JIMP 2 Report on the part of project completed using the Java language

Team 7

Table of Contents

Introduction

1.Program Description

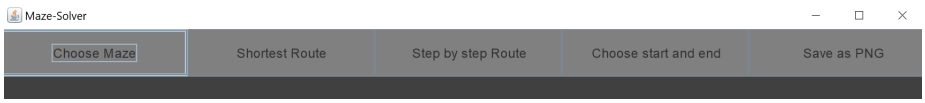
The goal of the program created for the project, written in Java using the Swing library, which was used for the graphical interface, was to find a path through a maze with a maximum size of 1024x1024 cells, counted along paths that can be navigated. The result of the program's operation should be a graphical representation of the sought path.

2.Main Difficulty of the Task

The main difficulty associated with creating the program was creating a properly functioning graphical interface that allows the shortest path in the maze to be seen, as well as the step-by-step path, and an algorithm enabling the saving of the program's result as an image.

3.User Program Handling

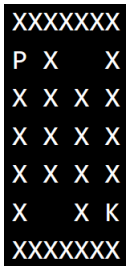
A .jar file is placed in the repository. The program user only needs to download, and then run it.



In the application, the user can see 5 buttons providing access to options.
The first one, labeled "Choose Maze", allows the user to launch the file explorer and select the file where the maze is saved. Such a file can be generated using a website provided as a tool for the project, or one can use sample mazes contained in the repository. This file can be in .txt format. In such a file, the following notations have been adopted:

- P – the start of the maze (entry)
- K – the end of the maze (exit)
- X – wall
- Space – a place where it is possible to move.

A sample .txt file looks as follows:



The file can also be in .bin format. Such a file is described as follows:

Sekcja 1 i 2 są obowiązkowe i zawsze występują, sekcja 3 oraz 4 są opcjonalne. Występują jeśli wartość pola *Solution Offset* z nagłówka pliku jest różna od 0.

Nagłówek pliku:

Nazwa pola	**Wielość w bitach**	**Opis**
File Id	32	Identyfikator pliku: 0x52524243
Escape	8	Znak ESC: 0x1B
Columns	16	Liczba kolumn labiryntu (numerowane od 1)
Lines	16	Liczba wierszy labiryntu (numerowane od 1)
Entry X	16	Współrzędne X wejścia do labiryntu (numerowane od 1)
Entry Y	16	Współrzędne Y wejścia do labiryntu (numerowane od 1)
Exit X	16	Współrzędne X wyjścia z labiryntu (numerowane od 1)
Exit Y	16	Współrzędne Y wyjścia z labiryntu (numerowane od 1)
Reserved	96	Zarezerwowane do przyszłego wykorzystania
Counter	32	Liczba słów kodowych
Solution Offset	32	Offset w pliku do sekcji (3) zawierającej rozwiązanie
Separator	8	słowo definiujące początek słowa kodowego – mniejsze od 0xF0
Wall	8	słowo definiujące ścianę labiryntu
Path	8	słowo definiujące pole po którym można się poruszać
Podsumowanie	**420**	**Sumarycznie nagłówek ma rozmiar 40 bajtów**

Słowa kodowe:

Nazwa pola	**Wielość w bitach**	**Opis**
Separator	8	Znacznik początku słowa kodowego
Value	8	Wartość słowa kodowego (Wall / Path)
Count	8	Liczba wystąpień (0 – oznacza jedno wystąpienie)

Sekcja nagłówkowa rozwiązania

Nazwa pola	**Wielość w bitach**	**Opis**
Direction	32	Identyfikator sekcji rozwiązania: 0x52524243
Steps	8	Liczba kroków do przejścia (0 – oznacza jeden krok)

Krok rozwiązania:

Nazwa pola	**Wielość w bitach**	**Opis**
Direction	8	Kierunek w którym należy się poruszać (N, E, S, W)
Counter	8	Liczba pól do przejścia (0 – oznacza jedno pole)

Pola liczone są bez uwzględnienia pola startowego.

The next button, labeled "Shortest Route", invokes the BFS (breadth-first search) algorithm to create the shortest path from the maze's starting point to its endpoint.

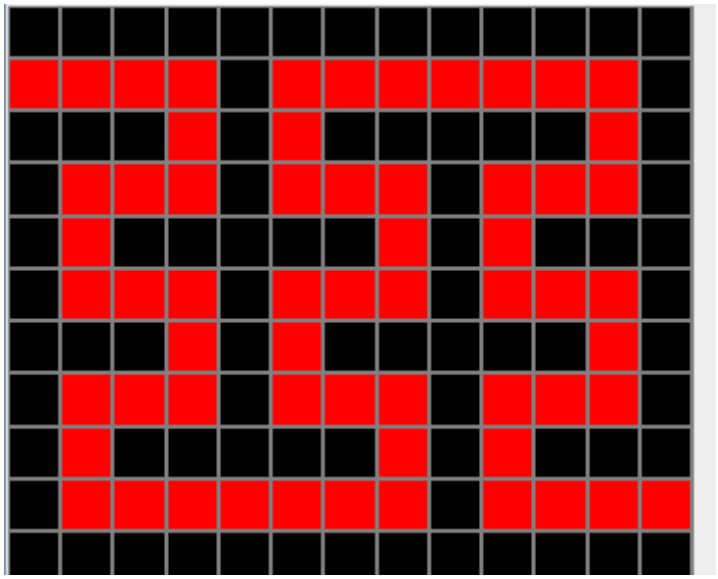
The third button, labeled "Step by step Route", invokes the right-hand rule algorithm to create a path that can be observed in real-time. Due to the algorithm's nature, it only works correctly when the loaded maze does not have walls disconnected from the rest.

The fourth button, "Choose start and end", allows for selecting new starting and ending points. The user does this by pressing the button and then clicking the selected points in the maze; the first one will become the new starting point, and the second one will become the new endpoint.

The last button, "Save as PNG", allows for saving the maze state as a PNG image.

4.Program Output

The result of the program's operation is a graphical representation of the path allowing escape from the (given) maze. An example of saving the solved maze as an image is shown below:



Solution Method

1.Selected algorithms for the solution and their descriptions

The algorithm we decided to use to find the shortest path through the maze is the BFS algorithm, which was implemented in the Parallel BFS version to increase its time efficiency. After loading the maze using the "Choose Maze" button, the loaded maze will be displayed. Then, by clicking the "Find Shortest Route" button, the BFS algorithm will be called. When it encounters a field marking the exit from the maze, it calls a method used to determine the path through the maze from the end. It starts in a field adjacent to the maze exit and then moves to the adjacent field with a value 1 less than its own until it encounters the value 1, marking the maze entrance. Then the maze drawing method is called again. If the user wishes, they can use the "Save as PNG" button to save the maze with the marked route. The same steps are performed on a binary file after converting it to a text file.

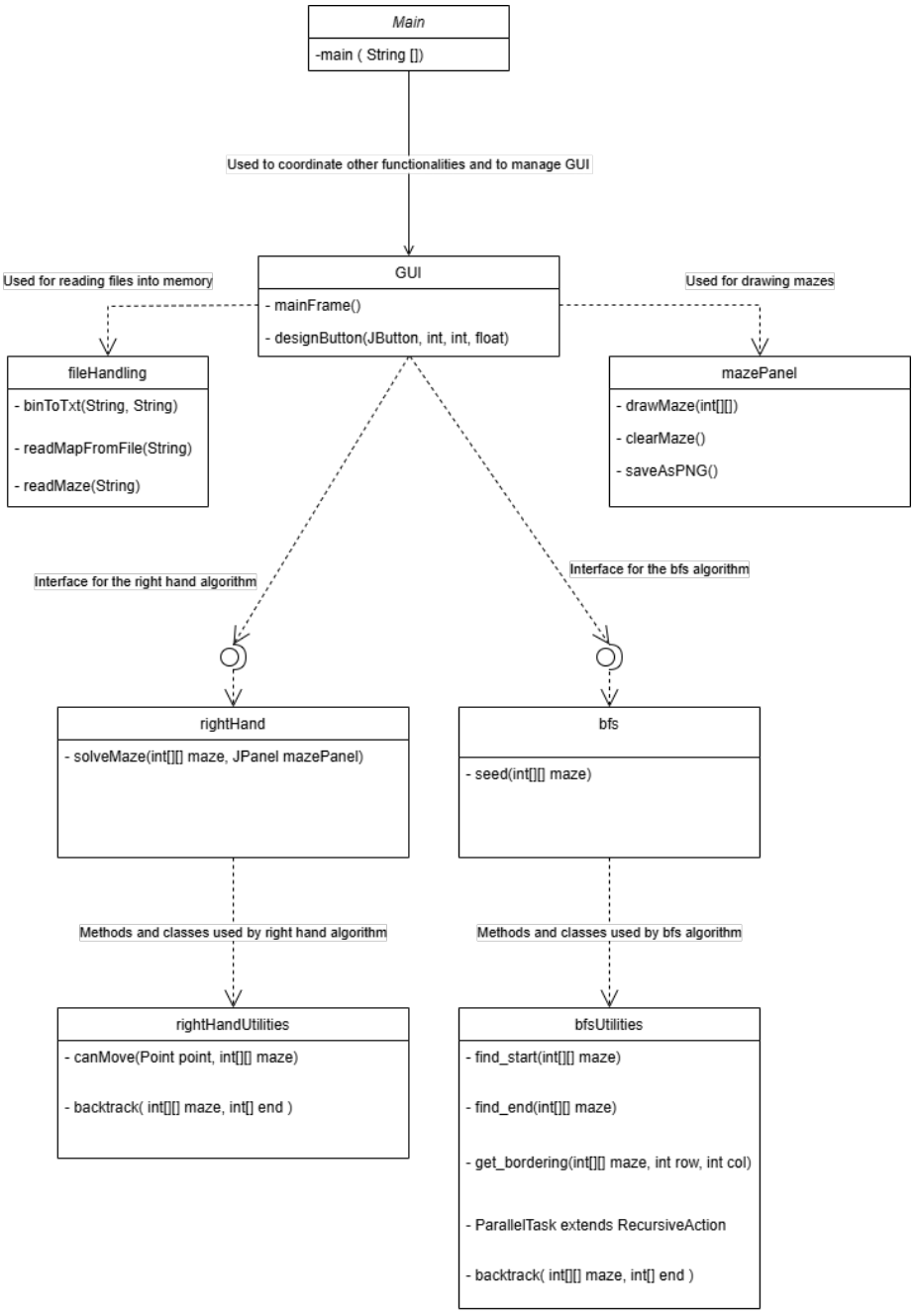
The algorithm allowing to watch the route creation is the right-hand rule algorithm. After invoking it on the loaded maze using the "Step by step Route" button, it will start navigating through the maze, always keeping the wall on the right side of the navigator. Therefore, it will not create a working path if the maze has walls disconnected from each other. In such a case, it will fall into a cycle, circling along the detached wall where it started moving.

2.Code Distribution

The code is divided into classes:

A) bfs.java serving as an interface to bfsUtilities.java used for handling the BFS algorithm
B) rightHand.java serving as an interface to rightHandUtilities.java used for handling the right-hand rule algorithm
C) fileHandling.java responsible for handling maze files
D) gui.java responsible for the graphical user interface
E) mazePanel.java responsible for handling the display and operation of the maze from the graphical interface side

The relationships between them can be seen in the class diagram on the next page.



Design Patterns Used

Singleton: This pattern is used in the bfs class in the context of the ForkJoinPool class, which is created only once and used by all threads in the

application.

Strategy: This pattern is used in the rightHand class, where the right-hand rule algorithm is implemented as a separate maze-solving strategy.

Facade: The gui class functions as a facade, e.g., the mainFrame() method, managing user interface interactions and invoking various functions in other classes.

Abstract Factory: The class handling maze file reading executes different methods depending on the data it receives, yet creating the appropriate object based on the input file type.

Observer: In the rightHand class, the SwingWorker mechanism is used to update the state to be passed to the user interface and refreshed in real-time.

Conclusions

Thanks to the project, we could practically use the Java language we have been learning this semester. It was also an opportunity to get acquainted with creating graphical interfaces, which we had not done before. We could also observe the difference in working with the C language and the much more convenient Java language.