JIMP2 Sprawozdanie z projektu w jezyku Java

Łukasz Jarzecki, Paweł Skierkowski

Czerwiec 2024

Contents

1 Wstep

Celem projektu było stworzenie aplikacji okienkowej, która pozwala na wczytanie labiryntu oraz znalazienie najkrótszej ścieżki do wyjścia. Program został utworzony w jezyku Java z wykorzystaniem biblioteki Swing. Labirynt jest wczytywany do programu z pliku tekstowego lub binarnego oraz daje możliwość zapisu rozwiazania do pliku.

1.1 Problem zadania

Problemem zadania było zaimplementowanie przejrzystego interfejsu graficznego, który umożliwa prosta obsługe zawartych w programie funkcji. W odróżnieniu od projektu w jezyku C, tym razem nie obowiazywało nas ograniczenie czasowe oraz zużycia pamieci jednak algorytm miał znajdywać najkrótsza ścieżke przez co musieliśmy wykorzystać inny algorytm, który został opisany w poniższej sekcji.

1.2 Sposób rozwiazania

Implementacja projektu polegała na stworzeniu kodu realizujacego algorytm Dijkstry, który wykorzystujac graf znajduje odległości pomiedzy kolejnymi wezłami przez co po dojściu do ostatniego punktu (wyjścia) jest w stanie wyznaczyć najkrótsza ścieżke.

2 Implementacja

2.1 Modularność

Kluczowym aspektem realizacji projektu było rozdzielenie go na moduły, co umożliwiło efektywna prace i ułatwia zarzadzanie kodem. Dokonaliśmy podziału na nastepujace moduły:

- **Operacje na plikach i potrzebne funkcje** (Maze.java): pobiera danych o labiryncie z pliku, tworzenie struktury, implementacja funkcji wykorzystywanych w pozostałych modułach, zapis rozwiazania. Dekodowanie pliku binarnego.
- **Interfejs graficzny** (GUI.java): moduł ten odpowiada za wyświetlanie okienka, w którym za pomoca stworzonych funkcji możliwe jest graficzne wyświetlanie działania algorytmu, zwiera działanie programu w przejrzysty interfejs.
- **Algorytm i implementacja grafu** (MazeGraph.java): w module tym zaimplementowany jest algorytm, który odpowiada za główne działanie programu. Tworzy graf a nastepnie przechodzi po wezłach szukajac najkrótszej ścieżki.
- **Wezły i przechowanie danych** (Node.java): przechowuje dane o wezłach i krawedziach wychodzacych z nich.
- **Main** (Main.java): uruchomienie interfejsu graficznego.
- **Kierunek i odległość** (Edge.java):przechowuje dane o krawedziach, długość i wezeł docelowy.

2.2 Diagram klas

Diagram klas został wygenerowany przy użyciu InteliJ IDEA.

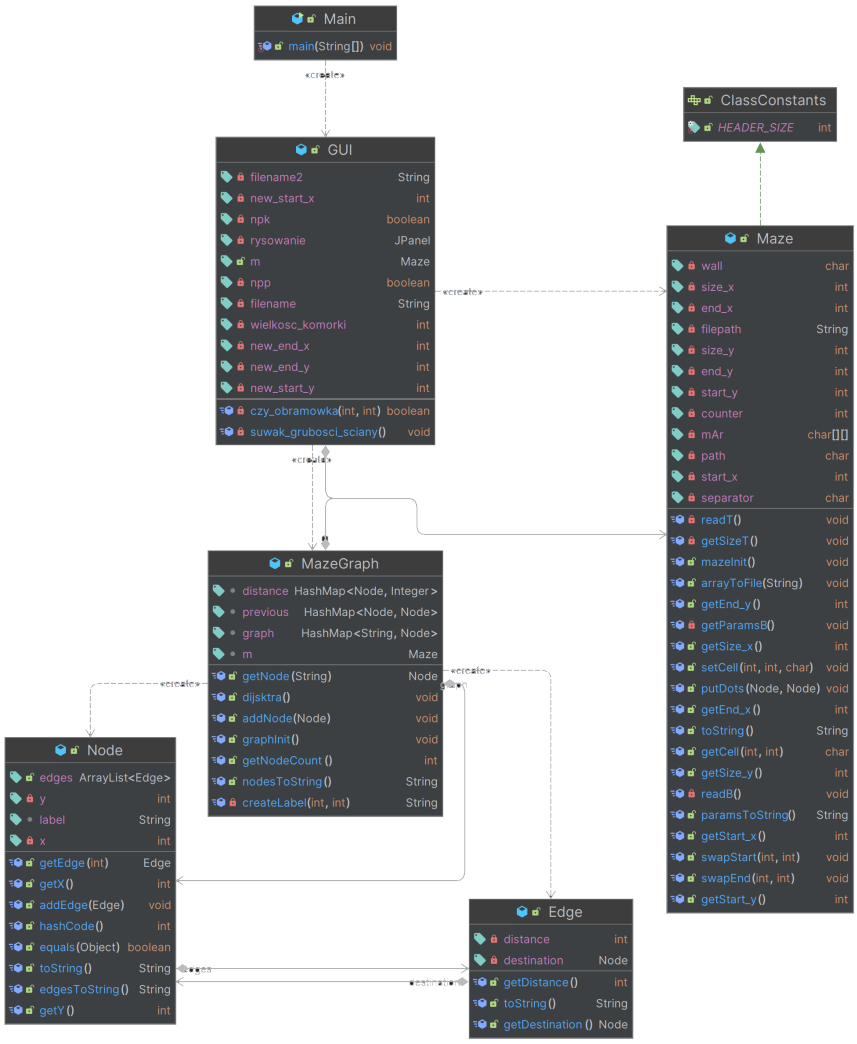

Figure 1: Diagram klas

2.3 Wzorce projektowe

Metoda wytwórcza - w klasie Maze w metodzie mazeInit() sprawdzane jest rozszerzenie pliku i w zależności od niego labirynt jest inicjowany w różny sposób.

Budowniczy - metody klasy Maze, takie jak getParamsB(), readB(), getSizeT(), readT(), krok po kroku odczytuja i konstruuja dane labiryntu.

Obserwator - w klasie Gui wykorzystywany jest ActionListener do reagowania na akcje użytkownika.

Kompozyt - w klasie MazeGraph labirynt jest reprezentowany za pomoca mapy wezłów, zawierajacych wychodzace z nich krawedzie, które z kolei zawieraja docelowe wezły. Każdy z tych obiektów może być traktowany indywidualnie, lub jako cześć wiekszej struktury.

3 Działanie programu

Klasa main uruchamia interfejs graficzny.

Figure 2: Main

W interfejsie zawarta jest zakładka menu umożliwiajacy otwarcie pliku, zapisanie, oraz zakładka narzedzia dzieki której możemy wybrać nowe punkty (poczatkowy i końcowy), znaleźć ścieżke i wybrać grubość ściany.

Figure 3: GUI

Dzieki algorytmowi w module MazeGraph przedstawionym we fragmencie poniżej mamy możliwość odczytania ścieżki.

```
while(!q.isEmpty()){
    Node n1 = q.poll();
    if(n1.equals(end)){
        System.out.println("Exit found, distance: " + distance.get(n1));
        break;
    }


    for(Edge e : n1.edges){
        Node n2 = e.getDestination();
        //System.out.println("Analyzing neighbour " + n2);
        int altDist = distance.get(n1) + e.getDistance();
        if(altDist < distance.get(n2)){
            previous.replace(n2, n1);
            distance.replace(n2, altDist);
            //System.out.println("New distance found: " + distance.get(n2));
            q.remove(n2);
            q.offer(n2);
        }

    }

}
```

Figure 4: Algorytm

Dalej mamy możliwość zmiany grubości ścian, oraz zapisania rozwiazania do pliku dzieki funkcji zaimplementowanej w module Maze.

```
public void arrayToFile(String filename) {  1 usage   Łukasz Jarzęcki +2

    File file = new File( parent: "solutions", child: "rozwiazanie_"+filename);
    FileWriter wr;

    if (!file.getParentFile().exists()) {
        file.getParentFile().mkdirs();
    }

    try {
        if (file.exists()) {
            file.delete();
        }
        file.createNewFile();
        wr = new FileWriter(file);

    } catch (IOException e) {
        throw new RuntimeException(e + " (Nie udało sie utworzyć pliku)");
    }
```

Figure 5: Enter Caption

3.1 Uruchamianie programu

Aby uruchomić program przez konsole należy poczatkowo go skompilować używajac komendy **javac .java**, na nastepnie wywołać stosujac **java Main**.

Obraz uruchomionej aplikacji prezentuje sie nastepujaco:

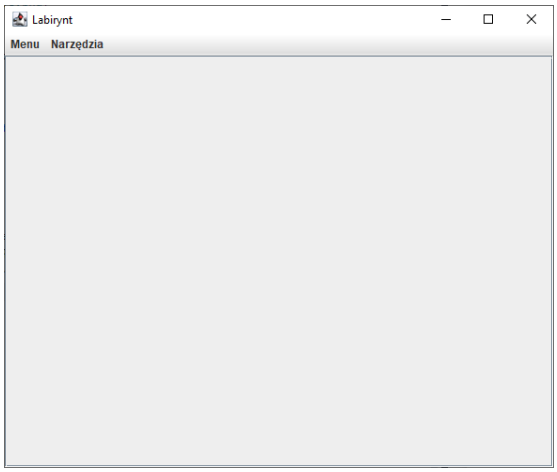

Figure 6: Uruchomione okno aplikacji

Po wczytaniu przykładowego labiryntu z pliku w zakładce Menu-¿Otwórz wyświetlony zostaje obraz labiryntu:

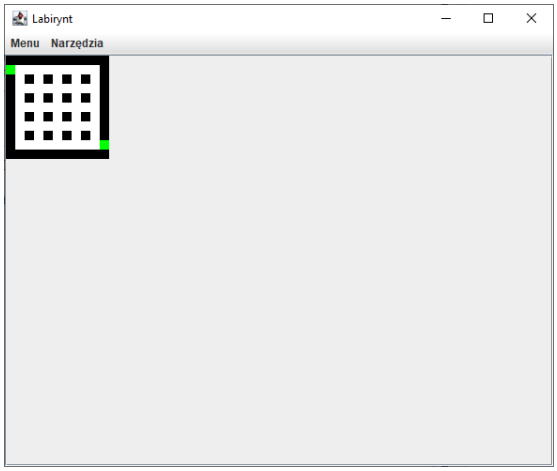

Figure 7: Przykładowy labirynt

Z zakładki narzedzia możemy wybrać opcje.

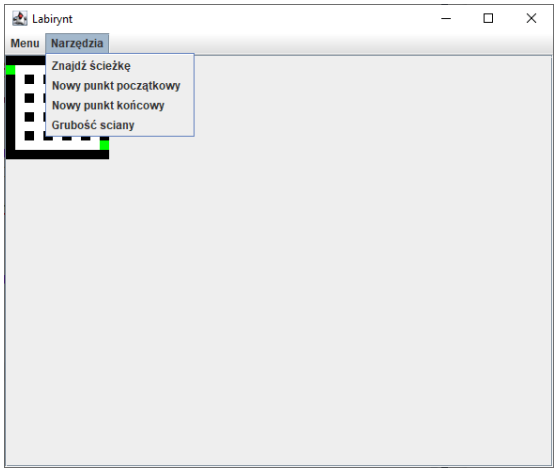

Figure 8: Narzedzia

Grubość ścian możemy dostosować wedle upodobania:

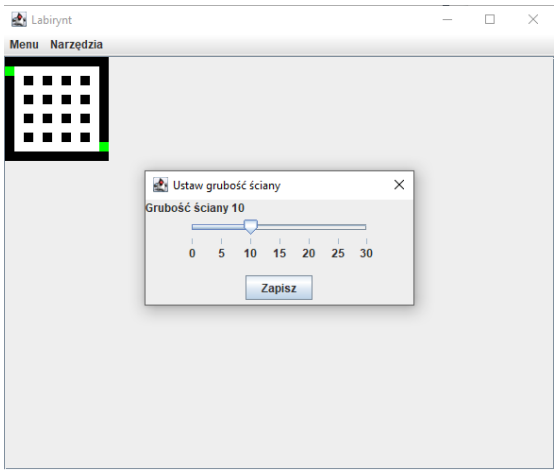

Figure 9: Grubość ścian

W pasku narzedzi bo wybraniu opcji **Nowy punkt poczatkowy** lub **Nowy punkt końcowy** możemy zmienić wejście i wyjście z labiryntu. Aby to zrobić po naciśnieciu przycisku należy nacisnac LPM na wybrany kwadrat (cześć ściany) która znajduje sie na obramowaniu (nie możemy wybrać wyjścia lub wejścia w środku labiryntu).

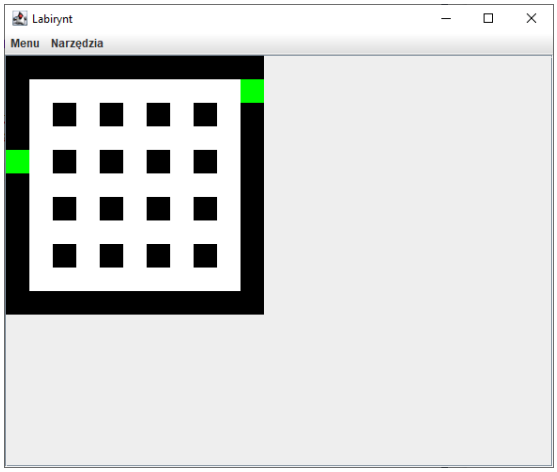

Figure 10: NPP i NPK

Przechodzac do najważniejszego punktu w pasku narzedzi znajduje sie opcja **Znajdź ścieżke**, po naciśnieciu której otrzymujemy obraz ścieżki wyjścia z labiryntu.

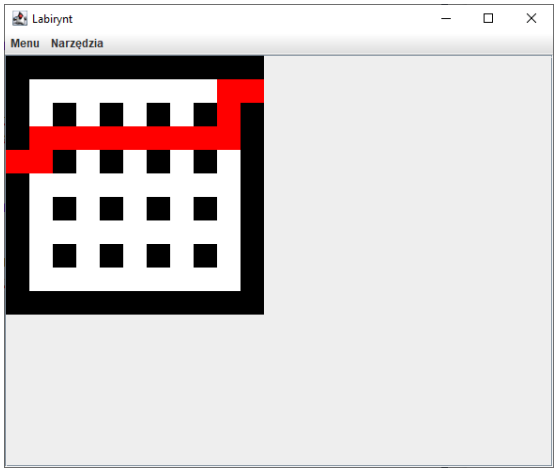

Figure 11: Znaleziona droga

Ostatnia już właściwościa jest możliwośc zapisu wyświetlanego labiryntu do folderu solutions.

Figure 12: Zapisane pliki

4 Wnioski

Realizacja tego projektu pozwoliła nam na zapoznanie sie ze środowiskiem Javy oraz biblioteki Swing. Poszerzyliśmy dalej nasza umiejetność pracy zespołowej przez dobry podział obowiazków i organizacje czasu. Mogliśmy dzieki temu niezależnie pracować nad projektem w tym samym czasie dodajac nowe funkcjonalności równolegle.

Projekt był wymagajacy pod katem implementacji nowego algorytmu, którego nie wykorzystywaliśmy w innych projektach oraz obsługa bilbioteki Swing. To co podobało nam sie bardziej, niż w przypadku projektu w C to efekt końcowy który jest schludny i estetyczny.

Po ukończeniu pracy jestem w stanie stwierdzić że praca nad tym projektem dodała nam obu wiele doświadczenia w zakresie pracy w jezyku Java. Projekt ten daje nam możliwość bycia kreatywnym przy pracy nad kolejnymi nowymi zadaniami, zarówno tymi które znamy jak i nowymi.