EGCO 221 – Group Project 1 (Rat in a Maize)

1. Given a maize of size **rows * cols**, each cell in the maize contains one of the following:
   - 1 = ground ; Rat can move to this cell
   - 0 = wall ; Rat cannot move to this cell
   - R = ground with Rat ; After moving up/down/left/right to another ground, this cell will be set to 1
   - F = ground with Food ; When moving to this cell to take Food, this cell will be set to R

   Our purpose is to find at least 1 path for the Rat to get all Foods.

2. **Programing**: the program must be written in Java.
   - 2.1 Read maize layout from input file.
     - The maize must have at least 3 rows & 3 cols; rows and cols don't need to be equal.
     - You can assume that the file doesn't contain input error i.e. there are always 1 Rat, >= 1 Foods, and either 0 or 1 in other cells.
     - But your program must still be able to handle missing file.
   - 2.2 Keep asking user for move direction up/down/left/right. If the Rat can move in that direction, move it. At any time, the user must be able to switch to auto mode.
   - 2.3 In auto mode, find a solution to get all Foods (if there is one) starting from the switching state in (2.2).
     - You don't need to find the best solution (e.g. shortest path) or find all possible solutions.
     - Any 1 solution is enough since this project focuses mainly on backtracking
   - 2.4 The program should be able to loop for a new game with different maize files and handle invalid user input in (2.2).
   - 2.5 Your source files (.java) must be in folder Project1_XXX where XXX = ID of the group representative, assuming that this folder is under Maven's "src/main/java" structure. The first lines of all source files must be comments containing English names & IDs of all members.

3. This puzzle is generally known as "Rat in a Maize" puzzle and the common setup is having only 1 Food in the maize. There are many approaches to solving it, but you're required to use backtracking with either explicit or implicit stack in this project. Recursive methods are counted as using (runtime) stack implicitly.
   - You can search and use any algorithm or even pieces of code, provided that the sources are properly acknowledged.
   - These sources must be from your own research e.g. paper, textbook, Internet, etc.
   - Using classmates as references is considered cheating.

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ solutions ---
java.io.FileNotFoundException: src\main\java\Project1\maize.txt (The system c
New 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)
u
       col_0  col_1  col_2  col_3  col_4
row_0      0      R      1      1      F
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

User input 2 >> Enter move (U = up, D = down, L = left, R = right A = auto)
u
Cannot move

User input 3 >> Enter move (U = up, D = down, L = left, R = right A = auto)
l
Cannot move

User input 4 >> Enter move (U = up, D = down, L = left, R = right A = auto)
r
       col_0  col_1  col_2  col_3  col_4
row_0      0      1      R      1      F
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

User input 5 >> Enter move (U = up, D = down, L = left, R = right A = auto)
r
       col_0  col_1  col_2  col_3  col_4
row_0      0      1      1      R      F
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

User input 6 >> Enter move (U = up, D = down, L =
r

+++++ Find Food +++++
       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
```

Demo 1 (maize_1.txt)
Fully manual

Design a better user interface by yourself. Good user interface doesn't mean fancy output, but rather how your program is easy to understand & to use even without user manual. For example,

- Can user continue playing without having to restart the program?
- Are the instructions clear enough?
- Do you warn about valid user input and/or handle invalid user input?
- Is the output properly formatted and understandable?

```
User input 7 >> Enter move (U = up, D = down, L = left, R = right A = auto)
d
       col_0  col_1  col_2  col_3  col_4
row_0      0      1      1      1      1
row_1      1      1      1      0      R
row_2      0      1      0      F      1
row_3      1      1      1      0      1
row_4      0      0      1      1      1

User input 8 >> Enter move (U = up, D = down, L = left, R = right A = auto)
d
       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      F      R
row_3      1      1      1      0      1
row_4      0      0      1      1      1

User input 9 >> Enter move (U = up, D = down, L = left, R = right A = auto)
r
Cannot move

User input 10 >> Enter move (U = up, D = down, L = left, R = right A = auto)
l

+++++ Find Food +++++
       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
------------------------------------------------------------------------
```

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ solutions ---
java.io.FileNotFoundException: src\main\java\Project1\maize.txt (The system c
New 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)
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 2 >> Enter move (U = up, D = down, L = left, R = right A = auto)
a

===== Finding Food 1 =====
       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

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

===== 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      F      1
row_3      1      1      1      0      1
row_4      0      0      1      1      1

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

Demo 2 (maize_1.txt)
Manual then Auto

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ solutions ---
java.io.FileNotFoundException: src\main\java\Project1\maize.txt (The system c
New 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)
a

===== Finding Food 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

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

===== Finding Food 2 =====
       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

No solution !!
```

Demo 3 (maize_2.txt)
Auto, no solution

4. **Report in THAI**: describe at least the following
   4.1 Short user manual
   4.2 **All data structures** in your project e.g. arrays, ArrayList, ArrayDeque, LinkedList, runtime stack, etc.
      - Reason for using each of them
      - Values kept in each data structure. If they are objects, also explain class design.
   4.3 **Algorithm** to find at least 1 solution to get all Foods. Source code listing without any explanation isn't counted as a proper report. Explain at least the following:
      - Forwarding steps : from Rat position, what is the order of your search directions (up-down-left-right, left-right-up-down, etc.)? And what does your algorithm do in each step (calculating xxx, keeping xxx in stack, etc.)?
      - Backtracking steps : what are conditions that trigger backtracking? And what does your algorithm do in each backtracking step?
      - Finding next Food : what does your algorithm do before starting the next search (resetting xxx, clear xxx, etc. after finding previous Food)?
      - How does the algorithm conclude that there is no solution?
      - Even if your algorithm is implemented as recursive method, you still need to explain the above points in the context of recursive calls.
   4.4 Demonstrate how your algorithm works in until a solution is found or not found, using maize layouts in maize_1.txt and maize_2.txt in fully Auto mode

      For each layout, show the maize & stack content after every forwarding step by the program and after every backtracking step. Don't conjure up the moves. Your demo must match what your program actually does. Note that there can be >= 1 solutions to the puzzle. Yours may or may not be the same as the given examples.

      *** If your program can solve only 1-Food puzzle, use maize_3.txt and maize_4.txt for demos instead.
   4.5 Any limitation of your program. Convenient limitation isn't counted as proper limitation. For example, don't give a limitation that your program will crash if the user enters "x" just because you are too lazy to check for valid input
   4.6 References or declaration of originality
      - If you design the data structures/algorithms and write the whole program by yourself, explicitly declare that everything is done by yourself. But if it is found later that this is not true, it will be counted as cheating
      - Otherwise, valid references/URLs must be given
      - A report without any reference or declaration of originality will get point deduction

*** The project can be done in a group of <= 4 students. But each group must do it by themselves. **Everyone involved in cheating will get ZERO point**

Grading

Programming (10 points)

2 points Minimum requirements + correct solution (or no solution) to find 1 Food

- If you can't solve n-Foods puzzle, make your program work on maize_3.txt and maize_4.txt

2 points Correct solution (or no solution) to find all Foods

2 points User interface + exception handling

- This part must be your own effort. Even if 2 groups take algorithms or pieces of code from the same source, it would be impossible to have the same/similar coding for the user interface. Therefore, same (or too similar) user interface coding is considered cheating

4 points Programming (2 points) + OOP with Java Collection APIs (2 points)

- Although this is not a programming course, your program should still be well structured. Excessive use of static methods as C-style functions is strongly discouraged.
- Even if you use pieces of code from the Internet. Convert it to proper OOP with Java collection APIs.

Report (10 points)

3 points Data structures, classes

5 points Algorithm, demos for 2 initial states

2 points Manual, limitation, reference/declaration of originality, others (writing, format, etc.)

Submission

1. A report in only 1 PDF file. The front page must contain names & IDs of everyone in your group
2. Source files (*.java)
3. File readme.txt containing names & IDs of everyone in your group
4. Put all files in folder Project1_XXX (see 2.5) and zip the folder
   - The group representative submits the whole project to Google Classroom
   - The other group members submit only readme.txt to Google Classroom