Opis Projektu "Labirynt"

(link do repozytorium github:
`https://github.com/AleksandraGrabowska04/labirynt.git`)

20 stycznia 2025

1 Cel projektu

Celem projektu "Labirynt"jest sprawdzenie, jak różne algorytmy radzą sobie z rozwiązywaniem labiryntów. Projekt umożliwia użytkownikowi nie tylko generowanie i wizualizację labiryntów, ale także przeprowadzenie testów porównujących algorytmy takie jak DFS, BFS, Dijkstra i A*. Wyniki analiz są przedstawiane w formie wykresów, co pozwala lepiej zrozumieć różnice w ich działaniu pod względem wydajności oraz liczby wykonanych kroków potrzebnych do rozwiązania labiryntu.

2 Opis projektu

Projekt "Labirynt"to aplikacja, która pozwala na tworzenie, rozwiązywanie i analizowanie labiryntów za pomocą różnych algorytmów. Dzięki niej można generować labirynty o rozmiarach $n \times n$ (gdzie n to liczba pól ściany labiryntu), rozwiązywać je automatycznie, a następnie wizualizować działanie algorytmów krok po kroku. Wyniki działania algorytmów są porównywane i prezentowane w formie wykresów, co ułatwia ich zrozumienie i ocenę.

2.1 Funkcjonalności projektu

1. **Generowanie labiryntów**
   - Możliwość tworzenia labiryntów o różnych wymiarach $n \times n$.
   - Labirynty są generowane w postaci macierzy (pól) zer i jedynek, a następnie przekształcane na grafy, w których pola to węzły, a ścieżki to połączenia między nimi.
   - Labirynt zawsze ma jedno wejście (w lewym górnym rogu) i jedno wyjście (w prawym dolnym rogu).
2. **Rozwiązywanie labiryntów za pomocą algorytmów przeszukiwania**
   - Zaimplementowano cztery algorytmy:
     - **DFS (Depth-First Search)** – algorytm eksploruje jak najdalsze ścieżki, cofając się tylko w razie potrzeby. Jest szybki, ale nie zawsze znajduje najkrótszą ścieżkę.
     - **BFS (Breadth-First Search)** – algorytm bada wszystkie możliwe ścieżki równolegle, co gwarantuje znalezienie najkrótszej drogi, ale jest bardziej czasochłonny.
     - **Dijkstra** – algorytm przeszukujący grafy w sposób optymalny, uwzględniając koszt przejścia między węzłami (przydatny w przypadku labiryntów z różnymi wagami krawędzi).

- **A*** – algorytm heurystyczny, który stara się "przewidzieć", którędy idzie ścieżka do wyjścia na podstawie dostępnych dla niego danych.
- Użytkownik może zobaczyć, jak każdy algorytm przeszukuje labirynt i jaką ścieżkę wybiera.

3. **Graficzna wizualizacja**
   - Labirynty są rysowane w formie macierzy 2D, gdzie:
     - 1 oznacza ścianę, gdzie cały labirynt jest otoczony ścianami,
     - 0 oznacza wolne pole (ścieżka).
   - Proces przeszukiwania labiryntu można śledzić krok po kroku, co pozwala zobaczyć, jak algorytm podejmuje decyzje.
4. **Porównanie algorytmów**
   - Po rozwiązaniu labiryntu projekt generuje pliki, które zawierają informacje takie jak:
     - Liczba kroków wykonanych przez każdy algorytm.
     - Odwiedzone węzły.
     - Ścieżka rozwiązania labiryntu.
   - Następnie wyniki są prezentowane na wykresach (np. słupkowych lub liniowych), co ułatwia porównanie.

2.2 Technologie użyte w projekcie

- **C/C++**
  - Wykorzystywane do podstawowej logiki projektu, takiej jak:
    - Generowanie labiryntów w postaci macierzy $n \times n$.
    - Implementacja algorytmów przeszukiwania.
    - Zamiana macierzy labiryntu na postać grafu za pomocą struktury danych.
- **Python**
  - Używany do:
    - Generowania wykresów porównujących wyniki algorytmów za pomocą biblioteki Matplotlib.
    - Obsługi analizy danych i ich wizualizacji.
- **Java**
  - Stosowana do:
    - Rysowania labiryntów z graficznym interfejsem użytkownika za pomocą Java2D.
    - Wizualizacji kroków przeszukiwania w czasie rzeczywistym.
- **Lua i XMake**
  - **Lua**: Język skryptowy używany do definiowania konfiguracji w pliku `xmake.lua`.
  - **XMake**: Narzędzie do automatyzacji budowania projektu. W projekcie służy do:
    - Konfigurowania i zarządzania procesem kompilacji aplikacji.
    - Obsługi zależności projektu.
    - Ułatwienia tworzenia wieloplatformowego kodu źródłowego (np. na Windows, Linux).

    Skrypty w Lua umożliwiają prostą i czytelną definicję procesu budowania, co przyspiesza zarządzanie kompilacją w porównaniu z bardziej złożonymi narzędziami, takimi jak CMake.

3 Członkowie zespołu

W skład zespołu pracującego nad projektem wchodzą:

- Jakub Malinowski (lider zespołu)
  - Nr indeksu: 288561
  - Nazwa użytkownika na platformie GitHub: `at-eee`
- Aleksandra Grabowska
  - Nr indeksu: 288559
  - Nazwa użytkownika na platformie GitHub: `AleksandraGrabowska04`
- Jakub Markowski
  - Nr indeksu: 285709
  - Nazwa użytkownika na platformie GitHub: `kuba913`
- Krystian Dzikiewicz
  - Nr indeksu: 289935
  - Nazwa użytkownika na platformie GitHub: `LionDoge`

4 Podział pracy

(Pełen podział pracy zmieniający się na przestrzeni czasu można zobaczyć wewnątrz pliku „*podzial_prac.md*" na stronie repozytorium projektu.)

- Jakub Malinowski (at-eee):
  - Koordynowanie projektem,
  - Generowanie losowych labiryntów (w postaci macierzy),
  - Zaprogramowanie algorytmu A* (A Gwiazdka) do rozwiązywania labiryntów.
- Aleksandra Grabowska (AleksandraGrabowska04):
  - Zaprogramowanie algorytmu Dijkstry do rozwiązywania labiryntów,
  - Generowanie wykresów porównujących skuteczność wybranych algorytmów.
- Jakub Markowski (kuba913):
  - Graficzna reprezentacja labiryntu oraz demonstracja kroków wykonywanych przez algorytmy podczas obierania przez nich prawidłowych ścieżek szukających wyjścia z labiryntu.
- Krystian Dzikiewicz (LionDoge):
  - Opracowanie struktury danych przechowującej labirynt w postaci grafu,
  - Zaprogramowanie algorytmu BFS i DFS do rozwiązywania labiryntów,
  - Przygotowanie testów jednostkowych.

5 Instrukcja użytkowania

5.1 Wymagania wstępne

Program działa zarówno na systemie *Microsoft Windows* jak i *Linux*, jednak do użytkowania zalecane jest korzystanie z systemu *Linux* w dystrybucji *Debian* (np. *Ubuntu*).

Do uruchomienia programu jest wymagane następujące oprogramowanie/narzędzia:

- xmake
- gcc i g++
- java
- python3
- matplotlib do python'a

Do ich zainstalowania na systemie Linux w dystrybucji debian (zalecane, ponieważ na nim głównie były wykonywane prace) możemy użyć w terminalu polecenia:

```
sudo apt-get install xmake gcc g++ java python3
```

A następnie polecenia:

```
pip install matplotlib
```

5.2 Uruchomienie programu

Następnie aby sklonować repozytorium (z: `https://github.com/AleksandraGrabowska04/labirynt.git`) wpisujemy w terminalu następujące polecenie:

```
git clone https://github.com/AleksandraGrabowska04/labirynt.git
```

Potem aby uruchomić program:

```
cd labirynt
xmake build
```

A następnie:

```
xmake run labyrinth [rozmiar(y) labiryntu/ow] [(opcjonalnie) nazwa docelowego
    folderu dla wynikow]
```

Na przykład:

```
xmake run labyrinth 30 60 120 240
```

5.3 Uruchomienie reprezentacji graficznej (opcjonalne)

Aby uruchomić graficzną reprezentację labiryntu i ścieżek rozwiązywanych przez algorytmy (musimy znajdować się w nadkatalogu katalogu projektowego (*labirynt*)):

```
java -cp './labirynt/target/classes' main/Main
```

5.3.1 Obsługa okna programu do reprezentacji graficznej

- *strzałka w prawo* - Przechodzi do graficznej reprezentacji następnego algorytmu.
- *strzałka w lewo* - Wraca do graficznej reprezentacji poprzedniego algorytmu.
- *spacja*:
  - *wciśnięta jednokrotnie* - odtworzenie przebiegu ściezki obieranej przez algorytm rozwiązujący labirynt.
  - *wciśnięta jednokrotnie* - Przewinięcie do końca odtwarzania.
  - *wciśnięta trzykrotnie* - powrót do stanu początkowego.

5.4 Utworzenie i uruchomienie wykresów porównujących (opcjonalne)

Aby opcjonalnie uruchomić wykresy porównujące skuteczności algorytmów (musimy znajdować się w podkatalogu *src*):

```
python3 generate_plots.py
```

5.4.1 Obsługa okien wykresów porównujących

Obsługa okien jest intuicyjna i odbywa się za pomocą panelu użytkownika narzędzia *matplotlib* znajdującego się w każdym z okienek (instrukcja obsługiwania nawigacji okien w matplotlib: `https://matplotlib.org/3.2.2/users/navigation_toolbar.html`).

6 Przykładowe uruchomienie z wykresami

Przykładowa komenda:

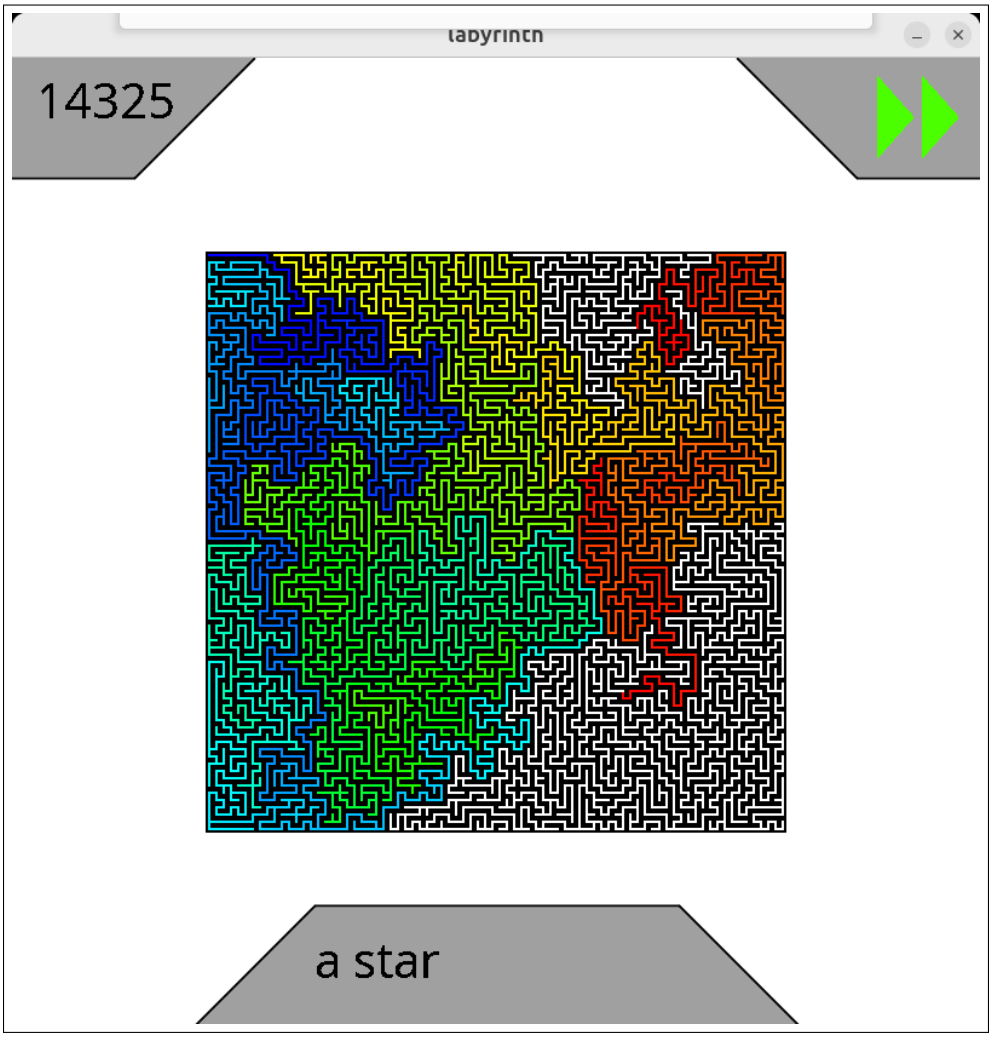
```
xmake run labyrinth 30 60 120 240
```

Listing 1: Uruchamianie programu z różnymi rozmiarami labiryntów

Uruchamia program, który generuje labirynty o wymiarach podanych w komendzie:

- Labirynt o wymiarach 30×30,
- Labirynt o wymiarach 60×60,
- Labirynt o wymiarach 120×120,
- Labirynt o wymiarach 240×240.

Ostatni podany rozmiar, w tym przypadku 240×240, zostanie użyty do wizualnej prezentacji w Java. W prezentacji można zobaczyć, jak każdy algorytm (BFS, DFS, Dijkstra, A*) krok po kroku rozwiązuje labirynt. Wizualizacja pokazuje wybierane ścieżki, zaznaczając proces przeszukiwania.



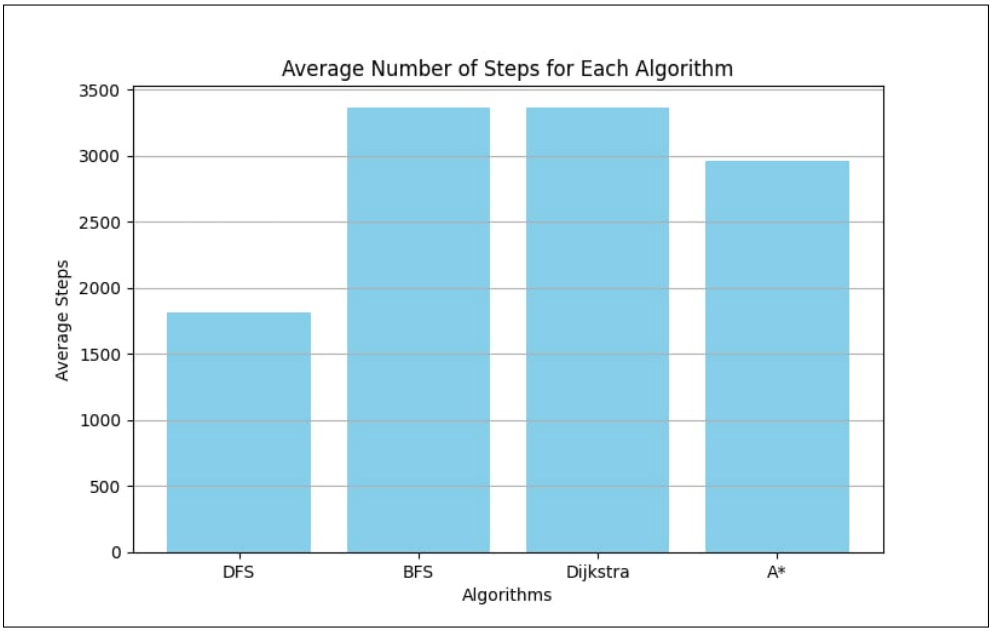
Rysunek 1: Przykładowa graficzna reprezentacja labiryntu wraz z szukaniem przez algorytm ścieżek wyjścia dla algorytmu A* w java2D. W lewym górnym rogu numer obecnie obecnie wykonywanego kroku.

Po zakończeniu działania programu można użyć następującej komendy, aby wygenerować wykresy porównujące algorytmy:

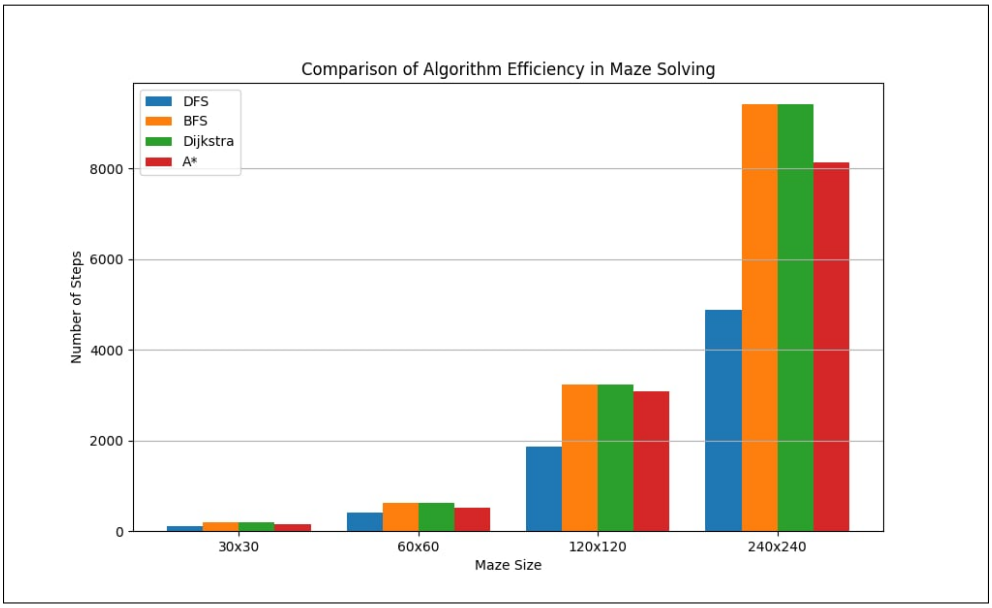
```
python3 generate_plots.py
```

Listing 2: Generowanie wykresów

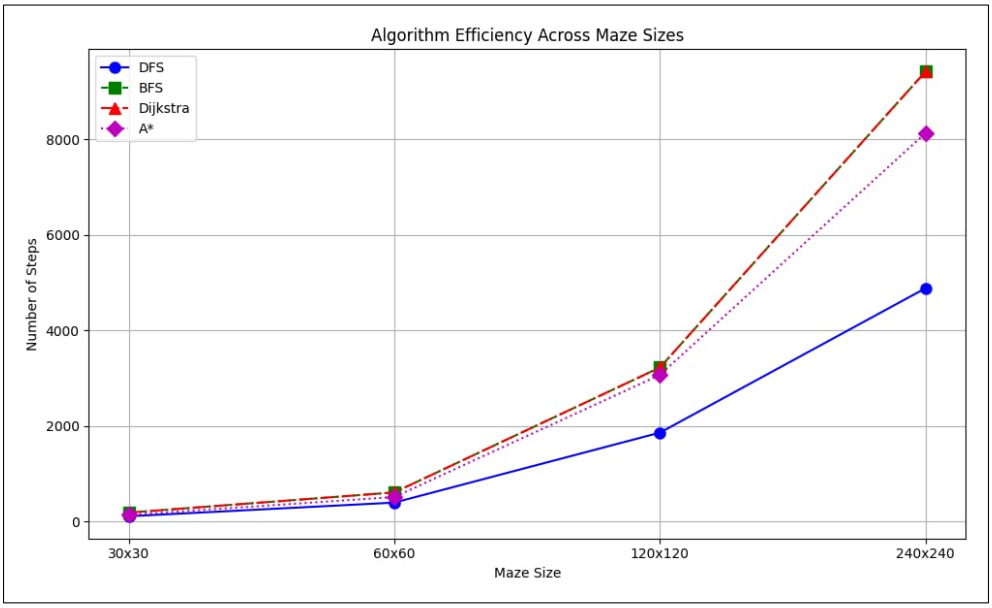
Komenda ta tworzy wykresy, które przedstawiają porównanie skuteczności algorytmów BFS, DFS, Dijkstra oraz A*.



Rysunek 2: Średnia liczba kroków algorytmów BFS, DFS, Dijkstra i A*.



Rysunek 3: Porównanie skuteczności algorytmów w rozwiązywaniu labiryntu na wykresach słupkowych.



Rysunek 4: Porównanie skuteczności algorytmów w rozwiązywaniu labiryntu na wykresie liniowym.