**Gra „Unik”**

**Autor: Mikołaj Janczak**

**Spis treści:**

1. Wstęp
2. Instalacja
3. Uruchamianie gry
4. Struktura kodu
5. Implementacja
6. Plik konfiguracyjny
7. Gra

**Wstęp**

**Nazwa gry:** Unik

**Opis gry:** Unik to gra, w której gracz steruje bohaterem starającym się unikać rakiet poruszających się w różnych kierunkach. Celem gry jest przetrwać jak najdłużej, unikając kolizji z rakietami.

**Platforma:** Windows

**Wymagania systemowe:**

* **Minimalne:**
  + System operacyjny: Windows 10/11
  + Procesor: Intel Core i3
  + Pamięć: 2GB RAM
  + Grafika: Karta graficzna zgodna z OpenGL 3.3
  + Dysk 100MB wolnego miejsca

**Instalacja**

**Kroki instalacji:**

1. Pobierz kod źródłowy gry.
2. Zainstaluj wszystkie zależności (patrz poniżej).
3. Skonfiguruj środowisko pod kątem zależności
4. Skompiluj projekt.

**Zależności:**

* Biblioteka SFML (Simple and Fast Multimedia Library)

**Struktura projektu**

**Przegląd kodu:**

* ***‘headers/’:*** Katalog z plikami nagłówkowymi (.h)
* ***‘source/’*:** Katalog z plikami źródłowymi (.cpp)
* ***‘graphic/’:*** Katalog z grafiką (tekstury bohatera i rakiet)
* ***‘config.txt’:*** Plik konfiguracyjny z parametrami gry

**Diagram klas:**

**WSTAWIC ZDJECIE**

**Opis modułów:**

* **Object:** Klasa bazowa dla wszystkich obiektów w grze (Hero, Rocket).
* **Hero:** Klasa reprezentująca bohatera sterowanego przez gracza.
* **Rocket:** Klasa reprezentująca rakiety poruszające się po mapie.
* **Engine:** Klasa zarządzająca logiką gry, mapą, rakietami i bohaterem.
* **Game:** Klasa zarządzająca główną pętlą gry.
* **GameSFML:** Klasa integrująca bibliotekę SFML do renderowania grafiki.
* **HeroControllerSFML:** Klasa zarządzająca inputem użytkownika dla bohatera przy użyciu SFML.

**Schemat architektury:**

* Silnik gry (**‘Engine’**) zarządza stanem gry, przetwarza ruchy rakiet i bohatera, oraz sprawdza kolizje.
* **‘Game’** rozszerza **‘Engine’** o główną pętlę gry.
* **‘GameSFML’** rozszerza **‘Game’** o funkcjonalności renderowania grafiki przy użyciu SFML.

**Implementacja**

**Główne klasy i funkcje:**

**Klasa Object**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* ***‘Object()’:*** Konstruktor klasy Object, inicjalizuje obiekt.
* ***‘~Object()’:*** Destruktor klasy Object.
* ***‘void startingPosition()’***: Ustawia początkową pozycję obiektu na (0,0).
* ***‘int getLocationX() const’:*** Zwraca współrzędną x lokalizacji obiektu.
* ***‘int getLocationY() const’:*** Zwraca współrzędną y lokalizacji obiektu.
* ***‘void move()’:*** Wirtualna funkcja do poruszania obiektem, do nadpisania w klasach pochodnych.

**Klasa Hero**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* ***‘Rocket()’***: Konstruktor klasy Rocket, inicjalizuje rakietę z losowym kierunkiem.
* ***‘~Rocket():’*** Destruktor klasy Rocket.
* ***‘int getDirection() const’***: Zwraca aktualny kierunek rakiety.
* ***‘void setDirection’():*** Ustawia losowy kierunek rakiety.
* ***‘void startingPosition()’:*** Ustawia początkową pozycję rakiety na jednej z krawędzi mapy.
* ***‘bool isAtEdge(int direction) const’:*** Sprawdza, czy rakieta jest na krawędzi mapy w danym kierunku.
* ***‘void move()’:*** Porusza rakietą w zależności od jej kierunku.

**Klasa Engine**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* ***‘Engine(const char fn)’:*** Konstruktor klasy Engine, inicjalizuje silnik gry.
* ‘~Engine():’ Destruktor klasy Engine.
* ***‘void readConfig(const char fn)’:*** Wczytuje konfigurację gry z pliku.
* ***‘void createMap()’:*** Tworzy mapę gry.
* ***‘void createHero()’:*** Tworzy bohatera gry.
* ***‘void createRockets()’:*** Tworzy rakiety i ustawia ich początkowe pozycje.
* ***‘void replaceRocket(int index)’:*** Zastępuje rakietę na nowej pozycji.
* ***‘bool checkCollision()’:*** Sprawdza kolizje między bohaterem a rakietami.
* ***‘void Analysis()’:*** Analizuje stan gry, aktualizuje pozycje bohatera i rakiet.
* ***‘void MoveRockets()’:*** Zarządza ruchem rakiet w oddzielnym wątku.

**Klasa Game**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* ***‘Game(const char fn)’:*** Konstruktor klasy Game, inicjalizuje grę.
* ***‘Game()’:*** Destruktor klasy Game.
* ***‘void Run()’:*** Główna pętla gry.
* ***‘void Show()’:*** Wyświetla stan gry.

**Klasa GameSFML**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

* ***‘GameSFML(const char fn)’:*** Konstruktor klasy GameSFML, inicjalizuje grę z grafiką SFML.
* ***‘~GameSFML()’:*** Destruktor klasy GameSFML.
* ***‘void Show() override’:*** Rysuje bohatera i rakiety na ekranie.
* ***‘void Run() override’:*** Uruchamia główną pętlę gry.

**Klasa HeroControllerSFML**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* ***‘HeroControllerSFML(Hero& h)’***: Konstruktor klasy **HeroControllerSFML**, inicjalizuje kontroler bohatera.
* ***‘void handleInput(const sf::Event& event)’***: Obsługuje wejście użytkownika przy użyciu SFML.

**Plik konfiguracyjny**

Przykładowa zawarto pliku ***‘config.txt’***:  
14  
5

Plik konfiguracyjny config.txt zawiera dwie wartości, które definiują ustawienia gry:

* Pierwsza wartość (14) - określa*, ile rakiet* będzie obecnych w grze.
* Druga wartość (5) - określa *prędkość* poruszania się rakiet.   
  Skala prędkości wynosi od 0 do 10, gdzie wyższa wartość oznacza szybsze poruszanie się rakiet.

**Gra**

Po uruchomieniu gry otwiera się nowe okno które prezentuje „Unik”.  
Nasz bohater zostanie umieszczony na środku planszy.  
W celu unikania rakiet możemy poruszać się za pomocą wciskania przycisków:

* **‘W’** : ruch postaci do góry
* ‘S’ : ruch postaci w dół
* ‘A’: ruch postaci w lewo
* ‘D’: ruch postaci w prawo

Po trafieniu naszej postaci przez rakietę okno zamknie się.  
 W konsoli zostanie wypisany komunikat: *„Game Over! You were hit by a rocket!”*

Obraz zawierający zrzut ekranu

Opis wygenerowany automatycznie