Gra komputerowa
Aircraft Shooter

Autor:
Dominik Budzyński

Spis treści

1.Wstęp

Celem projektu było stworzenie gry komputerowej – Aircraft Shooter, polegającej na na niszczeniu przeciwników pod postacią samolotów, za pomocą pocisków wystrzeliwanych przez samolot gracza, którym można sterować.

Projekt został zrealizowany z wykorzystaniem języka programowania C++ oraz bibliotek SDL2 (Simple DirecMedia Layer) oraz SDL2_image.

2.Funkcjonalność

Gra umożliwia sterowanie obiektem gracza pod postacią samolotu za pomocą strzałek. Nowi przeciwnicy pojawiają się w odstępie czasowym większym niż 500 [ms] i mniejszym niż 2000 [ms] pod warunkiem, że ich ilość jest mniejsza niż 5. Z samolotu gracza co 300 [ms] wystrzeliwane są pociski, które jeśli trafią w samolot przeciwnika powodują jego zniszczenie. Przeciwnicy pojawiając się w górnej części okna, zaczynają zmierzać w kierunku gracza.

Wymagania systemowe:

1. Visual Studio 2022
2. Biblioteki SDL2 oraz SDL2_image
3. C++ Language Standard: ISO C++ 20 Standard

3.Analiza problemu

1. Pierwszym zagadnieniem wymaganym do zrealizowania było opanowanie podstawowej składni i funkcji którejś z dostępnych bibliotek graficznych umożliwiających operacje na oknie.
2. Stworzenie własnych grafik
3. Opracowanie sposobu na uzyskanie stałej liczby klatek wyświetlanych w ciągu sekundy

Na potrzeby uzyskania stałej ilości klatek na sekundę wykorzystano stworzone timery (stopery). Wykorzystano je do mierzenia czasu, w milisekundach, trwania każdej klatki (jednego przejścia przez pętlę główna programu).

Dla przykładu, jeśli stała liczba klatek na sekundę którą chcemy uzyskać wynosi 60, wiadomo, że jedna klatka powinna trwać około 1000/60 = 16.66 [ms]

Program mierzy czas trwania jednej klatki i jeśli jest on mniejszy od wyznaczonego, usypia swoje działanie na tyle milisekund ile brakuje do wyznaczonej wartości.

4. Opracowanie funkcjonalności odpowiadającej za „ruch” tekstur.

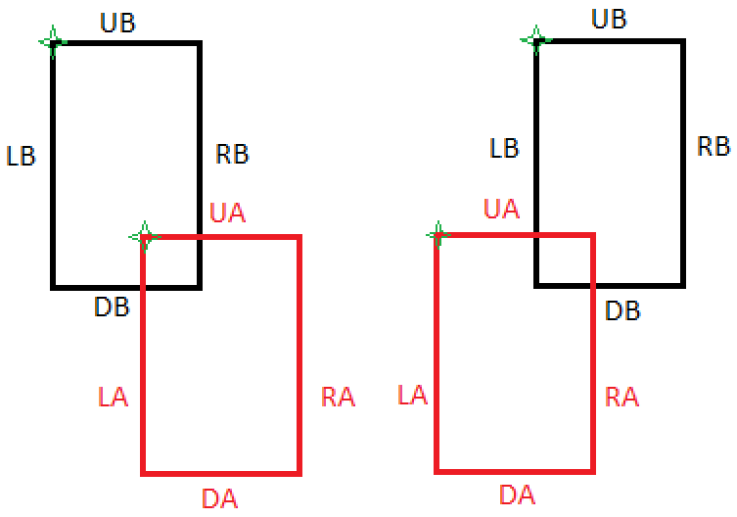
   Ruch tekstur zrealizowany został za pomocą zmiany pozycji wyświetlania konkretnego obiektu. Każdy z obiektów posiada własną wartość prędkości z jaką się porusza. Wartość ta decyduje o ile pikseli poruszy się każdy z obiektów w ciągu trwania każdej klatki.

5. Opracowanie sposobu na przechowywanie oraz wyświetlanie wielu obiektów tego samego rodzaju.
   Aby gra była bardziej ekscytująca warto zadbać o możliwość wyświetlania więcej niż jednego przeciwnika. Język C++ umożliwia dynamiczne zarządzanie ilością przechowywanych obiektów za pomocą klasy vector. Wykorzystano jej funkcjonalność do tworzenia i wyświetlania wielu przeciwników w jednym momencie.
6. Opracowanie sposobu na wykrywanie kolizji konkretnych obiektów.

   Wykrywanie kolizji może zostać zrealizowane na kilka sposobów np. wykrywanie kolizji pikseli lub wykrywanie kolizji pozycji wyświetlanych tekstur.

   W tym projekcie zastosowano wykrywanie kolizji za pomocą pozycji wyświetlanych tekstur.

   Poniżej przedstawiono schemat rozumowania za pomocą którego utworzono wykrywanie kolizji.



Kolizja ma być wykrywana kiedy położenie jednego z obiektów będzie kolidowało z położeniem drugiego. Tzw. prostokąty położenia, które dostarcza biblioteka SDL, za punkt odniesienia biorą lewy górny róg wyświetlanej tekstury (na grafice zaznaczone zieloną gwiazdką). Kolizja nastąpi kiedy lewy górny róg lub prawy górny róg tekstury A (czerwonej) będzie znajdował się powyżej dolnej granicy tekstury B (Db) oraz poniżej górnej granicy tekstury B (UB). Jednak tylko to nie wystarczy, należy wziąć pod uwagę również lewą krawędź oraz prawą krawędź. W tym programie inna sytuacja nie wystąpi ponieważ pociski zawsze uderzają w przeciwników lecąc od dolnej części ekranu.

7. Generowanie losowego położenia przeciwników oraz losowych odstępów czasowych dla których mają się oni pojawiać.

   Aby uzyskać losowe wartości, w projekcie wykorzystano algorytm Mersenne'a.

4.Opis składowych

Projekt został podzielony na:

- Pliki
  - Pliki nagłówkowe
  - Pliki źródłowe
- Klasy
  - Każda klasa zawiera swoją deklarację w oddzielnym pliku nagłówkowym

Poniżej przedstawiono wykaz klas:

- Game
- Texture
- MovingObject
- Timer

4.1 Opis klasy Game

Do zadań klasy Game należą:

- obsługa inicjalizacji biblioteki SDL oraz SDL_Image
- tworzenie okna oraz zmiennych związanych z oknem
- obsługa ruchu tekstury gracza

Klasa Game w większości korzysta z funkcji bibliotecznych SDL, a nie z metod napisanych przez autora.

4.2 Opis klasy Texture

Do zadań klasy Texture należą:

- ładowanie tekstur
- przechowywanie informacji na temat tekstur
- zwracanie informacji o danej teksturze
- kontrolowanie położenia tekstury
- identyfikacja tekstury (ta funkcjonalność ma pokrycie w kolejnej klasie)

Obiekt klasy Texture można stworzyć na dwa sposoby:

- nie podając żadnych argumentów podczas tworzenia – wtedy domyślna pozycja tekstury na ekranie będzie wynosić x = 0, y = 0 (biblioteka SDL2 za początek układu współrzędnych obiera lewy górny róg okna)
- jako argumenty podając kolejno:
  - xpos – pozycje początkową X
  - ypos - pozycję początkową Y
  - w – szerokość tekstury
  - h – wysokość tekstury

4.3 Opis klasy MovingObject

Klasa MovingObject korzysta z wielu metod klasy Texture. Wynika to z tego, że klasa MovingObject przechowuje w sobie vector o elementach typu Texture. Pozwala to na tworzenie wielu obiektów klasy Texture za pomocą jednego obiektu klasy MovingObject. Dzięki temu w pętli głównej programu można utworzyć jeden obiekt i za pomocą niego zarządzać wyświetlaniem i zachowaniem wielu tektur.

Klasa MovingObject odpowiada za:

- dodawanie wielu tekstur tego samego rodzaju (np. wiele przeciwników, wiele pocisków)
- kontrolowanie każdej tekstury z osobna
- niszczenie tekstur tego samego rodzaju

4.4 Opis klasy Timer

Klasa Timer to prymitywny stoper, który w pętli głównej programu wykorzystany został do mierzenia czasu między konkretnymi zdarzeniami.

- posiada możliwość wystartowania czasu i jego wyzerowania
- posiada możliwość pobrania czasu działania stopera

Każda z metod powyższych klas została dokładniej opisana przez komentarze w kodzie.

5.Przyszły rozwój – elementy do poprawy

Projekt posiada kilka elementów które wymagają poprawy:

- brak interakcji pojazdu gracza z pojazdami przeciwników
- sposób pojawiania się i znikania przeciwników – docelowo przeciwnicy powinni być renderowani „nad” oknem a w razie nie zestrzelenia, usuwani „pod” oknem, w taki sposób aby sprawiały wrażenie przelatywania przez przestrzeń większą niż okno programu
- dodanie zliczania punktów i wyświetlania ich na ekran
- dodanie udźwiękowienia