

Agent AI Pathfinder Report

by Steffan Benson

FILE	1348_STEFFAN_BENSON_AGENT_AI_PATHFINDER_REPORT_110679_6 43865668.DOCX		
TIME SUBMITTED	29-MAR-2018 12:34PM (UTC+0100)	WORD COUNT	795
SUBMISSION ID	85800453	CHARACTER COUNT	4801

Name: Steffan John Benson

P Number: P151818

Module Title: Agent AI

Module Code: M5X01022

Module Lecturer: Gordon Dickers

Assignment Title: Pathfinding AI

Contents

Search Algorithm and Implementation	3
User Guide	3
Testing.....	4
Screenshots	5
References	8
Figure 1 - Maze once target node has been reached.	5
Figure 2 - Portion of whereCanIGo() method.	5
Figure 3 - Checks for target node and dead ends.	6
Figure 4 - File reader and start/target node print outs.	6
Figure 5 - maze print out.	7

Search Algorithm and Implementation

The search algorithm utilised to search the maze pertains to the Depth First Search (DFS) method. DFS methods will continue down a specified branch until it reaches the end of the branch, before returning to a node where an unexplored branch begins[1][2].

Implementing this into the solution is done by searching in each direction, which are in order of up, down, left and right, and checking to see if there is a wall (#) or an empty space () in each direction. If a check returns a wall, the following code is ignored. If the search finds an empty space, the position is modified to reflect the movement of the AI. If multiple empty spaces have been found, the AI will move in the direction it finds first, as per the order mentioned above. Once the AI reaches a 'dead end', it will return to the last node on the stack where an empty space was recorded and not explored. The return to this node is done by popping all previous nodes off of the stack, until an unexplored node is at the top of the stack. This process is repeated as many times as necessary until it reaches the target node, signified by the letter Y. The AI will check to see if it has detected the target node, and if the check returns positive, the search will terminate and the position of the target node is printed below the final maze view.

In order to prevent the AI from returning to the start location, a check is performed to see if the AI has moved. If the AI has moved, the startSpace variable is changed from true to false, which will prevent the current location of the AI from being reset to the start location.

User Guide

In order to run the program, open Microsoft Visual Studio 2015. Open the MazeSearch.sln file, clean and build the solution before running. Run without debugging, using the CTRL+F5 shortcut, and the pre-programmed maze will open. Should you wish to test with a different sized maze, the height and width values at the top of the programme should be modified to the height and width of the maze plus 1 for both values. The name of the .txt to be read should also be changed before testing with a different maze. Once the programme begins, the search begins automatically and will run until the target node has been located. The maze may flicker slightly as the search is performed, this is unavoidable as the maze is cleared and reprinted for every movement of the AI.

Multiple mazes have been provided for further testing, however the use of an online maze generator can be used to make more[3].

Testing

Test No.	Test Description	Expected Result	Actual Result	Pass/Fail
1	Test to ensure file reader works.	Maze File is read in successfully.	Maze File is read in successfully.	Pass
2	Search for start location.	Start location located and printed at end of search.	Start location located and printed at end of search.	Pass
3	AI movement in 4 different directions.	AI can move in all 4 specified directions.	AI can move in all 4 specified directions.	Pass
4	AI wall recognition.	AI recognizes walls and will not travel through them.	AI recognizes walls and will not travel through them.	Pass
5	AI backtracking if it reaches a dead end.	AI can return to previous nodes.	AI can return to previous nodes.	Pass
6	AI target node detection.	AI can detect the target node.	AI can detect the target node.	Pass
7	Search termination once target node has been reached.	The search stops once the AI detects the target node.	The search stops once the AI detects the target node.	Pass

Screenshots

```
C:\Windows\system32\cmd.exe

#S#####
#.#.....#
#...### #.#####
#.#...#.#.....#
#...##### #
#.#.#...#####
#.#...#.#.#.....#
#.#.#.#.#.#.....#
#.#.#.#####.#
#.#...#Y#####
#.#.#.#o....#
#.#.#.#####
#.#.#.....#
#.#.#.#####.#
#.#.....#...#
#####

Start located at position: (1,0) of maze
Goblet is located at position: (9,10) of maze
Press any key to continue . . .
```

Figure 1 - Maze once target node has been reached.

```

28
29 void whereCanIGo(void)
30 {
31     bool upMove = false;
32     bool downMove = false;
33     bool leftMove = false;
34     bool rightMove = false;
35
36     if (startSpace == true)
37     {
38         current.x = start.x;
39         current.y = start.y;
40         startSpace = false;
41     }
42
43     char lookUp, lookDown, lookLeft, lookRight;
44     lookUp = maze[WIDTH*(current.y - 1) + current.x];
45     lookDown = maze[WIDTH*(current.y + 1) + current.x];
46     lookLeft = maze[WIDTH*current.y + (current.x - 1)];
47     lookRight = maze[WIDTH*current.y + (current.x + 1)];
48
49     if (lookUp == ' ')
50     {
51         if (1 < current.x < WIDTH)
52         {
53             if (1 < current.y < HEIGHT)
54             {
55                 //go.push_back(WIDTH*(current.y - 1) + current.x);
56                 temp.y = (current.y - 1);
57                 temp.x = current.x;
58                 dfsStack.push(temp);
59                 upMove = true;
60                 /*cout << "UP" << endl;*/
61             }
62         }
63     }
64 }

```

Figure 2 - Portion of whereCanIGo() method.

```

113
114     if (upMove == false && downMove == false && leftMove == false && rightMove == false)
115     {
116         /*maze[WIDTH*current.y + current.x] = '-';*/
117         dfsStack.pop();
118     }
119
120     if (lookUp == 'Y' || lookDown == 'Y' || lookLeft == 'Y' || lookRight == 'Y')
121     {
122         temp.x = current.x;
123         temp.y = current.y;
124         dfsStack.push(temp);
125         finished = true;
126     }

```

Figure 3 - Checks for target node and dead ends.

```

128
129 int main(void)
130 {
131     string line;
132     ifstream myfile("maze.txt");
133     if (myfile.is_open())
134     {
135         int count = 0;
136         while (getline(myfile, line))
137         {
138             cout << line << endl;
139             line.copy(&maze[WIDTH*count], WIDTH, 0);
140             count++;
141         }
142         myfile.close();
143     }
144     else cout << "Unable to open file";
145     cout << endl << endl << endl;
146
147     //maze print and start position search
148     for (int y = 0; y < HEIGHT; y++)
149     {
150         for (int x = 0; x < WIDTH; x++)
151         {
152             if (maze[WIDTH*y + x] == 'S')
153             {
154                 start.x = x;
155                 start.y = y;
156             }
157         }
158         cout << endl;
159     }
160
161     position startPosition;
162     startPosition.x = 1;
163     startPosition.y = 0;
164     dfsStack.push(startPosition);
165
166     while (finished != true) { ... }
167
168     cout << endl << "Start located at position: (" << start.x << ", " << start.y << ") of maze" << endl;
169     cout << "Goblet is located at position: (" << current.x << ", " << current.y << ") of maze" << endl;
170
171     return 0;
172 }

```

Figure 4 - File reader and start/target node print outs.

```

165
166     while (finished != true)
167     {
168         whereCanIGo();
169         current = dfsStack.top();
170         maze[WIDTH*current.y + current.x] = '.';
171         cout << "Current Location: (" << current.x << ", " << current.y << ")" << endl;
172         system("cls");
173         //maze print and start position search
174         for (int y = 0; y < HEIGHT; y++)
175         {
176             for (int x = 0; x < WIDTH; x++)
177             {
178                 if (y == current.y && x == current.x)
179                     cout << "o";
180                 else
181                     cout << maze[WIDTH*y + x];
182             }
183             cout << endl;
184         }
185     }

```

Figure 5 - maze print out.

References

- [1] K. Moore and K. Jenkinson, "Depth-First Search (DFS) | Brilliant Math & Science Wiki." [Online]. Available: <https://brilliant.org/wiki/depth-first-search-dfs/>. [Accessed: 16-Mar-2018].
- [2] "BFS vs DFS for Binary Tree - GeeksforGeeks." [Online]. Available: <https://www.geeksforgeeks.org/bfs-vs-dfs-binary-tree/>. [Accessed: 16-Mar-2018].
- [3] Unknown, "Maze Generator." [Online]. Available: <http://www.delorie.com/game-room/mazes/genmaze.cgi>. [Accessed: 19-Mar-2018].

Agent AI Pathfinder Report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

60/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

GRADING FORM: SMU GENERIC_GJ

STEFFAN BENSON

60

RESEARCH-INFORMED LITERATURE



Refs included -You've reviewed the required SE principles related to code dev but include a ref to these.

3.5

KNOWLEDGE AND UNDERSTANDING

 Good understanding of request data structures and their use. Dfs algorithm also well understood. No shortest path? Good understanding of requisite SE principles and C++ code.

9

ANALYSIS

 Analysis of problem and selection of appropriate algorithms and data structures in evidence. Choice of animation method not argued for fully. Separation of solution structure not implemented.

10

PRACTICAL APPLICATION AND DEP.

 The solution loads multiple mazes and searches them correctly. Finds objects. Good. Some editing of code to change selected maze is needed - why not add a file selector method? Animation of search could be improved - single step only. Try to separate your code into appropriate .h/.cpp files. Comments could be better. No shortest path functionality at present.

34

SKILLS FOR PROFESSIONAL PRAC.

 Refs appropriate for this. Report well presented and structured. Evidence of good SE practice.

3.5