Swinburne University of Technology  
Faculty of Science, Engineering, and Technology

**COS30019: Introduction to Artificial Intelligence**

Assignment 1B: Robot Navigation

|  |  |  |
| --- | --- | --- |
| Date of report submission |  | --/--/20 |
| Lab Supervisor |  | --- |
| Group |  | 12:30pm Monday  Even/Odd Weeks |

|  |  |
| --- | --- |
| Name | Student ID |
| Jimmy Trac | 101624964 |
|  |  |
|  |  |

This page is mostly blank.

The purpose of this page is to separate the cover page from the report when printed double-sided.

Robot Navigation

Jimmy Trac | 101624964

A. Table of Contents

[I. Table of Contents 3](#_Toc36992348)

[II. Instructions for use 3](#_Toc36992349)

[II. 1. Standard Output 4](#_Toc36992350)

[II. 2. GUI Output 5](#_Toc36992351)

[II. Search Algorithms 6](#_Toc36992352)

[III. Features / Bugs / Missing 6](#_Toc36992353)

[IV. Research Initiatives 6](#_Toc36992354)

[V. Conclusion 6](#_Toc36992355)

[VI. Acknowledgements 6](#_Toc36992356)

[VII. References 6](#_Toc36992357)

B. Instructions for use

The Robot Navigation Program, or navigation program, has two primary modes of use:

1. Standard Output (Console output),
2. Graphical User Interface (GUI) Output

The following format is used for command line input:

|  |
| --- |
| robot-navigation.exe [filename] [method] <string: “gui”> <int: agentdelay> |

Where the square brackets (“[ ]”) refer to required parameters, and the angled brackets (“< >”) refer to optional parameters.

By specifying a third parameter, the program uses GUI output to visualise the search. Furthermore, the search can be artificially slowed by specifying an integer value for the fourth argument. For example:

|  |
| --- |
| robot-navigation.exe robot-nav-map.txt bfs gui 20 |

The program then runs a GUI showing an agent performing Breadth-First Search, slowing down the simulation by taking 20 delays before each action.

B. 1. Standard Output

Standard output refers to the pure CLI use of the program as specified by the requirements. It is the default mode of the program after providing the minimum number of parameters, and outputs the filename, method, number of nodes, and the path, in the following format:

|  |
| --- |
| filename method number\_of\_nodes  path |

Figure B.1.1 and B.1.2 show the standard outputs of the program.

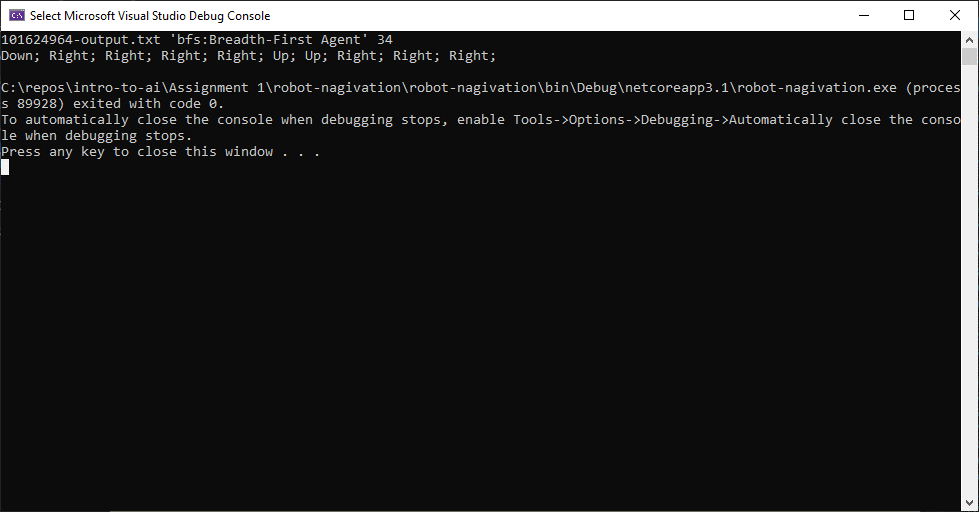


Figure B.1.1 – Screenshot showing the console output of the program performing Breadth-First Search

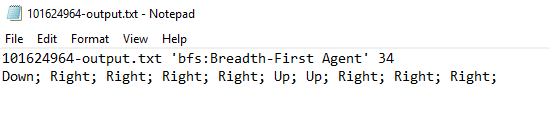


Figure B.1.2 – Screenshot showing the format of the output file “101624964-output.txt”

In the case that the program is unable to find a solution in the proposed map, the program outputs “No Solution Found” in place of the path, as shown in figure II.1.3.

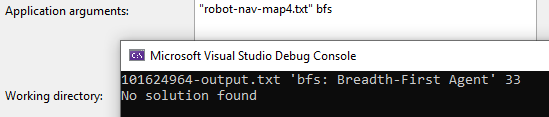


Figure B.1.3 – No solution found on “robot-nav-map4.txt”

B. 2. GUI Output

The primary use-case of the program of the navigation program is the GUI representation of the search space and node tree as seen in figure B.2.1.

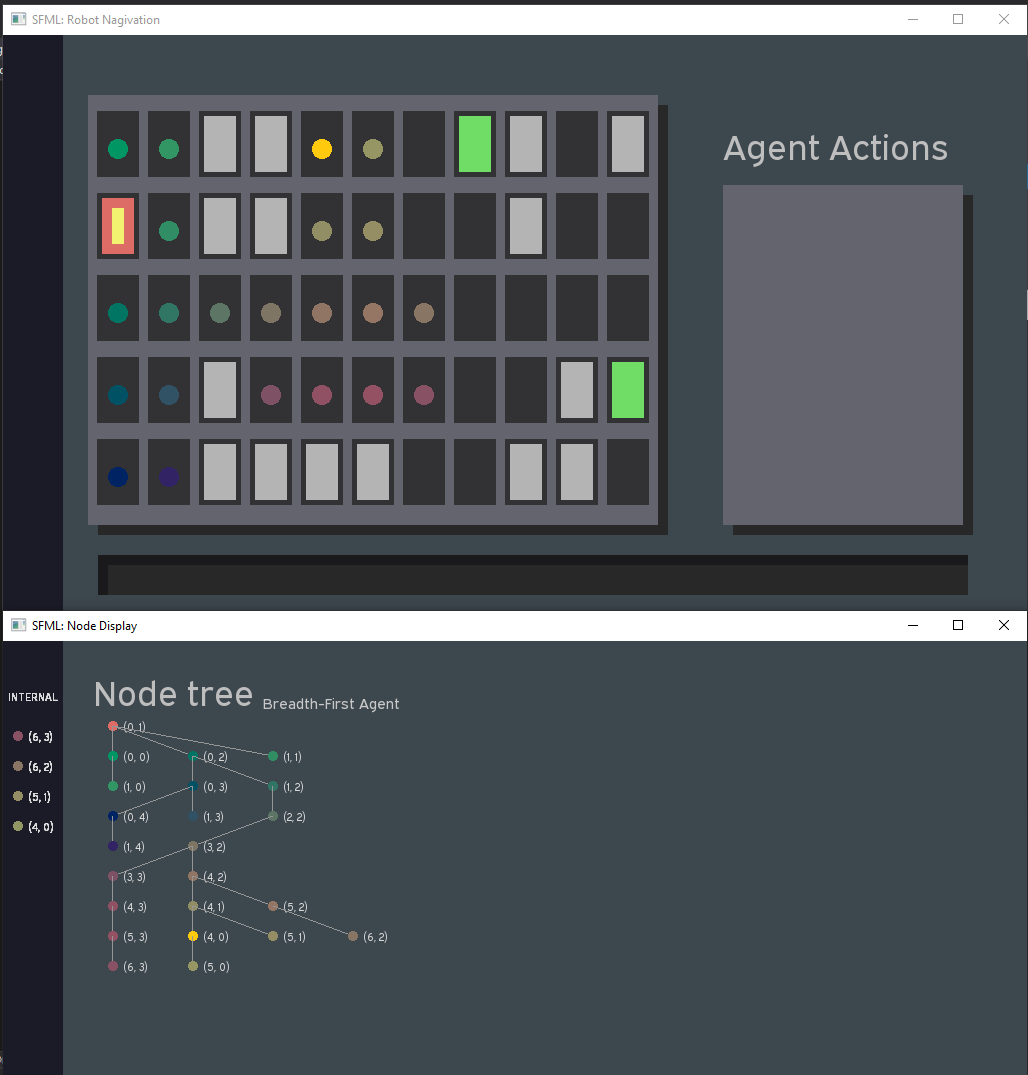


Figure B.2.1 – the Navigation Program’s two main windows

The GUI is composed of two windows:

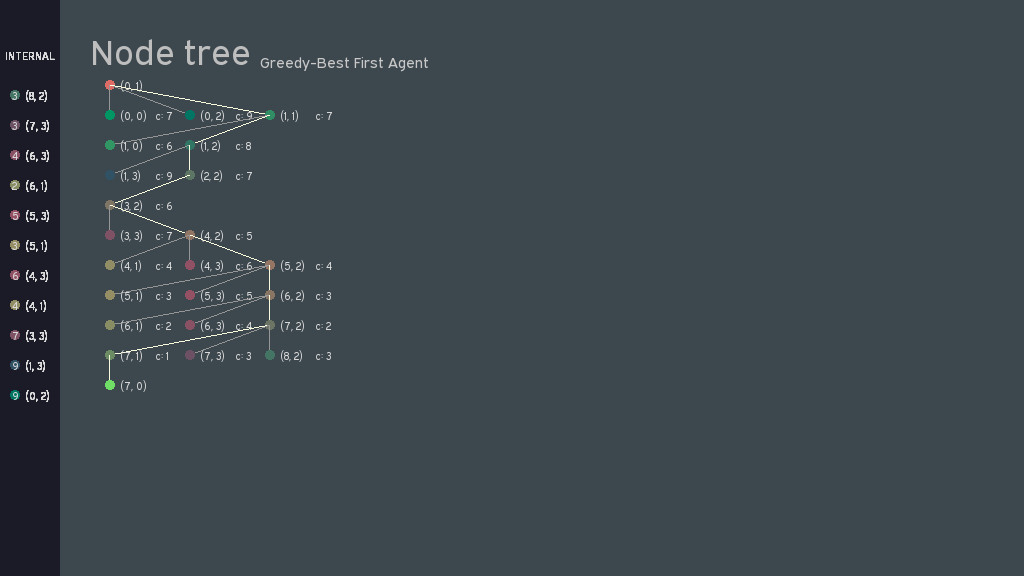
1. Agent Actions, and
2. Node Tree.



Figure B.2.2 – Screenshot of the Agent Actions window

The Agent Actions screen is the primary window of the program, showing a representation of the map in the left along with the currently searching nodes, searched nodes, and the final agent path. Figure B.2.2 shows the agent actions screen once the search has been completed, with the searched nodes showing as coloured circles and the actions list provided. The numbers in the circles represents the cost of that node being the sum of the cost and heuristic function of applicable.

In addition, there is a agent path shown that transitions from blue to red, showing the starting, ending, and transitional positions/nodes of the agent.



The Node Tree window shows a representation of the search problem as a node tree, with each node corresponding to the location and colour as seen in the Agent Actions window. This window shows the tree traversal of the agent

1. Introduction

This template is a general-use template for university reports and the like. There are two main formats depending on the size of your report. The first is the style depicted here, with large stepped titles designed for reasonably large reports. The second can be found a few pages down, which has simpler title layouts for shorter reports.

* This first style for long reports
* Second style further down for short ones
* For more formal or a slight twist, nko12 is designed for literature with a double-column effect

II. Search Algorithms

To achieve this, it would be necessary to have uniform grammar, pronunciation and more common words. If several languages coalesce, the grammar of the resulting language is simpler and more regular than that of the individual languages. The new common language will be simpler and more regular than the existing European languages. It will be as simple as Occidental; in fact, it will be Occidental. To an English person, it will seem like simplified English, as a sceptical Cambridge friend of mine told me what Occidental is.

III. Implementation

As the primary design goal of the robot navigation program was to visualise the searching methods of the agent, the following considerations were of high importance:

* Separating the different viewing modes / primary use modes of the program, i.e. console view, console output, and GUI output.
  + Console View referred to a pure console view of the agent movement, Console Output is as required in the requirements with only path traversals.
* The Agent could not “move” and search but differentiated between *searching* and *moving*.
  + For example, the agent is only allowed to move one square at a time, and therefore would not be able to perform a Breadth-First Search as it would “teleport” between tree layers
* The Agent must return information to the main program, and therefore, does not perform the search in one go – there cannot be a while loop within the agent to perform the search
* Finally, the GUI must be intuitive and useful, with design factors such as:
  + Visualising the movement of the agent on a static image, this is solved with the blue-red colour line.
  + Visualising the tree node and associating each node with a position, this was solved with both a position marker on each node and an associated colour
  + Allowing the user to artificially slow down the search algorithm to visualise the search better

In order to meet the above requirements, a pseudo-Model View Controller paradigm was used, where the views were split into interfaces and the message object between all classes was the *ProgramData* class. The full UML diagram is given in the appendix.

Figure 3.1 shows the top-level implementation with the Model and View, with the controller being the Main program itself. The ActionActions box refers to the Enumeration used to represent the actions of the agent.

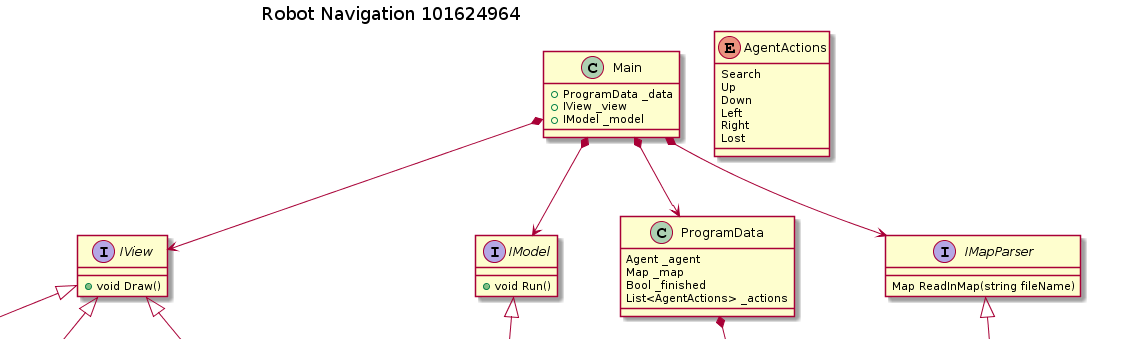


Figure 3.1 – A high level view of the navigation program

A breakdown of the responsibilities of each component is as follows:

* View:
  + Does not accept any input whatsoever
  + Takes Program Data object and creates a visual representation of the world, be it a Console Output or a GUI.
* Model:
  + Represents the “Logic” of the world, containing methods for moving the agent, checking for collisions, and creating the percepts for the agent from the Program Data.
  + The model “Runs” the simulation by calling the agent and when the agent is finished, will rely the message to the Main Controller.
* Main:
  + Acting as a controller, the Main takes user input and converts that into the correct components.
  + This includes the correct agent and map input.
  + The Main then provides error handling and text output as required.

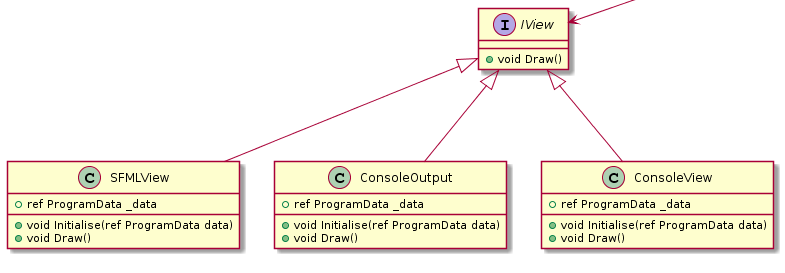


Figure 3.2 – The IView interface and its superclasses

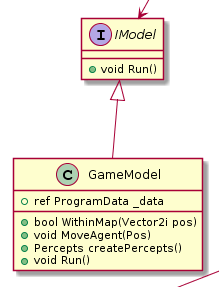


Figure 3.3 – The Imodel Interface and the GameModel, showing the logical methods of the model

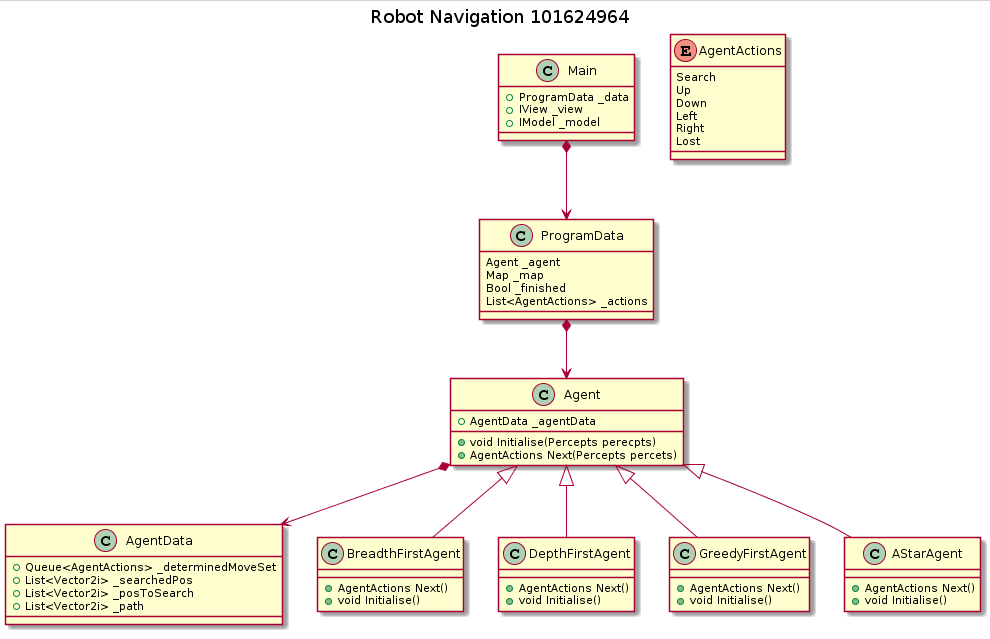


Figure 3.4. The Agent Hierarchy with the agents inheriting from one base class and all data stored within the Program Data Class

IV. Features / Bugs / Missing

Features:

* Extensive GUI representation of the search space with the agent actions and node map windows.
  + Search space animates depending on the searching pattern of the agent
  + The node map shows the actual agent traversing the actual map in real-time, including traversals up the tree
* The movement of the agent is given as a blue-to-red line through the map to represent the path taken.
* The cost of each node is shown in both windows to represent the logic taken by the agent
* Each searched and explored node has a colour associated with it, which changes based on the “region” of the map. This colour can be cross-referenced with the node tree to, relatively quickly, find the corresponding nodes between the two windows.
* The Node Tree window, on the left side, contains information on the internal state of the agent. This represents either the internal stack or queue of the agent.
* Though not a high priority, the colours were chosen such that each colour had differing brightness levels for accessibility, particularly with red-green colour-blindness, as seen in figure 4.1.

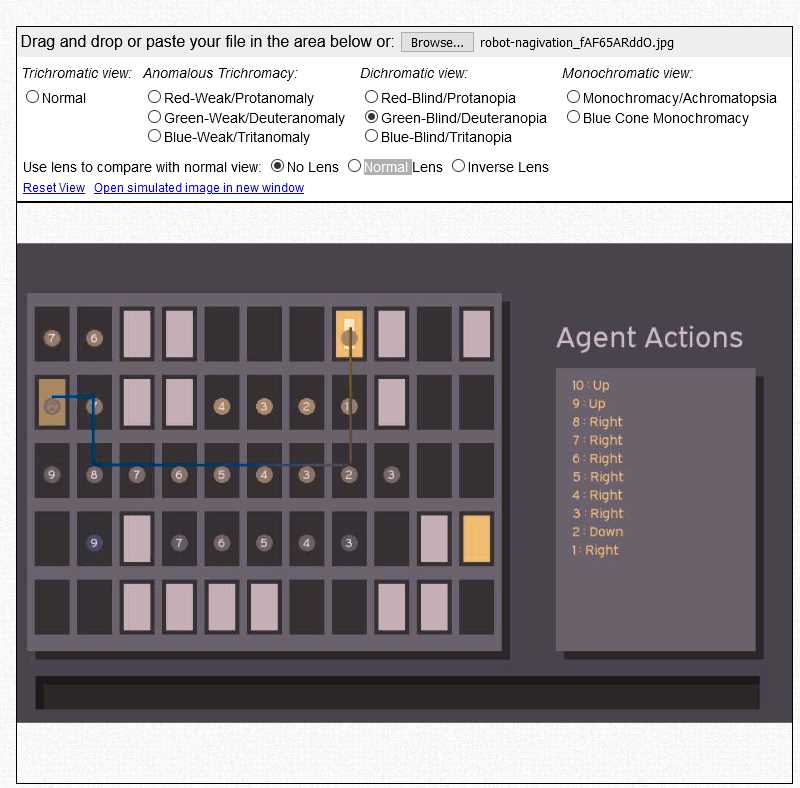


Figure 4.1 – Simulated red-green colour-blindness. Source: www.color-blindness.com/coblis-color-blindness-simulator

V. Research Initiatives

VI. Conclusion

VII. Acknowledgements

VIII. References