Maze Solver
Dokumentacja projektu

Tomasz Rogalski

12 czerwca 2024

Spis treści

1 Opis ogólny

Celem projektu jest napisanie programu w Javie z interfejsem graficznym, który pozwala na wczytanie labiryntu z pliku tekstowego lub binarnego. Również powinien mieć on opcje wyświetlania ścieżki rozwiązania labiryntu oraz animacji, która symulowałaby przejście labiryntu przez użytkownika.

2 Uruchomienie programu

2.1 Jak korzystać z programu?

W celu uruchomienia programu, najpierw należy go skompilować poleceniem *javac Main.java*. Do działania programu potrzebujemy labiryntu w pliku tekstowym lub binarnym.

2.2 Uruchomienie programu

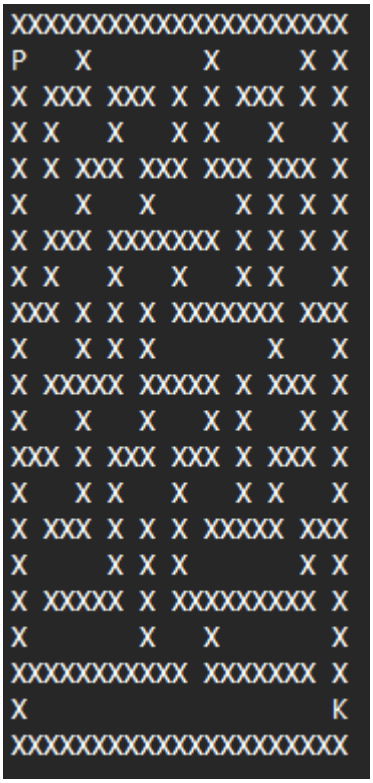
Program uruchamiamy przy użyciu komendy:

```
java -Xss30m Main
```

Program korzysta z metody rekurencyjnej do przetwarzania dużej tablicy. W przypadku rekurencji, każda wywołana metoda dodaje nową ramkę na stosie, co może prowadzić do jego szybkiego przepełnienia, zwłaszcza gdy tablica jest duża, a liczba wywołań rekurencyjnych znaczna. Domyślny rozmiar stosu w JVM może nie być wystarczający dla takiej operacji, co może skutkować błędem ”StackOverflowError”. Aby zapobiec temu problemowi, zwiększamy rozmiar stosu do 30 MB przy użyciu opcji *-Xss30m*. Pozwala to na obsługę głębokiej rekurencji i zapewnia, że program będzie działał poprawnie nawet przy bardzo dużych labiryntach.

2.3 Dane wejściowe

Program przyjmuje labirynt w postaci pliku tekstowego lub binarnego. W tym pierwszym labirynt powinien się składać z: ”X” - ścian, ” ” - pustych pól, po których można się poruszać oraz ”P” i ”K”, czyli początku i końca labiryntu. Dane wejściowe w formacie binarnym składają się na 4 główne sekcje: nagłówek pliku, sekcja kodująca zawierająca powtarzające się słowa kodowe, nagłówek sekcji rozwiązania oraz sekcja rozwiązania zawierająca powtarzające się kroki które należy wykonać aby wyjść z labiryntu. Przykładowy labirynt w pliku tekstowym powinien wyglądać tak:



Rysunek 1: Labirynt smallmaze.txt

3 Opis funkcjonalności

3.1 Modularność

Podział programu na moduły to była podstawa w celu łatwiejszej pracy. Pozwoliło to na prostsze zarządzanie kodem i wprowadzanie do niego zmian.

1. DFS - moduł implementujący algorytm przeszukiwania w głąb (DFS) do znajdowania ścieżki w labiryncie.
    - solve(int startX, int startY) - rozpoczyna przeszukiwanie od punktu startowej zwracając listę punktów tworzących znalezioną ścieżkę lub pustą listę, jeśli rozwiązanie nie istnieje.
    - isValid(int x, int y) - sprawdza czy dane współrzędne są w granicach labiryntu.
    - PathOnList(Map<Point,Point> kroki, Point start, Point end) - odtwarza ścieżkę na podstawie mapy kroków i przetwarza ją na listę.

- getVisited() zwraca tablicę odwiedzonych pól.

2. BFS - moduł odpowiadający za algorytm przeszukiwania wszerz (BFS) do znajdowania najkrótszej ścieżki w labiryncie.
    - solve() - zwraca listę z punktami o najkrótszej ścieżce lub pustą listę, jeśli nie ma rozwiązania.
    - isValidMove(int x, int y) - sprawdza czy rich dla danych współrzędnych jest możliwy.
    - reconstructPath(HashMap<Point, Point> kroki) - odtwarza ścieżkę na podstawie mapy kroków i przetwarza ją na listę.
    - getVisited() zwraca tablicę odwiedzonych pól.
3. AnimatedDFS - moduł odpowiadający za animację algorytmu DFS.
    - solve(int startX, int startY) - zwraca listę punktów z krokami do animacji algorytmu DFS lub pustą listę, jeśli nie będzie rozwiązania.
    - dfs(int x, int y) - uzupełnia listę z krokami i sprawdza czy istnieje rozwiązanie.
    - isValid() - sprawdza czy dane współrzędne są w granicach labiryntu.
4. Screenshot - moduł umożliwiający zapisanie obrazu labiryntu jako pliku PNG.
    - saveComponentAsPNG(Component componen, String filename) - zapisuje obraz komonentu jako plik PNG.
5. BinHeader - moduł czytający nagłówki z pliku binarnego labiryntu i interpretuje jego strukturę. Jego metody, jedynie zwracają odczytane wartości z pliku binarnego.
6. BinKonwerter - moduł odpowiadający za konwersję wartości z pliku binarnego na znaki do pliku tekstowego.
7. BinInput - moduł konwertujący plik binarny na tekstowy.
8. NewPK - moduł odpowiedzialny za wybór nowych punktów startowego i końcowego.
    - startSelection() - odpowiada za czytanie myszki i wybór nowych punktów.

- clearOldPoints() - czyści stare punkty startowe i końcowe.

9. Rysowanie - moduł, który wizualizuje labirynt oraz działanie algorytmów przeszukiwania.
    - przekazaniepliku(String plik) - odczytuje labirynt z pliku i przygotowuje go do wyświetlenia.
    - setSolutionPath(List<Point> path) - ustawia i wyświetla ścieżkę rozwiązania na panelu.
    - setVisited(boolean[][] visited) - ustawia i wyświetla odwiedzone pola na panelu.
    - resetSolution() - resetuje ścieżkę rozwiązania i tablicę odwiedzonych pól.
    - stopAnimation() - zatrzymuje animację przeszukiwania.
    - clearPath() - czyści aktualnie wyświetlaną ścieżkę i przerywa animację.
    - setCzyKoniec(int value) - ustawia wartość wskazującą zakończenie animacji.
    - getCzyKoniec() - zwraca wartość wskazującą zakończenie animacji.
    - animateDFS(List<Point> path) - animuje kroki algorytmu DFS na panelu.
    - paintComponent(Graphics g) - rysuje labirynt, odwiedzone pola, animację oraz ścieżkę rozwiązania na panelu.
    - getKomorkax() - zwraca szerokość komórki labiryntu.
    - getKomorkay() - zwraca wysokość komórki labiryntu.
    - getStartX() - zwraca współrzędną X punktu startowego.
    - getStartY() - zwraca współrzędną Y punktu startowego.
    - getMaze() - zwraca labirynt jako listę znaków (do algorytmu BFS).
    - getMazeDFS() - zwraca labirynt jako dwuwymiarową tablicę znaków.

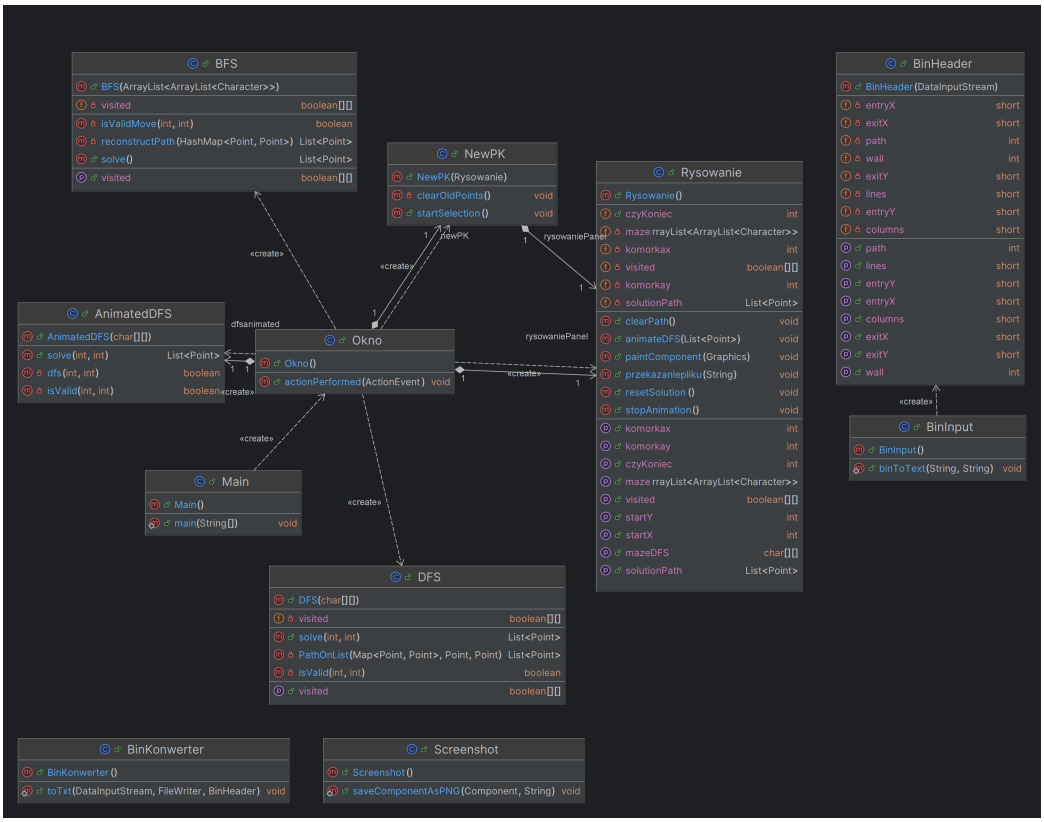
10. Okno - moduł odpowiadający za interfejs graficzny.
    - Okno() - konstruktor inicjalizujący interfejs użytkownika, ustawiający okno oraz komponenty.
    - actionPerformed(ActionEvent e) - obsługuje zdarzenia, takie jak kliknięcia w menu lub przycisku.

3.2 Wzorce projektowe

- Polecenie - każde działanie przycisku w klasie Okno można postrzegać jako polecenie. Kliknięcie przycisku powoduje wykonanie jakiejś akcji.
- Kompozyt - w kodzie używane są kontenery Swing (JPanel, JScrollPane, JMenuBar, JMenu) do organizowania i zarządzania komponentami interfejsu graficznego, co tworzy nam hierarchię obiektów.
- Obserwator - niektóre komponenty interfeju "obserwują" zdarzenia i reagują na nie poprzez dodanie ActionListener'a.

3.3 Diagram klas

Diagram klas został wygenerowany w programie Intellij IDEA.

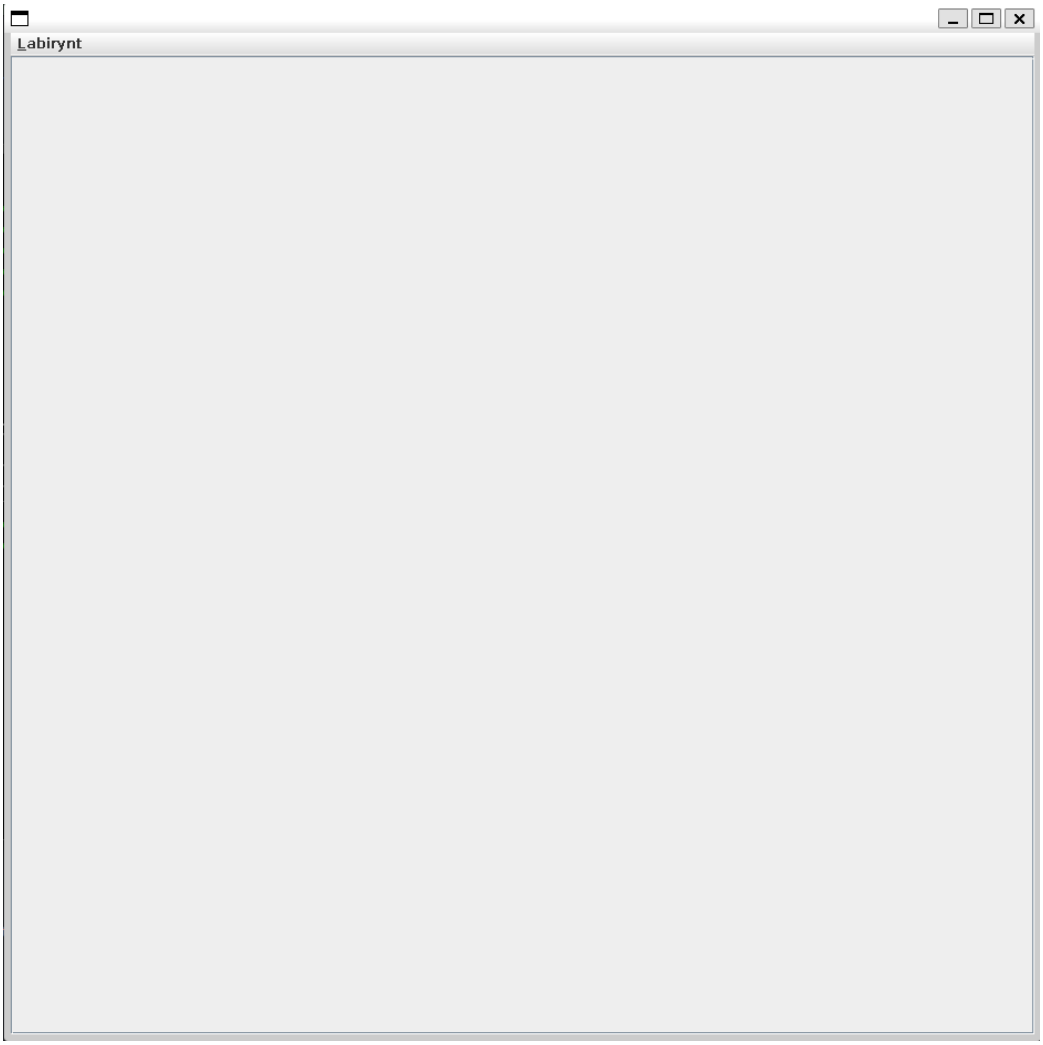


Rysunek 2: Diagram klas projektu

4 Instrukcja użytkownika

4.1 Obsługa programu

1. Po uruchomieniu programu zgodnie z instrukcjami z punktu 2. pojawi nam się takie okno:

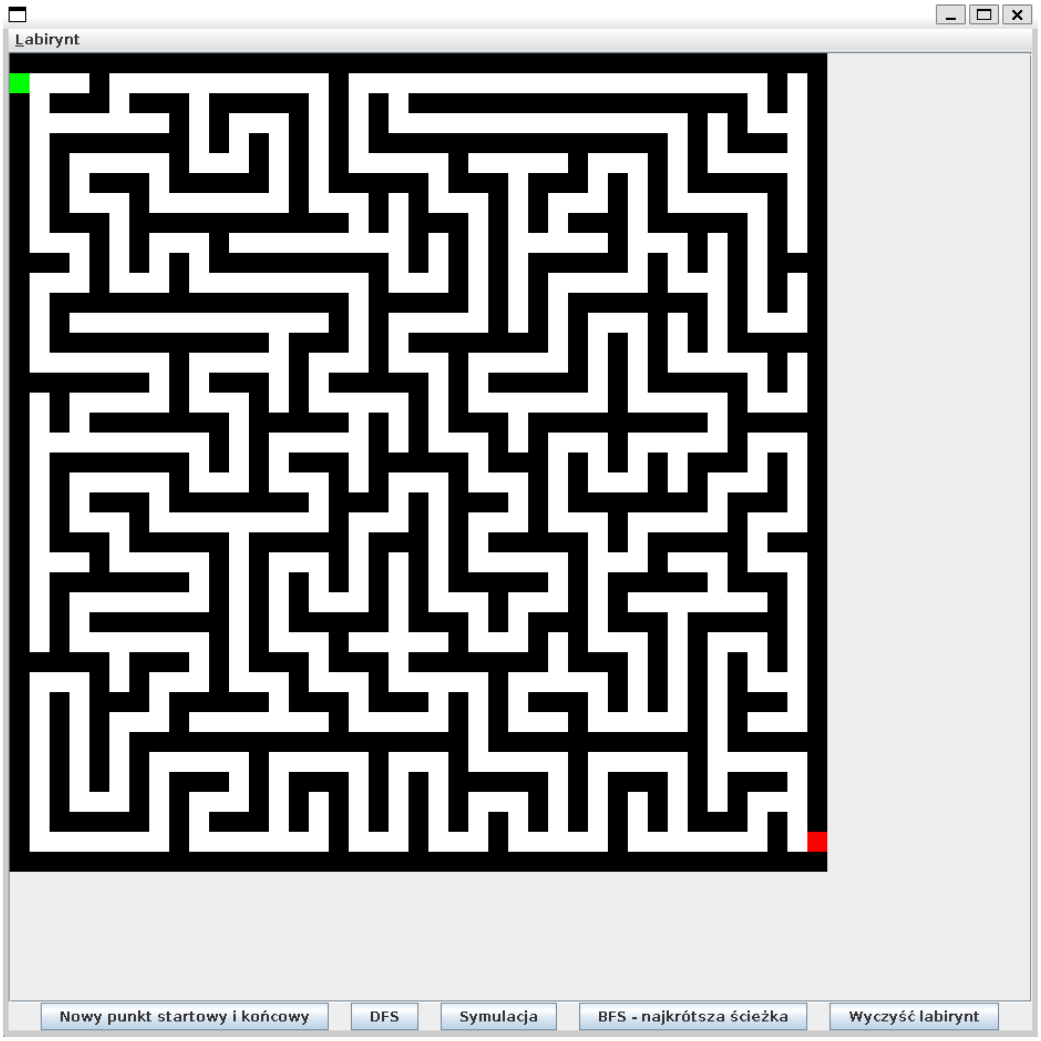


Rysunek 3: Okno startowe

2. W celu wczytania labiryntu wybieramy opcję z paska menu ”Labirynt” i klikamy odpowiednią opcję.

Rysunek 4: Pasek menu

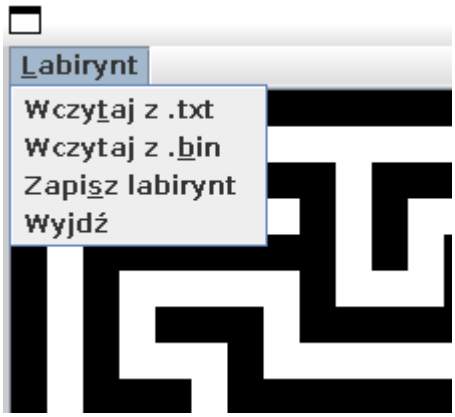
3. Po wczytaniu labiryntu, wyświetli nam się on na ekranie z dodatkowymi opcjami.



Rysunek 5: Wyświetlany labirynt

- ”Nowy punkt startowy i końcowy” pozwala użytkownikowi wybrać myszką nowe punkty startowe i końcowe. W przypadku wybrania ściany lub punktu poza labiryntem, program poinformuje nas o błędzie.
- Przyciski ”DFS” i ”BFS - najkrótsza ścieżka” wyświetlają rozwiązania wraz z odwiedzonymi polami przez dane algorytmy.
- ”Symulacja” tworzy animację i symuluje przechodzenie labiryntu przez użytkownika poprzez algorytm DFS.

- ”Wyczyść labirynt” po prostu czyści aktualnie wyświetlaną ścieżkę labiryntu.
- Dodatkowo klikając ”Labirynt” w pasku menu pojawi się nam nowa opcja ”Zapisz”, która zapisuje aktualnie wyświetlany stan labiryntu jako plik PNG.



Rysunek 6: Opcja ”Zapisz”

4.2 Wyjście

W celu wyjścia z aplikacji należy kliknąć przycisk ”X” zamknięcia okna lub ”Wyjdź”, które pojawia się po kliknięciu ”Labirynt” w pasku menu.

5 Testowanie

Do przetestowania kodu w środowisku Unixowym wykorzystaliśmy Linux Ubuntu i skrajne przypadki labiryntów, tzn. bardzo mały, bardzo duży, z kilkoma możliwymi rozwiązaniami oraz z pliku binarnego. Jeżeli pojawią się błędy, będziemy w stanie je zlokalizować dość szybko przez różnicę badanych plików. Testowe labirynty:

- smallmaze.txt - labirynt 10x10,
- maze.txt - labirynt 20x20,
- bigmaze.txt - labirynt 250x250,
- hugemaze.txt - labirynt 512x512,
- maze1024_1024.txt - labirynt 1024x1024,

- maze.bin - labirynt 256x256 zapisany w pliku binarnym,
- maze_cycles_5_5.txt - labirynt 5x5 z wieloma ścieżkami rozwiązania,
- maze_cycles_256_256.txt - labirynt 256x265 z wieloma ścieżkami rozwiązania,
- maze_cycles_512_512.txt - labirynt 512x512 z wieloma ścieżkami rozwiązania,
- maze_forks_5_5.txt - labirynt 5x5 z jednym rozwiązaniem z długą ścieżką,
- maze_forks_1023_1023.txt- labirynt 1023x1023 z jednym rozwiązaniem z długą ścieżką,
- maze_turns_256_255.txt- labirynt 256x256 z bardzo długą ścieżką rozwiązania.

6 Podsumowanie

Projekt pozwolił mi zapoznać się z biblioteką Swing oraz zdobyć doświadczenie przy tworzeniu interfejsu graficznego i animacji w Javie. Dodatkowo nauczyłem się również odczytywać pliki binarne w tym środowisku, co początkowo sprawiało mi problemy.
Największe trudności sprawiło mi stworzenie symulacji, czyli akcja dotycząca przycisku "Symulacja". Dużo czasu spędziłem nad animacją "cofania się" rysowanej ścieżki.
Podsumowując, podczas pracy nad projektem zdobyłem wiele praktycznych umiejętności w pisaniu programów w Javie. Nauczyłem się radzić sobie z różnymi problemami, co znacznie poszerzyło moje umiejętności programistyczne. Projekt ten był dla mnie cennym doświadczeniem, które z pewnością wykorzystam w przyszłych przedsięwzięciach programistycznych.