JIMP 2 Sprawozdanie z części projektu wykonanej przy pomocy języka Java

Zespół 7

Spis treści

Wstęp

1.Opis programu

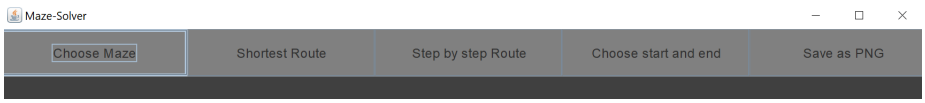
Celem programu stworzonego na potrzeby projektu, napisanego w języku Java wraz z użyciem biblioteki Swing, w której wykonany został interfejs graficzny, było znalezienie drogi przez labirynt o maksymalnych rozmiarach 1024x1024 komórek, liczonych po ścieżkach, po których można się poruszać. Wynikiem działania programu powinno być graficzne ukazanie szukanej ścieżki.

2.Zasadnicza Trudność Zadania

Główną trudnością związaną z tworzeniem programu było stworzenie poprawnie działającego interfejsu graficznego dającego możliwość zobaczenia najkrótszej ścieżki w labiryncie oraz ścieżki krok po kroku, a także algorytmu umożliwiającego zapisanie efektu działania programu w formie zdjęcia.

3.Obsługa programu przez użytkownika

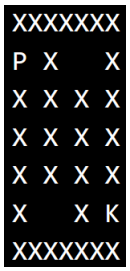
W repozytorium umieszczony jest plik w formacie .jar, Użytkownik programu musi jedynie pobrać go a następnie uruchomić.



W aplikacji Użytkownik zobaczyć może 5 przycisków dających dostęp do opcji. Pierwszy z nich, opatrzony treścią "Choose Maze", pozwala na uruchomienie eksploratora plików, oraz wybranie pliku w którym zapisany jest labirynt.Plik taki można wygenerować za pomocą strony internetowej będącej narzędziem dostarczonym na potrzeby projektu, lub skorzystać można z przykładowych labiryntów zawartych w Repozytorium. Plik ten może być formatu .txt. W takim pliku przyjęte zostały następujące oznaczenia:

- P – początek labiryntu (wejście)
- K – koniec labiryntu (wyjście)
- X – ściana
- Spacja – miejsce, po którym jest możliwość poruszania się.

Przykładowy wygląd pliku .txt prezentuje się w następujący sposób:



Plik może być również formatu .bin. Taki plik opisany jest w następujący sposób:

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.

Kolejny przycisk, opatrzony napisem "Shortest Route", wywołuje algorytm BFS (breadth-first search) w celu utworzenia najkrótszej drogi punktu początkowego do końcowego labiryntu.

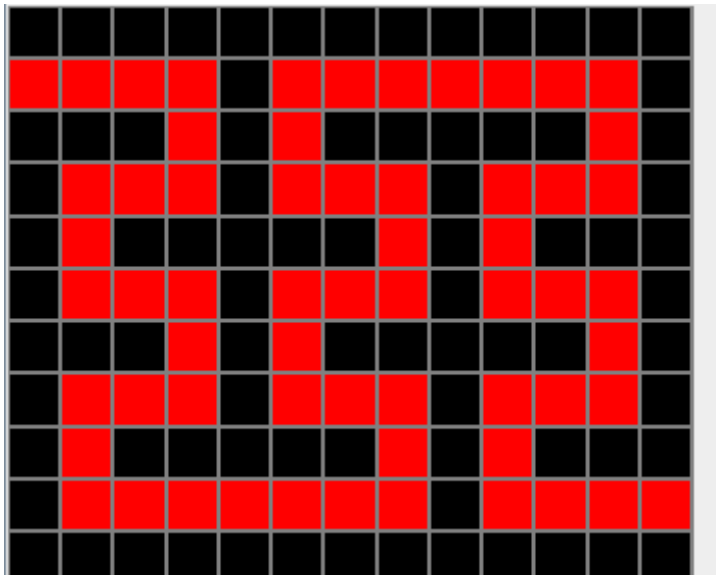
Trzeci z przycisków, na którym zobaczyć możemy "Step by step Route", wywołuje algorytm prawej ręki tworzenie ścieżki przez którego obserwować możemy w czasie rzeczywistym. Przez specyfikę algorytmu działa on poprawnie jedynie gdy wczytany labirynt nie ma ścian odłączonych od reszty.

Czwarty przycisk "Choose start and end" pozwala na wybranie nowych punktów początkowego i końcowego. Użytkownik robi to poprzez naciśnięcie przycisku, a następnie naciśnięcie wybranych przez siebie punktów w labiryncie, pierwszy z nich zostanie nowym punktem początkowym a drugi nowym punktem końcowym.

Ostatni z przycisków, "Save as PNG" pozwala na zapisanie stanu labiryntu jako zdjęcia w formacie png.

4. Wyjście programu

Wynikiem działania programu jest graficzne przedstawienie ścieżki pozwalającej wydostać się z (za)danego labiryntu. Przykład zapisania rozwiązanego labiryntu jako zdjęcia znajduje się poniżej:



Sposób Rozwiązania

1.Wybrane do rozwiązania algorytmy oraz opisy ich działania

Algorytmem który zdecydowaliśmy się wykorzystać do znalezienia najkrótszej drogi przez labirynt jest algorytm BFS, który został zaimplementowany w wersji Parallel BFS w celu zwiększenia jego wydajności czasowej. Po wczytaniu labiryntu za pomocą przycisku "Choose Maze" wyświetlony zostanie wczytany labirynt. Następnie po kliknięciu przycisku "Find Shortest Route" wywołany zostanie algorytm BFS. Gdy natrafi on na pole oznaczające wyjście z labiryntu dochodzi do wywołania metody służącej do wyznaczenia od końca trasy prowadzącej przez labirynt. Zaczyna się ona w polu sąsiadującym z końcem labiryntu, i następnie przechodzi do sąsiadującego pola o wartości o 1 mniejszej niż jej własna, aż do napotkania wartości 1 oznaczającej wejście do labiryntu. Następnie ponownie wywołana zostaje metoda rysująca Labirynt. Użytkownik jeśli chce może skorzystać z przycisku "Save as PNG" aby zapisać labirynt z zaznaczoną trasą. Te same kroki wykonywane są na pliku binarnym, po przepisaniu go na plik tekstowy.

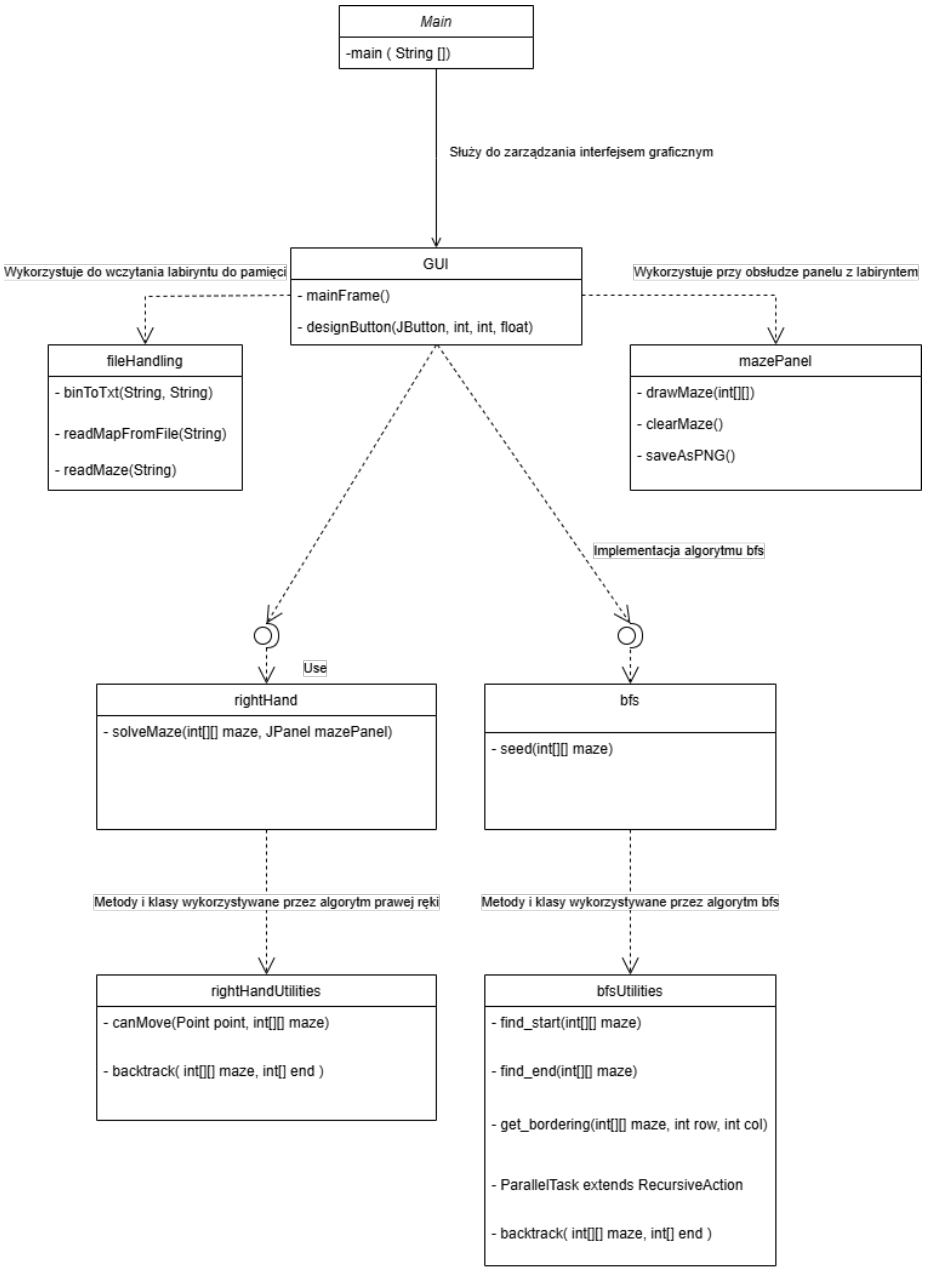
Algorytm pozwalający na obejrzenie tworzenia trasy to algorytm prawej ręki. Po wywołaniu go na wczytanym labiryncie przy pomocy przycisku "Step by step Route" rozpocznie on przechodzenie przez labirynt tak, by przy poruszaniu się zawsze mieć ścianę po prawej stronie poruszającego się. Z tego powodu nie stworzy on działającej pracy jeśli w labiryncie pojawiają się niepołączone ze sobą ściany. W takim wypadku wpadnie on w cykl krążenia wzdłuż odciętej ściany przy której rozpoczął poruszanie się.

2.Podział Kodu

Kod podzielony został na klasy:

A) bfs.java służąca jako interfejs bfsUtilities.java używanek do obsługi algorytmu BFS
B) rightHand.java służąca jako interfejs rightHandUtilities.java używanej do obsługi algorytmu prawej ręki
C) fileHandling.java odpowiadającą za obsługę plików z labiryntami
D) gui.java odpowiadającą za interfejs graficzny użytkownika
E) mazePanel.java odpowiadającą za obsługę wyświetlania oraz działania labiryntu od strony powłoki graficznej

Relacje między którymi zobaczyć można na zamieszonym na kolejnej stronie diagramie Klas.



Wykorzystane Wzorce Projektowe

Singleton: Wzorzec ten jest wykorzystywany w klasie bfs w kontekście klasy ForkJoinPool, która jest tworzona tylko raz i używana przez wszystkie wątki w aplikacji.

Strategia: Wzorzec ten jest wykorzystywany w klasie rightHand, gdzie algorytm ręki prawej jest implementowany jako osobna strategia rozwiązania labiryntu.

Fasada: Klasa gui funkcjonuje jako fasada, np. metoda mainFrame(), zarządza interakcjami z interfejsem użytkownika i wywołuje różne funkcje w innych klasach.

Fabryka Abstrakcyjna: Klasa obsługujące odczyt labiryntu z pliku, wykonuje różne metody zależnie od otrzymanych przez siebie danych, jednak tworząc odpowiedni obiekt na podstawie typu pliku wejściowego.

Obserwator: W klasie rightHand mechanizm SwingWorker jest wykorzystywany aby aktualizacje stanu były przekazywane do interfejsu użytkownika i odświeżane w czasie rzeczywistym.

Wnioski

Dzięki projektowi mogliśmy w praktyce zastosować język Java którego uczyliśmy się w aktualnym semestrze. Była to również okazja zapoznania się z tworzeniem interfejsów graficznych, czego nie mieliśmy okazji robić wcześniej. Zauważyć mogliśmy również różnicę w pracy przy użyciu języka C, oraz mającego wiele więcej udogodnień języka Java.