Dokumentacja Końcowa Projektu Labirynt

Polina Nesterova i Martyna Kochalska

22 kwietnia 2024

Streszczenie

Ten dokument jest sprawozdaniem z postępów prac nad projektem labiryntu w języku C. Zawiera informacje o głównym celu projektu, szczegółowy opis struktury katalogów, diagram modułów oraz prezentację wyników uzyskanych podczas uruchamiania programu. Dodatkowo, dokonano podsumowania całego projektu oraz analizy współpracy.

Contents

1 Cel projektu

Celem projektu jest opracowanie programu, który umożliwi znalezienie drogi przez wczytany labirynt. Labirynt będzie reprezentowany przez plik tekstowy, który zawiera definicje punktów, takich jak punkt wejścia (oznaczony jako "P"), punkt wyjścia (oznaczony jako "K"), ściany (oznaczone jako "X") oraz miejsca, po których można się poruszać (oznaczone jako spacje). Przykładowy format labiryntu 5 x 4:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
P X       X             X       X               X
X X XXX X XXX XXX X XXX X X X XXX X X XXX
X   X X   X X     X     X   X   X     X X   X
X X X X XXX X XXXXXXXXXXXXXXXX XXXXXXXXX X
X X   X     X                             X X
X XXX X XXX X XXXXXXX X XXXXX XXXXXXX X X
X X               X   X X   X       X   X X X
X X XXXXX X XXX XXX X XXXXX XXX X XXX X X
X         X                       X   X   K
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Wymagania dotyczące rozmiaru labiryntu

Maksymalny rozmiar labiryntu, który może być przekazany do rozwiązania, to 1024 x 1024 liczony po ścieżkach, po których można się poruszać.

Wyjście z programu

Program powinien generować listę wykonanych kroków, która pokaże drogę przez labirynt. Przykładowa lista kroków może wyglądać następująco:

- START
- FORWARD 1
- TURNLEFT
- FORWARD 4
- TURNRIGHT
- FORWARD 3
- STOP

Ograniczenia dotyczące pamięci

Program napisany w języku C nie powinien używać więcej pamięci niż 512 kB w czasie całego swojego działania. Ważne jest również odpowiednie zarządzanie pamięcią w programie, aby uniknąć wycieków, nadmiernego zapisywania do pamięci oraz wielokrotnego zwalniania pamieci.

Wsparcie dla środowiska Linux

Program musi być zdolny do uruchomienia w środowisku Linux. Weryfikacja programów będzie odbywać się podczas ich obrony na maszynach uczelnianych.

Obsługa wczytywania i zapisywania labiryntu w postaci pliku binarnego

Program musi umożliwiać wczytanie i zapisanie labiryntu w postaci pliku binarnego.

Załączony plik Makefile

Do projektu należy dołączyć plik Makefile, który pozwoli na kompilację programu.

Opracowanie programu spełniającego powyższe cele będzie kluczowe dla sukcesu projektu. Wszelkie dodatkowe funkcje lub ulepszenia będą mile widziane, ale należy najpierw zapewnić, że program spełnia podstawowe wymagania.

2 Struktura programu

2.1 Struktura folderów

W katalogu projektu znajdują się wszystkie pliki z kodem źródłowym oraz cztery dodatkowe foldery:

- dokumentacja
- default_maps
  - Folder zawierający pliki z labiryntami. Te pliki będą wykorzystywane do testowania programu i mogą być wczytywane przez użytkownika w celu przetestowania działania programu.
- chunks
  - W tym folderze znajdują się chunki, na które dzieli się labirynt. Chunki te mogą być wykorzystywane w różnych fazach działania programu, np. w procesie przetwarzania labiryntu.
- exe

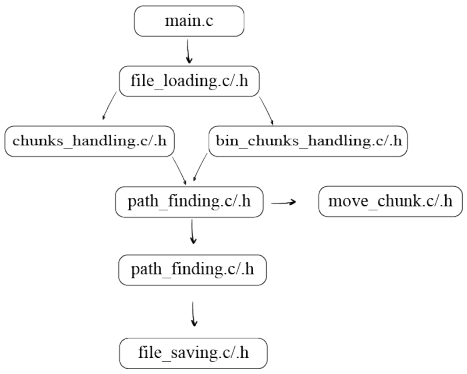
  - Jest to folder, w którym znajduje się wykonywalna wersja programu. Po zbudowaniu programu, użytkownik może uruchomić go z poziomu tego folderu.

- tmp
  - Jest to folder, w którym znajdują się pliki tymczasowe niezbędne dla programu

- results
  - Jest to folder, w którym znajdują się wyniki działania programu. Są to pliki przechowujące listę kroków, których można użyć do przejścia labiryntu.

2.2 Diagram modułów

Projekt Labirynt składa się z modułów pokazanych na obrazku. Każdy z nich składa się z pliku nagłówkowego oraz pliku z kodem źródłowym. Posiada również główny moduł sterujący działaniem programu - main.c.



3 Opis plików wykorzystywanych przez program

- `Pliki Wejściowe` – program korzysta z plików zawierających informacje o labiryncie. Możliwe są dwa rodzaje plików wejściowych: tekstowy (z rozszerzeniem '.txt') oraz binarny (.bin).

  Plik tekstowy zapisany powinien być za pomocą znaków 'X', ' ', 'P' i 'K', gdzie znaki te oznaczają kolejno ścianę, przejście, początek i koniec labiryntu.

  Plik binarny składa się z 4 głównych sekcji:

  1. Nagłówka pliku
  2. Sekcji kodująca zawierająca powtarzające się słowa kodowe
  3. Nagłówka sekcji rozwiązania
  4. Sekcji rozwiązania zawierające powtarzające się kroki które należy wykonać aby wyjść z labiryntu

  Dokładny opis pliku binarnego jest możliwy do pobrania tutaj: https://isod.ee.pw.edu.pl/

- `Pliki Wyjściowe` – program generuje plik wyjściowy, w którym zapisywane są wyniki działania programu. Taki plik jest opcjonalny i jego nazwa może być podana przez użytkownika jako argument wywołania.

- `Pozostałe pliki` - Program tworzy pliki półtymczasowe przechowywujące podzielone fragmenty labiryntu w formie plików tekstowych. Pliki te przechowywane są w podfolderze projektu /chunks, który to jest czyszczony przy każdorazowym uruchomieniu programu. Dodatkowo w tym folderze tworzy pojedyczny plik path.txt, pełniący funkcję stosu dla ścieżek.

4 Kompilacja programu

Program jest kompilowany za pomocą pliku Makefile, który definiuje reguły kompilacji dla poszczególnych plików źródłowych oraz sposób łączenia ich w końcowy plik wykonywalny.
Aby skompilować program, wystarczy uruchomić polecenie 'make' w katalogu zawierającym plik Makefile.
Następnie użytkownik musi wejść w katalog 'exe' i uruchomić program, wpisując w konsoli nazwę programu razem z dodaniem nazwy pliku wczesniej wygenerowanego labiryntu.

```
./maze -f ../default_maps/[nazwapliku.txt/.bin]
```

- `-f [nazwapliku.txt]` parametr obowiązkowy. Odpowiada za dodanie pliku wejściowego. Wymaga podania nazwy ścieżki do pliku z odpowiednio sformatowanym labiryntem.

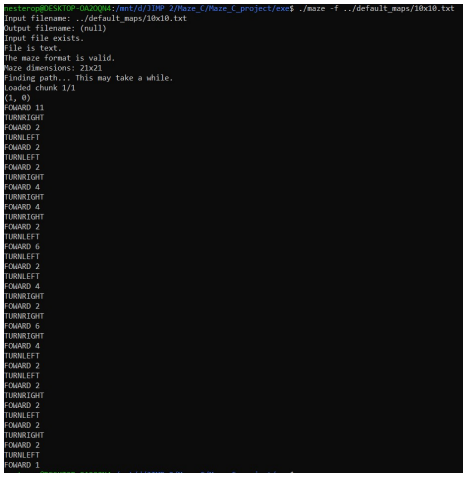
- `-o [nazwapliku.txt]` parametr opcjonalny. Po dodaniu parametru wynik działania programu zostanie przekierowany do wskazanego pliku. W przypadku braku jego istnienia plik zostanie utworzony.
- `-h` parametr opcjonalny. Wyświetla krótką pomoc do programu.

5 Przykładowe wywołania i wyniki programu

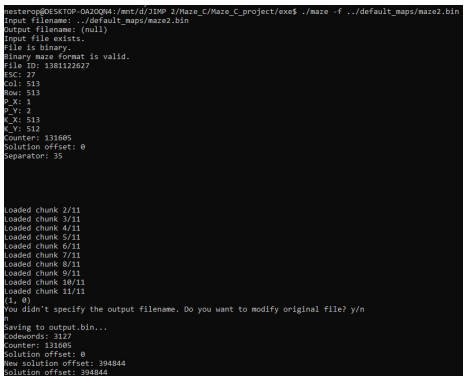
W niniejszym rozdziale zaprezentujemy zastosowania programu wraz z efektami dla różnych przypadków, aby zilustrować funkcjonowanie naszej aplikacji.

5.1 Podstawowe użycie, gdzie plik wejściowy jest podany jako argument

Wywołanie: `./maze -f ../default_maps/labirynt.txt` Wynik: Wywołanie:



`./maze -f ../default_maps/maze.bin` Wynik:

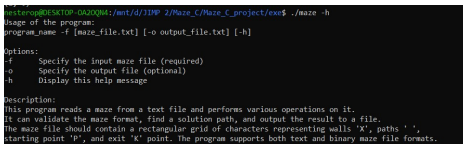


5.2 Użycie z podaniem pliku wyjściowego

Wywołanie: `./maze -f ../default_maps/labirynt.txt -o wynik.txt` Wynik:

5.3 Wyświetlenie krótkiej pomocy

Wywołanie: `./maze -h` Wynik:



6 Zmiany względem specyfikacji

W niniejszym rozdziale przedstawimy różnice pomiędzy specyfikacją funkcjonalną i implementacyjną a ostateczną wersją programu.

6.1 Implementacja Algorytmów

W dokumentacji funkcjonalnej rozważano użycie algorytmów BFS lub DFS do znajdowania najkrótszej ścieżki w labiryncie. W implementacji ostatecznie zastosowano algorytm Dijkstry, który został uznany za bardziej odpowiedni w kontekście specyfiki projektu. Ta zmiana wynikała z analizy wymagań i założeń projektowych oraz zapewniła skuteczne i efektywne rozwiązanie problemu znalezienia najkrótszej ścieżki w labiryncie. Algorytm Dijkstry został wybrany ze względu na jego dokładność i szybkość działania, co potwierdziły testy działania programu.

6.2 Diagram modułów

W początkowej wersji diagramu modułów projektu występowały podstawowe komponenty. Jednak wraz z rozwojem projektu struktura modułów znacząco się rozrosła. Dodano nowe moduły, takie jak obsługa plików binarnych (‘bin_-chunks_handling‘) oraz podział labiryntu na chunki (‘chunks_handling‘). Te zmiany umożliwiły lepsze zarządzanie pamięcią oraz dostosowanie projektu do dodatkowych wymagań i funkcjonalności.

6.3 Obsługiwane błędy

W początkowej wersji projektu obsługa błędów koncentrowała się głównie na kilku podstawowych sytuacjach. Jednak w miarę rozwoju projektu dodano obsługę nowych błędów, takich jak nieprawidłowy format pliku binarnego, nieprawidłowy format labiryntu po podziale na chunki, błędy związane z parametrami wejściowymi oraz zapisem pliku wynikowego. Dodanie obsługi tych błędów miało na celu zapewnienie użytkownikowi dokładnych informacji o przyczynach ewentualnych problemów z działaniem programu.

6.4 Zmiany w typach danych

Początkowo używano podstawowych typów danych, takich jak ‘int‘ i ‘char‘. Jednak w trakcie rozwoju projektu zdecydowano się na zmianę niektórych typów na bardziej odpowiednie. Wprowadzono ‘int16_t‘ do reprezentowania wymiarów labiryntu, co poprawiło zarządzanie pamięcią. Dodatkowo użyto ‘unsigned int‘ w implementacji algorytmu przeszukiwania grafu dla lepszej wydajności. .

6.5 Wywoływanie programu

Sposób wywoływania programu pozostał niezmieniony od początku projektu. Użytkownik nadal uruchamia program z linii poleceń, przekazując nazwę pliku z labiryntem jako argument. Opcjonalnie można również podać nazwę pliku wynikowego. Program nadal obsługuje flagi -f dla pliku wejściowego oraz -o dla pliku wyjściowego, a także flagę -h dla wyświetlenia krótkiej pomocy.

7 Podsumowanie projektu

Projekt został zakończony z sukcesem, osiągając wszystkie założone cele. Dzięki szczegółowemu planowaniu i skrupulatnej implementacji, udało się stworzyć kompletny program do analizy labiryntów. Nowe moduły, takie jak podział labiryntu na chunki, poruszanie się między nimi oraz obsługa plików binarnych, wprowadziły istotne ulepszenia w funkcjonalności. Dodatkowo, optymalizacje kodu, takie jak zmiana typu danych w celu poprawy wydajności, przyczyniły się do efektywności programu. Finalnie, projekt spełnił oczekiwania, zapewniając użytkownikom prosty i skuteczny sposób analizy labiryntów.

8 Podsumowanie współpracy

Oczywiście: Współpraca przy projekcie labiryntów była płynna i efektywna. Regularny kontakt umożliwił szybkie dostosowanie się do zmian wprowadzonych do wymagań projektowych. Korzystanie z Git’a pozwoliło nam na równoczesną pracę nad projektem, eliminując stres. Ogólnie rzecz biorąc, współpraca była satysfakcjonująca.

9 Wnioski

Projekt został pomyślnie zakończony, spełniając wszystkie założone cele. Opracowany program umożliwia skuteczną analizę labiryntów i generuje odpowiednie wyniki.
Dodanie nowych modułów, takich jak podział labiryntu na chunki czy obsługa plików binarnych, wniosło istotne ulepszenia w funkcjonalności programu, poszerzając jego możliwości.
Działania mające na celu optymalizację kodu, takie jak zmiana typu danych w celu poprawy wydajności, przyczyniły się do efektywności programu.
Projekt spełnił oczekiwania użytkowników, dostarczając prosty w obsłudze i skuteczny sposób analizy labiryntów.
Współpraca podczas projektu przebiegała płynnie i efektywnie, co pozwoliło na elastyczne dostosowanie się do zmian oraz uniknięcie zbędnego stresu.