

Computer Games Development

Software Functional Specification

Year IV

Donal Howe

C00249662

[Draft]

24/04/2023

**DECLARATION**

Text, table

Description automatically generated

**Table Of Contents**

[1 Introduction 1](#_Toc133343859)

[2 Technical Design 2](#_Toc133343860)

[2.1 Introduction 2](#_Toc133343861)

[2.2 Header Files 2](#_Toc133343862)

[2.3 Data structures used 19](#_Toc133343863)

[2.4 Storing of Data 20](#_Toc133343864)

[2.4.1 How is the Data Stored? 20](#_Toc133343865)

[2.4.2 When is the Data Stored? 20](#_Toc133343866)

[3 User Flow 21](#_Toc133343867)

[4 Class Diagram 22](#_Toc133343868)

[5 CRC Cards 23](#_Toc133343869)

[6 Sequence Diagram 26](#_Toc133343870)

[7 Technologies 27](#_Toc133343871)

###### List Of Figures

[Figure 3‑1 Class Diagram 22](#_Toc133343903)

[Figure 4‑1 CRC Card "Grid" 23](#_Toc133343904)

[Figure 4‑2 CRC Card “Game" 23](#_Toc133343905)

[Figure 4‑3 CRC Card "Menu" 23](#_Toc133343906)

[Figure 4‑4 CRC Card "Astar" 23](#_Toc133343907)

[Figure 4‑5 CRC Card "Dstar Lite" 24](#_Toc133343908)

[Figure 4‑6 CRC Card "Dijkstras" 24](#_Toc133343909)

[Figure 4‑7 CRC Card "Depth First Search" 24](#_Toc133343910)

[Figure 4‑8 CRC Card "Lpa Star" 24](#_Toc133343911)

[Figure 4‑9 CRC Card "Cell" 24](#_Toc133343912)

[Figure 4‑10 CRC Card "Screen Size" 25](#_Toc133343913)

[Figure 4‑11 CRC Card "Mode" 25](#_Toc133343914)

[Figure 4‑12 CRC Card "Debug" 25](#_Toc133343915)

[Figure 4‑13 CRC Card "Race" 25](#_Toc133343916)

[Figure 4‑14 CRC Card "Grid Size" 25](#_Toc133343917)

[Figure 4‑15 CRC Card "Which Algorithm" 25](#_Toc133343918)

[Figure 5‑1 Sequence Diagram 26](#_Toc133343919)

###### List Of Tables

[Table 2‑1 Description of The Helper classes and structs code. 3](#_Toc133343898)

[Table 2‑2 Description of The Utilised Class Images 7](#_Toc133343899)

[Table 2‑3 Description of The Code 19](#_Toc133343900)

[Table 2‑4 Storing of The Data 20](#_Toc133343901)

[Table 6‑1Technologies used 27](#_Toc133343902)

# Introduction

The objective of this project is to compare the benefits and drawbacks of using commonly used heuristic based guided pathfinding algorithms to the incremental algorithm known as Dstar Lite. This project will discuss the direct benefits of each algorithm in depth, from Astar, Dijkstra's search algorithm, Lifelong Planning Astar, and the non-guided algorithm known as Depth First Search when compared to D star Lite within a game’s context.

# Technical Design

The purpose of this document is to effectively communicate the technical details and design decisions of the system/algorithm to the readers.

It could include software architecture, algorithm design, class specifications, pseudo code, etc. with tools such as UML, Class Diagram, CRC Cards.

## Introduction

This section shows the technical design and over all architecture of the code including, the enums and structs that were used to control the different algorithms and grid sizes being run in the application. The header files of each algorithm and the rest of the files in the game. It will discuss what each variable and function is and does.

## Header Files

|  |  |
| --- | --- |
| Helper classes and structs code images | Description |
|  | The Enum class called “WhichAlgorithm” which controls which algorithm is being used a certain time. This Enum class contains the name for every pathfinding algorithm in the project. |
|  | The Enum class called “GridSize” controls the size of the grid which is being used in the program ranging from “small” to “very Large”.  “small” = “10x10” grid  “Large” “50x50” grid  “Very Large” = “100x100” grid |
|  | This Enum class called “Race” depicts whether you want to race the algorithms in comparison to dstar lite on a chosen path |
|  | The Enum class called “debug” toggles whether the user wants to see the variable values for Dstar Lite on the screen. This is only available with the small grid size |
|  | The struct called “Screen Size” struct which controls the size of each window |
|  | The Enum class called “Mode” which controls which mode the application is in. behaving differently depending on which one it is in |

Table 2‑1 Description of The Helper classes and structs code.

|  |  |
| --- | --- |
| Utilised class images | Description |
|  | **Public variables of Cell Class**  **The header file for the Cell(node) which has all the current functions in use.**  **Part A**  Cell() = default constructor  ~Cell() =default destructor  setStartColour()=sets the start cell of the paths colour  setEndColour()=sets the end cell of the paths colour  setColour()= sets the colour of the cell  Set/GetMarked()= sets and gets the m\_marked bool  Set/GetTraversable() = sets and gets the m\_traversable bool |
|  | **Public variables of Cell Class**  **the header file for the Cell(node) which has all the current functions in use.**  **Part B**  Set/Get EndPoint() = sets and gets the endpoint bool  Set/Get StartPoint() = sets and gets the endpoint bool  Set/Get ID= sets and gets the m\_ID for the cell  Set/Get Gcost() = sets and gets the gcost value for the cell  Set/Get Hcost() = sets and gets the Hcost value for the cell  Set/Get Rhscost() = sets and gets the Rhscost value for the cell |
|  | **Public variables of Cell Class**  **the header file for the Cell(node) which has all the current functions in use.**  **Part C**  Set/Get RisenBool() =sets and gets the isRisen bool  raiseCost() = raises the hcost of a cell  Set/Get Weight()= sets and gets the weight of a cell  Set/Get getPos() = sets and gets the position of a cell  getRect() =returns the rectangleShape of the cell  initRect()= initialises the rectangle shape and text  Set/Get Prev() = sets and gets the parent pointer of a cell  getNeighbours()= returns the list that stores the neighbours of that cell |
|  | **Public variables of Cell Class**  **the header file for the Cell(node) which has all the current functions in use.**  **Part D**  setNeighbours()= stores the neighbours of the cell into a list  Set/Get Predecessors() = sets and gets the predecessors of the cell  Set/Get Fcost() = sets and gets the fcost value of a cell  Set/Get JumpPoint() = sets and gets the m\_isjumpPoint Boolean  m\_Gcost= gcost value  m\_Hcost= hcost value  m\_RHScost= rhscost value  m\_Fcost= fcost value  m\_Xpos= the x position of the cell |
|  | **Public variables of Cell Class**  **the header file for the Cell(node) which has all the current functions in use.**  **Part E**  m\_Ypos= the y position of the cell in the grid  m\_weight= the cost to move to that cell  m\_isInOpenList= bool to check if that cell is in the open list  SetKey()/GetKey() = sets and gets the key value for the cell  m\_key= holds the key value for the cell  m\_rhsText= is the rhs value in text  m\_gcostText= is the gCostvalue in text  m\_keyText= is the Key value in text |
|  | **Private variables to Cell Class**  m\_marked= used to check if the cell has been visited  m\_Endpoint= used to check if the cell is the end goal of a path  m\_isStartPoint= used to check if a cell is a start point of a path  m\_traversable= check to see if the cell can be traversed  m\_jumpPoint= check to see if the cell is a jumPoint  m\_HcostRisen= check to see if the cells hcost has risen  m\_HcostLowered= check to see if the cells hcost has been lowered  m\_ID = holds the Id of the cell in the grid  m\_pos= holds the position of the cell in the grid  m\_rect= the sf::rectangleShape of the cell  m\_prev = the parent pointer to the cell  m\_neighbour = the list of neighbours of that cell  m\_predecessors = the list of predecessors of that cell |
|  | **private member variables of the grid class**  m\_font= font used for cells  m\_ptrCell = used in the grid setup  m\_maxCells = stores the max possible cells  m\_numberOfRows = stores the number of rows for the grid  m\_numberOfCols = stores the number of columns for the grid |
|  | **public members of the Grid class**  **Part A**  Grid() = default constructor  ~Grid() = default destructor  setMAXCELLS()= sets the maximum cells of the grid  setColumns()= sets the number of columns of the grid  setRows()= sets the number of rows of the grid  resetGrid() = resets the grid for transition down in grid sizes  resetAlgorithm() = resets the grid so a new algorithm can be run on the grid  atIndex() = gets the cell on the grid using the x and y position  m\_tableVector = the two-dimensional grid which holds the grid |
|  | public members of the Grid class  Part B  m\_startPosChosen = bool to set a cell as the start cell in the search  M\_endPosChosen = bool to set a cell as the end cell in the search  M\_INFINITY= double which is set to the max value possible divided by 10  setNeighbours() = sets the neighbours of a cell  setPredecessors() = sets the predecessors of a cell  setupGrid() = sets up the grid  heuristic() = calculates the heuristic value of a cell  render() = used to render the grid |

Table 2‑2 Description of The Utilised Class Images

|  |  |
| --- | --- |
| Algorithm’s classes: UI and games class images | Description |
|  | **Functor used in Dstar Lite**  returns the cell with the lower key value or in the case of an equal key value return the cell with lowest key value. |
|  | **Private members of the Dstar Lite Class**  K\_M = key modifier which is the offset for change in start position of the search  dstarLiteTimer = timer to track time for completion of the search  dstarGoalFound = termination condition of the search |
|  | **Public members of the Dstar Lite Class**  getTimer() = returns the time for search completion  getDstarPathFound() = returns the termination condition  U\_pq= priority queue which holds the cells and compares them against eachother  DstarLiteMain()= main loop of dstar lite  updateVertex() = updates the cells during the search  ComputeShortestPath() = computes the shortest path  initDstar() = initialises the grid for dstar lite to work  calculateDstarKey() = calculates the key for each cell  s\_Last = used for tracking position of robot |
|  | **Functor used in “Astar”.**  It compares each cell based on their hcost + their gcost and returns the lower of the two. |
|  | **“Astar” class as declared in the header file.**  m\_astarTimer= timer used to track time of search completion  m\_AstarDone = termination completion  getTimer() = returns m\_ astarTimer  getIdDone() = returns m\_ AstarDone  AstarInit()=initialises the grid for astar to work  computeShortestPath() computes the shortest path using astar  m\_stack = holds the path  Astar()= default constructor  ~Astar() = default destructor |
|  | **Functor used in “LpaStar”.**  returns the cell with the lower key value or in the case of an equal key value return the cell with lowest key value. |
|  | **Private members of the**  **Lifelong Planning Astar Class**  m\_clock = clock used to track time for completion  m\_K\_M = key modifier works as offset for change in heuristic  m\_LpaStarTimer = timer used to track time for completion  m\_LpaPathFound= termination condition |
|  | **Public members of the**  **Lifelong Planning Astar Class**  Set/Get terminationCondition()= sets and gets the termination condition  getTimer() = returns m\_LpaStarTimer  initLPAStar() = initialises the grid for lpa star to work  LPAStar() = computes the shortest path using the lpa star algorithm  updateNode()= updates each node in the path  CalulateKey() = calculates the key to a cell  LpaStar()=default constructor  ~LpaStar() = default destructor |
|  | **Functor used in “Dijkstra’s” search algorithm.**  It compares each cell based on their gcost and returns the lower of the two. |
|  | **“Dijkstra’s” Class as declared in the header file.**  m\_dijkstrasPathFound = termination condition  m\_dijkstasTimer= timer used to track time for completion  getTimer() = returns m\_dijkstrasTimer  getDijkstrasPathFound() returns m\_dijkstrasPathFound.  computeShortestPath() =computes the path using dijkstras search algorithm  Dijkstras() = default constructor  ~Dijkstras()= default destructor |
|  | **“Depth First Search” Class as declared in the header file.**  m\_clock = clock for timer  m\_depthGoalFound = termination condition  m\_depthfirstSearchTimer= timer used to track timer for completion  initComplete = bool for initialisation  getTimer() = returns m\_depthfirstSearchTimer  setTimerBool() = used to reset timer  computeShortestPath() = computes the path using depth first search algorithm  initDepth() = initialises the grid for depth first search  DepthFirstSearch() = default constructor  ~DepthFirstSearch() default destructor |
|  | **“Menu” class as private member variables in the header file.**  **Part A**  m\_gridSwitcher = controls grisSize Enum  m\_slgSwitcher = controls WhichAlgorithm Enum  m\_raceDecider = controls Race Enum  m\_debugDecider = controls debug enum  m\_rect = rectangleShape  m\_rectVec = holds all the sf::rectangleShapes  m\_font = font used  m\_text[] = array which holds all of the text  m\_yPosition = initial y position of the rectangleShape  m\_xPosition = initial x position of the rectangleShape  m\_offset = offset for positions |
|  | **“Menu” class as private member variables in the header file.**  **Part B**  From m\_leftColXpos to m\_positionOffset are all positional values and offsets  m\_XsizeOffset = used for change in rectangleShape size  originalSize= original value for size  m\_MAX\_TXT\_RECTANGELS = max text allowed |
|  | **“Menu” class as public member variables in the header file.**  Menu() = default constructor  ~Menu() = default destructor  getalg() = returns m\_slgSwitcher  getRaceStatus() = returns m\_raceDecider  getdebugStatus() = returns m\_debugDecider  getVec() = returns m\_rectVec  setGridSize() = sets and gets the grid size also selects the algorithm to be used  render()= renders the menu |
|  | **Public members to class “Game”, as declared in the header file.**  **Part A**  Game() = default constructor  ~Game() = default destructor  m\_switcher= switched algorithm used  m\_gridSizeState = switchers grid size  m\_raceState = controls if algorithms race  m\_debugState= controls the debug rendering  m\_mode = controls which mode the application is in  m\_astar= Astar object used to run Astar search  m\_dijkstras= Dijkstra’s object used to run Dijkstra’s search  m\_depthFirstSearch = as depth First Search object used to run depth First Search |
|  | **Public members to class “Game”, as declared in the header file.**  **Part B**  m\_dstarLite= Dstar Lite object used to run Dstar Lite search  m\_LpaStar= Lifelong Planning A star object used to run Lpa star  m\_grid = grid object for editable grid  m\_gridTwo = used for grid with Dstar Lite  m\_jps = Jump Point Search star object used to run jump point search  m\_ida = Iterative Deepening A Star used to run Ida star  run() = run function used in game loop  m\_temp = controls the wall placement |
|  | **Private members to class “Game”, as declared in the header file.**  **Part A**  PlayMode() = runs the application in play mode  TestingMode() = runs the application in testing mode  m\_srtChosen = bool to check if a start cell has been chosen  m\_EndChosen = bool to check if the end cell has been chosen  Temp = wall control on grid one  tempOne = wall control on grid two  m\_exitGame = bool to control exiting the application  from m\_tempsEnd to m\_cellVar are all used to run the algorithms and find the star positions and end positions as cells |
|  | **Private members to class “Game”, as declared in the header file.**  **Part B**  From m\_window to m\_windowAstar = all of the windows used in the application  From m\_startCell\_id to m\_EndCellTwo\_id= used to track the ids of each start and end cell on both windows  From processEvents() to render() = functions used in the game loop |

Table 2‑3 Description of The Code

## Data structures used

The different types of data structures used in the projects are as follows:

1. Std::vector<Cell\*> - generally used in the application for the building of the 2D grid
2. Std::vector<Std::vector<Cell\*>> - is used to store the entire grid
3. Std::stack<Cell\*> - this is used to store the paths in the application
4. Std::list<Cell\*> - this is used to store the neighbours and predecessors of a cell
5. Static array of type text and int – used for text and rectangles shapes
6. Static Struct – used for the screen sizes
7. Object Class – used for algorithms menu, game and grid
8. Static Enum Class – used for different functions in the application
9. Std::queue<Cell\*> - used during the searches
10. std::priority\_queue<Cell\*, std::vector<Cell\*>, functor used > - used during the searches for different algorithm

## Storing of Data

### How is the Data Stored?

The Data is stored inside of an excel file for each algorithm. It sores the time which it takes for the algorithm to complete the path in seconds. Each algorithm has three separate excel files for the times stored on the three separate grid sizes from small, medium, and large grid sizes and as such the times stored reflect this.

### When is the Data Stored?

The Data is stored after the algorithm has been run and the user can also select the testing mode which will give the algorithms a random start and end position. This will avoid any positional or path length bias as the path is completely randomised on the grid.

The Data was then collected and used for comparison purposes.

|  |  |
| --- | --- |
| Code/File/Data Storing Images | Description |
|  | Example for the data stored in the excel file.  The example is the data stored in “AstarTime.csv” this is the data collected on the small grid running the Astar Algorithm |
|  | Example of the name of the algorithms excel file |
|  | Example of how the algorithm times is stored in code.  Description of code: This code collects the time taken for the algorithm to finish then inputs that time into the excel sheet. |

Table 2‑4 Storing of The Data

# User Flow

1. The user opens the application
2. The user selects the grid size they want to run their algorithms on
3. The user selects the algorithm that they want to use
4. The user selects the start and end points of the search on the graph (start point selcted with mouse left click and end point selected with mouse right click)
5. The user will then see the path the algorithm takes
6. The user can place down walls on the grid and see how the algorithm reacts Note. This can be done before step 4 ( wall placed will scroll wheel on the mouse)
7. The user will then see how the algorithm reacts to changes in the path
8. The user will then select to race the algorithm and will see the path Dstar Lite used
9. The user will then select to run debug and run a new path
10. The user will then see how Dstar Lite makes changes to the grid
11. The user will then select a different algorithm to see how it compares to Dstar Lite
12. Repeat step 11 until the user is finished with the application
13. The user can then see the times taken for each algorithm in their own excel sheets

# Class Diagram

Diagram

Description automatically generated

Figure 3‑1 Class Diagram

# CRC Cards

Graphical user interface, text, application, email

Description automatically generated

Figure 4‑1 CRC Card "Grid"

Graphical user interface, text

Description automatically generated

Figure 4‑2 CRC Card “Game"

Graphical user interface, text, application

Description automatically generated

Figure 4‑3 CRC Card "Menu"

Table

Description automatically generated with medium confidence

Figure 4‑4 CRC Card "Astar"

Graphical user interface, text, application

Description automatically generated with medium confidence

Figure 4‑5 CRC Card "Dstar Lite"

Graphical user interface, text, application

Description automatically generated

Figure 4‑6 CRC Card "Dijkstras"

Graphical user interface, text

Description automatically generated

Figure 4‑7 CRC Card "Depth First Search"

Table

Description automatically generated with low confidence

Figure 4‑8 CRC Card "Lpa Star"

Table

Description automatically generated with medium confidence

Figure 4‑9 CRC Card "Cell"

Text

Description automatically generated

Figure 4‑10 CRC Card "Screen Size"

Graphical user interface, text, application

Description automatically generated

Figure 4‑11 CRC Card "Mode"

Table

Description automatically generated with medium confidence

Figure 4‑12 CRC Card "Debug"

Table

Description automatically generated

Figure 4‑13 CRC Card "Race"

Text, table

Description automatically generated with medium confidence

Figure 4‑14 CRC Card "Grid Size"

A picture containing graphical user interface

Description automatically generated

Figure 4‑15 CRC Card "Which Algorithm"

# Sequence Diagram

Diagram, box and whisker chart

Description automatically generated

Figure 5‑1 Sequence Diagram

# Technologies

These are the technologies used for the completion of this application

|  |  |
| --- | --- |
| Technology | Image |
| SFML 2.5.1 |  |
| C++ |  |
| Visual Studio 2022 |  |
| Excel |  |

Table 6‑1Technologies used