

Agent AI Report - Steffan Davies

by Steffan Luke Evans

FILE	2095_STEFFAN_LUKE_EVANS_AGENT_AI_REPORT_-_STEFFAN_DAVIES_110679_957348114.DOCX		
TIME SUBMITTED	05-APR-2018 09:26PM (UTC+0100)	WORD COUNT	783
SUBMISSION ID	86089767	CHARACTER COUNT	5155



AGENT AI: SEARCH ALGORITHMS

Steffan Davies – P149401



Table of Contents

Introduction	2
Algorithms used and their Implementations	2
User Guide	2
Test Schedule	3
Appendices	4
References	7

Introduction

The aim of this report is to provide an overall highlight of the pathfinding AI that has to be written and implemented into the game. This will include identifying and describing any algorithms that have been implemented, a test schedule, user guide, program listing and screenshots that show that program and its functions.

Algorithms used and their Implementations

An algorithm that will be used to create a pathfinding AI will be Depth First Search (DFS). This algorithm is being used for this task because it is used for traversing graph or data structures[1][2][3]. It does this with the use of stacks/LIFO (Last-In-First-Out) queues[4]. The maze has been loaded into the program as a '.maz' file and by using the '#define' width and height, the maze can be loaded into the program as a 16 by 16 maze.

To track the position co-ordinates, a typedef structure was created. It contains two integers. These are the x and y values that will be used to identify a position of the maze. A stack function was also created and included the position typedef structure. By using this, the program will be able to pop and push position co-ordinates.

In the main, two positions were created. These are the start and current positions. The start position will be used to initialise the stack and the current position will keep track of the current co-ordinates during the maze traversal. Using a while loop, the directions of the maze traversal were to be implemented. however, these are not properly functioning and is therefore incomplete.

The loop also uses system("CLS"). This clears the screen every time a loop is done. This was included to make the display easier to look at for the user[5]. System("PAUSE") is also used just before the while loop. This pauses the program until the user decides to continue running it and presses a key[5].

The program includes a 'printMaze' function. The purpose of this function is to allow the user to select a maze and then print the selected maze. To do this, the user must pick a number between 1 and 3. This function is then called in the while loop to show the maze in its up-to-date form.

User Guide

When the program starts, you will be given a choice of 3 mazes to choose from. Each maze is 16 by 16 but they contain different routes to the exit and some contain more dead ends than others. To pick the maze, you will have to enter the number of the maze you want to use. 1 being easy, 2 being medium and 3 being hard. Once the maze has been selected by the user, the user will then be shown the selected maze. Once the user presses any key, the program will continue to run automatically until the user decides to quit or pause the program.

Test Schedule

Test No	Test Description	Expected Result	Actual Result
1	This test is to check if the maze can print the maze in the while loop.	It prints the maze.	It didn't print the maze. The issue was that the file couldn't be opened in the main. To fix this, a function will be made so that it can be called into the main.
2	This test is to check if the maze is able to print the maze by calling the 'printMaze' function in the while loop.	It prints the maze.	The maze prints as expected.
3	To test if the start position is in the correct position of the maze. This will be tested by using the direction variables and printing their coordinates.	Start position is in the correct position.	As expected, the start position was in the correct position.
4	This test will look at whether the current position will be able to move by looking for empty spaces next to the current position in the maze.	The current position will move down due to that being the only available empty space.	It detects that there is an empty space, however it doesn't move. Since the current position does not move, it continues to increase the y-axis co-ordinate.

Appendices

Appendix 1: Program Listing

```

1  ////////////// size of maze
2  #define WIDTH 16
3  #define HEIGHT 16
4  char maze[WIDTH*HEIGHT];
5  ///////////////
6  #include <stack>
7  #include <iostream>
8  #include <fstream>
9  #include <string>
10 using namespace std;
11
12 ////////////// Position Coordinates
13 typedef struct
14 {
15     int x;
16     int y;
17 } position;
18
19 stack<position> dfsStack; // DFS Stack traversal
20 int mazeSelection;
21 void printMaze();
22
23 /////////////// MAIN ///////////////
24 int main(void)
25 {
26     ////////////// Selecting the maze //////////////
27     cout << "Please select a maze " << endl << endl;
28     cout << "1 = Easy " << endl << "2 = Medium " << endl << "3 = Hard " << endl;
29     cin >> mazeSelection;
30     cout << endl;
31
32     printMaze(); // Calling method to print selected maze
33
34     ////////////// Initialising Positions //////////////
35     position start; // initialise start position
36     start.x = 1;
37     start.y = 1;
38
39     position current; // initialise current position
40     current.x = start.x;
41     current.y = start.y;
42
43     dfsStack.push(start); // initialise stack with start position
44
45     ////////////// Directions
46     char UP, DOWN, LEFT, RIGHT;
47
48     UP = maze[WIDTH*(current.y - 1) + current.x];
49     DOWN = maze[WIDTH*(current.y + 1) + current.x];
50     LEFT = maze[WIDTH*current.y + (current.x - 1)];
51     RIGHT = maze[WIDTH*current.y + (current.x + 1)];
52
53     system("PAUSE"); // Pauses the program
54

```

```
55 // DFS while loop
56 while (!dfsStack.empty()) // while stack is not empty
57 {
58     //system("PAUSE");
59     system("CLS"); // clears the screen
60
61     if (UP == ' ')
62     {
63         current.y--;
64         dfsStack.push(current);
65     }
66     cout << UP << " UP" << endl; // prints 1 up of current position
67
68     if (DOWN == ' ')
69     {
70         current.y++;
71         dfsStack.push(current);
72     }
73     cout << DOWN << " DOWN" << endl; // prints 1 down of current position
74
75     //if (DOWN == '#')
76     //{
77     //    dfsStack.pop();
78     //}
79
80     if (LEFT == ' ')
81     {
82         current.x--;
83         dfsStack.push(current);
84     }
85     cout << LEFT << " LEFT" << endl; // prints 1 left of current position
86
87     if (RIGHT == ' ')
88     {
89         current.x++;
90         dfsStack.push(current);
91     }
92     cout << RIGHT << " RIGHT" << endl << endl; // prints 1 right of current position
93
94     printMaze(); // Calling method to print selected maze
95     //cout << "popping (" << current.x << ", " << current.y << ")" << endl;
96 }
97 return 0;
98 }
99
```

```
100 void printMaze() // method used to print selected maze
101 {
102     /// Maze 1 ///
103     if (mazeSelection == 1)
104     {
105         string line;
106         ifstream myfile("maze1.maz");
107         if (myfile.is_open())
108         {
109             int count = 0;
110             while (getline(myfile, line))
111             {
112                 cout << line << endl;
113                 line.copy(&maze[WIDTH*count], WIDTH, 0);
114                 count++;
115             }
116             myfile.close();
117         }
118     }
119
120     /// Maze 2 ///
121     if (mazeSelection == 2)
122     {
123         string line;
124         ifstream myfile("maze2.maz");
125         if (myfile.is_open())
126         {
127             int count = 0;
128             while (getline(myfile, line))
129             {
130                 cout << line << endl;
131                 line.copy(&maze[WIDTH*count], WIDTH, 0);
132                 count++;
133             }
134             myfile.close();
135         }
136     }
137
138     /// Maze 3 ///
139     if (mazeSelection == 3)
140     {
141         string line;
142         ifstream myfile("maze3.maz");
143         if (myfile.is_open())
144         {
145             int count = 0;
146             while (getline(myfile, line))
147             {
148                 cout << line << endl;
149                 line.copy(&maze[WIDTH*count], WIDTH, 0);
150                 count++;
151             }
152             myfile.close();
153         }
154     }
155     cout << endl;
156 }
```

References

- [1] S. Books Online LLC, 'Algorithms in a Nutshell'.
- [2] 'DFS algorithm for graph (With pseudocode, example and code in C, C++, Java, Python)'. [Online]. Available: <https://www.programiz.com/dsa/graph-dfs>.
- [3] 'Depth First Search Tutorials & Notes | Algorithms | HackerEarth'. [Online]. Available: <https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/>.
- [4] S. Russell and P. Norvig, 'AI: A Modern Approach'.
- [5] 'Clear Screen and More'. [Online]. Available: <https://mathbits.com/MathBits/CompSci/Screen/clear.htm>.

Agent AI Report - Steffan Davies

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

40/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

GRADING FORM: SMU GENERIC_GJ

STEFFAN LUKE EVANS

41

RESEARCH-INFORMED LITERATURE



Refs included and used appropriately - well done. Good to see you've looked at Norvig and Russell. You've reviewed required SE principles related to code dev but include a ref to these.

2

KNOWLEDGE AND UNDERSTANDING

	An essential understanding of request data structures and their use but your understanding of the dfs algorithm appears to be lacking. It looks like you've got the overall idea but your code shows you lack the knowledge and or understanding of how to select adjacent nodes as part of the dfs algorithm. Hence your agent/Albot gets stuck in the maze. Other aspects of the problem scenario interpreted appropriately and coded.	8
---	--	----------

ANALYSIS

	Analysis of problem in evidence, essential algorithms and appropriate components identified. Analysis of of dfs and conversion to C++ code flawed (see previous section).	6
---	---	----------

PRACTICAL APPLICATION AND DEP.

	Animation present that faithfully shows this! Variable mazes can be loaded - good	22
---	---	-----------

SKILLS FOR PROFESSIONAL PRAC.

	Refs included. Do try to avoid too many electronic references. Test schedule provided. Comments used appropriately too. Good.	3
---	---	----------