Dokumentacja Programu Znajdywania Najkrótszej Ścieżki w Labiryncie z Interfejsem Graficznym

Mikolaj Ciuraba, Adam Wylot

Czerwiec 2024

Spis Treści

1 Wstęp do programu **3**
- 1.1 Specyfikacja programu . . . 3
- 1.2 Przeznaczenie . . . 3
- 1.3 Działanie programu . . . 3
- 1.4 Ograniczenia . . . 6

2 Opis zrealizowanej funkcjonalności **7**
- 2.1 Kompilacja programu . . . 7
- 2.2 Uruchamianie programu . . . 7
- 2.3 Obsługa błędów . . . 7
- 2.4 Wczytanie pliku z labiryntem . . . 7
- 2.5 Przedstawienie graficzne labiryntu . . . 7
- 2.6 Znajdywanie najkrótszej ścieżki . . . 7
- 2.7 Przedstawienie graficzne ścieżki . . . 7
- 2.8 Zapis informacji o ścieżce . . . 9

3 Opisy wykorzystywanych plików **10**
- 3.1 Pliki wejściowe . . . 10
  - 3.1.1 Plik tekstowy . . . 10
  - 3.1.2 Plik binarny . . . 10
- 3.2 Pliki wyjściowe . . . 12
  - 3.2.1 Plik tekstowy . . . 12
  - 3.2.2 Plik binarny . . . 12
  - 3.2.3 Plik *.png* . . . 12

4 Diagram UML programu **13**

5 Opis klas i interfejsów programu **15**

6 Pakiet common **15**
- 6.1 BinaryFH . . . 15
- 6.2 InputFH . . . 16
- 6.3 TextFH . . . 17
- 6.4 Cell . . . 17
- 6.5 CellQueue . . . 18
- 6.6 Core . . . 18
- 6.7 Maze . . . 19
- 6.8 PathFinder . . . 20

6.9 Settings 20
6.10 Validator 21

7 Sekcja GUI **22**
7.1 Przyciski 22
7.1.1 FindPathButton 22
7.1.2 MazeButton 22
7.1.3 PaletteButton 23
7.1.4 SettingsButton 23
7.1.5 SwitchButton 23
7.2 Panele 24
7.2.1 CanvasPanel 24
7.2.2 MazePanel 25
7.2.3 ScrollPanel 26
7.2.4 SettingsPanel 26
7.2.5 ToolsPanel 27
7.2.6 UtilityPanel 28
7.3 Główne Okno 28
7.3.1 MainWindow 29
7.3.2 MenuBar 29

8 Interfejsy **31**
8.1 ICallBack 31
8.2 IfileHandler 31
8.2.1 Metody 31

9 Wykorzystywane algorytmy **32**
9.1 Opis działania algorytmu BFS 32

1 Wstęp do programu

1.1 Specyfikacja programu

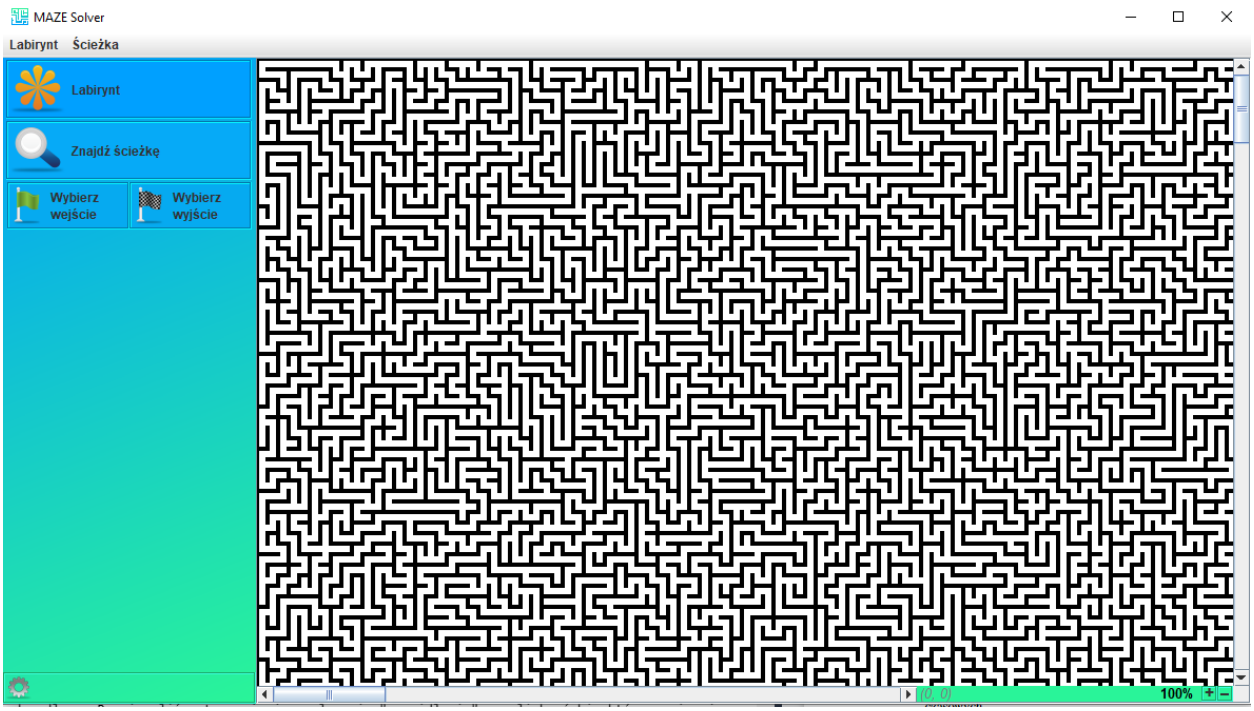
- System operacyjny: **multiplatformowy**
- Wymagana Java zainstalowana na komputerze.
- Użyty język programowania: **Java**
- Zgodne kompilatory: **javac**
- Zalecana pamięć operacyjna: **2GB**

1.2 Przeznaczenie

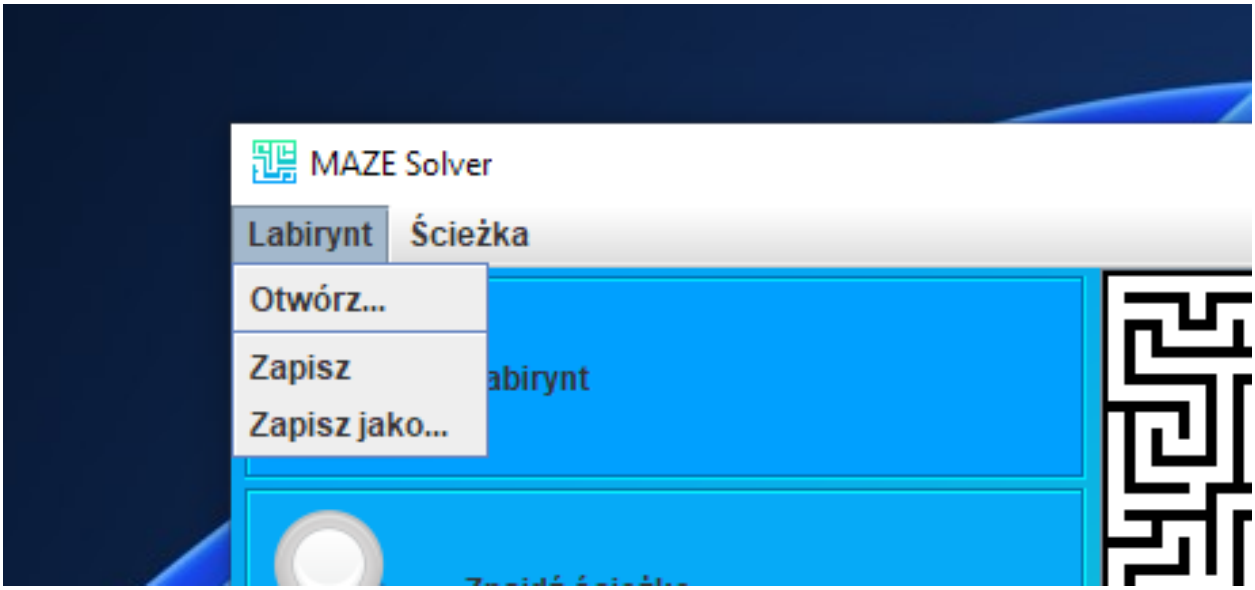
Program szuka i graficznie przedstawia rozwiązanie wprowadzonego do niego labiryntu jako plik binarny lub tekstowy. Rozwiązanie (o ile istnieje) labiryntu jest albo najkrótszą ścieżką od wytycznych z pliku (wejście, wyjście - prawy górny i lewy dolny róg labiryntu) albo określonymi przez użytkownika koordynatami wyjścia i wejścia. Mechanizm działania i poruszania się po programie, wraz z jego funkcjonalnością, jest bardziej szczegółowo opisany w *Rozdział 1.3*

1.3 Działanie programu

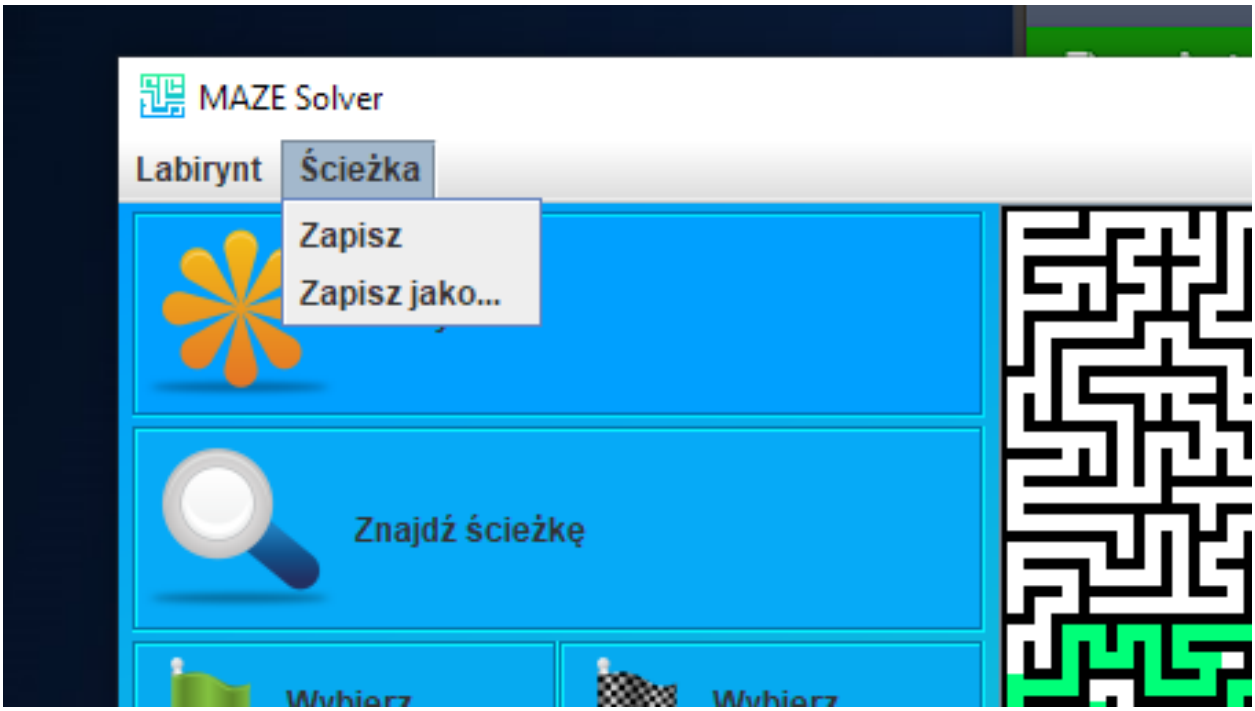
Program posiada interfejs użytkownika, wykorzystując biblioteke Swing, który umożliwia proste poruszanie się po programie. Najpierw należy wczytać labirynt z pliku tekstowego, bądź binarnego, przez kliknięcie w pasek narzędzi i wybranie pliku. Po wczytaniu labiryntu, jest on odczytywany i przedstawiany graficznie po prawej stronie okna, co widać na *Zdjęcie 1.* Użytkownik może znaleźć najkrótszą ścieżkę w labiryncie, która również narysuje się w oknie, na labirycnie. Ponadto, użytkownik może zaznaczyć koordynaty wejścia i wyjścia nakierowywując myszką na daną komórkę labiryntu (dla ułatwienia, na komórce rysuje się niebieski kwadrat, który informuje o tym, że jest aktualnie zaznaczona). Program posiada możliwość zmiany koloru ścieżki, koloru tła programu, zapisu znaku ścieżki, ściany, pustego pola, wejścia i wyjścia z labiryntu względem formatu pliku. Również ma funkcjonalność zapisu ścieżki (wraz z labiryntem) do pliku tekstowego, binarnego i pliku o formacie *png*.



Zdjęcie 1: Wygląd programu



Zdjęcie 2: Pasek menu na górze programu z rozwiniętą opcją "Labirynt".



Zdjęcie 3: Pasek menu na górze programu z rozwiniętą opcją "Ścieżka".

1.4 Ograniczenia

Podstawowym ograniczeniem programu jest format pliku (Patrz *Sekcja 3.1*), który z góry jest ustalony. Dodatkowe ograniczenia:

- plik z labiryntem nie może zawierać nieokreślonych znaków, które nie są obsługiwane przez program (znaki te można zobaczyć i zmienić w ustawieniach programu),
- wprowadzony do programu labirynt musi zawierać dokładnie jedno wejście i dokładnie jedno wyjście,
- labirynt musi być poprawnie skonstruowany – nie może być otwartych ścian zewnętrznych,
- wejście oraz wyjście muszą znajdować się na którejś ze ścian zewnętrznych lub w miejscu komórki,
- w miejscu komórki labiryntu nie może znajdować się znak ściany lub znak, który nie jest przypisany do żadnej opcji,
- natomiast w rogu komórki dozwolony jest wyłącznie znak ściany.

2 Opis zrealizowanej funkcjonalności

2.1 Kompilacja programu

Program został napisany w Javie, jako projekt typu *maven*. Aby skompilować program, należy otworzyć kod źródłowy w środowisku IDE i nacisnąć przycisk odpowiedzialny za kompilacje i uruchamianie się programu. Można też skompilować program z poziomiu wiersza poleceń. Żeby to zrobić należy mieć zainstalowany Apache Maven na komputerze. Następnie:

- udać się w wierszu poleceń do lokalizacji głównego folderu projektu,
- wpisać polecenie **mvn compile**

Warto zaznaczyć, że wszystkie zależności zapisane są w pliku *pom.xml*, który również znajduje się w głównym katalogu programu.
Można też utworzyć plik *.jar* używając polecenia **mvn clean package** lub tworząc go z poziomu IDE. Skompilowany plik *maze-solver-x.x.jar* pojawi się w katalogu **target**.

2.2 Uruchamianie programu

Z poziomu IDE wystarczy nacisnąć przycisk odpowiedzialny za uruchamianie programu.
Z poziomu wiersza poleceń należy:

- udać się w wierszu poleceń do lokalizacji głównego folderu projektu,
- wpisać polecenie **mvn exec:java -Dexec.mainClass="Main"**

Plik *.jar* wystarczy kliknąć dwukrotnie na plik programu lub udać się w wierszu poleceń do lokalizacji pliku i wpisać polecenie **java -jar <ścieżka>**.

2.3 Obsługa błędów

Program obsługuje błędy za pomocą pakietu Exceptions i klasy Validator (szerzej opisane w *Sekcja 6.10. Validator*).

2.4 Wczytanie pliku z labiryntem

Program przyjmuje dwa formaty plików z labiryntem: plik tekstowy oraz plik binarny. Oba zostały szerzej omówione w *sekcji 3.1 Pliki wejściowe*. Program odczytuje z nich informacje o labiryncie i zamienia w dwuwymiarową tablicę obiektów typu *Cell*.

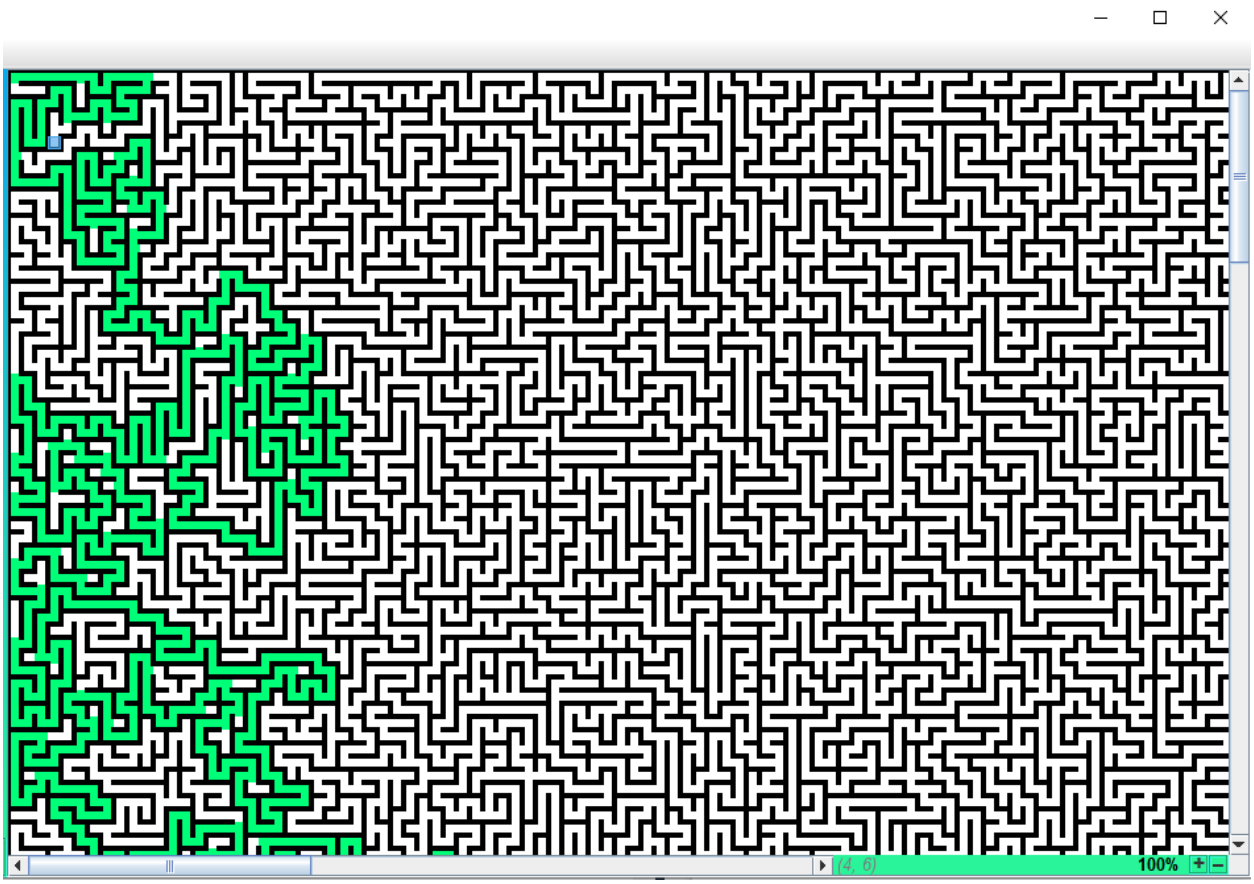
2.5 Przedstawienie graficzne labiryntu

2.6 Znajdywanie najkrótszej ścieżki

Program wykorzystuje algorytm BFS do znajdywania najkrótszej ścieżki w labiryncie, względem punktu wejścia i wyjścia, które mogą być ustawione przez użytkownika programu. Algorytm BFS został opisany w *Sekcja 6.*

2.7 Przedstawienie graficzne ścieżki

Po odnalezieniu ścieżki, zostaje ona nałożona na panel labiryntu w oknie, w kolorze wybranym w ustawieniach programu.



Zdjęcie 4: Wygląd okna z labiryntem

2.8 Zapis informacji o ścieżce

Program ma funkcjonalność zapisu ścieżki wraz z labiryntem do pliku tekstowego (*.txt*), binarnego (*.bin*), bądź też do obrazu (*.png*).

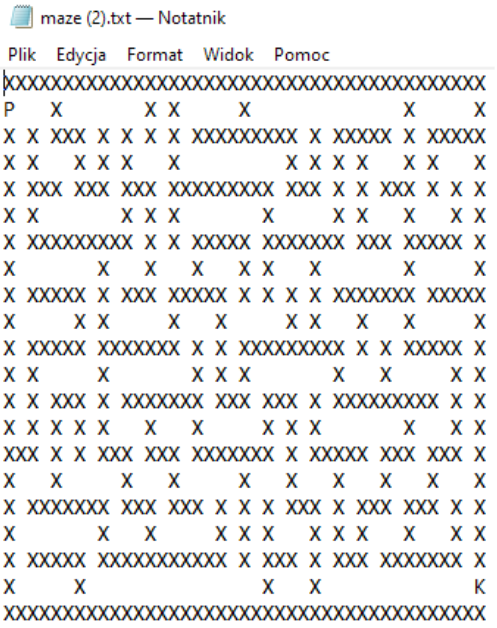
3 Opisy wykorzystywanych plików

3.1 Pliki wejściowe

Program obsługuje dwa typy plików, czyli plik tekstowy (*.txt*) oraz plik binarny (*.bin*).

3.1.1 Plik tekstowy

Przykładowy plik tekstowy:



Zdjęcie 5: przykładowy plik tekstowy z zakodowanym labiryntem

Jak dobrze widać jest to forma labiryntu opisanego przez znaki, gdzie znak **X** oznacza ścianę, znak *spacji* przejście (lub komórkę), znak **P** to wejście do labiryntu, natomiast znak **K** to jego wyjście.
Nasz program umożliwia zmianę tych znaków. Można je określić w module Settings.h.

3.1.2 Plik binarny

Plik binarny jest podzielony na 4 sekcje. Są to:

1. nagłówek pliku,
2. sekcja kodująca zawierająca powtarzające się słowa kodowe - opis labiryntu,
3. nagłówek sekcji rozwiązania,
4. sekcja rozwiązania zawierające powtarzające się kroki, które należy wykonać aby przejść przez labirynt najkrótszą możliwą ścieżką.

Sekcja 1. i 2. są obowiązkowe i zawsze występują, sekcja 3. oraz 4. są opcjonalne. Występują tylko i wyłącznie wtedy, gdy wartość pola **Solution Offset** z nagłówka pliku jest różna od 0 - istnieje odnalezione rozwiązanie labiryntu.

Nagłówek tego 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

Opis słów kodowych w pliku:

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	32	Liczba kroków do przejścia (0 – oznacza jeden krok)

w pliku .pdf z opisem pliku binarnego Steps miało **8 bitów, co nie pozwalało na zapis każdej ścieżki, a nawet dowolnej w labiryncie **1024x1024**, więc zmieniliśmy tę wartość na 32 bity (zakres zmiennej Integer)*

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)

3.2 Pliki wyjściowe

Po znalezieniu ścieżki z wczytanego labiryntu, użytkownik może zapisać labirynt ze ścieżką, do podanego przez niego pliku w oknie dialogowym lub do pliku, który pierwotnie został wczytany do programu – program nadpisuje go.

3.2.1 Plik tekstowy

Plik tekstowy posiada zapisaną ścieżkę jako znaki w labiryncie, które pokazuja drogę.

3.2.2 Plik binarny

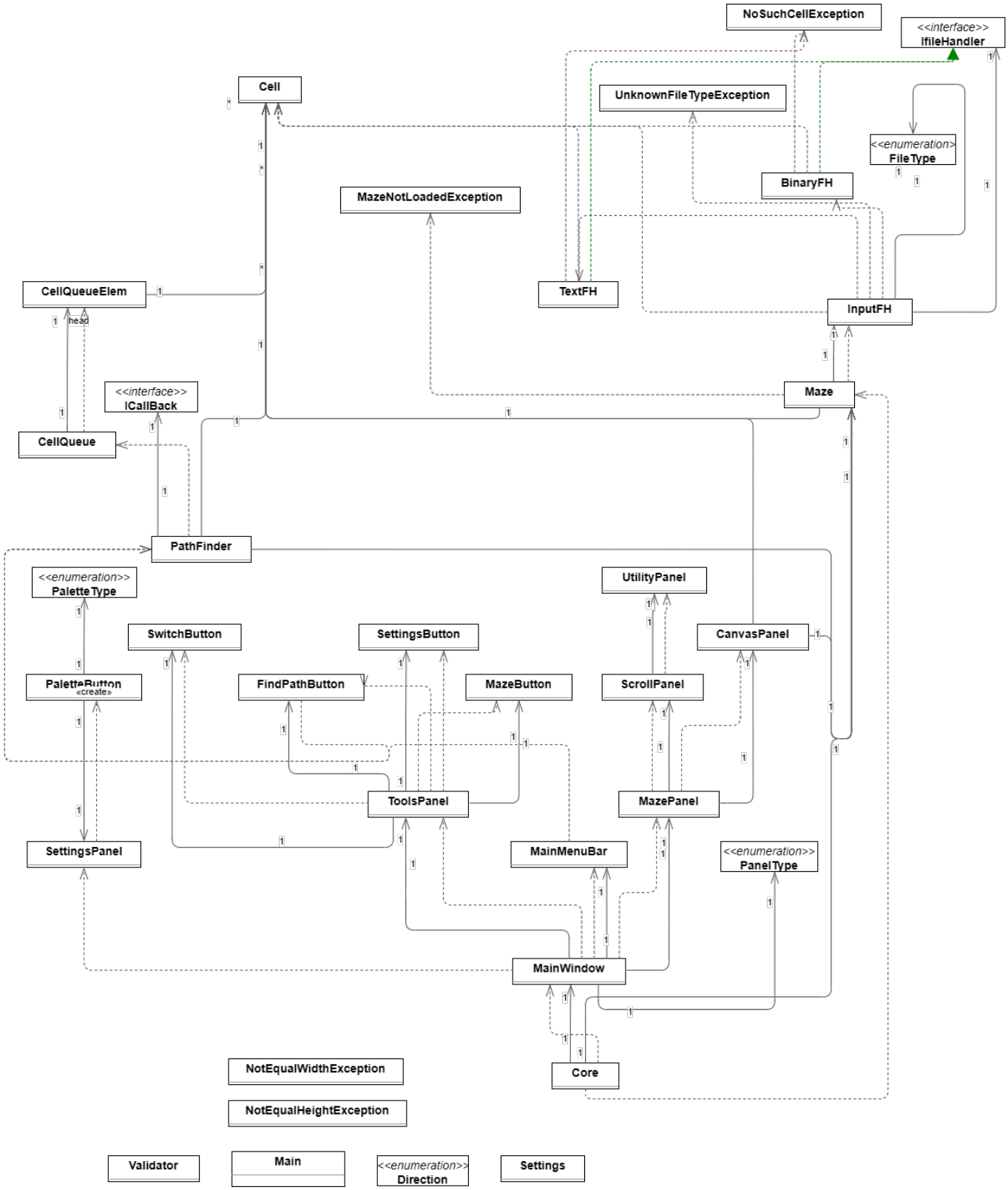
Do pliku binarnego jest dopisywana sekcja z rozwiązaniem, która jest opisana w *Sekcja 3.1.2*. Program zmienia pole **Solution Offset**, znajdującej się w nagłówku pliku, na wartość różną od zera, oraz zapisuje finalną wersję rozwiązania w odpowiedniej sekcji pliku z labiryntem. Dokładniej mówiąc pole Solution Offset zawiera informację, w którym miejscu w pliku zaczyna się opis rozwiązania labiryntu. Wartość 0 oznacza, że rozwiązanie nie istnieje lub nie zostało jeszcze odnalezione.

3.2.3 Plik *.png*

Możliwym jest również zapisanie labiryntu (do wyboru wraz ze ścieżką) do pliku z roszerzeniem *.png* – czyli graficznie.

4 Diagram UML programu

Diagram UML programu znajduje się na następnej stronie, zawierając wszystkie użyte klasy i interfejsy w programie, wraz z ich zależnościami. Szczegółowy opis danej klasy lub interfejsu znajduje się w kolejnej sekcji.



Zdjęcie 6: Diagram UML programu

5 Opis klas i interfejsów programu

Opis klas i interfejsów programu został podzielony na dwie sekcje, *Sekcja 6 i Sekcja 7*, gdzie pierwsza sekcja za opis pakietu *common*, który zawiera większość logiki programu oraz sekcji **GUI**, która odpowiada za interfejs użytkownika i związane z nim funkcje.

6 Pakiet common

Pakiet common zawiera główną logikę programu. Znajdują się tam informacje logiczne o labirycnie i jego komórkach, jak i sama funkcjonalność odczytywania pliku. W tym pakiecie również znajduje się cała logika dotycząca znajdywania ścieżki na podstawie wczytanego labiryntu (za pomocą algorytmu BFS, który opiera się na kolejce komóek, głebiej opisane w *Sekcja 8*). Ponadto, znajduje się tutaj funkcjonalność sprawdzania poprawności formatu wczytywanego pliku, oraz jego ustawienia, tak samo jak i ogólne ustawienia programu.

6.1 BinaryFH

Klasa BinaryFH odpowiada za operacje związane z obsługą plików binarnych zawierających labirynt. Ta klasa posiada pola takie same jak format pliku binarnego, opisanego w *Sekcja 3.1.2*. Ma funkcjonalność odczytu jak i zapisu informacji z lub do pliku binarnego. Tak jak reszta handlerów pliku, implementuje ona interfejs *IfileHandler*

Pola w tej klasie:

- **path** (*String*): Ścieżka do pliku.
- **head** (*int*): Nagłówek pliku.
- **escape** (*byte*): Znak ucieczki.
- **width** (*int*): Szerokość labiryntu.
- **height** (*int*): Wysokość labiryntu.
- **entryX** (*short*): Współrzędna X punktu wejścia.
- **entryY** (*short*): Współrzędna Y punktu wejścia.
- **exitX** (*short*): Współrzędna X punktu wyjścia.
- **exitY** (*short*): Współrzędna Y punktu wyjścia.
- **reserved** (*byte[]*): Zarezerwowane bajty.
- **counter** (*int*): Licznik.
- **solutionOffset** (*int*): Przesunięcie rozwiązania.
- **separator** (*byte*): Separator.
- **word_wall** (*byte*): Znak ściany.
- **word_path** (*byte*): Znak przejścia.
- **maze_cells** (*char [] []*): Tablica komórek labiryntu.

Metody klasy i ich opis:

- **BinaryFH(String path)**: Konstruktor klasy. Inicjalizuje obiekt klasy na podstawie ścieżki do pliku binarnego.
- **isBinaryFile(String path)**: Metoda statyczna sprawdzająca, czy podany plik jest plikiem binarnym.
- **saveMaze(Maze maze, String filePath, boolean doWritePath)**: Metoda statyczna zapisująca labirynt do pliku binarnego.
- **readBinaryFile(String path)**: Metoda odczytująca plik binarny i inicjalizująca komórki labiryntu.
- **getWidth()**: Metoda zwracająca szerokość labiryntu.
- **getHeight()**: Metoda zwracająca wysokość labiryntu.
- **getStartCellX()**: Metoda zwracająca współrzędną X punktu wejścia.
- **getStartCellY()**: Metoda zwracająca współrzędną Y punktu wejścia.
- **getEndCellX()**: Metoda zwracająca współrzędną X punktu wyjścia.
- **getEndCellY()**: Metoda zwracająca współrzędną Y punktu wyjścia.
- **getCellAt(int x, int y)**: Metoda zwracająca komórkę labiryntu o podanych współrzędnych.

6.2 InputFH

Klasa *InputFH* odpowiada za operacje związane z odczytem plików wejściowych jak i jest "fabryką" dla czytnika pliku z labiryntem. Jego głównym zadaniem jest rozróżnienie jaki typ pliku został podany do odczytu, oraz ustawienie specyfikacji tego pliku. Tak jak reszta handlerów pliku, implementuje ona interfejs *IfileHandler*

Opis pól tej klasy:

- **fileType** (*FileType*): Typ pliku.
- **filePath** (*String*): Ścieżka do pliku.
- **fileHandler** (*IfileHandler*): Obiekt implementujący interfejs IFileHandler

Konstruktor:

- **InputFH(String path)**: Konstruktor klasy. Inicjalizuje obiekt klasy na podstawie ścieżki do pliku wejściowego.

Metody i ich opis:

- **FileHandler()**: Metoda zwracająca handler pliku.
- **getPath()**: Metoda zwracająca ścieżkę do pliku.
- **recognizeFileType(String path)**: Metoda rozpoznająca typ pliku na podstawie jego ścieżki.
- **readCells(int width, int height)**: Metoda odczytująca komórki labiryntu z pliku wejściowego i zwracająca je w postaci dwuwymiarowej tablicy komórek.

6.3 TextFH

Klasa *TextFH* odpowiada za operacje związane z obsługą plików tekstowych. Posiada ona funkcjonalność odczytu labiryntu i zapisu jego rozwiązania. Tak jak reszta handlerów pliku, implementuje ona interfejs *IfileHandler*

Pola tej klasy

- **path** (*String*): Ścieżka do pliku.
- **width** (*int*): Szerokość labiryntu.
- **height** (*int*): Wysokość labiryntu.
- **map** (*char[][]*): Dwuwymiarowa tablica reprezentująca labirynt.

Konstruktor

- **TextFH(String path)**: Konstruktor klasy. Inicjalizuje obiekt klasy na podstawie ścieżki do pliku tekstowego.

Opis metod tej klasy:

- **isTextFile(String path)**: Metoda statyczna sprawdzająca, czy plik o podanej ścieżce jest plikiem tekstowym.
- **saveMaze(Maze maze, String filePath, boolean doWritePath)**: Metoda statyczna zapisująca labirynt do pliku tekstowego.
- **readWidth()**: Metoda odczytująca szerokość labiryntu z pliku tekstowego.
- **readHeight()**: Metoda odczytująca wysokość labiryntu z pliku tekstowego.
- **readFile()**: Metoda odczytująca zawartość pliku tekstowego i zapisująca ją do tablicy **map**.
- **getStartCellX()**: Metoda interfejsu *IfileHandler* zwracająca współrzędną X komórki startowej.
- **getStartCellY()**: Metoda interfejsu *IfileHandler* zwracająca współrzędną Y komórki startowej.
- **getEndCellX()**: Metoda interfejsu *IfileHandler* zwracająca współrzędną X komórki końcowej.
- **getEndCellY()**: Metoda interfejsu *IfileHandler* zwracająca współrzędną Y komórki końcowej.
- **getCellAt(int x, int y)**: Metoda interfejsu *IfileHandler* zwracająca komórkę o podanych współrzędnych.

6.4 Cell

Klasa *Cell* reprezentuje pojedynczą komórkę w labiryncie. Opisuje ona komórkę posiadając informacje apropo jej koordynatach i ścianach wokół komórki, jako zmienna walls typu byte. Wartośc '1' oznacza przejście a wartość '0' oznacza ścianę.

Pola tej klasy:

- **x** (*int*): Współrzędna X komórki.
- **y** (*int*): Współrzędna Y komórki.

- **walls** (*byte*): Zmienna określająca stan ścian komórki. Każdy z pierwszych (od prawej strony) czterech bitów reprezentuje jedną ze ścian, gdzie 1 oznacza otwartą ścianę, a 0 zamkniętą.

Konstruktor:

- **Cell(int x, int y)**: Konstruktor klasy. Inicjalizuje obiekt klasy na podstawie podanych współrzędnych, ustawiając początkowo wszystkie ściany jako zamknięte.

Opis metod tej klasy:

- **setPass(boolean is_open, Direction direction)**: Metoda ustawiająca stan danej ściany komórki (otwarta lub zamknięta) w określonym kierunku.
- **getX()**: Metoda zwracająca współrzędną X komórki.
- **getY()**: Metoda zwracająca współrzędną Y komórki.
- **isWallOpen(Direction direction)**: Metoda sprawdzająca, czy ściana w określonym kierunku jest otwarta.

6.5 CellQueue

Ta klasa reprezentuje własnoręcznie napisaną kolejke komórek. Każdy element tej kolejki posiada informacje na temat danej komórki, następnej komórki w kolejce i też rodzica danej komórki.

Pola tej klasy:

- **head** (*CellQueueElem*): Referencja do pierwszego elementu kolejki.
- **last** (*CellQueueElem*): Referencja do ostatniego elementu kolejki.
- **size** (*int*): Liczba elementów w kolejce.

Konstruktor

- **CellQueue()**: Konstruktor klasy. Inicjalizuje kolejkę jako pustą.

Metody:

- **getCell()**: Metoda zwracająca komórkę znajdującą się na szczycie kolejki.
- **getSize()**: Metoda zwracająca liczbę elementów w kolejce.
- **add(Cell cell)**: Metoda dodająca nową komórkę do kolejki.
- **next()**: Metoda usuwająca pierwszą komórkę z kolejki.
- **clean()**: Metoda usuwająca wszystkie elementy z kolejki, resetująca jej stan.
- **goToParent()**: Metoda przenosząca wskaźnik aktualnej komórki na komórkę rodzica. Zwraca 0, jeśli operacja powiodła się, a 1 w przeciwnym przypadku.

6.6 Core

Ta klasa jest główną klasą projektu. Posiada informacje na temat nazwy programu, rozmiaru okna, które są ustawiane przez blok inicjalizujący statyczny. Rozpoczyna ona pracę programu, wyświetlając główne okno.

Ponadto, tworzy ona obiekt interfejsu 'ICallBack' w wątku głównym, by po wykonaniu zadania w wątku wczytującym labirynt rozpoczęło się wczytywanie labiryntu do panelu. Wczytywanie labiryntu do programu dzieje się również na innym wątku, by program nie zacinał się przy wczytywaniu dużych rozmiarów labiryntu.

Pola tej klasy:

- **programName** (*String*): Nazwa programu.
- **scaleX** (*double*): Skala szerokości ekranu.
- **scaleY** (*double*): Skala wysokości ekranu.
- **mainWindow** (*MainWindow*): Referencja do głównego okna aplikacji.
- **maze** (*Maze*): Referencja do aktualnie wczytanego labiryntu.

Metody

- **start()**: Metoda rozpoczynająca działanie aplikacji przez tworzenie głównego okna.
- **printError(String message)**: Metoda służąca do wyświetlania błędów w konsoli.
- **loadMazeFile(String path)**: Metoda wczytująca plik z labiryntem na osobnym wątku, aby zapobiec zacięciu się programu przy dużych plikach.

6.7 Maze

Klasa reprezentująca labirynt w programie. Posiada informacje na temat komórek w labiryncie (które są przetrzymywane jako dwuwymiarowa tablica typu Cell). Posiada również listę komórek, które składają się na najkrótszą ścieżkę w labiryncie.

Pola tej klasy:

- **width** (*int*): Szerokość labiryntu.
- **height** (*int*): Wysokość labiryntu.
- **cells** (*Cell[][]*): Tablica komórek labiryntu.
- **startCell** (*Cell*): Komórka startowa labiryntu.
- **endCell** (*Cell*): Komórka końcowa labiryntu.
- **path** (*ArrayList<Cell>*): Ścieżka w labiryncie.
- **inputFH** (*InputFH*): Obiekt obsługujący wczytywanie pliku labiryntu.

Konstruktor

- **Maze(String filePath)**: Konstruktor klasy Maze, który tworzy obiekt labiryntu na podstawie pliku z labiryntem. Rzuca wyjątkiem MazeNotLoadedException w przypadku błędów podczas wczytywania lub tworzenia labiryntu.

Metody tej klasy:

- **mazeClickedAt(int x, int y)**: Metoda obsługująca kliknięcie na labiryncie, ustawiająca komórkę startową lub końcową w zależności od aktywności odpowiedniego przycisku narzędziowego.

6.8 PathFinder

Klasa odpowiedzialna za znajdywanie ścieżki w labiryncie, używając do tego wcześniej opisanej kolejki komórek, oraz implementacji algorytmu BFS na znajdywanie najkrótszej ścieżki. Dziedziczy ona po klasie *Thread*, aby wykonywać operację znajdywania ścieżki na oddzielnym wątku, by przyspieszyć pracę programu.

Pola tej klasy:

- `private static boolean isTaskActive`: Zmienna statyczna, która śledzi, czy zadanie jest aktywne.
- `private Maze maze`: Labirynt, w którym szukana jest ścieżka.
- `private final ArrayList<Cell> path`: Lista przechowująca komórki znalezionej ścieżki.
- `private boolean foundPath`: Zmienna określająca, czy ścieżka została znaleziona.
- `private ICallBack callback`: Obiekt implementujący interfejs `ICallBack` służący do powiadomienia o zakończeniu zadania.

Konstruktor

- **PathFinder(Maze maze)**: Konstruktor klasy *PathFinder*, inicjalizuje obiekt wątku dla określonego labiryntu.

Metody:

- **run()**: Metoda implementująca główną logikę algorytmu znajdowania ścieżki w labiryncie na oddzielnym wątku.
- **setPath(CellQueue queue)**: Metoda ustawiająca znalezioną ścieżkę na podstawie kolejki komórek przeszukanych przez algorytm.

6.9 Settings

Klasa odpowiadająca za przetrzymywanie informacji na temat ogólnych ustawień programu. Te ustawienia obejmują znaki określające ścianę i przejście w komórce, wejście i wyjście z labiryntu i znak ścieżki. Ponadto, przetrzymuje informacje o wyglądzie programu. Klasa posiada prywatny i pusty konstruktor.

Pola klasy:

- **WALL_CHAR**: Znak reprezentujący ścianę w labiryncie.
- **PASS_CHAR**: Znak reprezentujący przejście w labiryncie.
- **ENTRY_CHAR**: Znak reprezentujący wejście do labiryntu.
- **EXIT_CHAR**: Znak reprezentujący wyjście z labiryntu.
- **PATH_CHAR**: Znak reprezentujący ścieżkę w labiryncie.
- **PATH_COLOR**: Kolor używany do oznaczania ścieżki w labiryncie.
- **BACKGROUND_COLOR**: Kolor tła labiryntu.

Metody:

- **getBackgroundColor()**: Metoda zwracająca kolor tła labiryntu.

- **setBackgroundColor(Color color)**: Metoda ustawiająca kolor tła labiryntu na podstawie parametru *color*.

6.10 Validator

Klasa odpowiedzialna za sprawdzenie poprawności formatu wczytywanego pliku z labiryntem. Posiada prywatny konstruktor.

Metody

- **isValid(char[][] map)**: Ta metoda przyjmuje tablicę znaków ‘map‘, która reprezentuje labirynt. Sprawdza ona poprawność struktury labiryntu oraz znaków w tablicy. Algorytm najpierw sprawdza, czy labirynt jest otoczony ścianami. Następnie weryfikuje poprawność położenia wejścia i wyjścia, upewniając się, że każde z nich występuje dokładnie raz oraz jest dostępne. Kolejnym krokiem jest sprawdzenie, czy wszystkie znaki w labiryncie są poprawne. Metoda analizuje również ściany i przejścia wewnątrz komórek labiryntu, aby upewnić się, że są one zgodne z oczekiwaniami. Jeśli jakakolwiek nieprawidłowość zostanie wykryta, metoda rzuca wyjątek *InvalidFileException* z odpowiednim komunikatem o błędzie. W przypadku poprawnego labiryntu zwraca wartość 0.

7 Sekcja GUI

Jak wcześniej zostało wspomniane, program został podzielony na częśc implementacyjną ze strony logicznej i ze strony graficznej (frontend). Teraz zostanie opisana sekcja graficzna, która odpowiada za rysowanie labiryntu, przedstawienie graficzne okien i funkcjonalności programu. Wygląd głównego okna programu został zwizualizowany w *Zdjęcie 1.* Cały interfejs graficzny został napisany z wykorzystaniem wbudowanej biblioteki Swing Javy. Po każdym opisie klasy, pojawi się zdjęcie wizualizujące jej wygląd w programie okienkowym.

7.1 Przyciski

Przyciski zostały zapakowane w pakiet 'buttons' dla lepszej organizacji przestrzeni projektowej. Każdy przycisk ma swoją własną klasę.

7.1.1 FindPathButton

Klasa *FindPathButton* reprezentuje przycisk służący do znajdowania ścieżki w labiryncie, po uwczesnym wczytaniu labiryntu z pliku. Tak samo jak każda inna klasa przycisku, dziedziczy ona po klasie JButton.

Pola:

- **icon** (*ImageIcon*): Ikona przycisku.

Konstruktor:

- **FindPathButton(Dimension dimension)**: Konstruktor klasy. Tworzy przycisk i wczytuje ikonę przycisku "Znajdź ścieżkę". Ustawia preferowany rozmiar przycisku oraz jego wygląd. Dodaje funkcjonalność przycisku, aby po jego kliknięciu rozpoczął się proces znajdowania ścieżki w labiryncie.

Metody:

- **click()**: Metoda wywoływana po kliknięciu przycisku. Sprawdza, czy labirynt został wczytany. Jeśli tak, tworzy instancję klasy *PathFinder* i rozpoczyna wątek poszukiwania ścieżki. Po zakończeniu poszukiwań, wywołuje callback, który zaznacza znalezioną ścieżkę na panelu labiryntu oraz włącza odpowiednie opcje w menu głównym. W przypadku niepowodzenia, wyświetla komunikat o braku znalezionej ścieżki.

7.1.2 MazeButton

Klasa *MazeButton* reprezentuje przycisk służący do przełączania panelu na panel z labiryntem.

Konstruktor

- **MazeButton(Dimension dimension)**: Konstruktor klasy. Tworzy przycisk i wczytuje ikonę przycisku "Labirynt". Ustawia preferowany rozmiar przycisku oraz jego wygląd. Dodaje funkcjonalność przycisku, aby po jego kliknięciu aktywowany był panel z labiryntem.

Metody

- **click()**: Metoda wywoływana po kliknięciu przycisku. Aktywuje panel z labiryntem w oknie głównym aplikacji.

7.1.3 PaletteButton

Klasa *PaletteButton* reprezentuje przycisk służący do wyboru koloru dla ścieżki lub tła w panelu ustawień.

Pola

- **icon** (*ImageIcon*): Ikona przycisku.
- **parent** (*SettingsPanel*): Referencja do panelu ustawień, którego dotyczy przycisk.
- **type** (*PaletteType*): Typ przycisku określający, czy dotyczy on koloru ścieżki czy tła.
- **color** (*Color*): Aktualny wybrany kolor.

Konstruktor

- **PaletteButton(PaletteType type, SettingsPanel parent)**: Konstruktor klasy. Tworzy przycisk do wyboru koloru, inicjalizując jego typ (ścieżka lub tło) oraz referencję do panelu ustawień. Ustawia preferowany rozmiar przycisku, jego wygląd oraz dodaje funkcjonalność przycisku do wyboru koloru.

Metody

- **click()**: Metoda wywoływana po kliknięciu przycisku. Otwiera okno do wyboru koloru. Po wybraniu koloru aktualizuje odpowiednią wartość koloru w panelu ustawień oraz odświeża status panelu.

7.1.4 SettingsButton

Klasa *SettingsButton* reprezentuje przycisk służący do otwierania panelu ustawień programu.

Pola

- **icon** (*ImageIcon*): Ikona przycisku.

Konstruktor

- **SettingsButton(Dimension dimension)**: Konstruktor klasy. Inicjalizuje przycisk, wczytując ikonę przycisku z pliku. Ustawia preferowany rozmiar przycisku, jego wygląd oraz dodaje funkcjonalność przycisku do otwierania panelu ustawień.

Metody

- **click()**: Metoda wywoływana po kliknięciu przycisku. Ustawia panel ustawień jako aktywny panel w głównym oknie aplikacji.

7.1.5 SwitchButton

Klasa *SwitchButton* reprezentuje przycisk przełącznika, który może być w stanie aktywnym lub nieaktywnym.

Pola

- **instances** (*ArrayList<SwitchButton>*): Lista wszystkich instancji przycisków przełącznika.
- **text** (*String*): Tekst wyświetlany na przycisku.
- **active** (*boolean*): Flaga określająca, czy przycisk jest w stanie aktywnym.

- **inactiveIcon** (*ImageIcon*): Ikona przycisku w stanie nieaktywnym.
- **activeIcon** (*ImageIcon*): Ikona przycisku w stanie aktywnym.
- **inactiveColor** (*Color*): Kolor tła przycisku w stanie nieaktywnym.
- **activeColor** (*Color*): Kolor tła przycisku w stanie aktywnym.

Konstruktor

- **SwitchButton(String Text, Dimension dim, String inactivePath, String activePath, Color inactiveColor, Color activeColor)**: Konstruktor klasy. Inicjalizuje przycisk przełącznika z podanymi parametrami, wczytując ikony przycisku z odpowiednich ścieżek. Ustawia preferowany rozmiar przycisku, jego wygląd oraz dodaje funkcjonalność przycisku do przełączania stanu aktywności.

Metody

- **click()**: Metoda wywoływana po kliknięciu przycisku. Zmienia stan aktywności przycisku.
- **setActive(boolean active)**: Metoda ustawiająca stan aktywności przycisku na podstawie przekazanego argumentu. Aktualizuje wygląd przycisku zgodnie z nowym stanem.
- **isActive()**: Metoda zwracająca informację o aktualnym stanie aktywności przycisku.
- **deactivateAll()**: Metoda deaktywująca wszystkie przyciski przełącznika.

7.2 Panele

Panele, podobnie jak przyciski, zostały zapakowane w pakiet 'panels' dla łatwiejszej organizacji plików projektu.

7.2.1 CanvasPanel

Klasa *CanvasPanel* jest panelem służącym do wyświetlania labiryntu oraz interakcji z nim. Jest to panel odpowiedzialny za całą logikę rysowania labiryntu. Aby przyspieszyć proces działania graficznego wyświetlania labiryntu, używa anonimowej klasy wewnętrznej (Thread) do stworzenia nowego wątku, na którym odbywają się procesy graficzne.

Pola

- **background** (*JPanel*): Panel tła.
- **mazeP** (*JPanel*): Panel labiryntu.
- **overlayP** (*JPanel*): Panel nakładki.
- **maze** (*Maze*): Obiekt reprezentujący labirynt.
- **path** (*ArrayList<Cell>*): Ścieżka w labiryncie.
- **mazeImage** (*BufferedImage*): Obraz reprezentujący labirynt.
- **finalImage** (*BufferedImage*): Ostateczny obraz labiryntu z ewentualną ścieżką.
- **chosenImage** (*BufferedImage*): Ikona wskazywana przez myszkę w labiryncie.

- **isRenderingThreadActive** (*boolean*): Flaga określająca, czy wątek renderowania jest aktywny.
- **wallThickness** (*int*): Grubość ścian w labiryncie.
- **cellSize** (*int*): Rozmiar komórki w labiryncie.
- **renderScale** (*double*): Skala renderowania labiryntu.
- **wallColor** (*Color*): Kolor ścian w labiryncie.
- **backgroundColor** (*Color*): Kolor tła labiryntu.
- **mouseInArea** (*boolean*): Flaga określająca, czy mysz znajduje się w obszarze panelu.
- **mouseX** (*int*): Pozycja X myszki w panelu.
- **mouseY** (*int*): Pozycja Y myszki w panelu.

Konstruktor

- **CanvasPanel()**: Konstruktor klasy. Inicjalizuje panele tła, labiryntu i nakładki oraz ustawia domyślne wartości pól.

Metody

- **saveMazeImage(String path, boolean withPath)**: Metoda zapisująca obraz labiryntu do pliku.
- **renderMaze(Maze maze)**: Metoda renderująca labirynt.
- **renderPath(ICallBack callBack)**: Metoda renderująca ścieżkę w labiryncie.
- **drawMaze(Graphics2D g2d, int width, int height)**: Metoda rysująca labirynt.
- **paintCell(Graphics2D g2d, Cell cell)**: Metoda rysująca komórkę w labiryncie.
- **setCanvasSize(int width, int height)**: Metoda ustawiająca rozmiar panelu.
- **changeBackgroundColor(Color color)**: Metoda zmieniająca kolor tła.
- **setMousePositionText(int x, int y)**: Metoda ustawiająca pozycję myszki.
- **initMazePanel()**: Metoda inicjalizująca panel labiryntu.
- **initOverlayPanel()**: Metoda inicjalizująca panel nakładki.

7.2.2 MazePanel

Klasa odpowiedzialna za przetrzymywanie panelu labiryntu. Posiada ona całą możliwą interakcę z labiryntem (wyrysowanie ścieżki, labiryntu, przybliżenie labiryntu).

Pola

- **mazeLoaded** (*boolean*): Flaga informująca o załadowaniu labiryntu.
- **canvasPanel** (*CanvasPanel*): Panel rysujący labirynt.
- **scrollPanel** (*ScrollPanel*): Panel umożliwiający przewijanie labiryntu.

Konstruktory

- **MazePanel()**: Konstruktor klasy. Inicjalizuje panele i ustawia domyślne wartości pól.

Metody

- **markPath(ArrayList<Cell> path)**: Metoda oznaczająca ścieżkę w labiryncie.
- **renderMaze(Maze maze)**: Metoda renderująca labirynt.
- **zoom(double value)**: Metoda przybliżająca lub oddalająca labirynt.
- **getScrollPanel()**: Metoda zwracająca panel umożliwiający przewijanie.
- **getCanvasPanel()**: Metoda zwracająca panel rysujący labirynt.
- **isMazeLoaded()**: Metoda sprawdzająca, czy labirynt został załadowany.

7.2.3 ScrollPanel

Klasa *ScrollPanel* jest rozszerzeniem *JScrollPane* i zawiera dodatkowy panel umożliwiający interakcję z labiryntem.

Pola

- **utilityPanel** (*UtilityPanel*): Panel narzędziowy umożliwiający interakcję z labiryntem.

Konstruktor

- **ScrollPanel(Component view)**: Konstruktor klasy. Inicjalizuje panel przewijania oraz dodatkowy panel narzędziowy.

Metody

- **getUtilityPanel()**: Metoda zwracająca panel narzędziowy.

Klasa wewnętrzna: ScrollPanelLayout Klasa wewnętrzna *ScrollPanelLayout* dziedziczy po *ScrollPaneLayout* i jest odpowiedzialna za układ dodatkowego panelu obok poziomego paska przewijania.

- **layoutContainer(Container parent)**: Metoda przesłonięta z klasy nadrzędnej, odpowiedzialna za układ kontenera. Dodaje dodatkowy panel obok poziomego paska przewijania.

7.2.4 SettingsPanel

Klasa *SettingsPanel* reprezentuje panel ustawień w interfejsie użytkownika.

Pola

- **leftPanel** (*JPanel*): Panel zawierający lewą część ustawień.
- **rightPanel** (*JPanel*): Panel zawierający prawą część ustawień.
- **font** (*Font*): Czcionka używana w panelu ustawień.
- **backButton** (*JButton*): Przycisk służący do powrotu do poprzedniego panelu.

- **okButton** (*JButton*): Przycisk służący do zatwierdzenia zmian ustawień.
- **wallCharTF** (*JTextField*): Pole tekstowe do wprowadzenia znaku dla ściany.
- **passCharTF** (*JTextField*): Pole tekstowe do wprowadzenia znaku dla przejścia.
- **entryCharTF** (*JTextField*): Pole tekstowe do wprowadzenia znaku dla wejścia.
- **exitCharTF** (*JTextField*): Pole tekstowe do wprowadzenia znaku dla wyjścia.
- **pathCharTF** (*JTextField*): Pole tekstowe do wprowadzenia znaku dla ścieżki.
- **pathColor** (*Color*): Kolor ścieżki.
- **backgroundColor** (*Color*): Kolor tła programu.

Konstruktor

- **SettingsPanel()**: Inicjalizuje panel ustawień, tworząc pola tekstowe, przyciski oraz inne elementy interfejsu.

Gettery

- **getPathColor()**: Zwraca kolor ścieżki.
- **getBackgroundColor()**: Zwraca kolor tła programu.

Settery

- **setPathColor(Color color)**: Ustawia kolor ścieżki na podany.
- **setBackgroundColor(Color color)**: Ustawia kolor tła programu na podany.

Metody

- **makeLabel(String text)**: Tworzy etykietę z podanym tekstem i dodaje ją do panelu.
- **makeTextField(String text, Dimension maxSizeDim, Dimension gapDim)**: Tworzy pole tekstowe z podanym tekstem, maksymalną wielkością i odstępem, a następnie dodaje je do panelu.
- **updateStatus()**: Aktualizuje status przycisku zatwierdzenia w zależności od wprowadzonych zmian.
- **setSettings()**: Ustawia wartości ustawień na podstawie wprowadzonych danych.

7.2.5 ToolsPanel

Klasa *ToolsPanel* reprezentuje panel narzędzi w interfejsie użytkownika.

Pola

- **mazeButton** (*MazeButton*): Przycisk do ładowania labiryntu.
- **findPathButton** (*FindPathButton*): Przycisk do znajdowania ścieżki w labiryncie.
- **setStartButton** (*SwitchButton*): Przycisk do ustawiania punktu startowego w labiryncie.
- **setEndButton** (*SwitchButton*): Przycisk do ustawiania punktu końcowego w labiryncie.

- **settingsButton** (*SettingsButton*): Przycisk do otwierania panelu ustawień.

Konstruktor

- **ToolsPanel(int width, int height)**: Inicjalizuje panel narzędzi, tworząc przyciski i ustawiając ich wymiary oraz pozycje.

Gettery

- **getStartButton()**: Zwraca przycisk do ustawiania punktu startowego.
- **getEndButton()**: Zwraca przycisk do ustawiania punktu końcowego.
- **getFindPathButton()**: Zwraca przycisk do znajdowania ścieżki w labiryncie.

Metody

- **paintComponent(Graphics g)**: Metoda odpowiedzialna za rysowanie tła panelu z gradientem kolorów.

7.2.6 UtilityPanel

Klasa *UtilityPanel* reprezentuje panel narzędzi pomocniczych w interfejsie użytkownika.

Pola

- **color** (*Color*): Kolor tła panelu.
- **zoomIn** (*JButton*): Przycisk powiększania widoku.
- **zoomOut** (*JButton*): Przycisk pomniejszania widoku.
- **zoomLabel** (*JLabel*): Etykieta wyświetlająca aktualny poziom powiększenia widoku.
- **positionLabel** (*JLabel*): Etykieta wyświetlająca aktualne położenie kursora.
- **zoom** (*int*): Aktualny poziom powiększenia widoku.

Konstruktor

- **UtilityPanel(int width, int height)**: Inicjalizuje panel narzędzi pomocniczych, ustawiając jego wymiary, tło i zawartość.

Metody

- **setZoomText()**: Aktualizuje tekst na etykiecie *zoomLabel* z informacją o aktualnym poziomie powiększenia widoku.
- **setXYlabel(int x, int y)**: Aktualizuje tekst na etykiecie *positionLabel* z aktualnym położeniem kursora.

7.3 Główne Okno

Główne okno reprezentuje całość interfejsu programu w jednym miejscu. Inicjalizacja wszystkich przycisków, wymiary okien, nazwa programu i wszystkie inne najbardziej podstawowe ustawienia zostały zawarte w tych dwóch klasach.

7.3.1 MainWindow

Klasa *MainWindow* reprezentuje główne okno aplikacji.

Pola

- **panelType** (*PanelType*): Typ aktualnie aktywnego panelu.
- **menuBar** (*MenuBar*): Pasek menu.
- **toolsPanel** (*ToolsPanel*): Panel narzędzi.
- **activePanel** (*JPanel*): Aktualnie aktywny panel.
- **mazePanel** (*MazePanel*): Panel z labiryntem.

Konstruktory

- **MainWindow()**: Inicjalizuje główne okno aplikacji z domyślnymi wymiarami.
- **MainWindow(int width, int height)**: Inicjalizuje główne okno aplikacji z podanymi wymiarami.

Metody

- **setActivePanel(PanelType panelType)**: Ustawia aktywny panel na podstawie przekazanego typu panelu.
- **loadMaze(Maze maze)**: Ładuje labirynt do panelu labiryntu i aktualizuje wygląd interfejsu użytkownika.
- **loadIcon(String path)**: Ładuje ikonę aplikacji z podanej ścieżki i ustawia ją jako ikonę okna.

7.3.2 MenuBar

Klasa *MenuBar* reprezentuje pasek menu w aplikacji.

Pola

- **mazeMenu** (*JMenu*): Menu z opcjami dotyczącymi labiryntu.
- **pathMenu** (*JMenu*): Menu z opcjami dotyczącymi ścieżki w labiryncie.
- **mainWindow** (*JFrame*): Referencja do głównego okna aplikacji.

Konstruktor

- **MenuBar(JFrame frame)**: Inicjalizuje pasek menu dla podanego głównego okna aplikacji.

Metody

- **getPathMenu() : JMenu**: Zwraca menu dotyczące ścieżki w labiryncie.
- **createOpenMM() : JMenuItem**: Tworzy i zwraca opcję otwarcia pliku w menu labiryntu.
- **createSaveMM(boolean doWritePath) : JMenuItem**: Tworzy i zwraca opcję zapisu pliku w menu labiryntu.
- **createSaveAsMM(boolean doWritePath) : JMenuItem**: Tworzy i zwraca opcję zapisu pliku

jako w menu labiryntu.

8 Interfejsy

Program posiada dwa własnoręcznie napisane interfejsy.

8.1 ICallBack

Interfejs *ICallBack* definiuje metodę *onThreadComplete()*, która będzie wywoływana po zakończeniu wątku.

Metody (abstrakcyjne)

- **onThreadComplete() : void**: Metoda do przesłaniania przez klasy implementujące ten interfejs, służy do wywoływania po zakończeniu pracy danego wątku.

8.2 IfileHandler

Interfejs *IFileHandler* definiuje zestaw metod, które muszą być zaimplementowane przez klasy obsługujące operacje na plikach (BinaryFH, InputFH, TextFH). Zostało to stworzone do prostszej operacji na plikach i unifikacji.

8.2.1 Metody

- **getWidth() : int**: Zwraca szerokość labiryntu.
- **getHeight() : int**: Zwraca wysokość labiryntu.
- **getStartCellX() : int**: Zwraca współrzędną X komórki startowej labiryntu.
- **getStartCellY() : int**: Zwraca współrzędną Y komórki startowej labiryntu.
- **getEndCellX() : int**: Zwraca współrzędną X komórki końcowej labiryntu.
- **getEndCellY() : int**: Zwraca współrzędną Y komórki końcowej labiryntu.
- **getCellAt(int x, int y) : Cell**: Zwraca komórkę labiryntu o podanych współrzędnych.

9 Wykorzystywane algorytmy

Nasz program korzysta tylko z jednego algorytmu. Jest to powszechnie znany algorytm **breadth-first search** *(BFS)*, który w bardzo prosty sposób pozwala na znalezienie najkrótszej ścieżki do wybranego wierzchołka w grafie skierowanym lub nieskierowanym.

9.1 Opis działania algorytmu BFS

Labirynt można traktować jako graf, gdzie komórka wejściowa jest wierzchołkiem startowym, a komórka wyjściowa jest traktowana jako moment przerwania działania algorytmu.
Z wierzchołka można przejść tylko do sąsiednich wierzchołków - maksymalnie czterech, o ile istnieje droga pomiędzy nimi.
Droga pomiędzy wierzchołkami istnieje wtedy, gdy pomiędzy komórkami w labiryncie nie ma ściany. Warto zaznaczyć, że **drogi w grafie nie mają wagi**. Wszystkie są jednakowej długości. Jest to warunek konieczny do prawidłowego działania algorytmu BFS.

Algorytm rozpoczyna swoje działanie od wierzchołka startowego, który jest wejściem do labiryntu *(jest tylko jedno wejście, także jest tylko jeden wierzchołek startowy)*. Pierwszym krokiem jest dodanie tego wierzchołka do kolejki *"do sprawdzenia"*. Kolejne kroki algorytmu są powtarzane do momentu wczytania wierzchołka reprezentującego wyjście z labiryntu lub do wyczerpania się elementów w kolejce. Pusta kolejka z nieodnalezioną ścieżką oznacza, że droga od wejścia do wyjścia nie istnieje.

Kolejne kroki algorytmu:

1. Pobranie kolejnego w kolejce wierzchołka:
   - kolejka jest pusta - kończy wykonywanie algorytmu
   - jest jakiś element w kolejce - przechodzi do jego analizy
2. Sprawdzenie, czy wczytany wierzchołek nie jest wyjściem:
   - jest wyjściem - kończy wykonywanie algorytmu
   - nie jest wyjściem - przechodzi do analizy sąsiadów
3. Sprawdzenie ilości sąsiednich wierzchołków:
   - ma tylko jednego sąsiada (tego, z którego dostało się do tego wierzchołka) - przechodzi do kolejnej iteracji algorytmu
   - ma 2 sąsiadów - przechodzi do kolejnego kroku
   - ma 3 lub 4 sąsiadów - dodaje aktualny wierzchołek do listy odwiedzonych rozwidleń i przechodzi do kolejnego kroku
4. Dodanie nowych, nieodwiedzonych sąsiadów na koniec kolejki
5. Powrót do 1. punktu (zapętlenie)

Algorytm BFS przeszukuje graf, rozszerzając się stopniowo na coraz bardziej oddalone od wejścia wierzchołki. Najpierw sprawdza wszystkie wierzchołki oddalone o 1 krok od wierzchołka startowego, potem oddalone o 2 kroki, 3 kroki, 4 kroki itd. aż do znalezienia wyjścia lub przejścia przez wszystkie wierzchołki w grafie.

Aby było możliwe odtworzenie tej ścieżki, algorytm dodając wierzchołki do kolejki, zapisuje przy nich informacje, z którego wierzchołka nastąpiło przejście (kto jest "rodzicem" wierzchołka). Dzięki temu cofając się po rodzicach, aż do momentu znalezienia się w wierzchołku startowym, można odtworzyć przebieg najkrótszej ścieżki.