

Dokumentacja Projektu AISD

Spis treści

1. Zespół
 2. Opis projektu
 3. Wymagania
 4. Struktura katalogów
 5. Główne klasy i pliki
 6. Implementacje algorytmów
 7. Opis działania
 8. Zależności
-

Zespół

Zespół jest złożony z 4 osób.

- Dorian Konwiński - programista, współtwórca dokumentacji
 - Maciej Krawczyk - kierownik zespołu, współtwórca dokumentacji
 - Jakub Kwiatkowski - główny programista, UI designer
 - Kacper Nadolny - programista, twórca dokumentacji
-

Opis projektu

Not Enough Ale to gra symulacyjna z graficznym interfejsem użytkownika, wykorzystująca bibliotekę SFML. Użytkownik może budować sieć połączeń pomiędzy różnymi typami budynków (farmy, tawerny, karczmy, skrzyżowania), zarządzać zasobami oraz przeprowadzać symulacje przepływu surowców. Gra umożliwia zapisywanie i wczytywanie stanu rozgrywki.

Wymagania

- Kompilator C++ (standard C++17)

- Visual Studio 2022
 - Biblioteka SFML (zawarta już w projekcie)
-

Struktura katalogów

```
ProjektAISD/
|
├─ Button.cpp / hpp          # Klasa przycisku GUI
├─ Convex.cpp / hpp          # Logika otoczki
├─ Functions.cpp / hpp       # Funkcje pomocnicze, narzędziowe
├─ FileCompression.cpp / hpp # Funkcje związane z kompresją plików
├─ FlowAlgorithm.cpp / hpp   # Obliczenia przepływów podczas tury
├─ Game.cpp / hpp            # Główna logika gry
├─ Line.cpp / hpp            # Klasa reprezentująca połączenie (drogę)
├─ Menu.cpp / hpp            # Menu gry
├─ Node.cpp / hpp            # Klasa reprezentująca budynek/węzeł
├─ Źródło.cpp                # Plik główny (main)
├─ Texture/                  # Folder z teksturami do gry
└─ ...
```

Główne klasy i pliki

Game (Game.cpp / Game.hpp)

- Centralna klasa zarządzająca logiką gry, renderowaniem, obsługą zdarzeń, zapisem i wczytywaniem stanu gry.
- Najważniejsze metody:
 - run() – główna pętla gry.
 - processEvents() – obsługa zdarzeń okna.
 - update() – aktualizacja stanu gry.
 - render() – rysowanie elementów na ekranie.
 - handleMouseInput() – obsługa myszy (dodawanie węzłów, linii, interakcje z UI).
 - handleKeyboardInput() – obsługa klawiatury (np. zoom, zamknięcie gry).
 - SaveGame() / LoadGame() – zapis i odczyt stanu gry do/z pliku.
 - TurnEnd() – zakończenie tury, uruchomienie algorytmu przepływu.

Node (Node.cpp / Node.hpp)

- Reprezentuje budynek/węzeł na mapie (farma, tawerna, karczma, skrzyżowanie).
- Właściwości: Pozycja, typ, pojemność, tekstura.

Line (Line.cpp / Line.hpp)

- Reprezentuje połączenie (drogę) pomiędzy dwoma węzłami.
- Właściwości: Wskaźniki na węzły początkowy i końcowy, pojemność, koszt, tekstura.

Button (Button.cpp / Button.hpp)

- Prosty przycisk GUI, wykorzystywany do obsługi trybów gry i akcji użytkownika.

Functions (Functions.cpp / Functions.hpp)

- Zbiór funkcji narzędziowych, m.in. do obsługi kolizji, rysowania, kompresji danych.

Menu (Menu.cpp / Menu.hpp)

- Obsługa menu gry (np. ekran startowy, wybór opcji).

Źródło.cpp

- Plik główny, uruchamiający aplikację i inicjalizujący główne komponenty.

Implementacje algorytmów

1. Przepływ w sieci - obliczanie maksymalnego przepływu z minimalnym kosztem - algorytm Busackera-Gowena; w implementacji do symulacji przejścia od farm do karczm przez piwiarnie problem został podzielony na dwie warstwy. Wyznaczane są przepływy z farm do piwiarni, a następnie z piwiarni do karczm.
 $O(F \cdot T)$; $T = O((V + E)\log V)$, F = iteracje pętli
2. Wyznaczanie drogi przepływu - znalezienie najszybszej drogi do karczmy - algorytm Dijkstry z rozszerzony o potencjały.
 $O((V + E)\log V)$
3. Otoczki wypukłe - do tworzenia pola i wzmacnianie plonów farm; losuje się 15 punktów w pewnym obszarze do konstrukcji otoczki - algorytm Grahama.
 $O(n \log n)$, $n = 15$

4. Sprawdzanie położenia obiektów - szukanie farm w polu otoczki - wyszukiwanie binarne
 $O(\log n)$, n = ilość punktów otoczki
 5. Zapis danych do pliku - efektywne oszczędzanie pamięci i kompresja danych gry - kodowanie Huffmana.
 $O(n \log n)$, n = ilość unikalnych znaków
 6. Szukanie wzorca w tekście - szybkie powracanie do informacji z poprzednich tur - algorytm Boyera-Moore'a.
 $O(n \cdot m)$, n = długość tekstu, m = długość wzorca
-

Opis działania

1. Uruchomienie gry: Tworzone jest okno SFML, ładowane są tekstury i czcionki.
 2. Interakcja użytkownika: Użytkownik może dodawać budynki i drogi, zmieniać tryby za pomocą przycisków, przesuwać widok, zapisywać/wczytywać grę.
 3. Symulacja: Po zakończeniu tury uruchamiany jest algorytm przepływu, który oblicza dostawy i koszty.
 4. Zapis/Wczytanie: Stan gry (budynki, drogi, statystyki) jest kompresowany i zapisywany do pliku, a następnie może być odczytany.
-

Zależności

- SFML: Biblioteka do obsługi grafiki, dźwięku i wejścia/wyjścia.
- Standardowa biblioteka C++: Wykorzystywana do obsługi kontenerów, strumieni, itp.