

AI Pathfinder

by Mathew Wade Alex Taylor

FILE	2885_MATHEW_WADE_ALEX_TAYLOR_AI_PATHFINDER_110679_126522 5034.DOCX		
TIME SUBMITTED	29-MAR-2018 12:16PM (UTC+0100)	WORD COUNT	1002
SUBMISSION ID	85798092	CHARACTER COUNT	6522

Agent AI: AI Pathfinder

Student Name: Mathew Taylor

P Number: 149885

Module Title: Game Programming and Graphics

Assignment Title: OpenGL Based 3D Game

Module Lecturer's Name: Gordon Dickers

Content

Table of Figures.....	2
Introduction	3
Algorithms.....	3
User Guide	3
Program Listing	4
whereCanIGo()	4
mazeSetup()	6
finalOutput()	7
main()	8
Testing.....	10
Images	11
References	13

Table of Figures

Figure 1 – Maze Selection	11
Figure 2 - Input Item to search for.....	11
Figure 3 - Maze Searched by AI with start position, AI position and the end point position output ...	12
Figure 4 - Output of the path the AI followed to find the item in the maze (see figure 1 for the rest of the maze above)	12

Introduction

This report will cover the algorithms implemented into the AI pathfinder. A brief user guide, the program listing, any testing that was undertaken during the development and screenshots of the final working product.

Algorithms

The main algorithm used in the AI pathfinder is the DFS (Depth First Search). The DFS starts at the root node and travels down a path, should it find a dead end, it will return to the last unexplored location[1]. The DFS is used in the AI pathfinder and handles all the locations for the AI to move to in the maze. When the AI has reached a dead end in the maze the top value on the stack will be popped off the stack, allowing the AI to return to a location with another unexplored path[2]. This algorithm is run until the item in the maze has been found[1].

User Guide

Upon running the AI pathfinder program, the user will be prompted with a choice of which maze they would like the AI pathfinder to search, the different maze options are listed above the input. The chosen maze must be input correctly (if the choice is "Maze.maz" then the user must input "Maze.maz"). Should the user input an incorrect choice, the console will ask the user to input the choice again.

Once the maze choice has been selected the user will be asked to input what item the AI pathfinder must search through the maze for. Both maze options currently only have "F" items in them but searching for other items will not crash the program, only return a message stating that the item could not be found.

During the search of the maze the AI character is represented as "@", the paths the AI character travelled to that led to dead ends are represented with "o" and the direct path taken from the start to the item by the AI is represented by "x". The start position is and "S" and the item currently in the maze is an "F".

Program Listing

whereCanIGo()

The whereCanIGo function handles the checking of whether the AI can move up, down, left or right in the maze, this is dependent on where a space is in the maze.

```
//Handles the movement of the AI
void whereCanIGo()
{
    char lookUp, lookDown, lookLeft, lookRight; //Holds the next position to
    check in the maze
    bool moveUp = false, moveRight = false, moveDown = false, moveLeft = false;

    lookUp = maze[WIDTH*(ai.y - 1) + ai.x]; //Set lookUp to the position above
    the AI's current position
    lookDown = maze[WIDTH*(ai.y + 1) + ai.x]; //Set lookDown to the position
    below the AI's current position
    lookLeft = maze[WIDTH*ai.y + (ai.x - 1)]; //Set lookLeft to the position to
    the left of the AI's current position
    lookRight = maze[WIDTH*ai.y + (ai.x + 1)]; //Set lookRight to the position to
    the right of the AI's current position

    if (lookLeft == ' ' || lookLeft == choice[0]) //Check if there is space or
    and F left of the AI's current position
    {
        if (lookLeft == choice[0]) //Check if the item to be found is to the
        left of the AI
        {
            itemFound = true;
            //cout << "LEFT" << endl;
            nextPos.x = (ai.x - 1);
            nextPos.y = (ai.y);
            dfsStack.push(nextPos); //Add the next position for the AI to be moved
            to to the stack
            moveLeft = true;
        }
    }
}
```

```

    }

    if (lookDown == ' ' || lookDown == choice[0]) //Check if there is space or
    and F down of the AI's current position
    {
        if (lookDown == choice[0]) //Check if the item to be found is below
        the AI
        {
            itemFound = true;
            //cout << "DOWN" << endl;
            nextPos.x = (ai.x);
            nextPos.y = (ai.y + 1);
            dfsStack.push(nextPos); //Add the next position for the AI to be moved
            to the stack
            moveDown = true;
        }

        if (lookRight == ' ' || lookRight == choice[0]) //Check if there is space or
        and F right of the AI's current position
        {
            if (lookRight == choice[0]) //Check if the item to be found is to the
            right of the AI
            {
                itemFound = true;
                //cout << "RIGHT" << endl;
                nextPos.x = (ai.x + 1);
                nextPos.y = (ai.y);
                dfsStack.push(nextPos); //Add the next position for the AI to be moved
                to the stack
                moveRight = true;
            }

            if (lookUp == ' ' || lookUp == choice[0]) //Check if there is space or and F
            up of the AI's current position
            {
                if (lookUp == choice[0]) //Check if the item to be found is above the
                AI
                {
                    itemFound = true;
                    //cout << "UP" << endl;
                    nextPos.x = (ai.x);
                    nextPos.y = (ai.y - 1);
                    dfsStack.push(nextPos); //Add the next position for the AI to be moved
                    to the stack
                    moveUp = true;
                }

                if (!moveUp && !moveDown && !moveLeft && !moveRight) //Check if the AI has
                run out of possible moves
                {
                    dfsStack.pop(); //pop off the stack
                    maze[WIDTH*ai.y + ai.x] = '.'; //set dead end position to a '.'
                }
            }
        }
    }
}

```

mazeSetup()

The mazeSetup function handles the inputs from the user and the initial generation of the maze by reading from the file containing the chosen maze. This function will set the start position of the AI.

```
//Initial set up of the maze before the actual AI search
void mazeSetup()
{
    string line;
    FILE *ptr = fopen("MazeList.txt", "r"); //Opens the .txt file containing the
names of the possible maze to search

    char buffer[MAX_CHAR_SIZE];
    bool fileOpen = false;

    do
    {
        cout << "Input which file to search: " << endl;

        do
        {
            fscanf(ptr, "%s", buffer);
            cout << buffer << endl; //console outputs the list of mazes the
user may choose from

        } while (!feof(ptr));

        //scanf("%s", choice);
        cin >> choice; //User inputs maze choice
        system("cls"); //Clear the console

        ifstream myfile(choice); //Opens the file chosen by the user

        //Check if file was able to be opened
        if (myfile.is_open())
        {
            fileOpen = true;
            int count = 0;

            while (getline(myfile, line))
            {
                line.copy(&maze[WIDTH*count], WIDTH, 0); //Read in the
maze and add it to the maze variable
                count++;
            }

            myfile.close();
        }
        else cout << "Unable to open file" << endl;

        rewind(ptr); //Reset the pointer

    } while (!fileOpen);

    cout << "Input what item to search for in the maze: " << endl;
    scanf("%s", choice); //User Inputs what item to search for

    //Output maze
    for (int y = 0; y<HEIGHT; y++)
    {
        for (int x = 0; x < WIDTH; x++)
```

```

        {
            if (maze[WIDTH*y + x] == 'S') //Check if start position has
            been found
            {
                //Set start point
                start.x = x;
                start.y = y;
            }

            if (maze[WIDTH*y + x] == choice[0]) //Check if exit has been
            found
            {
                //Set finish point
                finish.x = x;
                finish.y = y;
            }
            cout << maze[WIDTH*y + x] << " ";
        }
        cout << endl;
    }
    cout << "Start located at position:(" << start.x << "," << start.y << ")" <<
    endl; //Output start position

    //Set AI's position to the start position
    ai.x = start.x;
    ai.y = start.y;

    dfsStack.push(start); //Add the start position to the stack

    cout << "AI position:(" << ai.x << "," << ai.y << ")" << endl; //Output the
    AI's current position

    system("cls"); //Clear the console
}

```

finalOutput()

The finalOutput function handles all the outputs after the search is complete. This function outputs whether the item has been found or not and if the item has been found, the location of the item. The last output is the path the AI took to get from the start position to the item.

```

//Handles the final outputs, like where the end points is
void finalOutput(bool finished, bool itemFound)
{
    //Check if the item has been found and finish is true
    if (finished == true && itemFound == true)
        cout << "End Point Found: (" << ai.x << "," << ai.y << ")" << endl;
    //Output the Finish Point
    else
        cout << "Item could not be found" << endl;

    int xCount = 0;

    //Count the total number of 'x' in the maze
    for (int y = 0; y < HEIGHT; y++)
    {
        for (int x = 0; x < WIDTH; x++)
        {
            if (maze[WIDTH*y + x] == 'x')
            {
                xCount++;
            }
        }
    }
}

```

```

    }
    cout << endl;
}

pos *criticalPath = new pos[xCount]; //Holds the positions of the path the AI
took from the start to the finish

for (int count = 0; count < xCount; count++)
{
    criticalPath[count] = dfsStack.top(); //Grab the top value from the
stack
    dfsStack.pop(); //pop off the top value from the stack
}

cout << "Critical Path: ";
cout << endl;

//Output critical path found
for (int count2 = xCount - 1; count2 > 0; count2--)
{
    if (count2 > 1)
        cout << "(" << criticalPath[count2].x << ", " <<
criticalPath[count2].y << ") -> ";
    else
        cout << "(" << ai.x << ", " << ai.y << ")";
}
cout << endl;
}

```

main()

The main handles the actual search algorithm and prints out the maze, while constantly clearing the console so the user can see the maze updating as the AI searches through it.

```

//Handles the actual search algorithm
int main(void)
{
    mazeSetUp(); //Runs the main setup

    bool finished = false; //Determines if the search has been finished or not

    while (!finished)
    {
        system("cls"); //Clear the console

        maze[WIDTH*ai.y + ai.x] = 'x'; //Change the location of the ai in the
maze to an x
        whereCanIGo();

        ai = dfsStack.top(); //Set ai to the top value of the dfsStack

        //Output maze
        for (int y = 0; y<HEIGHT; y++)
        {
            for (int x = 0; x < WIDTH; x++)
            {
                if (x == ai.x && y == ai.y)
                    cout << "@" << " "; //Output the AI in the maze
                else

```

```

        cout << maze[WIDTH*y + x] << " "; //Space out the
maze to make it more pleasing to look at
    }
    cout << endl;
}

cout << "Start located at position:(" << start.x << "," << start.y <<
")" << endl; //Output the Start point
cout << "AI position:(" << ai.x << "," << ai.y << ")" << endl;
//Output the AI's current position

if (itemFound) //Check if the AI has found the item
{
    finished = true;
}

if (ai.x == start.x && ai.y == start.y) //Check if the AI has returned
to the start of the maze
{
    finished = true;
}

};

finalOutput(finished, itemFound); //Run the final output

return 0;
}

```

Testing

Test No.	Test Description	Expected Result	Actual Result	Pass/Fail
1	Maze Selection.	Can choose different maze.	Can choose different maze.	Pass.
2	Item Selection.	Can choose an item to find.	Can choose an item to find.	Pass.
3	AI can movement in the maze.	AI can move up, down, left and right.	AI can move up, down, left and right.	Pass.
4	AI will stay within the walls.	AI doesn't travel through walls.	AI doesn't travel through walls.	Pass.
5	AI moves backwards if dead end is found.	AI moves backwards if dead end is found.	AI moves backwards if dead end is found.	Pass.
6	AI can detect if the item to be found is next to it.	AI stops searching through the maze if item is next to it.	AI stops searching through the maze if item is next to it.	Pass.
7	User inputs something other than one of the maze options.	Console outputs a message asking the user to input a choice again.	Console outputs a message asking the user to input a choice again.	Pass.
8	User inputs an item that does not exist to find.	Console outputs message saying that the item couldn't be found.	Console outputs message saying that the item couldn't be found.	Pass.
9	AI leaves a trail of x's to show the path it's taken.	User can see the x's appear on screen as the AI moves.	User can see the x's appear on screen as the AI moves.	Pass.
10	AI leaves o's when it reaches a dead end.	User can see the o's appear on screen as the AI leaves a dead end.	User can see the o's appear on screen as the AI leaves a dead end.	Pass.

Images

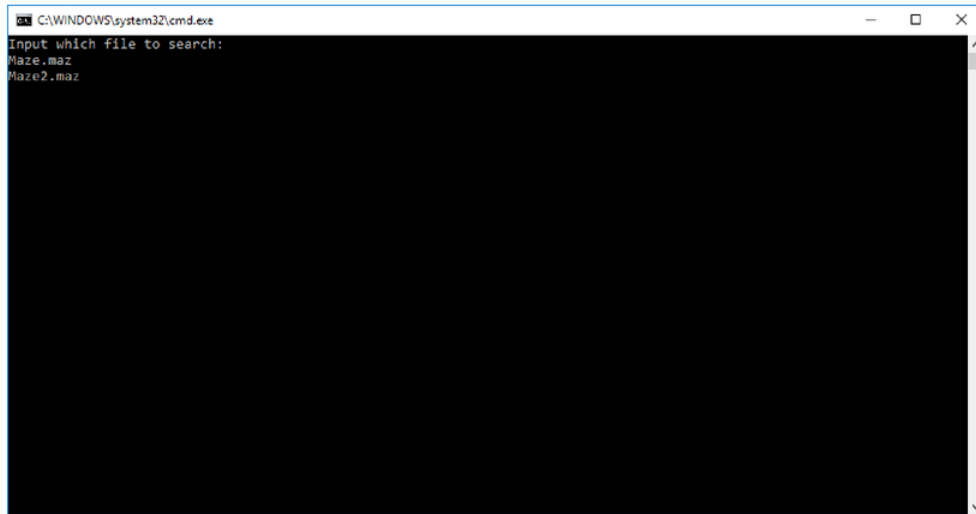


Figure 1 – Maze Selection

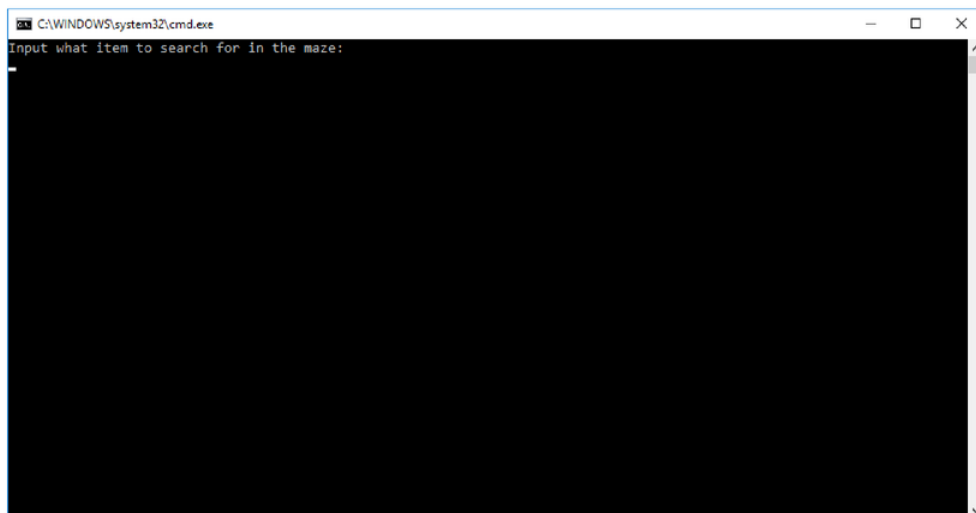


Figure 2 - Input Item to search for

References

- [1] K. Moore and K. Jenkinson, "Depth-First Search (DFS) | Brilliant Math & Science Wiki," *Brilliant*, 2002. [Online]. Available: <https://brilliant.org/wiki/depth-first-search-dfs/>. [Accessed: 29-Mar-2018].
- [2] L. L. Larmore, "DFS and BFS Algorithms using Stacks and Queues," pp. 1–4.

FINAL GRADE

GENERAL COMMENTS

Instructor

64/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

GRADING FORM: SMU GENERIC_GJ

MATHEW WADE ALEX TAYLOR

63.50

RESEARCH-INFORMED LITERATURE



Some refs included - good. Key algorithms identified and implemented. You've reviewed required SE principles related to code dev but include more refs to these please.

3

KNOWLEDGE AND UNDERSTANDING



Good understanding of request data structures and their use. Dfs algorithm also well understood. No shortest path? Good understanding of SE principles and C++ code demonstrated but it would have been nice to see you attempt a shortest path algorithm..

10

ANALYSIS



You've demonstrated a good analysis of the problem and chosen suitable algorithms and data structures in your solution. You might have included further work on choosing/selecting a suitable shortest path algorithm.

10

PRACTICAL APPLICATION AND DEP.



Algorithms implemented correctly. Good commenting in places - try to include a description of the function at the beginning of the function definition.; Evidence of separation of interface from

37

implementation. Maze is loaded and animated. Item found. Well done.

SKILLS FOR PROFESSIONAL PRAC.



Some refs included - good. Report well structures and presented. Evidence of good SE practice in implementation- well structured code. You've reviewed required SE principles related to code dev but include more refs to these please.

3.5