**Gra „Unik”**

**Autor: Mikołaj Janczak**

**Spis treści:**

1. Wstęp
2. Instalacja
3. Uruchamianie gry
4. Struktura kodu
5. Implementacja
6. Założenia funkcjonalne
7. Zwracane kody
8. Plik konfiguracyjny
9. 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.

**Instalacja**

**Dla środowiska CLion**

**Kroki instalacji:**

1. Pobierz pliki źródłowe gry.
2. Zainstaluj wszystkie zależności (patrz poniżej).
3. Otwórz projekt za pomocą CLion.
4. Skompiluj projekt za pomocą środowiska CLion.

**Zależności:**

* Biblioteka SFML (Simple and Fast Multimedia Library) – znajdująca się pod ścieżką C:\SFML
* Dla środowiska CLion stworzony jest plik CMakeLists.txt który przechowuje już ustawione ścieżki do biblioteki SFML. Dlatego wymagane jest zainstalowanie tej biblioteki w konkretnym katalogu.

**Dla innych środowisk**

**Kroki instalacji:**

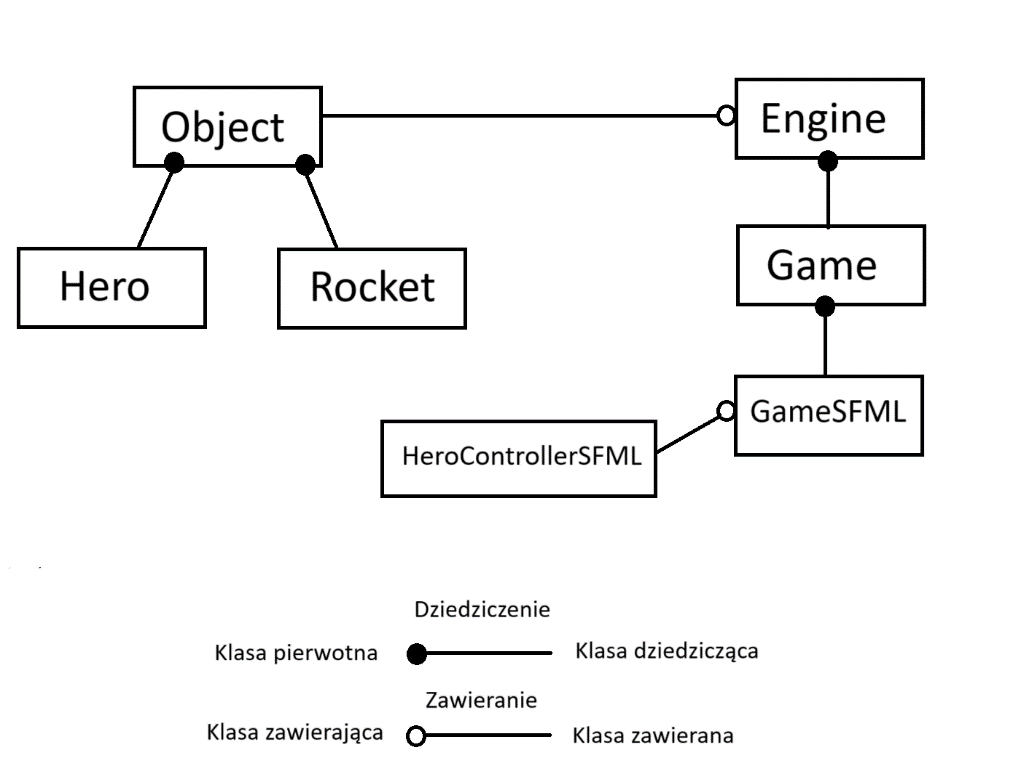
1. Pobierz kod źródłowy gry.
2. Zainstaluj wszystkie potrzebne biblioteki wykorzystywane w projekcie.
3. Skompiluj projekt.

**Struktura projektu**

**Przegląd kodu:**

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

**Diagram klas:**

****

**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:**

Wszystkie klasy mają zablokowany konstruktor kopiujący oraz operator przypisania, aby zapobiec niezamierzonym kopiowaniom obiektów.

**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 (char input) = 0’:*** Wirtualna funkcja do poruszania obiektem, do nadpisania w klasach pochodnych.

**Klasa Hero**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* ***‘Hero()’: Konstruktor klasy Hero, inicjalizuje bohatera.***
* ***‘~Rocket():’*** Destruktor klasy Rocket.
* ***‘~Hero()’:*** Destruktor klasy Hero.
* ***‘void startingPosition() override’:*** Ustawia początkową pozycję bohatera.
* ***‘void move(char input) override’:*** Porusza bohaterem w zależności od inputu.

**Klasa Rocket**

**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, oprogramowanie

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

**Założenie funkcjonalne**

* **Sterowanie postacią**
  + Gracz steruje bohaterem za pomocą klawiszy
    - ‘W' : ruch postaci do góry
    - 'S' : ruch postaci w dół
    - 'A' : ruch postaci w lewo
    - 'D' : ruch postaci w prawo
* **Ruch rakiet:**
  + Rakiety poruszają się w losowo wybranych kierunkach.
  + Rakiety zmieniają kierunek ruchu po osiągnięciu krawędzi mapy.
  + Prędkość rakiet jest konfigurowalna i może być ustawiona w pliku konfiguracyjnym.
* **Kolizje**
  + Gra kończy się, gdy bohater zostanie trafiony przez rakietę.
  + Po kolizji, wyświetla się komunikat "Game Over! You were hit by a rocket!" i okno gry się zamyka.
* **Konfiguracja gry**
  + Plik konfiguracyjny config.txt definiuje liczbę rakiet oraz prędkość ich poruszania się.
  + Liczba rakiet i ich prędkość mogą być zmieniane poprzez edycję pliku konfiguracyjnego.
* **Rozpoczęcie gry:**
  + Bohater rozpoczyna grę na środku planszy.
  + Rakiety są rozmieszczane losowo na krawędziach mapy.
* **Zależności:**
  + Gra wymaga biblioteki SFML do renderowania grafiki oraz obsługi wejścia użytkownika.
* **Struktura projektu:**
  + Kod źródłowy jest podzielony na różne klasy i pliki nagłówkowe, które zarządzają różnymi aspektami gry (np. bohater, rakiety, silnik gry, grafika SFML).
* **Obsługa błędów:**
  + Różne kody wyjścia informują o błędach, takich jak niepowodzenie w ładowaniu tekstur czy błędy w pliku konfiguracyjnym.

**Kody zwracane przez program**

Program "Unik" zwraca różne kody wyjścia w zależności od sytuacji. Poniżej znajduje się lista kodów wyjścia i ich znaczenie:

* ***0***: Program zakończył się pomyślnie.
* ***420***: Błąd podczas ładowania tekstury bohatera. Sprawdź, czy plik *hero.png* znajduje się w katalogu *graphic*.
* ***421***: Błąd podczas ładowania tekstury rakiety. Sprawdź, czy plik *rocket.png* znajduje się w katalogu *graphic*.
* ***500***: Błąd podczas otwierania pliku konfiguracyjnego. Upewnij się, że plik *config.txt* jest dostępny i ma poprawny format.
* ***600***: Błąd związany z prędkością gry. Prędkość gry musi mieścić się w zakresie od 0 do 10.

**Plik konfiguracyjny**

**a**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.

Plik konfiguracyjny powinien zawierać się w tym samym katalogu co plik wykonywalny czyli *‘Unik.exe’*

**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

Obraz zawierający zrzut ekranu

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie