Project Report

เรื่อง

Rat Maize

เสนอ

รศ.ดร.รังสิพรรณ มฤคทัต

จัดทำโดย

นางสาวประภาสิริ	วรรณวงศ์	6513116
นายพุฒิพงศ์	โยมะบุตร	6513134
นายปาฏิหาริย์	เขม่นกิจ	6513170
นายภัทรดนัย	สอนสว่าง	6513172

รายงานนี้เป็นส่วนหนึ่งของวิชา

Data Structure and Algorithm

ภาคเรียนที่ 2 ปีการศึกษา 2566

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithm (EGCO 221) จัดทำขึ้นเพื่อใช้อธิบายประกอบการทำงานโปรแกรมของ Rat in a Maize ซึ่งประกอบไปด้วยส่วนของคู่มือการใช้งานโปรแกรมเบื้องต้น และการอธิบายในส่วนของ Code และ Algorithm รวมไปถึงข้อจำกัดต่าง ๆ ในการใช้งานโปรแกรม

ทางคณะผู้จัดทำขอขอบพระคุณ รศ.ดร.รังสิพรรณ มฤคทัต ผู้ให้ความรู้ และแนวทางการศึกษา สุดท้ายนี้ทางคณะผู้จัดทำหวังว่ารายงานฉบับนี้จะสามารถเป็นประโยชน์ไม่มากก็น้อยแก่ผู้อ่านทุกท่าน

ขอขอบพระคุณ

คณะผู้จัดทำ

สารบัญ

คู่มือการใช้งาน (Short user manual)

1. เมื่อกด run program จะแสดงผลข้อความดังกล่าว

```
=====                                  Welcome to Rat in a Maize game                                        =====
<<<                                            How to play                                                   >>>
      In the maize will have a rat 'R' you need to move a rat to find the food
      We have 4 file for you to choose layout of maize 'maize_1.txt' 'maize_2.txt' 'maize_3.txt' 'maize_4.txt'
      You need to choose 1 file before play this game Ex. >> Input filename = maize_1.txt
      In a game input 'L' to move left, 'R' to move right, 'U' to move up and 'D' to move down
      If you are lazy Input 'A' to play auto Good Luck!!!

Input file name =
```

เพื่อให้ผู้เล่นทราบภาพรวมเกม และกฎกติกาการเล่น ดังนี้

1.1 ในเกมนี้จะมีหนู ซึ่งจะแสดงผลด้วยตัว “R” ซึ่งเราต้องพามันไปหาอาหาร

1.2 มีทั้งหมด 4 ไฟล์ที่มีหน้าตาแตกต่างกัน 'maize_1.txt' 'maize_2.txt' 'maize_3.txt' และ 'maize_4.txt' ให้ผู้ใช้เลือกว่าจะเล่นอันไหน โดยให้เลือกเพียง 1 ไฟล์เท่านั้น โดยการพิมพ์ตามรูปแบบ(format) ดังกล่าว เช่น “Input file name = maize_1.txt”

1.3 เพื่อเดินเกม ผู้เล่นสามารถพิมพ์ ‘L’ เพื่อเคลื่อนที่หนูไปทางด้านซ้าย, พิมพ์ ‘R’ เพื่อเคลื่อนที่ไปยังด้านขวา, พิมพ์ ‘U’ เพื่อเคลื่อนที่ไปยังด้านบน, พิมพ์ ‘D’ เพื่อเคลื่อนที่ไปยังด้านล่าง และสามารถพิมพ์ ‘A’ เพื่อกดเล่นอัตโนมัติได้(auto) โดยในการพิมพ์แต่ละตัวอักษรสามารถพิมพ์ได้ทั้งพิมพ์เล็ก และพิมพ์ใหญ่

```
Input file name = maize_1.txt

       col_0  col_1  col_2  col_3  col_4
row_0      0      1      1      1      F
row_1      1      R      1      0      1
row_2      0      1      0      F      1
row_3      1      1      1      0      1
row_4      0      0      1      1      1
User input 1 >> Enter move (U = up, D = down, L = left, R = right, A = Auto)
Input >> :l
       col_0  col_1  col_2  col_3  col_4
row_0      0      1      1      1      F
row_1      R      1      1      0      1
row_2      0      1      0      F      1
row_3      1      1      1      0      1
row_4      0      0      1      1      1
```

```
User input 16 >> Enter move (U = up, D = down, L = left, R = right, A = Auto)
Input >> :a
Rat path ->
Start -> (row 2, col 3, R)
Right -> (row 2, col 4)
Down  -> (row 3, col 4)
Down  -> (row 4, col 4)
Left  -> (row 4, col 3)
Left  -> (row 4, col 2)
Up    -> (row 3, col 2)
Left  -> (row 3, col 1)
Up    -> (row 2, col 1)
Up    -> (row 1, col 1)
Right -> (row 1, col 2)
Up    -> (row 0, col 2)
Right -> (row 0, col 3)
Right -> (row 0, col 4)
End   -> (row 0, col 4, F)
```

(ตัวอย่างการกดเล่นอัตโนมัติ)

1.4 เมื่อหาอาหารครบจนหมด จะแสดงผลให้เลือกไฟล์ maize ใหม่เพื่อเล่นต่อ

```
==== Finding Food 2 ====
        col_0  col_1  col_2  col_3  col_4
row_0       0      1      1      1      R
row_1       1      1      1      0      1
row_2       0      1      0      1      1
row_3       1      1      1      0      1
row_4       0      0      1      1      1

=====                                  Welcome to Rat in a Maize game                                      =====
<<<                                             How to play                                                 >>>
      In the maize will have a rat 'R' you need to move a rat to find the food
      We have 4 file for you to choose layout of maize 'maize_1.txt' 'maize_2.txt' 'maize_3.txt' 'maize_4.txt'
      You need to choose 1 file before play this game Ex. >> Input filename = maize_1.txt
      In a game input 'L' to move left, 'R' to move right, 'U' to move up and 'D' to move down
      If you are lazy Input 'A' to play auto Good Luck!!!

Input file name =
```

Data Structure ที่นำมาใช้ทั้งหมด

1. โครงสร้างข้อมูลใน Class play

```
package Project1;
import java.util.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Scanner;

3 usages
public class play {
    13 usages
    private int start_row;
    13 usages
    private int start_col;
    6 usages
    private int count_food,round = 1,food=1;
    11 usages
    private String input;
    29 usages
    List<List<Character>> maze = new ArrayList<>();
    1 usage
    play(int s_row, int s_col, List<List<Character>> maze, int food){...}
    Codeium: Refactor Explain Docstring
    1 usage
    public void play_game() {...}
    Codeium: Refactor Explain Docstring
    4 usages
    public void swap_position(int row_2, int col_2) {...}
    Codeium: Refactor Explain Docstring
    2 usages
    public boolean check(int row, int col) {...}
    Codeium: Refactor Explain Docstring
    5 usages
    public boolean findPath(int row, int col, int endRow, int endCol, Deque<String> path) {...}
    Codeium: Refactor Explain Docstring
    5 usages
    public static void printMaze(List<List<Character>> maze,int count) {...}

}
```

เป็น Class ที่เก็บข้อมูลทั้งหมดทั้งหมดที่จะนำมาใช้ โดยใช้วิธีการจัดเก็บข้อมูลดังนี้

1.1 ArrayList ใช้ในการเก็บข้อมูลของ Maze โดยเลือกใช้เพราะง่ายต่อการเพิ่มข้อมูลและการเรียกใช้ข้อมูล เพราะ ArrayList สามารถเข้าถึงข้อมูลใน index ที่ระบุได้ทันที โดยในโปรแกรมมีการใช้ method ของ ArrayList ดังนี้

1.1.1 add() : เป็นฟังก์ชั่นที่ใช้ในการเพิ่มข้อมูลไปใน ArrayList

1.1.2 get() : เป็นฟังก์ชั่นที่ใช้ในการเข้าถึงข้อมูลใน ArrayList

1.1.3 size() : เป็นฟังก์ชั่นที่ใช้ในการคืนค่าขนาดของ ArrayList

1.1.4 set() : เป็นฟังก์ชั่นที่ใช้ในการแก้ไขค่าที่อยู่ใน ArrayList

1.2 ArrayDeque ใช้ในการเก็บข้อมูลของ Path ที่เดิน โดยเลือกใช้เพราะสามารถเพิ่มหรือนำข้อมูลออกได้จากทั้ง 2 ทาง(First / Last) โดยในโปรแกรมมีการใช้ method ของ ArrayDeque ดังนี้

1.2.1 push() : เป็นฟังก์ชั่นที่ใช้ในการเพิ่มข้อมูลไปใน ArrayDeque

1.2.2 pop() : เป็นฟังก์ชั่นที่ใช้ในการคืนค่าและลบข้อมูลใน ArrayDeque

1.2.3 isEmpty() : เป็นฟังก์ชั่นที่ใช้เช็คว่าใน ArrayDeque มีข้อมูลอยู่ไหม

1.2.4 size() : เป็นฟังก์ชั่นที่ใช้ในการคืนค่าขนาดของ ArrayDeque

2. โครงสร้างข้อมูลใน Class project 1

```
package Project1;
import java.util.*;
import java.io.*;

public class Project1{
    7 usages
    public static int count=0;
  ✨ Codeium: Refactor Explain Docstring
    public static void main(String[] args) {...}
}
```

เป็น Class ที่ทำหน้าที่เป็น Main Class ที่ทำหน้าที่เปิดไฟล์และรับค่าจากไฟล์มาเก็บไว้ในArrayList ก่อนจะส่งต่อมายัง Class play

อัลกอริทึมที่ใช้ในการแก้ปัญหา

การแก้ปัญหานี้เลือกใช้ ArrayDeque ในการเก็บเส้นทาง เนื่องจากสามารถเข้าถึงข้อมูลจากข้างบนและข้างล่าง โดยเราจะใช้ AarrayDeque 1 ตัวในการแก้ปัญหานี้ และตัวแปรชื่อ “path” คือ ArrayDeque ที่ไว้ใช้เก็บเส้นทาง

โดยขั้นตอนในการประมวลผลมีดังนี้

1. เริ่มจากเราหาก่อนว่า Food(F) อยู่ที่ตำแหน่งไหนโดยใช้ for-loop ในการค้นหา โดยตำแหน่ง(I,j)คือตำแหน่งของFood ที่หาเจอ และ start_row, start_col คือตำแหน่งของ rat(R) จากนั้นเมื่อเจอ Food แล้วส่งตำแหน่ง(i,j) ไป Function findPath โดย Argument ที่ส่งไปมีทั้งหมด 5 ตัว start_row, start_col, i , j ,path โดยที่กล่าวมาข้างต้นเป็นวิธีการของ Forwarding Step

2. หลังจากเราส่ง Argument ไปเรียบร้อยแล้ว Function findPath ก็จะเริ่มทำงานของมัน โดยเริ่มจากหาเส้นทางจาก parameter ใน Function คือ row, col ไปยัง endRow, endCol จากนั้นใน Function มีเงื่อนไข if-else อยู่โดย if-else มีทั้งหมด 4 ตัว คือ เงื่อนไขสำหรับ Rat เดินทางไปทาง ล่าง, ขวา, ลง, ซ้าย โดย if-else 4 ตัวนี้จะอยู่ข้างใน if(check(row, col) && maze.get(row).get(col) != '0') และมี if-else สำหรับตรวจว่าตอนนี้ row==endRow และ col==endCol และตำแหน่งของ row, col เท่ากับ ‘F’ หรือไม่

3. อัลกอริทึมของ Function นี้จะเริ่มหาเส้นทางทุกเส้นทางที่เป็นไปได้เพื่อหาFood โดยถ้าเริ่มหาเส้นทางไปแล้วไม่เจอหรือไม่ใช่Food มันจะถอยหลังกลับมาที่จุดเดิม และยกเลิกเส้นทางนั้นแล้วหาเส้นทางใหม่ เมื่อเจอ Food แล้ว มันจะ return ค่า true มาเรื่อยๆ เพราะว่า if-else ของเรานั้นจะทำงานได้ก็ต่อเมื่อเงื่อนไขเป็น true แล้วเก็บค่า เป็น String ลงใน Path ทีเป็น ArrayDeque โดยวิธีการที่กล่าวมสาทั้งหมดเป็นวิธีการของ Backtracking(recursive)

4. ถ้าใน maze นั้นมี Food ทั้งหมดมากกว่า 1 ตัว loopจะเริ่มจะตำแหน่งของตัวที่ 1 จากนั้นFunction findPath จะหาเส้นทางเพื่อหา Food ตัวที่ 1 พอจบตัวที่ 1 เสร็จ loop จะหาตำแหน่งของตัวถัดไป จากนั้นใช้Function findPath เพื่อหาเส้นทางเหมือนเดิม ถ้าไม่มีตัวถัดไปแล้วloopจะหยุดการทำงานไปเอง

5. ถ้าเมื่อ loop หาตำแหน่งของ Food แล้วจากนั้น Function findPath ทำทุกเส้นทางเพื่อหาFood แล้วได้ loopนี้จะหยุดทำงานและจะprint ว่า “No Solution!”

6. Recursive function (ฟังก์ชันรีเคอร์ชีพ) คือฟังก์ชันที่เรียกใช้ตัวเองเพื่อแก้ปัญหาบางอย่างโดยการแบ่งปัญหาให้เล็กลง จากนั้นรวมผลลัพธ์เข้าด้วยกัน ส่วนของ path เมื่อเข้า Function findPath เสร็จแล้ว Function นี้จะทำงาน แบบ Recursive function โดยมันจะเรียกตัวเองซ้ำ ๆ จนกว่าจะเจอ Food เพื่อหยุดการเรียกใช้ตัวเอง และไม่ให้โปรแกรมเรียกใช้ตัวเองแบบไม่รู้จบ จากนั้นนำเส้นทางที่เจอ Food ใส่ลงใน path โดยใช้ .push() และ print ออกมาเพื่อ โดยใช้ .pop()

แสดงการทำงานของ Source Code

1. Class Project1 (Main class) : ทำหน้าที่เป็น main class ของโปรแกรม รับ Input เป็นชื่อไฟล์และตรวจสอบ Input

```java
package Project1;
import java.util.*;
import java.io.*;

public class Project1{
    7 usages
    public static int count=0; //ใช้เก็บจำนวน column
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        boolean opensuccess = false;
        String map;
        String path = "src/main/java/Project1/";

        System.out.println("=====                                Welcome to Rat in a Maize game                                =====");
        System.out.println("<<<                                        How to play                                            >>>");
        System.out.println("      In the maize will have a rat 'R' you need to move a rat to find the food ");
        System.out.println("      We have 4 file for you to choose layout of maize 'maize_1.txt' 'maize_2.txt' 'maize_3.txt' 'maize_4.txt'");
        System.out.println("      You need to choose 1 file before play this game Ex. >> Input filename = maize_1.txt");
        System.out.println("      In a game input 'L' to move left, 'R' to move right, 'U' to move up and 'D' to move down");
        System.out.println("      If you are lazy Input 'A' to play auto Good Luck!!!");
        System.out.println();
        System.out.print("Input file name = ");
        map = sc.nextLine();
        String file_name = path+map;

        int count_row=0,count_food=0; //ใช้เก็บจำนวน row และ Food
        char temp ;
        int start_row=0,start_col=0; //ใช้ระบุตำแหน่งของ rat
        int end_row=0, end_col=0;  //ใช้ระบุตำแหน่งของ food
```

```java
List<List<Character>> maze = new ArrayList<>();

while(!opensuccess) {
    try{
        File file = new File(file_name);
        Scanner filescan = new Scanner(file);
        opensuccess = true;
        System.out.print( "\n");
        while(filescan.hasNext()){
            List<Character> row = new ArrayList<>();
            String line = filescan.nextLine();
            String[] col = line.split( regex: ",");
            count = col.length;
            for(int i=0;i<count;i++){
               temp = col[i].trim().charAt(0);
                if(temp=='R'){
                    start_col = i;
                    start_row = count_row;
                }
                if(temp=='F'){
                    end_col = i;
                    end_row = count_row;
                    count_food++;
                }
                row.add(temp);
            }
            maze.add(row);
            count_row++;
        }
```

รับ input เป็นชื่อไฟล์และอ่านไฟล์แต่ละบรรทัด เอาเข้าไปเก็บไว้ใน Arraylist เพื่อใช้เป็น layout ของ row แต่ละแถวใน maze และเก็บค่า start_row,start_col เป็นตำแหน่งของ rat ใน maze และ end_col,end_row เป็นตำแหน่งของ food ใน maze

```
            }
            maze.add(row);
            count_row++;
        }
        play.printMaze(maze,count);//function ที่ใช้ในการปริ้ท maze ออกมา

        play play_auto  = new play(start_row, start_col, maze,count_food, end_row, end_col );
        play_auto.play_game();    //function ที่ใช้ในการเล่นเกม
        filescan.close();

    }catch (FileNotFoundException e) {
        System.out.println("Not Found File!!!");
        System.out.print("Please Input New file name  = ");
        map = sc.nextLine();
        file_name = path + map;
    }
}
```

ส่งตำแหน่งของ rat และ food ให้กับ class play

ใช้ตรวจสอบว่า Input ของ user ถูกหรือไม่ มีไฟล์อยู่ภายในเครื่องหรือไม่

2. Class Play : ทำหน้าที่เป็น class ที่ใช้ในการเล่น rat in a maize

```
public class play {
    13 usages
    private int start_row;
    13 usages
    private int start_col;
    1 usage
    private int end_row, end_col;
    6 usages
    private int count_food,round = 1,food=1;    //ใช้เก็บจำนวนรอบที่ user เล่น และจำนวนอาหารที่เจอแล้ว

    11 usages
    private String input; //ใช้เก็บ input ที่ user ป้อนเข้ามา

    29 usages
    List<List<Character>> maze = new ArrayList<>();
    1 usage
    play(int s_row, int s_col, List<List<Character>> maze, int food, int e_row, int e_col) {
        this.start_row = s_row;
        this.start_col = s_col;
        this.maze = maze;
        this.count_food = food;
        this.end_row = e_row;
        this.end_col = e_col;
    }
```

//Constructor

หาก User เล่นแบบ Manual

```
Scanner sc = new Scanner(System.in);
while (true) {
    System.out.print("User input " + round + " >> Enter move " );
    System.out.println("(U = up, D = down, L = left, R = right, A = Auto)");
    System.out.print("Input >> :");
    input = sc.nextLine();
    round ++;
    int row_2;
    int col_2;
    if (Objects.equals(input, b: "R") || Objects.equals(input, b: "r")) {
        row_2 = start_row;
        col_2 = start_col + 1;
        swap_position(row_2, col_2);
    }
    if (Objects.equals(input, b: "L") || Objects.equals(input, b: "l")) {
        row_2 = start_row;
        col_2 = start_col - 1;
        swap_position(row_2, col_2);
    }
    if (Objects.equals(input, b: "U") || Objects.equals(input, b: "u")) {
        row_2 = start_row - 1;
        col_2 = start_col;
        swap_position(row_2, col_2);
    }
    if (Objects.equals(input, b: "D") || Objects.equals(input, b: "d")) {
        row_2 = start_row + 1;
        col_2 = start_col;
        swap_position(row_2, col_2);
    }
```

เช็ค Input ของ User

หากเป็น ‘R’ จะขยับตำแหน่งของ rat ไปทางขวา

หากเป็น ‘L’ จะขยับตำแหน่งของ rat ไปทางซ้าย

หากเป็น ‘U’ จะขยับตำแหน่งของ rat ไปข้างบน

หากเป็น ‘D’ จะขยับตำแหน่งของ rat ไปข้างล่าง

Function ที่ใช้ในการขยับตำแหน่ง

```
public void swap_position(int row_2, int col_2) {
    if (check(row_2, col_2)) {
        if (maze.get(row_2).get(col_2) != '0' && maze.get(row_2).get(col_2) != 'F') {
            char temp = maze.get(start_row).get(start_col);
            maze.get(start_row).set(start_col, maze.get(row_2).get(col_2));
            maze.get(row_2).set(col_2, temp);
            printMaze(maze, Project1.count);
            start_row = row_2;
            start_col = col_2;
        } else {
            System.out.println("Cannot move\n");
        }
        if (maze.get(row_2).get(col_2) == 'F') {
            maze.get(start_row).set(start_col, '1');
            maze.get(row_2).set(col_2, 'R');
            printMaze(maze, Project1.count);
            start_row = row_2;
            start_col = col_2;
            System.out.println("+++++ Find Food +++++");
            food++;
            printMaze(maze, Project1.count);
            count_food--;
            if (count_food == 0) {
                System.out.println("----------------------------------------");
                Project1.main( args: null);
            }
        }
    } else {
        System.out.println("Cannot move");
    }
}
```

swap_position() จะรับค่า row และ col ของตำแหน่งที่จะเคลื่อนที่ไป และเก็บไปลงที่ row_2 และ col_2 โดยจะแบ่งเป็น 2 กรณี ดังนี้

1.หากตำแหน่งที่จะเคลื่อนที่ไปไม่ใช่ ‘0’ และ ‘F’ จะสร้างตัวแปร temp ไว้เก็บตำแหน่งของ rat และ set ตำแหน่งของ rat เป็น row_2 และ col_2 และเปลี่ยน row_2 และ col_2 เป็นตำแหน่งของ rat แทน

2.หากตำแหน่งที่จะเคลื่อนที่ไปคือ ‘F’ จะ set ตำแหน่งเก่าของ rat เป็น 1 และ set ตำแหน่งของ F เป็น rat แทน

หาก User เล่นแบบ Auto

```
if(Objects.equals(input, b: "A") || Objects.equals(input, b: "a")) {
    while (count_food != 0) {
        Deque<String> path = new ArrayDeque<>(); //สร้าง ArrayDeque เอาไว้เก็บ
        boolean foundPath = false; //สร้าง foundpath เอาไว้กำหนดเงื่อนไข
        System.out.println("Rat path ->");
        System.out.println("Start -> (row " + start_row + ", col " + start_col + ", R)");
        for (int i = 0; i < maze.size(); i++) {
            for (int j = 0; j < maze.get(i).size(); j++) {
                if (maze.get(i).get(j) == 'F') {
                    if (findPath(start_row, start_col, i, j, path)) {
                        while (!path.isEmpty()) {
                            String poppedElement = path.pop();
                            System.out.println("" + poppedElement);
                        }
                        System.out.println();
                        System.out.println("==== Finding Food " + food + " ====");
                        food++;

                        start_row = i;
                        start_col = j;
                        printMaze(maze, Project1.count);
                        System.out.println();
                        count_food--;
                        foundPath = true; // Set flag to true indicating path found
                        if(count_food==0) Project1.main( args: null);//ถ้าไม่มี F ใน maizeแล้ว
                        break; // Exit inner loop
                                                                    จะจบการเล่น
                    }
                }
            }
            if (foundPath) break; //ถ้า foundpath = true จะหยุด for loop
        }
```

ใช้ while loop ในการเช็คว่ายังมี F อยู่ใน maze และจะใช้ for loop หาตำแหน่งของ F เมื่อเจอ F จะเรียกใช้ findpath() เมื่อ return ค่า findpath() ออกมาเป็น true จะทำการ pop() สิ่งที่อยู่ใน ArrayDeque ออกมาจนหมด และเปลี่ยน foundpath ให้เป็น true

```
        }
        if (foundPath) break; // Exit outer loop if path found
    }
    if (!foundPath) { //ถ้า foundpath = false จะหยุดการเล่น
        System.out.println();
        System.out.println("No solution !!!!");
        System.out.println();
        Project1.main( args: null);
    }
}
```

หากไม่มีการเปลี่ยน foundpath ให้เป็น true จะจบการเล่นทันที

```
public boolean findPath(int row, int col, int endRow, int endCol, Deque<String> path) {
    if (row == endRow && col == endCol && maze.get(row).get(col)=='F' ) {
        maze.get(row).set(col, 'R');
        path.push( e: "End   -> (row " + row + ", col " + col  + ", F)" );
        return true;
    }
    if ( check(row, col) && maze.get(row).get(col) != '0') {
        char originalCell = maze.get(row).get(col);
        maze.get(row).set(col, '0'); // Mark the current cell as visited
        if(findPath( row: row + 1, col, endRow, endCol, path)){
            if (originalCell != 'F') maze.get(row).set(col, '1'); // Unmark the current cell if it's not food
            path.push( e: "Down  -> (row " + (row+1) + ", col " + col + ")");
            return true;
        }
        if(findPath(row, col: col+1, endRow, endCol, path)){
            if (originalCell != 'F') maze.get(row).set(col, '1'); // Unmark the current cell if it's not food
            path.push( e: "Right -> (row " + row + ", col " + (col+1) + ")");
            return true;
        }
        if(findPath( row: row - 1, col, endRow, endCol, path)){
            if (originalCell != 'F') maze.get(row).set(col, '1'); // Unmark the current cell if it's not food
            path.push( e: "Up    -> (row " + (row-1) + ", col " + col + ")");
            return true;
        }
        if(findPath(row , col: col-1, endRow, endCol, path)){
            if (originalCell != 'F') maze.get(row).set(col, '1'); // Unmark the current cell if it's not food
            path.push( e: "Left  -> (row " + row + ", col " + (col-1)+ ")");
            return true;
        }
```

Findpath() จะทำงานแบบ recursive function จะเรียกตัวเองซ้ำจนกว่าจะเจอ F ซึ่ง findpath() นี้จะทำการค้นหาเส้นทางโดยลงข้างล่างก่อนแล้วจึงไปทางขวา ขึ้นบน และไปทางซ้ายตามลำดับ และเมื่อเจอ F แล้วเรียบร้อย จะ push ตำแหน่งลงใน arraydeque และจะ return true ให้กับ findpath() ที่เรียกตนใช้งาน findpath() ก่อนหน้านั้นจะ push ตำแหน่งของตน และ return true ให้ไปเรื่อย ๆ จนกว่าจะถึง Findpath()แรกที่เรียกใช้

```
        }
        maze.get(row).set(col, originalCell); // Unmark the current cell if the path is not found
    }
    return false; //ถ้าไม่เข้าเงื่อนไขใด ๆ จะ return false
}
```

```java
public static void printMaze(List<List<Character>> maze,int count) {
    int round = 0;
    System.out.print("        ");
    for (int i=0; i<count; i++){
        System.out.print("col_"+i+"  ");
    }
    System.out.println();
    for (List<Character> row : maze) {
        System.out.print("row_"+ round +"     ");
        round++;
        for (char cell : row) {
            System.out.print(cell + "      ");
        }
        System.out.println();  // Move to the next line after printing each row
    }
}
```

//เป็น function ที่เอาไว้ print maze ออกมา

```java
2 usages
public boolean check(int row, int col) {
    return row >= 0 && row < maze.size() && col >= 0 && col < maze.get(row).size();
}
```

check () เอาไว้เช็คว่าตำแหน่งที่เอา check() ไปใช้เช็คนั้นมีอยู่ใน maze หรือไม่

ตัวอย่างการแสดงผล

ตัวอย่างไฟล์ maize_1.txt

```
Input file name = maize_1.txt

        col_0  col_1  col_2  col_3  col_4
row_0     0      1      1      1      F
row_1     1      R      1      0      1
row_2     0      1      0      F      1
row_3     1      1      1      0      1
row_4     0      0      1      1      1
User input 1 >> Enter move (U = up, D = down, L = left, R = right, A = Auto)
Input >> :
```

เมื่อ Input a ให้โปรแกรมรัน auto

```
        col_0  col_1  col_2  col_3  col_4
row_0     0      1      1      1      F
row_1     1      R      1      0      1
row_2     0      1      0      F      1
row_3     1      1      1      0      1
row_4     0      0      1      1      1
```

Forwarding Step เราจะใช้ for loop ในการเช็คแต่ละแถวว่ามี F อยู่หรือไม่ และเก็บตำแหน่งของ F ไว้

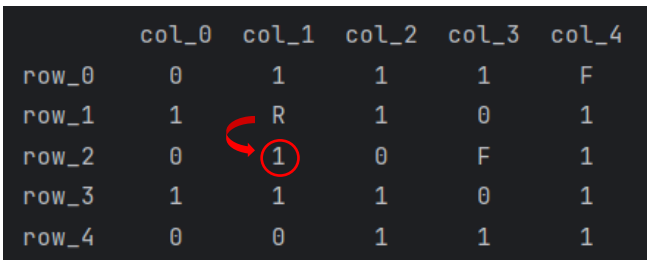
```
Rat path ->
Start -> (row 1, col 1, R)
Down  -> (row 2, col 1)
Down  -> (row 3, col 1)
Right -> (row 3, col 2)
Down  -> (row 4, col 2)
Right -> (row 4, col 3)
Right -> (row 4, col 4)
Up    -> (row 3, col 4)
Up    -> (row 2, col 4)
Up    -> (row 1, col 4)
Up    -> (row 0, col 4)
End   -> (row 0, col 4, F)

==== Finding Food 1 ====
        col_0  col_1  col_2  col_3  col_4
row_0     0      1      1      1      R
row_1     1      1      1      0      1
row_2     0      1      0      F      1
row_3     1      1      1      0      1
row_4     0      0      1      1      1
```

Backtracking Step เราจะใช้ function findpath ในการหาเส้นทางการเดินของ Rat โดยเราจะใช้ push() เส้นทางการเดินทางเข้าไปใน Stack และ ใช้ pop() ในการ print เส้นทางการเดินทางออกมา

ซึ่งจะอธิบายต่อหน้าถัดไป

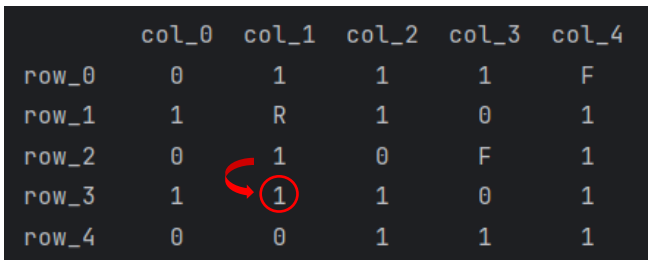
โดย function findpath จะทำงาน Recursive algorithm โดยจะเรียกทำตัวเองซ้ำ ๆ และจะรวมโดยการหา path จะทำจากเดินลงล่างก่อนหากไปต่อไม่ได้จะเดินทางขวาและหากไปต่อไม่ได้อีกจะเดินขึ้นและทางขวาตามลำดับ

ตัวอย่าง

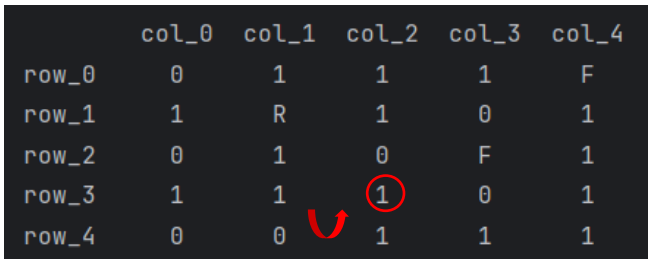


	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

ใน function findpath เราจะเริ่มเดินลงก่อนหาก สามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

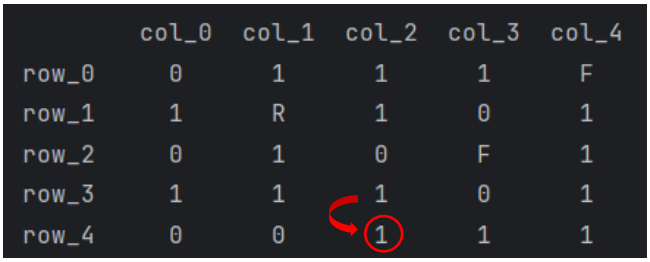


	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1



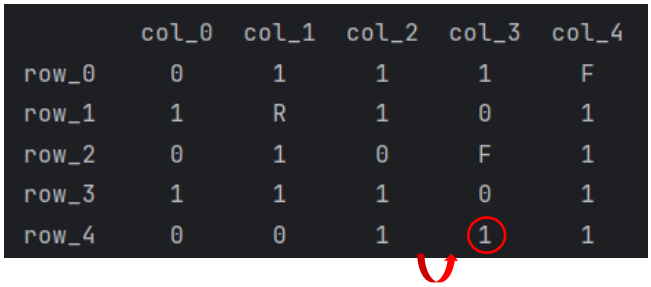
	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

เมื่อไม่สามารถเดินลงได้ เราจะเริ่มเดินทางขวาต่อ ถ้าสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ



	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

สามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ



	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

เมื่อไม่สามารถเดินลงได้ เราจะเริ่มเดินทางขวาต่อ ถ้าสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

เมื่อไม่สามารถเดินลงย้อนทางเดิมและเดินขวาได้ เราจะเริ่มเดินขึ้นบนต่อ ถ้าสามารถขึ้นลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	F
row_1	1	R	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

เมื่อเจอ F แล้วเราจะ push ตำแหน่งเก็บไว้ใน Stack และ return true กลับไปให้ กับ findpath ก่อนหน้าที่เรียกใช้ เพื่อ push ตำแหน่งลงใน stack จนครบ

D=down, R=right, U=up,L=left

node	function	Stack	push
1	Findpath(1)	D	
2	Findpath(2)	D D	
3	Findpath(3)	D D R	
4	Findpath(4)	D D R D	
5	Findpath(5)	D D R D R R	
6	Findpath(6)	D D R D R R U	
7	Findpath(7)	D D R D R R U U	
8	Findpath(8)	D D R D R R U U U	
9	Findpath(9)	D D R D R R U U U U	
9	Return true To Findpath(8)	D D R D R R U U U	U
8	Return true To Findpath(7)	D D R D R R U U	U
7	Return true To Findpath(6)	D D R D R R U	U
6	Return true To Findpath(4)	D D R D R R	U
5	Return true To Findpath(4)	D D R D R	R
4	Return true To Findpath(3)	D D R D	R
3	Return true To Findpath(2)	D D R	D
2	Return true To Findpath(1)	D D	R
1		D	D
	End of function		D

แต่สิ่งที่เอาไปเก็บใน Stack จริงๆคือ

D = Down -> (row x,col y)

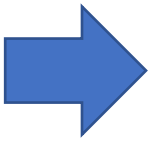
R = Right -> (row x,col y)

U = Up -> (row x,col y)

L = Left -> (row x,col y)

Down -> (row x,col y)
Down -> (row x,col y)
Right -> (row x,col y)
Down -> (row x,col y)
Right -> (row x,col y)
Right -> (row x,col y)
Up -> (row x,col y)
Up -> (row x,col y)
Up -> (row x,col y)
Up -> (row x,col y)

POP



Down -> (row x,col y)
Right -> (row x,col y)
Down -> (row x,col y)
Right -> (row x,col y)
Right -> (row x,col y)
Up -> (row x,col y)
Up -> (row x,col y)
Up -> (row x,col y)
Up -> (row x,col y)

จึงได้ output ดังนี้

```
Down  -> (row 2, col 1)
Down  -> (row 3, col 1)
Right -> (row 3, col 2)
Down  -> (row 4, col 2)
Right -> (row 4, col 3)
Right -> (row 4, col 4)
Up    -> (row 3, col 4)
Up    -> (row 2, col 4)
Up    -> (row 1, col 4)
Up    -> (row 0, col 4)
```

```
Rat path ->
Start -> (row 1, col 1, R)
Down  -> (row 2, col 1)
Down  -> (row 3, col 1)
Right -> (row 3, col 2)
Down  -> (row 4, col 2)
Right -> (row 4, col 3)
Right -> (row 4, col 4)
Up    -> (row 3, col 4)
Up    -> (row 2, col 4)
Up    -> (row 1, col 4)
Up    -> (row 0, col 4)
End   -> (row 0, col 4, F)

==== Finding Food 1 ====
       col_0  col_1  col_2  col_3  col_4
row_0      0      1      1      1      R
row_1      1      1      1      0      1
row_2      0      1      0      F      1
row_3      1      1      1      0      1
row_4      0      0      1      1      1
```

หา F อีกตัว

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	R
row_1	1	1	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	R
row_1	1	1	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

Forwarding Step เราจะใช้ for loop ในการเช็คแต่ละแถวว่ามี F อยู่หรือไม่ และเก็บตำแหน่งของ F ไว้

เริ่มใช้ function findpath

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	R
row_1	1	1	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

ใน function findpath เราจะเริ่มเดินลงก่อนหากสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	R
row_1	1	1	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

	col_0	col_1	col_2	col_3	col_4
row_0	0	1	1	1	R
row_1	1	1	1	0	1
row_2	0	1	0	F	1
row_3	1	1	1	0	1
row_4	0	0	1	1	1

node	function	Stack	push
1	Findpath(1)	D	
2	Findpath(2)	D D	
3	Findpath(3)	D D L	
2	Return true To findpath(2)	D D	L
1	Return true To findpath(1)	D	D
	End of function		D

จึงได้ output ดังนี้

```
Down  -> (row 1, col 4)
Down  -> (row 2, col 4)
Left  -> (row 2, col 3)
```

```
Rat path ->
Start -> (row 0, col 4, R)
Down  -> (row 1, col 4)
Down  -> (row 2, col 4)
Left  -> (row 2, col 3)
End   -> (row 2, col 3, F)

==== Finding Food 2 ====
        col_0  col_1  col_2  col_3  col_4
row_0     0      1      1      1      1
row_1     1      1      1      0      1
row_2     0      1      0      R      1
row_3     1      1      1      0      1
row_4     0      0      1      1      1
```

ตัวอย่างไฟล์ maize_2.txt

```
Input file name = maize_2.txt

        col_0  col_1  col_2  col_3  col_4  col_5
row_0     0      0      1      1      1      R
row_1     1      1      0      1      0      1
row_2     0      F      1      0      1      1
row_3     1      1      0      1      F      1
User input 1 >> Enter move (U = up, D = down, L = left, R = right, A = Auto)
Input >> :
```

เมื่อ Input a ให้โปรแกรมรัน auto

จะเหมือนกับ maize_1.txt

```
        col_0  col_1  col_2  col_3  col_4  col_5
row_0     0      0      1      1      1      R
row_1     1      1      0      1      0      1
row_2     0      F      1      0      1      1
row_3     1      1      0      1      F      1
```

Forwarding Step เราจะใช้ for loop ในการเช็คแต่ละแถวว่ามี F อยู่หรือไม่ และเก็บตำแหน่งของ F ไว้

เริ่มใช้ function findpath

```
        col_0  col_1  col_2  col_3  col_4  col_5
row_0     0      0      1      1      1      R
row_1     1      1      0      1      0      1
row_2     0      F      1      0      1      1
row_3     1      1      0      1      F      1
```

```
        col_0  col_1  col_2  col_3  col_4  col_5
row_0     0      0      1      1      1      R
row_1     1      1      0      1      0      1
row_2     0      F      1      0      1      1
row_3     1      1      0      1      F      1
```

```
        col_0  col_1  col_2  col_3  col_4  col_5
row_0     0      0      1      1      1      R
row_1     1      1      0      1      0      1
row_2     0      F      1      0      1      1
row_3     1      1      0      1      F      1
```

ใน function findpath เราจะเริ่มเดินลงก่อนหากสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

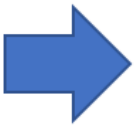
	col_0	col_1	col_2	col_3	col_4	col_5
row_0	0	0	1	1	1	R
row_1	1	1	0	1	0	1
row_2	0	F	1	0	1	1
row_3	1	1	0	1	F	1

เมื่อไม่สามารถเดินลง เดินขวา และเดินขึ้นได้ ก็จะเดินซ้ายแทน

node	function	Stack	push
1	Findpath(1)	D	
2	Findpath(2)	D D	
3	Findpath(3)	D D D	
4	Findpath(4)	D D D L	
4	Return true To findpath(3)	D D D	L
3	Return true To findpath(2)	D D	D
2	Return true To findpath(1)	D	D
1	End of function		D

Down -> (row x,col y)
Down -> (row x,col y)
Down -> (row x,col y)
Left -> (row x,col y)

POP



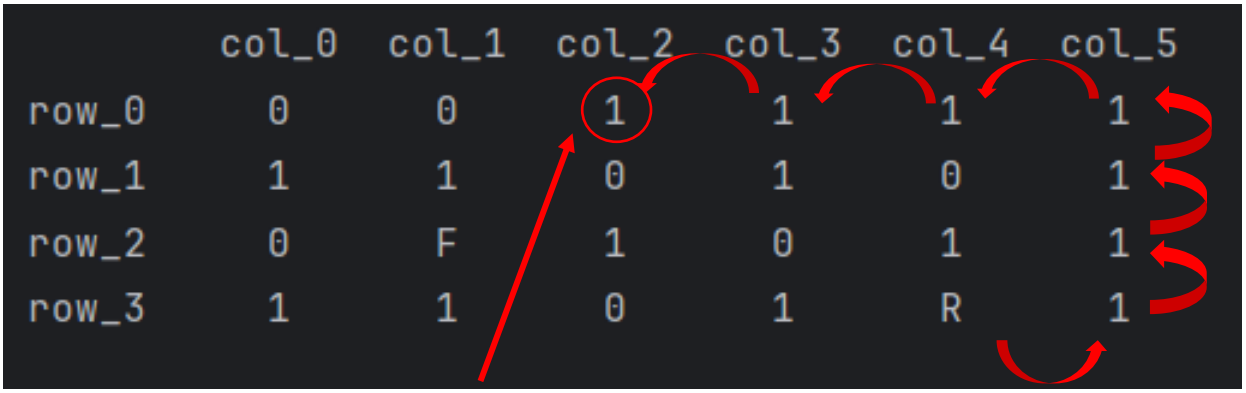
Down -> (row x,col y)
Down -> (row x,col y)
Left -> (row x,col y)

จึงได้ output ดังนี้

```
Down  -> (row 1, col 5)
Down  -> (row 2, col 5)
Down  -> (row 3, col 5)
Left  -> (row 3, col 4)
```

```
Rat path ->
Start -> (row 0, col 5, R)
Down  -> (row 1, col 5)
Down  -> (row 2, col 5)
Down  -> (row 3, col 5)
Left  -> (row 3, col 4)
End   -> (row 3, col 4, F)

==== Finding Food 1 ====
        col_0  col_1  col_2  col_3  col_4  col_5
row_0     0      0      1      1      1      1
row_1     1      1      0      1      0      1
row_2     0      F      1      0      1      1
row_3     1      1      0      1      R      1
```

เริ่มหา F ตัวถัดไป



เมื่อ R มาอยู่ในตำแหน่งนี้ ไม่สามารถเดินลง เดินขึ้น เดินซ้าย หรือเดินขวาได้ จึงจะ return false ให้กับ function findpath ก่อนหน้า จึงจะไม่มีการ push ลงใน stack จึงได้ output เป็น No solution!!!

node	function	Stack	push
1	Findpath(1)	R	
2	Findpath(2)	R U	
3	Findpath(3)	R U U	
4	Findpath(4)	R U U U	
5	Findpath(5)	R U U U	
6	Findpath(6)	R U U U L	
7	Findpath(7)	R U U U L L	
7	Findpath(8)	R U U U L L L	
6	Return false To Findpath(6)	R U U U L L	
5	Return false To Findpath(8)	R U U U L	
4	Return false To Findpath(7)	R U U U	
3	Return false To Findpath(6)	R U U	
2	Return false	R U	

	To Findpath(4)		
1	Return false To Findpath(4)	R	
	End of function		

ข้อจำกัดโปรแกรม

1. ถ้ากด Auto จะหา Food ทั้งหมดในทีเดียว ไม่สามารถหาแค่ 1 Food ได้

2. เมื่อเริ่มโปรแกรมโปรแกรมจะวนลูปไปเรื่อย ๆ ไม่สามารถออกได้

3. ถ้า maze มีขนาดใหญ่มาก อาจทำให้เกิด OutOfMemoryError จากการทำ recursive ใน findPath ได้

4. ถ้า maze มีขนาดใหญ่มากจะใช้เวลาในการเพิ่มข้อมูลนานเพราะมีการเก็บ maze เป็น ArrayList

5. ในโหมด Auto path ที่เจออาจไม่ใช่เส้นทางที่สั้นที่สุด

บรรณานุกรม(References)

Rat in a Maze. (2023). [ออนไลน์]. ได้จาก: https://www.enjoyalgorithms.com/blog/rat-in-a-maze [สืบค้นเมื่อ 22 กุมภาพันธ์ 2567].

Java Program for Rat in a Maze | Backtracking-2. (2022). [ออนไลน์]. ได้จาก: https://www.geeksforgeeks.org/java-program-for-rat-in-a-maze-backtracking-2/ [สืบค้นเมื่อ 22 กุมภาพันธ์ 2567].

Rat in a Maze. (2023). [ออนไลน์]. ได้จาก: https://www.geeksforgeeks.org/rat-in-a-maze/ [สืบค้นเมื่อ 22 กุมภาพันธ์ 2567].

Recursive function. (2024). [ออนไลน์]. ได้จาก: https://marcuscode.com/2020/12/recursive-function [สืบค้นเมื่อ 8 มีนาคม 2567].