



**Politechnika  
Śląska**

## **PROJEKT INŻYNIERSKI**

Biblioteka umożliwiająca wzbogacenie gry w środowisku Unity zdarzeniami wykorzystującymi śledzenie wzroku

**Jakub FERENS**  
**Nr albumu: 295641**

**Kierunek:** Informatyka  
**Specjalność:** Bazy danych i inżynieria systemów

**PROWADZĄCY PRACĘ**  
**Dr hab. inż. Paweł Kasprowski**  
**KATEDRA Informatyki Stosowanej**  
**Wydział Automatyki, Elektroniki i Informatyki**

**Gliwice 2025**



## **Tytuł pracy**

Biblioteka umożliwiająca wzbogacenie gry w środowisku Unity zdarzeniami wykorzystującymi śledzenie wzroku

## **Streszczenie**

Praca "Biblioteka umożliwiająca integrację zdarzeń bazujących na śledzeniu wzroku w środowisku Unity" demonstruje praktyczne wykorzystanie techniki śledzenia wzroku w kontekście gier wideo. W ramach tej pracy zaprezentowano sposóbłączenia tej innowacyjnej technologii do silnika Unity, aby tworzyć gry, gdzie interakcje i reakcje graczy, zależne od kierunku ich spojrzenia, mają istotny wpływ na dynamikę gry. Projekt obejmował również opracowanie prototypu gry, który uwydatnia potencjał i funkcjonalność śledzenia wzroku w rozgrywce.

## **Słowa kluczowe**

Unity, śledzenie wzroku, interakcje w grach

## **Thesis title**

A library that enriches a game in the Unity environment with eye tracking events

## **Abstract**

The thesis "A library that enriches a game in the Unity environment with eye tracking events" demonstrates the practical application of eye-tracking technology in the context of video games. This work presents a method for incorporating this innovative technology into the Unity engine to create games where player interactions and reactions, dependent on the direction of their gaze, significantly influence game dynamics. The project also included the development of a game prototype, highlighting the potential and functionality of eye tracking in gameplay.

## **Key words**

Unity, eye tracking, game interactions



# Spis treści

<b>1 Wstęp</b>	<b>1</b>
<b>2 Analiza tematu</b>	<b>3</b>
2.1 Sformułowanie problemu . . . . .	4
<b>3 Wymagania i narzędzia</b>	<b>5</b>
3.1 Wymagania funkcjonalne . . . . .	5
3.2 Wymagania niefunkcjonalne . . . . .	6
3.3 Opis narzędzi . . . . .	7
3.3.1 Unity . . . . .	8
3.3.2 Unity ProBuilder . . . . .	8
3.3.3 Visual Studio Code . . . . .	8
3.3.4 Tobii eyetracker 5 . . . . .	8
3.3.5 Tobii Unity SDK 5.0.0.3 . . . . .	8
3.4 Przypadki użycia - diagram UML . . . . .	9
<b>4 Specyfikacja zewnętrzna</b>	<b>11</b>
4.1 Wymagania sprzętowe i programowe . . . . .	11
4.1.1 Sprzęt . . . . .	11
4.1.2 Oprogramowanie . . . . .	11
4.2 Sposób instalacji . . . . .	11
4.3 Sposób obsługi . . . . .	11
4.4 Scenariusze korzystania z systemu . . . . .	12
4.4.1 Obszar startowy . . . . .	12
4.4.2 Pierwszy poziom . . . . .	13
4.4.3 Drugi poziom . . . . .	15
4.4.4 Trzeci poziom . . . . .	18
<b>5 Specyfikacja wewnętrzna</b>	<b>21</b>
5.1 Przedstawienie idei i architektura systemu . . . . .	21
5.1.1 Struktura systemu w unity . . . . .	21
5.1.2 Zalety architektury Unity: . . . . .	22

5.2	Komponenty, moduły, biblioteki, przegląd ważniejszych klas . . . . .	22
5.2.1	Tobii Unity SDK . . . . .	22
5.2.2	Unity ProBuilder . . . . .	22
5.2.3	TextMeshPro . . . . .	22
5.2.4	Shader Graph . . . . .	22
5.2.5	Główne klasy interakcji z obiektami . . . . .	23
5.2.6	Klasy pomocnicze do interakcji . . . . .	23
5.3	Zastosowane wzorce projektowe . . . . .	24
5.3.1	Wzorzec projektowy singleton . . . . .	24
5.3.2	Wzorzec projektowy komponent . . . . .	24
5.4	Przegląd wybranych fragmentów kodu . . . . .	25
5.4.1	Wykrywanie wzroku na podstawie klasy ColorChangeOnGazeChildren . . . . .	25
5.4.2	Poruszanie obiektami z użyciem wzroku na przykładzie klasy MoveOnGaze . . . . .	25
5.4.3	Proste animacje z użyciem ResizeOnGaze . . . . .	28
5.5	Implementacja w Unity . . . . .	30
5.5.1	Mechanizmy zabezpieczające gracza i poprawiające jakość rozgrywki	30
5.5.2	System ekranów menu . . . . .	32
5.5.3	Automatyzacja rozmieszczania obiektów w edytorze . . . . .	33
5.5.4	Tworzenie materiałów z użyciem shadergraph . . . . .	33
5.5.5	Logika 'wsiadania do statku' jako przykład projektowania interakcji z komponentem MoveOnGaze . . . . .	35
5.5.6	MoveOnWayPointsOnGaze: Przykład labiryntu . . . . .	37
<b>6</b>	<b>Weryfikacja i walidacja</b> . . . . .	<b>41</b>
6.1	Sposób testowania w ramach pracy . . . . .	41
6.2	Przypadki testowe zakres testowania . . . . .	41
6.2.1	Test skryptu AutoObjectPlacement . . . . .	42
6.2.2	Test skryptu ColorChangeOnGazeChildren . . . . .	43
6.2.3	Test skryptu ShowSymbolOnGaze oraz materiału indykatów spojrzenia . . . . .	43
6.2.4	Testy związane z wykrywaniem wzroku poszczególnych elementów .	43
6.2.5	Testy skryptu MoveOnGaze . . . . .	44
6.2.6	Testy interakcji ze statkiem . . . . .	44
6.2.7	Testy skryptu MoveWhenNotLookedAt przypisanego do kulek . . . . .	44
6.2.8	Analiza testów funkcjonalnych poziomów gry . . . . .	45
6.3	Wykryte i usunięte błędy . . . . .	45
6.4	Wyniki badań eksperymentalnych . . . . .	46

<b>7 Podsumowanie i wnioski</b>	<b>49</b>
<b>Bibliografia</b>	<b>51</b>
<b>Spis skrótów i pojęć</b>	<b>55</b>
<b>Lista dodatkowych plików, uzupełniających tekst pracy</b>	<b>57</b>
<b>Spis rysunków</b>	<b>60</b>



# Rozdział 1

## Wstęp

Współczesny rozwój technologii informacyjnych i komunikacyjnych nieustannie przekształca różnorodne aspekty życia codziennego, a jednym z jego najbardziej dynamicznych obszarów jest przemysł gier wideo. Wraz z postępem technologicznym, gry wideo ewoluują, oferując użytkownikom coraz to nowsze sposoby interakcji i zaangażowania użytkowników.

Niniejsza praca inżynierska koncentruje się na badaniu i wykorzystaniu technologii śledzenia wzroku w środowisku programistycznym Unity, z wykorzystaniem Tobii EyeTracker 5. Technologia śledzenia wzroku, coraz częściej stosowana w różnych dziedzinach, otwiera nowe perspektywy także w sektorze gier komputerowych, oferując innowacyjne sposoby interakcji.

Celem niniejszego projektu jest eksploracja potencjału, jaki technologia śledzenia wzroku wnosi do świata gier wideo, ze szczególnym uwzględnieniem jej wpływu na sposób, w jaki użytkownicy wchodzą w interakcję z grami. Praca koncentruje się na praktycznym zastosowaniu śledzenia wzroku, realizując projekt w formie demonstracyjnego prototypu, który pozwala na empiryczne ocenienie i zrozumienie zalet oraz ograniczeń tej technologii w kontekście gier wideo. W prototypie demonstracyjnym znajdują się interakcje związane z wykrywaniem spojrzenia na obiekt. Te interakcje obejmują:

- obracanie obiektu
- zmiana koloru obiektu
- zmiana kształtu obiektu
- przemieszczenie się obiektu do danego punktu
- możliwość przesuwania obiektu używając obiektów pomocniczych

Na podstawie tych interakcji zostaną stworzone trzy poziomy demonstracyjne.

Praca została podzielona na rozdziały ze względu na obszary tematyczne, które pełnią rolę etapów projektu:

- Analiza tematu
- Wymagania i narzędzia
- Specyfikacja zewnętrzna
- Specyfikacja wewnętrzna
- Weryfikacja i walidacja
- Podsumowanie i wnioski

## Rozdział 2

### Analiza tematu

Analiza technologii śledzenia wzroku w kontekście gier wideo i aplikacji Unity z wykorzystaniem biblioteki eye trackingu zaczyna się od uwagi na dynamiczny rozwój tej dziedziny. Technologia ta umożliwia głębokie badanie ludzkiej uwagi i percepcji świata poprzez monitorowanie ruchów gałek ocznych oraz identyfikację elementów, na które skupiamy uwagę, często podświadomie. Jest to kluczowe dla zrozumienia interakcji między użytkownikiem a cyfrowym środowiskiem, dając nowe możliwości w projektowaniu interfejsów użytkownika i doświadczeń immersyjnych.

Istotnym elementem w kontekście śledzenia wzroku jest jego aplikacja podczas rozwoju gry do formułowania decyzji kreatywnych. Technologia eye tracking umożliwia głębsze zrozumienie, jak gracze doświadczają gry. Szczegółowe informacje na ten temat znajdują się w artykule [3].

Większość gier głównego nurtu (ang. mainstream) stosuje technologię śledzenia wzroku przede wszystkim do celów dodatkowych, takich jak interakcja z interfejsem użytkownika, rozszerzanie pola widzenia czy uproszczone celowanie. Przykładem może być “Assassin’s Creed® Valhalla”, gdzie za pomocą wzroku możemy próbować celować łukiem postaci, delikatnie przesuwać kamerę w świecie gry, wybierać przedmioty do interakcji lub doświadczać dodatkowych efektów związanych z oświetleniem [8]. Te dodatki raczej wzbogacają nasze doświadczenia z gry, niż stanowią jej rdzeń.

W artykule opublikowanym podczas konferencji SBGames w 2011 roku, autorzy szczegółowo omawiają różnorodne metody implementacji śledzenia wzroku w grach wideo. Rozważane są tam zarówno proste techniki, jak umieszczenie kurSORA w miejscu spojrzenia użytkownika, jak i bardziej zaawansowane podejścia, takie jak modyfikacje kodu gry lub tworzenie gier od podstaw z myślą o wykorzystaniu tej technologii. Artykuł ten analizuje również, jak różne gatunki gier, od gier planszowych po strzelanki i gry akcji, mogą korzystać ze śledzenia wzroku, uwzględniając ich specyficzne wymagania i ograniczenia [1].

Nowatorskim rozwiązaniem jest także wykorzystywanie eyetrackingu do renderowanie fowlane, dzięki tej technice, elementy w centrum uwagi gracza są renderowane z wysoką

rozdzielczością, podczas gdy obszary peryferyjne, poza bezpośrednim polem widzenia, są przedstawiane z niższą jakością. To innowacyjne wykorzystanie śledzenia wzroku pozwala na znaczne oszczędności zasobów systemowych, jednocześnie podnosząc jakość i płynność doświadczenia wizualnego[6].

Warto jednak zauważyć, że potencjał tej technologii wykracza poza te stosunkowo proste zastosowania reprezentowane przez gry głównego nurtu. Brakuje gier, które wykorzystywałyby śledzenie wzroku do bardziej zaawansowanych form interakcji jak przykładowo gra wykorzystująca śledzenie wzroku do wywołania zamieszania u gracza o której możemy przeczytać w artykule[5].

## 2.1 Sformułowanie problemu

Zintegrowanie technologii śledzenia wzroku w grach platformowych jest koncepcją, która do tej pory była stosunkowo niezbadana i rzadko wykorzystywana w głównym nurcie gier wideo. Większość obecnych gier korzysta z tej technologii w sposób marginalny, skupiając się na prostych funkcjach, takich jak interakcja z interfejsem użytkownika czy uproszczone celowanie. Istnieje znaczący potencjał do eksploracji, jak technologia śledzenia wzroku może służyć jako fundament dla bardziej złożonych i interaktywnych mechanik rozgrywki, szczególnie w gatunku gier platformowych. W tym kontekście, można by zbadać, w jaki sposób integracja śledzenia wzroku wpływa na tradycyjne elementy gry platformowej, takie jak nawigacja, rozwiązywanie zagadek i interakcja ze środowiskiem gry.

W ramach dema technologicznego, skupiono się na tworzeniu gry platformowej, gdzie śledzenie wzroku nie jest tylko dodatkiem, ale kluczowym elementem gameplay'u. Mogłoby to obejmować takie interakcje, jak zmiana koloru i kształtu obiektów, przesuwanie i rotacja elementów środowiska, sterowane wyłącznie wzrokiem gracza. Te mechaniczne innowacje nie tylko demonstrują możliwości technologii śledzenia wzroku, ale również otwierają nowe możliwości dla projektowania gier, oferując graczom unikalne wyzwania i sposoby interakcji z grą. Taki projekt stanowi krok naprzód w eksploracji, jak śledzenie wzroku może przekształcić tradycyjne doświadczenie gry wideo, przynosząc nowe wymiary interakcji i immersji w cyfrowych światach.

# Rozdział 3

## Wymagania i narzędzia

### 3.1 Wymagania funkcjonalne

W przypadku projektu związanego z grą platformową wykorzystującą technologię śledzenia wzroku, możemy podzielić wymagania funkcjonalne na kilka kluczowych kategorii:

- Interakcje z wykorzystaniem śledzenia wzroku (sterowane za pomocą Tobii SDK):
  - Gra powinna umożliwiać zmianę koloru wybranych obiektów w środowisku gry poprzez skierowanie na nie wzroku.
  - Gra powinna umożliwiać zmianę kształtu wybranych obiektów w środowisku gry za pomocą śledzenia wzroku.
  - Gracz powinien być w stanie przesuwać i obracać obiekty w grze, wykorzystując tylko ruchy oczu.
- Podstawowe mechaniki gier platformowych (sterowane klawiaturą i myszą):
  - Ruch postaci: Poruszanie się postaci w lewo, prawo, skakanie i inne typowe ruchy w grach platformowych, kontrolowane tradycyjnie za pomocą klawiatury.
  - Poruszanie kamerą: Kontrola kamery, pozwalająca graczowi na zmianę perspektywy lub przeglądanie środowiska gry, sterowana za pomocą myszy.
  - Akceptacja i interakcja z obiektami: Ta funkcja umożliwia graczowi świadome wykonywanie działań, takich jak aktywacja mechanizmów czy zbieranie przedmiotów w grze, za pomocą klawiatury i myszy. Taki sposób sterowania został wybrany jako jeden z zalecanych bazując na artykule "An Evaluation of an Eye Tracker as a Device for Computer Input"[10] aby zapobiec niezamierzonym interakcjom, które mogą wynikać z nieświadomego skierowania wzroku na dany element - zjawisku znanemu jako 'problem dotyku Midasa' poruszanym szerzej w artykule "King Midas and the Golden Gaze"[2].

Dzięki temu gracze mają pełną kontrolę nad swoimi działaniami, minimalizując ryzyko przypadkowego aktywowania funkcji czy obiektów

- Interfejs użytkownika:
  - Z uwagi na charakter demonstracyjny gry, rozbudowany interfejs użytkownika nie jest wymagany. Podstawowe instrukcje dotyczące rozgrywki i sterowania mogą być prezentowane przez osobę nadzorującą grę.
  - Gra powinna zapewniać podstawowe wskazówki lub instrukcje na ekranie dotyczące używania technologii śledzenia wzroku.
- Zgodność sprzętowa i oprogramowanie:
  - Gra musi być kompatybilna wyłącznie z urządzeniami do śledzenia wzroku, które wspierają Tobii Unity SDK.
  - Wymagania systemowe gry, w tym obsługiwane systemy operacyjne i specyfikacje sprzętowe, powinny być jasno określone, z naciskiem na kompatybilność z Tobii Unity SDK.
- Integracja z Unity:
  - Użycie najnowszej stabilnej wersji silnika Unity.
  - Stosowanie praktyk programowania w Unity.
  - Wykorzystanie narzędzi Unity do zarządzania scenami, assetami i animacjami
- Testowanie i debugowanie w Unity:
  - Regularne testowanie i debugowanie podczas rozwoju gry.
  - Przeprowadzenie testów użyteczności, zwłaszcza dla interakcji śledzenia wzroku.

## 3.2 Wymagania niefunkcjonalne

- Wydajność:
  - Gra powinna działać płynnie na różnych konfiguracjach sprzętowych, w tym na laptopach z zintegrowanymi kartami graficznymi.
  - System śledzenia wzroku Tobii powinien zapewniać szybką i bezopóźnioną reakcję.
- Skalowalność:

- Gra powinna być dostosowana do różnych rozdzielczości ekranu, w tym 1080p, 2.5k i 4k.
  - Możliwy dalszy rozwój gry skupi się na optymalizacji zachowań obiektów i parametryzacji funkcji, a niekoniecznie na dodawaniu nowych poziomów.
- Użyteczność:
    - Gra posiada minimalny interfejs użytkownika, z naciskiem na intuicyjność i prostotę.
    - Podstawowe mechaniki sterowania (chodzenie na klawiaturze WASD, rozglądzanie się myszką) są proste i dostępne dla większości graczy.
  - Niezawodność:
    - Stabilność i niezawodność gry są zapewnione przez Unity oraz przez konstrukcję poziomów, które uniemożliwiają utknięcie gracza.
    - Testowanie gry jest wykonywane manualnie, zarówno pod kątem poszczególnych funkcjonalności, jak i całości.
  - Kompatybilność:
    - Gra jest kompatybilna z wszystkimi systemami operacyjnymi wspierającymi Unity i Tobii Eyetracker.
    - Dalsze plany mogą obejmować rozwój pod kątem współpracy z innymi urządzeniami śledzącymi wzrok, ale nie jest to priorytetem w obecnym projekcie.
  - Bezpieczeństwo:
    - Gra nie przechowuje żadnych danych użytkowników, minimalizując kwestie bezpieczeństwa danych.

### 3.3 Opis narzędzi

Kryteria wyboru narzędzi niezbędnych do realizacji tego projektu opierały się przede wszystkim na kwestii efektywności czasowej i łatwości adaptacji. W kontekście ograniczonego czasu na naukę i implementację, decyzja o wykorzystaniu konkretnych narzędzi, jak na przykład ProBuilder w miejsce bardziej zaawansowanych opcji takich jak Blender, była podyktowana ich prostotą użycia i szybkością osiągania zamierzonych rezultatów. Chociaż te wybory mogły odbiegać od optymalnych pod względem technicznym rozwiązań, zapewniały one odpowiednią równowagę pomiędzy jakością a efektywnością czasową, co było kluczowe w kontekście ograniczeń czasowych projektu.

### **3.3.1 Unity**

Wybór Unity jako silnika do projektu był podyktowany nie tylko jego łatwością użycia i dostępnością zasobów edukacyjnych, ale również chęcią głębszego zapoznania się z tym narzędziem oraz porównania możliwości, jakie oferuje dla pojedynczego twórcy, w kontekście innych silników, takich jak Unreal Engine. Unity stanowiło bardziej przystępne środowisko, umożliwiając efektywną pracę nad projektem, a jednocześnie dając możliwość eksploracji jego funkcjonalności i elastyczności w tworzeniu gier przez pojedynczego dewelopera.

### **3.3.2 Unity ProBuilder**

Wybór Unity ProBuilder był motywowany jego szybkością i prostotą w użyciu, co pozwoliło na natychmiastowe rozpoczęcie prac projektowych bez konieczności nauki skomplikowanego oprogramowania do modelowania 3D, jakim jest Blender. Narzędzie to umożliwiło projektowanie bezpośrednio w scenie Unity, co znacznie ułatwiło dopasowanie skali i organizację elementów gry, a jednocześnie zapewniało wystarczające możliwości do realizacji założeń projektowych.

### **3.3.3 Visual Studio Code**

Visual Studio Code zostało wybrane ze względu na osobiste preferencje oraz komfort pracy, który oferuje to środowisko programistyczne. Pomimo pewnych ograniczeń w zakresie Intellicode, narzędzie to zapewniło odpowiednią funkcjonalność i integrację z Unity, co było ważne dla sprawnej realizacji projektu.

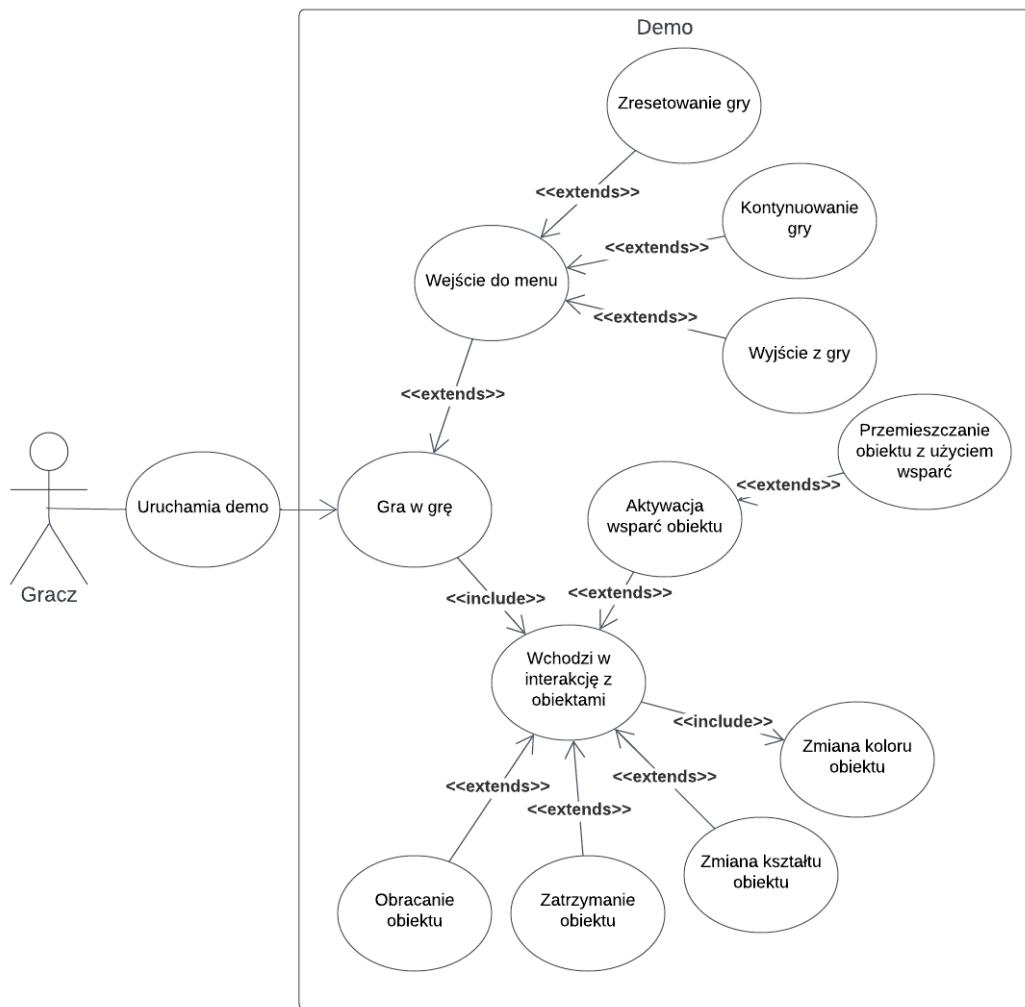
### **3.3.4 Tobii eyetracker 5**

Wybór Tobii Eye Tracker 5 jako narzędzia do śledzenia wzroku był kierowany jego dostępnością, łatwością rozpoczęcia pracy i wsparciem biblioteki SDK. Urządzenie to okazało się najlepszym wyborem dla początkujących w technologii śledzenia wzroku, choć w przyszłości planowane jest rozważenie użycia bardziej zaawansowanych urządzeń śledzących wzrok z otwartym kodem.

### **3.3.5 Tobii Unity SDK 5.0.0.3**

Użycie Tobii Unity SDK 5.0.0.3 było ścisłe powiązane z wyborem Tobii Eye Tracker 5. SDK to zapewniło niezbędne narzędzia do efektywnej integracji technologii śledzenia wzroku z projektem w Unity, co było kluczowe dla wdrożenia tej technologii w grze.

### 3.4 Przypadki użycia - diagram UML



Rysunek 3.1: Podpis rysunku po rysunkiem.



# Rozdział 4

## Specyfikacja zewnętrzna

### 4.1 Wymagania sprzętowe i programowe

#### 4.1.1 Sprzęt

Procesor X64 z wsparciem dla zestawu instrukcji SSE2 oraz karty graficzne DX10, DX11, i DX12 (dla Windows). Przynajmniej 2GB pamięci ram i 100MB wolnego miejsca na Dysku. Tobii eyetracker 5.

#### 4.1.2 Oprogramowanie

Windows 7 (SP1+), Windows 10, Windows 11 (64-bitowe wersje), bez potrzeby dodatkowych bibliotek. Oprogramowanie Tobii Experience.

### 4.2 Sposób instalacji

- Pobrać archiwum z demo.
- Rozpakować archiwum z demo.
- Pobrać i zainstalować Tobii Experience.
- Przed pierwszą grą skalibrować eyetracker w aplikacji Tobii Experience.

### 4.3 Sposób obsługi

- Uruchomienie gry przez plik .exe.
- Sterowanie postacią: klawisze WASD.
- Poruszanie kamerą: mysz.

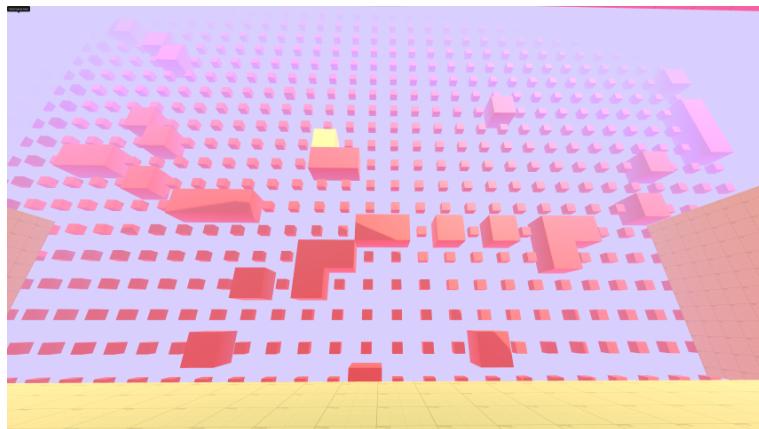
- Interakcje z obiektami: śledzenie wzroku, przesuwanie obiektów (spostrzeganie + LPM), wejście do pojazdu (spostrzeganie + LPM), wyjście z interakcji (klawisz Q), obsługa menu (ESC, mysz).

## 4.4 Scenariusze korzystania z systemu

W tej części zaprezentowano praktyczne przykłady, które pokazują, jak działają różne poziomy i mechaniki gry.

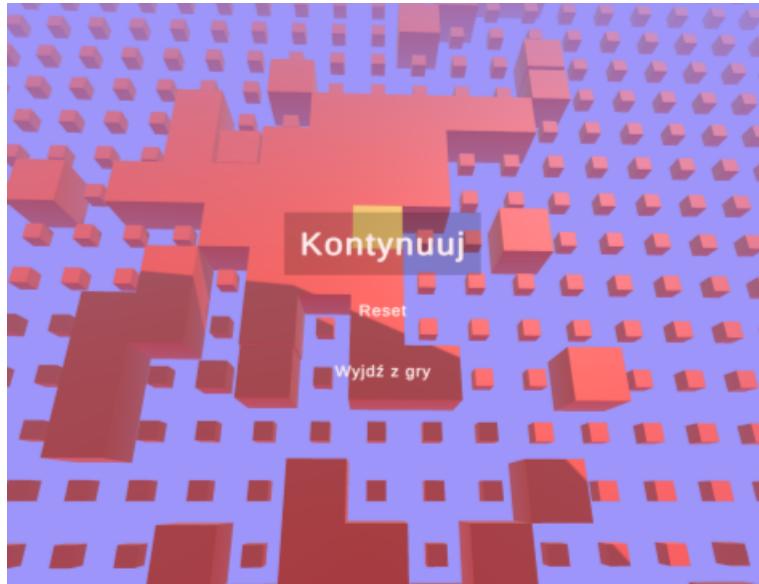
### 4.4.1 Obszar startowy

W obszarze startowym gracze mają możliwość wypróbowania jak działają interakcje wzrokowe, ucząc się jednocześnie, w jaki sposób system śledzenia wzroku odpowiada na ich działania. Czerwone kwadraty zmieniają kolor na żółty, gdy są gracz na nie spojrzy, a następnie powiększają się, co demonstruje reakcję systemu na interakcje wzrokowe użytkownika.



Rysunek 4.1: Siatka z czerwonymi kwadratami, z wyraźnie wyróżnionym żółtym kwadratem, stanowiącym punkt koncentracji wzroku.

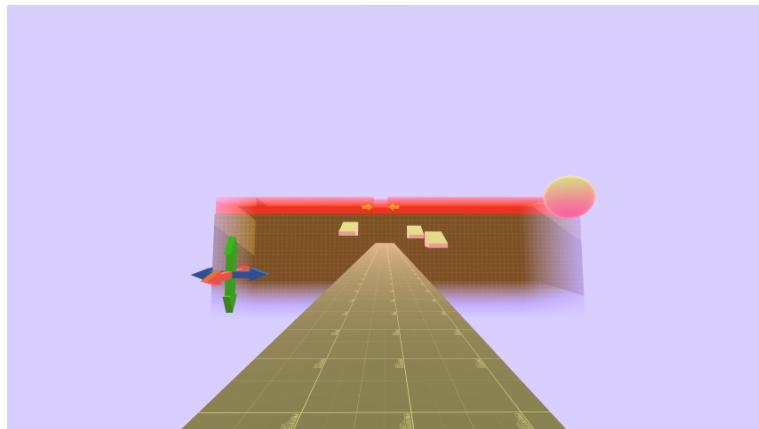
Dostęp do menu gry jest możliwy w dowolnym momencie poprzez naciśnięcie klawisza 'Esc'. W menu użytkownik może z łatwością kontynuować grę, zrestartować aktualny poziom lub zakończyć demo, korzystając z myszki do nawigacji.



Rysunek 4.2: Menu gry z widocznymi opcjami "Kontynuuj", "Reset" oraz "Wyjdź z gry"

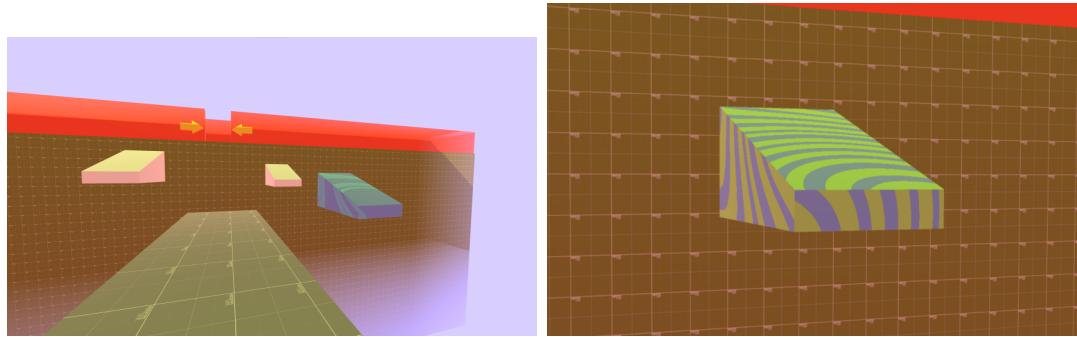
#### 4.4.2 Pierwszy poziom

Po opuszczeniu obszaru startowego gracz przechodzi do pierwszego poziomu składającego się z obiektu osi, kładki, 3 ramp i labiryntu oraz dodatkowej piłeczki która po spojrzeniu na nią odbija się parę razy po swoim niewidocznym pojemniku.



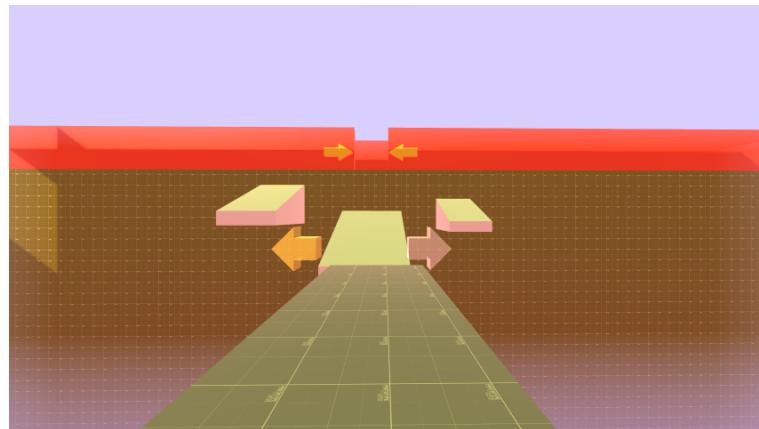
Rysunek 4.3: Widok pierwszego poziomu

Aby przejść dalej gracz musi opanować podstawowe przesuwanie obiektów z użyciem śledzenia wzroku. Gdy gracz skupi wzrok na danej kładce, jej kolor zmienia się na pulsujący wzór dwukolorowy. Ta zmiana sygnalizuje, w jakiej płaszczyźnie można przesuwać obiekt. Na przykład, obiekt po lewej stronie rysunku 4.4 może być przesuwany w prawo i lewo. Z kolei obiekt po prawej stronie pozwala na ruch w płaszczyźnie prawo-lewo oraz góra-dół.



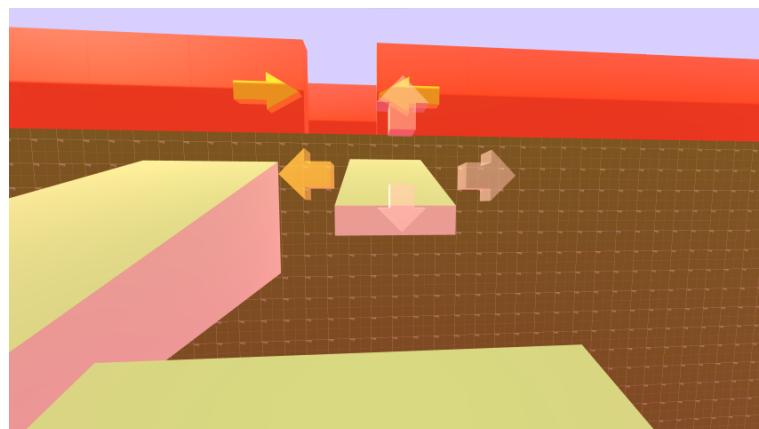
Rysunek 4.4: Obiekty z znacznikami skupienia wzroku

Aby przystąpić do przesuwania wybranego obiektu po skupieniu wzroku na nim klikamy lewy przycisk myszy w celu jego aktywacji, pojawią się wtedy obiekty wspierające (wsparcia) w postaci strzałek. Grot strzałki symbolizuje w której stronę będzie przemieszczany obiekt po spojrzeniu na niego.



Rysunek 4.5: Obiekt aktywny z widocznymi wsparciami podczas przesuwania w lewo

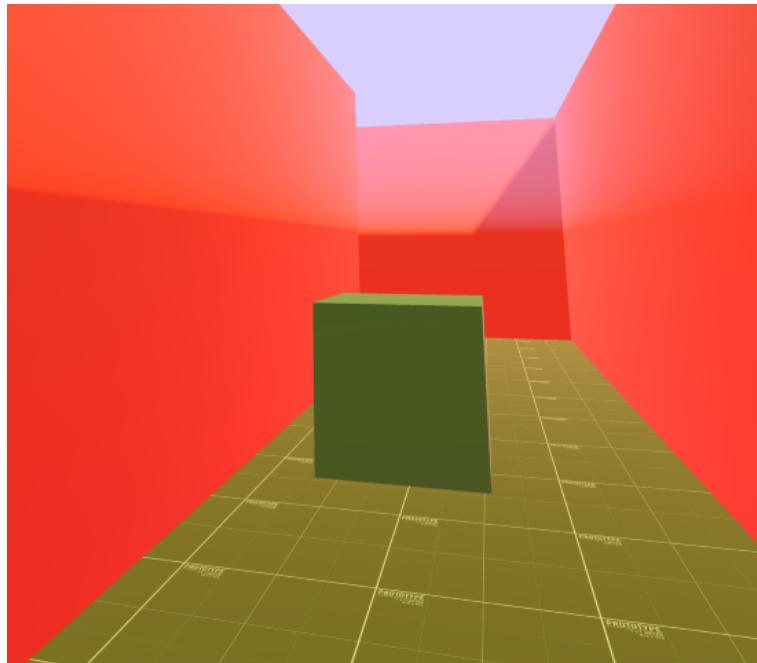
W przypadku obiektów poruszających się w więcej niż jednej płaszczyźnie pojawia się więcej obiektów wsparcia.



Rysunek 4.6: Demonstracja obiektu przesuwanego w dwóch wymiarach

Aby dezaktywować obiekt gracz musi spojrzeć się na niego oraz kliknąć lewy przycisk myszy, dodatkowo aby dezaktywować wszystkie aktywne obiekty można użyć klawisza 'Q'.

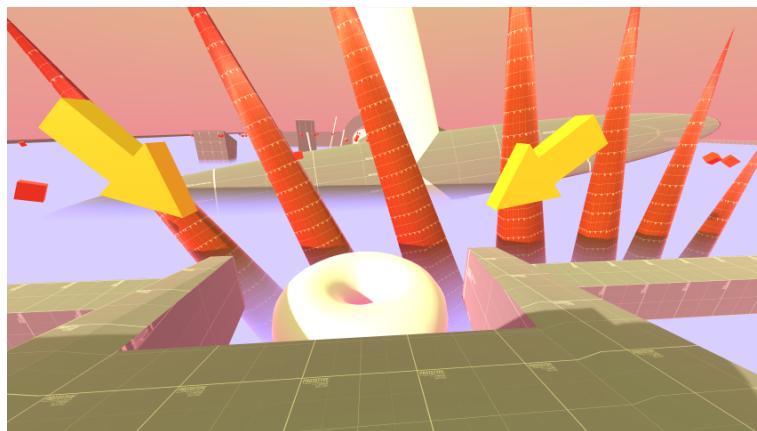
Poziom ten kończy labirynt z przewodnikiem który po spojrzeniu na niego ucieka w kierunku wyjścia z labiryntu zostawiając po sobie ślad ułatwiający nawigację.



Rysunek 4.7: "Uciekający" obiekt reagujący na wzrok gracza

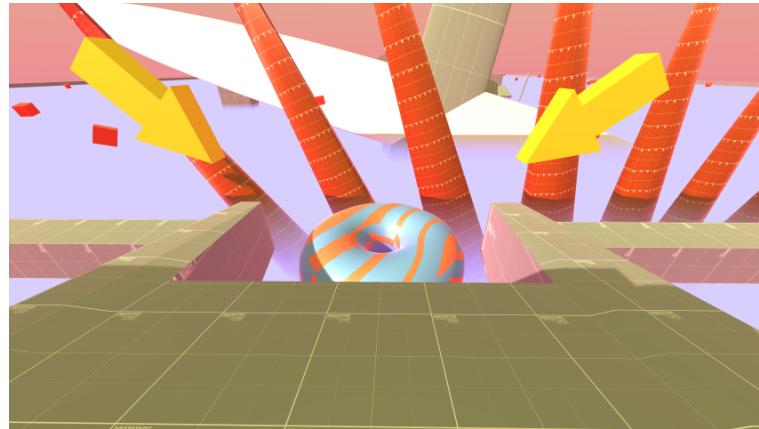
#### 4.4.3 Drugi poziom

Po opuszczeniu labiryntu gracz trafia na drugi poziom którego głównym elementem jest latające stateczki (biały obiekt wskazywany przez żółte strzałki) za pomocą sterowania wzrokiem. Gracz podczas lotu musi omijać przeszkody oraz czasami torować sobie drogę z wykorzystaniem śledzenia wzroku.



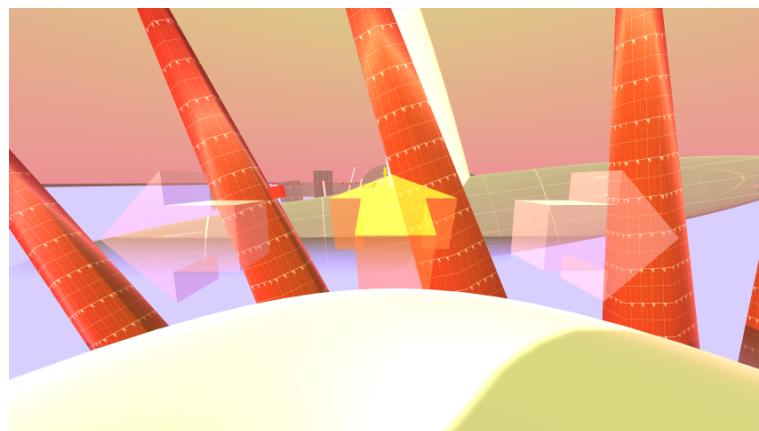
Rysunek 4.8: Widok drugiego poziomu, na pierwszym planie wskazywany przez żółte strzałki stateczek

Interakcja z stateczkiem działa na podobnej zasadzie do przesuwania obiektów: tzn. patrzymy na obiekt, klikamy lewy przycisk myszy. Różnica polega na tym, że interakcja ta powoduje umieszczenie gracza w środku statku.

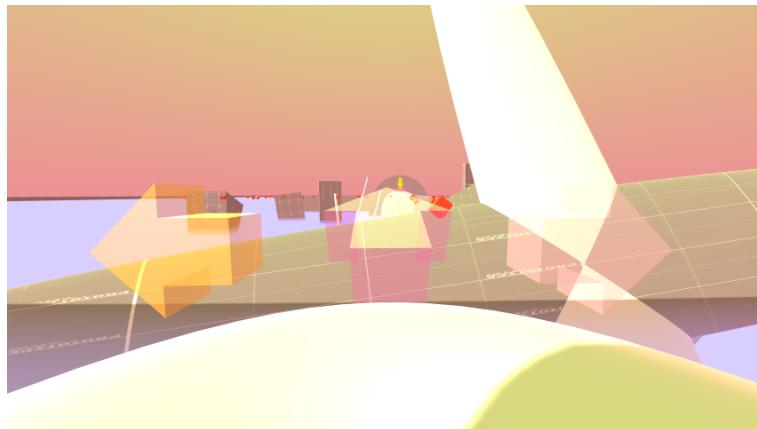


Rysunek 4.9: Stateczek po skupieniu na nim wzroku

Po wskoczeniu do stateczku gracz widzi 3 wsparcia pozwalające kolejno od lewej na jazdę w lewo, prosto oraz w prawo widoczne na rysunku 4.10. Na wypadek, gdyby gracz gdzieś utknął po obróceniu kamery do tyłu znajduje się jeszcze jedno wsparcie służące cofaniu.

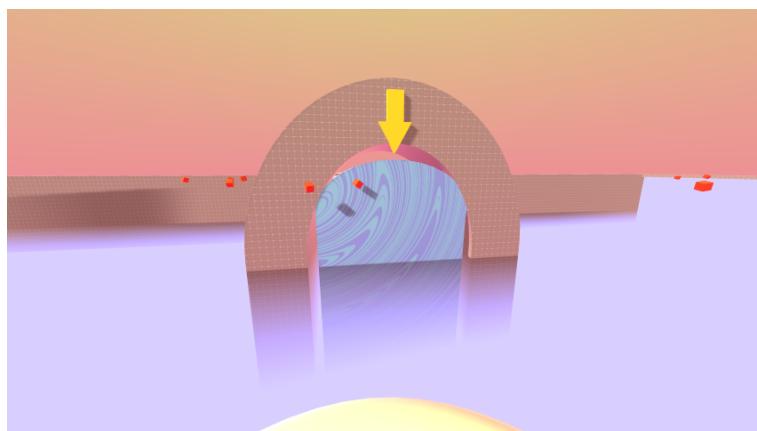


Rysunek 4.10: Wsparcia pozwalające na jazdę w lewo, prosto oraz w prawo



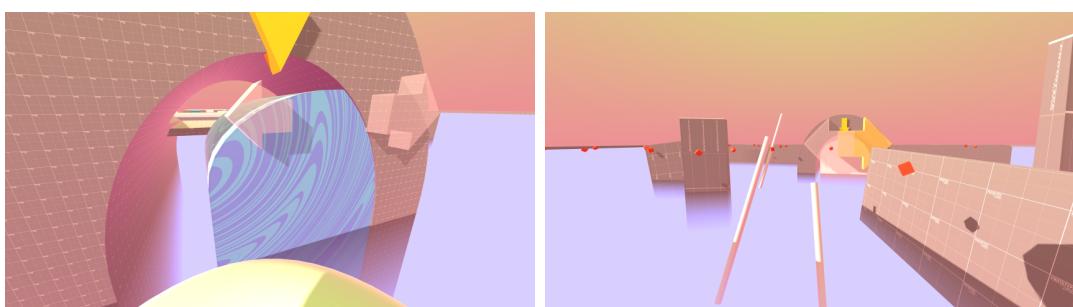
Rysunek 4.11: Przykład używania wsparcia do jazdy w lewo

Na końcu poziomu czeka gracza ostatnia przeszkoda w postaci przesuwanej bramy.



Rysunek 4.12: Przesuwana brama z znacznikiem skupienia wzroku.

Brama przesuwana jest na tej samej zasadzie jak rampy na poziomie pierwszym, zróżnicowanie odległości z których można wejść z nią w interakcję spowodowało, że niekiedy ciężko było wejść w interakcje ze wsparciami. Aby zniwelować ten problem użyto tutaj skrypt, który powoduje zmianę skali wsparć w zależności od ich dystansu względem gracza, co prezentuje rysunek 4.13.



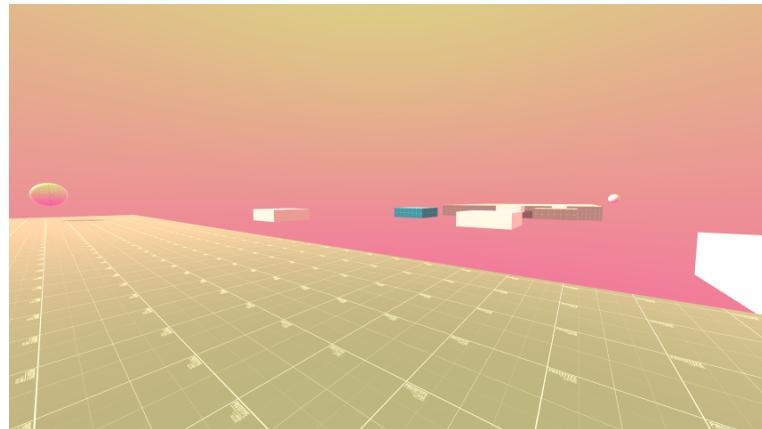
Rysunek 4.13: Wielkość wsparć w zależności od dystansu do gracza

Po przelecienniu przez bramę, gracz dostrzeże platformę. Może opuścić stateczek,

używając klawisza 'Q', aby kontynuować grę na 3 poziomie.

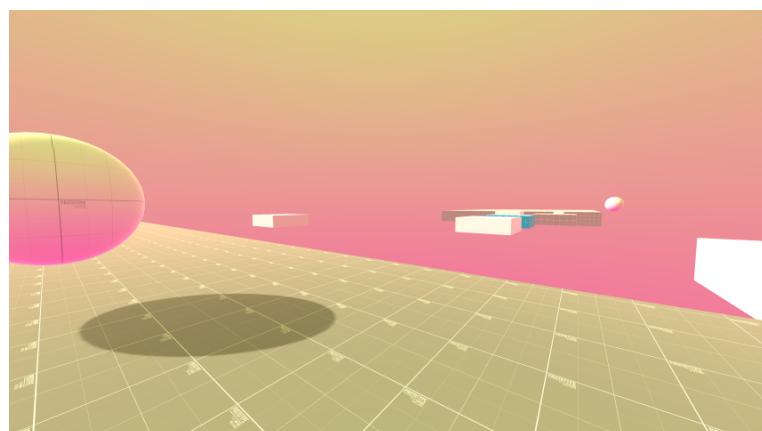
#### 4.4.4 Trzeci poziom

Po opuszczeniu stateczku przed graczem znajduje się pokonanie przepaści z platformami, białe platformy można poruszać, niebieska platforma porusza się samoczynnie.



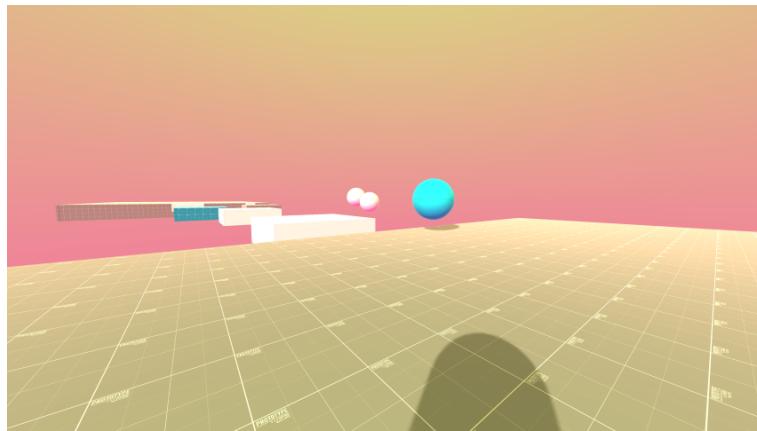
Rysunek 4.14: Widok trzeciego poziomu

Urozmaiceniem rozgrywki są tutaj ścigające gracza kule, które próbują go zepchnąć z platform utrudniając grę.



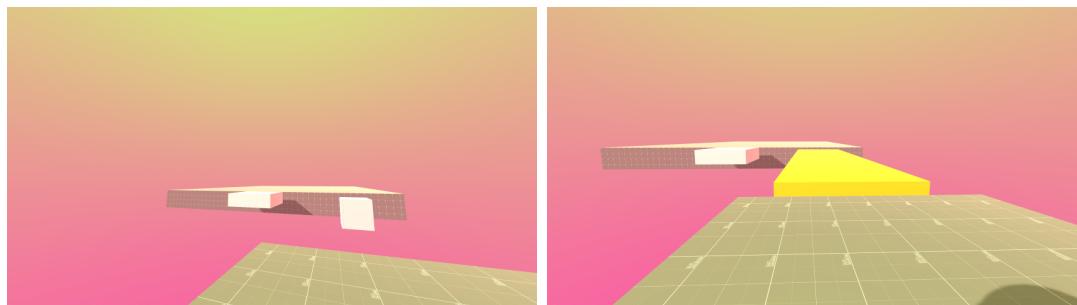
Rysunek 4.15: Zbliżająca się kula

Kule zaczynają się zbliżać do gracza w momencie gdy nie zwraca na nie uwagi i przestają momentalnie, gdy skupi na nich swój wzrok.



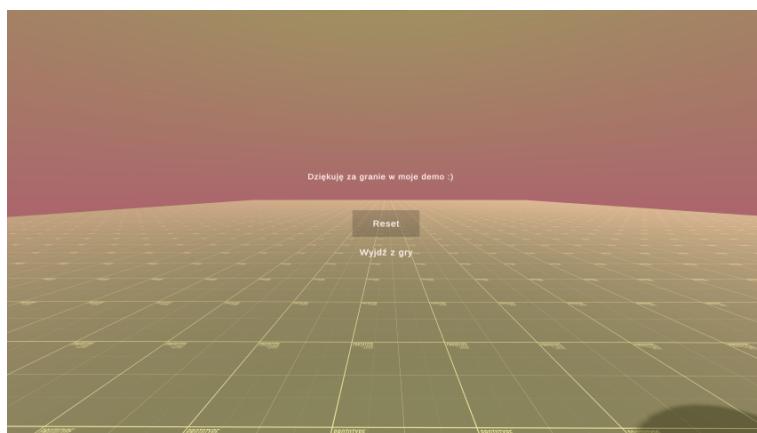
Rysunek 4.16: Kula podczas skupienia na niej wzroku

Ostatnią przeszkodą jest dynamiczna platforma która transformuje się z sześcianu po spojrzeniu na niego.



Rysunek 4.17: Dynamiczna platforma przed i w trakcie transformacji.

Po przeskoczeniu na szare podłożę gracz kończy demo, wszystkie kule znikają i pojawia się ekran końcowy.



Rysunek 4.18: Ekran końcowy



# Rozdział 5

## Specyfikacja wewnętrzna

W tej części pracy skupiono się na specyfikacji wewnętrznej, która zagłębia się w techniczne szczegóły implementacji systemu w Unity.

### 5.1 Przedstawienie idei i architektura systemu

Podstawą projektu jest silnik Unity, który posłużył jako platforma do zbudowania i zarządzania różnymi elementami gry. System został zaprojektowany z myślą o elastyczności i łatwości rozbudowy, co osiągnięto poprzez zastosowanie architektury opartej na scenach, obiektach i komponentach.

#### 5.1.1 Struktura systemu w unity

- Sceny w Unity:

Każdy element interakcji wzrokowej w grze został umieszczony w odpowiedniej scenie w Unity. Sceny te działają jak odrębne etapy lub poziomy gry, gdzie każda z nich ma własne unikalne zadanie i zestaw interakcji.

- Obiekty gry:

W ramach scen, wszystkie elementy interaktywne są reprezentowane jako obiekty gry (GameObject). Obiekty te mogą być prostymi formami, jak sześciany lub sfery, albo bardziej złożonymi modelami 3D, w zależności od potrzeb scenariusza gry.

- Komponenty i skrypty:

Do obiektów gry dodajemy komponenty, które definiują ich zachowanie. Te komponenty to skrypty C#, takie jak ColorChangeOnGaze.cs czy MoveOnWayPointsOnGaze.cs, które odpowiadają za różnorodne interakcje wzrokowe. Skrypty te manipulują właściwościami obiektów, takimi jak kolor,

pozycja, czy rozmiar, w odpowiedzi na spojrzenie gracza, wykorzystując technologię śledzenia wzroku Tobii.

### **5.1.2 Zalety architektury Unity:**

- Modularność: Dzięki wykorzystaniu komponentów, system jest modularny. Można łatwo dodawać, usuwać lub modyfikować funkcje poszczególnych obiektów gry bez wpływu na inne elementy.
- Elastyczność: Architektura oparta na scenach i obiektach pozwala na łatwe tworzenie różnorodnych scenariuszy gry, z różnymi zestawami interakcji wzrokowych.
- Łatwość rozwoju i debugowania: Unity zapewnia intuicyjne środowisko do tworzenia i testowania gry, co ułatwia debugowanie i rozwój projektu.

## **5.2 Komponenty, moduły, biblioteki, przegląd ważniejszych klas**

### **5.2.1 Tobii Unity SDK**

W projekcie zastosowano Tobii Unity SDK, co stanowi kluczowy element w obsłudze śledzenia wzroku. SDK to dostarcza zestaw narzędzi i API niezbędnych do integracji możliwości śledzenia wzroku z grą.

### **5.2.2 Unity ProBuilder**

Do projektowania i modelowania obiektów środowiska gry użyto ProBuilder, narzędzia Unity, które umożliwia łatwe tworzenie, edycję i teksturizację modeli 3D bezpośrednio w Unity.

### **5.2.3 TextMeshPro**

Do tworzenia zaawansowanych tekstów w menu i interfejsie użytkownika wykorzystano TextMeshPro. Jest to potężne narzędzie do renderowania tekstu w Unity, oferujące większą elastyczność i jakość niż standardowe komponenty tekstu.

### **5.2.4 Shader Graph**

Do tworzenia materiałów do znaczników skupienia wzroku oraz materiału skyboxu wykorzystano Shader Graph. Dzięki temu narzędziu można wizualnie tworzyć

zaawansowane shadery, co pozwala na lepsze efekty wizualne i większą kontrolę nad wyglądem gry.

### 5.2.5 Główne klasy interakcji z obiektami

W tej sekcji przedstawiamy klasy, które są kluczowe dla interakcji z obiektami w grze, opierając się na technologii śledzenia wzroku. Te klasy umożliwiają bezpośrednią manipulację obiektami w środowisku gry, reagując na działania gracza.

- **MoveOnGaze:** Skrypt umożliwiający przesuwanie obiektów w płaszczyznach zdefiniowanych podczas projektowania poziomu, aktywowany przez spojrzenie użytkownika.
- **MoveOnWayPointsOnGaze:** Zarządza ruchem obiektów do określonych punktów na scenie, reagując na spojrzenie użytkownika.
- **MoveWhenNotLookedAt:** Inicjuje ruch obiektów, gdy użytkownik na nie nie patrzy, utrudniając rozgrywkę.
- **SpinOnGaze:** Obraca obiekty w reakcji na spojrzenie użytkownika, pozwala na większą interaktywność scen.
- **ResizeOnGaze:** Zmienia rozmiar obiektów w odpowiedzi na spojrzenie użytkownika, pozwala na tworzenie platform z sześcianów oraz animacje niektórych obiektów.

### 5.2.6 Klasy pomocnicze do interakcji

W tej części skupiamy się na klasach pomocniczych, które wspierają główne interakcje w grze poprzez dodawanie efektów wizualnych i funkcjonalności poprawiających doświadczenie użytkownika.

- **ScaleWithDistance:** Automatycznie skaluje obiekty w zależności od ich odległości od kamery, ułatwiając interakcje na różnych dystansach.
- **ColorChangeOnGazeChildren:** Zmienia kolor obiektów i ich potomków, gdy gracz na nie spojrzy, sygnalizując interakcję wzrokową.
- **ShowSymbolOnGaze:** Uruchamia symbole na obiektach, kiedy na nie spojrzymy. Podobnie jak klasa ColorChangeOnGazeChildren, ale zamiast zmieniać kolor, zmienia specyficzne właściwości materiału za pomocą zmiennych boolowskich.

## 5.3 Zastosowane wzorce projektowe

### 5.3.1 Wzorzec projektowy singleton

Wzorzec Singleton został zastosowany w klasie UIControl, której głównym celem jest zarządzanie interfejsem użytkownika w grze. Singleton jest użyteczny, gdy potrzebujemy globalnego dostępu do pewnych zasobów lub funkcjonalności w całej aplikacji, a jednocześnie chcemy zapewnić, że istnieje tylko jedna instancja danej klasy.

W klasie UIControl, Singleton pozwala na łatwe zarządzanie stanem gry, takim jak kontrola nad pauzowaniem i wznowieniem rozgrywki, bez konieczności przekazywania referencji do tej klasy pomiędzy różnymi komponentami gry. Dzięki temu wzorcowi, stan UI jest konsekwentnie i centralnie zarządzany, co upraszcza strukturę kodu i ułatwia utrzymanie oraz rozwój aplikacji.

Implementacja Singletona w UIControl jest przykładem, jak wzorzec ten może być wykorzystany do efektywnego zarządzania globalnym stanem gry w Unity.

### 5.3.2 Wzorzec projektowy komponent

Wzorzec komponent jest kluczowym elementem projektowania gier w Unity, pozwalając na tworzenie elastycznych i modułowych systemów interakcji [7]. Zastosowanie tego wzorca w klasach MoveOnGaze i ShowSymbolOnGaze demonstruje jego moc w kreowaniu interaktywnego środowiska gry. Dzięki podejściu opartemu na komponentach, każda z tych klas może być rozwijana niezależnie, oferując różnorodne funkcjonalności, które mogą być łatwo integrowane z dowolnym obiektem w grze.

- **MoveOnGaze:** Klasa MoveOnGaze wykorzystuje wzorzec komponent do zapewnienia obiektom w grze zdolności do reagowania na spojrzenie gracza i przemieszczania się w przestrzeni. Jako oddzielny moduł, ten komponent może być rozwijany i udoskonalany niezależnie od innych aspektów gry. Możliwość dołączania MoveOnGaze do różnych obiektów bez konieczności modyfikacji ich podstawowej struktury czy funkcjonalności jest przykładem elastyczności, jaką oferuje wzorzec komponent.
- **ShowSymbolOnGaze:** ShowSymbolOnGaze jako komponent służy do wzbogacania obiektów o możliwość dynamicznego wyświetlania wizualnych symboli w odpowiedzi na interakcje wzrokowe użytkownika. Niezależny rozwój tej funkcjonalności ułatwia implementację złożonych efektów wizualnych, które mogą być dowolnie dodawane do obiektów w grze. Takie podejście pozwala na tworzenie bardziej interaktywnych i angażujących scenariuszy gry, jednocześnie zachowując klarowność i porządek w kodzie.

Wykorzystanie wzorca komponent w MoveOnGaze i ShowSymbolOnGaze pozwala na niezależny rozwój każdej z tych funkcjonalności, przy jednoczesnej możliwości ich integracji z dowolnym obiektem w grze. Daje to twórcom gier ogromną elastyczność w projektowaniu interakcji oraz umożliwia łatwe wprowadzanie zmian i ulepszeń. W efekcie, obiekt w grze wyposażony zarówno w MoveOnGaze, jak i ShowSymbolOnGaze staje się bardziej dynamiczny i interaktywny, oferując graczy bogatsze doświadczenie.

## 5.4 Przegląd wybranych fragmentów kodu

### 5.4.1 Wykrywanie wzroku na podstawie klasy ColorChangeOnGazeChildren

W początkowej fazie tworzenia interakcji z użyciem elementu GazeAware, proces ten wydawał się stosunkowo prosty. Jednakże, złożoność projektu wzrosła znacząco przy projektowaniu bardziej skomplikowanych struktur w demie, które składały się z wielu obiektów. Napotkano trudności w efektywnym wykrywaniu skupienia wzroku użytkownika na obiektach potomnych, co było kluczowe dla prawidłowego działania interakcji wzrokowych. Problem ten wynikał z ograniczeń komponentu GazeAware, który bazuje na wykrywaniu wzroku poprzez komponent mesh collider obiektu. W sytuacji, gdy mesh collider nie istnieje lub znajduje się w potomnych obiektach, pożądana interakcja nie była realizowana.

Aby rozwiązać ten problem, opracowano skrypt (patrz Rysunek 5.1), który zapewniał, że zmiana koloru dotyczy całego obiektu wraz z jego potomkami, niezależnie od umiejscowienia mesh collidera. Dzięki temu podejściu, cała struktura obiektu - zarówno główna, jak i jej składniki - reaguje na spojrzenie gracza, co znacząco poprawiło interaktywność i funkcjonalność elementów gry.

Metody Start() oraz Update() w skrypcie są wywoływane przez Unity odpowiednio przy inicjalizacji obiektu oraz przy każdej klatce gry.

### 5.4.2 Poruszanie obiektami z użyciem wzroku na przykładzie klasy MoveOnGaze

W projekcie, klasa MoveOnGaze jest jedną z bardziej zaawansowanych, odróżniając się zastosowaniem unikalnych metod interakcji wzrokowej. Charakteryzuje się ona wykorzystaniem dodatkowych obiektów pomocniczych, które rozszerzają funkcjonalność sterowania obiektem. Klasa MoveOnGaze, początkowo zaprojektowana do przesuwania obiektów za pomocą spojrzenia użytkownika, ewoluowała znacznie w trakcie rozwoju projektu. Jej funkcjonalność została rozszerzona o możliwość 'wsiadania' do obiektu i poruszania się z nim, co znacząco wzbogaciło dynamikę interakcji w grze. Pierwotna

```
1 public class ColorChangeOnGazeChildren : MonoBehaviour
2 {
3     private GazeAware _gazeAware;
4     public Color colorOnGaze = Color.yellow;
5     // Kolor pojawiający się po spojrzeniu
6
7     private Renderer[] _childRenderers;
8     // Lista wszystkich modułów renderujących obiektu i jego dzieci
9     private Color[] _originalColors;
10    // Lista oryginalnych kolorów obiektu i jego dzieci
11
12    void Start()
13    {
14        _gazeAware = GetComponent<GazeAware>();
15        _childRenderers = GetComponentsInChildren<Renderer>();
16
17        _originalColors = new Color[_childRenderers.Length];
18        for (int i = 0; i < _childRenderers.Length; i++)
19        {
20            _originalColors[i] = _childRenderers[i].material.color;
21            // Zapisanie oryginalnych kolorów
22        }
23    }
24
25    void Update()
26    {
27        if (_gazeAware.HasGazeFocus)
28        {
29            foreach (var renderer in _childRenderers)
30            {
31                renderer.material.color = colorOnGaze;
32                // Zmiana na zadany kolor jeśli wykryjemy wzrok na obiekcie
33            }
34        }
35        else
36        {
37            for (int i = 0; i < _childRenderers.Length; i++)
38            {
39                _childRenderers[i].material.color = _originalColors[i];
40            // Powrót do oryginalnego koloru w razie braku spoglądania na obiekt
41            }
42        }
43    }
44 }
```

---

Rysunek 5.1: Klasa z użyciem wykrywania wzroku w dzieciach

```
1  public float maxMovementSpeed = 1.0f;
2  public float dampingFactor = 0.95f;
3
4  public GameObject supportXNegative;
5  public GameObject supportXPositive;
6  public GameObject supportYPositive;
7  public GameObject supportYNegative;
8  public GameObject supportZPositive;
9  public GameObject supportZNegative;
10 // Obiekty u których wykrycie GazeAware spowoduje
11 przesunięcie obiektu względem odpowiedniej osi
12
13 public bool moveOnXAxis = true;
14 public bool moveOnYAxis = true;
15 public bool moveOnZAxis = true;
16 public bool mounting = false;
17 // Dodatkowa zmienna dzięki której
18 wprowadzono funkcję wsiadania w obiekt
19
20 public float forceMagnitude = 0.5f;
21
22 private Rigidbody _rigidbody;
23 public GameObject mountPosition;
24
25 private GazeAware _gazeAware;
26 private bool _isActivated = false;
27 PlayerMovement playerMovement;
```

---

Rysunek 5.2: Zmienne klasy MoveOnGaze

```
1      ...
2      void MoveObject()
3      {
4          float moveStep = forceMagnitude * Time.deltaTime;
5
6          if (moveOnXAxis)
7          {
8              if (supportXNegative.GetComponent<GazeAware>().HasGazeFocus)
9              {
10                  transform.Translate(Vector3.left * moveStep);
11              }
12              if (supportXPositive.GetComponent<GazeAware>().HasGazeFocus)
13              {
14                  transform.Translate(Vector3.right * moveStep);
15              }
16          }
17      (Podobny kod do poruszania po osi y i z)
18      ...

```

---

Rysunek 5.3: Podstawowe przesuwanie

funkcja klasy, ilustrowana na Rysunku 5.3, ograniczała się do podstawowego przesuwania obiektów. W miarę postępu prac, zaimplementowano innowacyjną opcję 'wsiadania' do obiektu i przemieszczania się z nim, co ukazano na Rysunku 5.4. Ta stopniowa rozbudowa klasy MoveOnGaze podkreśla, jak w trakcie pracy nad grą można rozwijać i ulepszać początkowe koncepcje, zwiększając złożoność i elastyczność projektu. Ważnym aspektem implementacji klasy jest użycie metody FixedUpdate(), która zapewnia niezależność logiki ruchu od szybkości renderowania klatek, wykorzystując fizykę do sterowania ruchem obiektu. Wprowadzenie zmiennej mounting przyczynia się do poprawy jakości interakcji, umożliwiając utrzymanie postaci gracza w stabilnej pozycji oraz ułatwiając wyjście z pojazdu. Dzięki tym ulepszeniom, MoveOnGaze stała się kluczowym elementem projektu, znacząco wzbogacając interaktywność i doświadczenie gracza.

### 5.4.3 Proste animacje z użyciem ResizeOnGaze

Klasa ResizeOnGaze jest klasą która przeszła najwięcej przemian, na samym początku polegała na manualnym skalowaniu skali przez co była nieintuicyjna i wymagała dużo czasu aby efekty z niej użyciem były użyteczne. Zmieniło się to po pracy nad shaderami i lepszemu zaznajomieniu z funkcją Lineralnej interpolacji pomiędzy wartościami (Lerp). Pozwoliła ona na mocne uproszczenie kodu i pozwoliła na proste tworzenie kształtów (patrz Rysunek 5.5). Klasa ta posiada także prosty skrypt ułatwiający interakcję, polegający na dokonaniu skalowania nawet po spuszczeniu wzroku z obiektu w celu ulepszenia płynności rozgrywki 5.6.

```
1      ...
2      void FixedUpdate()
3      {
4          if (_isActivated && mounting)
5          {
6              AddForceToMove();
7              DampenVelocity();
8          }
9      }
10     ...
11     void AddForceToMove()
12     {
13         if (moveOnXAxis)
14         {
15             if (supportXNegative.GetComponent<GazeAware>().HasGazeFocus)
16             {
17                 _rigidbody.AddForce(Vector3.left * forceMagnitude);
18             }
19             if (supportXPositive.GetComponent<GazeAware>().HasGazeFocus)
20             {
21                 _rigidbody.AddForce(Vector3.right * forceMagnitude);
22             }
23         }
24         (Podobny kod do poruszania po osi y i z)
25     ...

```

---

Rysunek 5.4: Przesuwanie z użyciem fizyki

```
1 ...
2 void ScaleObjectTowardsTarget()
3 {
4     Vector3 newScale = transform.localScale;
5
6     newScale.x = Mathf.Lerp(newScale.x, xSettings.targetScale,
7                             xSettings.speed * Time.deltaTime);
8     newScale.y = Mathf.Lerp(newScale.y, ySettings.targetScale,
9                             ySettings.speed * Time.deltaTime);
10    newScale.z = Mathf.Lerp(newScale.z, zSettings.targetScale,
11                           zSettings.speed * Time.deltaTime);
12
13    if (newScale == new Vector3(xSettings.targetScale,
14                                ySettings.targetScale, zSettings.targetScale))
15    {
16        isScaling = false;
17    }
18
19    transform.localScale = newScale;
20 }
21 ...
```

---

Rysunek 5.5: Skalowanie z użyciem Lepr

## 5.5 Implementacja w Unity

W tej sekcji przedstawiono kluczowe aspekty procesu tworzenia gry platformowej z obsługą śledzenia wzroku w środowisku Unity. Rozdział koncentruje się na technikach programistycznych stosowanych w Unity, opisując m.in. mechanizmy zabezpieczające gracza przed wypadnięciem z poziomu, system ekranów menu, proces automatycznego rozmieszczania obiektów w edytorze, wykorzystanie Shadergraph do tworzenia materiałów, a także logikę "wsiadania do stateczka" oraz projektowania interakcji z użyciem komponentu MoveOnGaze. Całość skupia się na integracji technologii śledzenia wzroku, która jest kluczowym elementem interakcji w grze.

### 5.5.1 Mechanizmy zabezpieczające gracza i poprawiające jakość rozgrywki

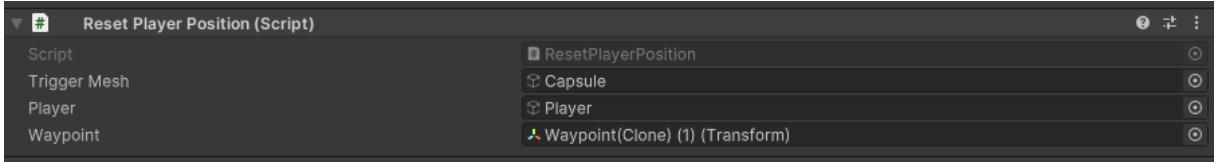
W ramach prezentowanego demo technologicznego, które jest grą platformową, istotnym elementem gwarantującym płynność rozgrywki jest wdrożenie mechanizmów zabezpieczających gracza przed niezamierzonym wypadnięciem z poziomu.

Te mechanizmy bazują na detekcji kolizji między graczem a specjalnie zaprojektowanymi obiektami. Są to głównie zielone płaszczyzny, przypisane do warstwy waypoints, niewidoczne dla kamery gracza, co ukazano na rysunku (5.7).

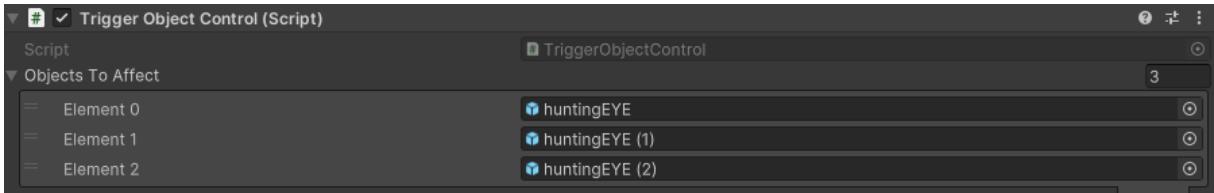
```
1 ...
2 void Update()
3 {
4     if (IsGazeFocused())
5     {
6         isScaling = true;
7         timeSinceLossOfFocus = 0.0f;
8     }
9     else
10    {
11        UpdateTimeSinceLossOfFocus();
12    }
13
14    if (isScaling)
15    {
16        ScaleObjectTowardsTarget();
17    }
18 }
19 ...
20 void UpdateTimeSinceLossOfFocus()
21 {
22     if (timeSinceLossOfFocus >= returnDelay)
23     {
24         isScaling = false;
25         ReturnToOriginalScale();
26     }
27     else
28     {
29         timeSinceLossOfFocus += Time.deltaTime;
30     }
31 }
32 ...
```

---

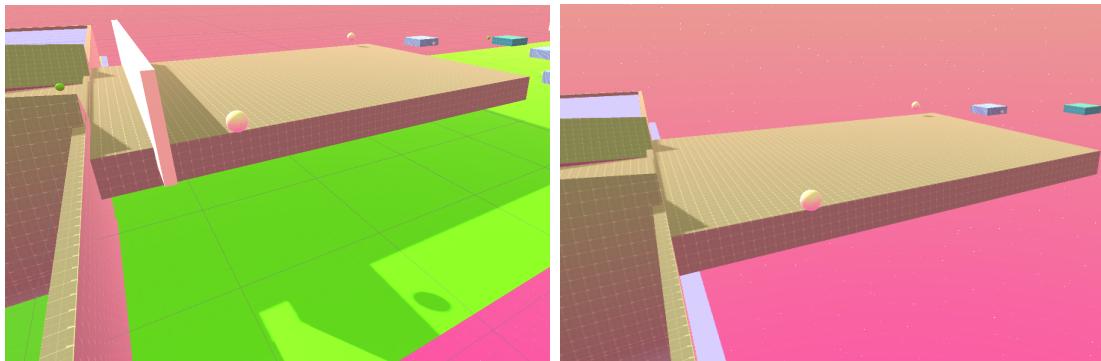
Rysunek 5.6: Metody odpowiedzialne za upłynnienie skalowania



Rysunek 5.8: Widok komponentu ResetPlayerPosition w Unity



Rysunek 5.9: Widok komponentu TriggerObjectControl



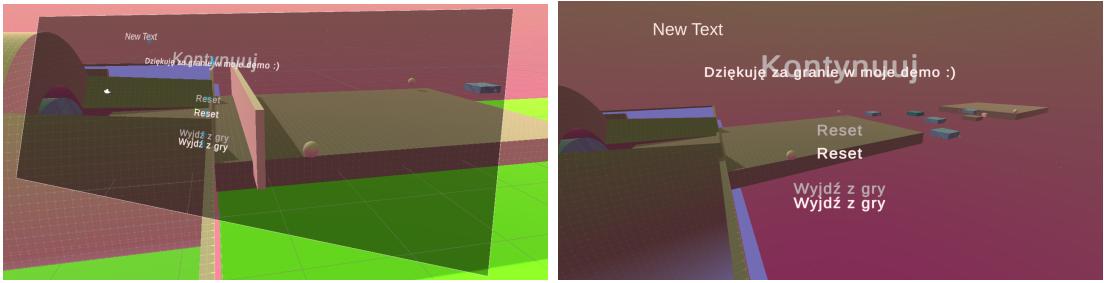
Rysunek 5.7: Porównanie widoku sceny po lewej z widokiem z kamery po prawej

Każdy z tych obiektów posiada skrypt, który po aktywacji przez kolizję z graczem, przemieszcza gracza do określonego waypointu. Skrypt ten wymaga określenia komponentu collidera aktywującego skrypt, waypointu docelowego oraz obiektu gracza, jak pokazano na rysunku (5.8).

Dodatkowo zaimplementowano mechanizm aktywujący pewne elementy gry, jak kule na trzecim poziomie, tylko wtedy, gdy gracz dotrze na ten poziom. Pozwala to na uniknięcie sytuacji, w której elementy te aktywują się zbyt wcześnie. Ten skrypt, zlokalizowany w obiekcie białej ściany (zob. rysunek 5.7), reaguje na wejście gracza, aktywując lub deaktywując określone obiekty na podstawie zmiennej boolowskiej (zob. rysunek 5.9).

### 5.5.2 System ekranów menu

System ekranów menu składa się z dwóch ekranów: ekranu głównego menu oraz ekranu końca gry. System ten jest zaimplementowany w obiekcie canvas który dzięki możliwości spięcia go z kamerą gracza pozwala na umieszczenie menu zawsze w widoku gracza i pracowanie z nim jako obiektem 2D. Sposób obsługi tego obiektu możemy zauważyc na rysunku 5.10. Oba widoki są zarządzane przez skrypt po starcie gry.



Rysunek 5.10: Porównanie obiektu menu w edytorze po lewej i efektu uzyskanemu na kamerze po prawej

### 5.5.3 Automatyzacja rozmieszczania obiektów w edytorze

W celu efektywnego i precyzyjnego rozmieszczania obiektów na poziomach, został zaimplementowany skrypt 'AutoObjectPlacement'. Skrypt ten umożliwia łatwą i szybką edycję układu obiektów w trybie edycji, bez konieczności ręcznego ustawiania każdego elementu. Dzięki wykorzystaniu skryptu w trybie edycji, możliwa jest natychmiastowa wizualizacja zmian w układzie sceny, co znacząco ułatwia proces projektowania i iteracji.

Skrypt 'AutoObjectPlacement' wykorzystuje atrybut [ExecuteInEditMode], co pozwala na jego działanie w trybie edycji w Unity. Pozwala to na dynamiczne rozmieszczenie i modyfikowanie obiektów bez konieczności uruchamiania samej gry. Poniżej przedstawiono fragmenty tego skryptu (Rysunek 5.11), w których użyto kluczowych metod takich jak RemoveChildren do usuwania istniejących obiektów i PlaceObjects do tworzenia nowych układów obiektów. Te metody pozwalają na elastyczne i szybkie tworzenie oraz modyfikowanie scen. Wykorzystanie tego skryptu znacznie przyspieszyło proces tworzenia i dostosowywania poziomów w omawianym demo. Na przykładzie rysunku 5.12 widać, jak skrypt pozwala na tworzenie złożonych układów obiektów, które można szybko modyfikować i dostosowywać do potrzeb gry. Dodatkowo na rysunku 5.12 po prawej stronie przedstawiono efekt zastosowania skryptu 'AutoObjectPlacement' do szybkiego tworzenia obiektów na różnej wysokości w świecie gry. Te obiekty, chociaż pełnią głównie rolę dodatków wizualnych, mają również praktyczne zastosowanie, ponieważ mogą wejść w kolizję z latającym stateczkiem. Dzięki temu skryptowi możliwe było efektywne wypełnienie przestrzeni gry interesującymi elementami, które dodatkowo wpływają na dynamikę i wyzwania rozgrywki.

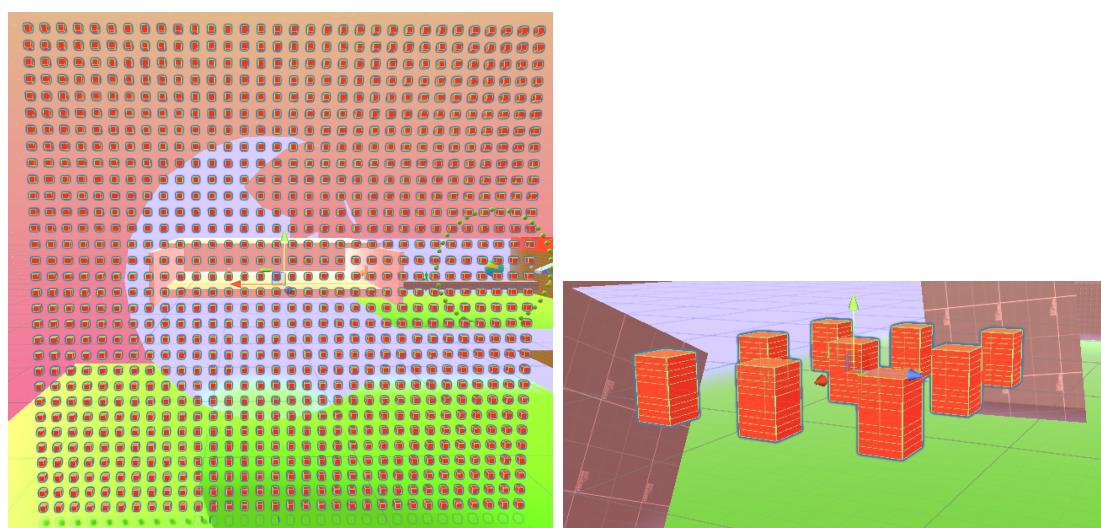
### 5.5.4 Tworzenie materiałów z użyciem shadergraph

Podczas pracy nad grafiką do gry, zrodził się pomysł stworzenia własnego skyboxu. Pierwsza wersja była inspirowana filmem [4], lecz w praktyce w środowisku gry 3D ujawniły się pewne niedoskonałości. W związku z tym, skybox został zmodyfikowany, aby zminimalizować widoczność tych niedoskonałości. Wiedza zdobyta podczas pracy

```
1 using UnityEngine;
2
3 [ExecuteInEditMode] // Wykonywanie w trybie edycji
4 public class AutoObjectPlacement : MonoBehaviour
5 {
6 ...
7     public bool run;
8     private bool isInEditMode = false;
9 ...
10    void Update()
11    {
12        if(run){
13            isInEditMode = Application.isPlaying;
14            if (!isInEditMode)
15                // Zabezpieczenie przed wykonywaniem skryptu podczas gry
16                {
17                    RemoveChildren(transform);
18                // Pozbywanie się poprzednich obiektów
19                    PlaceObjects(transform);
20                // Rozmieszczanie obiektów według właściwości
21                }
22            }
23        }
24    // Ta metoda jest wywoływana automatycznie, gdy zmieniamy wartości parametrów
25    ...
```

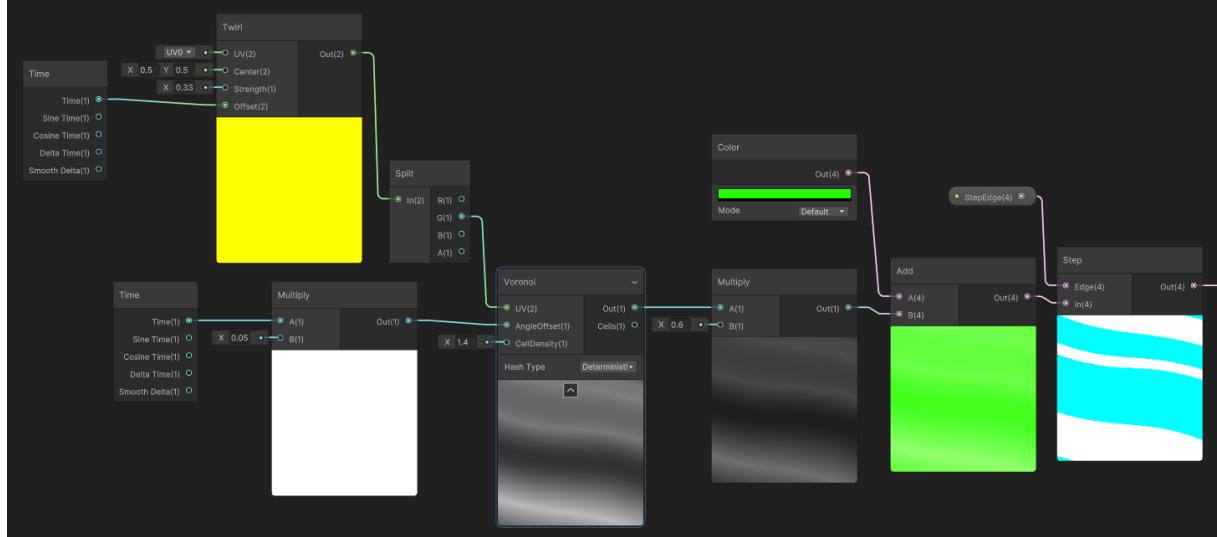
---

Rysunek 5.11: Fragmenty skryptu wykonywanego w edytorze



Rysunek 5.12: Efekty działania skryptu AutoObjectPlacement

nad skyboxem została wykorzystana do stworzenia autorskiego materiału służącego do sygnalizowania skupienia wzroku na obiekcie. Ten materiał wykorzystuje szum Woronoja do tworzenia animowanej tekstury, co przedstawiono na rysunku 5.13.



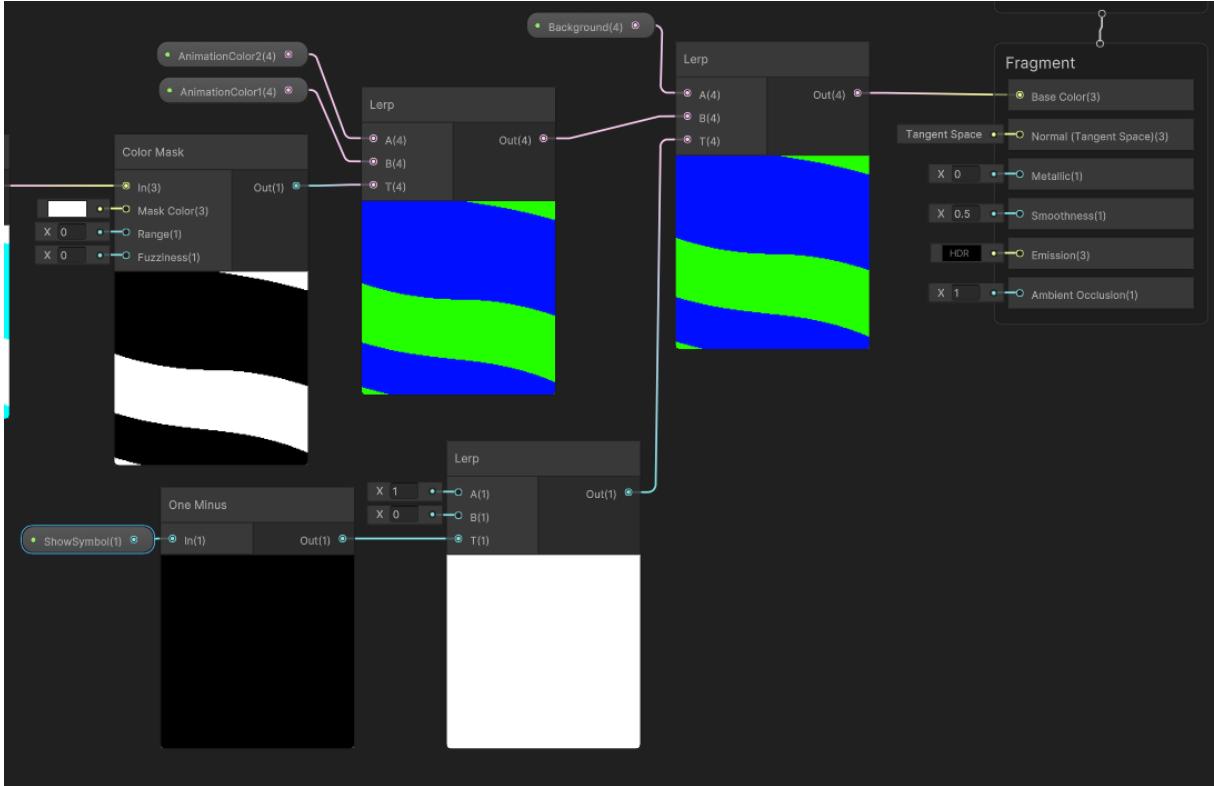
Rysunek 5.13: Fragment Shader Graph z użyтыm szumem Woronoja

Szum Woronoja jest wykorzystywany do generowania animacji, podczas gdy kształt animacji jest kształtowany przez węzeł UV Twirl (wir), który jest mocno zbliżony i przesunięty w pionie, co sprawia wrażenie przemieszczania się tekstury z góry na dół. Węzeł 'Step' zapewnia, że otrzymana tekstura ma ostre krawędzie.

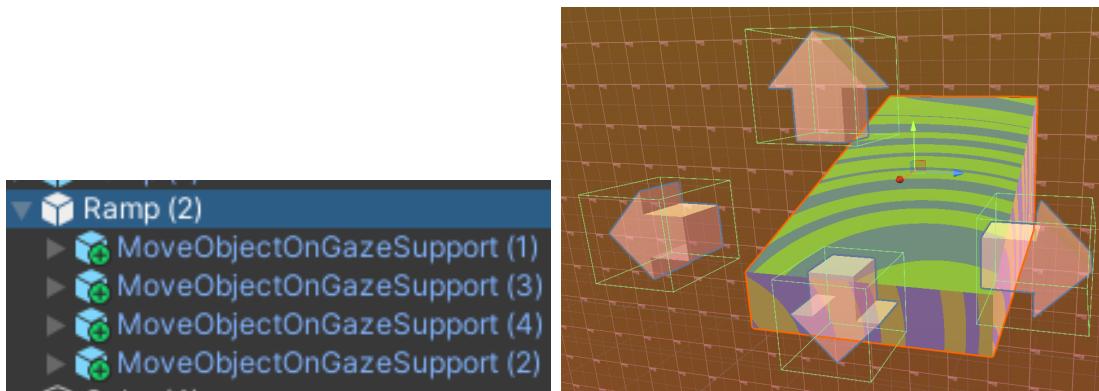
Kolejnym etapem było dodanie logiki odpowiedzialnej za wyświetlanie animacji tylko wtedy, gdy użytkownik patrzy na obiekt z danym materiałem. Zostało to osiągnięte przez użycie zmiennej 'ShowSymbol' oraz funkcji 'Lerp', co można zobaczyć na rysunku 5.14.

### 5.5.5 Logika 'wsiadania do statku' jako przykład projektowania interakcji z komponentem MoveOnGaze

Jednym z istotnych etapów rozwoju gry było stworzenie obiektów wykorzystujących komponent MoveOnGaze, który umożliwia przesuwanie ich za pomocą wzroku. Te obiekty, jak na przykład rampy, zostały skonstruowane z wykorzystaniem hierarchii, gdzie główny obiekt rampy zawiera dzieci będące wsparciami. Strukturę tę przedstawia lewa strona rysunku 5.15, natomiast struktura obiektu rampy w grze widoczna jest po prawej stronie tego samego rysunku.

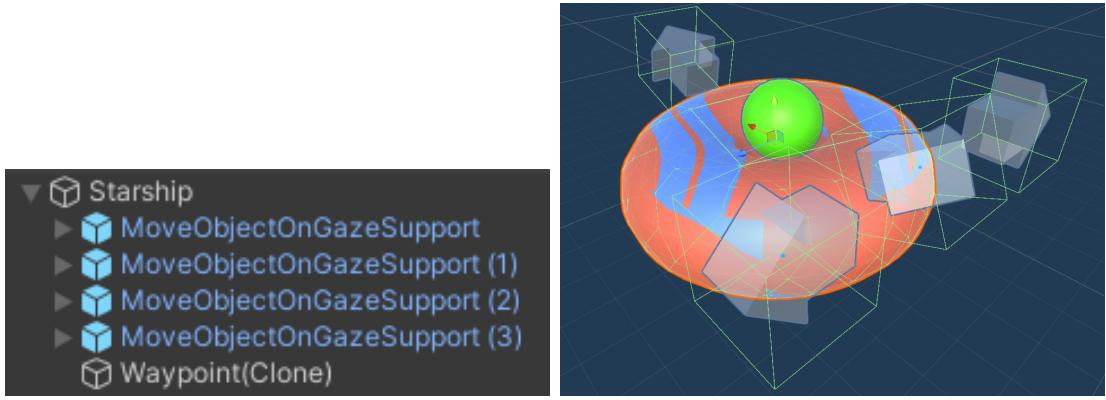


Rysunek 5.14: Wyjście shaderu



Rysunek 5.15: Hierarchia oraz dzieci obiektu składające się na rampę

Podobna struktura hierarchiczna została zastosowana w przypadku statku, co ukazuje rysunek 5.16. Różnica polega na dodatkowym obiekcie typu waypoint, który określa miejsce, gdzie gracz zostanie umieszczony po wejściu w interakcję ze statkiem.



Rysunek 5.16: Hierarchia oraz dzieci obiektu składające się na statek

Taka organizacja hierarchii obiektów zapewnia, że przy przesuwaniu głównego elementu, wszystkie z nim związane dzieci obiektu przemieszczają się razem z nim, eliminując potrzebę dodatkowej logiki do ich przesuwania. Jeżeli obiekty potomne nie powinny być przesuwane, powinny być umieszczone poza głównym obiektem.

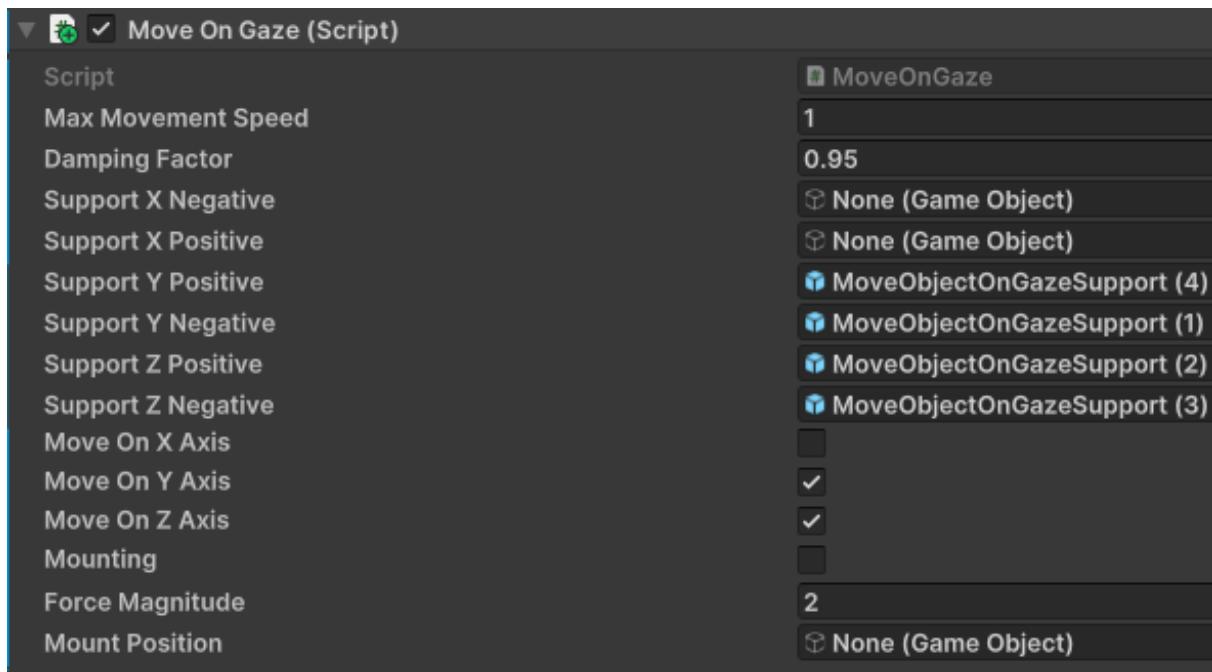
Konfiguracja skryptu MoveOnGaze wymaga zdefiniowania kilku parametrów, co zostało zilustrowane na rysunku 5.17:

- Określenie osi (X/Y/Z), po których obiekt ma być przesuwany.
- Dodanie obiektów wspierających dla wybranych osi.
- Ustawienie maksymalnej prędkości przesuwania oraz jej przyrostu za pomocą odpowiednio Max Movement Speed oraz Force Magnitude.
- W przypadku potrzeby przemieszczenia gracza razem z obiektem, zaznacza się zmienną 'mounting' i dodaje do 'Mounting Position' obiekt wskazujący miejsce, w którym gracz ma zostać umieszczony.
- Dostosowanie pozycji wszystkich obiektów zgodnie z wizją poziomu.

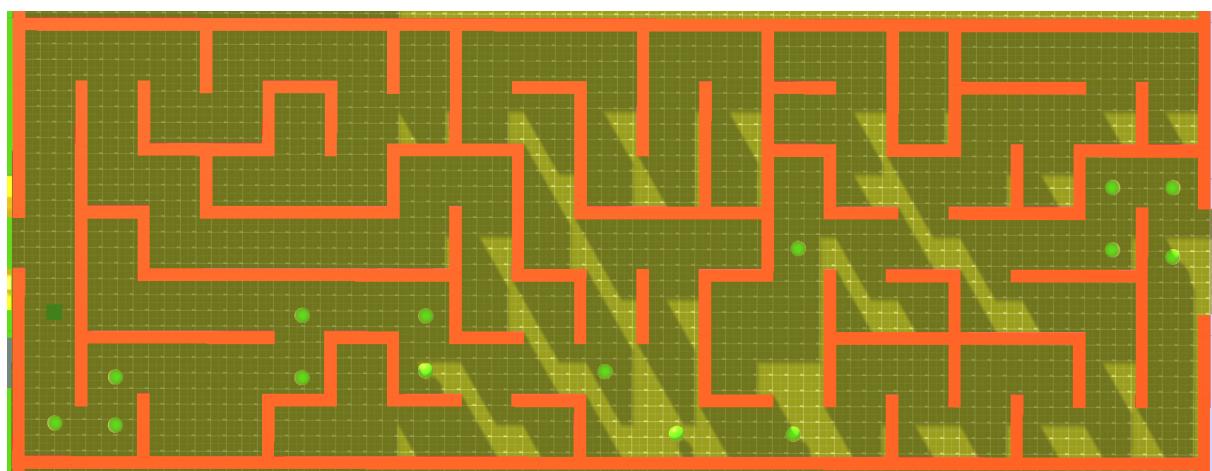
### 5.5.6 MoveOnWayPointsOnGaze: Przykład labiryntu

Ostatnią częścią implementacji jest skrypt używany w segmencie labiryntu, gdzie gracz ma za zadanie śledzić uciekający kwadrat. Rysunek 5.18 prezentuje widok z góry całego labiryntu, gdzie zielone kule oznaczają punkty kontrolne (waypoints) wyznaczające trasę ruchu kwadratu.

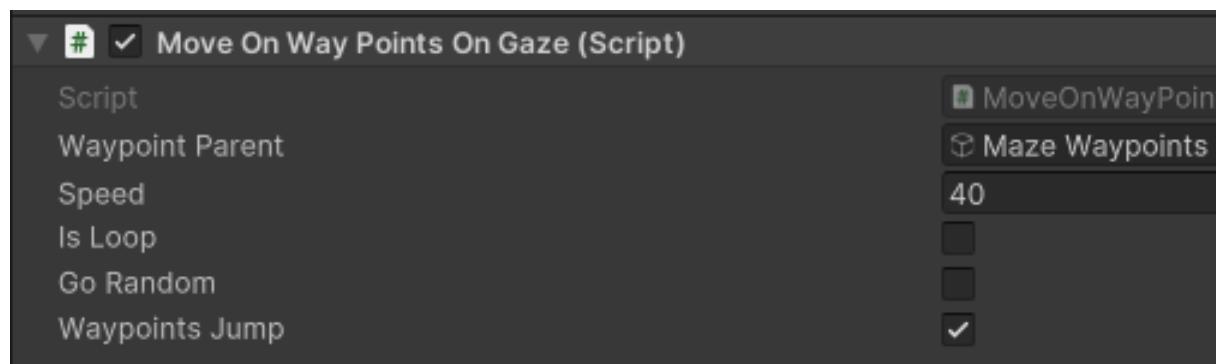
Skrypt jest aktywowany, gdy użytkownik skieruje wzrok na obiekt z tym komponentem. Aby skrypt działał prawidłowo, konieczne jest zdefiniowanie kilku parametrów. W 'Waypoint Parent' należy podać obiekt, którego dzieci stanowią waypoints. Parametr 'Speed' określa prędkość, z jaką porusza się obiekt. Dodatkowo, skrypt oferuje kilka opcji konfiguracji zachowania obiektu:



Rysunek 5.17: Konfiguracja skryptu MoveOnGaze



Rysunek 5.18: Perspektywa lotu ptaka na labirynt



Rysunek 5.19: Konfiguracja skryptu MoveOnWayPointsOnGaze

- 'Is Loop' umożliwia zapętlenie trasy, dzięki czemu po osiągnięciu ostatniego waypointa, obiekt wraca do pierwszego.
- 'Go Random' powoduje, że kolejne waypoints są wybierane losowo, co dodaje element nieprzewidywalności do ruchu obiektu.
- 'Waypoints Jump' określa, że obiekt może przemieszczać się, pokonując od 1 do 3 punktów kontrolnych za jednym razem. Ta wartość jest ustalona na stałe, ponieważ większa liczba punktów nie znalazła praktycznego zastosowania w demo.



# Rozdział 6

## Weryfikacja i walidacja

### 6.1 Sposób testowania w ramach pracy

W procesie tworzenia omawianego demo technologicznego, kluczowym elementem zapewniającym jego jakość, niezawodność oraz optymalizację, było zastosowanie iteracyjnego podejścia do rozwoju i testowania. Ten proces pozwolił na elastyczne zarządzanie zmianami i ciągłe doskonalenie produktu na każdym etapie jego tworzenia.

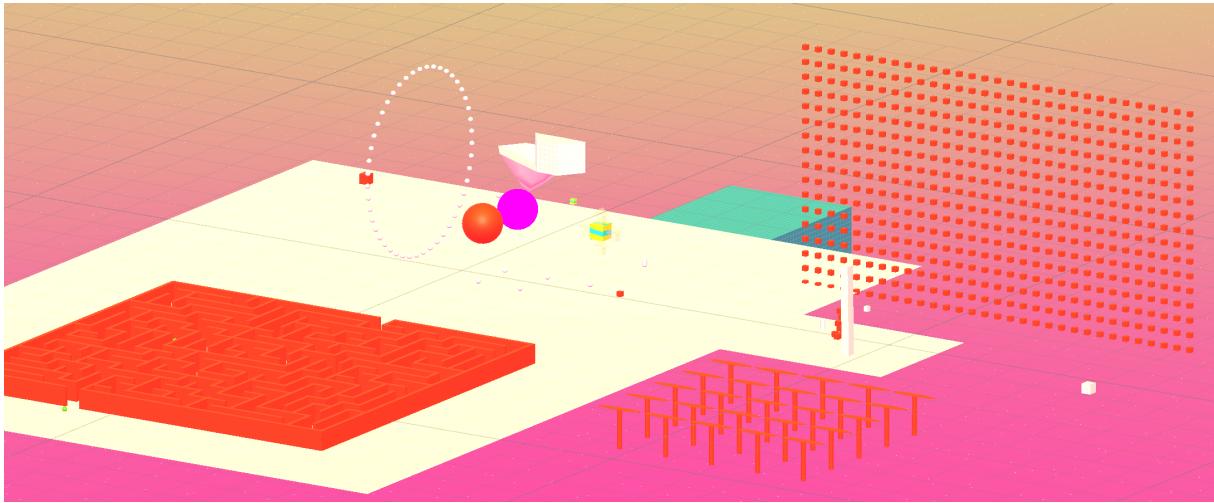
Od początku, szczególny nacisk został położony na testowanie każdego indywidualnego skryptu i funkcjonalności niezależnie, jeszcze przed ich zintegrowaniem z większym projektem. Taki sposób działania umożliwił wczesne wykrywanie i rozwiązywanie problemów na poziomie pojedynczych komponentów, co było kluczowe dla zachowania wysokiej jakości kodu oraz zapewnienia skuteczności demonstracji technologicznych aspektów gry.

Po etapie testowania jednostkowego, nastąpiło łączenie poszczególnych skryptów i elementów w spójne segmenty demo. Ten etap również zawierał testowanie, skoncentrowane na interakcji między różnymi skryptami i modułami. Było to istotne, aby upewnić się, że zintegrowane komponenty współpracują ze sobą płynnie i efektywnie, tworząc koherentne doświadczenie.

Ostatecznym etapem były kompleksowe testy całego demo, mające na celu nie tylko weryfikację funkcjonalności i wydajności, ale także ocenę ogólnego doświadczenia użytkownika. Testy te przeprowadzane były z udziałem osób trzecich, które nie miały wcześniejszej styczności z prezentowanym demo, co pozwoliło na uzyskanie obiektywnych informacji zwrotnych i dalsze dopracowanie projektu.

### 6.2 Przypadki testowe zakres testowania

W początkowej fazie projektu, testowanie zostało przeprowadzone na specjalnie zaprojektowanej scenie testowej, którą przedstawiono na Rysunku 6.1.



Rysunek 6.1: Widok sceny testowej w rzucie izometrycznym

Pierwszym etapem testowania było wdrożenie i weryfikacja podstawowych skryptów, które zostały określone na początku projektu. Rozpoczęło się od testowania działania Tobii Unity SDK, korzystając z typowego przykładu – obracania obiektem po spojrzeniu na niego. Wykorzystano do tego kod dostępny w dokumentacji Tobii Unity SDK[9].

Następnie, skupiono się na stworzeniu i testowaniu wszystkich skryptów wykorzystujących technologię śledzenia wzroku. Testy te polegały głównie na manualnym sprawdzaniu funkcjonalności skryptów. Ze względu na prostotę niektórych skryptów i brak potencjalnych miejsc, w których mogą wystąpić błędy, ich testowanie często odbywało się równolegle z innymi testami. Przykładami bardziej złożonych przypadków testów manualnych były:

### 6.2.1 Test skryptu AutoObjectPlacement

Cel: Sprawdzenie automatycznego rozmieszczania obiektów przez skrypt.

Zakres: Użycie skryptu do dynamicznego generowania i usuwania obiektów w środowisku.

Kroki testowe:

- Stworzenie nowego pustego obiektu.
- Przypisanie do niego skryptu AutoObjectPlacement.
- Przypisanie obiektów do rozmieszczenia przez skrypt.
- Zmiana właściwości skryptu w trakcie działania.

Oczekiwane wyniki: Skrypt skutecznie rozmieszcza odpowiednią ilość obiektów w określonych odstępach, usuwa poprzednie obiekty, generuje nowe tylko w edytorze Unity, i jest nieaktywny podczas działania gry.

### **6.2.2 Test skryptu ColorChangeOnGazeChildren**

Cel: Weryfikacja zmiany kolorów obiektów i ich potomków pod wpływem spojrzenia.

Zakres: Reakcja skryptu na różne struktury obiektów.

Kroki testowe:

- Skierowanie wzroku na różne obiekty.
- Obserwacja zmiany kolorów obiektów i ich dzieci.
- Skierowanie wzroku w innym kierunku niż testowane obiekty.
- Obserwacja powrotu obiektów do oryginalnego koloru.

Oczekiwane wyniki: Skrypt zmienia kolor zarówno głównego obiektu, jak i wszystkich jego dzieci niezależnie od ich struktury.

### **6.2.3 Test skryptu ShowSymbolOnGaze oraz materiału indykatorów spojrzenia**

Cel: Sprawdzenie reakcji obiektów na zmianę punktu koncentracji wzroku.

Zakres: Zmiana kolorów i animacji w odpowiedzi na spojrzenie.

Kroki testowe:

- Patrzenie na obiekt i obserwacja zmiany koloru na animację.
- Zmiana punktu koncentracji wzroku i obserwacja powrotu do koloru białego.

Oczekiwane wyniki: Obiekty mają biały kolor, gdy na nie nie patrzymy; pojawia się animacja podczas spoglądania na nie.

### **6.2.4 Testy związane z wykrywaniem wzroku poszczególnych elementów**

Cel: Zbadanie skuteczności wykrywania wzroku dla różnych rozmiarów elementów.

Zakres: Interakcje z obiektami w zależności od ich wielkości i odległości.

Kroki testowe:

- Zbliżenie się do obiektu w przewidzianej do interakcji odległości.
- Skierowanie wzroku na obiekt.
- Sprawdzenie reakcji na podstawie skryptu ColorChangeOnGazeChildren.

Oczekiwane wyniki: Wykrycie wzroku i odpowiednia interakcja z obiektami różnych wielkości.

### 6.2.5 Testy skryptu MoveOnGaze

Cel: Ocena działania skryptu umożliwiającego przemieszczanie obiektów przez spojrzenie.

Zakres: Aktywacja, przesuwanie i dezaktywacja obiektów.

Kroki testowe:

- Aktywacja obiektu ze skryptem.
- Pojawienie się obiektów wsparcia i ich interakcje.
- Chaotyczne patrzenie na obiekty wsparcia i obserwacja zachowań.
- Dezaktywacja obiektu i zniknięcie obiektów wsparcia.

Oczekiwane wyniki: Płynne przemieszczanie obiektu, właściwe działanie obiektów wsparcia i brak nieporządzanych zachowań.

### 6.2.6 Testy interakcji ze statkiem

Cel: Weryfikacja funkcjonalności lotu statku i interakcji z nim.

Zakres: Lot, kolizje, wsiadanie i wysiadanie z statku.

Kroki testowe:

- Sprawdzenie lotu statku.
- Testowanie kolizji ze środowiskiem.
- Wsiadanie i wysiadanie z statku.

Oczekiwane wyniki: Statek poprawnie lata, reaguje na kolizje i umożliwia interakcje z graczem.

### 6.2.7 Testy skryptu MoveWhenNotLookedAt przypisanego do kulek

Cel: Sprawdzenie reakcji kulek na spojrzenie gracza.

Zakres: Ruch kulek, kolizje i reakcja na spojrzenie.

Kroki testowe:

- Obserwacja ruchu kulek w kierunku gracza.
- Testowanie kolizji z graczem.
- Zatrzymanie kulek po zarejestrowaniu spojrzenia.

Oczekiwane wyniki: Kulki poruszają się w stronę gracza, reagują na kolizje i zatrzymują się, gdy gracz na nie spojrzy.

### 6.2.8 Analiza testów funkcjonalnych poziomów gry

- Test etapu początkowego

Cel: Weryfikacja reakcji siatki kostek na wzrok gracza.

Kroki testowe: Sprawdzenie reakcji kostek na spojrzenie: Każda kostka powinna na chwilę świecić na żółto i powiększać się. Przejście na pierwszy poziom po zakończeniu testów.

- Test pierwszego poziomu

Cel: Ocena interakcji z rampami i zachowanie w labiryncie.

Kroki testowe: Testowanie przesuwania ramp wzrokiem, aby umożliwić przejście do labiryntu. W labiryncie: Weryfikacja reakcji uciekającego kwadratu na spojrzenie oraz upewnienie się, że nie przechodzi on przez ściany. Sprawdzenie, czy zbliżanie się do ścian nie spowalnia postaci gracza.

- Test drugiego poziomu

Cel: Sprawdzenie mechaniki wsiadania do statku, nawigacji, kolizji oraz interakcji z bramą końcową.

Kroki testowe: Testowanie wsiadania do statku i nawigacji w każdym kierunku. Próby wlatywania w obiekty w celu testowania kolizji. Sprawdzenie mechaniki przesuwania i skalowania bramy końcowej w zależności od odległości od gracza. Testowanie wysiadania z statku na platformę.

- Test trzeciego poziomu

Cel: Weryfikacja interakcji z kulami, działania platform przesuwnych oraz mechaniki skakania postaci.

Kroki testowe: Sprawdzenie, czy kule pojawiają się tylko po wejściu gracza na poziom. Testowanie interakcji kul na wzrok gracza. Weryfikacja działania platform przesuwnych. Sprawdzenie, czy kule znikają po dotarciu do obszaru końcowego. Testowanie mechaniki skakania postaci, dostosowując ją do rozmieszczenia platform.

## 6.3 Wykryte i usunięte błędy

W trakcie rozwoju gry zidentyfikowano i naprawiono liczne błędy:

- **Skrypt AutoObjectPlacement:** Występował problem z jego nieskończonym wykonywaniem, co negatywnie wpływało na wydajność edytora. Problem rozwiązano przez wprowadzenie zmiennej aktywującej skrypt tylko podczas edycji.

- **Skrypt ColorChangeOnGazeChildren:** Początkowo skrypt nie zmieniał koloru dzieci obiektów. Naprawiono to przez pobieranie listy dzieci i zastosowanie jej w skrypcie.
- **Skrypt ShowSymbolOnGaze i powiązany materiał:** Napotkano błąd związany ze zmianą widoczności animacji, przez co znikała po spojrzeniu i nie wracała do stanu początkowego. Problem rozwiązano przez refaktoryzację kodu i poprawę kodu shadera.
- **Latacie statkiem:** W pierwszych iteracjach występowało szarpanie obiektu gracza po wejściu do statku. Problem rozwiązano, powiązując pozycję gracza ze statkiem przez waypoint.
- **Kolizje podczas latania statkiem:** Początkowo statek nie reagował na kolizje. Problem rozwiązano przez użycie metod fizyki zamiast prostego przesuwania obiektów.
- **Wypadanie gracza poza obszar gry:** Gracz spadał w nieskończoność. Problem rozwiązano przez dodanie obszarów kontrolnych i przenoszenie gracza do punktu kontrolnego.
- **Gracz przylepiał się do ścian labiryntu:** Problem rozwiązano przez znaczne zredukowanie siły tarcia gracza względem ścian.
- **Brak możliwości opuszczenia statku na końcu 2 poziomu:** Naprawiono przez wywołanie metody skoku z ruchu gracza po zakończeniu interakcji ze statkiem.
- **Kulki na 3 poziomie:** Pojawiały się niewłaściwie lub za wcześnie. Naprawiono przez refaktoryzację kodu odpowiedzialnego za interakcję.

## 6.4 Wyniki badań eksperymentalnych

W ramach badań eksperymentalnych przeprowadzono testy użyteczności, angażując osoby trzecie, które nie miały wcześniejszej styczności z demem gry. Ta metoda umożliwiła uzyskanie obiektywnych opinii na temat różnych aspektów projektu. Zauważono, że niektóre etapy, które twórca mógł przecenić ze względu na osobiste zaangażowanie, były zbyt długie i mogły prowadzić do uczucia zmęczenia wśród użytkowników końcowych. Te obserwacje są kluczowe dla dalszego dostosowywania rytmu i tempa gry, aby zwiększyć jej atrakcyjność dla szerszej grupy odbiorców.

Dodatkowo przeprowadzono testy wydajnościowe gry na dwóch urządzeniach z systemem Windows 11. Pierwszy test, wykonany na stacjonarnym komputerze z 16 GB pamięci RAM, kartą graficzną NVIDIA RTX 3060 i procesorem Ryzen 5 5600X,

w rozdzielczości 4K, wykazał wyniki w przedziale od 350 do 430 klatek na sekundę przy wyłączonej funkcji synchronizacji G-Sync. Drugi test, przeprowadzony na laptopie z procesorem Ryzen 7 6800U, zintegrowaną kartą graficzną i 16 GB pamięci RAM, w rozdzielczości 2880 x 1800, pokazał, że gra działała w przedziale 90-100 klatek na sekundę przy zasilaniu baterijnym, a liczba ta wzrastała do 110-130 klatek na sekundę przy korzystaniu z zasilacza. Wyniki te wskazują na wysoką wydajność gry na różnych konfiguracjach sprzętowych.



# Rozdział 7

## Podsumowanie i wnioski

Projekt ten skutecznie eksplorował potencjał technologii śledzenia wzroku w kontekście gier wideo, demonstrując nowatorskie sposoby interakcji użytkownika z grą. Poprzez praktyczne zastosowanie tej technologii w formie demonstracyjnego prototypu, udało się empirycznie ocenić i zrozumieć zarówno zalety, jak i ograniczenia śledzenia wzroku w świecie gier.

Zaimplementowane w prototypie interakcje, takie jak obracanie, zmiana koloru, kształtu czy przemieszczanie obiektów, nie tylko wzbogaciły doświadczenia użytkownika, ale również podkreśliły możliwości, jakie technologia ta oferuje. Szczególnie znaczące okazało się wykorzystanie śledzenia wzroku do przesuwania obiektów przy użyciu elementów pomocniczych, co wskazuje na nowe kierunki rozwoju w interakcji z grami.

Projekt ten potwierdza, że technologia śledzenia wzroku ma znaczący potencjał w przyszłościowym projektowaniu gier, oferując graczom bardziej intuicyjne i zanurzające formy interakcji. Jednakże, równie istotne jest uwzględnienie wyzwań i ograniczeń, które ta technologia stawia, szczególnie w kontekście dostępności i uniwersalności zastosowań.

Kierunki dalszych prac:

- Wsparcie dla zaawansowanych urządzeń śledzących wzrok: Rozszerzenie projektu o wsparcie dla bardziej zaawansowanych urządzeń śledzących wzrok, różnych od Tobii EyeTracker 5, pozwoliłoby na dalsze badanie i optymalizację tej technologii w różnych kontekstach.
- Poprawa jakości wizualnej: Dodanie większej liczby animacji, lepszej jakości modeli 3D oraz rozszerzenie ilości poziomów znacząco wzbogaciłoby wizualną atrakcyjność i złożoność projektu.
- Parametryzacja funkcji śledzenia wzroku: Badanie i dostosowywanie funkcji śledzenia wzroku w celu znalezienia najbardziej naturalnych i intuicyjnych metod interakcji dla użytkowników mogłoby znacząco wpłynąć na użyteczność i przyjemność z gry.



# Bibliografia

- [1] Samuel Almeida, Ana Isabel Veloso, Licinio Gomes Roque i Óscar Mealha. „The Eyes and Games: A Survey of Visual Attention and Eye Tracking Input in Video Games”. W: *SBC - Proceedings of SBGames 2011*. 2011, s. 5.
- [2] EyeTracking. *King Midas and the Golden Gaze*. 2020. URL: <https://www.eyetracking.com/king-midas-and-the-golden-gaze/> (term. wiz. 14.01.2024).
- [3] Borna Fatehi, Casper Harteveld i Christoffer Holmgård. „Guiding Game Design Decisions via Eye-Tracking: An Indie Game Case Study”. W: *2022 Symposium on Eye Tracking Research and Applications*. 2022, s. 1–7.
- [4] The Game Guy. *How to make a Gradient Skybox with Stars in Unity / Shader Graph Unity*. 2020. URL: <https://www.youtube.com/watch?v=yw2J9NWRdow> (term. wiz. 20.01.2024).
- [5] Michael Lankes, Maurice Sporn, Andreas Winkelbauer i Barbara Stiglbauer. „Looking Confused?—Introducing a VR Game Design for Arousing Confusion Among Players”. W: *2022 Symposium on Eye Tracking Research and Applications*. 2022, s. 1–6.
- [6] Thorsten Roth, Martin Weier, André Hinkenjann, Yongmin Li i Philipp Slusallek. „A quality-centered analysis of eye tracking data in foveated rendering”. W: *Journal of eye movement research* 10.5 (2017).
- [7] Unity Technologies. *Unity Documentation - Component*. 2024. URL: <https://docs.unity3d.com/ScriptReference/Component.html> (term. wiz. 24.01.2024).
- [8] Tobii. *Assassin's Creed® Valhalla - Eye Tracking Features*. 2020. URL: <https://gaming.tobii.com/games/assassins-creed-valhalla/> (term. wiz. 13.01.2024).
- [9] Tobii. *Getting Started - Tobii Developer Zone*. 2022. URL: <https://developer.tobii.com/pc-gaming/unity-sdk/getting-started/> (term. wiz. 21.01.2024).
- [10] Colin Ware i Harutune H Mikaelian. „An evaluation of an eye tracker as a device for computer input2”. W: *Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*. 1986, s. 183–188.



## **Dodatki**



# Spis skrótów i pojęć

Gameplay rozgrywka, odnosi się do doświadczeń i interakcji, które gracz ma podczas gry w grę wideo.

Demo demonstracja, wersja demonstracyjna gry lub oprogramowania, która pozwala użytkownikowi wypróbować ograniczone funkcje przed zakupem pełnej wersji.

Shader Graph graficzny edytor shaderów w Unity, umożliwia tworzenie shaderów za pomocą wizualnego interfejsu bez konieczności pisania kodu.

UI interfejs użytkownika (ang. *User Interface*), elementy wizualne gry lub aplikacji, przez które użytkownik wchodzi w interakcję z programem.

Lerp interpolacja liniowa (ang. *Linear Interpolation*), technika w grafice komputerowej i matematyce używana do płynnego przejścia między dwoma wartościami.

Waypoint punkt nawigacyjny, używany w grach do określania trasy lub celu dla postaci lub obiektów ruchomych.

Skybox technika używana do tworzenia tła otaczającego scenę 3D, zwykle przedstawiającą niebo lub horyzont.



# **Lista dodatkowych plików, uzupełniających tekst pracy**

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- film pokazujący działanie opracowanego oprogramowania,



# Spis rysunków

3.1	Podpis rysunku po rysunkiem. . . . .	9
4.1	Siatka z czerwonymi kwadratami, z wyraźnie wyróżnionym żółtym kwadratem, stanowiącym punkt koncentracji wzroku. . . . .	12
4.2	Menu gry z widocznymi opcjami "Kontynuuj", "Reset" oraz "Wyjdź z gry" . . . . .	13
4.3	Widok pierwszego poziomu . . . . .	13
4.4	Obiekty z znacznikami skupienia wzroku . . . . .	14
4.5	Obiekt aktywny z widocznymi wsparciami podczas przesuwania w lewo . . . . .	14
4.6	Demonstracja obiektu przesuwanego w dwóch wymiarach . . . . .	14
4.7	"Uciekający" obiekt reagujący na wzrok gracza . . . . .	15
4.8	Widok drugiego poziomu, na pierwszym planie wskazywany przez żółte strzałki stateczek . . . . .	15
4.9	Stateczek po skupieniu na nim wzroku . . . . .	16
4.10	Wsparcia pozwalające na jazdę w lewo, prosto oraz w prawo . . . . .	16
4.11	Przykład używania wsparcia do jazdy w lewo . . . . .	17
4.12	Przesuwana brama z znacznikiem skupienia wzroku. . . . .	17
4.13	Wielkość wsparć w zależności od dystansu do gracza . . . . .	17
4.14	Widok trzeciego poziomu . . . . .	18
4.15	Zbliżająca się kula . . . . .	18
4.16	Kula podczas skupienia na niej wzroku . . . . .	19
4.17	Dynamiczna platforma przed i w trakcie transformacji. . . . .	19
4.18	Ekran końcowy . . . . .	19
5.1	Klasa z użyciem wykrywania wzroku w dzieciach . . . . .	26
5.2	Zmienne klasy MoveOnGaze . . . . .	27
5.3	Podstawowe przesuwanie . . . . .	28
5.4	Przesuwanie z użyciem fizyki . . . . .	29
5.5	Skalowanie z użyciem Lerp . . . . .	30
5.6	Metody odpowiedzialne za upłynnienie skalowania . . . . .	31
5.8	Widok komponentu ResetPlayerPosition w Unity . . . . .	32
5.9	Widok komponentu TriggerObjectControl . . . . .	32

5.7 Porównanie widoku sceny po lewej z widokiem z kamery po prawej . . . . .	32
5.10 Porównanie obiektu menu w edytorze po lewej i efektu uzyskanemu na kamerze po prawej . . . . .	33
5.11 Fragmenty skryptu wykonywanego w edytorze . . . . .	34
5.12 Efekty działania skryptu AutoObjectPlacement . . . . .	34
5.13 Fragment Shader Graph z użyтыm szumem Woronoja . . . . .	35
5.14 Wyjście shaderu . . . . .	36
5.15 Hierarchia oraz dzieci obiektu składające się na rampę . . . . .	36
5.16 Hierarchia oraz dzieci obiektu składające się na statek . . . . .	37
5.17 Konfiguracja skryptu MoveOnGaze . . . . .	38
5.18 Perspektywa lotu ptaka na labirynt . . . . .	38
5.19 Konfiguracja skryptu MoveOnWayPointsOnGaze . . . . .	39
6.1 Widok sceny testowej w rzucie izometrycznym . . . . .	42