

**WYŻSZA SZKOŁA INFORMATYKI I ZARZĄDZANIA  
„Copernicus” we Wrocławiu**

---

**WYDZIAŁ INFORMATYKI**

Kierunek studiów: **Informatyka**

Poziom studiów: **Studia pierwszego stopnia-inżynierskie**

Specjalność: **Systemy i Sieci Komputerowe**

**PRACA DYPLOMOWA INŻYNIERSKA**

**Radosław Matusiak**

**Architektura, projekt oraz implementacja gry  
platformowej w silniku Unity.**

**Architecture, design and implementation of a platform game in the Unity  
engine**

**Ocena pracy:**.....

(ocena pracy dyplomowej, data, podpis promotora)

.....  
(Pieczęć Uczelni)

**Promotor:**

**Mgr Inż. Bartosz Majorowski**

---

**WROCŁAW 2022**

## **Streszczenie**

Celem pracy był projekt oraz implementacja gry platformowej w silniku Unity, wraz z opisem architektury. Przygotowana implementacja gry zawiera wersję demonstracyjną gry. Część grywalna projektu zawiera: mechaniki poruszania się postacią, modele przeciwników wraz z ich logiką oraz dodatkowe obiekty do kolekcjonowania przez bohatera gry. Specjalnie dla gry zostały stworzone modele 3D za pomocą programu Blender, wraz z ich animacjami.

## **Abstract**

The goal of this work was to design and implement a platform game in Unity engine, including a description of the architecture. The prepared game implementation contains a demo version of the game. The playable part of the project contains: mechanics of moving the character, models of opponents along with their logic and additional objects to be collected by the game hero. Specially for the game were created 3D models using Blender, along with their animations.

1 Wstęp.....	4
1.1 Gry komputerowe .....	4
1.2 Gry platformowe.....	5
1.3 Opis produktu .....	5
1.3.1 Rozgrywka .....	5
1.3.2 Bohater.....	6
1.3.3 Zagrożenia.....	6
1.3.4 Poziom .....	7
1.4 Cel pracy .....	7
1.5 Zakres pracy .....	7
1.6 Przegląd istniejących rozwiązań.....	8
1.6.1 Yoshi's Crafted World.....	8
1.6.2 Marsupilami: Hoobadventure .....	9
1.6.3 LittleBigPlanet 3 .....	10
2 Architektura i projekt.....	11
2.1 Wymagania funkcjonalne .....	11
2.2 Wymagania нефunkcjonalne .....	11
2.3 Architektura .....	12
2.3.1 Struktura projektu .....	12
2.3.2 Struktura wykorzystanych pakietów .....	13
2.3.3 Struktura obiektów na scenie .....	14
2.4 Projekt.....	15
2.4.1 Skrypty bohatera .....	16
2.4.2 Skrypty Przeciwników .....	25
2.4.3 Skrypty obiektów możliwych do zebrania.....	31
2.4.4 Skrypt obiektu GameManagerHelper .....	34
2.4.5 Skrypt obiektów typu Checkpoint .....	36
2.4.6 Skrypt obiektów typu Moving Platform .....	37
3 Opis technologii i dalszy rozwój projektu .....	39
3.1 Opis technologii.....	39
3.2 Testowanie.....	40
3.3 Dalszy rozwój projektu.....	41
Zakończenie .....	41
Spis ilustracji .....	42
Bibliografia.....	44

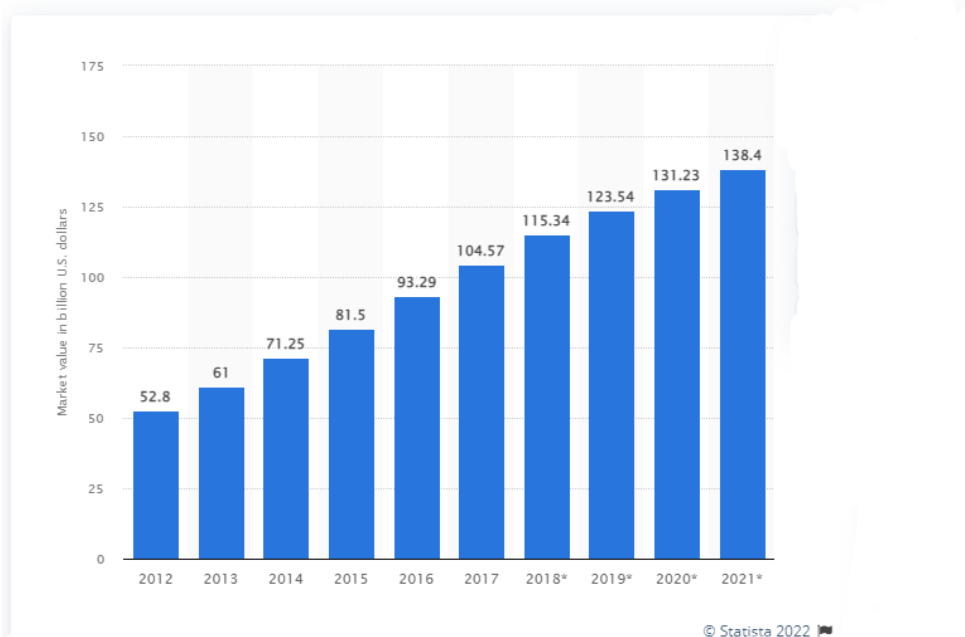
# 1 Wstęp

Ten rozdział jest wprowadzeniem w tematykę oraz terminologię gier komputerowych z podgatunku gier platformowych, ponadto zawiera opis implementowanego w ramach pracy produktu, wraz z zakresem pracy jak również przeglądem istniejących rozwiązań.

## 1.1 Gry komputerowe

Gry komputerowe są grami projektowanymi na dedykowane urządzenia takie jak, np. Sony PlayStation lub Microsoft Xbox, oraz komputery osobiste i gogle wirtualnej rzeczywistości. Pośród gier komputerowych możemy odnaleźć zróżnicowane spektrum podgatunków, takich jak m.in: gry zręcznościowe, gry first-person shooter (strzelanki pierwszoosobowe) lub gry platformowe. W dobie obecnej, rynek gier komputerowych jest prężnie rozwijającym się sektorem [1].

**Value of the global video games market from 2012 to 2021**  
(in billion U.S. dollars)



Rys 1.1 Wartość rynkowa branży gier wideo wyrażona w miliardach dolarów (źródło: [2])

## **1.2 Gry platformowe**

Gry platformowe są podgatunkiem gier akcji, w których osią rozgrywki jest poruszanie się bohaterem po urozmaiconym poziomie. Podczas rozgrywki, celem gracza jest przeskakiwanie między platformami, które charakteryzują się różnorodnym umieszczeniem w terenie. Wymaga to od gracza, używania zróżnicowanych mechanik poruszania takich jak, skakanie lub wspinaczka po ścianach do nawigowania bohaterem, w celu ukończenia poziomu. W czasie trwania rozgrywki, gracz musi umiejętnie wykorzystywać teren oraz mechaniki bohatera w celu unikania lub likwidacji zagrożeń, takich jak np. pułapki lub przeciwnicy. Dodatkowo gracz ma możliwość zbierania różnorodnych nagród i dodatków takich jak: dodatkowe życia lub „power-upy”[3] czyli dodatki zmieniające mechanikę poruszania się bohaterem.

## **1.3 Opis produktu**

Produktem jest gra komputerowa w podgatunku platformowej, przeznaczona do użytku na komputery osobiste. Do jej realizacji użyto silnika Unity, wykorzystując moduły do produkcji gier 2D oraz 3D.

### **1.3.1 Rozgrywka**

Rozgrywka toczy się na trójwymiarowej planszy, ale gracz ma możliwość poruszania się jedynie w dwóch wymiarach. Gracz, może poruszać się po platformach używając mechanik skakania oraz wspinania unikając przeszkód takich jak kolce czy armaty. Bohater sterowany przez gracza, podczas próby ukończenia poziomu ma możliwość dwukrotnego otrzymania obrażeń. Po otrzymaniu dwóch punktów obrażeń bohater gracza traci jedną możliwość ukończenia poziomu, tzw. Życie. W toku trwania rozgrywki, gracz ma możliwość zbierania dodatkowych „żyć”, które zwiększają ilość możliwych prób na poziom oraz monet, które są elementem ozdobnym i nie mają wpływu na rozgrywkę, ale ich zebrana ilość jest podsumowana na koniec poziomu. Na przestrzeni poziomu, gracz ma możliwość tymczasowego zapisu rozgrywki w odpowiednio przystosowanych to tego miejscach, tzw. Checkpoint. Gdy bohater kontrolowany przez gracza straci możliwość poruszania się, będzie mógł kontynuować rozgrywkę od ostatniego „Checkpointu”, który został aktywowany. W trakcie rozgrywki, za pomocą mechaniki „dashowania”, czyli szybkiego przemieszczania się w locie, gracz ma możliwość niszczenia niektórych obiektów, takich jak np. skrzynki. Po ukończeniu poziomu kontynuować dalszą grę lub zakończyć program.

### 1.3.2 Bohater

Bohater sterowany przez gracza posiada różnorodne umiejętności, które wymagają odpowiedniego wykorzystywania w czasie trwania rozgrywki. Głównymi umiejętnościami bohatera są: bieganie, skakanie, wspinaczka po ścianach i „dashowanie”.



Rys 1.3.2 Model 3D bohatera. (źródło: Własne)

### 1.3.3 Zagrożenia

W grze występują różnego rodzaju zagrożenia, które bohater sterowany przez gracza musi unikać. Niektóre z zagrożeń powodują natychmiastową utratę „życia” bohatera, a niektóre z przeszkód takie jak kolce, armaty lub piła powodują otrzymanie jednego punktu obrażeń.



Rys 1.3.2 Modele 3D. „Armata” i „szpikulce”. (źródło: Własne)

### **1.3.4 Poziom**

Rozgrywka ma miejsce na przygotowanym poziomie, który wypełniony jest zadaniami, które bohater kontrolowany przez gracza ma za zadanie wykonać. Głównym zadaniem gracza jest aktywowanie „Checkpointu”, który pełni rolę końca poziomu. Celami pobocznymi gracza są, znalezienie czterech różnokolorowych kryształów, odblokowanie wszystkich pobocznych „Checkpointów”, oraz zebranie jak największej ilości monet.

### **1.4 Cel pracy**

Celem pracy jest projekt oraz implementacja gry komputerowej typu gra platformowa, wraz z omówieniem architektury projektu w silniku Unity.

### **1.5 Zakres pracy**

W skład zakresu pracy wchodzi implementacja wersji demonstracyjnej gry na komputery osobiste. Praca składa się z dwóch rozdziałów. Pierwszy rozdział stanowi opis architektury projektu zawierający wymagania funkcjonalne oraz нефункционалне. Dalsza część stanowi projekt gry, który zawiera opis elementów stanowiących rdzeń rozgrywki. Drugi rozdział obejmuje dalsze perspektywy rozwoju projektu oraz opis technologii, które zostały wykorzystane w trakcie realizacji.

## 1.6 Przegląd istniejących rozwiązań

### 1.6.1 Yoshi's Crafted World

Gra autorstwa Good-Feel skierowana do najmłodszych odbiorców. Oprawa graficzna w tej produkcji stylizowana jest na dziecięce, papierowe wycinanki. Bohater gracza oprócz poruszania się po dwuwymiarowej przestrzeni, ma możliwość interakcji w trzecim wymiarze. Jest to dwunasta odsłona gier z serii „Yoshi” [8]. Gra została wydana wyłącznie na konsolę Nintendo Switch.



Rys 1.6.1 Zrzut ekranu z rozgrywki (źródło: [5])

#### Zalety

- Oprawa audio-wizualna
- Możliwość kooperacji z innym graczem
- Gra przystępna dla każdej grupy wiekowej

#### Wady

- Brak wyzwania dla doświadczonych graczy
- Gra została wydana jedynie na konsolę Nintendo Switch



### 1.6.2 Marsupilami: Hoobadventure

Gra oparta jest o komiks autorstwa André Franquin pt. „Marsupilami” [9]. W grze Ocellus Studio mamy do dyspozycji trzech, różnorodnych bohaterów. W grze znajduje się ponad 20 grywalnych plansz wraz z możliwym do dostosowania poziomem trudności. Dodatkowo gracz ma możliwość odblokowania pierwotnie ukrytych plansz wraz z postępem w rozgrywce [7].



Rys 1.6.2 Zrzut ekranu z rozgrywki (źródło: [6])

#### Zalety

- Oprawa audio-wizualna
- Responsywne i satysfakcjonujące sterowanie
- Zróżnicowany poziom trudności, z możliwością dostosowania przez gracza
- Gra została wydana na wiele popularnych obecnie konsol oraz na komputery osobiste.

#### Wady

- Mała liczba i różnorodność plansz.

### 1.6.3 LittleBigPlanet 3

Gra stworzona przez studio Sumo Digital oferuje różnorodną rozgrywkę podczas rozgrywki zarówno jedno, jak i wieloosobowej. W grze istnieje możliwość personalizacji bohatera, jak i tworzenie własnych plansz.



Rys 1.6.3 Zrzut ekranu z rozgrywki (źródło: [10])

#### Zalety

- Możliwość personalizowania rozgrywki
- Duża liczba dodatkowych plansz stworzonych przez graczy.

#### Wady

- Mała ilość zawartości dostarczonej przez twórców
- Gra wydana jedynie na Sony PlayStation 3 i Sony PlayStation 4

## **2 Architektura i projekt**

W tym rozdziale zawarto wymagania funkcjonalne i нефункционалне projektu, oraz projekt gry z opisem elementów rozgrywki.

### **2.1 Wymagania funkcjonalne**

Gra powinna

- Posiadać estetyczną, oraz klarowną warstwę audiowizualną.
- Umożliwić graczowi responsywne sterowanie bohaterem za pomocą klawiatury i/lub kontrolera
- Zawierać satysfakcjonujące wyzwanie dla gracza

Rozgrywka powinna

- Zawierać zróżnicowane mechaniki sterowania bohaterem
- Posiadać różnorodny, ciekawy w eksploracji poziom gry.
- Zawierać graficzną reprezentację postępów w przejściu poziomu gry.

### **2.2 Wymagania нефункционалне**

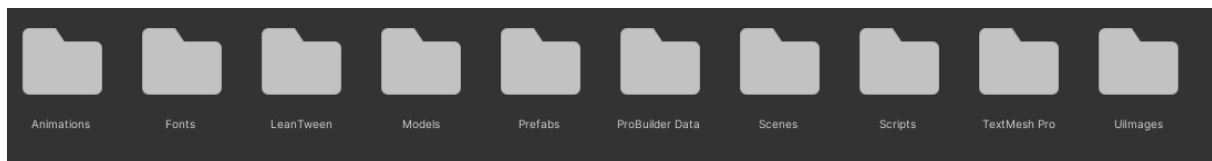
- Działanie gry na komputerach osobistych z systemem Windows 10 lub nowszym.
- Gry musi oferować płynność w rozgrywce na jak największej ilości urządzeniach.

## 2.3 Architektura

### 2.3.1 Struktura projektu

W projekcie znajduje się usystematyzowana struktura folderów, która prezentuje się następująco:

- Folder Animations – zawiera pliki AnimationClip, które zawierają animacje przypisane do konkretnych modeli w projekcie
- Folder Fonts – zawiera czcionki użyte w projekcie
- Folder LeanTween – zawiera pliki biblioteki LeanTween, która umożliwia generowanie klatek pośrednich między dwoma klatkami kluczowymi animacji[11].
- Folder Models – zawiera wykonane na potrzeby projektu modele 3D
- Folder Prefabs – zawiera prefaby [13], czyli GameObjecty wraz z komponentami, gotowe do implementacji na różnych scenach.
- Folder ProBuilder Data – zawiera pliki pakietu ProBuilder [14].
- Folder Scenes – zawiera gotowe sceny, czyli poziomy gry oraz menu główne.
- Folder Scripts – zawiera kody źródłowe skryptów obsługujące logikę gry
- Folder TextMesh Pro – zawiera pliki pakietu Textmesh Pro.
- Folder UiImages – zawiera wykonane na potrzeby projektu grafiki 2D interfejsu użytkownika.

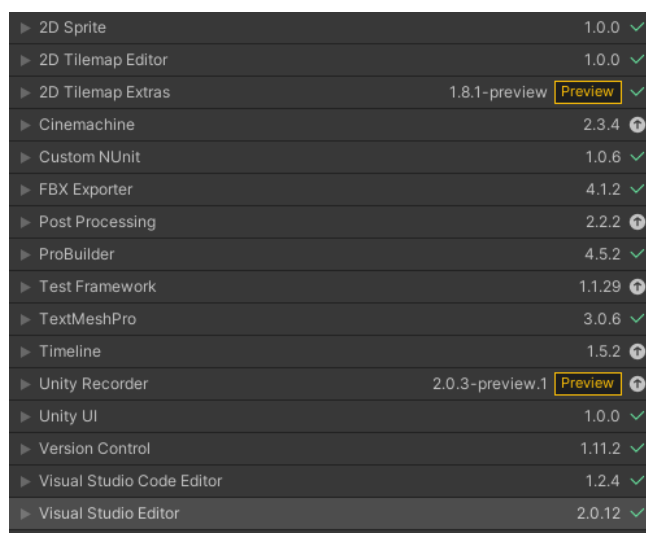


Rys 2.3.1 Zrzut ekranu z silnika Unity, przegląd katalogów (źródło: Własne)

## 2.3.2 Struktura wykorzystanych pakietów

W projekcie zostały użyte następujące pakiety:

- 2D Sprite – umożliwia dodawanie oraz edycję grafik 2D w silniku Unity.
- 2D Tilemap Editor – pakiet zawiera edytor Tilemap [16].
- 2D Tilemap Extras – pakiet jest rozszerzeniem funkcjonalności pakietu 2D Tilemap Editor o dodatkowe skrypty, takie jak np. możliwość dodawania własnych pędzli do malowania Tile na scenie.
- Cinemachine – pakiet zawiera skrypty umożliwiające większą kontrolę nad pracą kamery w programie.
- Custom NUnit – pakiet wymagany przez Unity Test Framework.
- FBX Exporter – pakiet umożliwia eksport obiektu do pliku fbx (patrz. ProBuilder).
- Post Processing – pakiet umożliwia wykorzystanie dodatkowych efektów graficznych w projekcie.
- ProBuilder – pakiet umożliwia tworzenie obiektów 3d bezpośrednio na scenie, np. w celach prototypowych.
- Test Framework – pakiet umożliwia wykonywanie testów automatycznych podczas edycji lub uruchomienia projektu.
- TextMeshPro – pakiet umożliwia wykorzystanie tekstu w projekcie.
- Timeline – pakiet umożliwia tworzenie sekwencji filmowych w grze (tzw cut-scene).
- Unity Recorder – pakiet umożliwia nagrywanie animacji lub plików video bezpośrednio z poziomu silnika Unity,
- Unity UI – pakiet umożliwia tworzenie interfejsów graficznych w projekcie.
- Version Control – pakiet umożliwia wykorzystanie kontroli wersji w projekcie za pomocą modułu Unity Collaborate lub Plastic SCM
- Visual Studio Code Editor, Visual Studio Editor – pakiety umożliwiają integrację środowisk Visual Studio i Visual Studio Code wraz z silnikiem Unity.



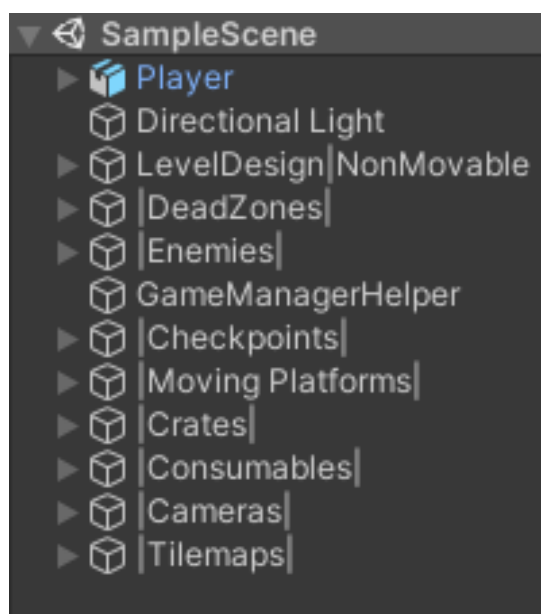
▶ 2D Sprite	1.0.0	✓
▶ 2D Tilemap Editor	1.0.0	✓
▶ 2D Tilemap Extras	1.8.1-preview	Preview ✓
▶ Cinemachine	2.3.4	⚙
▶ Custom NUnit	1.0.6	✓
▶ FBX Exporter	4.1.2	✓
▶ Post Processing	2.2.2	⚙
▶ ProBuilder	4.5.2	✓
▶ Test Framework	1.1.29	⚙
▶ TextMeshPro	3.0.6	✓
▶ Timeline	1.5.2	⚙
▶ Unity Recorder	2.0.3-preview.1	Preview ⚙
▶ Unity UI	1.0.0	✓
▶ Version Control	1.11.2	✓
▶ Visual Studio Code Editor	1.2.4	✓
▶ Visual Studio Editor	2.0.12	✓

Rys 2.3.2 Zrzut ekranu z silnika Unity, przegląd pakietów (źródło: Własne)

### 2.3.3 Struktura obiektów na scenie

Na scenie znajdują się następujące GameObjecty :

- Player – Jest to GameObject głównego gracza. Zawiera jego model 3d, szkielet potrzebny do animowania oraz element interfejsu użytkownika.
- Directional Light – Jest to źródło światła na scenie
- LevelDesign|NonMovable| - Jest to zbiorczy GameObject, zawierający elementy otoczenia, które nie posiadają możliwości ruchu
- |DeadZones| - Jest to zbiorczy GameObject zawierający strefy powodujące natychmiastową śmierć bohatera, bez modelu 3D.
- |Enemies| - Jest to zbiorczy GameObject, zawierający wszystkie GameObjecty zagrożeń bohatera w trakcie przechodzenia poziomu..
- GameManagerHelper – Jest to GameObject który zwiera główną logikę poziomu.
- |Checkpoints| - Jest to zbiorczy GameObject zawierający punkty kontrolne poziomu, do których po utracie życia może powrócić bohater.
- |MovingPlatforms| - Jest to zbiorczy GameObject zawierający wszystkie ruchome platformy które znajdują się na scenie.
- |Consumables| - Jest to zbiorczy GameObject zawierający wszystkie elementy możliwe do zebrania przez bohatera
- |Cameras| - Jest to zbiorczy GameObject zawierający wszystkie kamery znajdujące się na scenie.
- |Tilemaps| - Jest to zbiorczy GameObject zawierający wszystkie obiekty za pomocą Tilemapy.



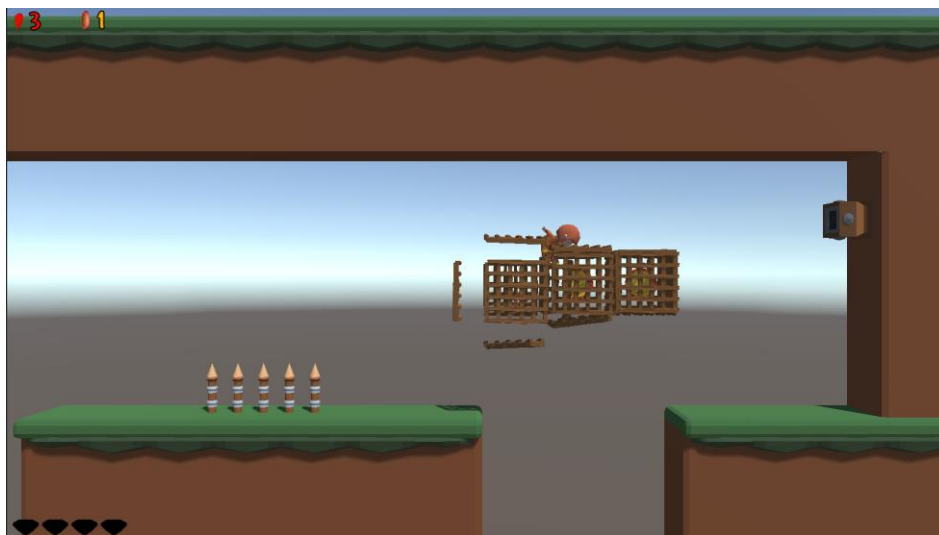
Rys 2.3.3 Zrzut ekranu z silnika Unity, przegląd GameObjectów na scenie(źródło: Własne)

## 2.4 Projekt

Gry i aplikacje tworzone w silniku Unity. Sceny składają się z Obiektów gry (GameObject) oraz z interfejsu użytkownika. Gra składa się z jednej sceny, wraz z zaprojektowanym interfejsem użytkownika.

Najważniejsze elementy gry to:

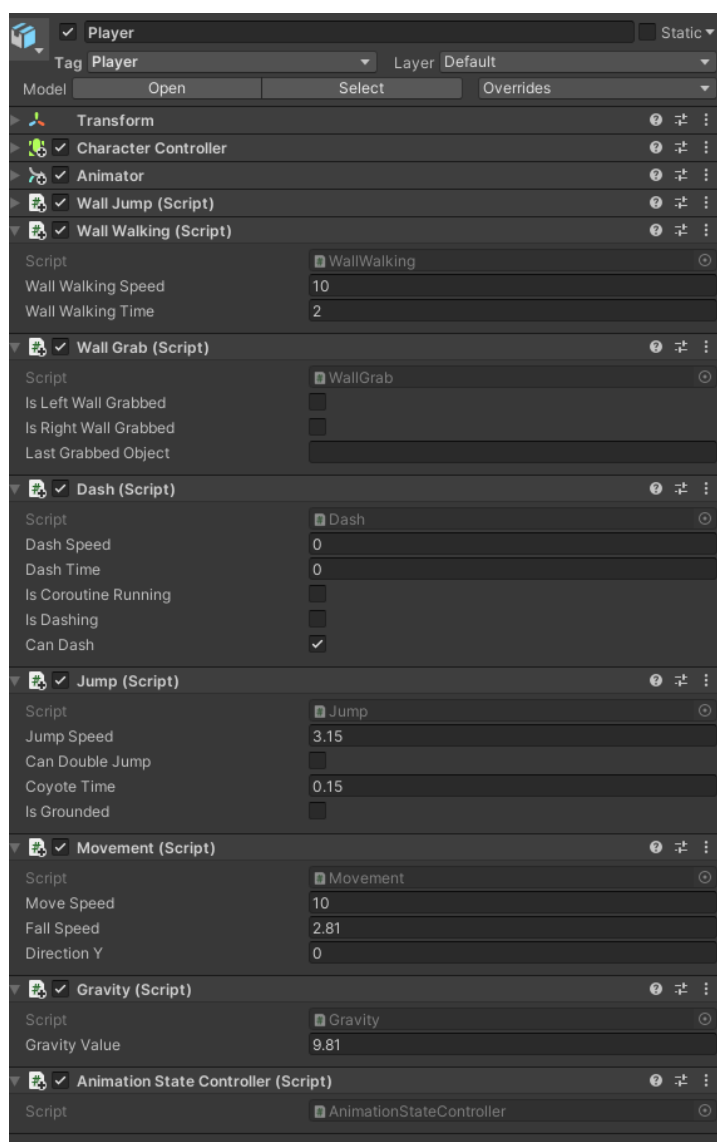
- Bohater
- Elementy otoczenia
- Zagrożenia gracza
- Checkpointy
- Obiekty możliwe do zebrania przez gracza.



Rys 2.4 Zrzut ekranu z gry (źródło: Własne)

### 2.4.1 Skrypty bohatera

Mechaniki poruszania się bohaterem zostały zaimplementowane w projekcie na bazie komponentu „Character Controller”, natomiast głównym skryptem kontrolującym mechanikę poruszania się jest „Movement”. Dodatkowym skryptem odnoszącym się do mechaniki poruszania się jest skrypt „Gravity”, symulujący grawitację oraz skrypt „Jump”, obsługujący mechanikę skakania. Do animowania bohatera został wykorzystany komponent „Animator”, a skryptem zajmującym się maszyną stanów animacji to „Animation State Controller”. Skrypty wykorzystane do obsługi mechaniki poruszania się po ścianach to „Wall Grab”, „Wall Jump” i „Wall Walking”. Obsługą mechaniki dashowania zajmuje się skrypt „Dash”.



Rys 2.4.1.1 Zrzut ekranu z silnika Unity, przegląd komponentów i skryptów Obiektu Gry „Player” (źródło: Własne)



Gracz ma możliwość poruszania się bohater po przeszkodach w dwóch wymiarach. Podczas skoku zastosowana technika tzw. Czasu kojota [17], czyli umożliwienie dodatkowego czasu na skok po opuszczeniu platformy w celu lepszego wyważenia rozgrywki. Bohater gracza ma możliwość podwójnego skoku, a każda wspinaczka po platformie umożliwia na dodatkowy skok. Za każdym razem, gdy bohater wyląduje na ziemi, resetuje się możliwość podwójnego skoku. Bohater sterowany przez gracza, podczas wspinania się po platformach ma możliwość nagłego puszczenia się z platformy po wciśnięciu odpowiedniego przycisku. Dodatkowo bohater sterowany przez gracza ma możliwość przeskakiwania między ścianami w ruchu wertykalnym, dzięki czemu poziomy gry mogą być bardziej urozmaicone. Bohater kontrolowany przez gracza, ma ograniczony czas na wspinanie się po platformach, symbolizowany symbolem graficznym nad bohaterem, o zmiennych kolorach. Kolor symbolu, informuje gracza o czasie, który został na zakończenie wspinaczki. Po zakończeniu określonego czasu na wspinaczkę, bohater automatycznie przerywa wspinaczkę, odrywając się od krawędzi ściany. W czasie gdy bohater jest w powietrzu, gracz po wciśnięciu odpowiedniej kombinacji klawiszy ma możliwość uruchomienia mechaniki dashowania, czyli przemieszczania się poziomo w locie na dużą odległość. W czasie dashowania gracz ma możliwość na niszczenie niektórych elementów otoczenia, takich jak np. skrzynki.



Rys 2.4.1.2 Zrzut ekranu z gry, wspinaczka bohatera. (źródło: Własne)



Rys 2.4.1.3 Zrzut ekranu z gry, dashowanie bohatera. (źródło: Własne)

### Skrypt *Movement*

```
using UnityEngine;

public class Movement : MonoBehaviour {
    CharacterController _controller;
    Gravity _gravity;
    AttachPlayer _attachPlayer;

    [SerializeField]
    public float _moveSpeed = 10f;
    [SerializeField]
    private float _fallSpeed = 2.81f;
    public float _directionY;

    void Start() {
        _controller = GetComponent<CharacterController>();
        _gravity = GetComponent<Gravity>();
        _attachPlayer = GameObject.FindGameObjectWithTag("MovingPlatform").GetComponent<AttachPlayer>();
        _directionY = 0f;
        //Animation bug fix
        this.enabled = false;
        this.enabled = true;
        //Animation bug fix
    }

    void Update() {
        Movements();
        Physics.IgnoreLayerCollision(0, 7);
    }

    void Movements() {
        if (!_attachPlayer.isOnMovingPlatform) {
            if (_directionY < -_fallSpeed)
                _directionY = -_fallSpeed;
        }

        float horizontalInput = Input.GetAxisRaw("Horizontal");
        Vector3 direction = new Vector3(horizontalInput, 0f, 0f);
        direction.y = _directionY;
        Vector3 movement = direction * Time.deltaTime * _moveSpeed;
        _controller.Move(movement);
    }
}
```

Rys 2.4.1.4 Kod źródłowy skryptu *Movement*. (źródło: Własne)

Skrypt *Movement* określa aktualny kierunek ruchu bohatera wraz z prędkością ruchu i prędkością spadania bohatera, jeżeli bohater nie znajduje się na poruszającej się platformie.

### Skrypt *Gravity*

```
using UnityEngine;

public class Gravity : MonoBehaviour {
    Movement _movement;
    CharacterController _controller;
    [SerializeField]
    private float _gravityValue = 9.81f;

    void Start() {
        _gravityValue = 9.81f;
        _movement = GetComponent<Movement>();
        _controller = GetComponent<CharacterController>();
    }

    void Update() {
        Gravitation();
    }

    private void Gravitation() {
        _movement._directionY -= _gravityValue * Time.deltaTime;
    }
}
```

Rys 2.4.1.5 Kod źródłowy skryptu *Gravity*. (źródło: Własne)

Skrypt *Gravity* zawiera implementację grawitacji.

## Skrypt Jump

```
using UnityEngine;

@ Unity Script (1 asset reference) | 6 references
public class Jump : MonoBehaviour {
    Movement _movement;
    CharacterController _controller;
    Gravity _gravity;
    WallGrab _wallGrab;
    Dash _dash;
    [SerializeField]
    public float _jumpSpeed = 3.15f;
    public bool _canDoubleJump = false;
    public float coyoteTime = 0.15f;
    public bool isGrounded;

    @ Unity Message | 0 references
    void Start() {
        _movement = GetComponent<Movement>();
        _controller = GetComponent<CharacterController>();
        _gravity = GetComponent<Gravity>();
        _wallGrab = GetComponent<WallGrab>();
        _dash = GetComponent<Dash>();
    }

    @ Unity Message | 0 references
    void Update() {
        IsOnGround();
        Jumping();
    }

    @ Unity Message | 0 references
    void OnDisable() {
        _canDoubleJump = true;
    }

    @ Unity Message | 0 references
    private void FixedUpdate() {
        InvokeRepeating("checkScript", 1.0f, 1.0f);
    }

    1 reference
    void Jumping() {
        if (coyoteTime > 0) {
            _canDoubleJump = true;
            GetComponent<Dash>().isCoroutineRunning = false;
            if (Input.GetButtonDown("Jump")) {
                _movement._directionY = _jumpSpeed;
            }
        } else {
            if (Input.GetButtonDown("Jump") && _canDoubleJump) {
                _movement._directionY = _jumpSpeed;
                _canDoubleJump = false;
            }
        }
    }

    1 reference
    void IsOnGround() {
        if (_controller.isGrounded) {
            coyoteTime = 0.15f;
            _wallGrab.enabled = true;
            _dash.canDash = true;
            _movement._directionY = 0.0001f;
            _wallGrab.LastGrabbedObject = "RESET GROUND";
        } else {
            coyoteTime -= Time.deltaTime;
        }
    }

    0 references
    void checkScript() {
        if (_gravity.enabled)
            this.enabled = true;
        else {
            this.enabled = false;
            _movement._directionY = 0;
        }
    }
}
```

Rys 2.4.1.6 Kod źródłowy skryptu *Jump*. (źródło: Własne)

Skrypt Jump zawiera implementację mechaniki skakania bohatera. Bohater ma możliwość pojedynczego lub podwójnego skoku. Gracz po opuszczeniu platformy, ma szansę na wykonanie skoku po utracie kontaktu z platformą, dzięki kompensacji czasu reakcji przy wykorzystaniu zmiennej coyoteTime. Po wykonaniu podwójnego skoku gracz traci możliwość skakania bohatera momentu wylądowania bohatera na ziemi.

### Skrypt Dash

```
1 using System.Collections;
2 using UnityEngine;
3 [Unity Script (1 asset reference)] 13 references
4 public class Dash : MonoBehaviour {
5     Gravity _gravity;
6     Movement _movement;
7     WallGrab _wallGrab;
8     CharacterController _controller;
9     [SerializeField]
10     private float dashSpeed;
11     [SerializeField]
12     private float dashTime;
13     public bool isCoroutineRunning = false;
14     public bool isDashing = false;
15     public bool canDash = true;
16     [Unity Message] 0 references
17     void Start() {
18         _gravity = GetComponent<Gravity>();
19         _movement = GetComponent<Movement>();
20         _controller = GetComponent<CharacterController>();
21         _wallGrab = GetComponent<WallGrab>();
22         dashTime = 0.4f;
23         dashSpeed = 150f;
24     }
25     [Unity Message] 0 references
26     void Update() {
27         DashingHandler();
28     }
29     1 reference
30     void DashingHandler() {
31         //dash right
32         if (Input.GetKeyDown(KeyCode.C) && Input.GetKey(KeyCode.RightArrow) && !isCoroutineRunning && !_controller.isGrounded && canDash) {
33             isCoroutineRunning = true;
34             canDash = false;
35             StartCoroutine(Dashing(new Vector3(0.1f, 0, 0), dashTime, dashSpeed));
36         }
37         //dash left
38         if (Input.GetKeyDown(KeyCode.C) && Input.GetKey(KeyCode.LeftArrow) && !isCoroutineRunning && !_controller.isGrounded && canDash) {
39             isCoroutineRunning = true;
40             canDash = false;
41             StartCoroutine(Dashing(new Vector3(-0.1f, 0, 0), dashTime, dashSpeed));
42         }
43     }
44     2 references
45     IEnumerator Dashing(Vector3 destination, float dashTime, float dashSpeed) {
46         while (dashTime > 0) {
47             isDashing = true;
48             _gravity.enabled = false;
49             _movement.enabled = false;
50             _wallGrab.enabled = false;
51             _controller.Move(new Vector3(destination.x * dashSpeed * Time.deltaTime, destination.y * dashSpeed * Time.deltaTime, destination.z * dashSpeed * Time.deltaTime));
52             yield return new WaitForSeconds(0.01f);
53             dashTime -= Time.deltaTime;
54         }
55         isDashing = false;
56         _gravity.enabled = true;
57         _movement.enabled = true;
58         _wallGrab.enabled = true;
59     }
60 }
```

Rys 2.4.1.6 Kod źródłowy skryptu Dash. (źródło: Własne)

Skrypt Dash zawiera implementację mechaniki dashowania, czyli szybkiego poruszania się bohaterem w locie w ruchu horyzontalnym. Podczas wykonywania takiego ruchu, wyłączane są inne skrypty, w celu uniknięcia kolidujących ze sobą mechanik

## Skrypt WallGrab

```
using UnityEngine;

public class WallGrab : MonoBehaviour {
    Movement _movement;
    CharacterController _controller;
    Gravity _gravity;
    Jump _jump;
    WallWalking _wallWalking;
    Dash _dash;
    public bool isLeftWallGrabbed;
    public bool isRightWallGrabbed;
    public string LastGrabbedObject;
    private float LeftArrowTapTime;
    private float RightArrowTapTime;
    private const float DOUBLE_TAP_TIME = 0.2f;

    void Start() {
        _movement = GetComponent<Movement>();
        _controller = GetComponent<CharacterController>();
        _gravity = GetComponent<Gravity>();
        _jump = GetComponent<Jump>();
        _wallWalking = GetComponent<WallWalking>();
        _dash = GetComponent<Dash>();
    }

    void Update() {
        WallGrabbing();
    }

    void WallGrabbing() {
        Vector3 rayOrigin = transform.position + new Vector3(0, 1f, 0);
        Vector3 rayDirection = new Vector3(0.5f, 0, 0);
        Debug.DrawRay(rayOrigin, rayDirection, Color.red);
        Debug.DrawRay(rayOrigin, -rayDirection, Color.red);

        Ray rayRight = new Ray(rayOrigin, rayDirection);
        Ray rayLeft = new Ray(rayOrigin, -rayDirection);
        if (Physics.Raycast(rayRight, out RaycastHit raycastHit, rayDirection.x) && raycastHit.transform.tag == "canGrab" && LastGrabbedObject != raycastHit.transform.name) {
            if (Input.GetKey(KeyCode.RightArrow)) {
                transform.rotation = Quaternion.Euler(0, 90, 0);
                LastGrabbedObject = raycastHit.transform.name;
                isRightWallGrabbed = true;
                _gravity.enabled = false;
                _movement.enabled = false;
                _wallWalking.enabled = true;
                _dash.enabled = false;
            }
        }
        if (Physics.Raycast(rayLeft, out RaycastHit raycastHit, rayDirection.x) && raycastHit.transform.tag == "canGrab" && LastGrabbedObject != raycastHit.transform.name) {
            if (Input.GetKey(KeyCode.LeftArrow)) {
                transform.rotation = Quaternion.Euler(0, -90, 0);
                LastGrabbedObject = raycastHit.transform.name;
                isLeftWallGrabbed = true;
                _gravity.enabled = false;
                _movement.enabled = false;
                _wallWalking.enabled = true;
                _dash.enabled = false;
            }
        }
    }
}
```

Rys 2.4.1.7 Kod źródłowy skryptu WallGrab. (źródło: Własne)

Skrypt *WallGrab* zawiera implementację mechaniki chwytania się ścian przez bohatera. W tym celu zostały użyte promienie, których kolizja z obiektem o oznakowaniu „canGrab” umożliwia bohaterowi zatrzymanie się na danym obiekcie

## Skrypt WallJump

```
1 using UnityEngine;
2
3 [UnityScript (1 asset reference) | 4 references]
4 public class WallJump : MonoBehaviour {
5     WallWalking _wallWalking;
6     Movement _movement;
7     CharacterController _controller;
8     Gravity _gravity;
9     WallGrab _wallGrab;
10    Jump _jump;
11    [UnityMessage | 0 references]
12    void Start() {
13        _wallWalking = GetComponent<WallWalking>();
14        _movement = GetComponent<Movement>();
15        _controller = GetComponent<CharacterController>();
16        _gravity = GetComponent<Gravity>();
17        _wallGrab = GetComponent<WallGrab>();
18        _jump = GetComponent<Jump>();
19    }
20
21    [UnityMessage | 0 references]
22    void Update() {
23        if (_wallGrab.isLeftWallGrabbed && Input.GetButton("Jump")) {
24            GetComponent<Dash>().isCoroutineRunning = false;
25            _gravity.enabled = true;
26            _movement.enabled = true;
27            _movement.directionY = _jump._jumpSpeed;
28            _wallGrab.isLeftWallGrabbed = false;
29            this.enabled = false;
30            _wallWalking.enabled = false;
31        }
32        if (_wallGrab.isRightWallGrabbed && Input.GetButton("Jump")) {
33            GetComponent<Dash>().isCoroutineRunning = false;
34            _gravity.enabled = true;
35            _movement.enabled = true;
36            _movement.directionY = _jump._jumpSpeed;
37            _wallGrab.isRightWallGrabbed = false;
38            this.enabled = false;
39            _wallWalking.enabled = false;
40        }
41    }
42 }
```

Rys 2.4.1.8 Kod źródłowy skryptu *WallJump*. (źródło: Własne)

Skrypt *WallJump* zawiera implementację mechaniki odskakiwania od ściany przez bohatera.

## Skrypt WallWalking

```

1 using UnityEngine;
2 @ Unity Script (1 asset reference) | 7 references
3 public class WallWalking : MonoBehaviour {
4     CharacterController _controller;
5     WallGrab _wallGrab;
6     WallJump _wallJump;
7     Gravity _gravity;
8     Jump _jump;
9     Movement _movement;
10     Dash _dash;
11     [SerializeField]
12     private float wallWalkingSpeed = 10f;
13     [SerializeField]
14     public float wallWalkingTime = 2f;
15     @ Unity Message | 0 references
16     void Start() {
17         _wallGrab = GetComponent<WallGrab>();
18         _controller = GetComponent<CharacterController>();
19         _gravity = GetComponent<Gravity>();
20         _jump = GetComponent<Jump>();
21         _movement = GetComponent<Movement>();
22         _dash = GetComponent<Dash>();
23         this.enabled = false;
24     }
25     @ Unity Message | 0 references
26     void Update() {
27         if (wallWalkingTime > 0) {
28             WallWalkingScript();
29             wallWalkingTime -= Time.deltaTime;
30             if (Input.GetKeyDown(KeyCode.X))
31                 wallWalkingTime = 0;
32         } else {
33             this.enabled = false;
34             _wallGrab.enabled = false;
35             _wallGrab.isRightWallGrabbed = false;
36             _wallGrab.isLeftWallGrabbed = false;
37             _gravity.enabled = true;
38             _movement.enabled = true;
39         }
40     }
41     @ Unity Message | 0 references
42     private void OnEnable() {
43         _wallJump = GetComponent<WallJump>();
44         _wallJump.enabled = true;
45     }
46     @ Unity Message | 0 references
47     private void OnDisable() {
48         wallWalkingTime = 2f;
49         _wallJump.enabled = false;
50         _dash.enabled = true;
51     }
52     1 reference
53     void WallWalkingScript() {
54         Vector3 rayDirection = new Vector3(0.8f, 0, 0);
55         RaycastHit raycastHit;
56     }
57     1 reference
58     void WallWalkingScript() {
59         Vector3 rayDirection = new Vector3(0.8f, 0, 0);
60         RaycastHit raycastHit;
61         Ray rayUpRight = new Ray(new Vector3(transform.position.x, transform.position.y + 2f, transform.position.z), rayDirection);
62         Ray rayDownRight = new Ray(new Vector3(transform.position.x, transform.position.y, transform.position.z), rayDirection);
63         Ray rayUpLeft = new Ray(new Vector3(transform.position.x, transform.position.y + 2f, transform.position.z), -rayDirection);
64         Ray rayDownLeft = new Ray(new Vector3(transform.position.x, transform.position.y, transform.position.z), -rayDirection);
65         //up right
66         Debug.DrawRay(new Vector3(transform.position.x, transform.position.y + 2f, transform.position.z), rayDirection, Color.cyan);
67         //down right
68         Debug.DrawRay(new Vector3(transform.position.x, transform.position.y, transform.position.z), rayDirection, Color.green);
69         //up left
70         Debug.DrawRay(new Vector3(transform.position.x, transform.position.y + 2f, transform.position.z), -rayDirection, Color.black);
71         //down left
72         Debug.DrawRay(new Vector3(transform.position.x, transform.position.y, transform.position.z), -rayDirection, Color.yellow);
73         if ((Physics.Raycast(rayUpLeft, out raycastHit, rayDirection.x)) || (Physics.Raycast(rayDownLeft, out raycastHit, rayDirection.x))) {
74             if (Physics.Raycast(rayUpLeft, out raycastHit, rayDirection.x) && (Physics.Raycast(rayDownLeft, out raycastHit, rayDirection.x))) {
75                 movementControls();
76             } else if (!Physics.Raycast(rayUpLeft, out raycastHit, rayDirection.x)) {
77                 wallWalkingControlsDown();
78             } else {
79                 wallWalkingControlsUp();
80             }
81         }
82         if ((Physics.Raycast(rayUpRight, out raycastHit, rayDirection.x)) || (Physics.Raycast(rayDownRight, out raycastHit, rayDirection.x))) {
83             if (Physics.Raycast(rayUpRight, out raycastHit, rayDirection.x) && (Physics.Raycast(rayDownRight, out raycastHit, rayDirection.x))) {
84                 movementControls();
85             } else if (!Physics.Raycast(rayUpRight, out raycastHit, rayDirection.x)) {
86                 wallWalkingControlsDown();
87             } else {
88                 wallWalkingControlsUp();
89             }
90         }
91     }
92     3 references
93     void wallWalkingControlsUp() {
94         Vector3 directionup = new Vector3(0f, 1, 0f);
95         if (Input.GetKey(KeyCode.UpArrow)) {
96             _controller.Move(directionup * Time.deltaTime * wallWalkingSpeed);
97         }
98     }
99     3 references
100     void wallWalkingControlsDown() {
101         Vector3 directiondown = new Vector3(0f, -1, 0f);
102         if (Input.GetKey(KeyCode.DownArrow)) {
103             _controller.Move(directiondown * Time.deltaTime * wallWalkingSpeed);
104         }
105     }
106     2 references
107     void movementControls() {
108         wallWalkingControlsUp();
109         wallWalkingControlsDown();
110     }
111 }

```

Rys 2.4.1.9 Kod źródłowy skryptu WallWalking. (źródło: Własne)

### *Skrypt WallWalking*

Skrypt *WallJump* zawiera implementację mechaniki poruszania się bohatera po ścianach. Bohater ma możliwość poruszania się po obiekcie gdy dwa promienie, znajdujące się odpowiednio na górze oraz dole modelu 3D są w trakcie kolizji z obiektem. Dodatkowo dwie zmienne *wallWalkingSpeed* oraz *wallWalkingTime* odpowiadają za prędkość poruszania się oraz czas który jest dostępny do wykorzystania tej mechaniki. Gdy zmienna *wallWalkingTime* jest równa zero, bohater traci możliwość poruszania się po ścianie.

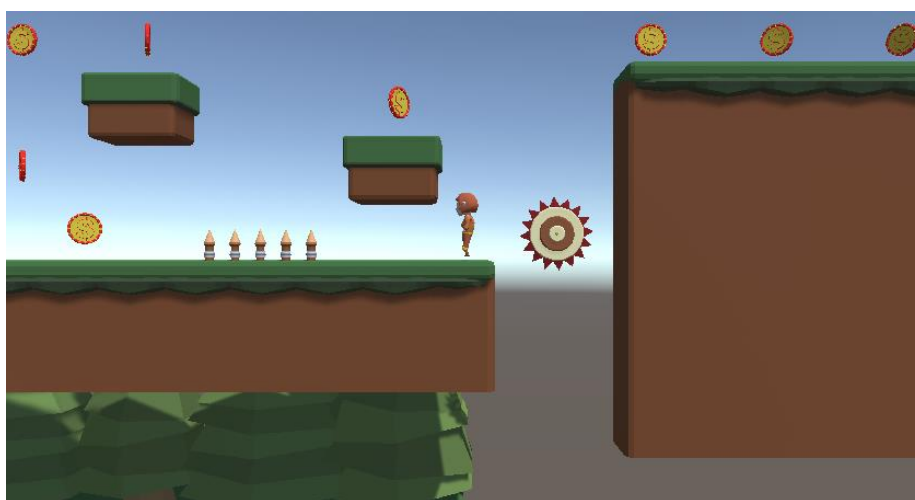


## 2.4.2 Skrypty Przeciwników

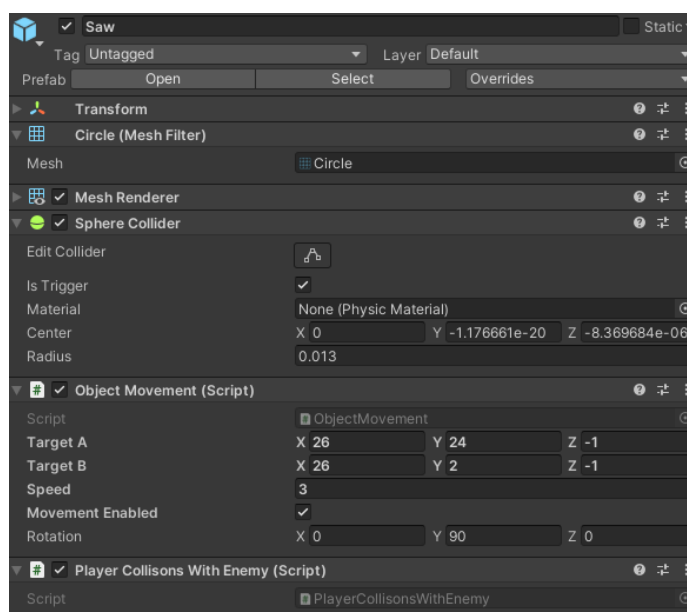
W projekcie zostały zaimplementowane trzy rodzaje przeciwników bohatera:

- Piła
- Armata
- Szpikulce

Piła jest to obracający się obiekt gry, porusza się on po z góry określonej ścieżce. Jeżeli zachodzi kolizja między bohaterem a piłą, bohater otrzymuje jeden punkt obrażeń oraz tymczasową niewrażliwość na inne zagrożenia, które mogą zadać jeden punkt obrażeń.



Rys 2.4.2.1 Zrzut ekranu z gry, piłą. (źródło: Własne)

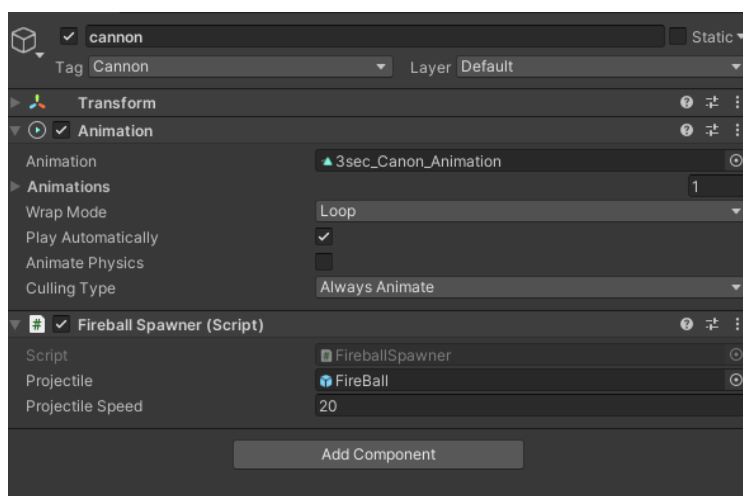


Rys 2.4.2.2 Zrzut ekranu z gry, przegląd komponentów i skryptów obiektu „Piła”. (źródło: Własne)

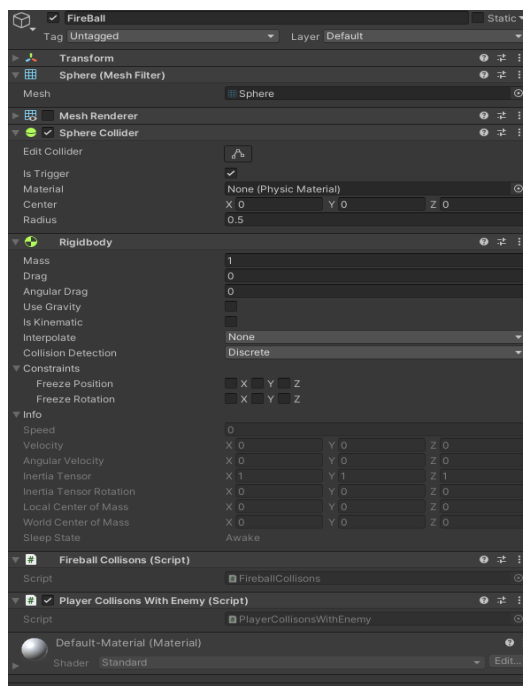
Armata jest to przeciwnik wysyłający ogniste kule na długie odległości, które po kontakcie z bohaterem lub ścianą zostają zniszczone. Gdy pocisk armaty i bohater zachodzą razem ze sobą w kolizję, gracz traci jeden punkt obrażeń. Model pocisku armaty jest generowany proceduralnie za pomocą „Particle System”. Gdy odpowiednia klatka animacji armaty jest odtwarzana, w tym samym momencie wykonywany jest skrypt instancjonujący prefab pocisku, który posyłany jest w stronę, w którą zwrócona jest armata.



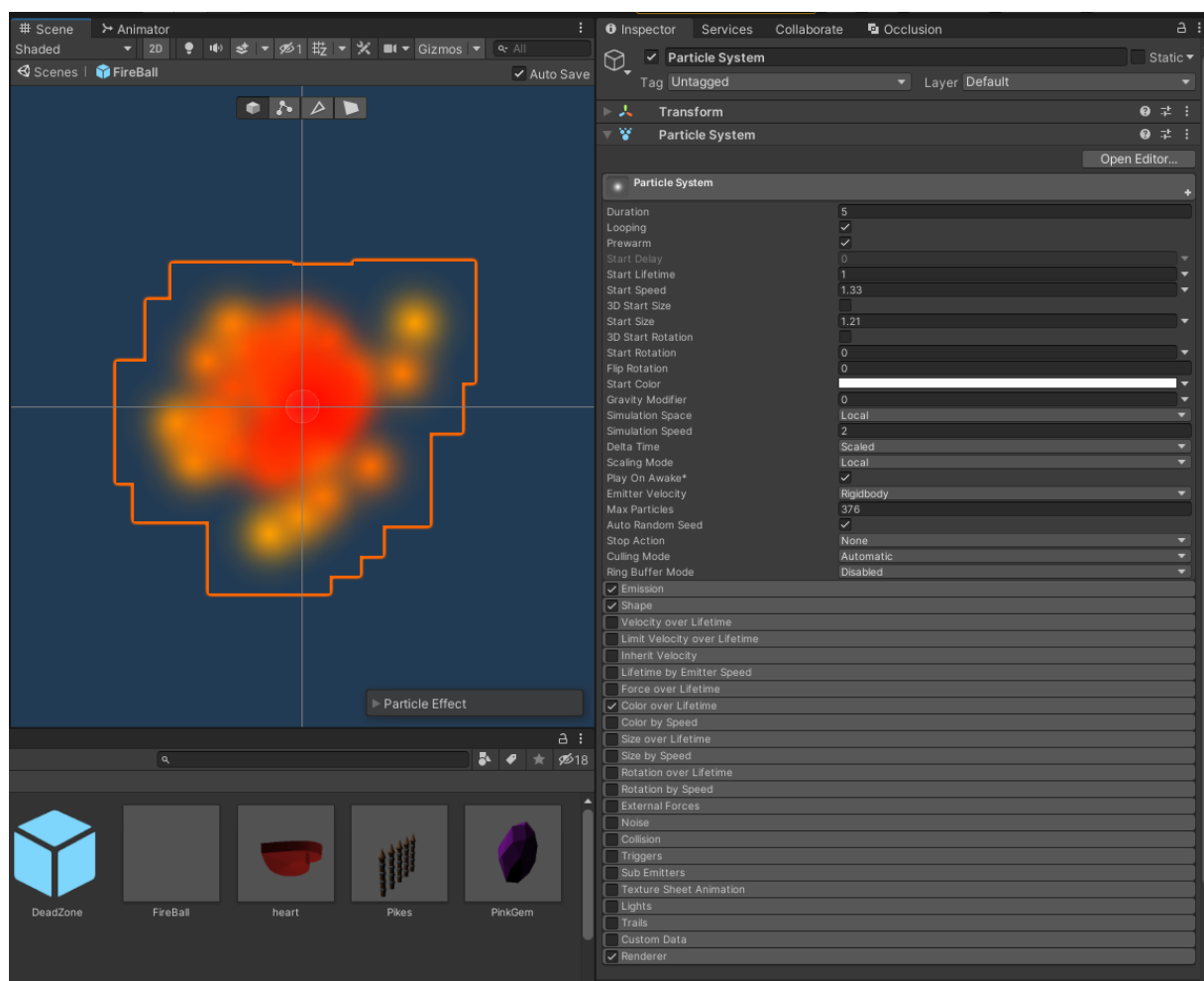
Rys 2.4.2.3 Zrzut ekranu z gry, wygląd armaty razem z pociskiem (źródło: Własne)



Rys 2.4.2.4 Zrzut ekranu z silnika Unity, komponenty prefabu armaty (źródło: Własne)



Rys 2.4.2.5 Zrzut ekranu z silnika Unity, komponenty prefabu pocisku armaty (źródło: Własne)



Rys 2.4.2.6 Zrzut ekranu z silnika Unity, komponenty prefabu pocisku armaty (źródło: Własne)

### Skrypt *FireballSpawner*

```
using UnityEngine;

@ Unity Script (1 asset reference) | 0 references
public class FireballSpawner : MonoBehaviour {
    [SerializeField] GameObject projectile;
    [SerializeField] float projectileSpeed;
    Animation _animation;

    @ Unity Message | 0 references
    private void Start() {
        _animation = GetComponent<Animation>();
    }

    0 references
    void SpawnFireball() {
        GameObject fireball = Instantiate(projectile, this.gameObject.transform.GetChild(0)) as GameObject;
        Rigidbody rb = fireball.GetComponent<Rigidbody>();
        rb.velocity = transform.up * projectileSpeed;
    }
}
```

Rys 2.4.2.7 Kod źródłowy skryptu *FireballSpawner* (źródło: Własne)

Skrypt *FireballSpawner* zawiera mechanikę instancjonowania prefabu pocisku armaty, wraz z jego poruszaniem.

### Skrypt *FireballCollisons*

```
using UnityEngine;

public class FireballCollisons : MonoBehaviour {
    private void OnTriggerEnter(Collider other) {
        if (other.tag != "IgnoreCollison")
            Destroy(this.gameObject);
    }
}
```

Rys 2.4.2.8 Kod źródłowy skryptu *FireballCollisons* (źródło: Własne)

Skrypt *FireballCollisons* odpowiada za niszczenie pocisku armaty po kolizji z obiektem nieposiadającym oznaczenia „IgnoreCollison”.

### Skrypt *PlayerCollisionsWithEnemy*

```
using UnityEngine;

public class PlayerCollisionsWithEnemy : MonoBehaviour {
    GameObject _player;
    GameObject _model;
    GameManagerScript _gm;

    private void Start() {
        _player = GameObject.FindGameObjectWithTag("Player");
        _model = GameObject.Find("Model");
        _gm = GameObject.FindGameObjectWithTag("GM").GetComponent<GameManagerScript>();
    }

    10 Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {
        if (other.CompareTag("Player") && !_gm._playerHited) {
            _gm._playerHited = true;
            _gm.hitpoints--;
        }
    }

    10 Unity Message | 0 references
    private void OnTriggerStay(Collider other) {
        if (other.CompareTag("Player") && !_gm._playerHited) {
            _gm._playerHited = true;
            _gm.hitpoints--;
        }
    }
}
```

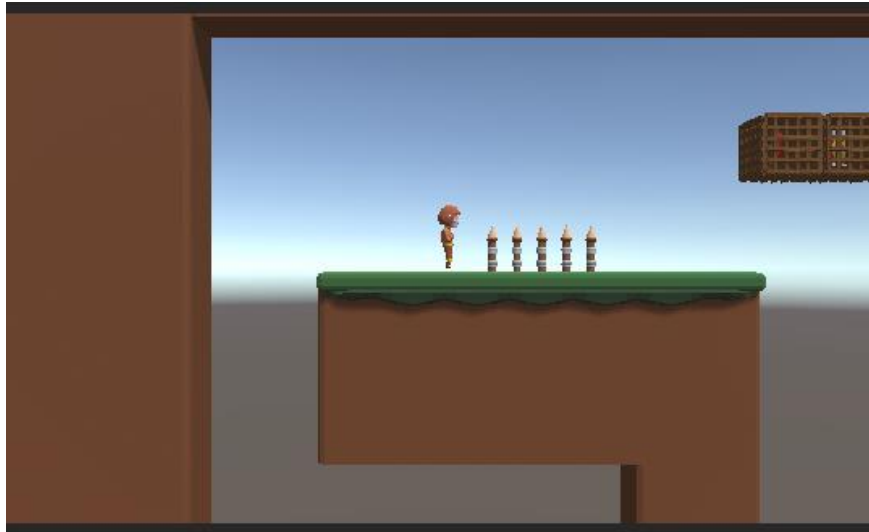
Rys 2.4.2.9 Kod źródłowy *PlayerCollisionsWithEnemy* (źródło: Własne)

Skrypt *PlayerCollisionsWithEnemy* rozpatruje kolizje między przeciwnikami a bohaterem, a w dalszej części odejmowanie punktów obrażeń.

Szpikulce to obiekty występujące w grze w dwóch formach:

- Animowane
- Statyczne

Obydwie wersje szpikulców zaprojektowane zostały tak aby po kolizji z bohaterem, odejmowały jeden punkt obrażeń z puli dostępnych dla bohatera. Animowana wersja tego zagrożenia, wysuwa się z podłoża lub ściany, i po czasie który zostanie podany w kodzie źródłowym wsuwa się z powrotem na miejsce pierwotne. Wersja statyczna nie posiada możliwości ruchu.



Rys 2.4.2.10 Zrzut ekranu z gry, wygląd animowanej wersji szpikulców (źródło: Własne)



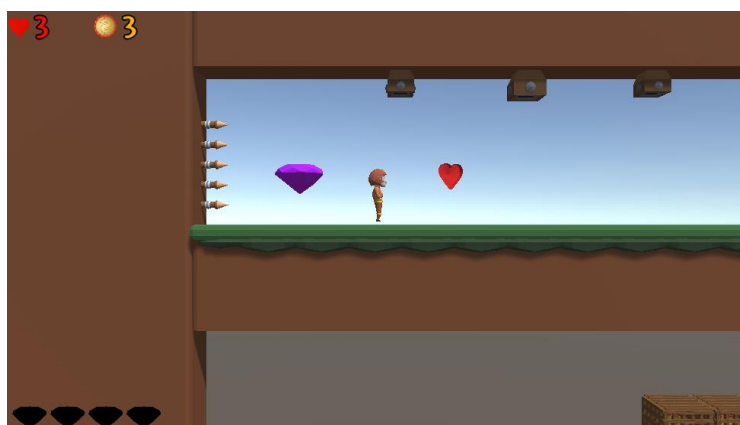
Rys 2.4.2.11 Zrzut ekranu z gry, wygląd statycznej wersji szpikulców (źródło: Własne)

### 2.4.3 Skrypty obiektów możliwych do zebrania

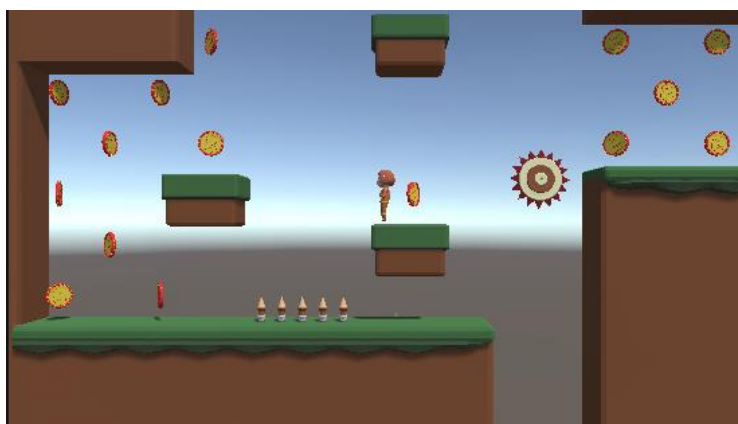
W grze występują obiekty możliwe do zebrania przez bohatera:

- Szmaragdy
- Monety
- Serca

Szmaragdy to obiekty występujące w poziomie gry w trudno dostępnych miejscach, których zdobycie jest opcjonalne do ukończenia poziomu. Szmaragdy występują w czterech kolorach, gracz jest informowany o postępach w ich zdobyciu za pomocą interfejsu wywołanego za pomocą klawisza „Tab”. Po ich zdobyciu zebrany szmaragd znika z planszy. Monety to obiekty występujące w dużej ilości w poziomie gry. Ich rolą jest wypełnienie wizualne planszy. Gracz ma możliwość w każdej próbie pokonania poziomu, na skompletowaniu wszystkich monet na planszy. Serca to obiekty, które po zebraniu przez gracza dodają dodatkowe „życia”, czyli możliwe próby pokonania poziomu. Po włączeniu gry gracz ma do dyspozycji trzy „życia”, dlatego każda dodatkowa próba przejściu poziomu, jest dla gracza bardzo cenna.



Rys 2.4.3.1 Zrzut ekranu z gry, wygląd szmaragdu oraz serca, wraz z interfejsem graficznym (źródło: Własne)



Rys 2.4.3.2 Zrzut ekranu z gry, wygląd monet (źródło: Własne)

### Skrypt *CoinScript*

```
using UnityEngine;
@ Unity Script (1 asset reference) | 0 references
public class CoinScript : MonoBehaviour {
    GameObject GameManager;
    @ Unity Message | 0 references
    private void Start() {
        transform.Rotate(0, Random.Range(-90, 90), 0, 0);
    }
    @ Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {
        if (other.CompareTag("Player")) {
            GameObject.Find("GameManagerHelper").GetComponent<GameManagerScript>().coinPoints++;
            Destroy(this.gameObject);
        }
    }
}
```

Rys 2.4.3.3 Kod źródłowy skryptu *CoinScript* (źródło: Własne)

Skrypt *CoinScript* zawiera implementację mechaniki kolekcjonowania żetonów rozmieszczonych na planszy.

### Skrypt *HeartScript*

```
using UnityEngine;
@ Unity Script (1 asset reference) | 0 references
public class HeartScript : MonoBehaviour {
    @ Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {
        GameObject.Find("GameManagerHelper").GetComponent<GameManagerScript>().lifePoints++;
        Destroy(this.gameObject);
    }
}
```

Rys 2.4.3.4 Kod źródłowy skryptu *CoinScript* (źródło: Własne)

Skrypt *HeartScript* zawiera implementację mechaniki kolekcjonowania serc rozmieszczonych na planszy, których otrzymanie będzie skutkowało się zwiększeniem możliwych prób przejścia poziomu.



## Skrypt *GemScript*

```
using UnityEngine;
using UnityEngine.UI;
// Unity Script (1 asset reference) | 0 references
public class GemScript : MonoBehaviour {

    [SerializeField] Texture pinkGemTexture;
    [SerializeField] Texture blueGemTexture;
    [SerializeField] Texture yellowGemTexture;
    [SerializeField] Texture greenGemTexture;
    //pink=1,yellow=2,green=3,blue=4
    int gemColour = 0;
    // Unity Message | 0 references
    private void Start() {
        GemLogic();
    }

    // Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {
        switch (gemColour) {
            case 1:
                GameObject.Find("GameManagerHelper").GetComponent<GameManagerScript>().pinkGemTaken = true;
                GameObject.Find("PinkGemImage").GetComponent<RawImage>().texture = pinkGemTexture;
                GameObject.Find("Canvas").GetComponent<GameUiTextScript>().StartCoroutine("ShowGems");
                Destroy(this.gameObject);
                break;
            case 2:
                GameObject.Find("GameManagerHelper").GetComponent<GameManagerScript>().yellowGemTaken = true;
                GameObject.Find("BlueGemImage").GetComponent<RawImage>().texture = blueGemTexture;
                Destroy(this.gameObject);
                break;
            case 3:
                GameObject.Find("GameManagerHelper").GetComponent<GameManagerScript>().greenGemTaken = true;
                GameObject.Find("YellowGemImage").GetComponent<RawImage>().texture = yellowGemTexture;
                Destroy(this.gameObject);
                break;
            case 4:
                GameObject.Find("GameManagerHelper").GetComponent<GameManagerScript>().blueGemTaken = true;
                GameObject.Find("GreenGemImage").GetComponent<RawImage>().texture = greenGemTexture;
                Destroy(this.gameObject);
                break;
        }
    }

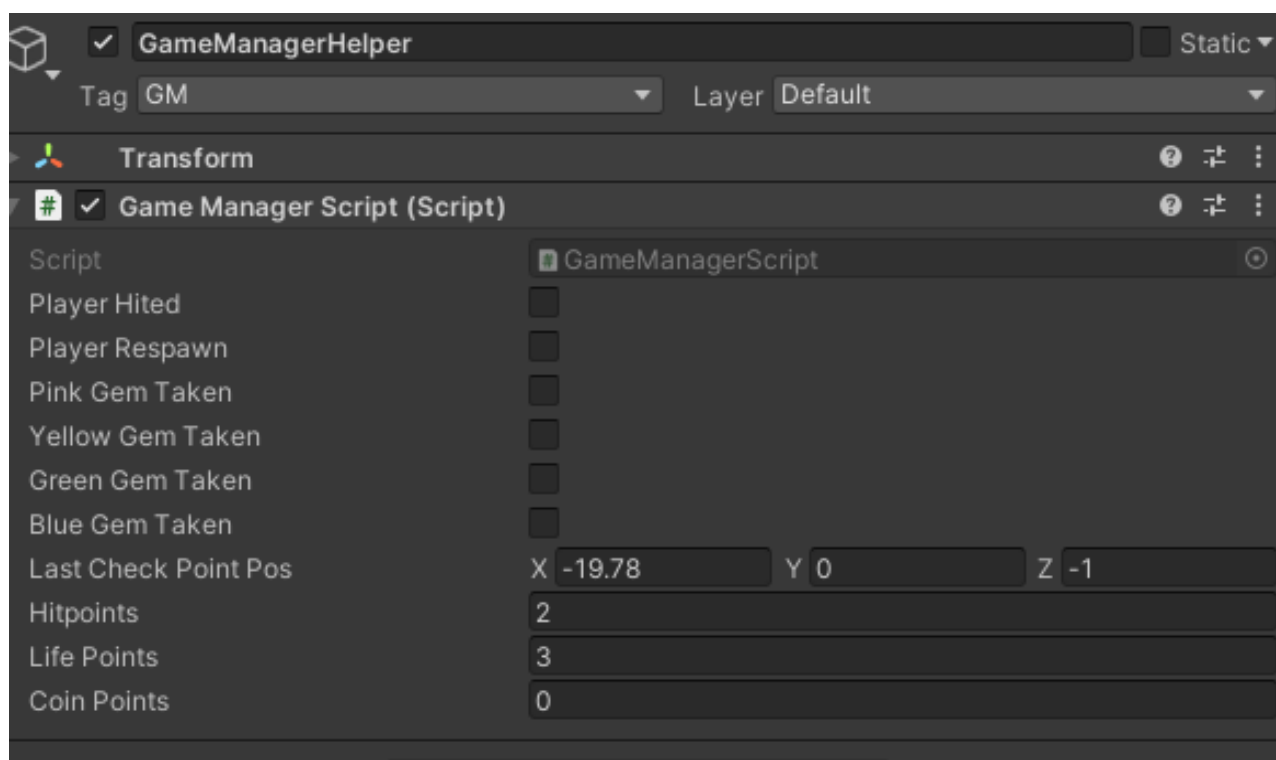
    // 1 reference
    void GemLogic() {
        switch (this.gameObject.name) {
            case "PinkGem":
                gemColour = 1;
                break;
            case "YellowGem":
                gemColour = 2;
                break;
            case "GreenGem":
                gemColour = 3;
                break;
            case "GlueGem":
                gemColour = 4;
                break;
        }
    }
}
```

Rys 2.4.3.5 Kod źródłowy skryptu *CoinScript* (źródło: Własne)

Skrypt *GemScript* zawiera implementację mechaniki kolekcjonowania szmaragdów, wraz ich reprezentacją graficzną w interfejsie użytkownika.

#### 2.4.4 Skrypt obiektu GameManagerHelper

Obiekt „GameManagerHelper” jest odpowiedzialny za przechowywanie skryptu „GameManagerScript”, którego zadaniem jest kontrola nad wszystkimi zmiennymi, które inne obiekty w danym poziomie wykorzystują. Skrypt odpowiedzialny jest za przechowywanie zmiennych ilości życia, ilości monet oraz punktów obrażeń. Dodatkowo w tym skrypcie przechowywane są zmienne informujące o podniesionych szmaragdach oraz o pozycji ostatnio aktywowanego checkpointu. W trakcie działania gry, w skrypcie „GameManagerScript” przechowywane są zmienne, które sprawdzają, czy bohaterowi aktualnie zadawane są obrażenia, oraz czy bohater kontrolowany przez gracza ma zostać przeniesiony do ostatnio aktywowanego checkpointu.



Rys 2.4.4.1 Zrzut ekranu z silnika Unity, przegląd komponentów i skryptu obiektu „GameManagerHelper”  
(źródło: Własne)

## Skrypt *GameManagerHelper*

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManagerScript : MonoBehaviour {
    GameObject _player;
    GameObject _canvas;
    GameManagerScript _gm;
    public bool _playerHited;
    public bool _playerRespawn;
    public bool pinkGemTaken = false;
    public bool yellowGemTaken = false;
    public bool greenGemTaken = false;
    public bool blueGemTaken = false;
    public Vector3 lastCheckPointPos;
    public int hitpoints;
    public int lifePoints;
    public int coinPoints;

    private void Start() {
        _player = GameObject.FindGameObjectWithTag("Player");
        _gm = GameObject.FindGameObjectWithTag("GM").GetComponent<GameManagerScript>();
        _canvas = GameObject.Find("Canvas");
        //fixing weird graphic bug
        StartCoroutine("HidePlayer");
    }

    private void Update() {
        HitpointsChecker();
        if (lifePoints == 0) {
            ReloadScene();
        }
    }

    public void RespawnPlayer() {
        _player.GetComponent<Dash>().isDashing = false;
        if (_canvas.GetComponent<GameUITextScript>().tabtoggle)
            _canvas.GetComponent<GameUITextScript>().StartCoroutine("ShowHideLife");
        lifePoints--;
        _playerHited = false;
        _playerRespawn = true;
        _player.SetActive(false);
        StartCoroutine(RespawnPlayerCoroutine(2));
        _player.GetComponent<Movement>().enabled = true;
        _player.GetComponent<Gravity>().enabled = true;
        hitpoints = 2;
    }

    public void HitpointsChecker() {
        if (hitpoints == 0) {
            RespawnPlayer();
        }
    }

    public void ReloadScene() {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        lifePoints = 2;
    }

    IEnumerator HidePlayer() {
        _player.transform.GetChild(1).GetComponent<SkinnedMeshRenderer>().enabled = false;
        _player.GetComponent<Movement>().enabled = false;
        yield return new WaitForSeconds(0.1f);
        _player.GetComponent<Movement>().enabled = true;
        yield return new WaitForSeconds(0.5f);
        _player.transform.GetChild(1).GetComponent<SkinnedMeshRenderer>().enabled = true;
    }

    IEnumerator RespawnPlayerCoroutine(float time) {
        yield return new WaitForSeconds(time);
        _player.transform.position = _gm.lastCheckPointPos;
        _player.SetActive(true);
        _playerRespawn = false;
    }
}
```

Rys 2.4.4.2 Kod źródłowy skryptu *GameManagerHelper* (źródło: Własne)

Skrypt *GameManagerHelper* odpowiada za kontrolę przebiegu rozgrywki na danym poziomie. Skrypt ten rozpatruje algorytm przywracania życia bohaterowi wraz ze sprawdzeniem otrzymywania obrażeń.

## 2.4.5 Skrypt obiektów typu Checkpoint

Checkpointy to obiekty, które po aktywacji przez bohatera wyznaczają miejsce, które po straceniu życia, czyli otrzymaniu dwóch punktów obrażeń lub natychmiastowej utracie próby, do wznowienia rozgrywki. Aktywacja obiektu następuje po kolizji z niewidzialnym colliderem, umieszczonym naprzeciwko modelu 3D obiektu. Jednocześnie może być aktywowany tylko jeden obiekt typu Checkpoint, każda kolejna aktywacja nadpisuje zmienną przechowywaną w skrypcie „GameManagerScript”, wyznaczającą miejsce, do którego zostanie przeniesiony bohater po utracie życia. Po aktywacji checkpointu, wyświetla się animacja uzyskania dostępu do tego obiektu.



Rys 2.4.5.1 Zrzut ekranu z gry, aktywacja checkpointu (źródło: Własne)

### Skrypt *CheckpointScript*

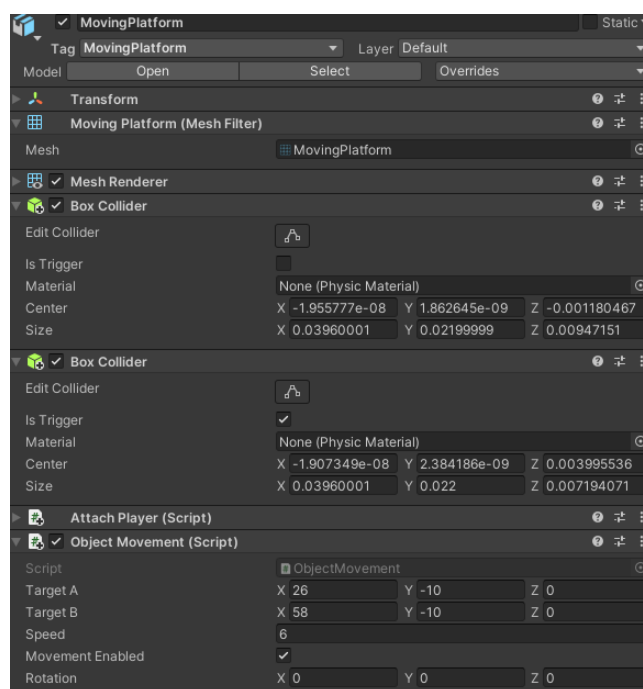
```
using UnityEngine;
[ScriptableReference]
public class CheckpointScript : MonoBehaviour {
    Animation _animation;
    GameManagerScript _gm;
    [SerializeField] private bool isCheckpointTriggerd = false;
    [SerializeField] public Vector3 _checkpointVector3;
    [UnityMessage]
    private void Start() {
        _animation = GetComponent<Animation>();
        _gm = GameObject.FindGameObjectWithTag("GM").GetComponent<GameManagerScript>();
    }
    [UnityMessage]
    private void OnTriggerEnter(Collider other) {
        if (other.CompareTag("Player")) {
            if (!isCheckpointTriggerd) {
                gameObject.transform.GetChild(1).gameObject.SetActive(true);
                _gm.lastCheckPointPos = _checkpointVector3;
                isCheckpointTriggerd = true;
                _animation.Play();
            }
        }
    }
}
```

Rys 2.4.5.2 Kod źródłowy skryptu *CheckpointScript* (źródło: Własne)

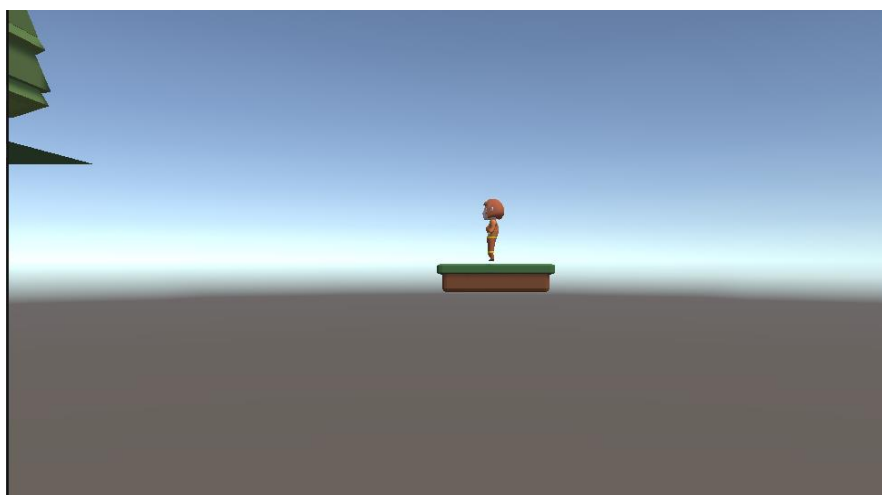
Skrypt *CheckpointScript* odpowiada za obsługę mechaniki zapisywania pozycji, do której po zakończeniu próby, gracz będzie miał możliwość kontynuowania rozgrywki.

## 2.4.6 Skrypt obiektów typu Moving Platform

Obiekty typu Moving Platform, to poruszające się między dwoma punktami platformy, po których bohater może się przemieszczać. W trakcie, gdy następuje kolizja między obiektami bohatera oraz MovingPlatform, gracz ma możliwość zeskoczenia z platformy. Za pomocą skryptu „Object Movement” istnieje możliwość personalizacji poruszania się platformy, a nawet zablokowania możliwości ruchu.



Rys 2.4.6.1 Zrzut ekranu z silnika Unity, przegląd komponentów i skryptu obiektu „MovingPlatform” (źródło: Własne)



Rys 2.4.6.2 Zrzut ekranu z gry, poruszanie się po ruchomej platformie (źródło: Własne)

## Skrypt *AttachPlayer*

```
using UnityEngine;
@ Unity Script (1 asset reference) | 2 references
public class AttachPlayer : MonoBehaviour {
    public GameObject Player;
    public bool isOnMovingPlatform = false;
    @ Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {
        if (other.gameObject == Player) {
            isOnMovingPlatform = true;
            Player.transform.parent = this.transform;
        }
    }
    @ Unity Message | 0 references
    private void OnTriggerExit(Collider other) {
        if (other.gameObject == Player) {
            isOnMovingPlatform = false;
            Player.transform.parent = null;
        }
    }
}
```

Rys 2.4.6.3 Kod źródłowy skryptu *AttachPlayer* (źródło: Własne)

Skrypt *AttachPlayer* na czas poruszania się bohatera po ruszającej się platformę, przypisuje pozycję obiektu bohatera do obiektu platformy.

## Skrypt *Object Movement*

```
using UnityEngine;
@ Unity Script (5 asset references) | 0 references
public class ObjectMovement : MonoBehaviour {
    [SerializeField] private Vector3 _targetA, _targetB;
    [SerializeField] private float _speed = 3.0f;
    [SerializeField] private bool _movementEnabled;
    [SerializeField] private Vector3 _rotation;
    private bool _switching = false;
    @ Unity Message | 0 references
    private void FixedUpdate() {
        transform.Rotate(_rotation * Time.deltaTime);
        if (_movementEnabled) {
            if (!_switching) {
                transform.position = Vector3.MoveTowards(transform.position, _targetB, _speed * Time.fixedDeltaTime);
            } else if (_switching) {
                transform.position = Vector3.MoveTowards(transform.position, _targetA, _speed * Time.fixedDeltaTime);
            }
            if (transform.position == _targetB) {
                _switching = true;
            }
            if (transform.position == _targetA) {
                _switching = false;
            }
        }
    }
}
```

Rys 2.4.6.5 Kod źródłowy skryptu *ObjectMovement* (źródło: Własne)

Skrypt *Object Movement* umożliwia poruszanie się obiektu między dwoma punktami, wraz z możliwością rotacji obiektu.

## 3 Opis technologii i dalszy rozwój projektu

### 3.1 Opis technologii

#### *Unity Engine*

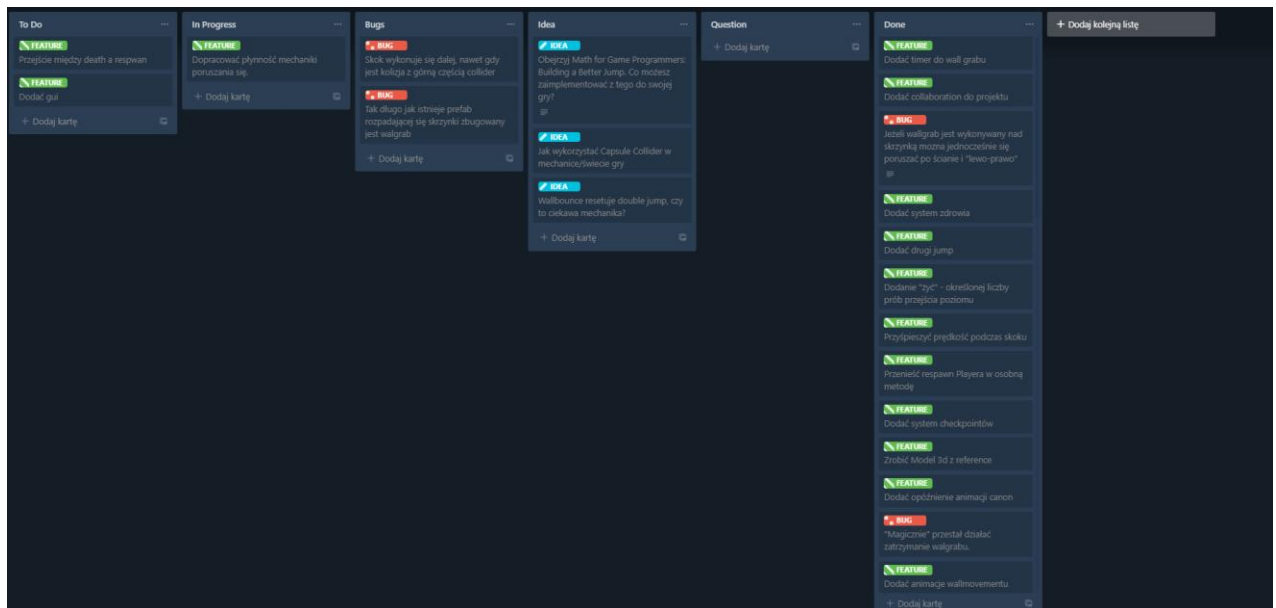
Unity Engine jest to silnik do tworzenia gier dwuwymiarowych lub trójwymiarowych autorstwa Unity Technologies. Silnik ten jest niezwykle popularny z uwagi na to, że głównym językiem do tworzenia skryptów to C#, chociaż możliwe jest również pisanie skryptów w UnityScript lub Boo.

#### *Blender*

Blender program typu open-source do modelowania i renderowania obiektów trójwymiarowych. Na potrzeby projektu, w tym programie zostały wytworzone wszystkie modele 3D znajdujące się w grze, oraz elementy interfejsu użytkownika.

#### *Trello*

Trello to aplikacja internetowa umożliwiająca tworzenie tablic w stylu Kanban. Na potrzeby projektu została stworzona tablica z opisem błędów oraz funkcji które w przyszłości mogą zostać zaimplementowane



Rys 3.1 Zrzut ekranu z programu Trello, tablica wykonana na potrzeby projektu (źródło: Własne)

### 3.2 Testowanie

W trakcie tworzenia projektu wiele elementów rozgrywki wymagało nieustannych udoskonaleń. W tym celu zostały przeprowadzone testy manualne. W ich skład wchodziły testy funkcjonalne oraz zamknięte play testy. Testy funkcjonalne zostały wykonane w celu sprawdzenia implementacji mechanik gry oraz postaci, czyli m.in.:

- Sprawdzenie kolizji między bohaterem a skrzynkami
- Sprawdzenie działania systemu checkpointów
- Sprawdzenie kolizji między bohaterem a przeciwnikami

W trakcie zamkniętych playtestów została udostępniona wersja testowa gry, na której testerzy sprawdzali funkcjonalność projektu, podczas zwykłej rozgrywki. Wszystkie błędy lub propozycje zmian zamieszczane były na tablicy Trello.



### **3.3 Dalszy rozwój projektu**

Na potrzeby niniejszej pracy, przygotowana została grywalna wersja demonstracyjna gry zawierająca jeden poziom. W dalszej kolejności planowane jest ukończenie kampanii, zawierającej kolejne trzy poziomy w odmiennej szacie graficznej oraz tematycznej. Dodatkowo, planowane jest stworzenie autorskiej ścieżki dźwiękowej. W przyszłości, planowane jest wydanie gry na platformy mobilne, docelowo telefony z systemem Android.

### **Zakończenie**

W pracy zrealizowano wszystkie cele założone przed rozpoczęciem pracy, a gra w obecnej formie jest satysfakcjonująca. Wersja demonstracyjna pozostawia ogromny potencjał rozwoju, szczególnie jeżeli w przyszłości powstanie mobilny port produkcji.

## Spis ilustracji

Rys 1.1 Wartość rynkowa branży gier wideo wyrażona w miliardach dolarów (źródło: [2]) ..	4
Rys 1.3.2 Model 3D bohatera. (źródło: Własne).....	6
Rys 1.3.2 Modele 3D. „Armata” i „szpikulce”. (źródło: Własne) .....	6
Rys 1.6.1 Zrzut ekranu z rozgrywki (źródło: [5]) .....	8
Rys 1.6.2 Zrzut ekranu z rozgrywki (źródło: [6]) .....	9
Rys 1.6.3 Zrzut ekranu z rozgrywki (źródło: [10]) .....	10
Rys 2.3.1 Zrzut ekranu z silnika Unity, przegląd katalogów (źródło: Własne) .....	12
Rys 2.3.2 Zrzut ekranu z silnika Unity, przegląd pakietów (źródło: Własne) .....	13
Rys 2.3.3 Zrzut ekranu z silnika Unity, przegląd GameObjectów na scenie(źródło: Własne)	14
Rys 2.4 Zrzut ekranu z gry (źródło: Własne) .....	15
Rys 2.4.1.1 Zrzut ekranu z silnika Unity, przegląd komponentów i skryptów Obiektu Gry „Player” (źródło: Własne) .....	16
Rys 2.4.1.2 Zrzut ekranu z gry, wspinaczka bohatera. (źródło: Własne).....	17
Rys 2.4.1.3 Zrzut ekranu z gry, dashowanie bohatera. (źródło: Własne) .....	18
Rys 2.4.1.4 Kod źródłowy skryptu <i>Movement</i> . (źródło: Własne) .....	18
Rys 2.4.1.5 Kod źródłowy skryptu <i>Gravity</i> . (źródło: Własne) .....	18
Rys 2.4.1.6 Kod źródłowy skryptu <i>Jump</i> . (źródło: Własne) .....	19
Rys 2.4.1.6 Kod źródłowy skryptu <i>Dash</i> . (źródło: Własne) .....	20
Rys 2.4.1.7 Kod źródłowy skryptu <i>WallGrab</i> . (źródło: Własne).....	21
Rys 2.4.1.8 Kod źródłowy skryptu <i>WallJump</i> . (źródło: Własne) .....	22
Rys 2.4.1.9 Kod źródłowy skryptu <i>WallWalking</i> . (źródło: Własne).....	23
Rys 2.4.2.1 Zrzut ekranu z gry, piłą. (źródło: Własne) .....	25
Rys 2.4.2.2 Zrzut ekranu z gry, przegląd komponentów i skryptów obiektu „Piła”. (źródło: Własne).....	25
Rys 2.4.2.3 Zrzut ekranu z gry, wygląd armaty razem z pociskiem (źródło: Własne) .....	26
Rys 2.4.2.4 Zrzut ekranu z silnika Unity, komponenty prefabu armaty (źródło: Własne) .....	26
Rys 2.4.2.5 Zrzut ekranu z silnika Unity, komponenty prefabu pocisku armaty (źródło: Własne).....	27
Rys 2.4.2.6 Zrzut ekranu z silnika Unity, komponenty prefabu pocisku armaty (źródło: Własne).....	27
Rys 2.4.2.7 Kod źródłowy skryptu <i>FireballSpawner</i> (źródło: Własne) .....	28
Rys 2.4.2.8 Kod źródłowy skryptu <i>FireballSpawner</i> (źródło: Własne) .....	28
Rys 2.4.2.9 Kod źródłowy <i>PlayerCollisionsWithEnemy</i> (źródło: Własne) .....	29
Rys 2.4.2.10 Zrzut ekranu z gry, wygląd animowanej wersji szpikulców(źródło: Własne)....	30

Rys 2.4.2.11 Zrzut ekranu z gry, wygląd statycznej wersji szpikulców(źródło: Własne) .....	30
Rys 2.4.3.1 Zrzut ekranu z gry, wygląd szmaragdu oraz serca, wraz z interfejsem graficznym(źródło: Własne).....	31
Rys 2.4.3.2 Zrzut ekranu z gry, wygląd monet (źródło: Własne) .....	31
Rys 2.4.3.3 Kod źródłowy skryptu <i>CoinScript</i> (źródło: Własne) .....	32
Skrypt <i>CoinScript</i> zawiera implementację mechaniki kolekcjonowania żetonów rozmieszczonych na planszy. ....	32
Rys 2.4.3.4 Kod źródłowy skryptu <i>CoinScript</i> (źródło: Własne) .....	32
Skrypt <i>HeartScript</i> zawiera implementację mechaniki kolekcjonowania serc rozmieszczonych na planszy, których otrzymanie będzie skutkowało się zwiększeniem możliwych prób przejścia poziomu.....	32
Rys 2.4.3.5 Kod źródłowy skryptu <i>CoinScript</i> (źródło: Własne) .....	33
Rys 2.4.4.1 Zrzut ekranu z silnika Unity, przegląd komponentów i skryptu obiektu „ <i>GameManagerHelper</i> ” (źródło: Własne).....	34
Rys 2.4.4.2 Kod źródłowy skryptu <i>GameManagerHelper</i> (źródło: Własne) .....	35
Skrypt <i>GameManagerHelper</i> odpowiada za kontrolę przebiegu rozgrywki na danym poziomie. Skrypt ten rozpatruje algorytm przywracania życia bohaterowi wraz ze sprawdzeniem otrzymywania obrażeń. ....	35
Rys 2.4.5.1 Zrzut ekranu z gry, aktywacja checkpointu (źródło: Własne) .....	36
Rys 2.4.5.2 Kod źródłowy skryptu <i>CheckpointScript</i> (źródło: Własne).....	36
Rys 2.4.6.1 Zrzut ekranu z silnika Unity, przegląd komponentów i skryptu obiektu „ <i>MovingPlatform</i> ” (źródło: Własne).....	37
Rys 2.4.6.2 Zrzut ekranu z gry, poruszanie się po ruchomej platformie (źródło: Własne) .....	37
Rys 2.4.6.3 Kod źródłowy skryptu <i>AttachPlayer</i> (źródło: Własne).....	38
Rys 2.4.6.5 Kod źródłowy skryptu <i>ObjectMovement</i> (źródło: Własne) .....	38
Rys 3.1 Zrzut ekranu z programu Trello, tablica wykonana na potrzeby projektu (źródło: Własne).....	39

## Bibliografia

- [1] Bankier.pl *Rynek gier nie zwalnia tempa. Jego wartość ciągle wzrasta*  
<https://www.bankier.pl/wiadomosc/Rynek-gier-nie-zwalnia-tempa-Jego-wartosc-ciagle-wzrasta-8145908.html>  
(na dzień 13.01.2022)
- [2] Statista *Value of the global video games market from 2012 to 2021*  
<https://www.statista.com/statistics/246888/value-of-the-global-video-game-market/>  
(na dzień 13.01.2022)
- [3] TheAlmightyGuru *Power-ups in 2d Platformers*  
[http://www.thealmightyguru.com/Wiki/index.php?title=Power-ups\\_in\\_2D\\_platformers](http://www.thealmightyguru.com/Wiki/index.php?title=Power-ups_in_2D_platformers)  
(na dzień 13.01.2022)
- [4] Unity Technologies *Introduction to Optimization with Unity*  
<https://learn.unity.com/tutorial/introduction-to-optimization-in-unity#5ff8ce16edbc2a0023134673>  
(na dzień 13.01.2022)
- [5] IGDB *Yoshi's Crafted World Press kit*  
<https://www.igdb.com/games/yoshis-crafted-world/presskit>  
(na dzień 13.01.2022)
- [6] Alice CLARET (Microids) *Press kit*  
[https://www.dropbox.com/sh/pbevju8hj0j8cnj/AAC\\_YdUNRh7O0JAU57T1jhGta?dl=0](https://www.dropbox.com/sh/pbevju8hj0j8cnj/AAC_YdUNRh7O0JAU57T1jhGta?dl=0)  
(na dzień 13.01.2022)
- [7] Microids *MARSUPILAMI: HOOBADVENTURE!*  
<https://www.microids.com/us/game-marsupilami-hoobadventure-us/>  
(na dzień 13.01.2022)
- [8] Nintendolife *All Yoshi Games*  
<https://www.nintendolife.com/games/browse?title=series%3Ayoshi>  
(na dzień 13.01.2022)
- [9] Franquin *Marsupilami*  
[http://www.franquin.com/marsu/index\\_marsu.php](http://www.franquin.com/marsu/index_marsu.php)  
(na dzień 13.01.2022)
- [10] IGDB *LittleBigPlanet 3 Press kit*  
<https://www.igdb.com/games/littlebigplanet-3/presskit>  
(na dzień 13.01.2022)
- [11] Idtech *What is tweening in animation?*  
<https://www.idtech.com/blog/what-is-tweening-in-animation>  
(na dzień 13.01.2022)
- [12] Dentedpixel *LeanTween Class*  
<http://dentedpixel.com/LeanTweenDocumentation/classes/LeanTween.html>  
(na dzień 13.01.2022)
- [13] Unity Documentation *Prefabs*  
<https://docs.unity3d.com/Manual/Prefabs.html>  
(na dzień 13.01.2022)
- [14] Unity *Probuilder*  
<https://unity.com/features/probuilder>  
(na dzień 13.01.2022)
- [15] Unity Documentation *TextMeshPro*  
<https://docs.unity3d.com/Manual/com.unity.textmeshpro.html>  
(na dzień 13.01.2022)
- [16] Unity Documentation *Tilemap*

<https://docs.unity3d.com/Manual/class-Tilemap.html>

(na dzień 15.01.2022)

[17] GiantBomb *Coyote Time*

<https://www.giantbomb.com/coyote-time/3015-9701/>

(na dzień 19.01.2022)

[18] Unity Documentation *Occlusion Culling*

<https://docs.unity3d.com/550/Documentation/Manual/OcclusionCulling.html>

(na dzień 19.01.2022)

Wrocław, dnia .....

**Wydział Informatyki**

Kierunek studiów: **Informatyka**

Radosław Matusiak  
(imię i nazwisko studenta)

6597  
(nr albumu)

## **OŚWIADCZENIE O UDOSTĘPNIANIU PRACY DYPLOMOWEJ**

Tytuł pracy dyplomowej: Architektura, projekt oraz implementacja gry platformowej w silniku Unity

Wyrażam zgodę (~~nie wyrażam zgody~~)<sup>1</sup> na udostępnianie mojej pracy dyplomowej.

.....  
(podpis studenta)

Wrocław, dnia.....

---

<sup>1</sup>Niepotrzebne skreślić.

**Wydział Informatyki**

Kierunek studiów: **Informatyka**

Radosław Matusiak  
(imię i nazwisko studenta)

6597  
(nr albumu)

## **OŚWIADCZENIE AUTORSKIE**

Oświadczam, że niniejszą pracę dyplomową pod tytułem:

Architektura, projekt oraz implementacja gry platformowej w silniku Unity napisałem/am samodzielnie. Nie korzystałem/am z pomocy osób trzecich, jak również nie dokonałem/am zapożyczeń z innych prac.

Wszystkie fragmenty pracy takie jak cytaty, ryciny, tabele, programy itp., które nie są mojego autorstwa, zostały odpowiednio zaznaczone i zamieszczono w pracy źródła ich pochodzenia. Treść wydrukowanej pracy dyplomowej jest identyczna z wersją pracy zapisaną na przekazywanym nośniku elektronicznym.

Jednocześnie przyjmuję do wiadomości, że jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdzi popełnienie przeze mnie plagiatu, skutkować to będzie niedopuszczeniem do dalszych czynności w sprawie nadania mi tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz złożenie zawiadomienia o podejrzeniu popełnienia przestępstwa.

.....  
(podpis studenta)