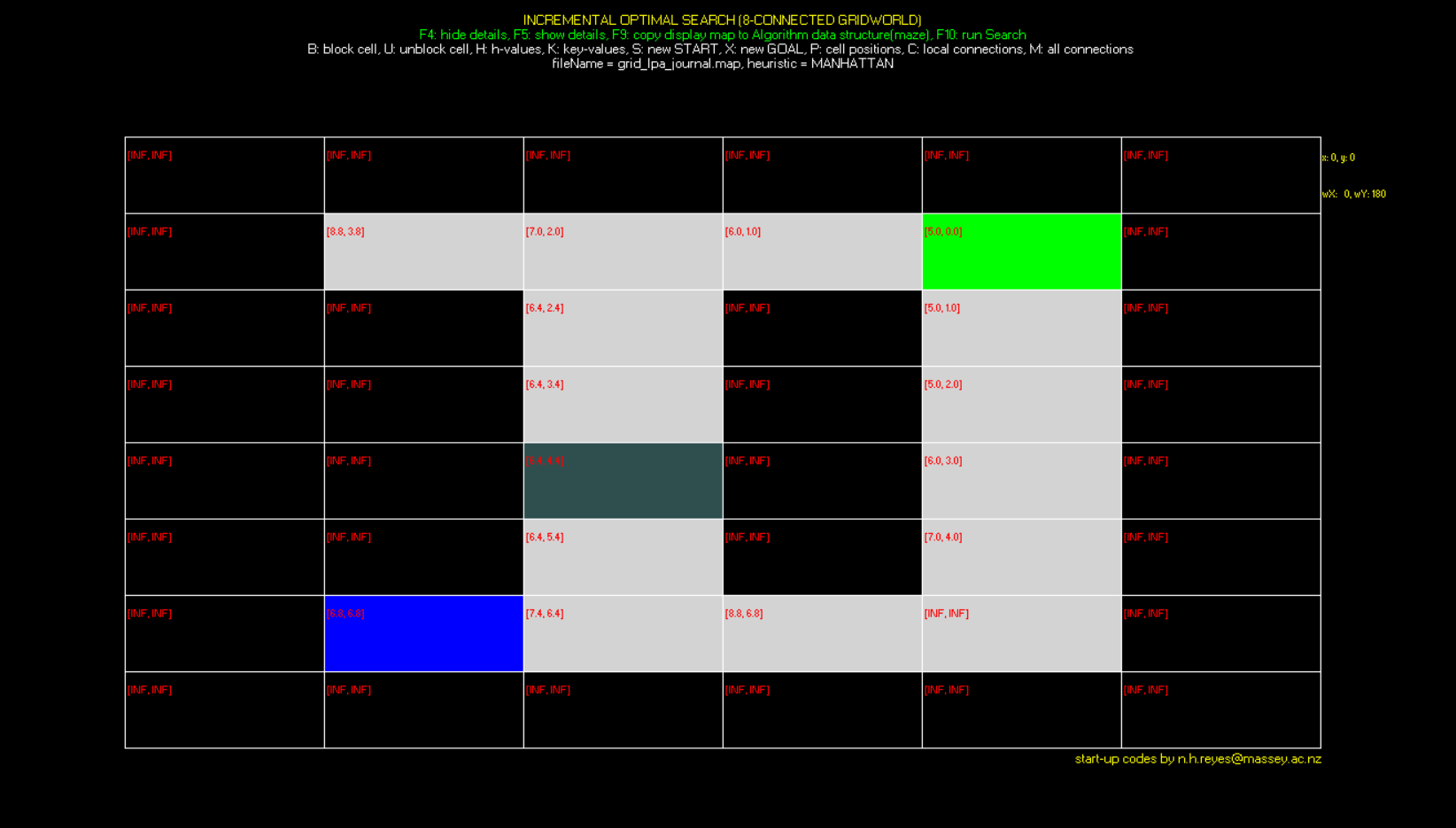
The D\* Lite’s skeletal code is not posted with screencaps as it will take a lot of space and the majority of the code is very similar to the LPA planning algorithm. The logic has been changed around from the pseudocode to suit D\* Lite.

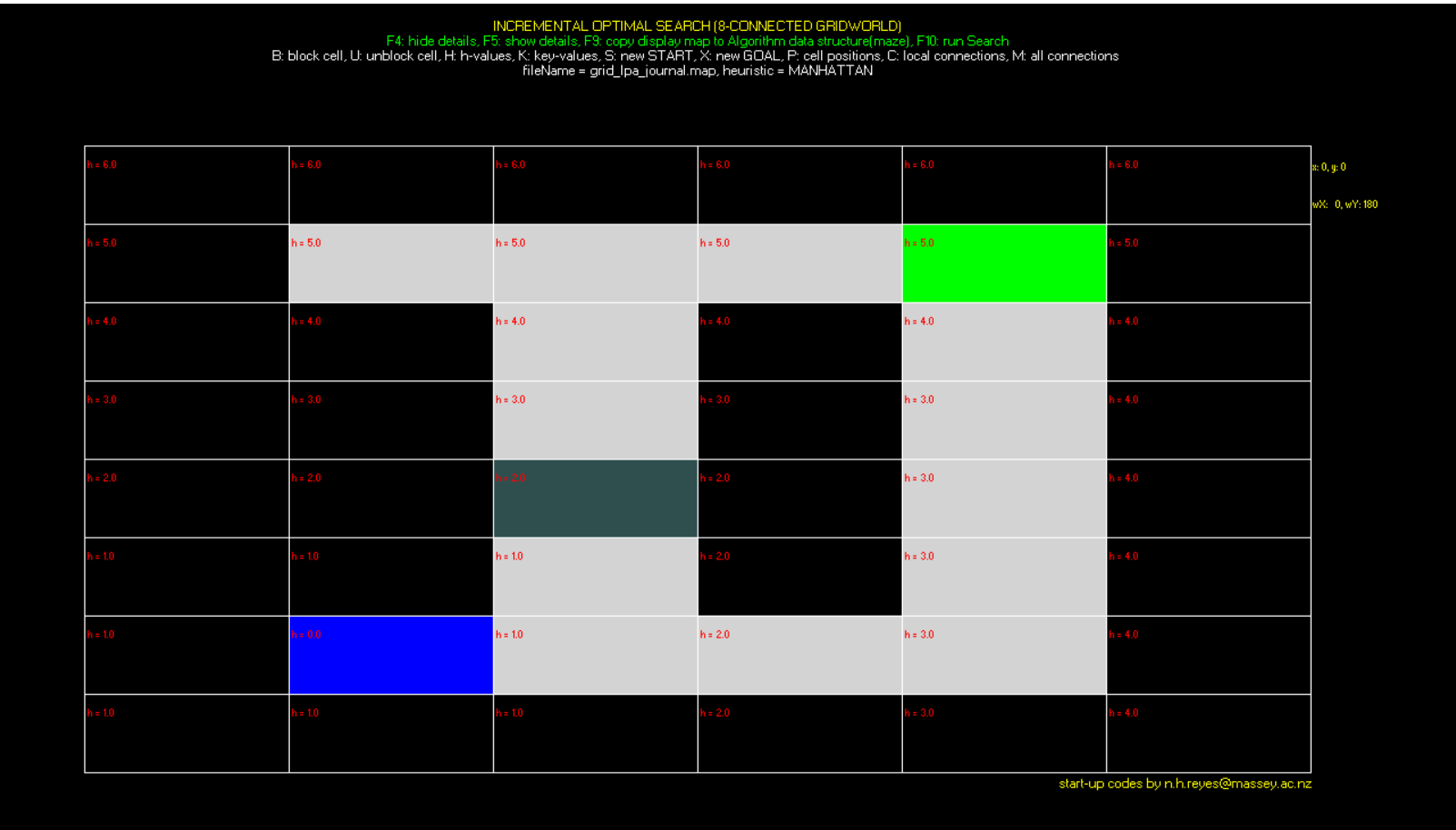
**4.0 SnapShots**

**Grid LPA Journal**

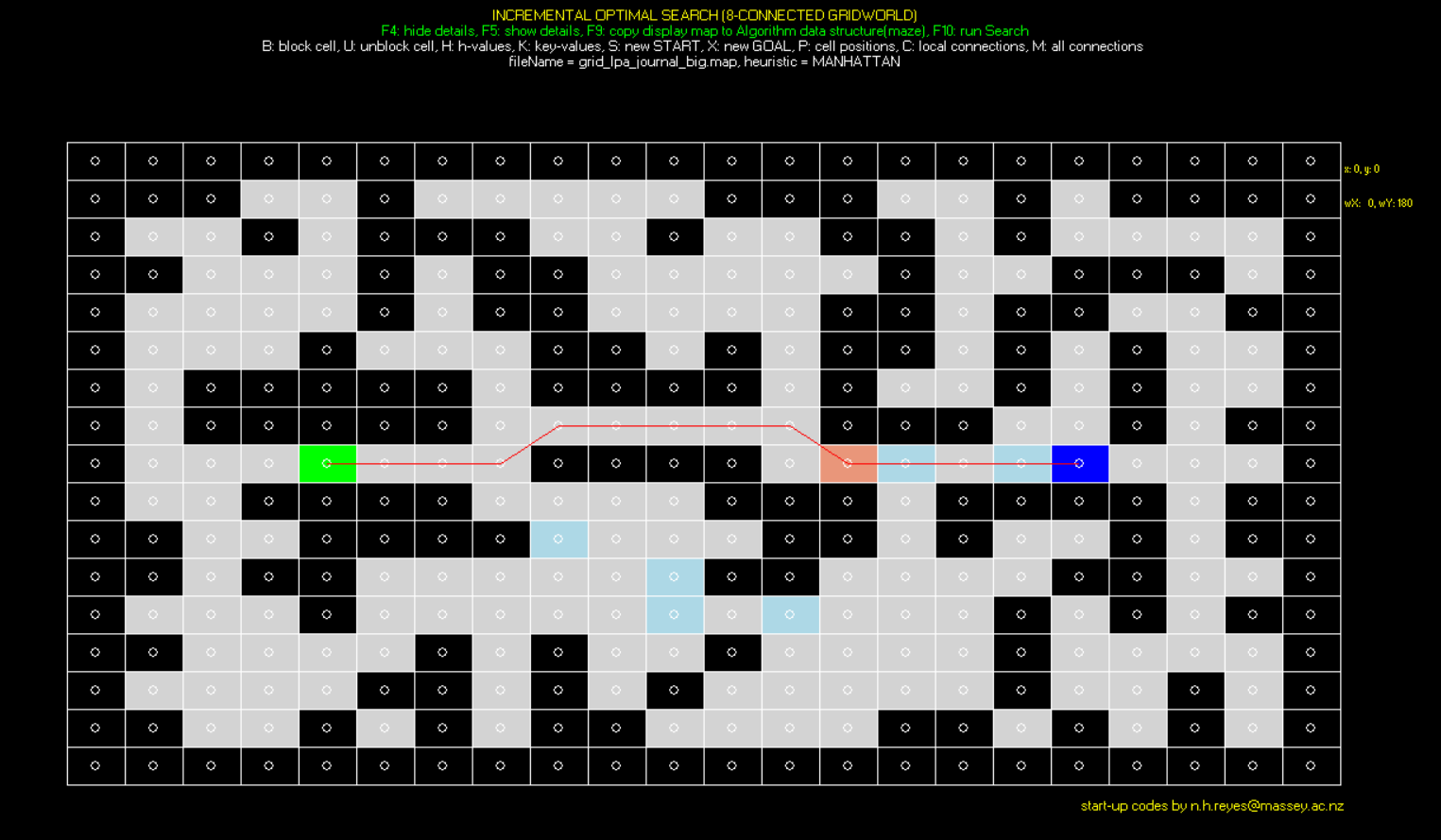
****

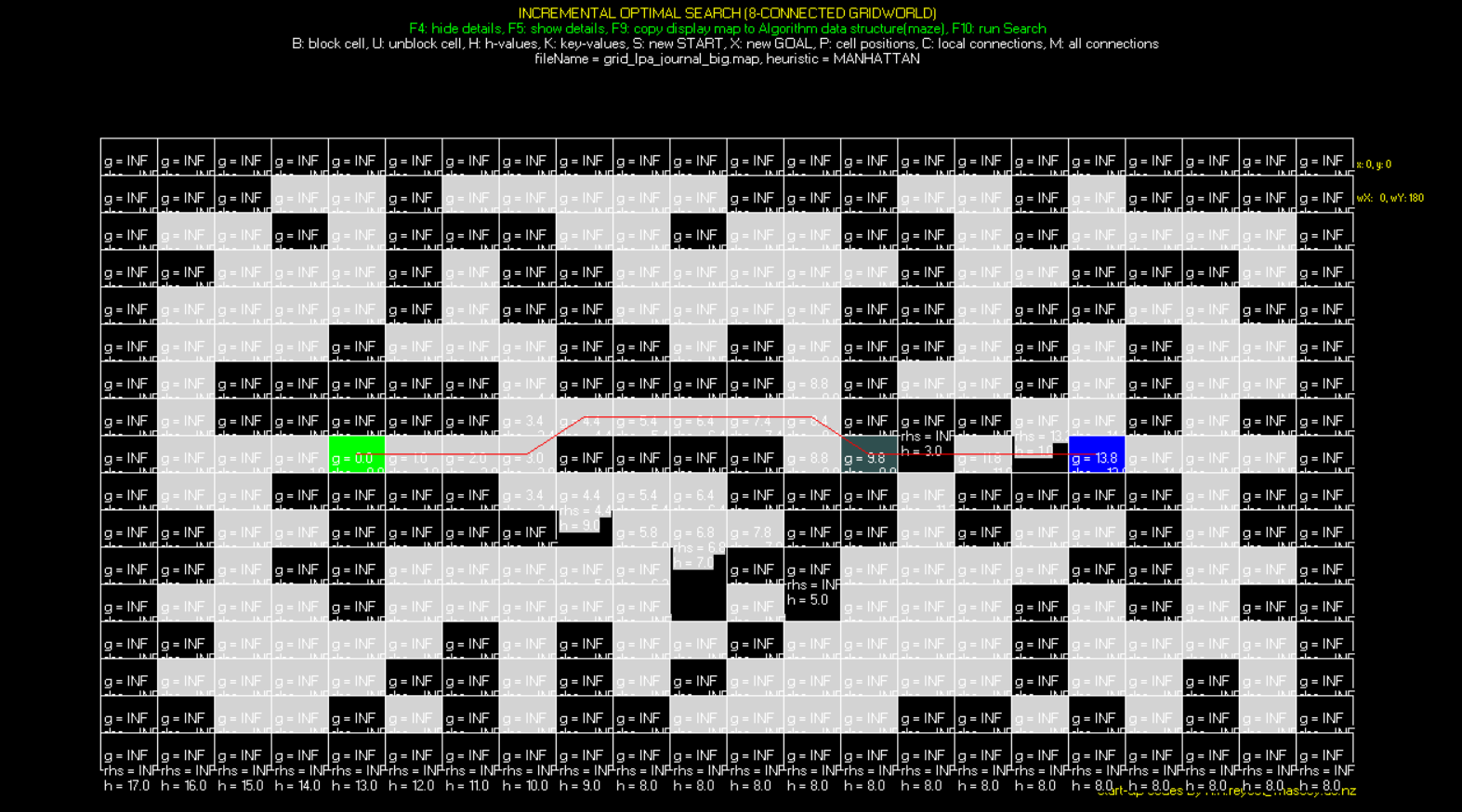
****

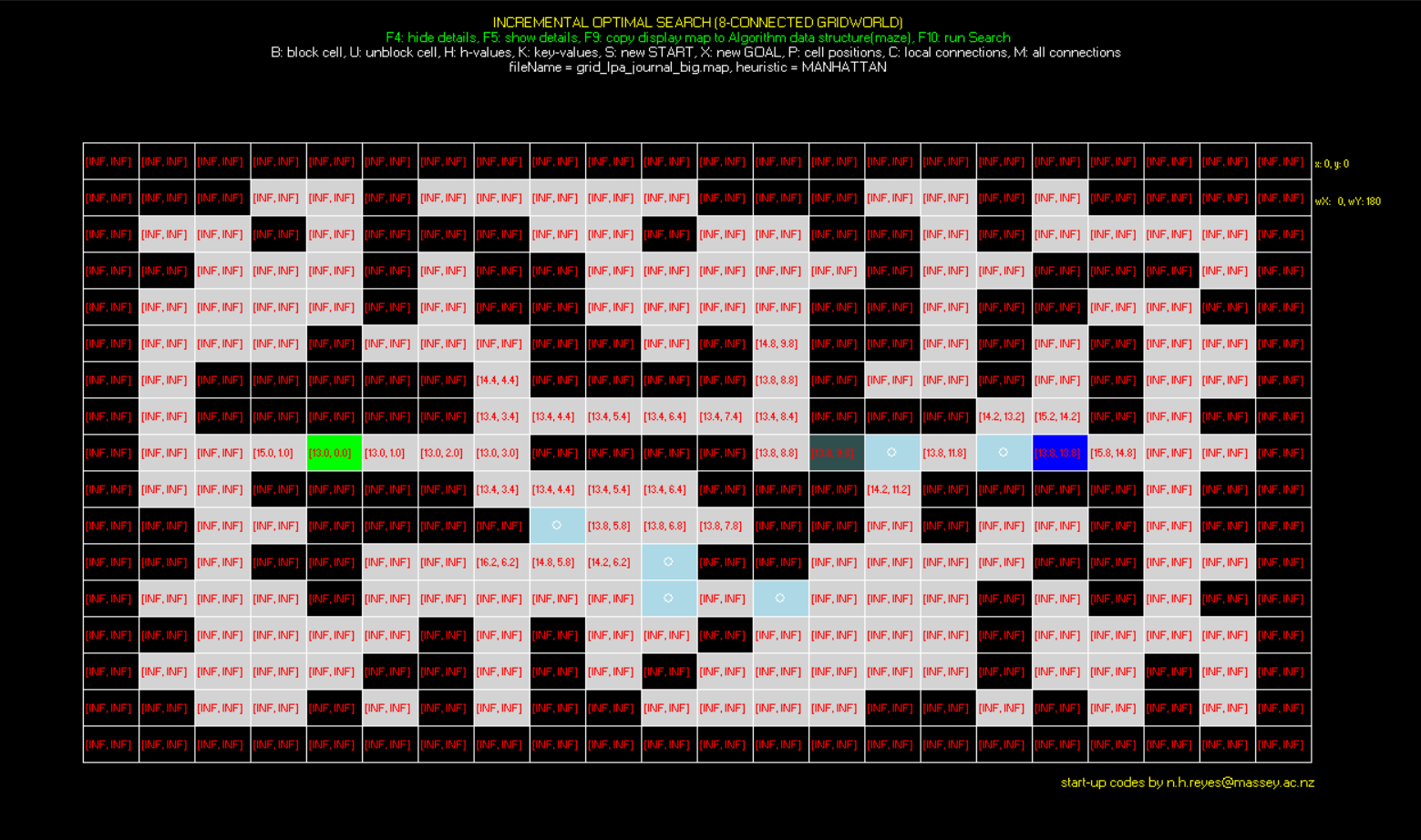
****

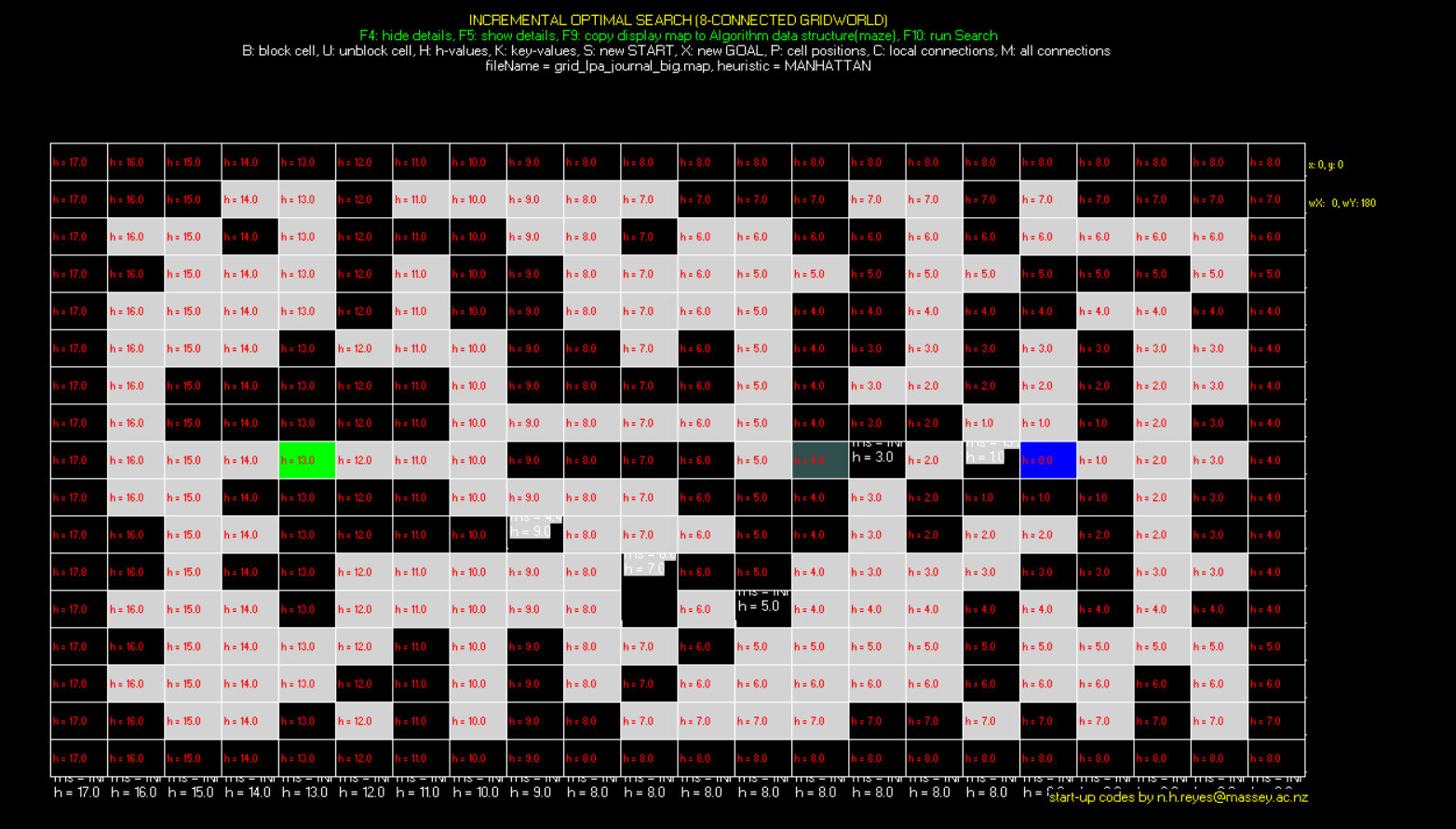
****

**Grid LPA Journal Big**

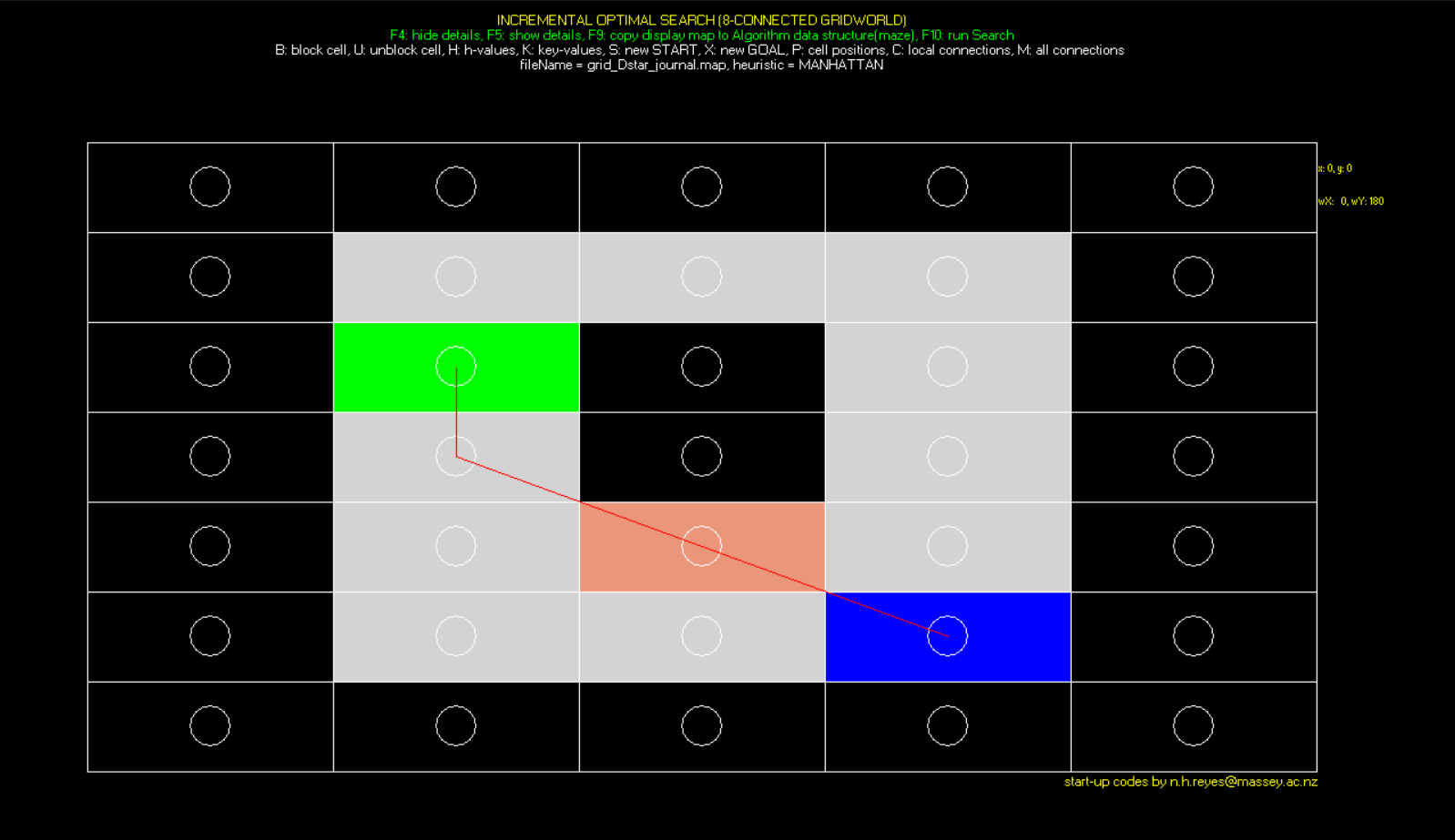
****

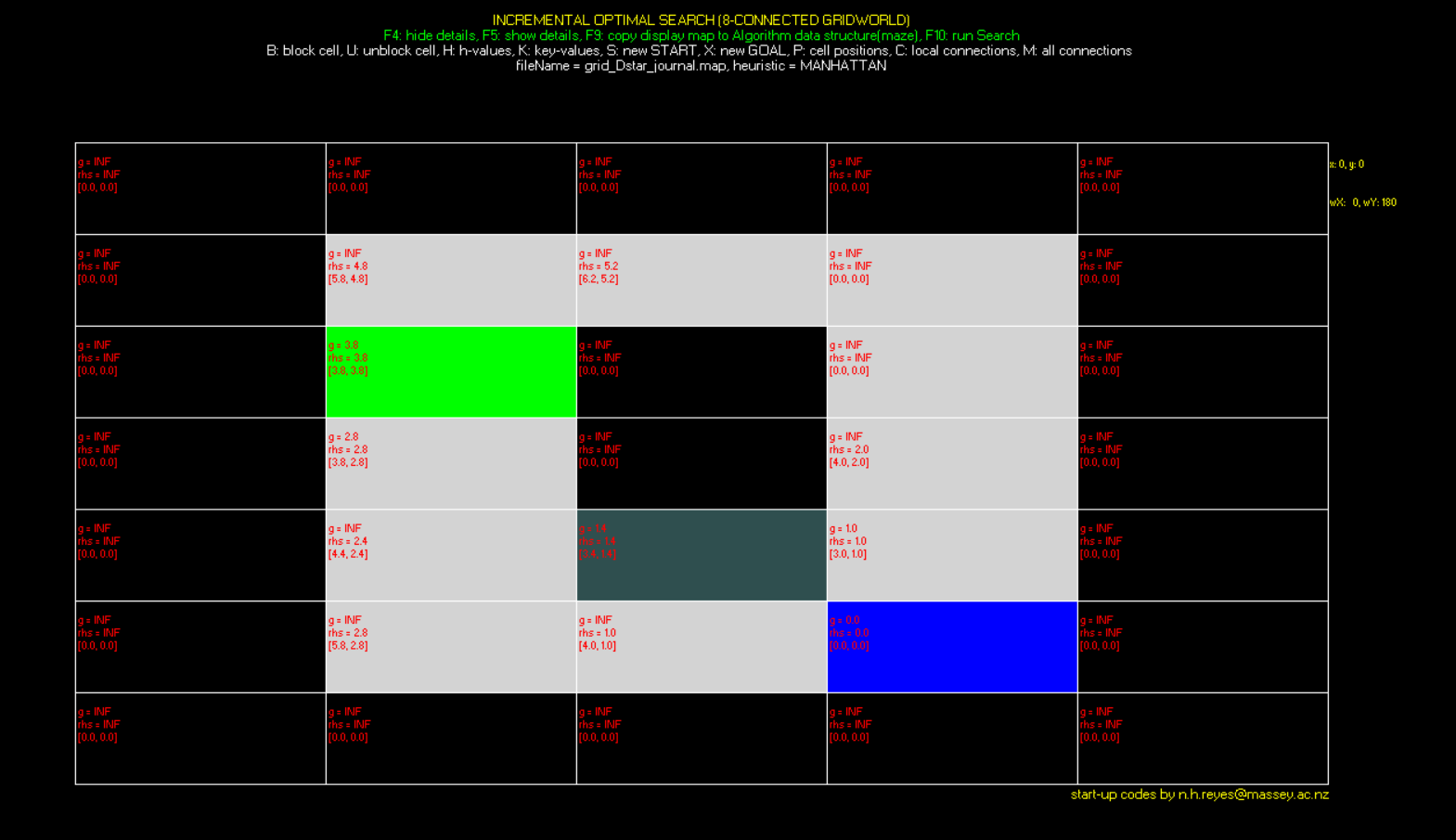
****

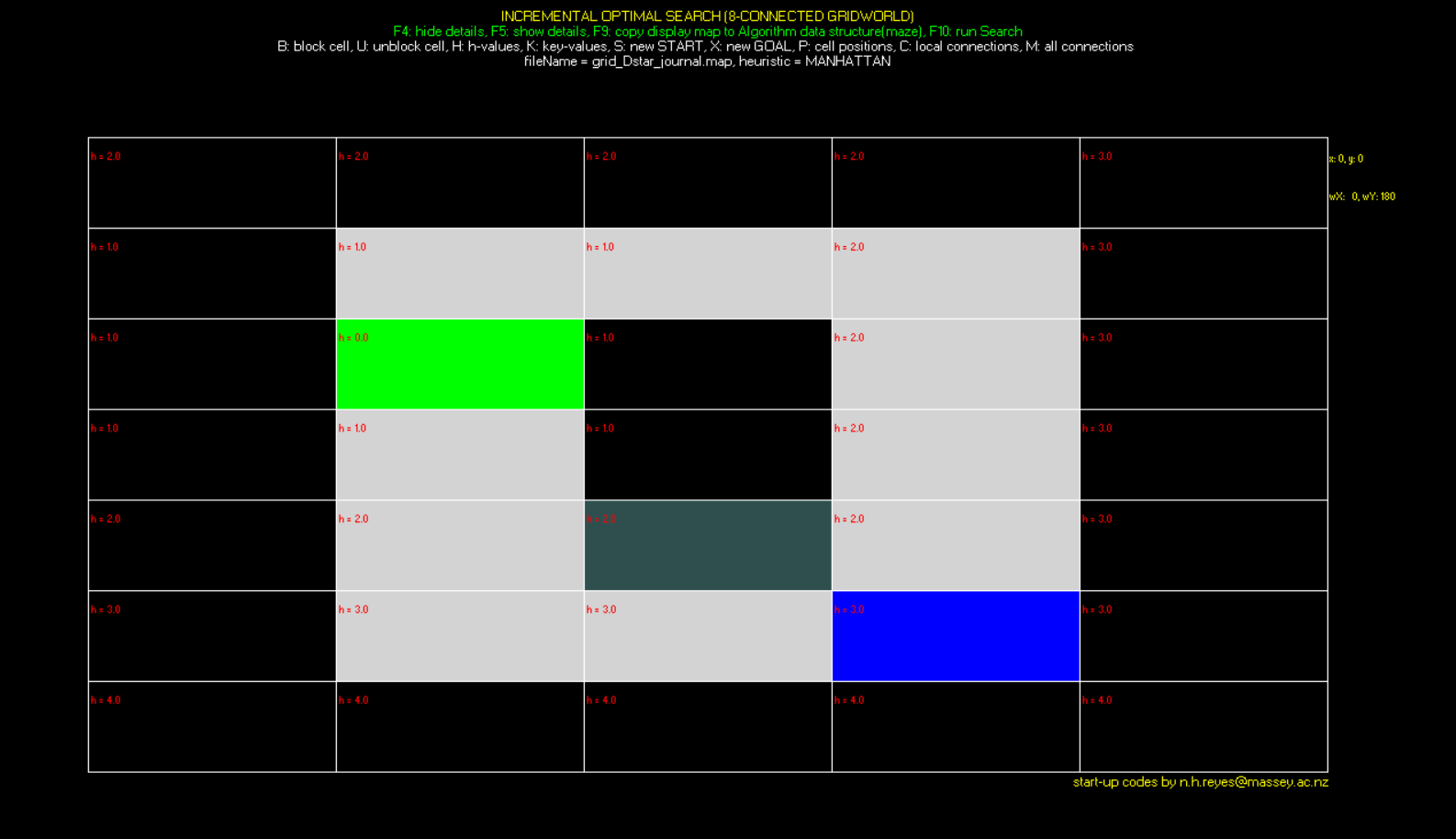
****

****

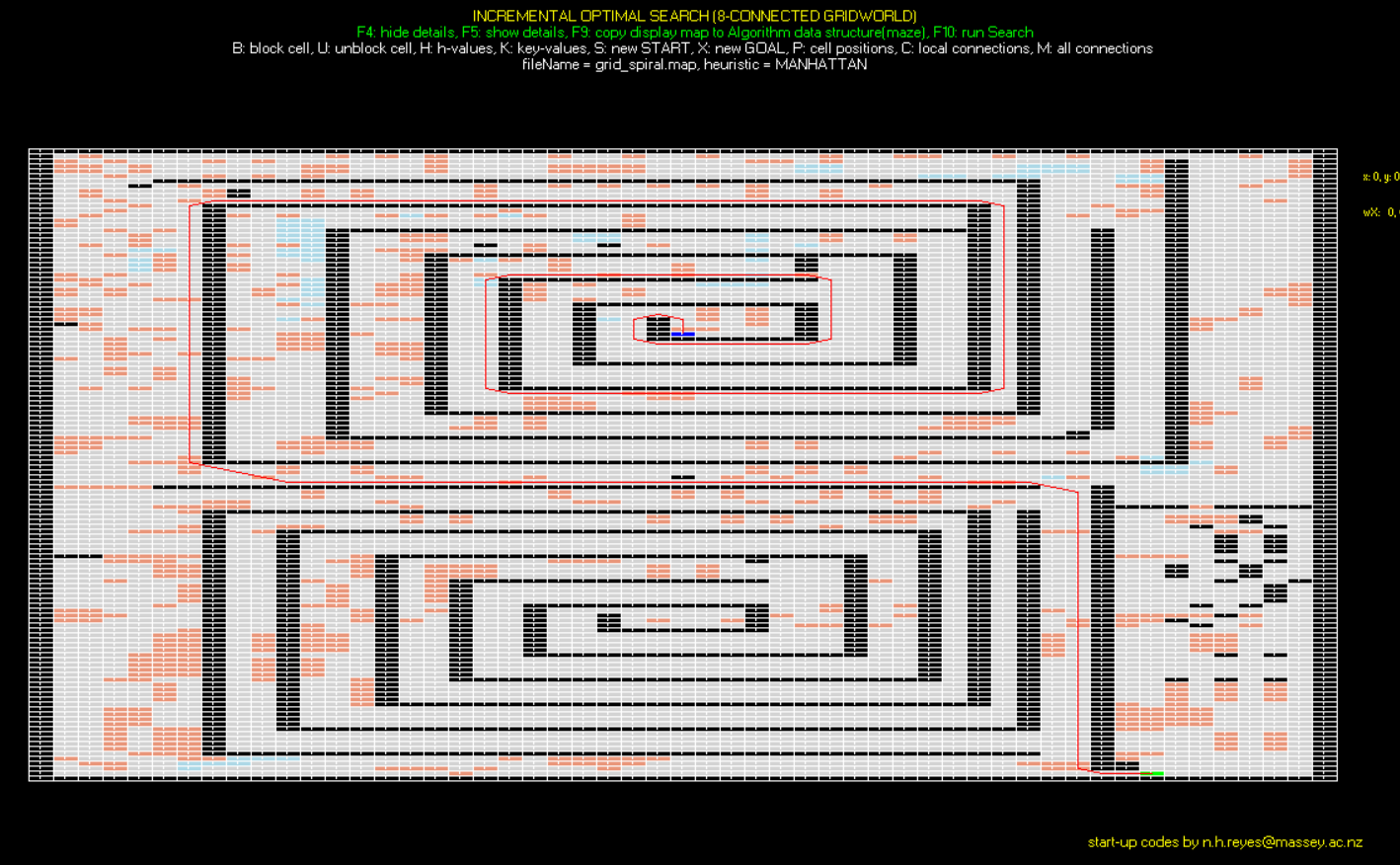
**Grid Dstar Journal**

****

****

****

**Grid Spiral**



**5.0 Tabulations of Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Max. Queue length** | **Path length** | **No. of state expansions** | **Vertex accesses** | **Actual Running Time (msec.)** |
| **Lifelong Planning A**  **Grid LPA Journal** | **6** | **6.828** | **27** | **200** | **0.001seconds** |
| **D\* Lite Final Version**  **D\* Lite Journal** | **11** | **13.828** | **74** | **576** | **0.003seconds** |
| **Lifelong Planning A**  **Grid LPA Journal Big** | **11** | **13.828** | **100** | **792** | **0.001seconds** |
| **Lifelong  Planning A**  **Grid Spiral** | **144** | **310.038** | **31877** | **254984** | **0.034seconds** |
| **D\* Lite Final Version**  **Grid Spiral** | **127** | **310.038** | **16523** | **132610** | **0.031seconds** |

**6.0 Discussion**

The LPA algorithm seems to be working perfectly as I could not find any mistakes or bugs when I was testing it out on grids. The D\* Lite is also working perfectly but I encountered bugs and errors when I was testing the replanning algorithm. The replanning portion would not respond during certain test cases and this would be due to simple error and logic malfunctions within the algorithm. Overall, the algorithm works well but if it was too perfect and precise, a lot more time and work would be required to identify all the bugs and errors within the algorithm. Another reason is probably because I followed the basic D\* Lite pseudocode and not the final version. My algorithm is lacking the constant K value and theoretically, that constant K value plays a heavy role in increasing efficiency from the LPA algorithm.

Another issue is the GUI restarts after the first run and I believe this error is due to the initialize function being called twice within the main function. I attempted with the removal of one of the initialize functions but the entire would not work after the removal so I reversed my actions and left both initializations in place. The start cell is also very buggy as the display disappears if it is chosen before the goal cell. To combat this, it is recommended that the goal cell is selected before the start cell.

For my tabulated results, I am not sure if my numbers are fitting and answering the context is asked for. I only tested both my algorithms for the spiral grid. On my first test, it took over a minute for the D\* Lite to find the shortest path while it took much less for the LPA. However on my second try, both algorithms had a very similar time at around 0.0031 and 0.0034 seconds respectively. The other differences in numbers is that the D\* Lite has half the number of vertexes accessed and close to half the number of state expansions when compared to LPA. The path length is the same for both but the max queue length is shorter for the D\* Lite. These values show that the D\* Lite was more efficient at tracing the shortest path for the spiral grid. This also follows the theoretical side of the algorithms as the D\* Lite is a refined and upgraded version that is built off the LPA. The values would also have a higher disparity if the constant K was included within the D\* Lite’s algorithm which would further demonstrate the increased efficiency of the D\* Lite’s algorithm when compared to the LPA.

**7.0 User Guide**

F4 = Hide details

F5 = Show details

F7 = Choose LPA Planning algorithm

F8 = Choose D\* Lite algorithm

F9 = Copy display map to algorithm data structure

F10 = Run Search

F11 = Replanning for D\* Lite algorithm

F12 = Replanning for LPA planning algorithm

D = Display shortest path for D\*lite algorithm (follow up from F8)

A = Display shortest path for LPA planning algorithm (follow up from F7)

H = display heuristic values

B = Block a cell

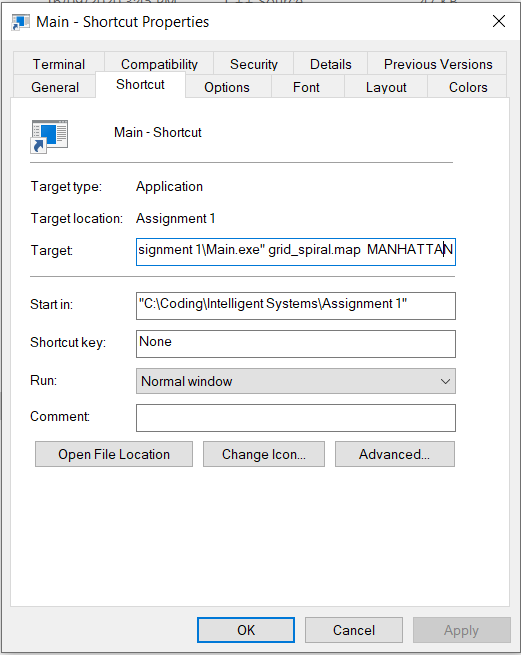
U = Unblock a cell

K = Display key values

S = Set start goal

X = Set end goal

The grid map is changed around by accessing the properties file on the shortcut of the main.exe. The heuristic is also swapped around with the same method.



The set start goal function is very buggy and only displays when the end goal is already constructed. For effective use, set the end goal before the start goal. From my testing, the program seems to work despite the start cell not being displayed.