



**Politechnika
Śląska**

PROJEKT INŻYNIERSKI

Implementacja zręcznościowej gry w środowisku Unity na okulary VR
Oculus Quest 2

Bartłomiej GORDON
Nr albumu: 295650

Kierunek: Informatyka, stacjonarne I stopnia inżynierskie
Specjalność: Grafika Komputerowa i Oprogramowanie (GKiO)

PROWADZĄCY PRACĘ
dr Ewa Lach

KATEDRA Grafiki Wizji Komputerowej I Systemów Cyfrowych
Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2024

Tytuł pracy

Implementacja zręcznościowej gry w środowisku Unity na okulary VR Oculus Quest 2

Streszczenie

Implementacja zręcznościowej gry przeznaczonej na urządzenie Oculus Quest 2 stworzonej przy pomocy silnika graficznego Unity i języka programowania C#. Rozgrywka opiera się na precyzyjnym niszczeniu pocisków wystrzeliwanych w stronę gracza z armat przy użyciu broni, takich jak szabla czy pistolet skałkowy. Główną trudnością jest unikanie wystrzeliwanych bomb, których detonacja redukuje punkty życia. System punktacji, faworyzujący poprawne zagrania oraz pociski specjalne pozwalające na spowolnienie czasu, dodatkowo angażują użytkownika. System wyboru dloni do trzymania narzędzi oraz możliwość dostosowania poziomu trudności gry zapewniają dostępność dla graczy o różnych preferencjach i umiejętnościach. Efekty audiowizualne, reagujące zarówno na działania podejmowane przez gracza, jak i zdarzenia zachodzące w samej rozgrywce dodatkowo zwiększą doświadczenia z rozgrywki.

Słowa kluczowe

VR, gra zręcznościowa, Unity

Thesis title

Implementation of an arcade game in the Unity environment for Oculus Quest 2 VR glasses

Abstract

Implementation of an arcade game for the Oculus Quest 2 goggles, created using the Unity graphics engine and the C# programming language. The gameplay is based on the precise destruction of bullets fired at the player from cannons using weapons such as a saber and a flintlock pistol. The main difficulty is avoiding launched bombs, the detonation of which reduces health points. The scoring system that favors correct moves and special projectiles that allow you to slow down time additionally engage the user. The hand selection system for holding tools and the ability to adjust the game's difficulty level ensure accessibility for players with different preferences and skills. Audiovisual effects that respond to both actions taken by the player and events occurring in the game itself further enhance the gameplay experience.

Key words

VR, arcade game, Unity

Spis treści

1 Wstęp	1
1.1 Wkład autora	1
1.2 Charakterystyka rozdziałów	2
2 Analiza tematu	3
2.1 Wirtualna rzeczywistość	3
2.2 Analiza istniejących rozwiązań	4
2.2.1 BeatSaber	4
2.2.2 Superhot VR	4
2.2.3 Fruit Ninja VR	4
2.3 Założenia projektu pracy dyplomowej	6
3 Wymagania i narzędzia	7
3.1 Wymagania funkcjonalne	7
3.1.1 Ogólne	7
3.1.2 Menu główne	7
3.1.3 Armaty	8
3.1.4 Pociski	8
3.2 Wymagania niefunkcjonalne	8
3.3 Metodyka pracy	8
3.4 Narzędzia	10
3.4.1 Unity	10
3.4.2 Język programowania C#	13
3.4.3 Visual Studio 2022	14
3.4.4 Blender	15
3.4.5 GitHub	15
3.4.6 Google Oculus Quest 2	16
4 Specyfikacja zewnętrzna	19
4.1 Wymagania sprzętowe i programowe	19
4.2 Instalacja	19

4.2.1	Instalacja aplikacji na urządzenie	19
4.2.2	Konfiguracja pierwszego uruchomienia	20
4.3	Przebieg gry	24
4.4	Nawigacja	25
4.5	Obsługa menu	26
4.6	Elementy gry	33
4.6.1	Szabla	33
4.6.2	Pistolet skałkowy	33
4.6.3	Pociski wrażliwe na przecięcia	36
4.6.4	Pociski wrażliwe na zestrzelenie	37
4.6.5	Bomba	37
4.6.6	Klepsydra	38
4.6.7	Sztaba złota	41
4.6.8	Serce	41
4.6.9	Elementy świata	41
5	Specyfikacja wewnętrzna	47
5.1	Opis bibliotek i paczek	47
5.1.1	Oculus XR Plugin	47
5.1.2	EzySlice	47
5.1.3	DoTween	47
5.1.4	Addressables	48
5.1.5	Input System	48
5.1.6	Text Mesh Pro	48
5.1.7	Shader Graph	48
5.1.8	Visual Effect Graph	49
5.1.9	Velocity Estimator	49
5.1.10	JsonUtilityEx	49
5.2	Opis wybranych klas	50
5.2.1	Główni zarządcy	50
5.2.2	Armaty i Pociski	55
5.2.3	Bronie	65
5.2.4	Punktacja i tabela wyników	71
5.2.5	Zapis i odczyt danych	74
5.2.6	Klasy użytkowe	75
6	Weryfikacja i walidacja	77
6.1	Metody testowania	77
6.1.1	Konfiguracja środowiska testowego	78
6.2	Identyfikacja problemów	78

7 Podsumowanie i wnioski	79
7.1 Możliwości rozwoju	79
7.1.1 Nowe pociski	79
7.1.2 Tabela wyników dostępna w sieci	79
7.1.3 Wprowadzenie fabuły	80
Bibliografia	83
Spis skrótów i symboli	87
Lista dodatkowych plików, uzupełniających tekst pracy	89
Spis rysunków	93
Spis tabel	95

Rozdział 1

Wstęp

Celem projektu jest stworzenie zręcznościowej gry na gogle VR (wirtualnej rzeczywistości) - Oculus Quest 2. Realizacja tej aplikacji wiąże się z koniecznością precyzyjnego określenia zasad rozgrywki, spójnego graficznie wirtualnego świata oraz implementacji interakcji użytkownika w intuicyjny i satysfakcjonujący sposób. Kluczowym celem jest zapewnienie graczom przyjemności z doświadczenia poprzez starannie zaprojektowaną sferę audiowizualną, która wzbogaca wirtualny świat odpowiednimi dźwiękami i efektami potwierdzającymi skutki zachowań gracza. Istotne jest także wywołanie poczucia rywalizacji z innymi. W projekcie będzie odpowiedzialny za to system punktacji którego kluczowe założenia mają zachęcać gracza do zdobywania jak największych ilości punktów poprzez precyzyjne i przemyślane działania.

Dodatkowym celem towarzyszącym tworzeniu aplikacji było zoptymalizowanie jej pod kątem płynnej rozgrywki. Komfort gracza zanurzonego w wirtualnym świecie jest kluczowym standardem dla współczesnych gier, w szczególności tych, będących związanych z wirtualną rzeczywistością. Wymaga to precyzyjnego doboru struktur danych, algorytmów oraz wysokiej optymalizacji zasobów wykorzystywanych w procesie tworzenia świata wirtualnego.

1.1 Wkład autora

W niniejszej pracy Autor w pełni zaprojektował i stworzył działającą grę zręcznościową przeznaczoną na gogle wirtualnej rzeczywistości. W tym celu wykorzystał silnik graficzny Unity, umożliwiający kompleksową integrację z urządzeniem Oculus Quest 2. Podczas realizacji projektu zastosowano język programowania wysokiego poziomu - C#, w celu stworzenia mechanicznych elementów rozgrywki, które zostały zintegrowane z aspektami graficznymi i dźwiękowymi, w celu dostarczenia innowacyjnego doświadczenia dla użytkownika. Wszystkie zasoby kreatywne, takie jak trójwymiarowe modele, pliki audio czy tekstury, zostały starannie wybrane z odpowiednich źródeł, posiadających licencje umożliwiające na wykorzystanie ich w projekcie. Wkład Autora opierał się również na

intensywnych testach aplikacji, zarówno w fazie rozwoju, jak i w środowisku końcowym, co zagwarantowało eliminację błędów znalezionych podczas implementacji oraz zoptymalizowaną stabilność i komfort użytkowania aplikacji.

1.2 Charakterystyka rozdziałów

Praca składa się z siedmiu rozdziałów, które stanowią kompleksowe omówienie tematyki podjętej w pracy. W rozdziale pierwszym jasno zarysowany jest cel projektu oraz szczegółowy wkład autora w proces implementacji. W rozdziale drugim analizowany jest obszar technologii wirtualnej rzeczywistości, a także współczesne podejście do gier zręcznościowych w tym medium.

Trzeci rozdział koncentruje się na wymaganiach, prezentując założenia aplikacji końcowej oraz metodologię tworzenia oprogramowania, wraz z opisem użytych narzędzi.

W rozdziale czwartym znajdują się informacje o procesie instalacji, precyzyjnie opisujące kroki niezbędne do uruchomienia aplikacji na goglach VR - Oculus Quest 2. Przedstawiony został również szczegółowy opis rozgrywki, zarysujący wszelkie dostępne w grze mechaniki wraz z dokładną prezentacją interfejsu. Rozdział piąty koncentruje się na aspektach technicznych aplikacji. W ramach tego rozdziału dokonano dogłębnej analizy różnych bibliotek, pakietów oprogramowania oraz wybranych klas zawartych w kodzie aplikacji niezbędnych do funkcjonowania modułów tworzących projekt.

Rozdział szósty opisuje proces testowania aplikacji w trakcie jej tworzenia oraz skutki znalezionych błędów na rozwój projektu.

Ostatni rozdział stanowi podsumowanie, prezentując wnioski wyciągnięte podczas implementacji projektu. Pokazuje on również możliwe ścieżki dalszego rozwoju oprogramowania.

Rozdział 2

Analiza tematu

W poniższym rozdziale poddana jest analizie wirtualna rzeczywistość wraz z jej aktualnymi trendami w postaci przykładowych gier zręcznościowych. W dalszej części zostały omówione założenia projektu w kontekscie innych tytułów o podobnej tematyce.

2.1 Wirtualna rzeczywistość

Wirtualna rzeczywistość jest technologią, która umożliwia użytkownikom interakcję z generowanym komputerowo środowiskiem, które symuluje realistyczne doświadczenia sensoryczne. Ta technologia wykorzystuje zestaw urządzeń, takich jak gogle VR (Rys. 2.1), które zanurzają użytkownika w wirtualnym świecie, często wykorzystując efekty dźwiękowe i wizualne. Są to urządzenia, które po założeniu na głowę pozwalają użytkownikowi oglądać wirtualne środowisko. Wyposażone są w ekranы wyświetlające obrazy oraz w przypadku niektórych modeli, we wbudowane sensory śledzące ruch.

Kontrolery odpowiadają za interakcję użytkownika z wirtualnym światem, umożliwiając manipulowanie obiekktami, wykonywanie ruchów czy interakcję z otoczeniem. Na przy-



Rysunek 2.1: Przedstawienie gogli VR od różnych producentów, źródło: [30].

kład, kontrolery dołączone do gogli wirtualnej rzeczywistości Oculus Quest 2 wyposażono w ergonomiczne wsparcie dla kciuka oraz udoskonalone funkcje dotykowe, co przyczynia się do znaczącego polepszenia doświadczeń użytkownika. Dodatkowo, w porównaniu z modelami konkurencyjnymi, wykazują one wydłużoną żywotność baterii, stanowiąc istotny postęp w dziedzinie wydajności i użytkowania [25]. Wirtualna rzeczywistość znajduje zastosowanie w obszarach takich jak:

- **Rozrywka** - gry wirtualnej rzeczywistości oferują unikalne doznania i emocje, które są trudne do osiągnięcia w tych tradycyjnych.
- **Szkolenie i edukacja** - VR znajduje zastosowanie w symulacjach szkoleniowych, medycynie, architekturze czy edukacji, umożliwiając interaktywne i realistyczne scenariusze.
- **Przemysł i projektowanie** - firmy wykorzystują VR do prototypowania, projektowania i symulacji, co pozwala na lepsze zrozumienie i wizualizację produktów czy procesów.

2.2 Analiza istniejących rozwiązań

2.2.1 BeatSaber

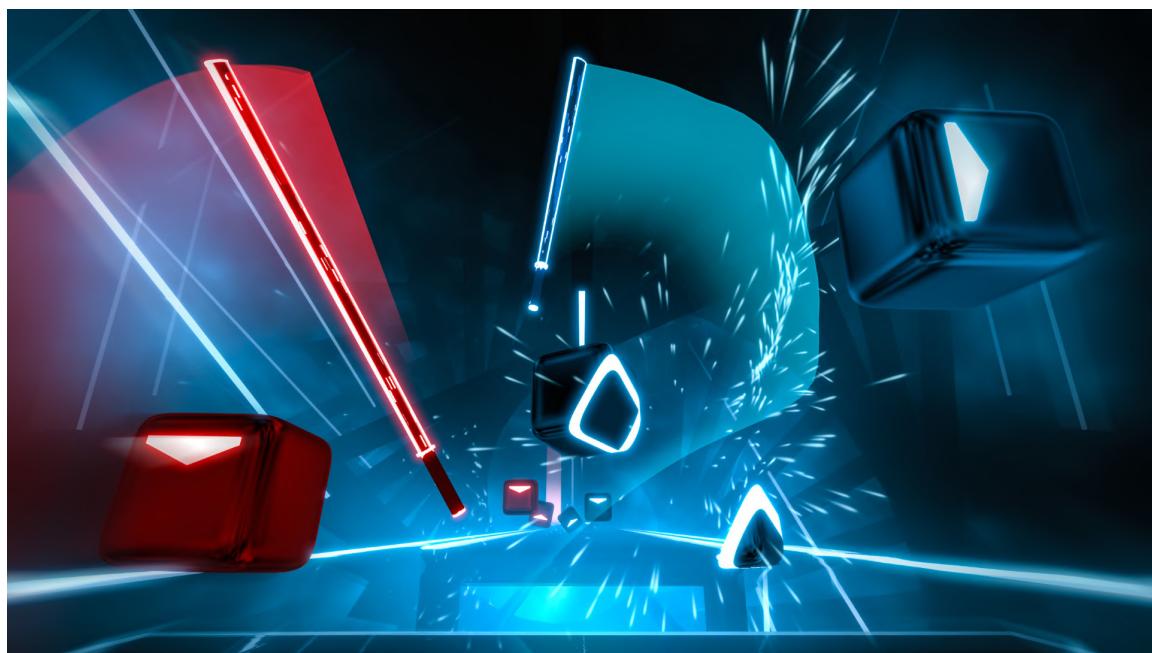
BeatSaber [4] to płatna dynamiczna gra rytmiczna VR stworzona przez studio Beat Games, która łączy elementy muzyki, szybkich ruchów i precyzyjnej interakcji. Gracze za pomocą kontrolerów przecinają oświetlone bloki rytmicznie poruszając się do muzyki. Gra zdobyła popularność dzięki prostocie swojego konceptu oraz intensywnej rozgrywce (Rys. 2.2).

2.2.2 Superhot VR

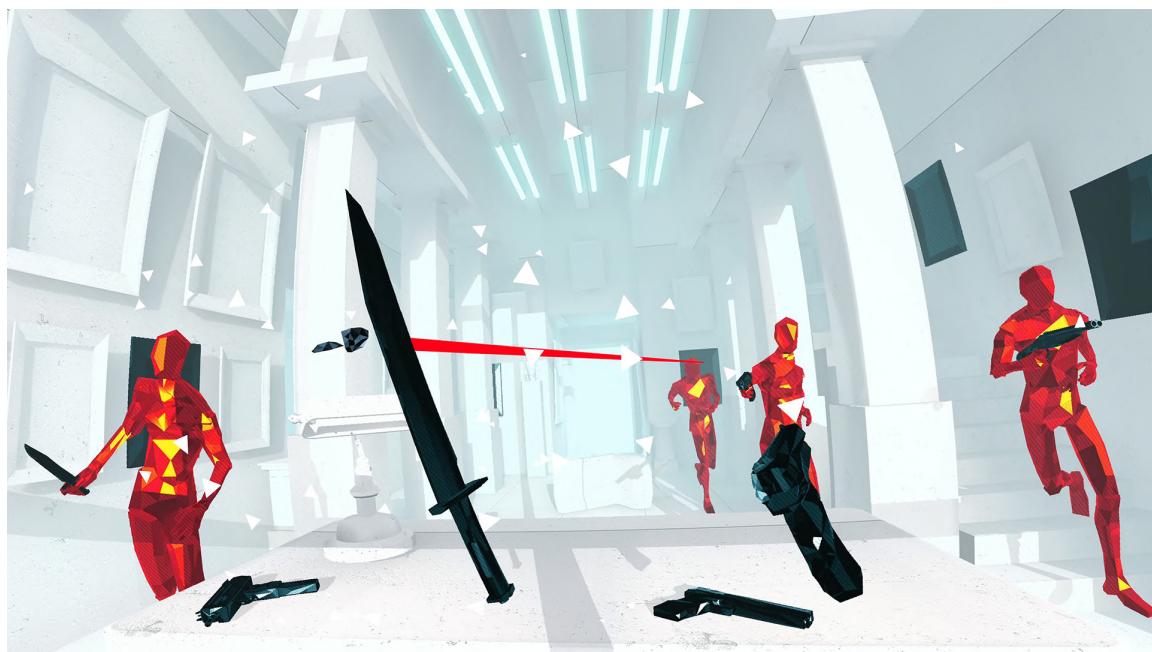
Gra SuperHot VR [24], stworzona przez polskie studio SUPERHOT Team, jest płatną produkcją oferującą innowacyjną mechanikę rozgrywki. Jej wyjątkowość polega na niezwykłym sposobie kontrolowania czasu, który zatrzymuje się, gdy gracz nie wykonuje ruchu, co wymusza strategiczne planowanie akcji. Grając w SuperHot VR, użytkownik musi w przemyślany sposób unikać nadlatujących pocisków oraz sprawnie atakować wrogów, co sprawia, że łączy w sobie elementy strategicznego myślenia i intensywnej akcji (Rys. 2.3).

2.2.3 Fruit Ninja VR

Gra Fruit Ninja VR [15], opracowana przez Halfbrick Studios, jest płatną produkcją, która przenosi popularną grę mobilną do świata wirtualnej rzeczywistości, umożliwiając



Rysunek 2.2: Zrzut ekranu przedstawiający fragment rozgrywki gry BeatSaber, źródło: https://store.steampowered.com/app/620980/Beat_Saber/.



Rysunek 2.3: Zrzut ekranu przedstawiający fragment rozgrywki gry Superhot VR, źródło: https://store.steampowered.com/app/617830/SUPERHOT_VR/.



Rysunek 2.4: Zrzut ekranu przedstawiający fragment rozgrywki gry Fruit Ninja VR, źródło: https://store.steampowered.com/app/486780/Fruit_Ninja_VR/.

graczom krojenie owoców za pomocą kontrolerów VR. Ta adaptacja charakteryzuje się prostą, ale satysfakcjonującą mechaniką rozgrywki, która angażuje użytkownika w dynamiczną interakcję z wirtualnym otoczeniem (Rys. 2.4).

2.3 Założenia projektu pracy dyplomowej

W trakcie rozgrywki z gry stworzonej w ramach pracy dyplomowej, gracz znajduje się na tratwie dryfującej po morzu, podczas gdy z pobliskiej wyspy oraz statku pirackiego skierowane są w jego kierunku salwy pocisków z wielu armat jednocześnie. Zadanie gracza polega na precyzyjnym odparciu ataków przy pomocy posiadanych broni. Niektóre z pocisków są podatne na unieszkodliwienie poprzez przecięcie, co wymaga użycia szabli, podczas gdy inne można zestrzelić wyłącznie za pomocą pistoletu. Wybór niewłaściwego rodzaju broni w kontakcie z danym pociskiem skutkuje karą, która w zależności od wybranego poziomu trudności, powoduje utratę punktów lub wyzerowanie mnożnika.

Oprócz precyzyjnego dobierania broni do niszczenia różnych rodzajów pocisków, gracz stoi również przed kolejnym taktycznym wyzwaniem w postaci specjalnych pocisków, które są podatne na oba typy obrażeń. Przykładem jednego z nich są bomby, których zniszczenie generuje falę uderzeniową, odpychającą wszystkie nadlatujące obiekty w stronę gracza i odejmującą jeden punkt z jego zdrowia. Inne specjalne pociski to na przykład klepsydra, która spowalnia czas, zmieniając dynamikę rozgrywki, lub serce, które odnawia jeden punkt życia gracza, niwelując konsekwencje błędnych decyzji podjętych wcześniej.

Rozdział 3

Wymagania i narzędzia

Rozdział ten skupia się na prezentacji wymagań postawionych w początkowym etapie opracowywania projektu tworzonego w ramach pracy dyplomowej, postawionych po szczegółowej analizie problemu. Przedstawione zostały również narzędzia użyte przy implementacji projektu.

3.1 Wymagania funkcjonalne

3.1.1 Ogólne

- Stworzenie trzech różnych poziomów trudności.
- Stacjonarna rozgrywka bez konieczności poruszania się.

3.1.2 Menu główne

- Wybór poziomu trudności.
- Wybór pseudonimu.
- Wybór preferowanej ręki do trzymania szabli, zgodnie z dominującą dlonią użytkownika.
- Możliwość zmiany poziomu głośności gry w ustawieniach.
- Automatyczny zapis punktów i pseudonimu gracza do tabeli wyników.
- Implementacja możliwości zmiany typu obrotu przy użyciu kontrolera (rotacja stopniowa lub ciągła).
- Możliwość włączenia trybu zwiększonego komfortu w ustawieniach.

3.1.3 Armaty

- Wystrzeliwanie pocisków po wcześniej wyliczonym rzucie ukośnym w stronę gracza.
- Ciągły obrót modelu armaty zgodny z trajektorią pocisku.

3.1.4 Pociski

- Dynamiczne przecinanie obiektów wrażliwych na szable.
- Destrukcja obiektów wrażliwych na pociski.
- Spersonalizowana dla każdego obiektu tekstura przecięcia.
- Spersonalizowana dla każdego pocisku ilość punktów za jego zniszczenie.
- Pojawienie się zawieszonego chwilowo w przestrzeni tekstu pokazującego ilość zdobytych punktów po zniszczeniu pocisku.

3.2 Wymagania niefunkcjonalne

- **Immersja** - rozgrywka zapewnia głębokie zaangażowanie użytkownika, umożliwiając pełne zanurzenie w wirtualnym świecie.
- **Łatwość obsługi** - prostotę i intuicyjność interakcji poprzez ograniczenie ilości przycisków potrzebnych do obsługi gry.
- **Stabilność** - zapewnienie ciągłego działania gry bez spadków liczby klatek na sekundę (FPS), eliminując potencjalne zagrożenia dla komfortu użytkownika, takie jak dezorientacja lub doznanie mdłości.
- **Wydajność** - dzięki wysokiej optymalizacji aplikacji, możliwa jest bezpośrednia rozgrywka na goglach VR. Eliminuje to potrzebę przekazywania danych gry przez komputer za pośrednictwem technologii transmisji.
- **Komfort** - zminimalizowanie czynników wywołujących uczucie mdłości takich jak gwałtowne ruchy kamery lub nadmierna ilość bodźców wizualnych, zapewniając tym samym wygodne doświadczenia użytkownikom.

3.3 Metodyka pracy

W ramach pracy dyplomowej, zdecydowano się na tworzenie projektu zgodnie z metodyką Kanban, która wywodzi się z japońskiego przemysłu samochodowego w latach 40. XX wieku i stała się znaczącą strategią zarządzania przepływem pracy i optymalizacji

procesów biznesowych. Literackie tłumaczenie słowa „Kanban” oznacza „wizualną kartę” lub „tablicę”, co oddaje istotę tego podejścia - wykorzystanie wizualnych narzędzi do monitorowania i zarządzania procesami produkcyjnymi, biznesowymi lub zadaniami [18]. Podstawowe zasady Kanban obejmują:

- **Wizualizacja pracy** - tablica Kanban stanowi centralny element, na którym zespoły mogą wizualnie śledzić przepływ pracy. Składa się z kolumn reprezentujących różne etapy procesu, a zadania są reprezentowane przez karty lub notatki.
- **Limitowanie pracy w toku** - ograniczenie liczby zadań, które mogą być wykonywane jednocześnie, ma na celu zwiększenie wydajności poprzez skupienie uwagi na ukończeniu bieżących zadań przed rozpoczęciem nowych.
- **Zarządzanie przepływem** - Kanban kładzie nacisk na płynny przepływ pracy poprzez przypisanie zadań do poszczególnych etapów procesu.

Typowa tablica Kanban składa się z trzech głównych kolumn:

- **Do zrobienia** - znajdują się tam zadania, które zostały zaplanowane do wykonania. Zadania te oczekują na przypisanie lub przesunięcie do kolejnego etapu.
- **W trakcie** - kolumna ta zawiera zadania, które są aktualnie wykonywane. Limitowana jest tu ilość pracy, aby zapobiec przeciążeniu i utracie wydajności.
- **Zrobione** - po zakończeniu zadania przenosi się je do tej kolumny. Jest to miejsce, w którym zespół oznacza zadanie jako ukończone.

Korzyściami płynącymi z korzystania z metodyki Kanban są na przykład:

- Poprawa komunikacji.
- Elastyczność i adaptacyjność.
- Optymalizacja przepływu pracy.
- Ciągłe doskonalenie.

Podsumowując, tablica Kanban jest narzędziem, które nie tylko usprawnia zarządzanie projektem, ale także promuje transparentność, ciągłe doskonalenie i efektywność w pracy zespołowej.

3.4 Narzędzia

3.4.1 Unity

Silnik Unity (Rys. 3.1) w wersji 2022.3.11f został wykorzystany jako główne narzędzie w realizacji tego projektu. Jest to jedno z najbardziej zaawansowanych oprogramowań dedykowanych dla twórców gier i aplikacji zarówno na platformy desktopowe, mobilne, jak i konsolowe. Silnik zajmuje przodującą pozycję w branży [29] gwarantując tym samym ogromną ilość materiałów dydaktycznych w sieci. Wszechstronność silnika jest jego kluczową cechą pozwalającą tworzyć zarówno projekty trójwymiarowe, jak i dwuwymiarowe przy korzystaniu z tego samego interfejsu graficznego [29]. Unity posiada rozbudowany edytor, który ułatwia proces tworzenia i testowania aplikacji w jednym zintegrowanym środowisku. Tworzenie aplikacji trójwymiarowych w silniku Unity odbywa się poprzez interakcję z obiektami gry na scenach.



Rysunek 3.1: Logo silnika Unity, źródło: www.unity.com

Sceny w tym konkretnym środowisku odnoszą się do przestrzeni, w której projektowana jest dana gra lub aplikacja. Na scenach można umieszczać różne obiekty takie jak: modele 3D, oświetlenie, kamery, emitery dźwięku, a także interfejs użytkownika, pozwalają one również programistom na wszelaką modyfikację tych właśnie obiektów, jak i tworzenie własnych, niestandardowych, które rozwiążają konkretne problemy danego produktu.

Istotnym aspektem silnika i cechą obiektów jest fakt, że każdy z nich należy do klasy *GameObject*, która sama z siebie nie ma żadnych właściwości. Programista sam musi nadać jej konkretne cechy, podłączając do niej różne komponenty które są podstawą architektury aplikacji tworzonej przy pomocy silnika Unity [8]. Każdy komponent dziedziczy po klasie *MonoBehaviour*, która udostępnia programowalny interfejs reagujący na zdarzenia silnika takie jak pojawienie się obiektu na scenie lub zmianę jego stanu. Mogą one reprezentować różne aspekty obiektu, takie jak wygląd, zachowanie fizyczne, logikę sterowania czy interakcje z użytkownikiem. Przykładowe komponenty obejmują *Transform*, który określa położenie, obrót i skalę obiektu, *Collider*, który definiuje obszar kolizji, a także skrypty programistyczne, które pozwalają na tworzenie niestandardowych zachowań. W silniku graficznym Unity, prefabrykaty (*prefab*) odgrywają istotną rolę, stanowiąc gotowe wzorce obiektów lub grup obiektów, skonfigurowane już zgodnie z naszymi potrzebami. Korzystanie z prefabrykatów umożliwia wielokrotne wykorzystanie tych samych elementów w grze. Dokonując zmian w prefabrykacie, modyfikacje odnoszą się do wszystkich instancji tego

prefabrykatu. Ważne jest zaznaczenie, że w kontekście elementów interfejsu użytkownika wszystkie obiekty takie jak, na przykład przyciski, tekst, panele czy obrazy, muszą być umieszczone na specjalnym obiekcie kanwy (*canvas*).

Skrypty stanowią fundamentalny element procesu tworzenia w interaktywnym środowisku Unity. Sama struktura skryptu skupia się wokół języka programowania *C#* i klasy *MonoBehaviour*, która stanowi swoisty szkielet każdego skryptu w Unity, umożliwiając dostęp do funkcji cyklu życia obiektów, takich jak *Start()*, *Update()* czy *FixedUpdate()*, które determinują zachowanie się obiektów w czasie rzeczywistym [29]. Unity wykorzystuje również koncepcję korutyn (*Coroutine*), umożliwiających asynchroniczne działania w czasie rzeczywistym. Korutyny pozwalają na wykonywanie operacji, które mogą zajmować znaczącą część czasu, takich jak animacje czy ładowanie zasobów, bez blokowania działania pozostałych elementów aplikacji.

Interfejs silnika

W tej sekcji zostanie opisany interfejs silnika graficznego Unity (Rys. 3.2) w jego standardowym wyglądzie i ułożeniu [29].

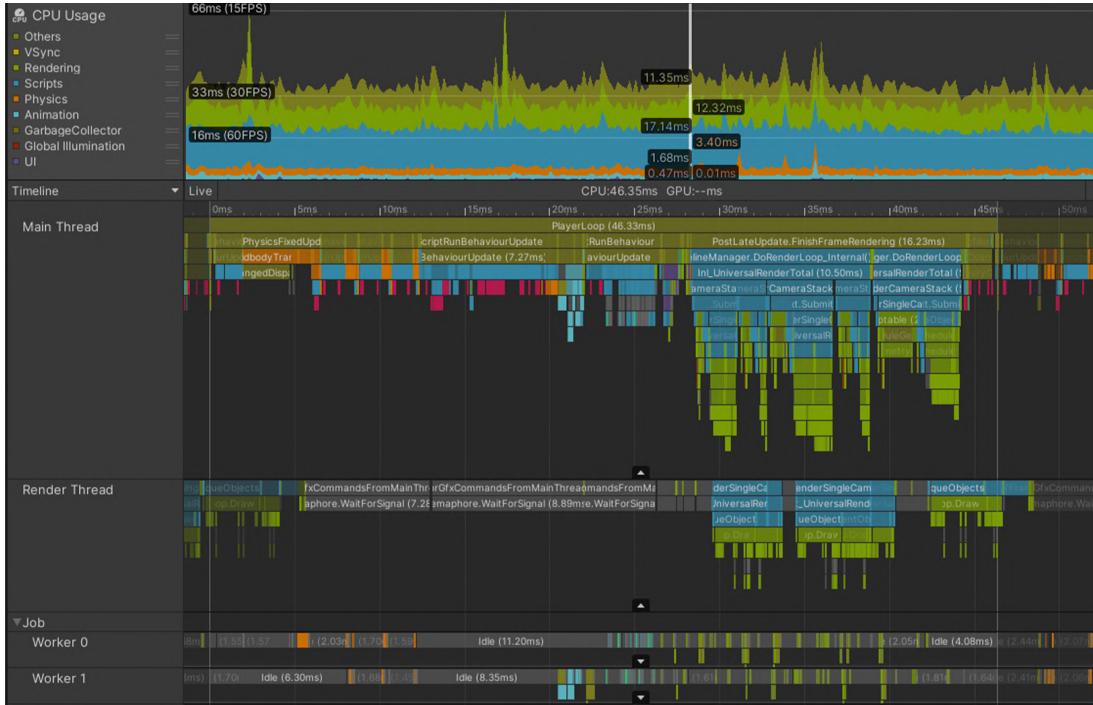


Rysunek 3.2: Edytor Unity

- **(A) Pasek narzędzi** zapewnia dostęp do konta Unity, a także do przycisków odpowiedzialnych za wejście w tryb rozgrywki, historii zmian, narzędzi do wyszukiwania, warstw widoczności i przycisku do zmiany ułożenia okien.
- **(B) Hierarchia** wyświetla wszystkie obiekty gry znajdujące się na scenie. Zapewnia także podgląd tego, w jakie sposób poszczególne obiekty gry znajdują się w relacjach względem siebie.

- (C) **Widok gry** symuluje i wyświetla w jaki sposób gra będzie wyglądać przez główną kamerę na scenie. Symulacja rozpoczyna się po naciśnięciu przycisku odpowiedzialnego za przejście w tryb rozgrywki.
- (D) **Widok sceny** pozwala na swobodną nawigację po scenie i jej edycję. Ma możliwość wyświetlania w trybie dwuwymiarowym, jak i trójwymiarowym, w zależności od projektu, nad którym się pracuje.
- (E) **Nakładka** zawiera podstawowe narzędzia, dzięki którym można manipulować obiektami gry na scenie.
- (F) **Okno inspektora** pozwala na przeglądanie i edycję komponentów znajdujących się na wybranym przez nas obiekcie gry.
- (G) **Okno projektu** wyświetla własną bibliotekę zasobów, z których można korzystać w danym projekcie. Przy importie nowych zasobów do projektu, będą one wyświetlać się właśnie w tym miejscu.
- (H) **Pasek statusu** zapewnia powiadomienia dotyczące różnych procesów odbywających się w tle.

Do monitorowania i optymalizacji wydajności aplikacji w silniku Unity wykorzystano zintegrowane narzędzie Profiler. Ten instrument pozwala gromadzić i prezentować dane dotyczące wydajności procesora, zużycia pamięci, renderowania obrazu i dźwięku poprzez wykresy, na których można zaobserwować spadki wydajności (Rys. 3.3). Dzięki możliwości uruchomienia narzędzia na urządzeniach podłączonych do komputera, można wykonać analizę wydajnościową już na samych goglach przeznaczonych do wirtualnej rzeczywistości oraz precyzyjne przetestować aplikację na docelowej platformie. Narzędzie Profiler jest użyteczne do identyfikowania obszarów aplikacji, które wymagają poprawy pod względem wydajności, takich jak kod, zarządzanie zasobami czy ustawienia sceny oraz kamery.



Rysunek 3.3: Interfejs narzędzia Profiler, źródło: [27].

3.4.2 Język programowania C#

Język programowania C# od swojego powstania przeszedł przez wiele etapów ewolucji, stając się jednym z głównych języków stosowanych w programowaniu aplikacji na platformę .NET [22]. Pierwsza wersja C# została wprowadzona przez firmę Microsoft w 2000 roku, a od tego czasu język ten stale ewoluował, dodając nowe funkcje, usprawniając wydajność i dostarczając programistom narzędzi do tworzenia bardziej skalowalnych i efektywnych aplikacji.

Ważnymi etapami w rozwoju języka C# było wprowadzenie Language-Integrated Query (LINQ) [7] w wersji 3.0, które znaczowo ułatwiało manipulację danymi oraz programowanie zapytań.

Znalazł on szerokie zastosowanie w różnorodnych dziedzinach, dzięki swojej wszechstronności i silnemu wsparciu ze strony platformy .NET. Kilka kluczowych obszarów wykorzystania obejmuje:

- Tworzenie aplikacji wyświetlanego na pulpicie.
- Programowanie gier.
- Tworzenie aplikacji webowych.
- Rozwój aplikacji mobilnych.
- Rozwiązania zaplecza stron internetowych.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class NewBehaviourScript : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10
11
12
13      // Update is called once per frame
14      void Update()
15      {
16
17
18
19 }
```

Rysunek 3.4: Przykładowy skrypt napisany w języku C# zintegrowany z silnikiem Unity

W silniku Unity język C# ma zastosowanie głównie przy tworzeniu klas reprezentowanych przez środowisko jako skrypty (Rys. 3.4).

Przyszłość języka C# wydaje się obiecująca, zwłaszcza w kontekście ciągłego rozwoju platformy .NET oraz staranności firmy Microsoft we wprowadzaniu innowacji. Planowane są kolejne ulepszenia związane z wydajnością, funkcjonalnością oraz integracją z chmurą, co sprawia, że C# nadal będzie atrakcyjnym wyborem dla programistów w różnych dziedzinach.

3.4.3 Visual Studio 2022

Jest to zintegrowane środowisko programistyczne przygotowane przez firmę Microsoft stanowiące fundament do tworzenia oprogramowania. Swoją przodującą pozycję na rynku [28] podobnych produktów takich jak na przykład: ‘Rider’, ‘VSCode’ lub ‘Eclipse’ zawdzięcza między innymi ciągłemu doskonaleniu bezpieczeństwa, wydajności i komfortu tworzenia oprogramowania.

Wybór tego środowiska był podyktowany cechami wyróżniającymi je od innych produktów takimi jak:

- Wygodne debugowanie kodu.
- Wysoka integracja z silnikiem Unity.
- Podpowiedzi w kodzie, również do zawartości bibliotek silnika Unity.

- Wyostrzona dokładność informacji o potencjalnym błędzie.
- Pomocne uzupełnianie fragmentów kodu zasilane przez sztuczną inteligencję.

Integracja z silnikiem Unity odbywa się poprzez doinstalowanie do środowiska dodatkowych pakietów wspierających tę opcję i ustawienie środowiska jako domyślnego w edytorze silnika.

3.4.4 Blender

Blender (Rys. 3.5) to kompleksowe oprogramowanie do modelowania, animacji, renderowania oraz tworzenia treści w środowisku 3D. Jest dostępny jako darmowe narzędzie z licencją open-source [33], co przyczynia się do jego popularności wśród społeczności twórców grafiki trójwymiarowej. Blender oferuje szeroki zakres funkcji, umożliwiając użytkownikom między innymi:

- Tworzenie złożonych modeli 3D.
- Podgląd zmian w czasie rzeczywistym.
- Animowanie obiektów.
- Tworzenie skomplikowanych sekwencji ruchu.
- Tworzenie symulacji fizycznych.



Rysunek 3.5: Logo oprogramowania Blender, źródło: www.blender.org

3.4.5 GitHub

GitHub jest platformą internetową umożliwiającą kontrolę wersji oprogramowania z wykorzystaniem systemu Git, jak i współpracę nad projektem. Dzięki interfejsowi użytkownika opartemu na przeglądarce, GitHub zapewnia intuicyjne narzędzia do udostępniania przestrzeni dyskowej na serwerze w postaci repozytoriów przechowujących projekt pozwalających jednocześnie na wygodne przeglądanie kodu i zgłaszanie problemów.



Rysunek 3.6: Gogle Oculus Quest 2 wraz z dwoma kontrolerami, źródło:
<https://www.meta.com/gb/quest/safety-center/quest-2/>

GitHub Actions

GitHub Actions jest to narzędzie przygotowane przez GitHub do automatyzacji procesów związanych z repozytoriami. Zastosowanie tego mechanizmu umożliwia inicjowanie różnorodnych sekwencji działań w odpowiedzi na konkretne wydarzenia, takie jak wprowadzenie nowych zmian lub zgodnie z ustalonym harmonogramem. To narzędzie pozwala programistom na wykonywanie testów, budowanie aplikacji w chmurze lub wysyłanie powiadomień z określonymi informacjami – wszystko w zautomatyzowany sposób.

"Narzędzie to stale kompliluje przyrostowe zmiany kodu wprowadzone przez programistów, uruchamia automatyczne testy i weryfikuje wdrażanie aplikacji na serwerach ulepszając tym oprogramowanie, jakość i produktywność"[19].

3.4.6 Google Oculus Quest 2

Oculus Quest 2 (Rys. 3.6) to samodzielne urządzenie VR, które nie wymaga podłączenia do komputera ani zewnętrznych czujników do śledzenia ruchu. Jest oparte na systemie operacyjnym Android, co sprawia, że możliwe jest korzystanie z narzędzi programistycznych, takich jak Android Data Bridge, do interakcji z urządzeniem poprzez interfejs wiersza poleceń.

Specyfikacja

Specyfikacja gogli ze strony producenta [21]:

- **Typ zestawu** - bezprzewodowy
- **CPU** - Snapdragon XR2, octa-core, Kryo 585 (1 x 2.84 GHz, 3 x 2.42 GHz, 4 x 1.8 GHz), 7 nm process technology
- **GPU** - Adreno 650
- **RAM** - 6 GB

- **System operacyjny** - Android 10
- **Połączenie** - WiFi 6, WiFi Streaming, 5.0 LE, USB-C 3.0
- **Wyświetlacz** - Single Fast switch LCD
- **Rozdzielcość** - 3664 x 1920 pikseli (1832 x 1920 pikseli na jedno oko)
- **Częstotliwość odświeżania** - 60, 72, 90 Hz
- **Sensory** - Żydroskop, Akcelerometr, Magnetometr

Android Data Bridge

Android Data Bridge (ADB) jest narzędziem deweloperskim zaprojektowanym dla systemu Android, umożliwiającym komunikację między komputerem a urządzeniem z Androidem. Pozwala na debugowanie, instalowanie i zarządzanie aplikacjami na urządzeniu Android z poziomu komputera.

Rozdział 4

Specyfikacja zewnętrzna

Rozdział ten skupia się na wymaganiach sprzętowych potrzebnych do uruchomienia aplikacji, a także metod instalacji produktu na gogle wirtualnej rzeczywistości. Dodatkowo został przedstawiony pełny zakres obsługi aplikacji.

4.1 Wymagania sprzętowe i programowe

Wymagania minimalne do uruchomienia aplikacji obejmują:

- Gogle Oculus Quest 2 z oprogramowaniem *Quest* w wersji 39 lub wyższej [9].
- Przestrzeń dyskowa na urządzeniu VR o minimalnej wielkości 100MB.
- Kabel USB-C, który umożliwia przesyłanie danych, jest niezbędny do podłączenia urządzenia Oculus Quest 2 do komputera. Jest stosowany wyłącznie podczas instalacji.

4.2 Instalacja

4.2.1 Instalacja aplikacji na urządzenie

Aby prawidłowo zainstalować aplikację, konieczne jest pobranie najnowszej wersji sterowników ADB (opisanych w rozdziale 3.4.6) ze strony producenta dla wybranego systemu operacyjnego [6]. Po wypakowaniu katalogu można przejść do weryfikacji poprawności przeprowadzonych czynności poprzez użycie terminala systemowego (Rys. 4.1 i 4.2).

Kolejnym etapem jest uruchomienie gogli Oculus Quest 2 oraz przeprowadzenie podstawowej konfiguracji konta użytkownika. Następnie konieczne będzie pobranie aplikacji Oculus zarówno na komputer stacjonarny, jak i urządzenie mobilne z oficjalnej strony producenta [20]. Po procesie powiązania gogli z kontem, konieczne będzie włączenie trybu

```
1 #Przejscie do folderu ze sterownikiem
2 $ cd <lokalizacja folderu platform-tools>
3 #Sprawdzenie wersji sterownika
4 $ adb version
```

Rysunek 4.1: Sprawdzenie wersji sterownika.

```
1 Android Debug Bridge version 1.0.41
2 Version 34.0.5-10900879
3 Installed as C:\platform-tools\adb.exe
4 Running on Windows 10.0.19045
```

Rysunek 4.2: Oczekiwany przykładowy rezultat sprawdzenia wersji sterownika.

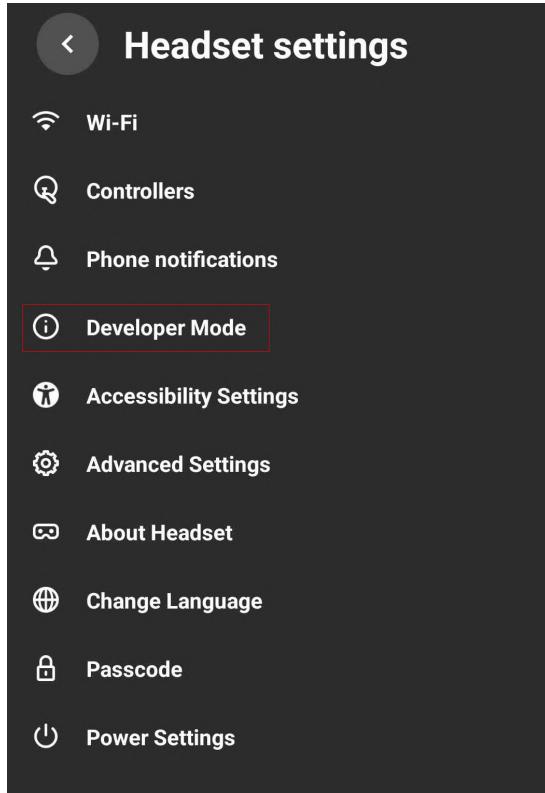
deweloperskiego w aplikacji mobilnej (Rys. 4.3), a także zezwolenie na instalację z nieznanego źródła w przypadku aplikacji na komputerze stacjonarnym (Rys. 4.4). Następnym krokiem będzie fizyczne podłączenie gogli do komputera za pomocą kabla USB-C, który umożliwia wymianę danych (Rys. 4.5). Poprawność wykonanych czynności można zweryfikować, korzystając z terminala systemowego (Rys. 4.6).

Ostatnim etapem jest przeprowadzenie instalacji strumieniowej na urządzenie. W celu osiągnięcia tego celu konieczne jest skopiowanie bezwzględnej ścieżki do instalatora aplikacji o rozszerzeniu pliku *.apk* zlokalizowanego w folderze *Build* (Rys. 4.7). Weryfikacja poprawności wykonanych działań możliwa jest poprzez użycie terminala systemowego (Rys. 4.8).

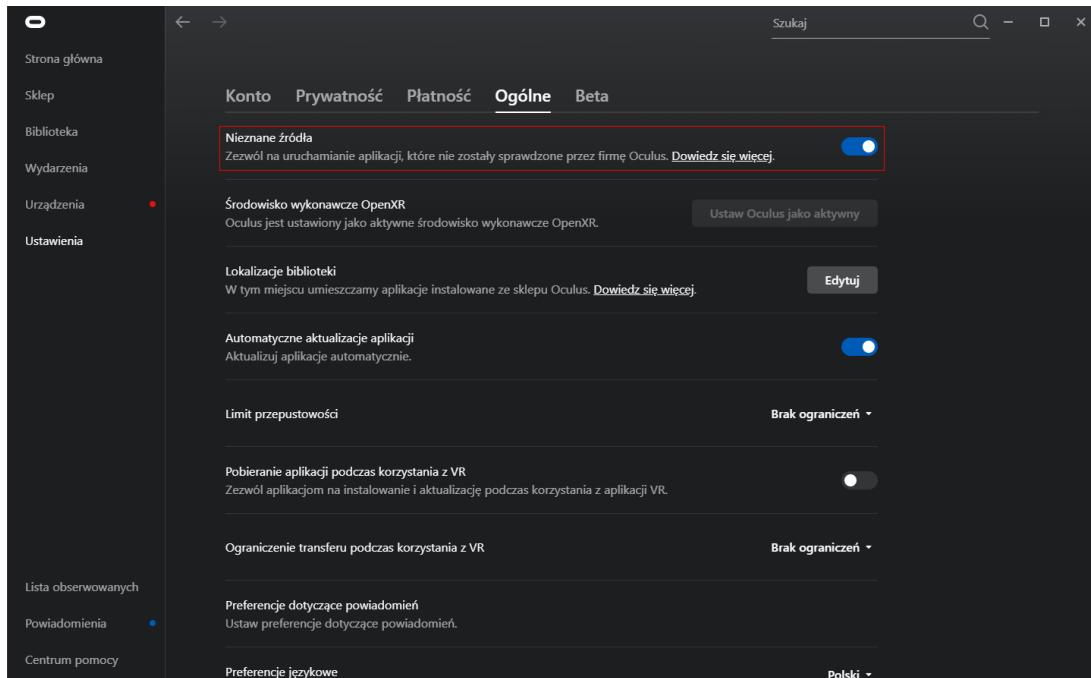
W przypadku poprawnego wykonania powyższych kroków, gogle można bezpiecznie odłączyć od komputera.

4.2.2 Konfiguracja pierwszego uruchomienia

W celu inicjacji aplikacji na goglach Oculus Quest 2, należy skorzystać z interfejsu użytkownika, który obejmuje pasek narzędzi znajdujący się poniżej linii wzroku użytkownika. Po zalogowaniu do urządzenia, należy wybrać kafelek oznaczony jako *App library*. Ten kafelek przeniesie użytkownika do biblioteki zainstalowanych aplikacji, gdzie można przeglądać dostępne opcje (Rys. 4.9). Kolejnym krokiem jest zezwolenie na dostęp urządzenia do aplikacji nieweryfikowanych przez oficjalnego dostawcę (w przypadku Oculus Quest 2 jest to firma Meta), poprzez rozwinięcie menu znajdującego się w górnej prawej części ekranu. Użytkownik powinien wybrać opcję oznaczoną jako *Unknown sources*. Wybór tej opcji pozwoli na wyświetlenie ukrytych aplikacji (Rys. 4.10). Ostatecznym etapem procedury jest wybór konkretnej aplikacji, którą użytkownik chce uruchomić. Po znalezieniu wcześniej zainstalowanej aplikacji w bibliotece, należy ją wybrać w celu jej uruchomienia (Rys. 4.11).



Rysunek 4.3: Zrzut ekranu z menu aplikacji mobilnej gogli Oculus Quest 2, gdzie zaznaczona jest opcja umożliwiająca włączenie trybu deweloperskiego.



Rysunek 4.4: Zrzut ekranu z menu aplikacji gogli Oculus Quest 2 na komputerze stacjonarnym, gdzie zaznaczona jest opcja umożliwiająca instalację aplikacji z nieznanych źródeł.

```
1 #Przejscie do folderu ze sterownikiem  
2 $ cd <lokalizacja folderu platform-tools>  
3 #Wylistowanie podpietych urzadzen  
4 $ adb devices -l
```

Rysunek 4.5: Wylistowanie podpietych urządzeń.

```
1 * daemon not running; starting now at tcp:5037  
2 * daemon started successfully  
3 List of devices attached  
4 1WMHH822B90366      device product:hollywood model:  
    Quest_2 device:hollywood transport_id:1
```

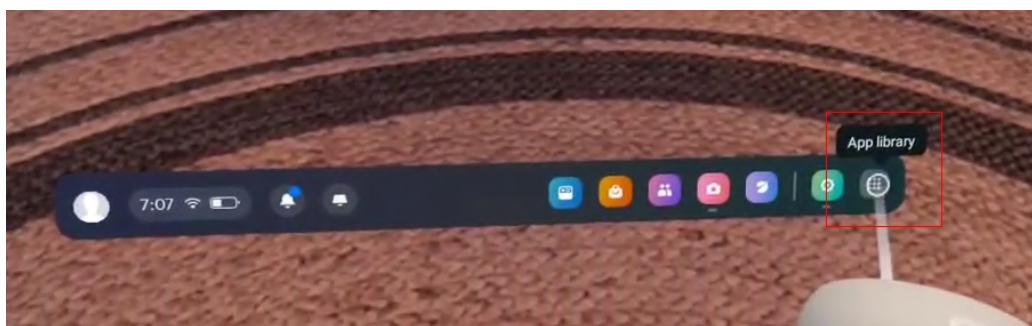
Rysunek 4.6: Oczekiwany przykładowy rezultat wylistowania podpietych urządzeń.

```
1 #Przejscie do folderu ze sterownikiem  
2 $ cd <lokalizacja folderu platform-tools>  
3 #Instalacja aplikacji na urzadzenie  
4 $ adb install -r <lokalizacja_pliku>
```

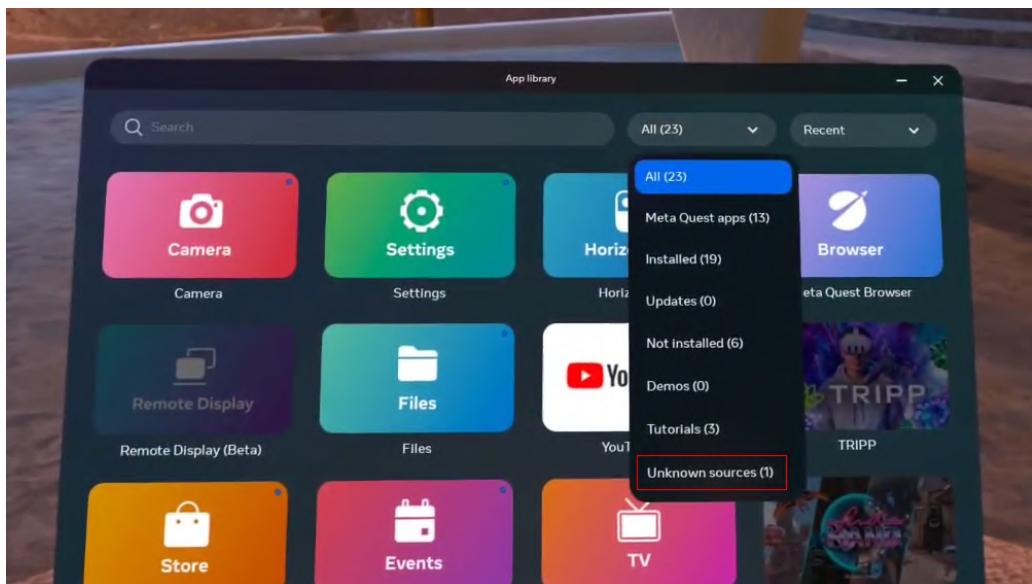
Rysunek 4.7: Instalacja aplikacji na urządzenie.

```
1 Performing Streamed Install  
2 Success
```

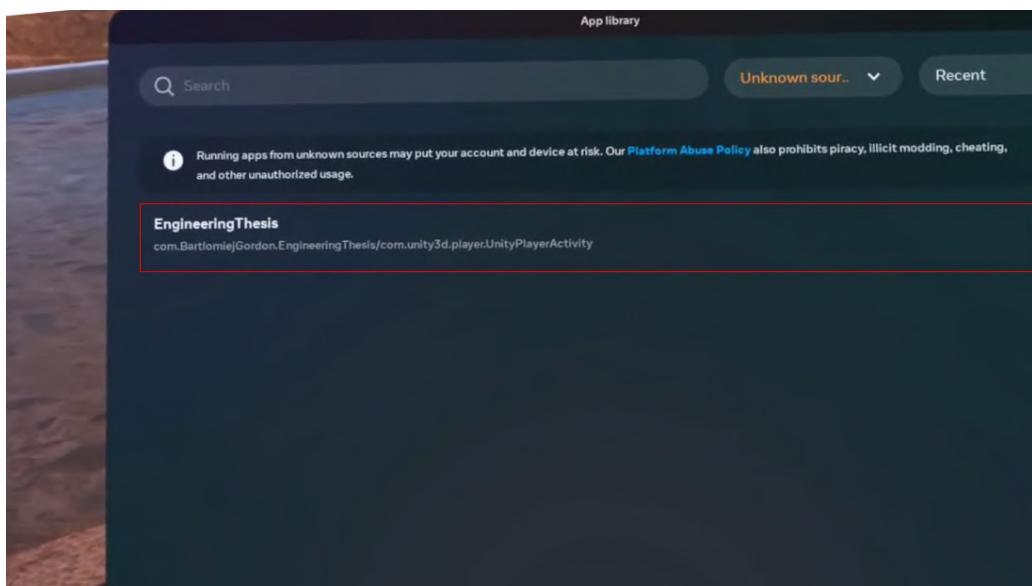
Rysunek 4.8: Oczekiwany przykładowy rezultat po instalacji aplikacji na urządzenie



Rysunek 4.9: Zrzut ekranu przedstawiający pasek narzędzi z zaznaczonym kafelkiem *App library*.



Rysunek 4.10: Zrzut ekranu przedstawiający bibliotekę aplikacji z rozwiniętym menu i zaznaczoną opcją *Unknown sources*.



Rysunek 4.11: Zrzut ekranu przedstawiający poprawnie zainstalowaną aplikację.

4.3 Przebieg gry

Celem rozgrywki jest jak najdłuższe przetrwanie gracza w obliczu bezustannych ataków owoców, warzyw, beczek, skrzyń i bomb wystrzeliwanych z armat umiejscowionych zarówno na wyspie, jak i na statku pirackim.

Akcja rozgrywa się na drewnianej tratwie unoszącej się na wodzie, gdzie gracz ma w zasięgu wzroku wszystkie wrogie armaty, a także tablicę z istotnymi informacjami dotyczącymi rozgrywki (Rys. 4.12).

Głównym celem gracza jest zdobycie jak największej ilości punktów, co można osiągnąć poprzez precyzyjne przecinanie nadciągających owoców i warzyw za pomocą szabli (*ang. sabre*), a także sprawną eliminację drewnianych obiektów, takich jak skrzynie i beczki, wykorzystując do tego celu pistolet skałkowy (*ang. flintlock pistol*).

Główną przeszkodą postawioną przed graczem są nieustannie wystrzeliwane w jego kierunku bomby z armat. Każde trafienie bomby wiąże się ze zmniejszeniem liczby punktów zdrowia gracza. Optymalną strategią jest zatem unikanie tych pocisków i pozwalanie im na spadnięcie do wody skutkujące zgaszeniem lontu i dezaktywacją bomby.



Rysunek 4.12: Zrzut ekranu widoku rozgrywki z szablą w prawej dłoni.

W grze dostępne są także specjalne pociski, takie jak klepsydra czy sztaba złota. Po ich zniszczeniu za pomocą dowolnej broni, wywoływane są indywidualne efekty, takie jak spowolnienie czasu lub nagłe zwiększenie liczby punktów. Okresowo, z konkretnych armat wystrzeliwana jest predefiniowana kombinacja składająca się z wielu pocisków, na przykład trzy bomby z rzędu. Zmusza to gracza do szybkiej reakcji oraz dostosowania strategii.

Gra oferuje trzy różne poziomy trudności, z każdym definiującym rozgrywkę poprzez

Tabela 4.1: Porównanie poziomów trudności

	Łatwy (easy)	Średni (medium)	Trudny (hard)
Tolerancja perfekcyjnego przecięcia	14.4%	10%	10%
Szansa na perfekcyjne zestrzelanie	33%	20%	15%
Liczba armat na wyspach	2	3	5
Maks. ilość punktów życia	3	2	1
Proporcja bomb do innych	5 : 95	9 : 91	12 : 88
Szansa na pocisk specjalny	4%	3.5%	2.5%
Szansa na sekwencję pocisków	6%	8%	8%
Przyrost mnożnika	0.3	0.2	0.1
Odjęcie punktów za brak trafienia	nie	nie	tak
Odjęcie mnożnika za brak trafienia	nie	tak	tak
Odstęp między wystrzałami	375ms-900ms	350ms-625ms	300ms-625ms

modyfikację różnych parametrów które zostały przedstawione w tabeli 4.1. Te zmiany obejmują odstępy między poszczególnymi pociskami, szansę na wystrzał specjalnej amunicji oraz inne wartości wpływające na tempo gry. Precyzyjne kalibracje tych parametrów zapewniają wyjątkowe doznania dla graczy o różnym stopniu zaawansowania.

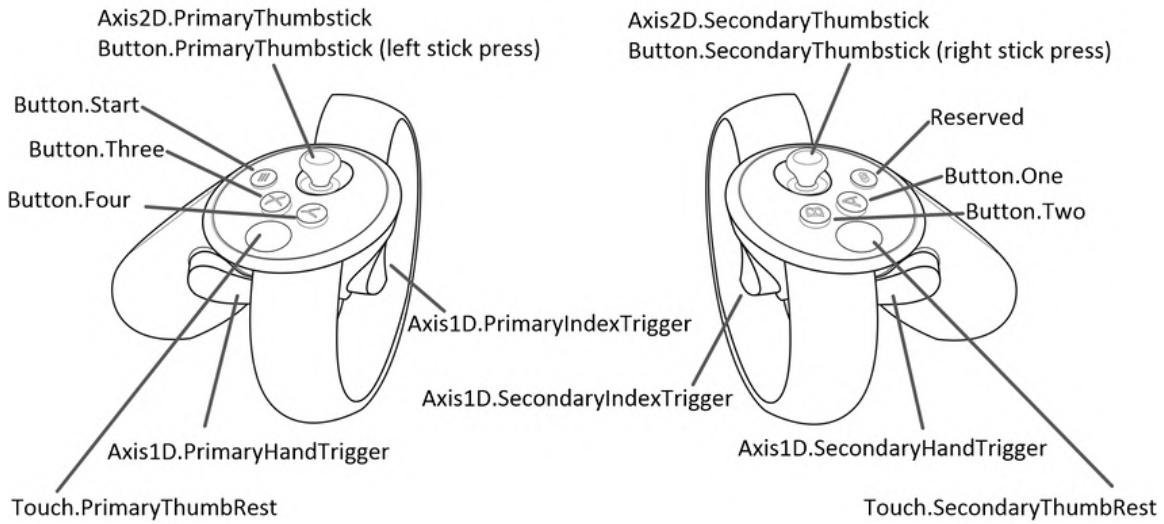
Różne poziomy trudności wpływają istotnie na zasady rozgrywki, takie jak liczba armat na wyspach, maksymalna ilość życia czy sposób naliczania oraz utraty mnożnika punktów – kluczowego elementu wpływającego na osiągane wyniki.

4.4 Nawigacja

Główym sposobem obsługi aplikacji jest korzystanie z dedykowanych kontrolerów Oculus Touch, stworzonych specjalnie dla gogli wirtualnej rzeczywistości, Oculus Quest 2. Użytkownik ma możliwość używania innych kontrolerów, jednakże nie są one wspierane przez aplikację.

Poniżej przedstawiono szczegółowe akcje przypisane do poszczególnych przycisków kontrolerów widocznych na rysunku 4.13:

- **Poruszanie przedmiotami** - polega na fizycznym ruchu kontrolerów w przestrzeni.
- **Wybór opcji w menu** - możliwy poprzez użycie `Axis1D.PrimaryIndexTrigger` lub `Axis1D.SecondaryIndexTrigger`.
- **Wystrzał z pistoletu** - akcja ta jest realizowana przez użycie `Axis1D.PrimaryHandTrigger` lub `Axis1D.SecondaryHandTrigger`, zależnie od dłoni, w której trzymany jest pistolet.
- **Obrót przy pomocy kontrolera** - aby dokonać obrotu należy pochylić w płaszczyźnie horyzontalnej gałkę `Axis2D.PrimaryThumbstick` lub `Axis2D.SecondaryThumbstick`.



Rysunek 4.13: Mapowanie przycisków kontrolera Oculus Touch, źródło:
<https://developer.oculus.com/>

4.5 Obsługa menu

Po uruchomieniu aplikacji, użytkownikowi prezentowane jest główne menu (Rys. 4.14), zawierające kilka opcji, takich jak:

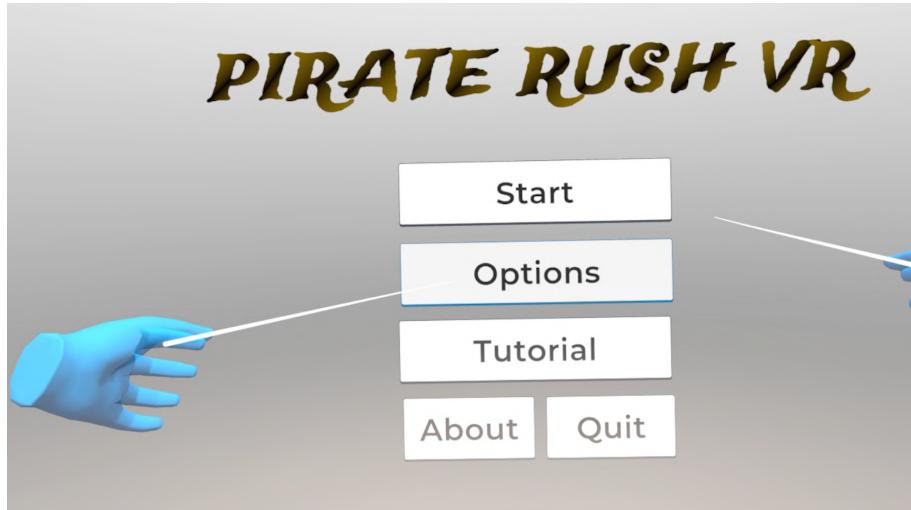
- Rozpoczęcie rozgrywki (*Start Game*).
- Ustawienia (*Settings*).
- Samouczek (*Tutorial*).
- Opis gry (*About*).
- Wyjście z gry (*Quit*).

Przycisk *Quit* umożliwia użytkownikowi bezpieczne opuszczenie rozgrywki, powracając do menu głównego przy jednoczesnym zapisie stanu gry.

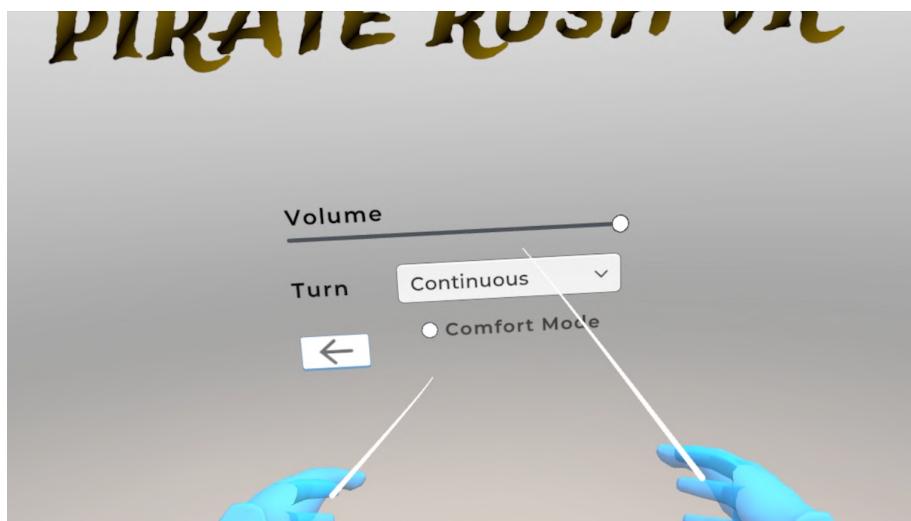
Sekcja *About* zawiera zwięzły tekst podsumowujący aplikację i jej cel w jednym zdaniu (Rys. 4.17).

W sekcji *Settings* (Rys. 4.15) użytkownik ma możliwość dostosowania różnych parametrów działania aplikacji:

- **Zmiana głośności efektów dźwiękowych i muzyki** - dostępny jest suwak umożliwiający regulację głośności.
- **Zmiana typu obrotu** - użytkownik ma do wyboru dwa tryby, 'ciągły' oraz 'skokowy'. Tryb 'ciągły' zapewnia naturalny obrót, jednak może wywoływać uczucie mdłości u niektórych użytkowników. Z tego powodu zaimplementowano tryb 'skokowy', pozwalający na obracanie się w interwałach 30°, zapewniając wygodniejsze korzystanie z aplikacji kosztem immersji.



Rysunek 4.14: Zrzut ekranu prezentujący interfejs menu głównego.



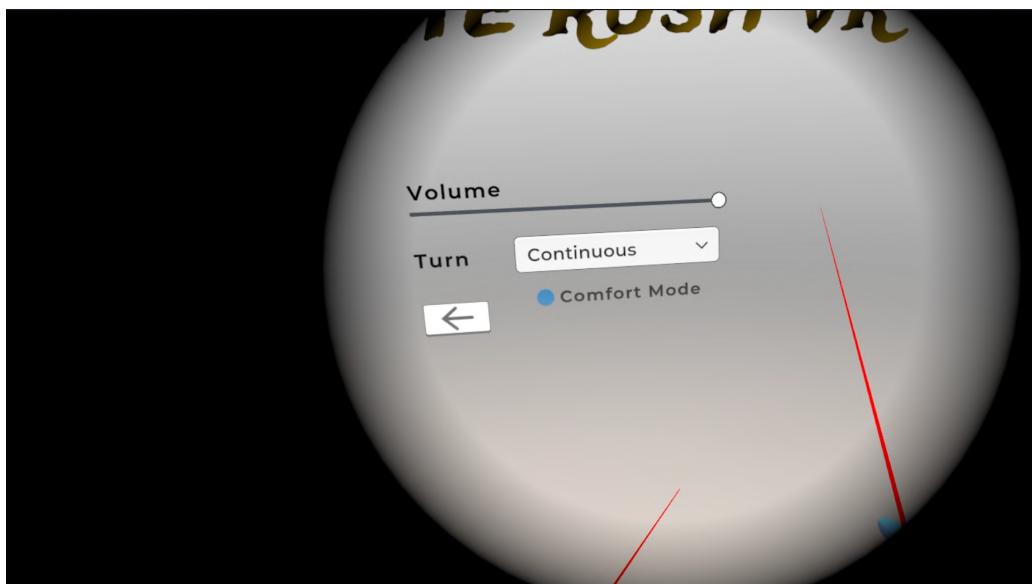
Rysunek 4.15: Zrzut ekranu prezentujący interfejs ustawień.

- **Włączenie trybu większego komfortu** - opcja ta ogranicza wizję peryferyjną gracza podczas obrotu kontrolerem, zapewniając większy komfort fizyczny podczas użytkowania aplikacji (Rys. 4.16).

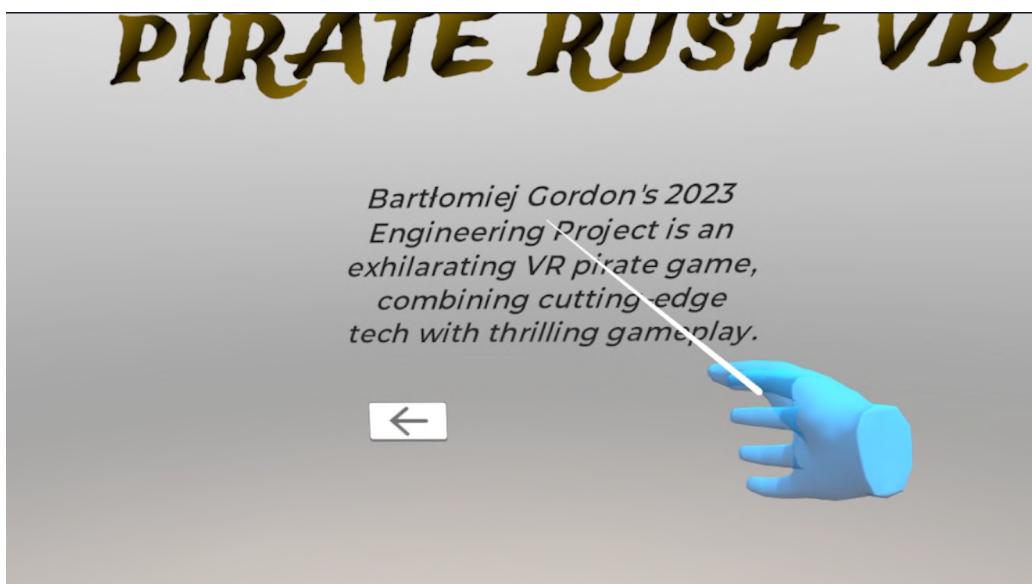
Po wybraniu preferencji w sekcji ustawień, zostają one przypisane do lokalnego profilu użytkownika. Te preferencje są automatycznie wczytywane przy każdym kolejnym uruchomieniu aplikacji.

Sekcja *Tutorial* stanowi przewodnik dla gracza, prezentujący piętnaście paneli opisujących kluczowe elementy gry (Rys. 4.18). Każdy panel zawiera tytuł, zdjęcie i opis, utrzymany w klimacie pirackim, aby podtrzymywać immersję nawet podczas nauki różnych mechanik i elementów gry.

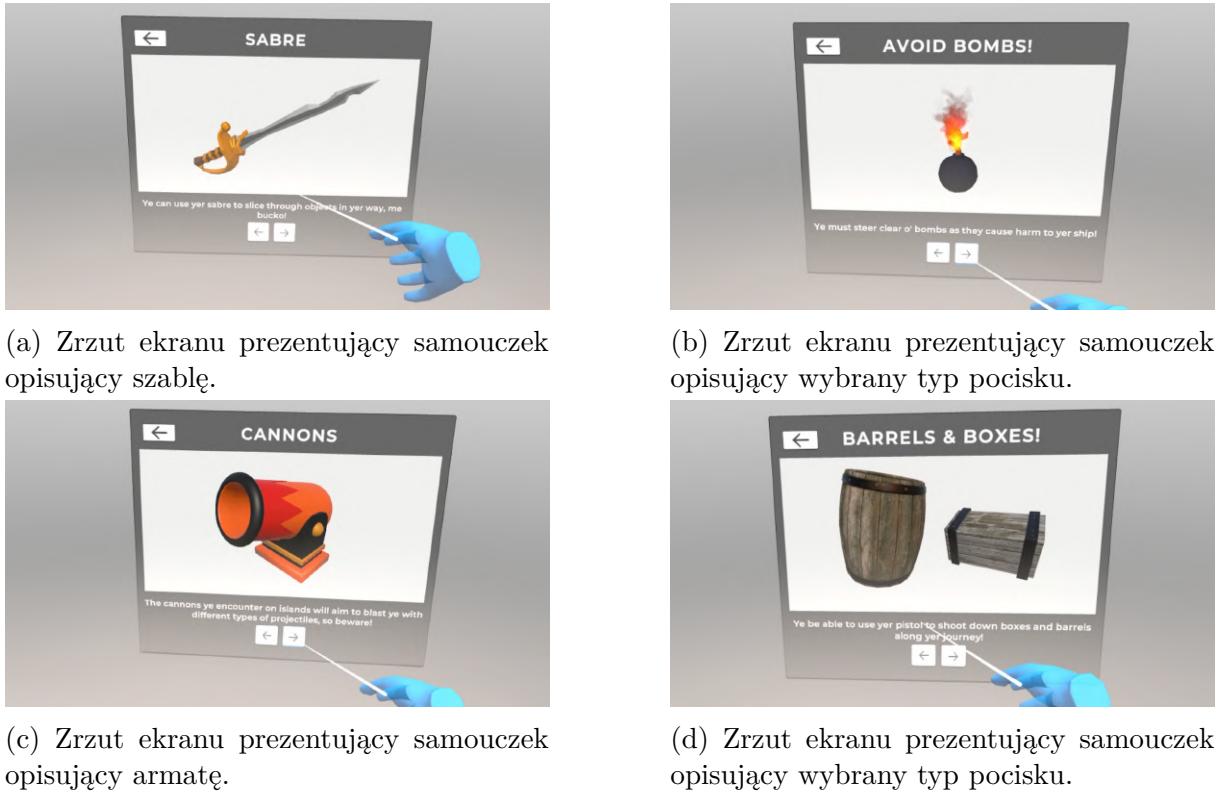
Rozpoczęcie rozgrywki wyświetla sekwencję ekranów, pozwalającą dostosować rozgrywkę do indywidualnych preferencji gracza. Wybór pseudonimu gracza to pierwszy panel który należy uzupełnić przed rozgrywką (Rys. 4.19). Gracz ma możliwość wyboru



Rysunek 4.16: Zrzut ekranu prezentujący widok trybu comfort mode w momencie obrotu manualnego.



Rysunek 4.17: Zrzut ekranu przedstawiający sekcję about



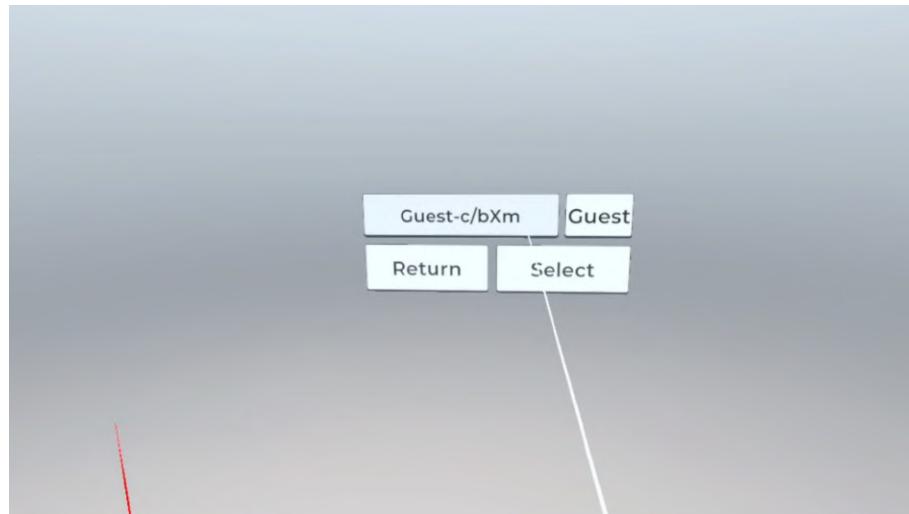
Rysunek 4.18: Przykładowe panele prezentujące się w samouczku.

pseudonimu poprzez wpisanie go w pole tekstowe. Umożliwia to klawiatura systemowa przygotowana przez firmę Meta (Rys. 4.20). Alternatywnie, istnieje opcja *Guest*, generująca unikatowy pseudonim dla gracza pozostającego anonimowym. Gracz może wybrać jeden z trzech dostępnych poziomów trudności. Wygląd panelu wyboru poziomu trudności przedstawiony został na rysunku 4.21. Kolejnym krokiem w procesie jest umożliwienie graczowi wyboru konfiguracji trzymania broni, które będą wykorzystane w trakcie rozgrywki (Rys. 4.22). Gracz ma do dyspozycji dwie różne opcje konfiguracyjne:

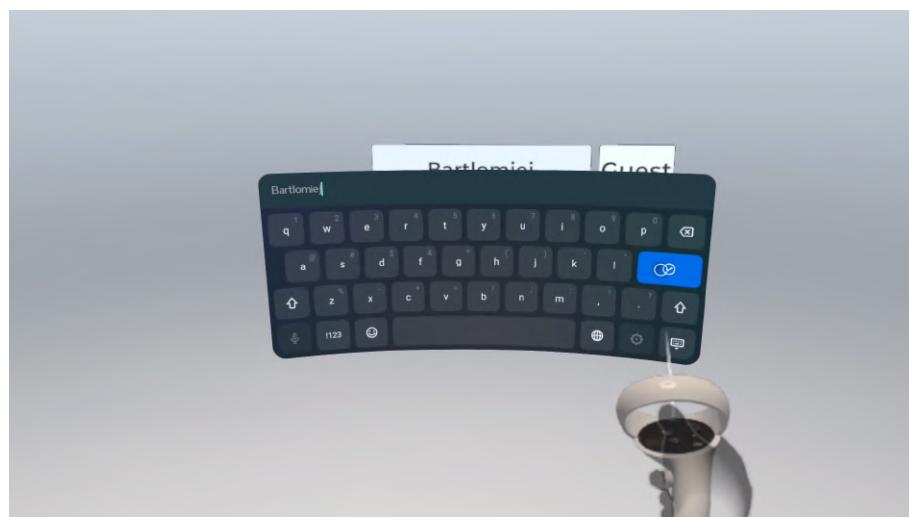
- **Pistolet skałkowy** - lewa dłoń, **szabla** - prawa dłoń
- **Szabla** - lewa dłoń, **pistolet skałkowy** - prawa dłoń

Po dokonaniu wyborów, gracz zostaje przeniesiony do menu startowego umieszczonego na scenie, na której odbywa się rozgrywka (Rys. 4.23). Menu to zapewnia płynne przejście do gry oraz dostęp do następujących opcji:

- Rozpoczęcie rozgrywki (*Start Game*)
- Tabela wyników (*Leaderboard*)
- Samouczek (*Tutorial*)
- Powrót do menu głównego (*Main menu*)



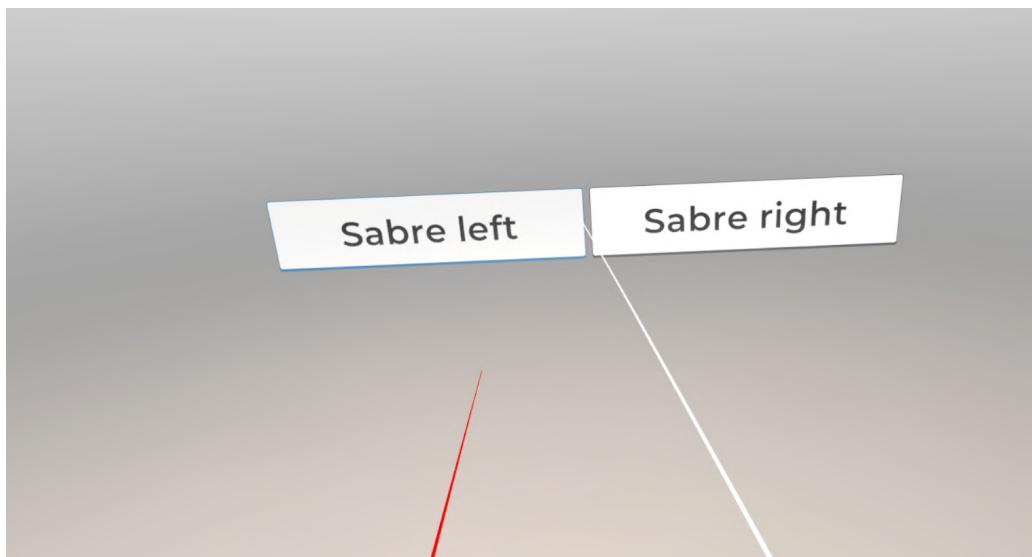
Rysunek 4.19: Zrzut ekranu przedstawiający panel wpisywania pseudonimu gracza.



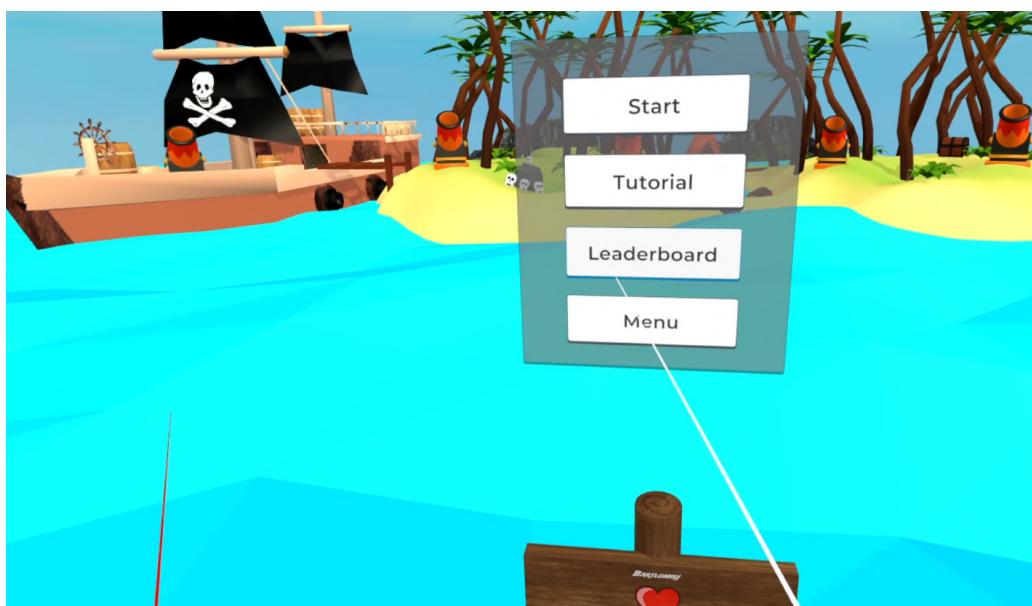
Rysunek 4.20: Zrzut ekranu przedstawiający klawiaturę systemową.



Rysunek 4.21: Zrzut ekranu przedstawiający panel wyboru poziomów trudności.



Rysunek 4.22: Zrzut ekranu przedstawiający panel wyboru konfiguracji broni.



Rysunek 4.23: Zrzut ekranu przedstawiający menu startowe przed rozgrywką.



Rysunek 4.24: Zrzut ekranu przedstawiający tabelę wyników.



Rysunek 4.25: Zrzut ekranu przedstawiający samouczek przed rozgrywką.

Przycisk *Main menu* w płynny i komfortowy sposób przenosi gracza z rozgrywki do menu głównego. Opcja ta jest przydatna jeżeli użytkownik postanowi zmienić wybrane przez siebie preferencje lub opuścić rozgrywkę.

Sekcja *Leaderboard* prezentuje punktację lokalnych graczy, którzy zmierzyli się z aktualnie wybranym poziomem trudności (Rys. 4.24). Podświetlony zostaje wynik gracza przeglądającego tabelę. Zestawienie można swobodnie przewijać jak standardową przewijalną listę.

Panel *Tutorial* zawiera te same informacje, które gracz może sprawdzić przed rozpoczęciem gry z tą różnicą, że teraz osadzony jest na scenie rozgrywki (Rys. 4.25). W ramach takiego rozwiązania udostępniana jest dodatkowa możliwość łatwego zdobycia bądź uzupełnienia brakującej wiedzy w każdym momencie.

Przycisk *Start Game* zamyka menu startowe, umożliwiając graczowi przejście do gry z wcześniejszej wybranej konfiguracją broni (Rys. 4.12).



Rysunek 4.26: Zrzut ekranu przedstawiający szabłę.

4.6 Elementy gry

Poniższy rozdział skupia się na kluczowych elementach stanowiących istotną część doświadczenia gracza. Przedstawione zostaną ich funkcje w interakcji z różnorodnymi obiektami w grze. Ponadto omówione zostają reakcje poszczególnych elementów na działania gracza. Elementy świata gry, takie jak woda, wyspy, czy pociski, także zostały opisane pod kątem ich wpływu na rozgrywkę.

4.6.1 Szabla

Szabla, zobrazowana na rysunku 4.26, jest głównym narzędziem gracza. Gracz korzysta z niej do rozcinania wszystkich obiektów wrażliwych na przecięcia.

Przecięcie obiektu przez środek z parametryzowaną przez aktualny poziom trudności tolerancją jest uznawane jako przecięcie perfekcyjne. Generuje to dodatkowe punkty dla gracza i wyświetla komunikat *Perfect* (Rys. 4.27). Gracz, mając do czynienia z niebezpiecznymi pociskami, takimi jak bomby, może wykorzystać trzon szabli do odbicia.

Przecięcie obiektu podatnego wyłącznie na zestrzelenie prowadzi do odjęcia połowy możliwych do otrzymania punktów od całkowitego wyniku gracza. Zachęca to graczy do wybrania odpowiedniego narzędzia. Dodatkowo, nieprawidłowe uderzenie generuje także iskry w miejscu kontaktu szabli z pociskiem, co służy jako wizualny sygnał informujący użytkownika o błędzie (Rys. 4.28).

4.6.2 Pistolet skałkowy

Pistolet skałkowy, zaprezentowany na rysunku 4.29, jest kolejnym kluczowym narzędziem używanym przez gracza. Służy do zestrzeliwania beczek i skrzyń, które nie są dostosowane do przecięć.

Częstotliwość strzałów z pistoletu skałkowego jest ograniczona, co wymusza na graczu



Rysunek 4.27: Zrzut ekranu przedstawiający komunikat o perfekcyjnym przecięciu.



Rysunek 4.28: Zrzut ekranu przedstawiający iskry po próbie przecięcia obiektu nieprzy-
stosowanego do przecinania.



Rysunek 4.29: Zrzut ekranu przedstawiający pistolet skałkowy.



Rysunek 4.30: Zrzut ekranu przedstawiający błysk wystrzału z pistoletu skałkowego.

strategiczne wykorzystanie broni. To ograniczenie sprawia, że konieczna jest umiejętność dobrego zarządzania kulami jak i dokonywania celnych strzałów.

Podobnie jak w przypadku szabli, istnieje reguła zależna od parametrów przypisanych do danego poziomu trudności definiująca szansę na zestrzelenie perfekcyjne. Generuje to dodatkowe punkty dla gracza i wyświetla komunikat *Perfect*.

Zestrzelenie obiektu podatnego wyłącznie na przeciecie prowadzi do odjęcia połowy możliwych do otrzymania punktów od całkowitego wyniku gracza. Zachęca to graczy do wybrania odpowiedniego narzędzia. Podczas wystrzału z lufy pistoletu skałkowego generowany jest efekt błysku, przedstawiony na rysunku 4.30.



Rysunek 4.31: Wszystkie pociski wrażliwe na przecięcia.



Rysunek 4.32: Zrzut ekranu przedstawiający efekt miąższa i rozpadu kokosa na dwie części.

4.6.3 Pociski wrażliwe na przecięcia

Pociski wrażliwe na przecięcie (Rys. 4.31) są kluczowym elementem rozgrywki, stanowiąc najczęstszy rodzaj pocisku wystrzeliwanego z armat na wyspie. Pociski tego typu są reprezentowane przez różnorodne owoce i warzywa, podatne na obrażenia zadawane przez szabłę.

Każdy z pocisków po otrzymaniu obrażeń generuje efekt wizualny w miejscu uderzenia, przedstawiający miąższ danego owocu. Kolor efektu jest zależny od predefiniowanego koloru dominującego wnętrza obiektu. Po przecięciu pocisku tworzone są dwie części w których środkach znajdują się tekstury adekwatne do wnętrz obiektów (Rys. 4.32 i Rys. 4.33).

Jeśli pocisk nie zostanie przecięty w przeciągu 5 sekund od wystrzelenia lub wpadnie do wody, znika on ze sceny z animacją.

Wartości punktowe dla różnych typów pocisków są uzależnione od stopnia trudności



Rysunek 4.33: Zrzut ekranu przedstawiający efekt miąższu i rozpadu granatu na dwie części.

trafienia ich za pomocą szabli zostały przedstawione w tabeli 4.2.

Tabela 4.2: Punkty za przecięcie różnych rodzajów pocisków

Nazwa pocisku	Punkty za przecięcie
Kiwi	200
Mango	200
Granat	150
Smoczy owoc	175
Cytryna	150
Melon	150
Pomarańcza	150
Banan	150
Kokos	150
Arbuз	125

4.6.4 Pociski wrażliwe na zestrzelenie

Pociski wrażliwe na zestrzelenie stanowią kolejny typ elementów wpływających na rozgrywkę. Różnią się one od pozostałych pocisków tym, że reagują na kule pistoletu skałkowego poprzez rozbicie się na mniejsze fragmenty (Rys. 4.35).

Wartości punktowe dla konkretnych pocisków tego typu zostały zdefiniowane i przedstawione w tabeli 4.3.

4.6.5 Bomba

Bomba (Rys. 4.36), jako specjalny rodzaj pocisku, odgrywa kluczową rolę w mechanice gry, stanowiąc zagrożenie dla gracza i wpływając na dynamikę rozgrywki. Bomba reaguje



Rysunek 4.34: Wszystkie pociski wrażliwe na zestrzelenie.

Tabela 4.3: Punkty za zestrzelenie różnych rodzajów pocisków.

Nazwa pocisku	Punkty za zestrzelenie
Beczka	250
Skrzynia	250

na wszystkie typy zadawanych obrażeń - zarówno na przecięcia szablą, jak i zestrzelenie przy pomocy pistoletu skałkowego.

Każde dotknięcie bomby powoduje utratę jednego punktu zdrowia gracza i dwustu punktów wraz z ewentualnym mnożnikiem punktów, jeśli poziom trudności przewiduje taką karę.

Po wybuchu, bomba emisuje silne odpychające fale we wszystkich kierunkach, wpływając na trajektorie innych nadlatujących pocisków. Równocześnie generuje huk i błysk, co stanowi efekt wizualny w grze, zwiększając jej dynamiczność (Rys. 4.37).

Prawidłowe zachowanie gracza wobec tego typu pocisku to unikanie bezpośredniego kontaktu z nim. Pozwolenie na wpadnięcie bomby do wody jest strategią obronną - podpalony lont gaśnie, a sama bomba traci swoją siłę rażenia, stając się niegroźna.

4.6.6 Klepsydra

Klepsydra (Rys. 4.38) stanowi unikatowy rodzaj pocisku specjalnego, który wprowadza wyjątkowe zmiany w mechanice rozgrywki. Działa ona jako manipulator czasu, wpływając na dynamikę gry. Klepsydra reaguje na wszystkie typy zadawanych obrażeń - zarówno na przecięcie szablą, jak i zestrzelenie za pomocą pistoletu skałkowego.

Nakłada ona na rozgrywkę chwilowe spowolnienie czasu. Efekt ten trwa sześć sekund i nie kumuluje się - jest to jednorazowy efekt. W trakcie tych sześciu sekund czas zwalnia o 60%, co daje graczowi możliwość lepszego dostosowania się do dynamiki rozgrywki.



(a) Pierwszy etap rozpadu pocisku.



(b) Drugi etap rozpadu pocisku.

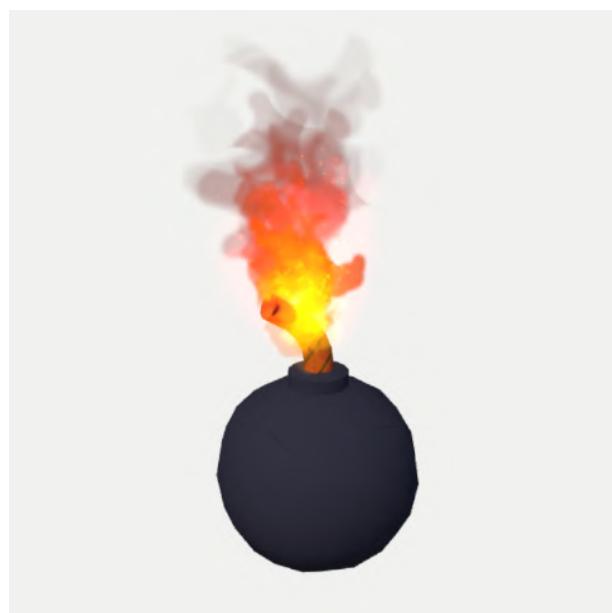


(c) Trzeci etap rozpadu pocisku.



(d) Czwarty etap rozpadu pocisku.

Rysunek 4.35: Zrzuty ekranu przedstawiające efekt rozbicia beczki.



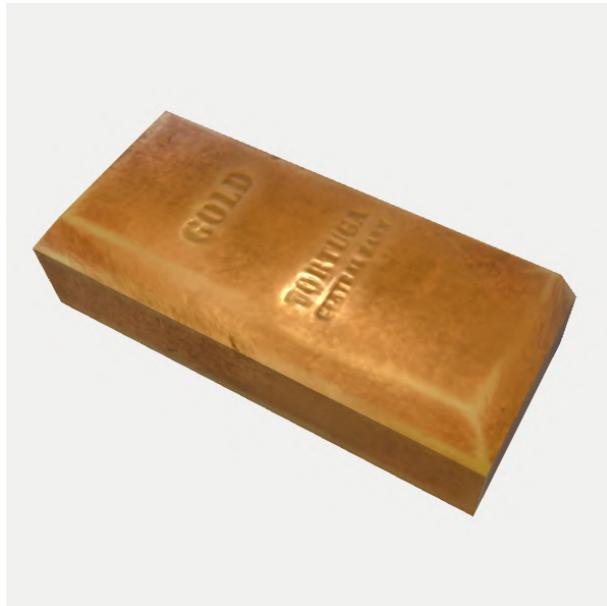
Rysunek 4.36: Zrzut ekranu przedstawiający bombę.



Rysunek 4.37: Zrzut ekranu przedstawiający efekt wybuchu bomby.



Rysunek 4.38: Zrzut ekranu przedstawiający klepsydrę.



Rysunek 4.39: Zrzut ekranu przedstawiający sztabę złota.

4.6.7 Sztaba złota

Sztaba złota (Rys. 4.39) jest specjalnym typem pocisku, reagującym na wszystkie typy zadawanych obrażeń - zarówno odcięcia szablą, jak i trafienia z pistoletu skałkowego.

Unikatowym efektem nakładanym poprzez jej zebranie jest dodanie do wyniku gracza tysiąca punktów.

4.6.8 Serce

Serce (Rys. 4.40) reaguje na wszystkie typy zadawanych obrażeń - zarówno na przecięcia szablą, jak i trafienia z pistoletu skałkowego. Jest to specjalny pocisk leczący, który przywraca graczowi jeden punkt życia, dodając jednocześnie czterysta punktów do jego wyniku. Jeśli gracz ma już pełne zdrowie, dodawane są tylko punkty do wyniku.

4.6.9 Elementy świata

Poniższy rozdział skupia się na kluczowych elementach stanowiących istotną część doświadczenia gracza. Przedstawione zostaną ich interakcje z różnorodnymi obiektami w grze.

Armata

Armaty (Rys. 4.41) są obiektami regularnie wystrzeliwującymi pociski w stronę gracza. Ich ilość jest definiowana poprzez parametr zależny od aktualnego poziomu trudności. Armaty ciągle dostosowują swój kąt obrotu, aby skierować się w stronę gracza i trajektorii wystrzelonego pocisku.



Rysunek 4.40: Zrzut ekranu przedstawiający serce.



Rysunek 4.41: Zrzut ekranu przedstawiający armatę.



Rysunek 4.42: Zrzut ekranu przedstawiający tratwę.

Tratwa

Tratwa (Rys. 4.42) to platforma, na której stoi gracz. Nie ma możliwości opuszczenia tratwy w trakcie rozgrywki. Przed graczem na platformie znajduje się tablica informacyjna.

Tablica informacyjna

Tablica informacyjna (Rys. 4.43) jest umieszczona na tratwie. Znajdują się na niej informacje dotyczące aktualnej rozgrywki takie jak pseudonim gracza, ilości punktów życia a także aktualny wynik i mnożnik.

Rekin

Rekin (Rys. 4.44) jest nieagresywnym agentem sztucznej inteligencji, pływającym po wodzie. Jego głównym zadaniem jest dodanie głębi rozgrywce.

Woda

Woda stanowi element świata, w którym naturalnie znikają pociski, a bomby po zetknięciu z wodą zostają zdezaktywowane.



Rysunek 4.43: Zrzut ekranu przedstawiający tablicę informacyjną.



Rysunek 4.44: Zrzut ekranu przedstawiający rekina.



Rysunek 4.45: Zrzut ekranu przedstawiający wyspę i statek piracki i wszystkie możliwe armaty.

Wyspa i Statek

Wyspa i statek (Rys. 4.45) są elementami świata, na których w określonych miejscach mogą pojawić się armaty.

Rozdział 5

Specyfikacja wewnętrzna

W niniejszym rozdziale przedstawiono kluczowe biblioteki oraz paczki wykorzystane w ramach projektu. Każdy z elementów stanowi istotny składnik aplikacji, zapewniając funkcjonalności umożliwiające rozwój oraz optymalizację. W dalszej części zostały opisane najważniejsze klasy w programie. Pozostałe klasy odpowiedzialne, między innymi za animacje, efekty czy obsługę interfejsu użytkownika, zostały opisane w załączonej do pracy dokumentacji wygenerowanej za pomocą programu Doxygen [10].

5.1 Opis bibliotek i paczek

5.1.1 Oculus XR Plugin

Oculus XR Plugin [2] stanowi zbiór narzędzi umożliwiających integrację aplikacji z urządzeniami rzeczywistości wirtualnej np. Oculus Quest 2. Dzięki temu pakietowi możliwe jest wykorzystanie funkcji charakterystycznych dla środowiska VR, takich jak śledzenie ruchu, renderowanie stereoskopowe czy interakcja z kontrolerami oraz wiele innych. Paczka ta stanowi fundament aplikacji dzięki swojej prostocie użycia.

5.1.2 EzySlice

EzySlice [5] jest biblioteką ułatwiającą operacje przecinania obiektów w trójwymiarowej przestrzeni. Pakiet ten pozwala na dynamiczne dzielenie obiektów na podstawie zdefiniowanych płaszczyzn, co jest przydatne m.in. w procesie tworzenia efektów fizycznych czy mechanik rozgrywki.

5.1.3 DoTween

DoTween [12] to narzędzie umożliwiające płynną interpolację pomiędzy różnymi wartościami w czasie animacji. Biblioteka ta dostarcza łatwych w użyciu funkcji, które po-

zwalają na tworzenie efektownych animacji dla elementów interfejsu użytkownika czy obiektów w grze.

5.1.4 Addressables

Addressables [3] to paczka umożliwiająca zarządzanie zasobami w sposób bardziej efektywny niż standardowe załadowanie zasobów w Unity. Pozwala na dynamiczne ładowanie zasobów w trakcie działania aplikacji oraz zarządzanie nimi, co przyczynia się do optymalizacji zużycia pamięci. Dostępne są również funkcje manualnego odładowywania zasobów.

5.1.5 Input System

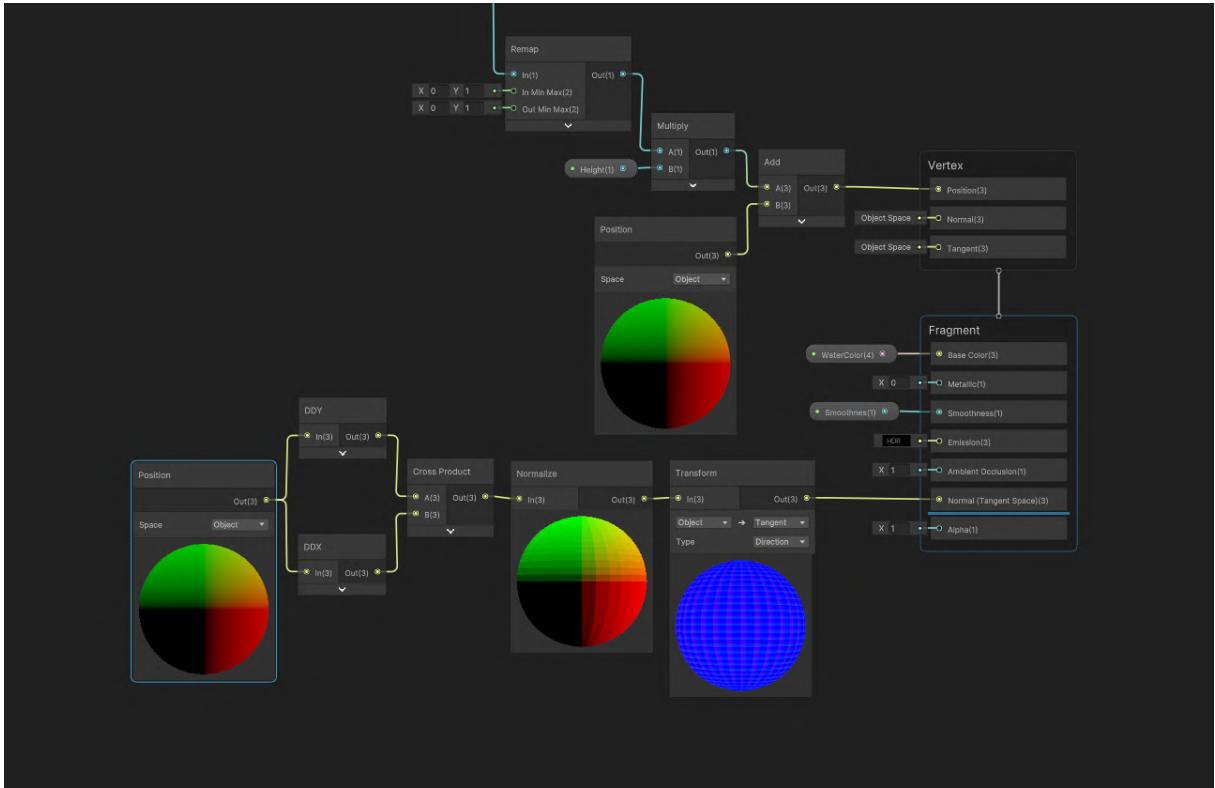
Input System [17] stanowi nowoczesne podejście do obsługi wejścia użytkownika w aplikacjach Unity. Ten pakiet zapewnia bardziej elastyczne i rozbudowane możliwości zarządzania różnymi rodzajami urządzeń wejściowych, ułatwiając tym samym projektowanie interfejsu oraz obsługę sterowania w grach. Wykorzystanie tego pakietu zostało podkutowane koniecznością mapowania przycisków znajdujących się na kontrolerach Oculus Touch na programowalne wejścia.

5.1.6 Text Mesh Pro

Text Mesh Pro [26] to zaawansowany system renderowania tekstu, który oferuje szereg funkcji znacznie przewyższających możliwości standardowego komponentu tekstowego w Unity. Pozwala na lepszą jakość renderowania tekstu, obsługę stylów tekstowych oraz efektów, co jest niezwykle istotne w przypadku interfejsów użytkownika oraz prezentacji informacji w grze. Paczka pierwotnie była niezależnym rozszerzeniem, które później zostało wykupione przez Unity Technologies i stało się integralną częścią środowiska Unity od wersji 2018.1.

5.1.7 Shader Graph

Shader Graph [1] umożliwia tworzenie graficznych jednostek cieniących bez konieczności pisania kodu. Ten pakiet dostarcza intuicyjnego interfejsu graficznego, który umożliwia projektowanie i manipulację jednostkami cieniącymi, co pozwala na osiągnięcie różnorodnych efektów wizualnych w aplikacji. Na rysunku 5.1 został przedstawiony fragment implementacji jednostki cieniącej odpowiedzialnej za wygląd i naturalny ruch wody w aplikacji.



Rysunek 5.1: Fragment kanwy Shader Graph wykorzystanej do powstania jednostki cieśniącej wody.

5.1.8 Visual Effect Graph

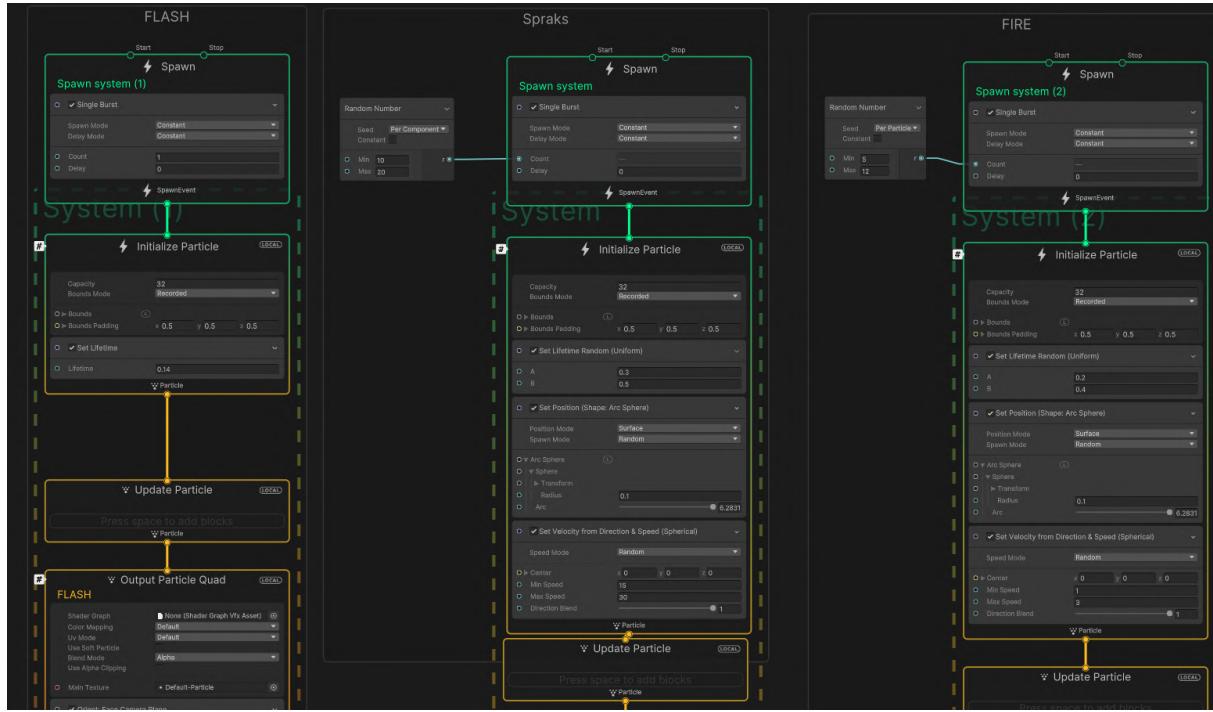
Visual Effect Graph [31] umożliwia tworzenie zaawansowanych efektów wizualnych, takich jak, efekty cząsteczkowe czy symulacje płynów. Dzięki temu pakietowi możliwe jest projektowanie i renderowanie efektów specjalnych, które dodają dynamizmu i atrakcyjności wizualnej do aplikacji. Na rysunku 5.2 został przedstawiony fragment implementacji efektu wizualnego wybucha bomby.

5.1.9 Velocity Estimator

Velocity Estimator [23] to narzędzie służące do oszacowania prędkości obiektów na podstawie danych z czujników. Umożliwia dokładne wyliczenie prędkości obiektów w trakcie ich ruchu, co może być wykorzystane m.in. do precyzyjnej symulacji fizyki w grach. Paczka ta została wykorzystana do ciągłej kalkulacji kierunku szabli.

5.1.10 JsonUtilityEx

JsonUtilityEx [11] to rozszerzenie wbudowanej funkcjonalności Unity, umożliwiające bardziej zaawansowane operacje na danych w formacie JSON [32]. Pozwala na bardziej elastyczne operowanie danymi poprzez prostą serializację typów programistycznych takich jak listy czy tablice.



Rysunek 5.2: Fragment kanwy Visual Effect Graph wykorzystanej do powstania efektu wybuchu bomby.

5.2 Opis wybranych klas

5.2.1 Główni zarządcy

W dziedzinie tworzenia gier, kluczową rolę odgrywa zarządzanie danymi, interakcjami i funkcjonalnościami. Klasy zarządcze, zaimplementowane przy użyciu wzorca projektowego singleton, stanowią istotny składnik kompleksowej struktury projektowej gier, ułatwiając skuteczne zarządzanie różnymi jej aspektami. Poprzez umożliwienie utworzenia jednej globalnej instancji danej klasy, singletony stają się centralnym punktem kontroli i zarządzania funkcjonalnościami gry. Niemniej jednak, należy zachować ostrożność w ich stosowaniu, ponieważ nadmierne użycie może prowadzić do problemów związanych ze stanem globalnym, utrudniać proces testowania, tworzyć silne zależności między klasami oraz generować trudności w obszarze zarządzania wielowątkowym środowiskiem.

GameManager

Klasa oparta na wzorcu projektowym singleton zarządzająca przebiegiem gry oraz przejściami interfejsu użytkownika.

Metody publiczne : Klasa GameManager

- `void StartGame()` - Inicjuje rozpoczęcie gry.

Metody prywatne : Klasa GameManager

- `void EndGame()` - Zarządza akcjami na zakończenie gry.
- `void SetHandItemsAccordingToPreference()` - Ustawia przedmioty w rękach zgodnie z preferencjami gracza.

Atrybuty prywatne : Klasa GameManager

- `GameObject startGamePanel` - Panel rozpoczęcia gry.
- `GameObject endGamePanel` - Panel zakończenia gry.
- `SetPlayerPreferences playerPrefs` - Ustawienia preferencji gracza.

AudioManager

Klasa oparta na wzorcu projektowym singleton odpowiedzialna za globalną obsługę dźwięku w grze.

Metody publiczne : Klasa AudioManager

- `void PlayOnTarget(GameObject target, Sound sound)` - Odtwarza dźwięk na określonym obiekcie gry.
- `void PlayAtPosition(Vector3 position, Sound sound)` - Odtwarza dźwięk w określonej pozycji na scenie.
- `void Play(AudioSource source, Sound sound, SoundType type)` - Odtwarza dźwięk przy użyciu dostarczonego źródła dźwięku umieszczonego na konkretnym obiekcie gry i parametrów dźwięku.
- `void PlayGlobal(Sound sound, SoundType type = SoundType.SFX)` - Odtwarza dźwięk globalnie.
- `void SetVolume(float value)` - Ustawia poziom głośności miksera audio.

Metody prywatne : Klasa AudioManager

- `IEnumerator FadeInMusic(Sound sound, float duration)` - Płynne przejście muzyki przez określony czas.
- `void SetMixer(AudioSource source, SoundType type)` - Ustawia mikser audio dla danego AudioSource na podstawie typu dźwięku.

Atrybuty prywatne : Klasa AudioManager

- `AudioMixerGroup masterMixer` - Mikser główny.
- `AudioMixerGroup sfxMixer` - Mikser efektów dźwiękowych.
- `AudioMixerGroup musicMixer` - Mikser muzyki.

ScoreManager

Klasa oparta na wzorcu projektowym singleton, która zarządza systemem punktacji w grze oraz tablicą wyników.

Metody publiczne : Klasa ScoreManager

- `void AddPoints(float points)` - Dodaje punkty do wyniku gracza, aktualizuje tablicę wyników i wyświetla aktualny wynik.
- `float CalculatePoints(float basePoints, bool isNegative, bool isCritical)`
- Oblicza wartości punktów uwzględniając różne warunki. Parametr basePoints odnosi się do punktacji bazowej, natomiast parametry isNegative oraz isCritical pozwalają uwzględnić wartość ujemną i krytyczną. Zwracana jest obliczona wartość punktów.
- `void ResetMultiplier()` - Resetuje mnożnik punktów do zera i aktualizuje wyświetlany wynik.
- `void DecrementMultiplier()` - Zmniejsza mnożnik punktów i aktualizuje wyświetlany wynik.
- `void IncrementMultiplier()` - Zwiększa mnożnik punktów.

Właściwości : Klasa ScoreManager

- `Leaderboard Leaderboard [get]` - Tablica wyników.
- `ScoreText ScoreText [set, get]` - Tekst z wynikiem.

Metody prywatne : Klasa ScoreManager

- `void DisplayNickname()` - Wyświetla pseudonim gracza na tablicy informacyjnej.
- `void CreateMockLeaderboardData()` - Tworzy testowe dane dla tablicy wyników do testów.
- `void DisplayScore()` - Wyświetla aktualny wynik na tablicy informacyjnej i zarządza powiadomieniami o najlepszym wyniku.

Atrybuty prywatne : Klasa ScoreManager

- `TMP_Text nicknameText` - Tekst z pseudonimem.
- `TMP_Text highscoreText` - Tekst z najlepszym wynikiem.
- `TMP_Text scoreDisplay` - Tekst z wyświetlonym wynikiem.
- `float multiplier` - Mnożnik punktów.

- `int multiplierCount` - Aktualna wartość licznika mnożników.
- `HighscoreEntry entry` - Dane do wpisu na listę najlepszych wyników.
- `float highscore` - Najlepszy wynik.

GlobalSettingManager

Klasa oparta na wzorcu projektowym singleton zarządzająca globalnymi ustawieniami i konfiguracjami gry.

Metody publiczne : Klasa GlobalSettingManager

- `void SetNickname(string newNickname)` - Ustawia pseudonim gracza.
- `void SetTurnType(TurnType turnType)` - Ustawia rodzaj manualnego obrotu gracza.
- `void SetVignetteUsage(bool isUsing)` - Ustawia używanie efektu winiety.
- `void SetVolume(float volume)` - Ustawia poziom głośności.
- `string GetNickname()` - Pobiera przechowywany pseudonim gracza.
- `TurnType GetTurnType()` - Pobiera przechowywany rodzaj tury.
- `bool GetVignetteUsage()` - Pobiera status używania efektu winiety.
- `float GetVolume()` - Pobiera przechowywany poziom głośności.

CannonsManager

Klasa oparta na wzorcu projektowym singleton zarządzająca armatami w grze.

Metody publiczne : Klasa CannonsManager

- `void Pause()` - Wstrzymuje działanie armat.
- `void UnPause()` - Wznawia działanie armat.
- `void ClearAllProjectiles()` - Usuwa wszystkie pociski ze sceny.

Właściwości : Klasa CannonsManager

- `List<ComboDatabase> ComboDatabases [get]` - Zwraca zestawienie wszystkich kombinacji pocisków.

Metody prywatne : Klasa CannonsManager

- `void SpawnCannons(int count)` - Tworzy armaty na scenie na podstawie podanej liczby.

Atrybuty prywatne : Klasa CannonsManager

- `AssetLabelReference combosLabel` - Referencja do etykiety zestawów kombinacji.
- `List<ComboController> cannonsOnScene` - Lista armat obecnych na scenie.
- `bool isPaused` - Flaga określająca, czy działanie armat jest wstrzymane.
- `TickEngine tickEngine` - Silnik zliczający takty systemu.

HealthManager

Klasa oparta na wzorcu projektowym singleton zarządzająca systemem zdrowia dla postaci w grze.

Metody publiczne : Klasa HealthManager

- `void TakeHit()` - Zadaje obrażenia graczowi.
- `void Heal(bool toMaximum)` - Leczy gracza, opcjonalnie do pełnego zdrowia.

Właściwości : Klasa HealthManager

- `event Action OnDeath [get]` - Wydarzenie wywoływanie po śmierci postaci.
- `event Action OnHealChange [get]` - Wydarzenie wywoływanie przy zmianie stanu zdrowia.

Metody prywatne : Klasa HealthManager

- `void DisplayHealth()` - Wyświetla aktualny stan zdrowia gracza.

Atrybuty prywatne : Klasa HealthManager

- `Transform healthContainer` - Kontener zdrowia.
- `HealthIcon healthIcon` - Ikona zdrowia.
- `int currentHealth` - Aktualna ilość zdrowia.
- `int maxHealth` - Maksymalna ilość zdrowia.
- `List<HealthIcon> healthIcons` - Lista ikon zdrowia.

SlowMotionManager

Klasa oparta na wzorcu projektowym singleton zarządzająca efektami zwolnionego tempa w grze.

Metody publiczne : Klasa SlowMotionManager

- `void Freeze(float duration)` - Spowalnia czas przez określony okres.

Metody prywatne : Klasa SlowMotionManager

- `IEnumerator ChangeTimeScaleOverDuration(float duration)` - Stopniowo zmienia skalę czasu przez określony czas, symulując zamrażanie i odmrażanie czasu.

Atrybuty prywatne : Klasa SlowMotionManager

- `float freezingSpeed` - Szybkość spowolnienia czasu.
- `float unfreezingSpeed` - Szybkość powrotu do normalnego biegu czasu.
- `float targetFreezeValue` - Docelowa wartość spowolnienia czasu.
- `Coroutine freezeCoroutine` - Korutyna odpowiedzialna za płynne spowolnienie czasu.

5.2.2 Armaty i Pociski

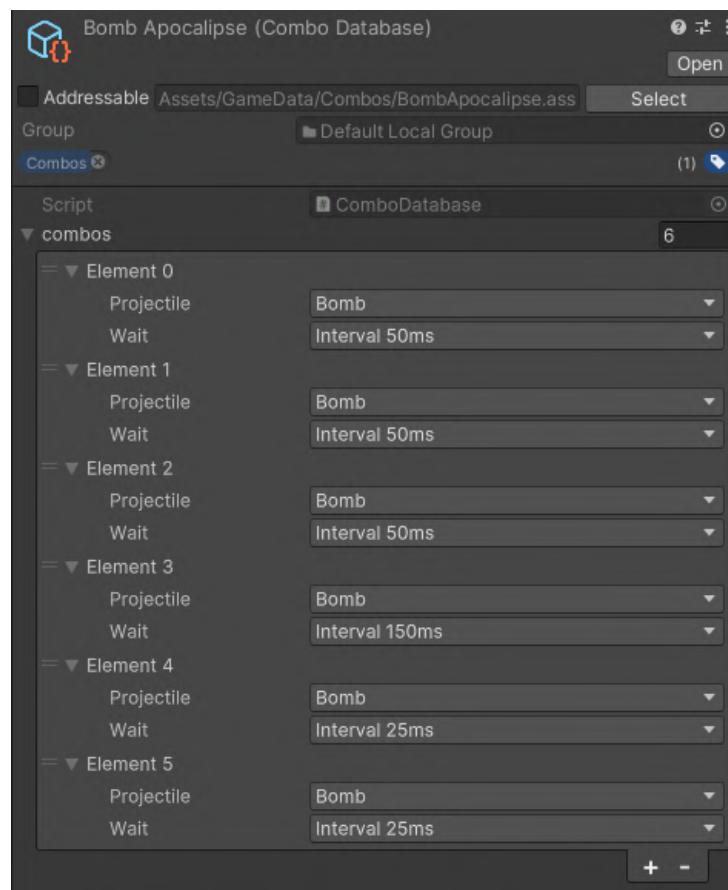
Mechanika związana z klasami armat i pocisków stanowi kluczowy, powiązany ze sobą, element rozgrywki. Hierarchia klas w tym obszarze programu została zobrazowana na rysunku 5.5. Główną zasadą jest autonomia działania każdej armaty, która samodzielnie decyduje, jaki pocisk wystrzelić w danym momencie. Nad wszystkimi armatami czuwa **CannonsManager**, który w momencie synchronizacji działania armat może włączyć globalną kombinację pocisków (Rys. 5.3), wybraną spośród zestawienia wszystkich kombinacji znajdującego się w grupie zasobów Addressables (Rys. 5.4).

CannonShooting

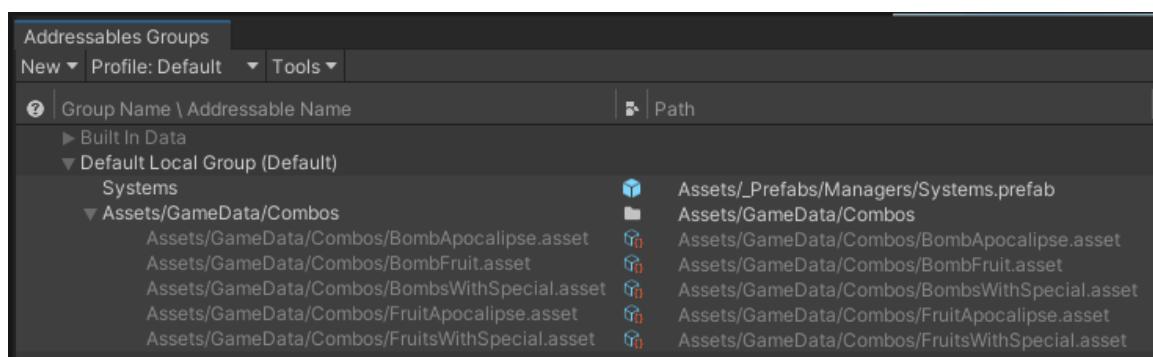
Klasa odpowiada za obsługę mechanizmu strzelania. Dziedziczy ona po klasie Unity, **MonoBehaviour**, co umożliwia jej bezpośrednie umieszczenie na obiekcie gry - cannon.

Metody publiczne: Klasa CannonShooting

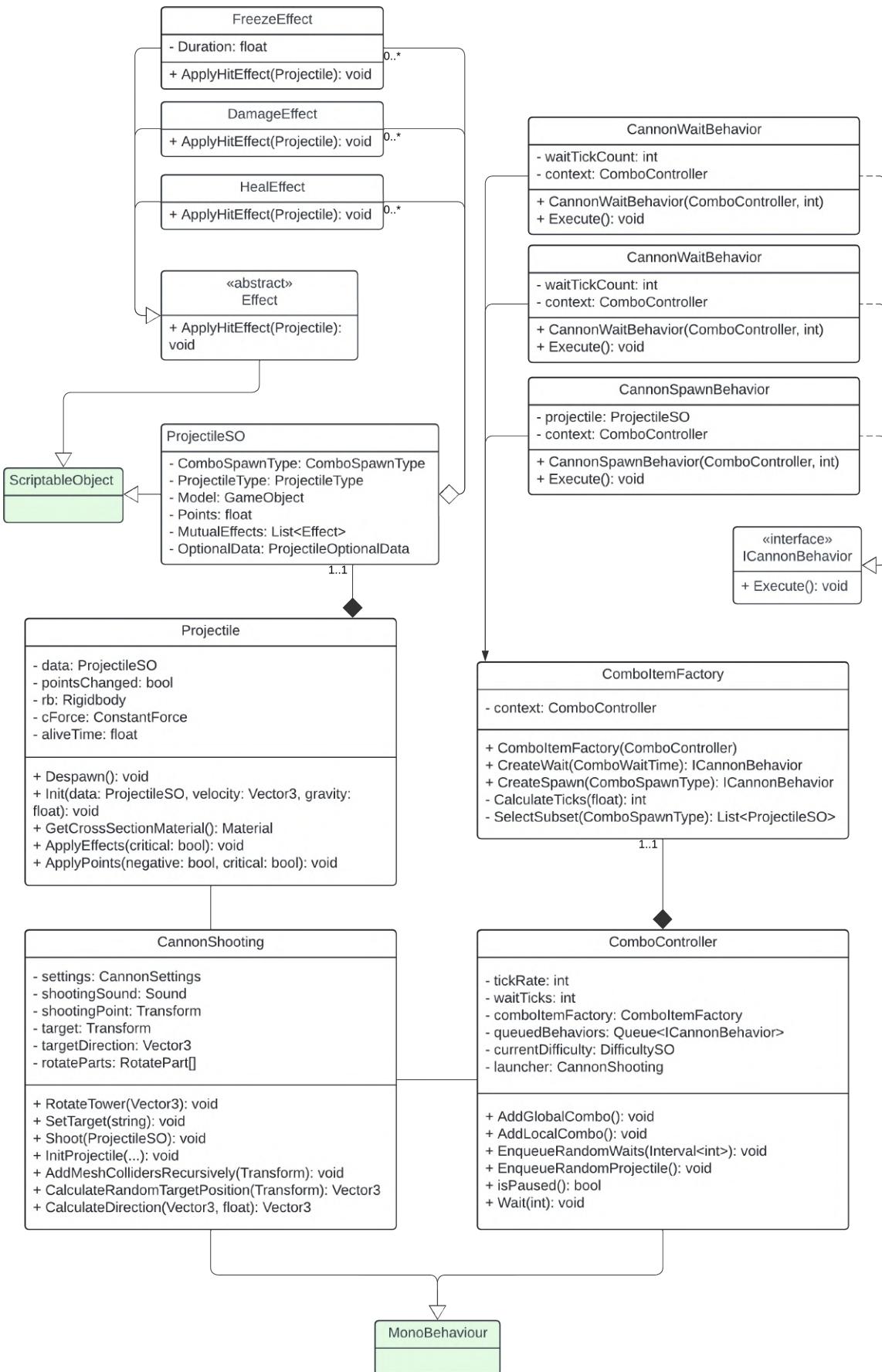
- `void Shoot(ProjectileSO projectile)` - Metoda odpowiadająca za wystrzelenie podanego pocisku w stronę gracza, nadając mu obliczoną prędkość i rotację.
- `Vector3 CalculateDirection(Vector3 target, float gravity = -9.81f)` - Metoda służąca do obliczenia prędkości i rotacji niezbędnych do nadania pociskowi odpowiednich właściwości.



Rysunek 5.3: Zrzut ekranu przedstawiający konfigurację kombinacji pocisków.



Rysunek 5.4: Zrzut ekranu przedstawiający zestawienie wszystkich dostępnych kombinacji pocisków na ekranie zarządzania zasobami Addressables.



Rysunek 5.5: Diagram przedstawiający hierarchię klas opisującą zależności pomiędzy armatą i pociskami.

Właściwości: Klasa CannonShooting

- `CannonSettings Settings [get]` - Zestaw globalnych ustawień.

Metody prywatne: Klasa CannonShooting

- `void RotateTower(Vector3 dir)` - Metoda zapewniająca ciągłą korektę rotacji, aby armata była obrócona w naturalny sposób wzduż trajektorii pocisku oraz w stronę gracza.
- `void SetTarget(string tag)` - Metoda służąca do ustawienia celu, na który armata będzie celować przez cały swój cykl życia. Standardowo używana jest inicjalizacja na tagu "Player".
- `void InitProjectile(ProjectileSO data, Vector3 direction, float gravity)` - Metoda nadająca wszystkie niezbędne właściwości pociskowi w momencie jego wystrzału.

Atrybuty prywatne: Klasa CannonShooting

- `Sound shootingSound` - Dźwięk wystrzału pocisku przez armatę.
- `Transform shootingPoint` - Punkt w lufie armaty, w którym pociski mają się pojawiać na początkowym etapie wystrzału.
- `Transform target` - Referencja do pozycji celu armaty.
- `Vector3 targetDirection` - Rotacja armaty niezbędna do prawidłowego ustawienia trajektorii.
- `RotatePart[] rotateParts` - Wszystkie części armaty podlegające rotacji.

ComboController

Klasa odpowiada za kontrolę zachowania się kombinacji pocisków w grze. Dziedziczy ona po klasie Unity, `MonoBehaviour`, co umożliwia jej bezpośrednie umieszczenie na obiekcie gry - cannon.

Metody publiczne : Klasa ComboController

- `void UpdateOnTick()` - Aktualizuje kontroler kombinacji pocisków podczas każdego taktu systemu.
- `void Wait(int waitTick)` - Ustawia czas oczekiwania w taktach.

Właściwości : Klasa ComboController

- `TickRate` - Zwraca częstotliwość taktów kontrolera kombinacji.

Metody prywatne : Klasa ComboController

- `void AddGlobalCombo()` - Dodaje globalną kombinację pocisków do kolejki zachowań.
- `void AddLocalCombo()` - Dodaje lokalną kombinację pocisków do kolejki zachowań.
- `void EnqueueRandomWaits(Interval<int> wait)` - Dodaje losowe oczekiwania do kolejki zachowań.
- `void EnqueueRandomProjectile()` - Dodaje losowy pocisk do kolejki zachowań.
- `void isPaused()` - Pozwala na sprawdzenie, czy armata jest w stanie oczekiwania.

Atrybuty prywatne : Klasa ComboController

- `int tickRate` - Zmienna przechowująca wartość częstotliwości taktów.
- `int waitTicks` - Licznik taktów do oczekiwania.
- `ComboItemFactory comboItemFactory` - Fabryka obiektów sekwencji.
- `List<ICannonBehavior> queuedBehaviors` - Kolejka zachowań armaty oczekujących na wykonanie.
- `Difficulty currentDifficulty` - Obecny poziom trudności.

ICannonBehavior

Interfejs definiuje zachowanie armaty w określonym taktie gry. Jest to klasa abstrakcyjna, która pozwala na implementację różnych rodzajów zachowań armaty i ich wykonywanie w określonych momentach w trakcie rozgrywki. Klasa ta pełni rolę wzorca projektowego Polecenie (Command Pattern) [13].

Metody publiczne : Klasa ICannonBehavior

- `void Execute()` - Metoda ta jest odpowiedzialna za egzekucję określonego zachowania armaty. Implementacja tej metody różni się w zależności od konkretnego rodzaju zachowania, które ma zostać wykonane w danym tiku czasowym.

CannonWaitBehavior

Klasa ta jest implementacją interfejsu `ICannonBehavior` i definiuje zachowanie armaty polegające na oczekiwaniu w danym taktie. Ten konkretny rodzaj zachowania armaty pozwala na wstrzymanie wykonywania innych akcji armaty przez określony czas, zanim zostanie wykonane kolejne zadanie.

Metody publiczne : Klasa CannonWaitBehavior

- `void Execute()` - Metoda ta odpowiada za oczekiwanie w danym takcie, zanim armata przejdzie do kolejnego zadania.

Atrybuty prywatne: Klasa CannonWaitBehavior

- `ComboController context` - Kontekst zarządcy kombinacji pocisków na którym wykonywane są akcje klasy.

CannonSpawnBehavior

Klasa ta jest implementacją interfejsu `ICannonBehavior` i definiuje zachowanie armaty polegające na wystrzeleniu konkretnego typu pocisku w danym taktie.

Metody publiczne : Klasa CannonSpawnBehavior

- `void Execute()` - Metoda ta jest odpowiedzialna za wystrzelanie określonego typu pocisku w danym taktie, co stanowi główne zadanie tej klasy.

Atrybuty prywatne: Klasa CannonSpawnBehavior

- `ProjectileSO projectile` - Informacje o pocisku który ma zostać wystrzelony.
- `ComboController context` - Kontekst zarządcy kombinacji pocisków na którym wykonywane są akcje klasy.

ComboItemFactory

Klasa ta odpowiada za dynamiczne tworzenie poszczególnych części sekwencji zachowań armaty w grze. Jest to istotny element, umożliwiający konstruowanie różnorodnych akcji związanych z zachowaniem armaty w zależności od typu i czasu oczekiwania. Klasa ta pełni rolę wzorca projektowego Fabryka abstrakcyjna (Abstract Factory) [14].

Metody publiczne : Klasa ComboItemFactory

- `ICannonBehavior CreateWait(ComboWaitTime wait)` - Metoda tworzy obiekt klasy `ICannonBehavior`, reprezentujący oczekiwanie. Jako argument przyjmuje wartość zdefiniowaną w klasie wyliczeniowej `ComboWaitTime`, zawierającą predefiniowane długości oczekiwania.
- `ICannonBehavior CreateSpawn(ComboSpawnType spawn)` - Ta metoda jest odpowiedzialna za tworzenie obiektu klasy reprezentującej pojawienie się nowych pocisków. Jako argument przyjmuje typ pocisku zdefiniowany w klasie `ComboSpawnType`.

Metody prywatne : Klasa ComboItemFactory

- `void CalculateTicks(float seconds)` - Metoda pozwalająca na wyliczenie dokładnej liczby taktów, które należy odczekać na podstawie określonej długości czasu oczekiwania podanej w sekundach.
- `List<ProjectileSO> SelectSubset(ComboSpawnType type)` - Ta metoda jest odpowiedzialna za wybór podzbioru pocisków na podstawie określonego typu. Zwraca listę obiektów `ProjectileSO`, na której będzie wylosowany pocisk związany z danym typem.

Atrybuty prywatne : Klasa ComboItemFactory

- `ComboController context` - Kontekst zarządcy kombinacji pocisków który przekazywany jest do poszczególnych klas zachowań armaty.

Projectile

Klasa zarządza pociskiem w grze. Dziedziczy ona funkcjonalności z klasy Unity, `MonoBehaviour`, co umożliwia bezpośrednie umieszczenie jej na obiekcie gry - `projectile`.

Metody publiczne : Klasa Projectile

- `void Init(ProjectileSO data, Vector3 velocity, float gravity)` - Inicjalizuje pocisk, przygotowując go do wystrzału. Przyjmuje dane pocisku, wyliczoną prędkość oraz kąt wymagany do trafienia w gracza, uwzględniając aktualną wartość grawitacji.
- `Material GetCrossSectionMaterial()` - Zwraca materiał nakładany na płaszczyznę przecięcia dwóch części pocisku. W przypadku braku przypisanego materiału, funkcja tworzy nowy, tymczasowy materiał o czerwonym kolorze.
- `void ApplyEffects(bool critical)` - Nakłada wszelkie efekty związane z pociskiem na rozgrywkę, uwzględniając możliwość krytycznego trafienia.
- `void ApplyPoints(bool negative, bool critical)` - Dodaje lub odejmuje punkty do wyniku gracza w zależności od właściwości pocisku. W przypadku trafienia krytycznego zmienia ilość punktów.

Właściwości : Klasa Projectile

- `ProjectileSO Data [get]` - Udostępnia dane definiujące wszystkie właściwości danego pocisku.
- `PointsChanged pointsChanged [get]` - Flaga sprawdzająca, czy punkty zostały już nadane. Przydatna w przypadku, gdy chcemy uniknąć wielokrotnego dodawania punktów za jedno uderzenie w obiekt.

Metody prywatne : Klasa Projectile

- `void Despawn()` - Usuwa pocisk ze sceny z towarzyszącą mu animacją, gdy ten pozostaje na scenie zbyt długo lub gdy wpada do wody.
- `void ApplyCritical()` - Realizuje efekty specjalne po perfekcyjnym zniszczeniu obiektu. Jest to szczególnie istotne w przypadku trafienia krytycznego.

Atrybuty prywatne : Klasa Projectile

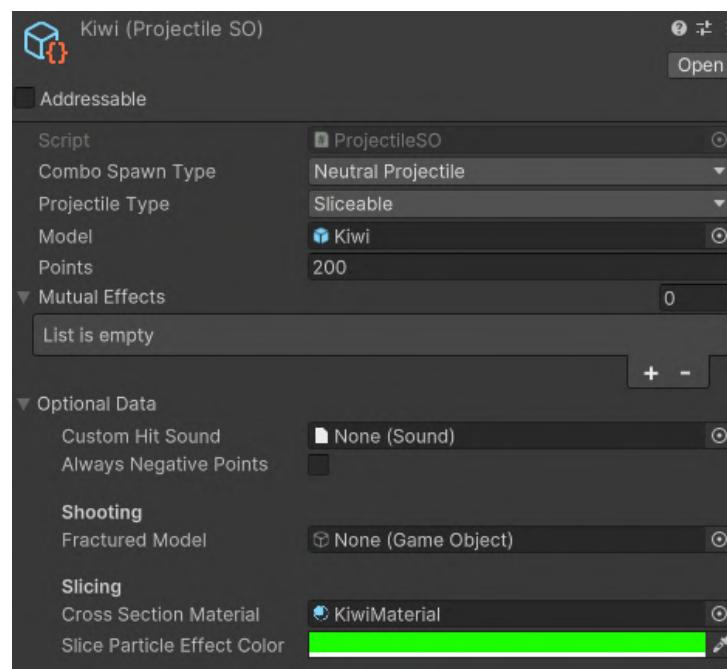
- `Rigidbody rb` - Referencja do komponentu odpowiadającego za fizykę obiektu pocisku.
- `ConstantForce cForce` - Referencja do dodatkowej siły grawitacyjnej obiektu pocisku.
- `float aliveTime` - Czas, przez który pocisk pozostaje aktywny na scenie, określający jego czas życia.

ProjectileSO

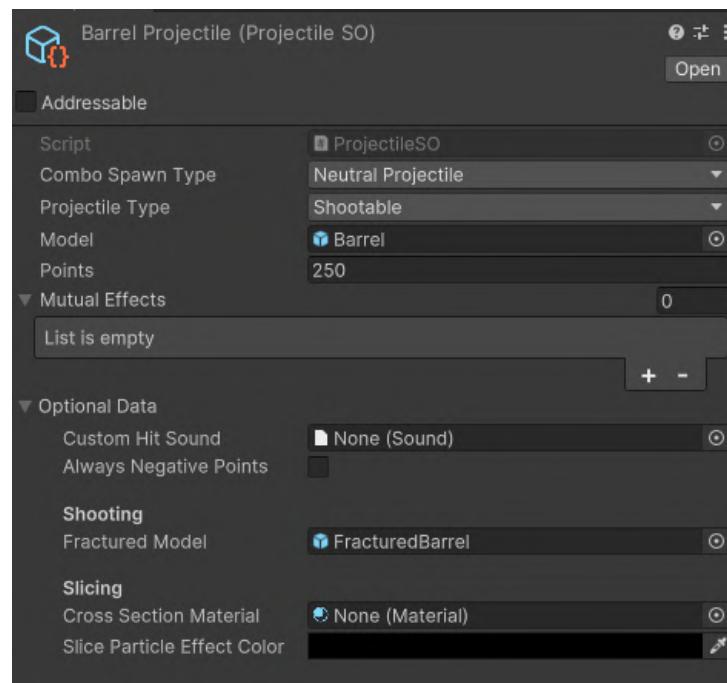
Klasa odpowiada za przechowywanie informacji dotyczące danego pocisku. Dziedziczenie po klasie Unity, `ScriptableObject`, co umożliwia wygodną modyfikację instancji w inspektorze edytora Unity, znaczco ułatwiając konfigurację (Rys. 5.6, 5.7, 5.8).

Właściwości : Klasa ProjectileSO

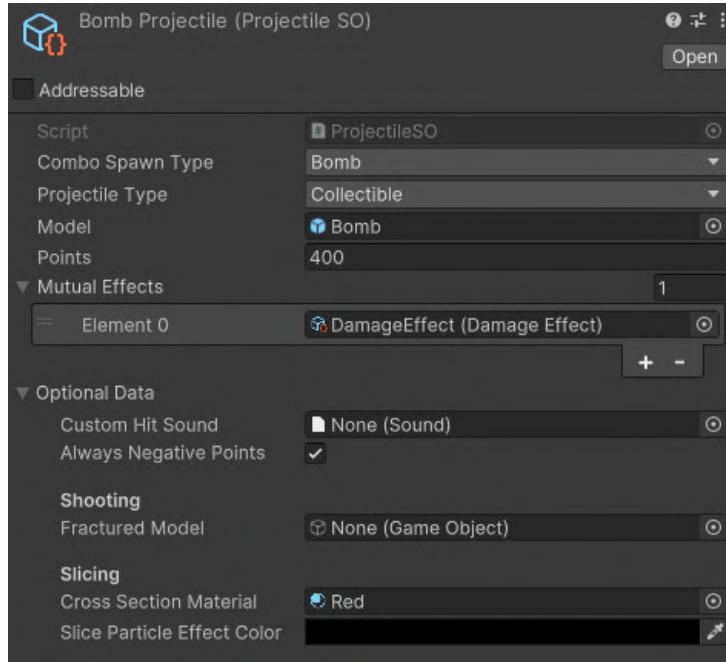
- `ComboSpawnType ComboSpawnType [get]` - Typ wyliczeniowy ustalająca typ pocisku w kontekscie jego znaczenia w przypadku określania kombinacji pocisków.
- `ProjectileType ProjectileType [get]` - Typ wyliczeniowy ustalająca typ pocisku w kontekscie jego wrażliwości na różne typy broni.
- `GameObject Model [get]` - Referencja do modelu graficznego obiektu.
- `float Points [get]` - Przechowuje wartość punktów przyznawanych po zniszczeniu danego pocisku.
- `List<Effect> MutualEffects [get]` - Lista efektów specyficznych dla danego pocisku, nakładanych po jego zniszczeniu.
- `ProjectileOptionalData OptionalData [get]` - Zawiera dodatkowe opcjonalne informacje o danym pocisku.



Rysunek 5.6: Zrzut ekranu przedstawiający konfigurację wybranego pocisku wrażliwego na przeciecia.



Rysunek 5.7: Zrzut ekranu przedstawiający konfigurację wybranego pocisku wrażliwego na zestrzelenie.



Rysunek 5.8: Zrzut ekranu przedstawiający konfigurację pocisku bomby.

ProjectileOptionalData

Klasa przechowuje dodatkowe informacje dotyczące zachowań i wyglądu pocisków w grze.

Właściwości : Klasa ProjectileOptionalData

- `Sound CustomHitSound [get]` - Określa dźwięk, który jest odtwarzany po trafieniu pocisku przez broń. Ten dźwięk jest niezależny od efektów dźwiękowych Szabli czy Pistoletu.
- `bool AlwaysNegativePoints [get]` - Flaga określająca, czy pocisk zawsze aplikuje punkty ujemne.
- `GameObject FracturedModel [get]` - Reprezentuje alternatywny model graficzny pocisku, który jest używany po jego zestrzeleniu przez pistolet.
- `Material CrossSectionMaterial [get]` - Materiał nakładany na powierzchnię przecięcia dwóch części pocisku.
- `Color SliceParticleEffectColor [get]` - Kolor, który jest przekazywany jako parametr do efektu wyrzutu miąższu z pocisku po jego przecięciu.

IEffect

To abstrakcyjna klasa, odpowiedzialna za implementację różnorodnych efektów, które mają być wywoływanie po zniszczeniu pocisku przez broń. Dziedziczenie po klasie Unity,

`ScriptableObject`, co umożliwia wygodną modyfikację instancji w inspektorze edytora Unity, znacząco ułatwiając konfigurację. Klasa ta pełni rolę wzorca projektowego Polecenie (Command Pattern) [13].

Metody publiczne : Klasa `IEffect`

- `void ApplyHitEffect(Projectile context)` - Jest to abstrakcyjna metoda, która nakłada konkretny efekt na podstawie zniszczenia pocisku. Przyjmuje jako argument obiekt `Projectile`, umożliwiając dostęp do kontekstu zniszczonego pocisku w celu zastosowania odpowiedniego efektu.

`HealEffect`

Metody publiczne : Klasa `HealEffect`

- `void ApplyHitEffect(Projectile context)` - Metoda nakładająca efekt regeneracji jednego punktu życia u gracza.

`DamageEffect`

Metody publiczne : Klasa `DamageEffect`

- `void ApplyHitEffect(Projectile context)` - Metoda nakładająca efekt utraty punktów życia u gracza.

`FreezeEffect`

Metody publiczne : Klasa `FreezeEffect`

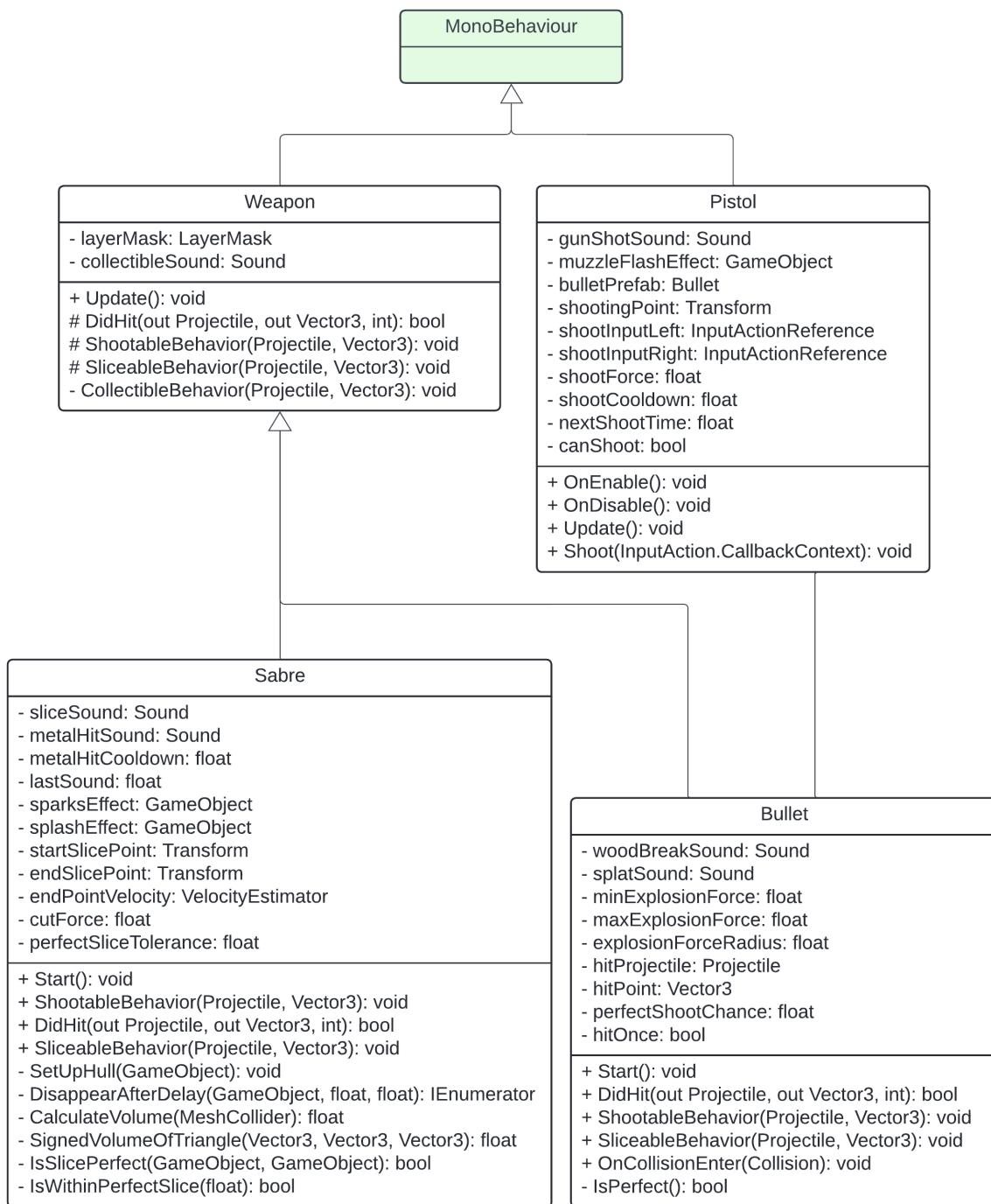
- `void ApplyHitEffect(Projectile context)` - Metoda nakładająca efekt spowolnienia czasu po zniszczeniu pocisku.

Właściwości : Klasa `FreezeEffect`

- `float Duration` - Określa czas trwania efektu spowolnienia czasu.

5.2.3 Bronie

W grze, bronie stanowią kluczowe narzędzia gracza, umożliwiające niszczenie pocisków wystrzeliwanych z armat. Hierarchia klas związana z obsługą broni została zobrazowana na rysunku 5.9. Założono, że gracz dysponuje szabłą w jednej ręce oraz pistoletem w drugiej. Pistolet strzela pociskami, które po pierwszym trafieniu zostają dezaktywowane, zaś szabla, korzystając z rozszerzenia `VelocityEstimator`, dokonuje ciągłych obliczeń płaszczyzny przecięcia miecza. W przypadku trafienia o wystarczającej sile, szabla dzieli pocisk na dwie połówki, wykorzystując w tym celu bibliotekę `EzySlice`.



Rysunek 5.9: Diagram przedstawiający hierarchię klas opisujący zależności pomiędzy uzbrojeniem gracza.

Weapon

Abstrakcyjna klasa bazowa odpowiedzialna za zarządzanie i obsługę funkcjonalności powiązanych z korzystaniem z broni w grze.

Metody chronione : Klasa Weapon

- `SliceableBehavior(Projectile projectile, Vector3 point)` - Metoda obsługująca zachowanie broni w kontakcie z pociskiem wrażliwym na przecięcia. Przyjmuje jako parametry obiekt pocisku oraz punkt kontaktu, w którym następuje interakcja.
- `ShootableBehavior(Projectile projectile, Vector3 point)` - Metoda obsługująca zachowanie broni w kontakcie z pociskiem wrażliwym na zestrzelenie. Przyjmuje jako parametry obiekt pocisku oraz punkt kontaktu, w którym następuje interakcja.
- `DidHit(out Projectile hit, out Vector3 point, int projectileLayer)` - Metoda ta sprawdza w każdej klatce czy broń jest w kontakcie z pociskiem, zwracając referencję do trafionego obiektu oraz punkt kontaktu, co umożliwia obsługę trafień w systemie gry.

Metody prywatne : Klasa Weapon

- `void CollectibleBehavior(Projectile projectile, Vector3 point)` - Metoda obsługująca zachowanie broni w kontakcie z pociskiem specjalnym. Przyjmuje jako parametry obiekt pocisku oraz punkt kontaktu, w którym następuje interakcja.

Atrybuty prywatne : Klasa Weapon

- `LayerMask layerMask` - Atrybut ten przechowuje informacje o warstwach obiektów, z którymi broń może wchodzić w interakcję.
- `Sound collectibleSound` - Jest to atrybut przechowujący referencję do dźwięku, który jest odtwarzany w momencie interakcji broni z obiektem specjalnym.

Sabre

Klasa odpowiada za obsługę szabli. Dziedziczy ona po klasie `Weapon`, co umożliwia jej bezpośrednie umieszczenie na obiekcie gry - sabre.

Metody chronione : Klasa Sabre

- `void SliceableBehavior(Projectile projectile, Vector3 point)` - Metoda ta została nadpisana z klasy bazowej i definiuje zachowanie szabli po zetknięciu z obiektem wrażliwym na przecięcie. Po poprawnym trafieniu, powoduje rozpad

pocisku na dwie osobne połówki, które dynamicznie powstają na podstawie płaszczyzny przecięcia obiektu. Ponadto, tworzony jest efekt wystrzelanego miąższu z pocisku o określonym kolorze.

- `void ShootableBehavior(Projectile projectile, Vector3 point)` - Metoda ta została nadpisana z klasy bazowej i definiuje zachowanie szabli po zetknięciu z obiektem wrażliwym na zestrzelenie. Odejmuje punkty i generuje efekt iskier z punktu kontaktu, skierowany w stronę przeciwną do obiektu.
- `void DidHit(out Projectile projectile, out Vector3 point, int layerMask)` - Na podstawie prędkości i położenia szabli, metoda ta określa, jaki typ pocisku został uderzony.

Metody prywatne : Klasa Sabre

- `void CalculateVolume(MeshCollider collider)` - Oblicza objętość siatki obiektu.
- `void SignedVolumeOfTriangle(Vector3 p1, Vector3 p2, Vector3 p3)` - Metoda obliczająca objętość jednego czworościanu powstałego z jednego trójkąta składającego się na siatkę obiektu.
- `void IsSlicePerfect(GameObject upperHull, GameObject lowerHull)` - Sprawdza, czy dwie połówki obiektu powstałe z przecięcia pocisku objętościowo są identyczne, z zachowaną tolerancją zależną od poziomu trudności.
- `void IsWithinPerfectSlice(float value)` - Metoda sprawdzająca, czy dana wartość objętości zawiera się w danej tolerancji perfekcyjnego przecięcia.
- `void SetUpHull(GameObject hull)` - Inicjalizuje konkretną połówkę obiektu powstałą z przecięcia pocisku, nadając jej m.in. prędkość w stronę przeciwną do Szabli i materiał nałożony na płaszczyznę przecięcia.
- `IEnumerator DisappearAfterDelay(GameObject obj, float delay, float animationTime)` - Korutyna, która powoduje zniknięcie danej połówki obiektu po określonym czasie z animacją.

Atrybuty prywatne : Klasa Sabre

- `Sound sliceSound` - Dźwięk przecięcia obiektu wrażliwego na przecięcie.
- `Sound metalHitSound` - Dźwięk uderzenia w obiekt wrażliwy na zestrzelenia.
- `float metalHitCooldown` - Minimalny czas, jaki system musi odczekać przed ponownym odtworzeniem dźwięku uderzenia w obiekcie wrażliwym na zestrzelenia.
- `GameObject sparksEffect` - Wizualny efekt iskier.

- `GameObject splashEffect` - Wizualny efekt miąższa.
- `Transform endSlicePoint` - Końcowy punkt ostrza szabli.
- `Transform startSlicePoint` - Początkowy punkt ostrza szabli.
- `VelocityEstimator endPointVelocity` - Estymator prędkości nałożony na górną część ostrza szabli.
- `float cutForce` - Siła odepchnięcia połówki pocisku po jego przecięciu.
- `float perfectSliceTolerance` - Tolerancja stosowana do obliczenia perfekcyjnego przecięcia.

Bullet

Klasa Bullet odpowiada za obsługę pocisku wystrzelanego przez pistolet skałkowy. Dziedziczy ona po klasie Unity, `Weapon`, co umożliwia jej bezpośrednie umieszczenie na obiekcie gry - bullet.

Metody chronione : Klasa Bullet

- `void SliceableBehavior(Projectile projectile, Vector3 point)` - Nadpisanie metody z klasy bazowej, która definiuje zachowanie kuli po zetknięciu z obiektem wrażliwym na przecięcie. Metoda ta odejmuje punkty i odrzuca obiekt w przeciwnym kierunku, wydając dźwięk zmiażdżonego owocu.
- `void ShootableBehavior(Projectile projectile, Vector3 point)` - Nadpisanie metody z klasy bazowej, która definiuje zachowanie kuli po zetknięciu z obiektem wrażliwym na zestrzelenie. Ta metoda dodaje punkty za poprawne trafienie, a także powoduje podział pocisku na małe fragmenty, którym nadawana jest eksplozywna siła w punkcie kontaktu z kulą.
- `void DidHit(out Projectile projectile, out Vector3 point, int layerMask)` - Metoda na podstawie kolizji kuli sprawdza, jaki typ pocisku został uderzony.

Metody prywatne : Klasa Bullet

- `void IsPerfect()` - Metoda wyliczająca na podstawie szansy na trafienie perfekcyjne, czy takie trafienie nastąpiło.

Atrybuty prywatne : Klasa Bullet

- `Sound woodBreakSound` - Dźwięk odtwarzany po zderzeniu się kuli z obiektem wrażliwym na zestrzelenie.

- `Sound splatSound` - Dźwięk odtwarzany po zderzeniu się kuli z obiektem wrażliwym na przecięcie.
- `float minExplosionForce` - Dolna granica losowej siły eksplozywnej nakładanej na fragmenty obiektu wrażliwego na zestrzelenie.
- `float maxExplosionForce` - Górna granica losowej siły eksplozywnej nakładanej na fragmenty obiektu wrażliwego na zestrzelenie.
- `float explosionForceRadius` - Zasięg eksplozywnej siły nakładanej na fragmenty obiektu wrażliwego na zestrzelenie.
- `Projectile hitProjectile` - Flaga określająca, czy dana kula uderzyła już w jakiś pocisk.
- `Vector3 hitPoint` - Pierwszy punkt uderzenia kuli w pocisk w przestrzeni świata.
- `float perfectShootChance` - Szansa na trafienie perfekcyjne pocisku.

Pistol

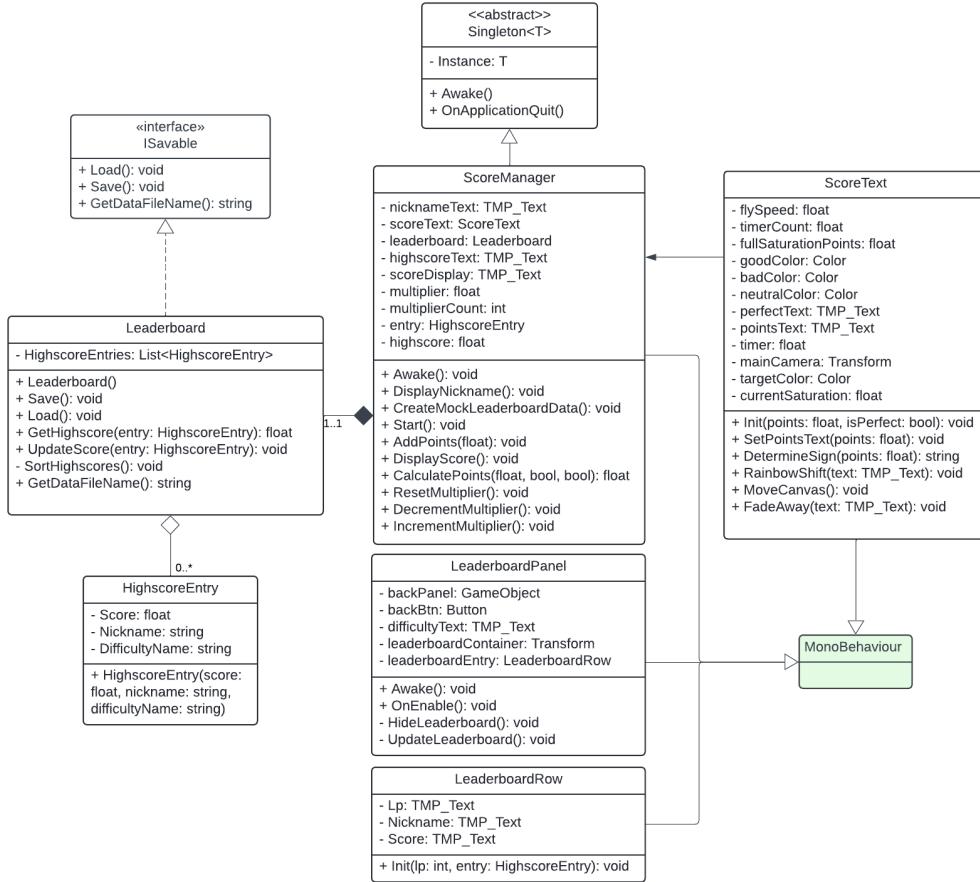
Klasa odpowiada za zarządzanie pistoletem w grze. Dziedziczy ona po klasie Unity, `MonoBehaviour`, co umożliwia jej bezpośrednie umieszczenie na obiekcie gry - pistol.

Metody publiczne: Klasa Pistol

- `void Shoot(InputAction.CallbackContext context)` - Metoda odpowiedzialna za obsługę funkcjonalności wystrzelenia kuli z pistoletu. Przycisk akcji wystrzału jest zależny od ręki, w której trzymana jest broń.

Atrybuty prywatne: Klasa Pistol

- `Sound gunShotSound` - Dźwięk wystrzału z pistoletu.
- `GameObject muzzleFlashEffect` - Efekt błysku z lufy po wystrzale.
- `Bullet bulletPrefab` - Prefabrykat wystrzeliwanej kuli.
- `Transform shootingPoint` - Punkt w lufie, z którego wystrzeliwane są kule.
- `InputActionReference shootInputLeft` - Referencja do przycisku wystrzału na lewym kontrolerze.
- `InputActionReference shootInputRight` - Referencja do przycisku wystrzału na prawym kontrolerze.
- `float shootForce` - Siła, z jaką wystrzeliwana jest kula.



Rysunek 5.10: Diagram przedstawiający hierarchię klas opisujący zależności pomiędzy elementami tworzącymi system punktacji.

- `float shootCooldown` - Minimalny czas pomiędzy wystrzałami.
- `bool canShoot` - Flaga informująca, czy w danej klatce można już oddać kolejny strzał.

5.2.4 Punktacja i tabela wyników

Prezentacja punktacji gracza odbywa się na trzy różne sposoby. Pierwszym z nich jest natychmiastowe wyświetlanie zdobytych punktów po zniszczeniu pocisku. Ten proces kontrolowany jest przez klasę `ScoreText`. Kolejnym sposobem jest sprawdzenie wyniku w tabeli wyników. Nad tą funkcją czuwa klasa `LeaderboardPanel`. Dodatkowo, informacje o wynikach mogą być wyświetlane na tablicy informacyjnej. Zarządzanie tymi operacjami należy do obowiązków `ScoreManager`, który nadzoruje zarząd tablicy informacyjnej oraz wszelkie inne zależności związane z punktacją. Hierarchia zależności klas systemu punktacji została zobrazowana na rysunku 5.10.

ScoreText

Klasa obsługująca tekst punktacji wyświetlany w przestrzeni gry po zniszczeniu pociągu.

Metody publiczne : Klasa ScoreText

- `void Init(float points, bool isPerfect)` - Inicjuje tekst punktacji z danymi punktów i statusu *Perfect*.

Metody prywatne : Klasa ScoreText

- `SetPointsText(float points)` - Ustawia tekst punktów i dostosowuje jego kolor na podstawie danych punktów.
- `DetermineSign(float points)` - Określa znak punktów i zwraca odpowiadający mu ciąg znakowy. W przypadku wyniku dodatniego dodaje przed symbol "+".
- `FlyAndFade()` - Korutyna kontrolująca lot i zanikanie tekstu punktacji.
- `RainbowShift(TMP_Text text)` - Metoda stopniowo zmieniająca kolor tekstu przez spektrum tęczy w przypadku zniszczenia perfekcyjnego.
- `MoveCanvas()` - Przenosi tekst na kanwie w stronę głównej kamery i dostosowuje jego obrót.
- `FadeAway()` - Stopniowo wygasza aktualny tekst.

Atrybuty prywatne : Klasa ScoreText

- `float flySpeed` - Prędkość poruszania się i zanikania tekstu.
- `float timerCount` - Czas trwania przed zniknięciem tekstu.
- `float fullSaturationPoints` - Punkty, przy których tekst osiąga pełną saturację koloru.
- `Color goodColor` - Kolor dla dodatnich punktów.
- `Color badColor` - Kolor dla ujemnych punktów.
- `Color neutralColor` - Kolor dla neutralnych punktów.
- `Color targetColor` - Docelowy kolor.
- `float currentSaturation` - Aktualna saturacja.
- `TMP_Text perfectText` - Obiekt tekstu do wyświetlania wiadomości "Perfect".
- `TMP_Text pointsText` - Obiekt tekstu do wyświetlania zdobytych punktów.
- `Transform mainCamera` - Referencja do głównej kamery.

Leaderboard

Klasa reprezentująca tablicę wyników, zarządzającą wpisami najlepszych wyników i dostarczającą metody do aktualizacji, pobierania i zapisywania wyników.

Metody publiczne : Klasa Leaderboard

- `float GetHighscore(HighscoreEntry entry)` - Pobiera najwyższy wynik dla danego wpisu.
- `void UpdateScore(HighscoreEntry entry)` - Aktualizuje wpis na liście najlepszych wyników lub dodaje nowy wpis, jeśli nie istnieje.

Atrybuty prywatne : Klasa Leaderboard

- `List<HighscoreEntry> HighscoreEntries` - Lista wpisów najlepszych wyników.

Metody prywatne : Klasa Leaderboard

- `void SortHighscores()` - Sortuje wpisy najlepszych wyników malejąco według wyniku.

LeaderboardRow

Klasa reprezentująca wiersz w tablicy wyników, wyświetlający wpis z najlepszym wynikiem.

Metody publiczne : Klasa LeaderboardRow

- `void Init(int lp, HighscoreEntry entry)` - Inicjuje wiersz tablicy wyników z danymi wpisu.

Atrybuty prywatne : Klasa LeaderboardRow

- `TMP_Text Lp` - Tekst z miejscem na liście.
- `TMP_Text Nickname` - Tekst z pseudonimem gracza.
- `TMP_Text Score` - Tekst z wynikiem.

LeaderboardPanel

Klasa zarządzająca funkcjonalnością panelu tablicy wyników, aktualizująca i wyświetlająca wpisy najlepszych wyników.

Metody prywatne : Klasa LeaderboardPanel

- `void HideLeaderboard()` - Ukrywa panel tablicy wyników i wyświetla panel tylny.
- `void UpdateLeaderboard()` - Aktualizuje panel tablicy wyników z wpisami najlepszych wyników dla wybranego poziomu trudności.

Atrybuty prywatne : Klasa LeaderboardPanel

- `GameObject backPanel` - Panel tylni.
- `Button backBtn` - Przycisk powrotu.
- `TMP_Text difficultyText` - Tekst z nazwą poziomu trudności.
- `Transform leaderboardContainer` - Kontener dla tablicy wyników.
- `LeaderboardRow leaderboardEntry` - Wiersz tablicy wyników.

HighscoreEntry

Klasa reprezentująca wpis na liście najlepszych wyników zawierający wynik, pseudonim gracza i nazwę poziomu trudności.

Atrybuty publiczne : Klasa HighscoreEntry

- `float Score` - Wynik osiągnięty przez gracza.
- `string Nickname` - Wybrany pseudonim gracza lub nazwa użytkownika.
- `string DifficultyName` - Nazwa poziomu trudności.

5.2.5 Zapis i odczyt danych

Zachowywanie i odczytywanie istotnych danych, które gracz chce przechowywać między sesjami rozgrywki, odbywa się przy użyciu klas opisanych w poniższym rozdziale. Przykład zastosowania tej funkcjonalności na podstawie tabeli wyników został przedstawiony na rysunku 5.10.

ISavable

Interfejs dla obiektów, które można zapisać i wczytać.

Metody publiczne : Klasa ISavable

- `void Load()` - Wczytuje dane dla obiektu.
- `void Save()` - Zapisuje dane dla obiektu.
- `void GetDataFileName()` - Pobiera nazwę pliku danych powiązaną z obiektem.

DataLoader<T> gdzie T to Object

Klasa statyczna odpowiedzialna za ładowanie zasobów określonego typu.

Metody statyczne : Klasa static List<T>

- `void Load(AssetLabelReference label)` - Ładuje zasoby typu T przy użyciu podanej referencji etykiety.

FileManager

Klasa statyczna odpowiedzialna za operacje na plikach.

Metody statyczne : Klasa FileManager

- `static bool WriteToFile(string a_FileName, string a_FileContents)` - Zapisuje zawartość do pliku.
- `static bool LoadFromFile(string a_FileName, out string result)` - Wczytuje zawartość z pliku.

5.2.6 Klasy użytkowe

W tej sekcji odbędzie się przedstawione ważniejsze klasy użytkowe, które stanowią fundament funkcjonowania gry i omówionych wcześniej mechanik.

SceneLoader

Klasa oparta na wzorcu projektowym singleton odpowiedzialna za ładowanie scen.

Metody publiczne : Klasa SceneLoader

- `LoadScene(int sceneIndex)` - Ładuje scenę używając podanego indeksu sceny.
- `LoadScene(SceneEnum sceneEnum)` - Ładuje scenę używając podanego typu wyliczeniowego zawierającego dostępne sceny.

Atrybuty publiczne : Klasa SceneLoader

- `Action OnSceneChanged` - Akcja wywoływana po zmianie sceny.
- `Action OnSceneFullyLoaded` - Akcja wywoływana po pełnym załadowaniu sceny.

Metody prywatne : Klasa SceneLoader

- `IEnumerator LoadSceneRoutine(int sceneIndex)` - Korutyna odpowiedzialna za ładowanie sceny.

Singleton<T> gdzie T to MonoBehaviour

Klasa bazowa zapewniająca pojedynczą statyczną instancję dla klas pochodnych.

Atrybuty statyczne : Klasa Singleton<T>

- `T Instance` - Statyczna instancja typu T. Główny filar wzorca projektowego Singleton.

Metody chronione : Klasa Singleton<T>

- `protected virtual void Awake()` - Wywoływane, gdy instancja skryptu jest ładowana.
- `protected virtual void OnApplicationQuit()` - Wywoływane, gdy aplikacja jest zamkana.

TickEngine

Klasa zajmująca się aktualizacją opartą na taktach i wywoływaniem zdarzeń w określonym tempie taktów.

Metody publiczne : Klasa TickEngine

- `void UpdateTicks(float delta)` - Aktualizuje takty na podstawie upływu czasu.

Atrybuty publiczne : Klasa TickEngine

- `event Action OnTick` - Zdarzenie wywoływanie przy każdym taktie.
- `int TickRate` - Tempo taktów, w jakim są przetwarzane.

Rozdział 6

Weryfikacja i walidacja

6.1 Metody testowania

W ramach projektu wykorzystano różnorodne metody testowania, mające na celu zapewnienie wysokiej jakości aplikacji. Przeprowadzono manualne testy na komputerze osobistym, którego szczegółowa specyfikacja została opisana w rozdziale 6.1.1. Dodatkowo, przeprowadzono testy na urządzeniu VR, Oculus Quest 2, w celu zbadania zachowania aplikacji w tym środowisku.

Testy integracyjne zostały przeprowadzone w środowisku edytora Unity. Poszczególne komponenty, które wymagały współpracy z innymi obiektami, rejestrowały informacje, ostrzeżenia oraz błędy w konsoli edytora. Dzięki temu możliwa była weryfikacja poprawności ich funkcjonowania. Podczas uruchamiania aplikacji w edytorze, sprawdzano poprawność wszystkich systemów reagujących na zdarzenia oraz w razie problemów informowano o błędach.

Dodatkowo, przeprowadzono testy integracji zasobów zarządzanych przez pakiet Addressables. Każde dodanie nowego zasobu do grupy było analizowane pod kątem zależności między pozostałymi zasobami.

Testy całej aplikacji odbyły się również na zbudowanej wersji programu. Podczas tych testów analizowano dziennik błędów oraz wydajność aplikacji. Wydajność monitorowano za pomocą narzędzia Profiler opisanego w rozdziale 3.4.1. Pozwalało to generować wykresy obrazujące zużycie zasobów komputera przez aplikację stworzoną w tym środowisku.

W trakcie analizy wydajnościowej identyfikowano elementy aplikacji, które miały największy wpływ na szybkość generowania klatek w grze. Pozwoliło to na dokładniejsze zrozumienie, które fragmenty aplikacji wymagały optymalizacji w celu poprawy płynności działania.

6.1.1 Konfiguracja środowiska testowego

Do przeprowadzania testów wykorzystano komputer o specyfikacji:

- **Procesor:** AMD Ryzen 5 5600X
- **Karta graficzna:** Nvidia GeForce GTX 1070 GamingX
- **Pamięć RAM:** 32GB (Taktowanie: 3200 MHz)
- **Dysk:** WD_BLACK SN850 NVMe 1TB
 - Szybkość odczytu: 7000 MB/s
 - Szybkość zapisu: 5300 MB/s
- **Płyta główna:** ASUS TUF GAMING B550-PLUS
- **System operacyjny:** Windows 10 Home

6.2 Identyfikacja problemów

Podczas testów ujawnił się główny problem z wydajnością aplikacji, zwłaszcza pod względem liczby klatek na sekundę. W późniejszym etapie rozwoju aplikacji, stwierdzono, że jej działanie jest znacznie spowolnione. Aby temu zaradzić, przeprowadzono kompresję tekstur oraz zoptymalizowano modele, zmniejszając liczbę trójkątów z których jest zbudowana. To pozwoliło na płynniejsze działanie aplikacji na urządzeniach VR.

Rozdział 7

Podsumowanie i wnoski

Cel pracy dyplomowej został osiągnięty, powstała aplikacja która w pełni spełnia wszystkie szczegółowe wymagania oraz cele założone na potrzeby umożliwienia użytkownikowi całkowicie immersywnej rozgrywki zręcznościowej w goglach wirtualnej rzeczywistości. Aplikacja cechuje intuicyjna obsługa, której integralną częścią są kompleksowe samouczki, wprowadzające gracza w dostępne mechaniki. Pomimo zaawansowanej grafiki i efektów wizualnych, dzięki swojej optymalizacji gra utrzymuje standardy obecnie obowiązujące w grach wirtualnej rzeczywistości, co przekłada się na stabilność rozgrywki i komfort użytkowania.

7.1 Możliwości rozwoju

7.1.1 Nowe pociski

W aktualnej wersji gry dostępne jest wyłącznie kilka różnych typów pocisków. Poszerzenie tej funkcjonalności znacząco wzbogaciłoby dynamikę rozgrywki. Taką modyfikację gry można byłoby dokonać nawet bez posiadania żadnych umiejętności programistycznych, wykorzystując intuicyjny interfejs silnika Unity. Dzięki wcześniej przygotowanemu systemowi, dodawanie nowych typów pocisków jest możliwe dla osób niezaznajomionych z programowaniem. Ta opcja otwiera szerokie pole do rozbudowy i dostosowywania rozgrywki, co znacząco zwiększyłoby jej atrakcyjność dla użytkowników.

7.1.2 Tabela wyników dostępna w sieci

Tablica wyników znajduje się w pliku przechowywanym lokalnie, co ogranicza możliwość porównywania wyników graczy jedynie do tych korzystających z tego samego urządzenia. Integracja aplikacji z platformami takimi jak PLAYFAB [34] umożliwiłaby przenesienie tablicy wyników do sieciowej bazy danych, co pozwoliłoby graczom na śledzenie i dzielenie się wynikami w czasie rzeczywistym z innymi użytkownikami.

7.1.3 Wprowadzenie fabuły

Warstwa narracji, która wprowadziłaby gracza w przedstawiony świat, z pewnością wzbogaciłoby głębię rozgrywki, pozwalając na jeszcze większe zanurzenie się użytkownika w grze. Istnieje wiele sposobów osiągnięcia tego celu, takich jak dodanie narratora opowiadającego historię, proste zadania, których ukończenie zapewniłoby dodatkowe informacje, lub system dialogów z postaciami niezależnymi, którego implementacji można dokonać przy użyciu popularnych narzędzi do tworzenia dialogów, jak na przykład Ink [16].

Bibliografia

- [1] *About Shader Graph*. 2020. URL: <https://docs.unity3d.com/Packages/com.unity.shadergraph@14.0/manual/index.html> (term. wiz. 14.12.2023).
- [2] *About the Oculus XR Plugin*. 2020. URL: <https://docs.unity3d.com/Packages/com.unity.xr.oculus@4.1/manual/index.html> (term. wiz. 14.12.2023).
- [3] *Addressables package*. 2020. URL: <https://docs.unity3d.com/Packages/com.unity.addressables@1.21/manual/index.html> (term. wiz. 14.12.2023).
- [4] BeatGames. *Beat Saber*. URL: <https://beatsaber.com/> (term. wiz. 14.12.2023).
- [5] DavidArayan. *ozy-slice*. 2023. URL: <https://github.com/DavidArayan/ozy-slice> (term. wiz. 03.01.2024).
- [6] Android Developers. *SDK Platform Tools release*. 2023-12-11. URL: [#downloads](https://developer.android.com/tools/releases/platform-tools) (term. wiz. 21.12.2023).
- [7] C# Documentation. *Language Integrated Query (LINQ)*. 2023-07-18. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/linq/> (term. wiz. 14.12.2023).
- [8] Unity documentation. *Introduction to components in Unity*. 2023. URL: <https://docs.unity3d.com/Manual/Components.html> (term. wiz. 14.12.2023).
- [9] Unity Documentation. *XR platform system requirements - 2022.3*. 2022-03. URL: <https://docs.unity3d.com/Manual/system-requirements.html> (term. wiz. 21.12.2023).
- [10] *Doxygen documentation*. 2024. URL: <https://www.doxygen.nl/> (term. wiz. 09.01.2024).
- [11] fuqunaga. *JsonUtilityEx*. 2023. URL: <https://gist.github.com/fuqunaga/b50b49cc08010ba37b07ac01c401a8f0> (term. wiz. 03.01.2024).
- [12] Daniele Giardini. *DoTween documentation*. 2020. URL: <https://dotween.demigiant.com/> (term. wiz. 14.12.2023).
- [13] Refactoring guru. *Command pattern*. 2014-2023. URL: <https://refactoring.guru/pl/design-patterns/command/csharp/example> (term. wiz. 14.12.2023).
- [14] Refactoring guru. *Factory pattern*. 2014-2023. URL: <https://refactoring.guru/pl/design-patterns/abstract-factory> (term. wiz. 14.12.2023).

- [15] Halfbrick. *Fruit Ninja VR*. URL: <https://www.halfbrick.com/games/fruit-ninja-vr> (term. wiz. 14.12.2023).
- [16] inkle. *Ink*. 2022. URL: <https://github.com/inkle/ink> (term. wiz. 03.01.2024).
- [17] *Input system*. 2020. URL: <https://docs.unity3d.com/Packages/com.unity.inputsystem@1.7/manual/index.html> (term. wiz. 14.12.2023).
- [18] Cassie Bottorff Laura Hennigan. *What Is A Kanban Board? The Ultimate Guide*. 2022-03-27. URL: <https://www.forbes.com/advisor/business/software/what-is-kanban-board/> (term. wiz. 20.12.2023).
- [19] Marco A Gerosa Mairieli Wessel Joseph Vargovich. *GitHub Actions: The Impact on the Pull Request Process*. 2022-06. URL: https://www.researchgate.net/publication/361600523_GitHub_Actions_The_Impact_on_the_Pull_Request_Process (term. wiz. 15.12.2023).
- [20] Meta. *Get started in VR*. 2023. URL: <https://www.meta.com/pl/en/quest/setup/> (term. wiz. 21.12.2023).
- [21] Meta. *SPECYFIKACJA TECHNICZNA*. 2023. URL: <https://www.meta.com/pl/quest/products/quest-2/tech-specs/#tech-specs> (term. wiz. 15.12.2023).
- [22] *.NET Documentation*. 2022. URL: <https://learn.microsoft.com/en-us/dotnet/> (term. wiz. 03.01.2024).
- [23] *SteamVR API Documentation*. 2023. URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/Valve.VR.InteractionSystem.VelocityEstimator.html (term. wiz. 03.01.2024).
- [24] SuperHotTeam. *SuperHotVR*. URL: <https://superhotgame.com/> (term. wiz. 14.12.2023).
- [25] Nicholas Sutrich. *What's new with the Oculus Quest 2 controllers?* 2020. URL: <https://www.androidcentral.com/whats-new-oculus-quest-2-controllers> (term. wiz. 07.01.2024).
- [26] *TextMesh Pro User Guide*. 2020. URL: <https://docs.unity3d.com/Packages/com.unity.textmeshpro@1.3/manual/index.html> (term. wiz. 14.12.2023).
- [27] Steve McGreal Thomas Krogh-Jacobsen. *Profiling in Unity 2021 LTS: What, when, and how*. 2022. URL: <https://blog.unity.com/engine-platform/profiling-in-unity-2021-lts-what-when-and-how> (term. wiz. 09.01.2024).
- [28] *Top IDE index*. 2023. URL: <https://pyp1.github.io/IDE.html> (term. wiz. 14.12.2023).
- [29] *Unity Documentation*. 2023. URL: <https://docs.unity3d.com/Manual/index.html> (term. wiz. 14.12.2023).

- [30] Pablo Dornhege Vincent Kaufmann Franziska Ritter. *Quickly explained: The VR headset jungle*. URL: <https://digital.dthg.de/en/quickly-explained-the-vr-headset-jungle/> (term. wiz. 07.01.2024).
- [31] *Visual Effect Graph*. 2020. URL: <https://docs.unity3d.com/Packages/com.unity.visualeffectgraph@12.0/manual/index.html> (term. wiz. 14.12.2023).
- [32] *What Is Json*. 2024. URL: <https://www.oracle.com/pl/database/what-is-json/> (term. wiz. 08.01.2024).
- [33] *What is open source?* 2024. URL: <https://opensource.com/resources/what-open-source> (term. wiz. 14.12.2023).
- [34] *What is PlayFab?* 2023. URL: <https://learn.microsoft.com/en-us/gaming/playfab/what-is-playfab> (term. wiz. 03.01.2024).

Dodatki

Spis skrótów i symboli

3D trójwymiar,

VR wirtualna rzeczywistość,

FPS klatki na sekundę,

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- dokumentację programu,
- pliki źródłowe programu,
- film pokazujący działanie opracowanej aplikacji.

Spis rysunków

2.1	Przedstawienie gogli VR od różnych producentów, źródło: [30].	3
2.2	Zrzut ekranu przedstawiający fragment rozgrywki gry BeatSaber, źródło: https://store.steampowered.com/app/620980/Beat_Saber/	5
2.3	Zrzut ekranu przedstawiający fragment rozgrywki gry Superhot VR, źródło: https://store.steampowered.com/app/617830/SUPERHOT_VR/	5
2.4	Zrzut ekranu przedstawiający fragment rozgrywki gry Fruit Ninja VR, źródło: https://store.steampowered.com/app/486780/Fruit_Ninja_VR/	6
3.1	Logo silnika Unity, źródło: www.unity.com	10
3.2	Edytor Unity	11
3.3	Interfejs narzędzia Profiler, źródło: [27].	13
3.4	Przykładowy skrypt napisany w języku C# zintegrowany z silnikiem Unity	14
3.5	Logo oprogramowania Blender, źródło: www.blender.org	15
3.6	Google Oculus Quest 2 wraz z dwoma kontrolerami, źródło: https://www.meta.com/gb/quest/safety-center/quest-2/	16
4.1	Sprawdzenie wersji sterownika.	20
4.2	Oczekiwany przykładowy rezultat sprawdzenia wersji sterownika.	20
4.3	Zrzut ekranu z menu aplikacji mobilnej gogli Oculus Quest 2, gdzie zaznaczona jest opcja umożliwiająca włączenie trybu deweloperskiego.	21
4.4	Zrzut ekranu z menu aplikacji gogli Oculus Quest 2 na komputerze stacjonarnym, gdzie zaznaczona jest opcja umożliwiająca instalację aplikacji z nieznanych źródeł.	21
4.5	Wylistowanie podpiętych urządzeń.	22
4.6	Oczekiwany przykładowy rezultat wylistowania podpiętych urządzeń.	22
4.7	Instalacja aplikacji na urządzeniu.	22
4.8	Oczekiwany przykładowy rezultat po instalacji aplikacji na urządzeniu	22
4.9	Zrzut ekranu przedstawiający pasek narzędzi z zaznaczonym kafelkiem <i>App library</i>	22
4.10	Zrzut ekranu przedstawiający bibliotekę aplikacji z rozwiniętym menu i zaznaczoną opcją <i>Unknown sources</i>	23

4.11 Zrzut ekranu przedstawiający poprawnie zainstalowaną aplikację.	23
4.12 Zrzut ekranu widoku rozgrywki z szabłą w prawej dloni.	24
4.13 Mapowanie przycisków kontrolera Oculus Touch, źródło: https://developer.oculus.com/	26
4.14 Zrzut ekranu prezentujący interfejs menu głównego.	27
4.15 Zrzut ekranu prezentujący interfejs ustawień.	27
4.16 Zrzut ekranu prezentujący widok trybu comfort mode w momencie obrotu manualnego.	28
4.17 Zrzut ekranu przedstawiający sekcję about	28
4.18 Przykładowe panele znajdujące się w samouczku.	29
4.19 Zrzut ekranu przedstawiający panel wpisywania pseudonimu gracza.	30
4.20 Zrzut ekranu przedstawiający klawiature systemową.	30
4.21 Zrzut ekranu przedstawiający panel wyboru poziomów trudności.	30
4.22 Zrzut ekranu przedstawiający panel wyboru konfiguracji broni.	31
4.23 Zrzut ekranu przedstawiający menu startowe przed rozgrywką.	31
4.24 Zrzut ekranu przedstawiający tabelę wyników.	32
4.25 Zrzut ekranu przedstawiający samouczek przed rozgrywką.	32
4.26 Zrzut ekranu przedstawiający szabłę.	33
4.27 Zrzut ekranu przedstawiający komunikat o perfekcyjnym przecięciu.	34
4.28 Zrzut ekranu przedstawiający iskry po próbie przecięcia obiektu nieprzystosowanego do przecinania.	34
4.29 Zrzut ekranu przedstawiający pistolet skałkowy.	35
4.30 Zrzut ekranu przedstawiający błysk wystrzału z pistoletu skałkowego.	35
4.31 Wszystkie pociski wrażliwe na przecięcia.	36
4.32 Zrzut ekranu przedstawiający efekt miąższa i rozpadu kokosa na dwie części.	36
4.33 Zrzut ekranu przedstawiający efekt miąższa i rozpadu granatu na dwie części.	37
4.34 Wszystkie pociski wrażliwe na zestrzelenie.	38
4.35 Zrzuty ekranu przedstawiające efekt rozbicia beczki.	39
4.36 Zrzut ekranu przedstawiający bombę.	39
4.37 Zrzut ekranu przedstawiający efekt wybuchu bomby.	40
4.38 Zrzut ekranu przedstawiający klepsydrę.	40
4.39 Zrzut ekranu przedstawiający sztabę złota.	41
4.40 Zrzut ekranu przedstawiający serce.	42
4.41 Zrzut ekranu przedstawiający armatę.	42
4.42 Zrzut ekranu przedstawiający tratwę.	43
4.43 Zrzut ekranu przedstawiający tablicę informacyjną.	44
4.44 Zrzut ekranu przedstawiający rekina.	44

4.45 Zrzut ekranu przedstawiający wyspę i statek piracki i wszystkie możliwe armaty.	45
5.1 Fragment kanwy Shader Graph wykorzystanej do powstania jednostki cie- niującej wody.	49
5.2 Fragment kanwy Visual Effect Graph wykorzystanej do powstania efektu wybuchu bomby.	50
5.3 Zrzut ekranu przedstawiając konfigurację kombinacji pocisków.	56
5.4 Zrzut ekranu przedstawiając zestawienie wszystkich dostępnych kombinacji pocisków na ekranie zarządzania zasobami Addressables.	56
5.5 Diagram przedstawiający hierarchię klas opisujący zależności pomiędzy ar- matą i pociskami.	57
5.6 Zrzut ekranu przedstawiający konfigurację wybranego pocisku wrażliwego na przecięcia.	63
5.7 Zrzut ekranu przedstawiający konfigurację wybranego pocisku wrażliwego na zestrzelenie.	63
5.8 Zrzut ekranu przedstawiający konfigurację pocisku bomby.	64
5.9 Diagram przedstawiający hierarchię klas opisujący zależności pomiędzy uzbrojeniem gracza.	66
5.10 Diagram przedstawiający hierarchię klas opisujący zależności pomiędzy ele- mentami tworzącymi system punktacji.	71

Spis tabel

4.1	Porównanie poziomów trudności	25
4.2	Punkty za przeciecie różnych rodzajów pocisków	37
4.3	Punkty za zestrzelenie różnych rodzajów pocisków.	38