

**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 ซึ่งประกอบไปด้วยส่วนของคู่มือการใช้งานโปรแกรมเบื้องต้นและการอธิบายในส่วนของ Code และ Algorithm รวมไปถึงข้อจำกัดต่าง ๆ ในการใช้งานโปรแกรม

     ทางคณะผู้จัดทำขอขอบพระคุณ รศ.ดร.รังสิพรรณ มฤคทัต ผู้ให้ความรู้ และแนวทางการศึกษา สุดท้ายนี้ทางคณะผู้จัดทำหวังว่ารายงานฉบับนี้จะสามารถเป็นประโยชน์ไม่มากก็น้อยแก่ผู้อ่านทุกท่าน

**ขอขอบพระคุณ**

**คณะผู้จัดทำ**

**สารบัญ**

**เรื่อง หน้า**

**4.1 คู่มือการใช้งาน (Short user manual) 4 – 5**

**4.2 Data Structure ที่นำมาใช้ทั้งหมด 6 – 7**

1. โครงสร้างข้อมูลใน Class play

2. โครงสร้างข้อมูลใน Class project 1

**4.3 อัลกอริทึมที่ใช้ในการแก้ปัญหา 8 – 9**

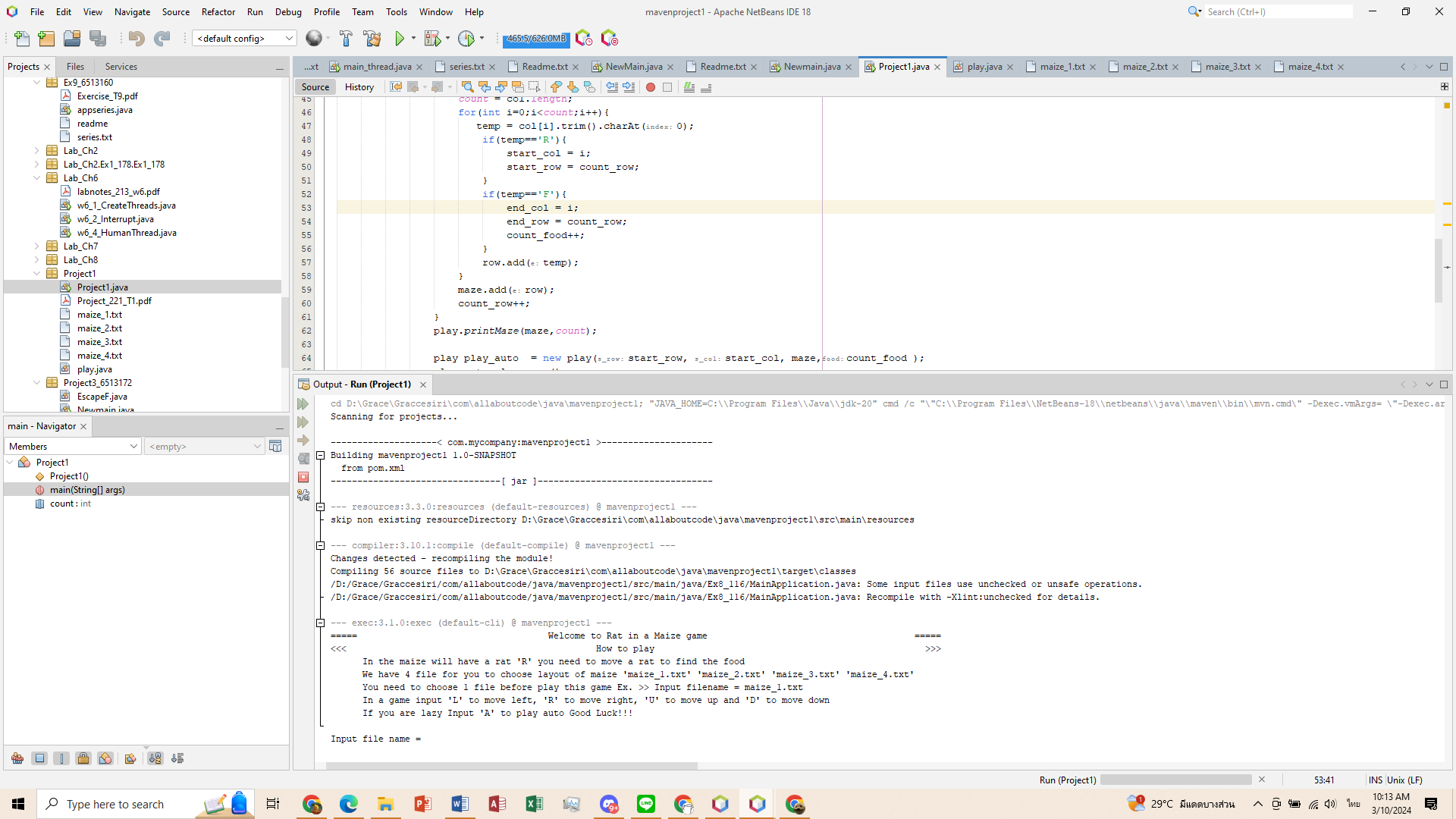
**4.4 แสดงการทำงานของ Source Code 10 - 25**

**4.5 ข้อจำกัดโปรแกรม 26**

**4.6 บรรณานุกรม 27**

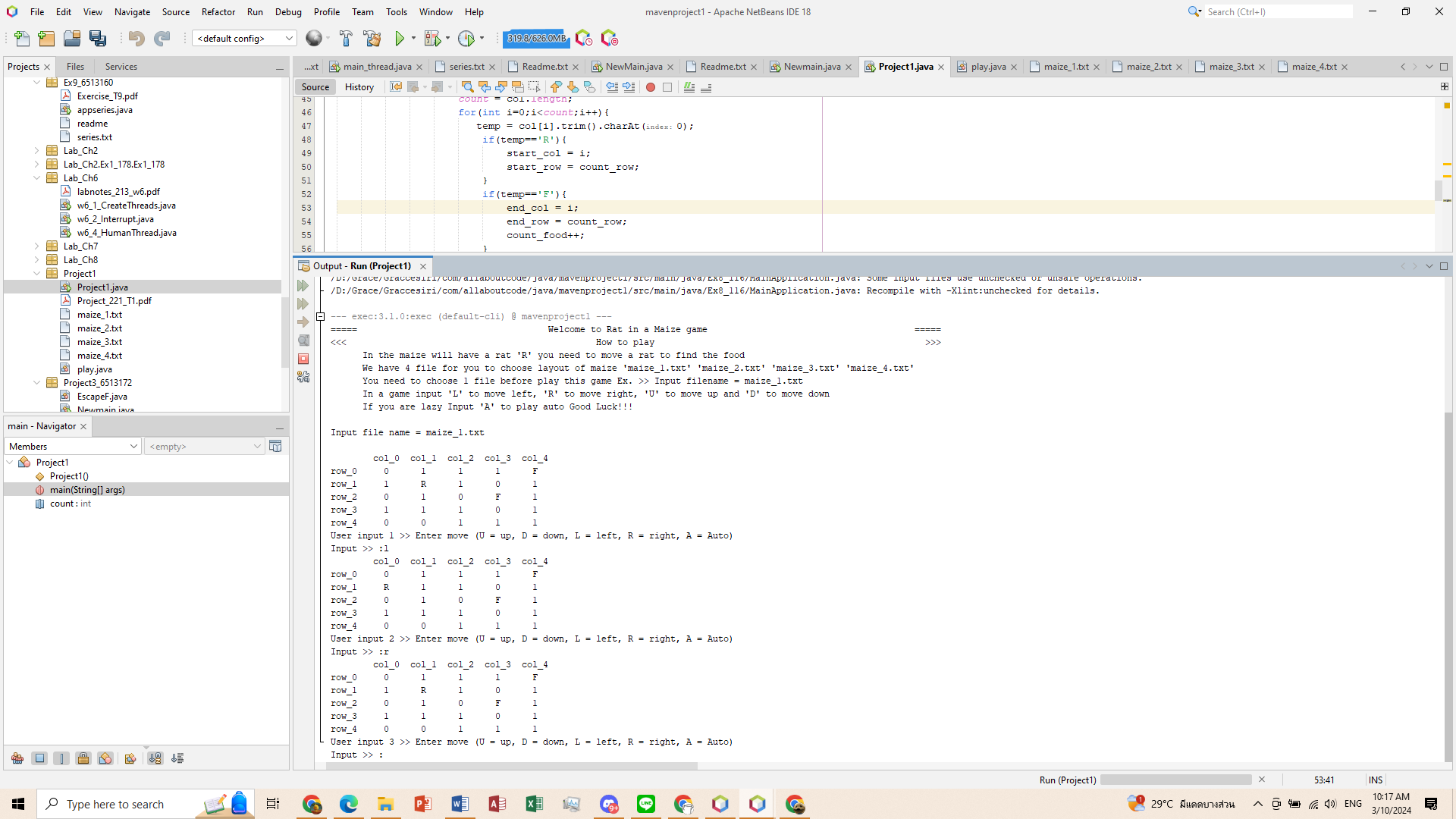
**คู่มือการใช้งาน (Short user manual)**

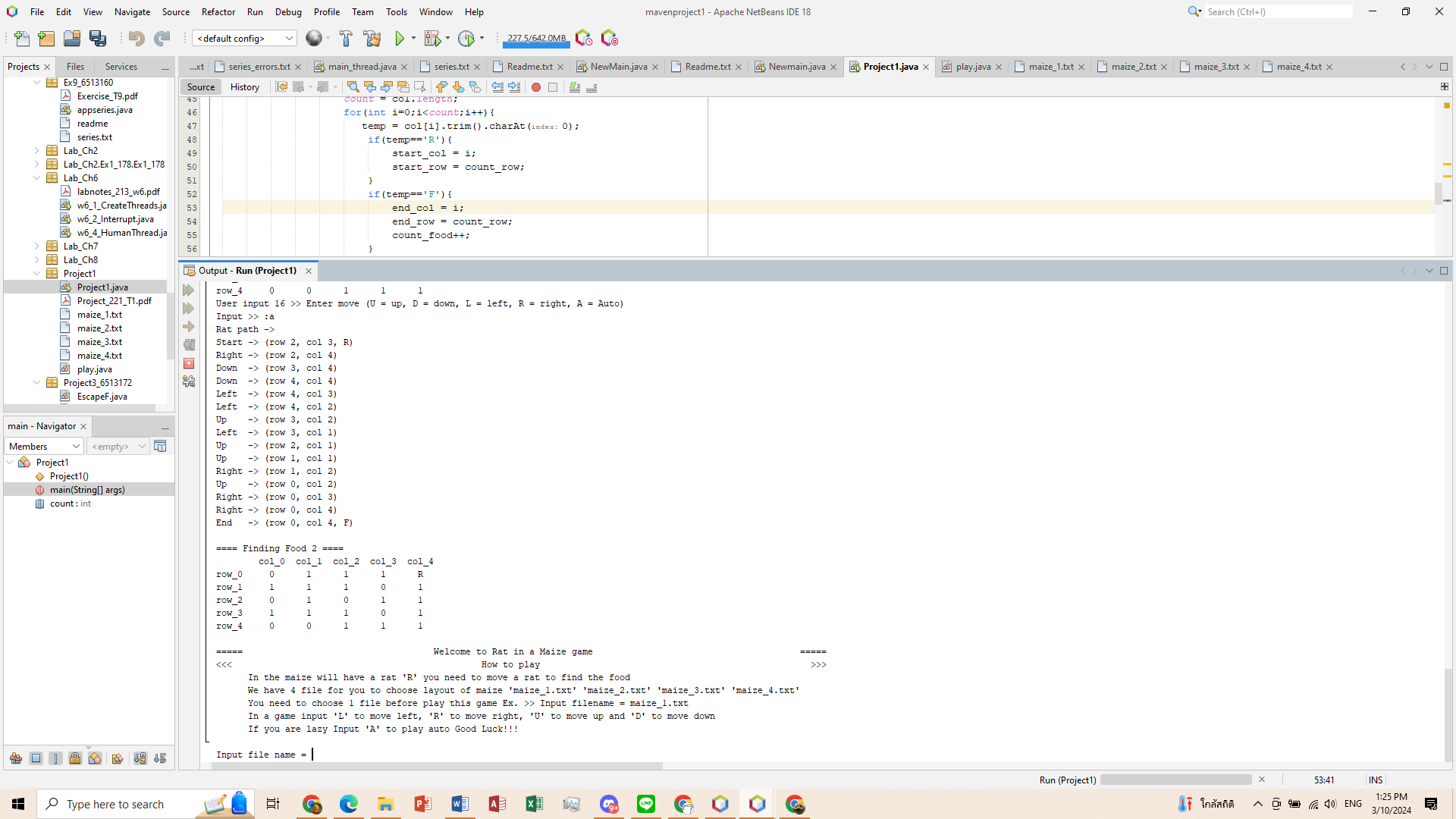
1. เมื่อกด run program จะแสดงผลข้อความดังกล่าว



เพื่อให้ผู้เล่นทราบภาพรวมเกม และกฎกติกาการเล่น ดังนี้

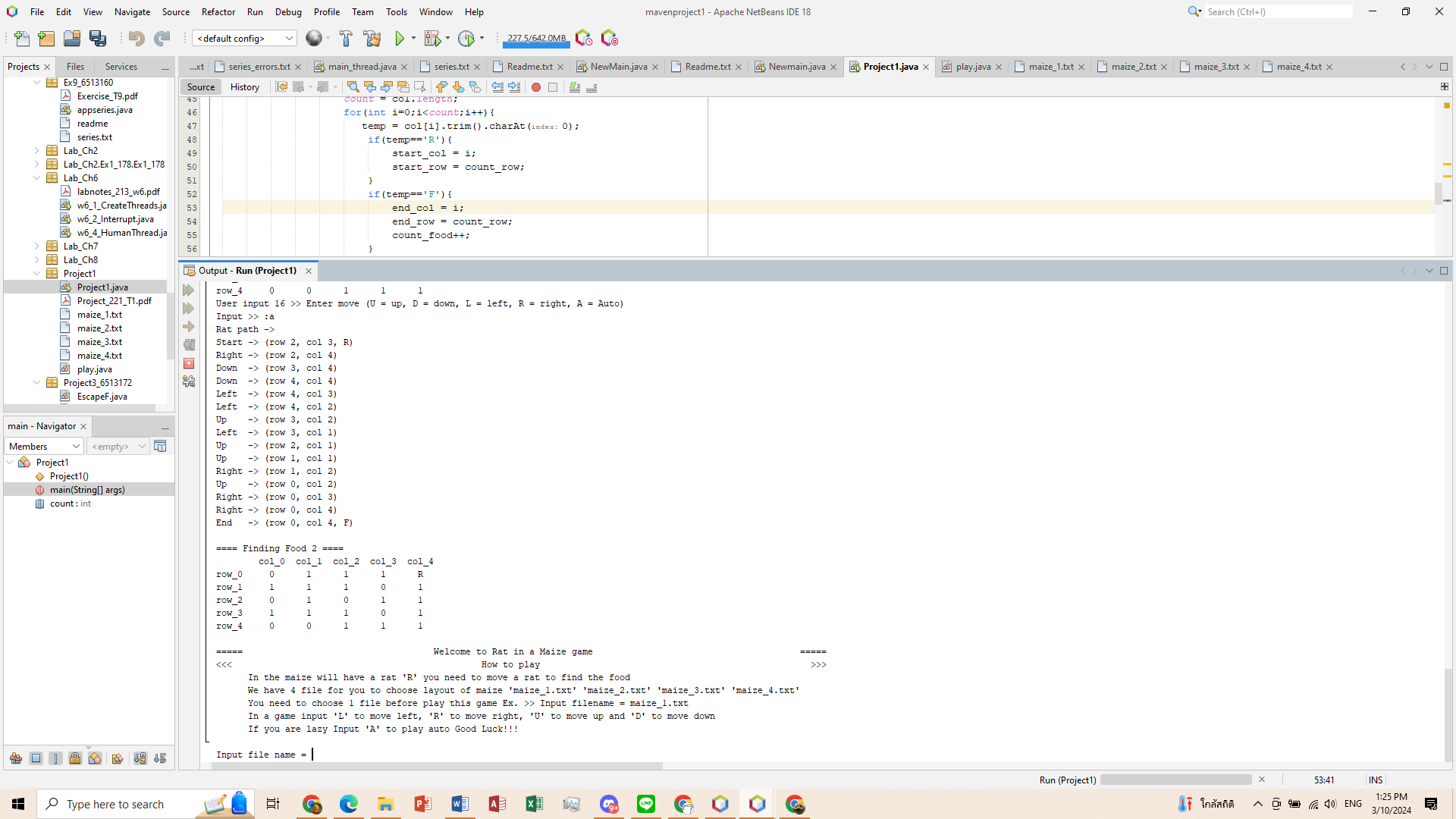
* 1. ในเกมนี้จะมีหนู ซึ่งจะแสดงผลด้วยตัว “R” ซึ่งเราต้องพามันไปหาอาหาร
  2. มีทั้งหมด 4 ไฟล์ที่มีหน้าตาแตกต่างกัน 'maize\_1.txt' 'maize\_2.txt' 'maize\_3.txt' และ 'maize\_4.txt' ให้ผู้ใช้เลือกว่าจะเล่นอันไหน โดยให้เลือกเพียง 1 ไฟล์เท่านั้น โดยการพิมพ์ตามรูปแบบ(format) ดังกล่าว เช่น “Input file name = maize\_1.txt”
  3. เพื่อเดินเกม ผู้เล่นสามารถพิมพ์ ‘L’ เพื่อเคลื่อนที่หนูไปทางด้านซ้าย, พิมพ์ ‘R’ เพื่อเคลื่อนที่ไปยังด้านขวา, พิมพ์ ‘U’ เพื่อเคลื่อนที่ไปยังด้านบน, พิมพ์ ‘D’ เพื่อเคลื่อนที่ไปยังด้านล่าง และสามารถพิมพ์ ‘A’ เพื่อกดเล่นอัตโนมัติได้(auto) โดยในการพิมพ์แต่ละตัวอักษรสามารถพิมพ์ได้ทั้งพิมพ์เล็ก และพิมพ์ใหญ่





(ตัวอย่างการกดเล่นอัตโนมัติ)

1.4 เมื่อหาอาหารครบจนหมด จะแสดงผลให้เลือกไฟล์ maize ใหม่เพื่อเล่นต่อ



**Data Structure ที่นำมาใช้ทั้งหมด**

**1. โครงสร้างข้อมูลใน Class play**



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

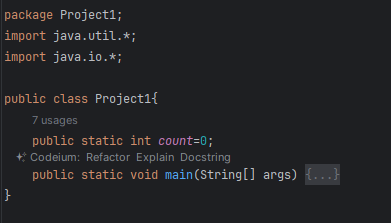
1.2.1 push() : เป็นฟังก์ชั่นที่ใช้ในการเพิ่มข้อมูลไปใน ArrayDeque

1.2.2 pop() : เป็นฟังก์ชั่นที่ใช้ในการคืนค่าและลบข้อมูลใน ArrayDeque

1.2.3 isEmpty() : เป็นฟังก์ชั่นที่ใช้เช็คว่าใน ArrayDeque มีข้อมูลอยู่ไหม

1.2.4 size() : เป็นฟังก์ชั่นที่ใช้ในการคืนค่าขนาดของ ArrayDeque

**2. โครงสร้างข้อมูลใน Class project 1**



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 Foodแล้ว มันจะ return ค่า true มาเรื่อยๆ เพราะว่า if-else ของเรานั้นจะทำงานได้ก็ต่อเมื่อเงื่อนไขเป็น true แล้ว เก็บค่า 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

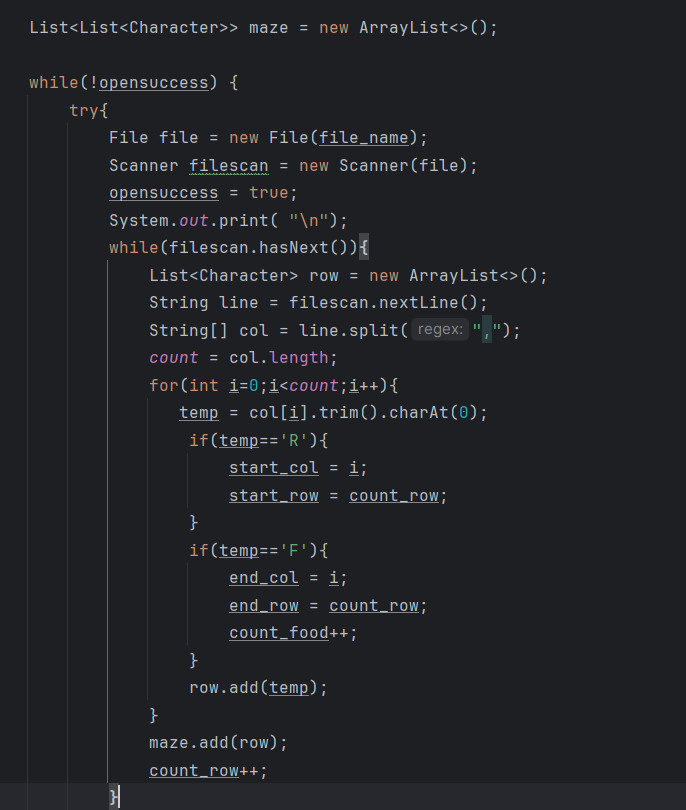


**//ใช้เก็บจำนวน column**

**//ใช้เก็บจำนวน row และ Food**

**//ใช้ระบุตำแหน่งของ rat**

**//ใช้ระบุตำแหน่งของ food**



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



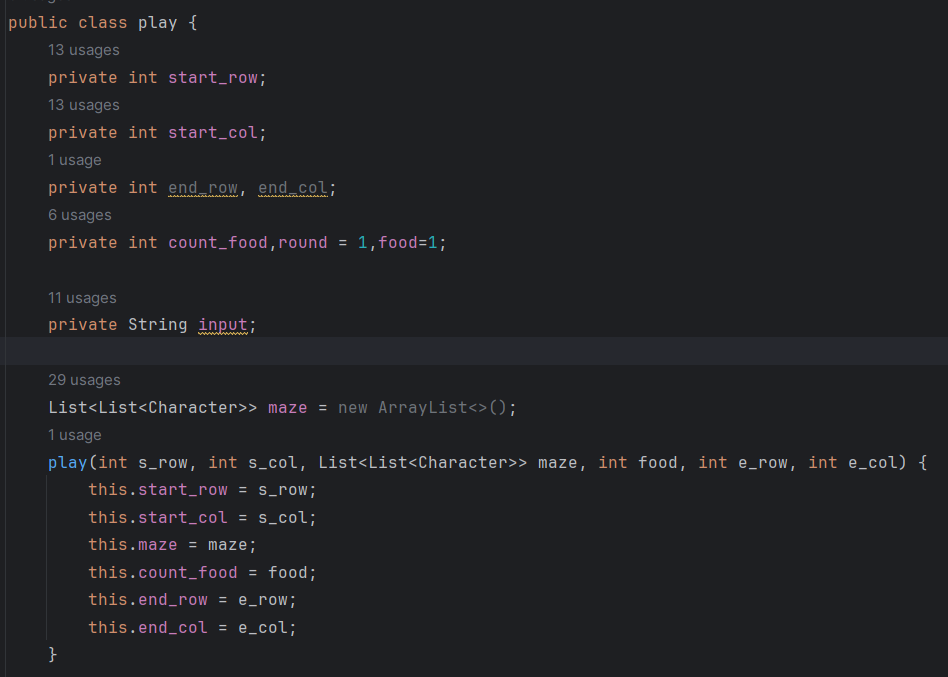
**//function ที่ใช้ในการปริ้ท์ maze ออกมา**

**//function ที่ใช้ในการเล่นเกม**

**ส่งตำแหน่งของ rat และ food ให้กับ class play**

**ใช้ตรวจสอบว่า Input ของ user ถูกหรือไม่ มีไฟล์อยู่ภายในเครื่องหรือไม่**

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

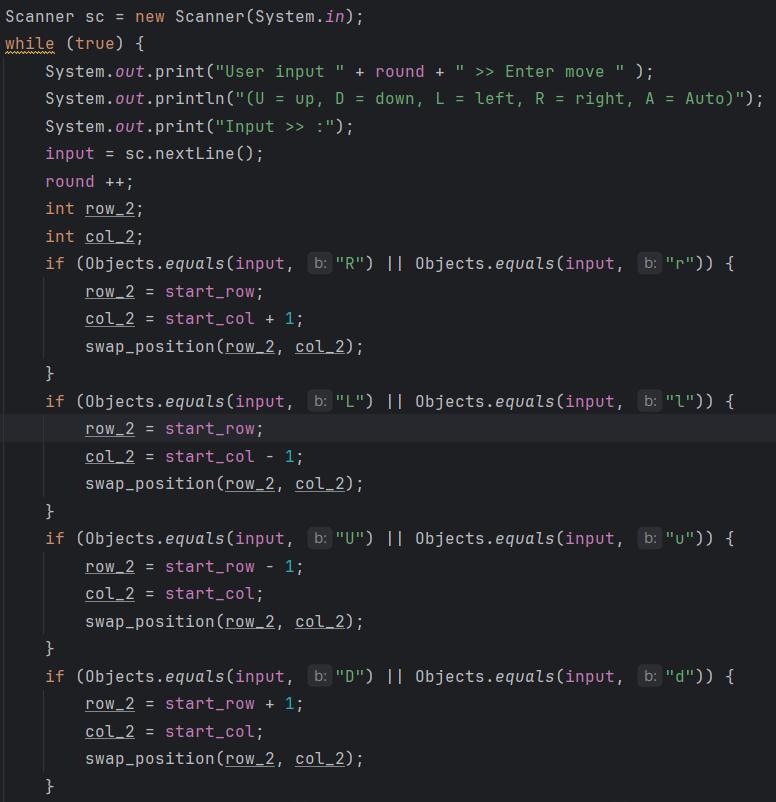


**//ใช้เก็บจำนวนรอบที่ user เล่น และจำนวนอาหารที่เจอแล้ว**

**//ใช้เก็บ input ที่ user ป้อนเข้ามา**

**//Constructor**

**หาก User เล่นแบบ Manual**



เช็ค Input ของ User

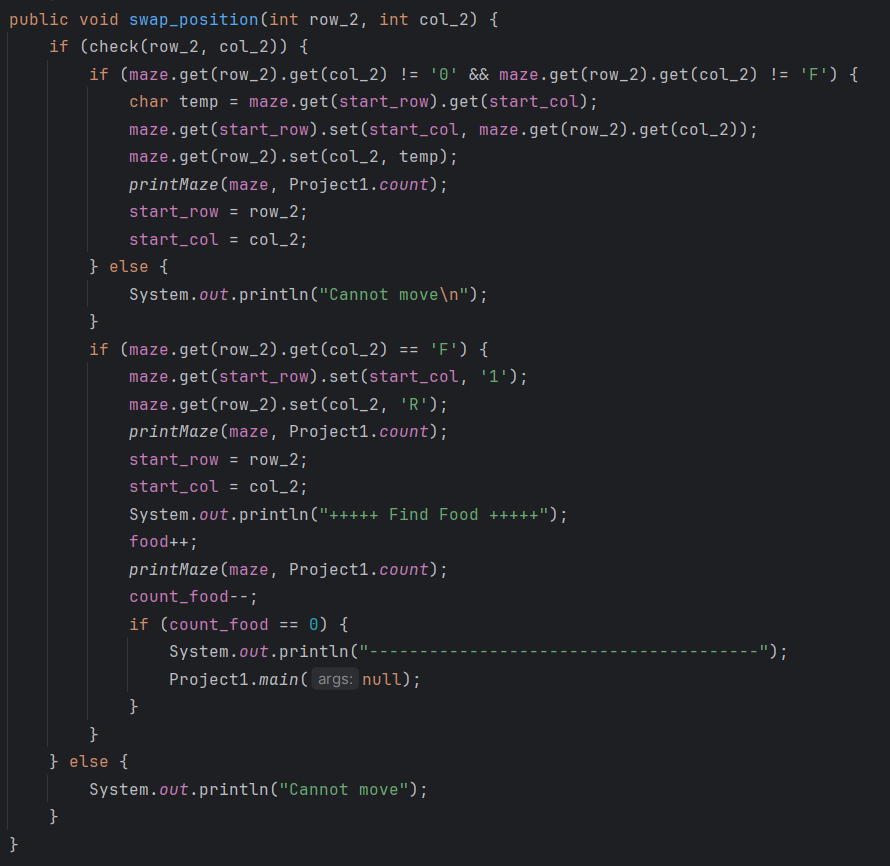
หากเป็น ‘R’ จะขยับตำแหน่งของ rat ไปทางขวา

หากเป็น ‘L’ จะขยับตำแหน่งของ rat ไปทางซ้าย

หากเป็น ‘U’ จะขยับตำแหน่งของ rat ไปข้างบน

หากเป็น ‘D’ จะขยับตำแหน่งของ rat ไปข้างล่าง

**Function ที่ใช้ในการขยับตำแหน่ง**



swap\_position() จะรับค่า row และ col ของตำแหน่งที่จะเคลื่อนที่ไป และเก็บไปลงที่ row\_2 และ col\_2 โดยจะแบ่งเป็น 2 กรณี ดังนี้

1.หากตำแหน่งที่จะเคลื่อนที่ไปไม่ใช่ ‘0’ และ ‘F’ จะสร้าง ตัวแปร 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**

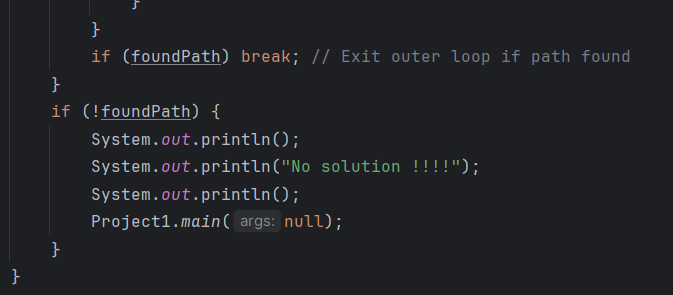
**//สร้าง ArrayDeque เอาไว้เก็บค่า**

**//สร้าง foundpath เอาไว้กำหนดเงื่อนไข**

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

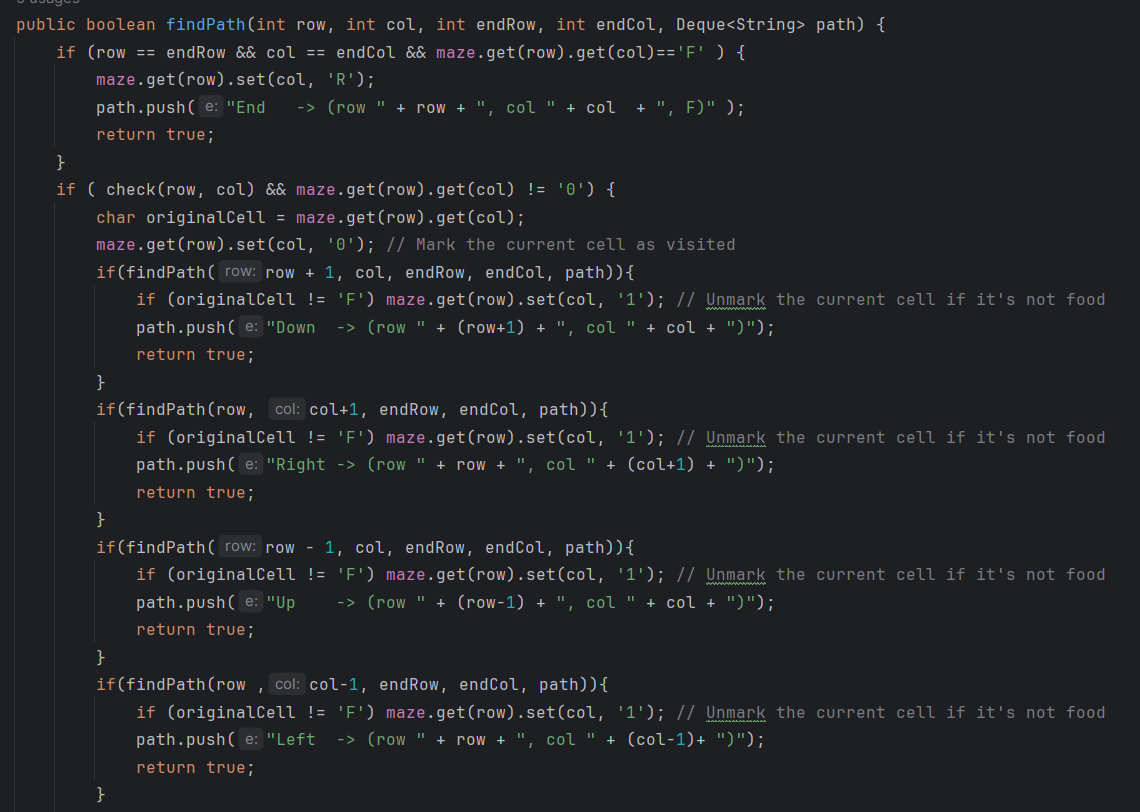
**//ถ้าไม่มี F ใน maizeแล้วจะจบการเล่น**

**//ถ้า foundpath = true จะหยุด for loop**

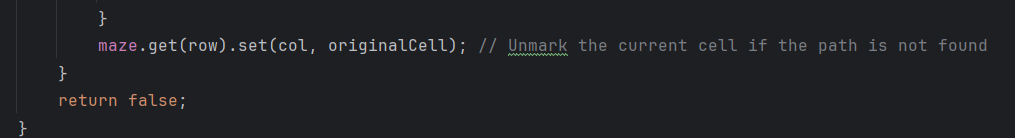


หากไม่มีการเป็น foundpath ให้เป็น true จะจบการเล่นทันที

**//ถ้า foundpath = false จะหยุดการเล่น**

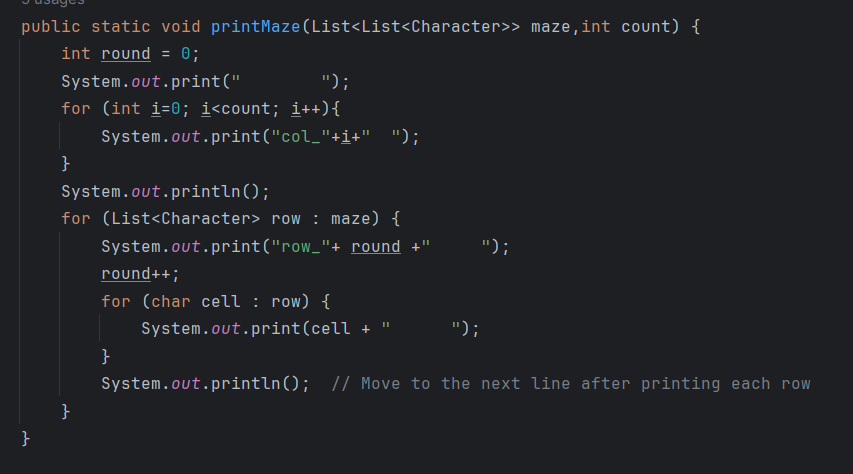


Findpath() จะทำงานแบบ recursive function จะเรียกตัวเองซ้ำจนกว่าจะเจอ F ซึ่ง findpath() นี้จะทำการค้นหาเส้นทางโดยลงข้างล่างก่อนแล้วจึงไปทางขวา ขึ้นบน และไปทางซ้ายตามลำดับ และเมื่อเจอ F แล้วเรียบร้อยจะ push ตำแหน่งลงใน arraydeque และจะ return true ให้กับ findpath() ที่เรียกตนใช้งาน findpath()ก่อนหน้านั้นจะ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

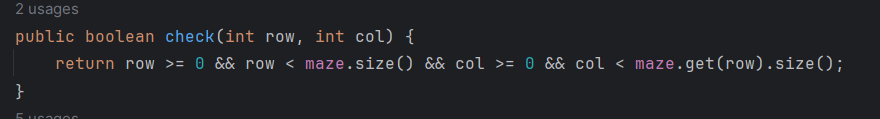


**//ถ้าไม่เข้าเงื่อนไขใด ๆ จะ return false**

**//ถ้าไม่เข้าเงื่อนไขใดๆ จะ return false**

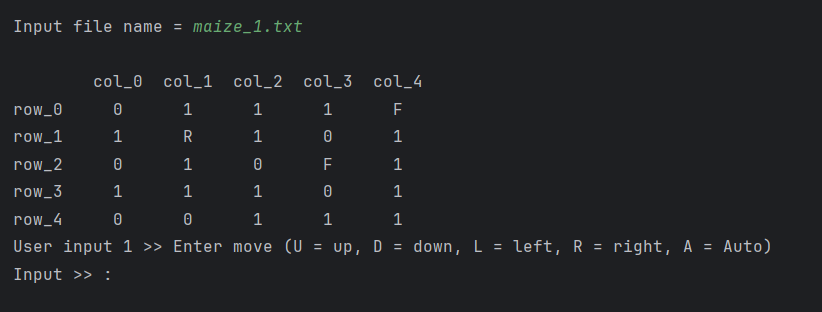


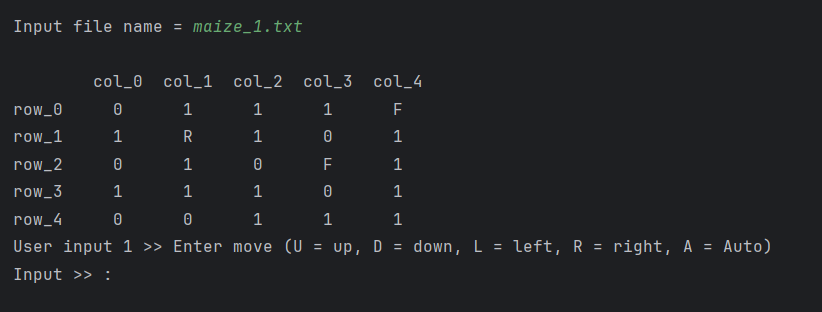
**//เป็น function ที่เอาไว้ print maze ออกมา**



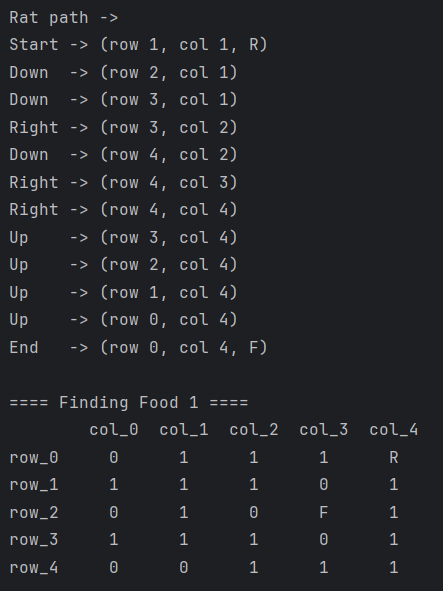
check () เอาไว้เช็คว่าตำแหน่งที่เอา check() ไปใช้เช็คนั้นมีอยู่ใน maze หรือไม่

**ตัวอย่างการแสดงผล**

**ตัวอย่างไฟล์ maize\_1.txt**

**เมื่อ Input a ให้โปรแกรมรัน auto**

Forwarding Step เราจะใช้ for loop ในการเช็คแต่ละแถวว่ามี F อยู่หรือไม่ และเก็บตำแหน่งของ F ไว้

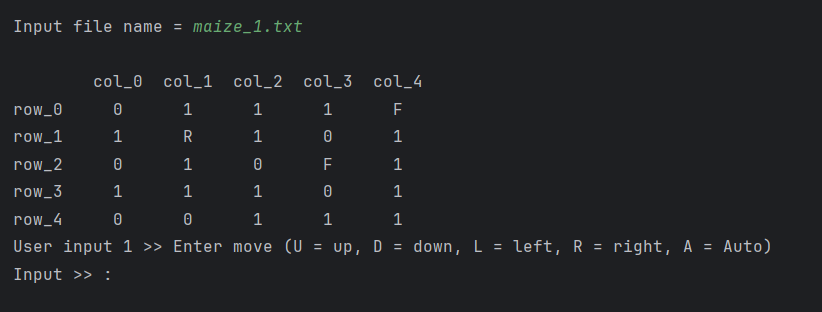


Backtracking Step เราจะใช้ function findpath ในการหาเส้นทางการเดินของ Rat โดยเราจะใช้ push() เส้นทางการเดินทางเข้าไปใน Stack และ ใช้ pop() ในการ print เส้นทางการเดินทางออกมา

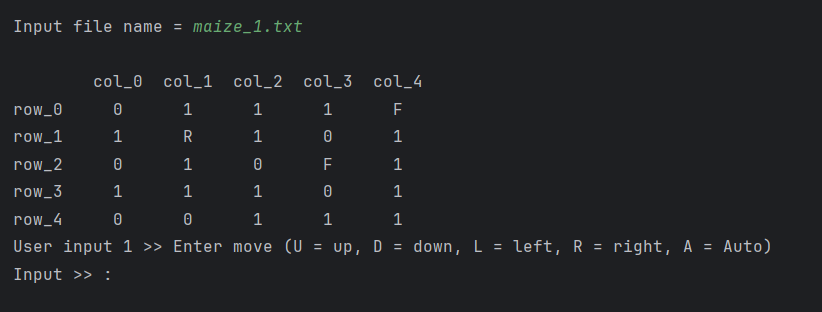
ซึ่งจะอธิบายต่อหน้าถัดไป

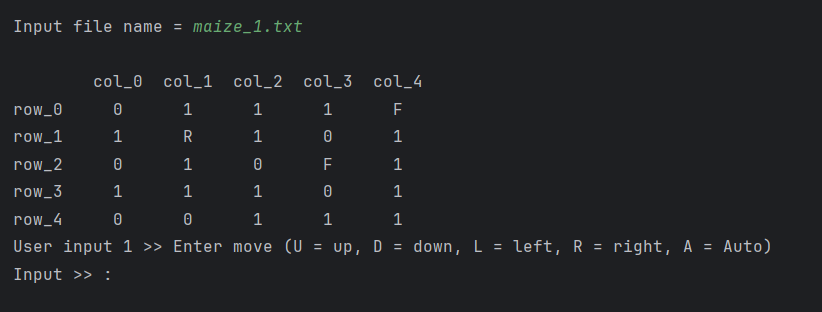
โดย function findpath จะทำงาน Recursive algorithm โดยจะเรียกทำตัวเองซ้ำ ๆ และจะรวมโดยการหา path จะทำจากเดินลงล่างก่อนหากไปต่อไม่ได้จะเดินทางขวาและหากไปต่อไม่ได้อีกจะเดินขึ้นและทางขวาตามลำดับ

**ตัวอย่าง**

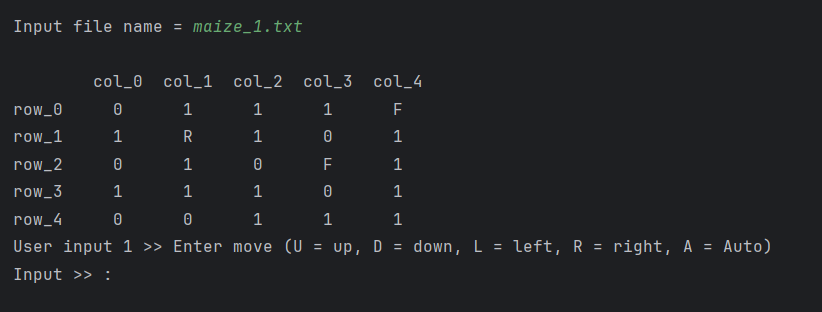


ใน function findpath เราจะเริ่มเดินลงก่อนหากสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

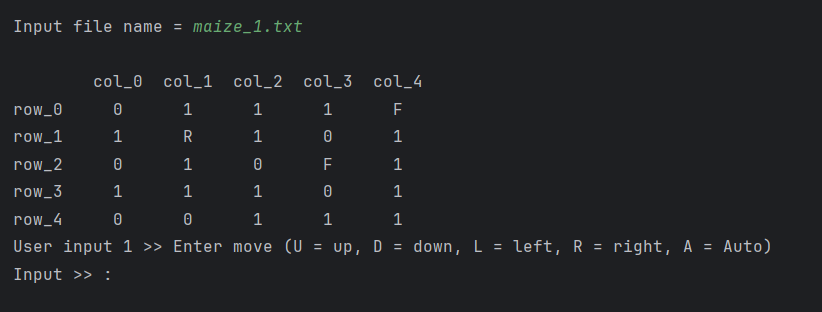




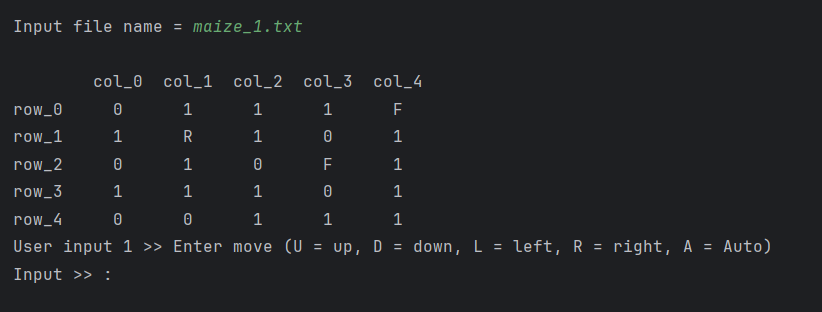
เมื่อไม่สามารถเดินลงได้ เราจะเริ่มเดินทางขวาต่อ ถ้าสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

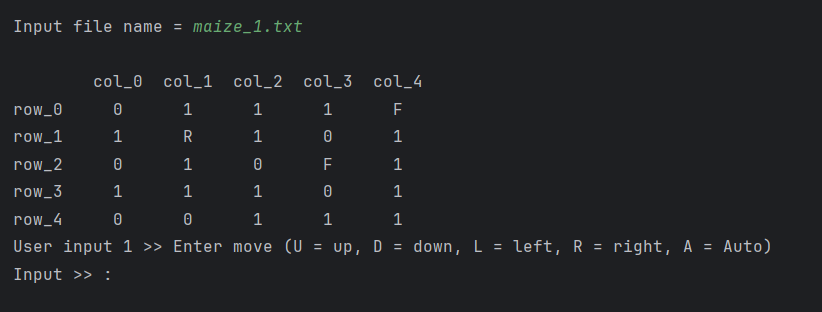


สามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

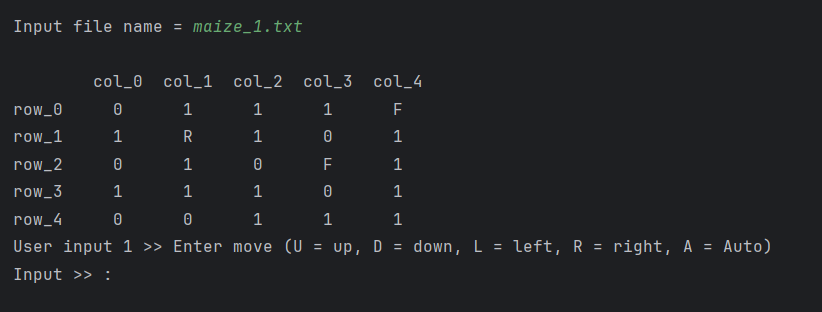


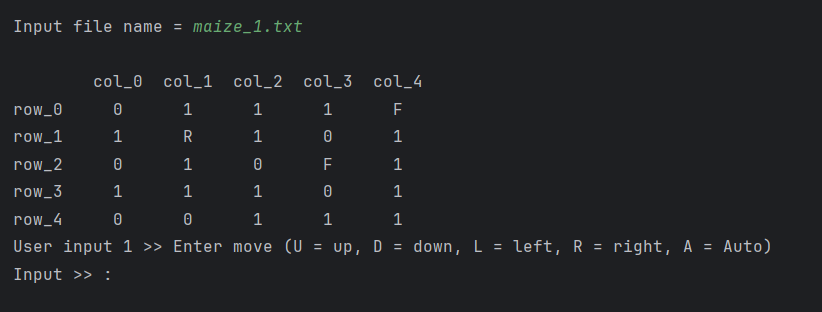
เมื่อไม่สามารถเดินลงได้ เราจะเริ่มเดินทางขวาต่อ ถ้าสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

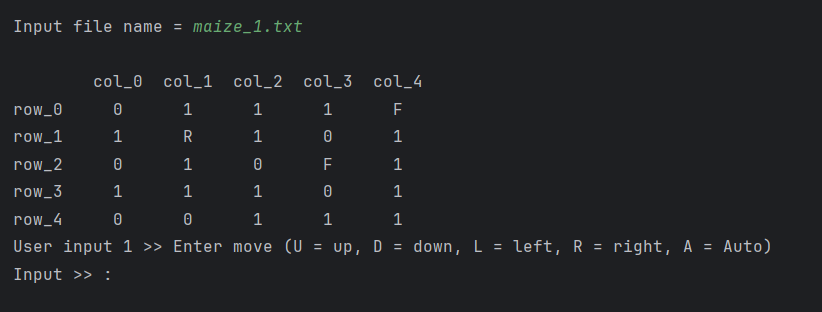




เมื่อไม่สามารถเดินลงย้อนทางเดิมและเดินขวาได้ เราจะเริ่มเดินขึ้นบนต่อ ถ้าสามารถขึ้นลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ







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** |

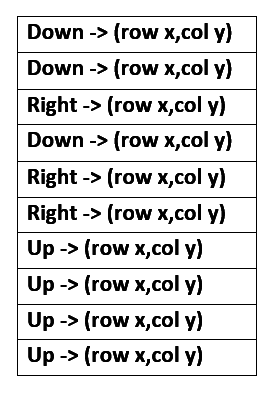
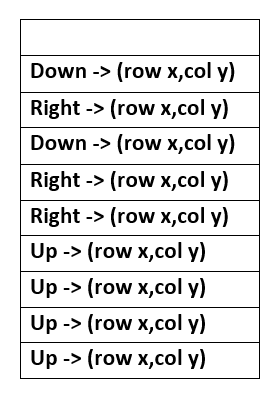
**แต่สิ่งที่เอาไปเก็บใน Stack จริงๆคือ**

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

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

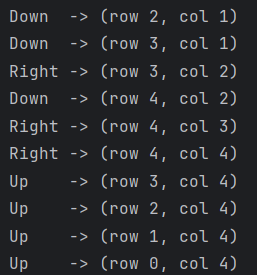
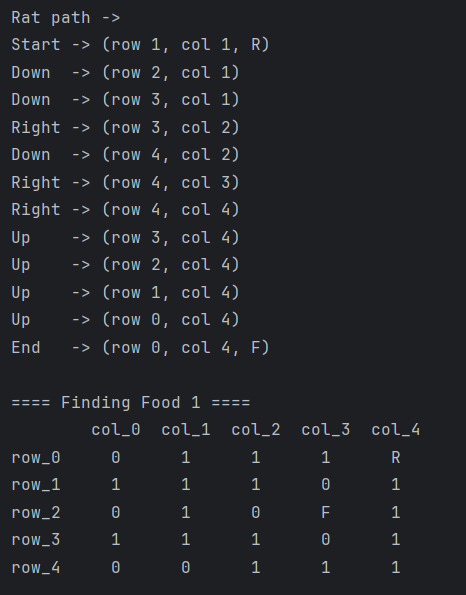
**U = Up -> (row x,col y)**

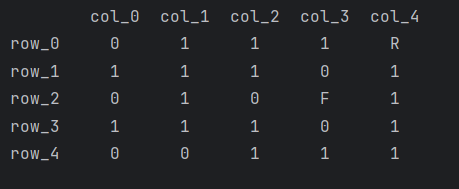
**L = Left -> (row x,col y)**

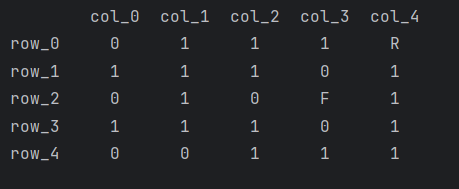


**POP**

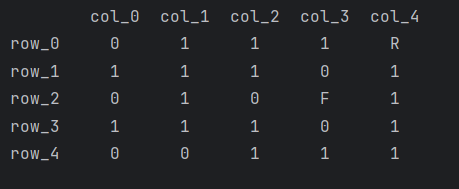
**จึงได้ output ดังนี้**



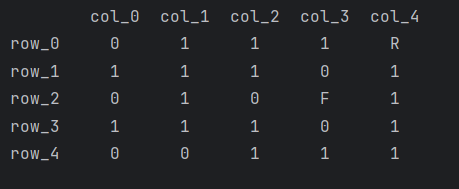
**หา F อีกตัว**

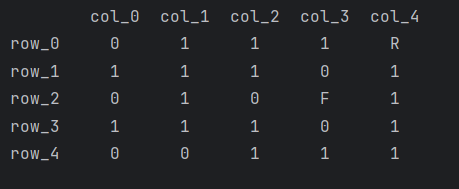


Forwarding Step เราจะใช้ for loop ในการเช็คแต่ละแถวว่ามี F อยู่หรือไม่ และเก็บตำแหน่งของ F ไว้

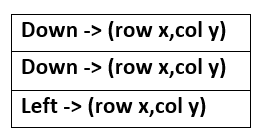
**เริ่มใช้ function findpath**

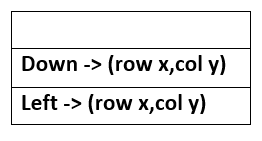
ใน function findpath เราจะเริ่มเดินลงก่อนหากสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

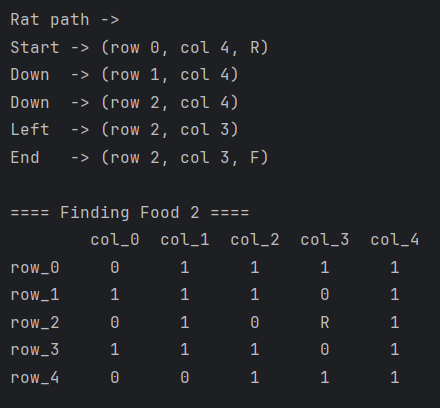


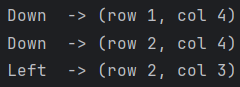


|  |  |  |  |
| --- | --- | --- | --- |
| **node** | **function** | **Stack** | **push** |
| **1** | **Findpath(1)** | **D** |  |
| **2** | **Findpath(2)** | **D D** |  |
| **3** | **Findpath(3)** | **D D L** |  |
| **2** | **Return true**  **To findpath(2)** | **D D** | **L** |
| **1** | **Return true**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

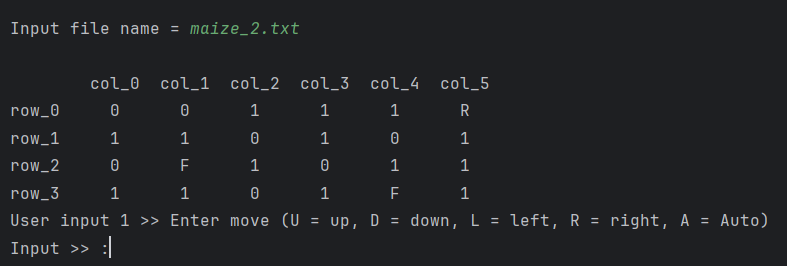




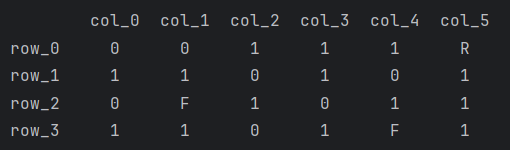
**จึงได้ output ดังนี้**



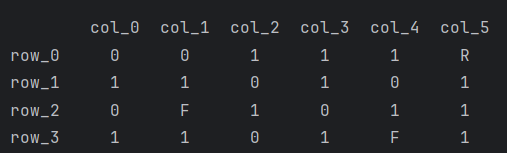
**ตัวอย่างไฟล์ maize\_2.txt**

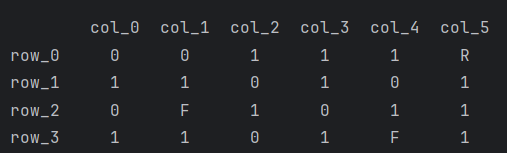


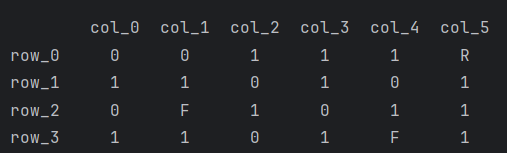
เมื่อ Input a ให้โปรแกรมรัน auto

จะเหมือนกับ maize\_1.txt

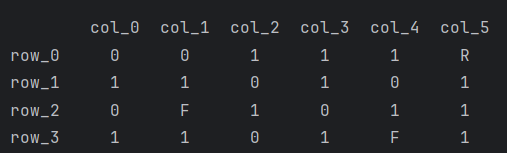
Forwarding Step เราจะใช้ for loop ในการเช็คแต่ละแถวว่ามี F อยู่หรือไม่ และเก็บตำแหน่งของ F ไว้

**เริ่มใช้ function findpath**



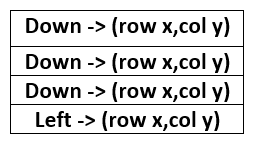
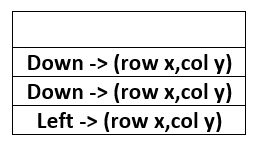


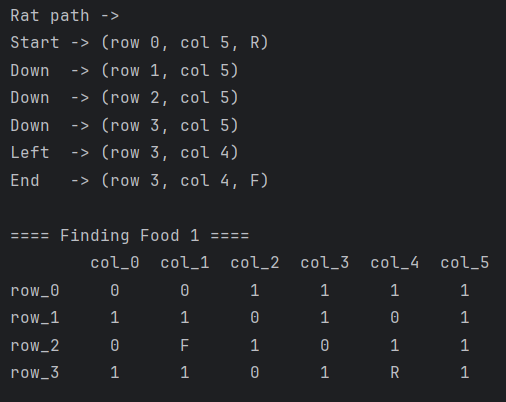
ใน function findpath เราจะเริ่มเดินลงก่อนหากสามารถเดินลงได้ เราจะเรียกใช้ function อีกที เพื่อหาเส้นทางการเดินต่อ

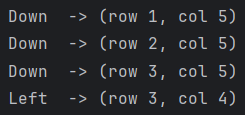


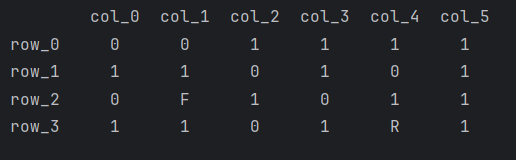
เมื่อใม่สามารถเดินลง เดินขวา และเดินขึ้นได้ ก็จะเดินซ้ายแทน

|  |  |  |  |
| --- | --- | --- | --- |
| **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** |





**จึงได้ output ดังนี้**

**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**  **To Findpath(4)** | **R U** |  |
| **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].