REV	DATA	ZMIANY
0.1	09.01.2025	*Maciej Kucharski (mackuchar@student.agh.edu.pl)*
0.2	11.01.2025	*Maciej Kucharski (mackuchar@student.agh.edu.pl)*
0.3	19.01.2025	*Maciej Kucharski (mackuchar@student.agh.edu.pl)*
0.4	22.01.2025	*Maciej Kucharski (mackuchar@student.agh.edu.pl)*
0.5	26.01.2025	*Maciej Kucharski (mackuchar@student.agh.edu.pl)*

GRA KOMPUTEROWA

ASTRO CRUISER

Autor: Maciej Kucharski
Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Spis treści

Lista oznaczeń

FPS	Frames Per Second
UFO	Unidentified Flying Object
IDE	Integrated Development Environment
2D	Two Dimensional
UML	Unified Modeling Language

1. Wstęp

Celem projektu było stworzenie gry zręcznościowej typu "space shooter" o nazwie "Astro Cruiser", w której gracz steruje statkiem kosmicznym i broni się przed falami przeciwników oraz unika pocisków chowając się za asteroidami, zbierając punkty. Celem jest uzyskanie jak najwyższego wyniku i przetrwanie jak najdłużej.

"Astro Cruiser" to dynamiczna gra 2D, w której gracz kontroluje statek kosmiczny poruszający się w dolnej części ekranu. Zadaniem gracza jest zestrzeliwanie nadlatujących z góry przeciwników i unikanie wystrzeliwanych przez wrogów pocisków chowając się za asteroidami które stanowią bariery. Gra zawiera system punktacji, ranking najlepszych wyników, fale przeciwników o rosnącym poziomie trudności oraz okazjonalnie pojawiające się UFO, za zestrzelenie którego przyznawane są dodatkowe punkty. W miarę postępów w grze, przeciwnicy stają się szybsi. Rozgrywka kończy się po utracie wszystkich żyć, a wynik gracza jest zapisywany w rankingu.

Zakres projektu:

- Wykorzystanie biblioteki Raylib do obsługi grafiki, dźwięku i wejścia.
- Implementacja mechaniki poruszania się statku gracza.
- Implementacja systemu strzelania pociskami przez gracza i przeciwników.
- Implementacja fal przeciwników o rosnącym poziomie trudności.
- Implementacja mechaniki pojawiania się UFO.
- Implementacja systemu kolizji między obiektami (pociski, statek gracza, przeciwnicy, asteroidy).
- Implementacja systemu punktacji i zapisu najlepszych wyników.
- Implementacja prostego interfejsu użytkownika (ekran startowy, menu pauzy, ekran śmierci).
- Implementacja muzyki w tle i efektów dźwiękowych.

2. Funkcjonalność

- Wymagania funkcjonalne:
    - Gracz może sterować statkiem w lewo i prawo za pomocą klawiszy A oraz D
    - Gracz może strzelać pociskami za pomocą klawisza spacji.
    - Przeciwnicy pojawiają się falami u góry ekranu i poruszają się na boki i w dół.
    - Asteroidy pojawiają się na początku gry i stanowią powoli niszczącą się barierę dla gracza
    - Zestrzelenie przeciwnika lub statku UFO dodaje punkty do wyniku gracza.
    - Kolizja statku gracza z przeciwnikiem lub jego pociskiem powoduje utratę życia.
    - Gra wyświetla aktualny wynik, liczbę żyć, numer fali i stan gry (rozpoczęta, pauza, zakończona).
    - Gra umożliwia zapisanie wyniku gracza i wyświetla ranking 10 najlepszych wyników.
    - Gra powinna posiadać ekran startowy z przyciskami: "Start", "Credits" i "Exit".
    - Gra powinna posiadać możliwość zapauzowania rozgrywki.

- Wymagania niefunkcjonalne:
    - Gra powinna działać z częstotliwością 60 klatek na sekundę.
    - Gra powinna być intuicyjna i łatwa w obsłudze.
    - Gra powinna być atrakcyjna wizualnie.
    - Kod gry powinien być czytelny, dobrze udokumentowany.
    - Gra powinna posiadać muzykę i efekty dźwiękowe.

3. Projekt techniczny

DIAGRAM UML

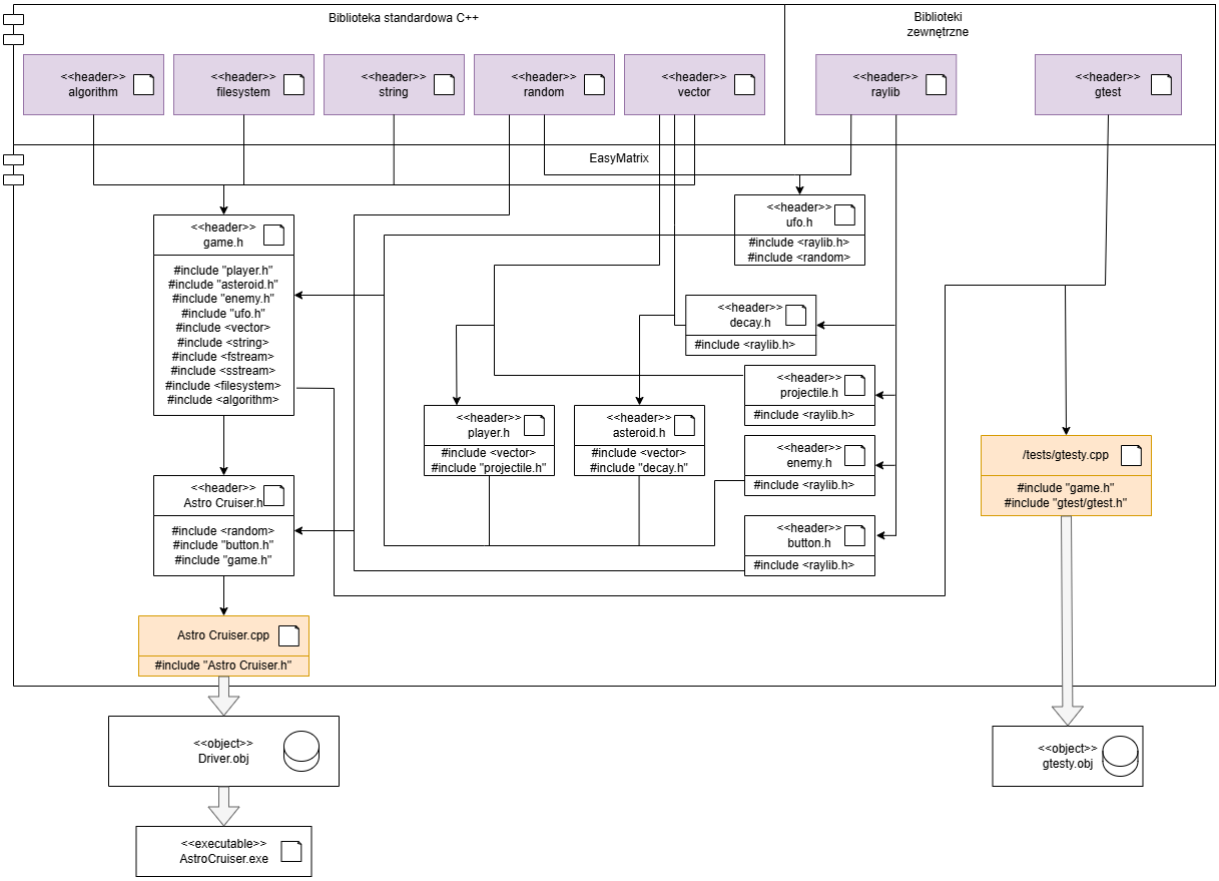


Diagram UML (figure 1) – strukturalna budowa plików w grze „Astro Cruiser” i zależności między plikami headerowymi.

3.1 Opis I hierarchia klas

- Game: Główna klasa zarządzająca logiką gry. Odpowiada za inicjalizację, aktualizację stanu, obsługę wejścia, sprawdzanie kolizji i rysowanie.
- Player: Reprezentuje statek gracza. Umożliwia poruszanie się, strzelanie i przechowuje informacje o pozycji.
- Projectile: Reprezentuje pociski wystrzeliwane przez gracza i przeciwników.
- Enemy: Reprezentuje przeciwników.
- Asteroid: Reprezentuje asteroidy.
- UFO: Reprezentuje specjalnego przeciwnika - UFO.
- Button: Reprezentuje przyciski interfejsu użytkownika.
- Decay: Reprezentuje efekt rozpadu asteroidy przez kolizje z pociskami

DIAGRAM KLAS

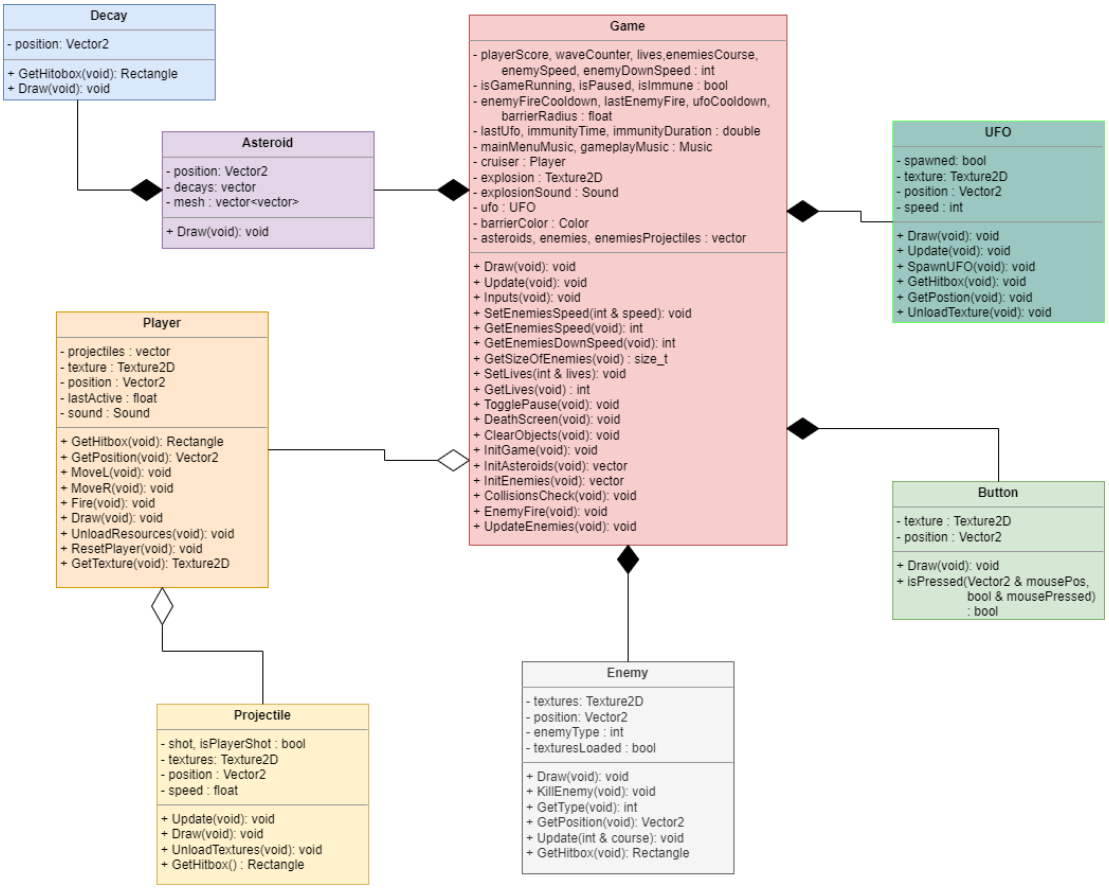


Diagram klas (figure 2) – wizualna reprezentacja zmiennych oraz funkcji zdefiniowanych w danych klasach i ich zależności między sobą.

3.2 Opis algorytmów oraz scenariusze rozgrywki

- **Algorytm poruszania się przeciwników:** Przeciwnicy poruszają się w dół ekranu, zmieniając kierunek w osi X po dotarciu do krawędzi. Ich prędkość rośnie wraz z postępem gry.
- **Algorytm strzelania przeciwników:** Przeciwnicy strzelają losowo z pewnym interwałem czasowym.
- **Algorytm kolizji:** Sprawdzanie kolizji opiera się na prostokątach ograniczających obiekty (hitboxach). Funkcja CheckCollisionRecs z biblioteki Raylib jest używana do wykrywania kolizji między dwoma prostokątami.
- **Algorytm generowania fal przeciwników:** Fale przeciwników są generowane z określoną liczbą i typem przeciwników. Każda kolejna fala jest trudniejsza (więcej przeciwników, szybsi przeciwnicy).
- **Algorytm punktacji:** Punkty są przyznawane za zestrzelenie przeciwników i UFO.

Scenariusze gry

- **Scenariusz 1: Rozpoczęcie nowej gry:**
    1. Użytkownik klika przycisk "Start" w menu głównym.
    2. Obiekt Game jest inicjalizowany (InitGame()).
    3. Tworzone są obiekty Player, Asteroid, Enemy.
    4. Rozpoczyna się pętla gry (Update()).
- **Scenariusz 2: Ruch statku gracza w lewo:**
    1. Gracz naciska klawisz strzałki w lewo.
    2. Metoda Inputs() klasy Game wykrywa wciśnięcie klawisza.
    3. Wywoływana jest metoda MoveL() klasy Player.
    4. Pozycja statku gracza jest aktualizowana.
- **Scenariusz 3: Zestrzelenie przeciwnika:**
    1. Gracz wystrzeliwuje pocisk.
    2. Pocisk porusza się w górę ekranu (Projectile::Update()).
    3. Metoda CollisionsCheck() klasy Game wykrywa kolizję pocisku z przeciwnikiem.
    4. Przeciwnik zostaje usunięty, a gracz otrzymuje punkty.
    5. Tworzony jest obiekt Explosion w miejscu trafienia przeciwnika.

3.3 Biblioteka Raylib

Raylib 5.0 - prosta i łatwa w użyciu biblioteka, zaprojektowana z myślą o programowaniu gier wideo i narzędzi multimedialnych. Jest to biblioteka typu open-source. Raylib jest napisana w języku C, ale oferuje interfejsy do wielu innych języków programowania, w tym C++. Poniżej znajduje się krótka specyfikacja biblioteki:

- **Główne cechy Raylib:**
    - **Brak zewnętrznych zależności:** Raylib jest biblioteką samodzielną, co ułatwia jej kompilację i integrację z projektami.
    - **Obsługa wielu platform:** Raylib działa na systemach Windows, Linux, macOS, Android, iOS oraz w przeglądarkach internetowych (przez WebAssembly).
    - **Sprzętowa akceleracja grafiki:** Raylib wykorzystuje OpenGL (w wersjach od 1.1 do 4.3) do akceleracji sprzętowej renderowania grafiki 2D i 3D.
    - **Obsługa grafiki 2D:** Funkcje do rysowania podstawowych kształtów geometrycznych, teksturowania, obsługi animacji, shaderów, itp.
    - **Obsługa grafiki 3D:** Funkcje do rysowania podstawowych obiektów 3D, obsługi modeli, teksturowania, oświetlenia, shaderów, itp.

- **Obsługa dźwięku:** Funkcje do ładowania i odtwarzania plików dźwiękowych w różnych formatach (np. WAV, OGG, MP3), obsługa strumieniowania dźwięku.
- **Obsługa wejścia:** Funkcje do obsługi klawiatury, myszy, dotyku, gamepadów.
- **Obsługa fizyki:** Zintegrowany moduł do obsługi fizyki 2D (opcjonalny).
- **Łatwość użycia:** Raylib oferuje prosty i intuicyjny interfejs API, wzorowany na starych frameworkach
- **Obszerna dokumentacja i przykłady:** Dostępna jest obszerna dokumentacja API oraz liczne przykłady, które ułatwiają naukę i korzystanie z biblioteki.

- **Dlaczego Raylib został wybrany do tego projektu:**
  - **Prostota i łatwość nauki:** Raylib jest idealny dla początkujących programistów gier ze względu na swój prosty interfejs API i dobrą dokumentację.
  - **Wystarczające możliwości:** Raylib oferuje wszystkie funkcje potrzebne do stworzenia gry 2D
  - **Brak zewnętrznych zależności:** Ułatwia to kompilację i dystrybucję gry.
  - **Wsparcie dla C++:** Bezproblemowa integracja z kodem C++.

4. Opis realizacji oraz implementacja

- **Środowisko programistyczne:**
  - Język programowania: C++
  - Biblioteka: Raylib

    Wykorzystane moduły Raylib:
    - rcore.h: Inicjalizacja okna, kontekstu graficznego, obsługa wejścia, pętla gry.
    - rshapes.h: Rysowanie podstawowych kształtów (używane do rysowania prostokątów kolizji, efektów rozpadu asteroid i tła).
    - rtextures.h: Ładowanie i rysowanie tekstur (statku gracza, przeciwników, pocisków, tła, przycisków, serc reprezentujących życie, eksplozji).
    - rtext.h: Obsługa tekstu (wyświetlanie wyniku, numeru fali, informacji o stanie gry, rankingu, ekranu credits).
    - raudio.h: Odtwarzanie dźwięków (strzałów, eksplozji, muzyki w tle).
  - IDE: Visual Studio
  - System kontroli wersji: Git, GitHub
  - Narzędzie do budowania: CMake
  - Google Test do testów jednostkowych
  - System operacyjny: Windows 11 - Środowisko, na którym rozwijano i testowano grę.

Do stworzenia gry inspirowałem się nagraniem **C++ Space Invaders Tutorial with raylib - Beginner Tutorial (OOP)**[8]. Pozwoliło mi to sprawnie rozpocząć projekt i rozdzielić sobie nakład pracy na oddzielne klasy, które odpowiadają za inne funkcje wykonywane w grze. Szczególną inspiracje z tego nagrania wziąłem do stworzenia klas Decay oraz Asteroid. Pozwoliło mi to stworzyć powoli rozpadające się bariery do których strzelają przeciwnicy. Początkowo miały to być tylko tektury które po 30 kolizjach miały znikać ale to rozwiązanie wydawało się bardziej obiecujące do tego typu gry.
Do stworzenia zapisu oraz odczytu uzyskałem pomoc od ChatGPT. Pomógł mi z wyborem odpowiedniej biblioteki do tego oraz wyboru algorytmu do sortowania wyników. Również uzyskałem wsparcie od tego oprogramowania do stworzenia CMakeLists.txt, gdyż pokazał mi jak ma wyglądać przykładowe. Jednakowo zostało to rozbudowane tak, że już nie przypomina w żadnym wypadku tego co zostało mi początkowo pokazane.

5. Opis wykonanych testów (*testing report*) - lista buggów, uzupełnień, itd.

Kod usterki	Data	Autor	Opis	Stan
Zgłoszono wyjątek w lokalizacji 0x0000000000000000 w AstroCruiser.exe: 0xC0000005: Naruszenie zasad dostępu podczas wykonywania w lokalizacji 0x0000000000000000.	11.01	MK	Błąd w odwołaniu się pamięci	Naprawione:
Int-make	19.01	MK	The member function can be made const.	Naprawione: Zmienna została zamieniona na const int
LNK2038	20.01	MK	mismatch detected for 'RuntimeLibrary': value 'MTd_StaticDebug' doesn't match value 'MDd_DynamicDebug' in gtesty.cpp.obj – więcej błędów	Nie naprawiony
LNK2005	20.01	MK	"void __cdecl std::_Facet_Register(class std::_Facet_base *)" (?_Facet_Register@std@@YA	Nie naprawiony

			XPEAV_Facet_base@1@@Z) already defined in msvcprtd.lib(locale0_implib.obj) – błedów tego typu występuje dużo więcej	
C4267	20.01	MK	'initializing': conversion from 'size_t' to 'int', possible loss of data	Ustawiono zmienną jako static_cast<int>
VCR003	25.01	MK	Function 'GetTopScores' can be made static	Nieprawiony lecz nie potrzebny aby naprawić

LISTA BUGÓW:
Znaleziono 25.01.2025r.
Po pokonaniu wszystkich przeciwników pojawiają się nowi lecz ich prędkość jest zwiększana dopiero po odbiciu się od ściany
Status: Nierozwiązane

- **Przykład użycia Google Test:**

Opis przygotowanych testów:

- Klasa **Player**

  PlayerInitialization – poprawne załadowanie tekstury statku sterowanego przez gracza
  PlayerMovement – sprawdzanie pozycji początkowej, ruch gracza w lewo a następnie w prawo i sprawdzanie czy jest na pozycji początkowej oraz wykonywanie w pętli ruchu w prawo aż do krańca ekranu
  PlayerFire – stworzenie obiektu w klasie, wektora przechowującego pociski, wywołanie funkcji Fire() – sprawdzenie czy w wektorze powstała nowa wartość. Sprawdzenie czy pocisk jest wystrzeliwany po opóźnieniu 0.6 sekundy i sprawdzenie ponowne
  PlayerReset – jak wyżej stworzenie obiektu wystrzelenie pocisku a następnie resetuje gracza do początkowej wartości – test polega na sprawdzeniu czy hitboxy tekstury są w odpowiednich miejscach i sprawdza czy wektor pocisków jest opróżniany

- Klasa **Projectile**

  ProjectileInitialization – sprawdzenie poprawnego stworzenia pocisku dla gracza oraz wroga wystrzeliwanego ze zdefiniowanej pozycji oraz prędkości. Sprawdza pozycje hitboxów oraz flage shot – która oznacza czy dany pocisk został stworzony

  ProjectileUpdate – sprawdzanie czy pocisk po wywołaniu funkcji Update zmienia swoją pozycje w zależności czy został wystrzelony przez gracza czy wroga

- Klasa **Enemy**

  EnemyInitialization – inicjalizacja wrogiej jednostki o typie 1 w danej pozycji i w ramach testu ma to sprawdzać za pomocą funkcji GetType() czy typ jednostki został podany poprawnie i sprawdzenie czy powstała ona w poprawnej lokalizacji. Sprawdza również czy poprawna tekstura został wczytana dla danego typu wroga

  EnemyUpdate – stworzenie obiektu w klasie oraz po wywołaniu funkcji Update() sprawdza czy obiekt znajduje się w pozycji o 2 większej w osi X

- Klasa **Asteroid**

  AsteroidInitialization – inicjalizacja asteroidy w danej pozycji i sprawdzenie czy powstała w danym miejscu oraz czy wektor obiektów decays nie jest pusty, gdyż jedna asteroida jest zbudowana z małych bloczków które mogą zostać zniszczone

- Klasa UFO

  UfoSpawn – stworzenie obiektu w klasie UFO, użycie funkcji SpawnUFO(), sprawdzenie czy pozycja y jest równa 90 oraz czy poprawnie pojawił się po lewej lub prawej stronie ekranu

  UfoUpdate – utworzenie ufo, wywołanie funkcji Update() sprawdzenie pozycji, czy się zmieniła

Niestety nie udało mi się dokonać aby biblioteka raylib funkcjonowała poprawnie z gtest od firmy Google. Dostaje mnóstwo błędów o kodzie LNK2038 oraz LNK2005. Po wielogodzinnych próbach nie doznałem pozytywnych rezultatów. Możliwe jest, że któreś zmienne się powielają między moim projektem a googletest i raylib, lecz nie mogłem znaleźć gdzie się to znajduje. Skorzystałem z pomocy internetu oraz ChatGPT – bez pozytywnych rezultatów.

6. Podręcznik użytkownika

- **Uruchamianie gry:**
  - Aby uruchomić grę, należy skompilować kod źródłowy za pomocą kompilatora C++ i make , a następnie uruchomić plik wykonywalny "AstroCruiser".

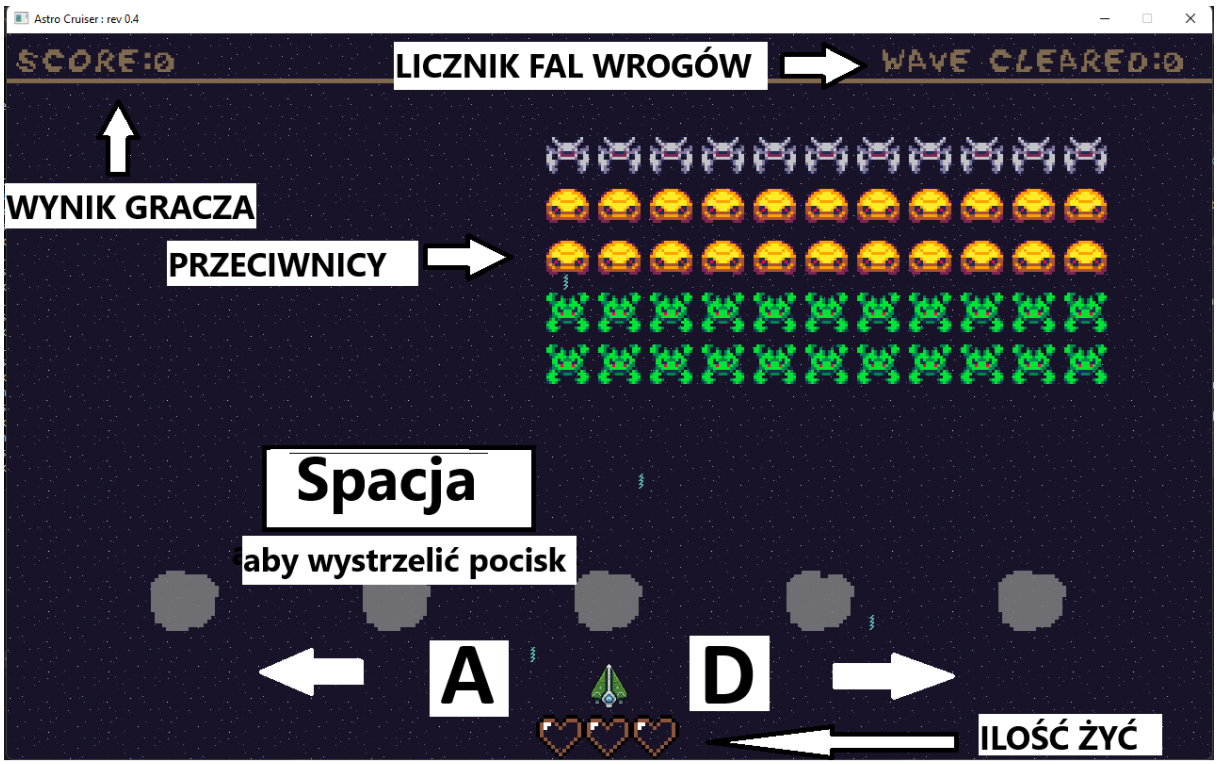
- **Cel gry:**
  - Zestrzel jak najwięcej przeciwników, unikając kolizji z wrogami i pociskami przez nimi wystrzeliwanymi. Po wyczyszczeniu fali przeciwników pojawiają się kolejne a gra staje się co raz trudniejsza. Przeciwnicy przyspieszają nieskończenie. Zdobądź jak najwięcej punktów i przetrwaj jak najdłużej.

  Na repozytorium znajduje się nagranie z rozgrywką z gry.

WYGLĄD MENU GŁÓWNEGO

Z tego miejsca użytkownik dostaje wybór rozpoczęcia rozgrywki, sprawdzenie twórców oraz wyłączenia gry

INTERFEJS UŻYTKOWNIKA



Klawisz A: Ruch statku w lewo, Klawisz D: Ruch statku w prawo, Spacja: Strzał, Klawisz P: Pauza

- **Punktacja:**

  Każdy typ przeciwnika jest inaczej punktowany. Od najbliższego do najdalszego są warci kolejno: 10, 20 oraz 30 punktów. Statek UFO pojawiający się okresowo na ekranie jest warty aż 100 punktów.

- **Ekran gry:**
  - W górnej części ekranu wyświetlany jest aktualny wynik i numer fali.
  - W dolnej środkowej części ekranu jest wyświetlana ilość żyć
  - Statek gracza znajduje się na dole ekranu.
  - Przeciwnicy nadlatują z góry.
  - Asteroidy pojawiają się nad graczem i są używane jako bariery przed pociskami
- **Pauza:**
  - Po wciśnięciu klawisza P gra zostaje zapauzowana.
  - Aby powrócić do gry, należy ponownie wcisnąć ponownie P.
  - Z menu pauzy można wrócić do menu głównego klikając przycisk "Return".
- **Koniec gry:**
  - Gra kończy się po utracie wszystkich żyć.
  - Wyświetlany jest ekran z informacją o wyniku gracza i możliwością restartu po nacisnieciu klawisza Enter

7. Metodologia rozwoju i utrzymania systemu

- **Metodologia:** W projekcie zastosowano podejście iteracyjne. Prace podzielono na mniejsze etapy, z których każdy obejmował implementację, testowanie i dokumentowanie określonych funkcji gry.

- **Zarządzanie kodem źródłowym**: Do zarządzania kodem źródłowym używany był system kontroli wersji Git, a repozytorium kodu umieszczono na platformie GitHub.

Możliwości rozwojowe projektu:

- dodanie nowych funkcji, takich jak power-upy,

- nowe typy przeciwników,

- tryb multiplayer

- poziomy trudności.

- walka z bossem, trudniejszym przeciwnikiem po wyczyszczeniu 5 fal

Bibliografia

[1] Cyganek B.: Introduction to Programming with C++ for Engineers, Wiley-IEEE Press, 2020.

[2] raylib: https://www.raylib.com/

[3] raylib API Documentation: https://www.raylib.com/cheatsheet/cheatsheet.html

[4] C++ Reference: https://en.cppreference.com/

[5] stackoverflow: https://stackoverflow.com

[6] Programming With Nick: Get Started in raylib in 20 minutes! https://www.youtube.com/watch?v=RGzj-PF7D74

[7] Programming With Nick: raylib Buttons Tutorial (OOP) : https://www.youtube.com/watch?v=0Ct9ZWEUm7M

[8] Programming With Nick: C++ Space Invaders Tutorial with raylib - Beginner Tutorial (OOP) : https://www.youtube.com/watch?v=TGo3Oxdpr5o

[9] ChatGPT

[10] Czcionka użyta w grze: "Space Madness" by Rose Frye Licensed under Creative Commons: By Attribution 4.0 International

http://creativecommons.org/licenses/by/4.0/ Źródło: https://modernmodron.itch.io/space-madness