

Agent AI Report

by Ethan Randell

FILE	4154_ETHAN_RANDELL_AGENT_AI_REPORT_110679_440536059.DOCX		
TIME SUBMITTED	29-MAR-2018 11:23PM (UTC+0100)	WORD COUNT	1336
SUBMISSION ID	85845010	CHARACTER COUNT	6656

Assignment Title: Agent AI: Search algorithms

Student Name: Ethan Randell

Student Number: P150496

Course Name: Agent AI

Course Number: M5X01022

Contents page

Introduction	
Description of Algorithms used + Important Algorithms for the solution	
User Guide	
Test Schedule + Screen shots of working program	
Bibliography	
Appendix	

Introduction

For this assignment the student has been asked to make a program that uses artificial intelligence to try and find an object inside a maze which has been created in a notepad file which will be read in by the program.

Description of Algorithms used

```
while (!keyFound) {
    current = dfsStack.top(); //top plus pop equals a stack pop operation

    maze[WIDTH*current.y + current.x] = 'o';
    WhereCanIGo(current);
    system("cls");
    //maze print and start position search
    for (int y = 0; y<HEIGHT; y++)
    {
        for (int x = 0; x<WIDTH; x++)
        {
            cout << maze[WIDTH*y + x];
        }
        cout << endl;
    }
    //push all adjacent nodes //mark where you are //draw maze with your current position marked
    cout << "Popping (" << current.x << ", " << current.y << ")" << endl;
}
```

The first algorithm used is a while loop this loop deals with all of the printing of the maze coordinates and printing out the actual maze as well and which path the artificial intelligence is taking.

Inside the while loop the stack pop operation has been implemented as well what this does it will take the position of "S" and until it can find "X" it will pop off all the positions of where it's been.

```
} if (myfile.is_open())
{
    int count = 0;
    while (getline(myfile, line))
    {
        cout << line << endl;
        line.copy(&maze[WIDTH*count], WIDTH, 0);
        count++;
    }
    myfile.close();
}
else cout << "Unable to open file";
cout << endl << endl << endl;
```

This IF statement allows the program to go and actually open up the .maz file which contains the maze itself, the maze was constructed using the * characters for the walls of the maze. If the IF statement can't find the file it will print out the statement unable to open file.

```

//maze print and start position search
for (int y = 0; y<HEIGHT; y++)
{
    for (int x = 0; x<WIDTH; x++)
    {
        if (maze[WIDTH*y + x] == 'S')
        {
            start.x = x;
            start.y = y;
        }
        cout << maze[WIDTH*y + x];
    }
    cout << endl;
}
cout << "Start located at position: (" << start.x << ", " << start.y << ") of maze" << endl;

```

This for loop searches for the character S (S is representing the artificial intelligence) inside the maze once S has been found the program will print out the position of S.

```

if (lookUp == ' ')
{
    location.y = (current.y - 1);
    location.x = current.x;
    dfsStack.push(location); //push (WIDTH*(current.y - 1) + current.x)
    moveUp = true;
}

```

What this IF statement does it looks for a space character if the artificial intelligence cannot find the space character but finds a star character it will have to go down the Y axis by minus one space. For the other if statements the current x's and y's are different.

These if statements are repeated for look up, look down, look left and look right

Important Algorithms for the Solution

One of the Important algorithms for the solution to work is the Key found while loop because this while loop one has the dfsStackPop algorithm this algorithm pops off the old position and goes to the next X and Y position.

```

while (!keyFound) {
    //push all adjacent nodes
    current = dfsStack.top(); //top plus pop equals a stack pop operation
    maze[WIDTH*current.y + current.x] = 'o';
    whereCanGo(current);
    system("cls");
    //maze print and start position search
    for (int y = 0; y<HEIGHT; y++)
    {
        for (int x = 0; x<WIDTH; x++)
        {
            cout << maze[WIDTH*y + x];
        }
        cout << endl;
    }
    //push all adjacent nodes //mark where you are //draw maze with your current position marked
    cout << "Popping (" << current.x << ", " << current.y << ")" << endl;
}

```

```

1  if (lookRight == ' ')
2  {
3      location.y = current.y;
4      location.x = (current.x + 1);
5      dfsStack.push(location); //push (WIDTH*(current.y) + current.x + 1)
6      moveRight = true;
7  }
8
9  if (!moveUp && !moveDown && !moveLeft && !moveRight)
10 {
11     dfsStack.pop();
12 }
13
14 if (lookUp == 'X')
15 {
16     location.y = (current.y - 1);
17     location.x = current.x;
18     dfsStack.push(location); //push (WIDTH*(current.y - 1) + current.x)
19     moveUp = true;
20     keyFound = true;
21     cout << "Keyfound (" << current.x << ", " << current.y << ")" << endl;
22 }
23

```

These if statements are very important to the solution these if statements actually allow the artificial to move so if the artificial intelligence can find a space to the right it will go one space to the right by using the dfs Stack Push. It's the same with the "X" version of the if statement as well this if statement tells the artificial intelligence what to find and once the artificial has found character "X" it will write out the coordinates of "X".

User Guide

Sadly, due to time constraints a main menu has not been made to allow the user to choose different mazes to load in.

To start the program	Buttons to press: Control + F5
----------------------	--------------------------------

Test 1: The Artificial Intelligence can actually find “X”

As shown in the screenshot the Artificial Intelligence can find X and also leads a trail of which path it has taken to get to X.

As shown in the screenshot the maze can print off the starting position of "S" (to allow it to print off the starting position you will have to remove the system("cls") function on line 71).

Test 3: Can print out "X" character coordinate

```
keyfound (1,12)
*****
*****
*****
00000000000000***
*****o***
*****o***
*****oo**
*****oo**
*****oo**
*****oo**
*****oo*
*****oo*
*****oo*
X*****o*
o*****o*
o*****o*
000000000000000*
*****
Popping (1,12)
Press any key to continue
```

As shown in the screenshot the maze can print off the position of "X" (to allow it to print off the starting position you will have to remove the system("cls") function on line 71).

Test 4: Can Print out the maze

```
*****  
*****  
oooooooooooo**  
* oooooooooo  
*      o  
*      *  
* oooooooooo  
*      oo  
* oooooooooo  
*      oo  
*      ***oo  
*      oo  
*      oo  
*****oo  
*****oo  
*****oo  
*X*****o  
*o*****o  
*o*****o  
*o*****o  
*oooooooooooo*  
*****  
  
Popping (1,12)  
Press any key to continue . . .
```

As you can see the program can print of the maze if the `cls` system line is not included into the program it will repeatedly draw the maze and also print out the position of the artificial intelligence, but with the inclusion of the system `cls` the maze will be drawn very quickly and imitate the artificial intelligence moving around the maze in real time.

Bibliography

Bibliography

- [1] G. Dickens, "Agent AI," UWTSD, [Online]. Available: <https://moodle.uwtsd.ac.uk/course/view.php?id=2266>. [Accessed 14 March 2018].

Appendix

This code was influenced by this source [1]

```
// reading a text file
#define WIDTH 16
#define HEIGHT 16
#include <iostream>
#include <fstream>
#include <string>
#include <stack>
using namespace std;
char maze[WIDTH*HEIGHT];
typedef struct p {
    int x;
    int y;
}position;

void WhereCanIGo(position);
bool moveLeft;
bool moveRight;
bool moveUp;
bool moveDown;
bool keyFound;

std::stack<position> dfsStack; //stack for dfs traversal
position start;

int main(void) {
    int startX; int startY;
    string line;
    ifstream myfile("maze1.maz");
    if (myfile.is_open())
    {
        int count = 0;
        while (getline(myfile, line))
```

```

        {
            cout << line << endl;
            line.copy(&maze[WIDTH*count], WIDTH, 0);
            count++;
        }
        myfile.close();
    }
    else cout << "Unable to open file";
    cout << endl << endl << endl;
    //maze print and start position search
    for (int y = 0; y<HEIGHT; y++)
    {
        for (int x = 0; x<WIDTH; x++)
        {
            if (maze[WIDTH*y + x] == 'S')
            {
                start.x = x;
                start.y = y;
            }
            cout << maze[WIDTH*y + x];
        }
        cout << endl;
    }
    cout << "Start located at position: (" << start.x << "," << start.y << ") of
maze" << endl;
    // maze has been loaded time to start the DFS Stack

    position current; //initialise current position
    current.x = start.x;
    current.y = start.y;

    dfsStack.push(current); //dfs initialise stack with start position

                                //dfs while loop
    while (!keyFound) {
        current = dfsStack.top(); //top plus pop equals a stack pop operation

        maze[WIDTH*current.y + current.x] = 'o';
        WhereCanIGo(current);
        system("cls");
        //maze print and start position search
        for (int y = 0; y<HEIGHT; y++)
        {
            for (int x = 0; x<WIDTH; x++)
            {
                cout << maze[WIDTH*y + x];
            }
            cout << endl;
        }
        //push all adjacent nodes //mark where you are //draw maze with your
current position marked
        cout << "Popping (" << current.x << "," << current.y << ")" << endl;
    }

    return 0;
}

```

```

void WhereCanIGo(position current) {
    position location; // temporarily holds adjacent position

    char lookUp, lookDown, lookLeft, lookRight;
    lookUp = maze[WIDTH*(current.y - 1) + current.x];
    cout << lookUp << endl;
    lookDown = maze[WIDTH*(current.y + 1) + current.x];
    cout << lookDown << endl;
    lookLeft = maze[WIDTH*(current.y) + current.x - 1];
    cout << lookLeft << endl;
    lookRight = maze[WIDTH*(current.y) + current.x + 1];
    cout << lookRight << endl;
    //4 tests
    if (lookUp == ' ')
    {
        location.y = (current.y - 1);
        location.x = current.x;
        dfsStack.push(location); //push (WIDTH*(current.y - 1) + current.x)
        moveUp = true;
    }

    if (lookDown == ' ')
    {
        location.y = (current.y + 1);
        location.x = current.x;
        dfsStack.push(location); //push (WIDTH*(current.y + 1) + current.x)
        moveDown = true;
    }

    if (lookLeft == ' ')
    {
        location.y = current.y;
        location.x = (current.x - 1);
        dfsStack.push(location); //push (WIDTH*(current.y) + current.x - 1)
        moveLeft = true;
    }

    if (lookRight == ' ')
    {
        location.y = current.y;
        location.x = (current.x + 1);
        dfsStack.push(location); //push (WIDTH*(current.y) + current.x + 1)
        moveRight = true;
    }

    if (!moveUp && !moveDown && !moveLeft && !moveRight)
    {
        dfsStack.pop();
    }

    if (lookUp == 'X')

```

```

{
    location.y = (current.y - 1);
    location.x = current.x;
    dfsStack.push(location); //push (WIDTH*(current.y - 1) + current.x)
    moveUp = true;
    keyFound = true;
    cout << "Keyfound (" << current.x << "," << current.y << ")" << endl;
}

if (lookDown == 'X')
{
    location.y = (current.y + 1);
    location.x = current.x;
    dfsStack.push(location); //push (WIDTH*(current.y + 1) + current.x)
    moveDown = true;
    keyFound = true;
    cout << "Keyfound (" << current.x << "," << current.y << ")" << endl;
}

if (lookLeft == 'X')
{
    location.y = current.y;
    location.x = (current.x - 1);
    dfsStack.push(location); //push (WIDTH*(current.y ) + current.x - 1)
    moveLeft = true;
    keyFound = true;
    cout << "Keyfound (" << current.x << "," << current.y << ")" << endl;
}

if (lookRight == 'X')
{
    location.y = current.y;
    location.x = (current.x + 1);
    dfsStack.push(location); //push (WIDTH*(current.y) + current.x + 1)
    moveRight = true;
    keyFound = true;
    cout << "Keyfound (" << current.x << "," << current.y << ")" << endl;
}

}

//void
//
//
//
//
//
//
//void AdjacentPositions()

```

```
//{  
//  
//}
```

Agent AI Report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

46/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

GRADING FORM: SMU GENERIC_GJ

ETHAN RANDELL

46

RESEARCH-INFORMED LITERATURE



A bibliography is included but try to review further texts/ papers. You've obviously reviewed required SE principles related to code dev as demonstrated in your implementation so include a ref to these in your report or code.

2

KNOWLEDGE AND UNDERSTANDING



Good understanding of request data structures and their use. Dfs algorithm understood but your understanding of the implementation of this in C++ needs to be improved. No shortest path?

7

ANALYSIS



Analysis of problem and selection of appropriate algorithms in evidence. Further work needed to solve slightly buggy implementation of dfs.

7

PRACTICAL APPLICATION AND DEP.



Maze is loaded from a file and can be modified. Good. Maze traversal is animated. Well done. You do have some issues with backtracking and your implementation is faulty in that it doesn't work with some maze patterns - needs more work here. Otherwise a good attempt showing essential requisite SE skills. Try to separate interface from implementation by using separate .h/.cpp files.

28

SKILLS FOR PROFESSIONAL PRAC.



Your report is structured and you have included a bibliography but at this level you really ought to be including more references, eg Norvig and Russell,

2

etc. Also reference C++ texts since your solution is in C++.