

# WYŻSZA SZKOŁA ZARZĄDZANIA „EDUKACJA”

Wydział Zarządzania  
Kierunek: informatyka

GRZEGORZ DZIKOWICKI  
NUMER ALBUMU: 24783

## PROJEKT GRY KOMPUTEROWEJ Z WYKORZYSTANIEM SILNIKA UNITY

**Praca inżynierska**

Praca napisana  
pod kierunkiem naukowym:

dr inż. Piotra Grobelnego

Wrocław 2020

# Spis treści

<b>Wstęp</b>	2
<b>1. Cel pracy, wymagania użytkownika, założenia, zakres i tezy pracy</b>	3
1.1. Cel pracy	3
1.2. Wymagania użytkownika	3
1.3. Założenia	3
1.4. Zakres	3
1.5. Teza pracy	3
<b>2. Badania własne</b>	4
2.1. Blender	4
2.2. Styl grafiki – Low-poly	5
2.3. Unity	6
<b>3. Metoda zrealizowania pracy</b>	7
3.1. Narzędzia badawcze	7
3.2. Technika tworzenia obiektów 3D	7
3.3. Środowisko Unity	8
3.4. Programowanie skryptów	9
<b>4. Wnioski</b>	13
<b>Spis literatury</b>	14
<b>Załączniki</b>	14
4.1. Zał. nr 1 Oświadczenie	14

# Wstęp

Gry komputerowe już od wielu lat kreują nowe trendy. Istnieją różne rodzaje gier. Gry typu AAA tworzone przez wielkie studia gier jak np. Ubisoft, Electronic Arts, czy też nasz rodzimy CD Project Red. Patrząc na gry komputerowe klasy AAA możemy śmiało powiedzieć, że przypominają filmy rodem z Disney'a czy też Pixar'a. W dzisiejszych czasach, aby oszczędzić sobie czasu, wiele dużych studiów używa technologii Motion Capture, aby odwzorować idealnie ruch postaci. Sprawia to, że potrzeba manualnego animowania postaci przestaje być kłopotem grafików, oraz animatorów, którzy mogą skupić się na innych aspektach swoich zadań.

Gry typu Indie to produkcje, które zazwyczaj są tworzone przez jedną osobę, bądź amatorskie studio złożone z kilku osób. Ze względu na brak funduszy oraz wyposażenia studia, produkcje te zazwyczaj są niskiej jakości, bądź średni czas potrzebny do przejścia gry jest o wiele krótszy niż gry AAA. Już dekadę temu, wielu graczy zmieniło upodobania co do grafiki, jaka pojawia się w grach komputerowych. Nawiązania do gier z przed dekady uwarunkowane są sentymentem do tamtego okresu. W 2019 do nagrody Paszportu Polityki 2019 została nominowana gra Dawida Ciślaka "We. The Revolution" która ukazuje nasze spory wokół sądów i polityki, jednakże podczas czasów rewolucji francuskiej.

W dzisiejszych czasach istnieje dużo gier, których grafika składa się z pikseli, voxeli czyli sześciokątów oraz w stylu low-poly który zostanie wykorzystany do tego projektu.

Low-poly jest to angielski termin określający siatkę modelu 3D, która składa się z małej liczby poligonów. Twórcy gier uważają, że low-poly stało się standardowym stylem grafiki gier komputerowych u dużej ilości osób rozpoczynających swoją przygodę z tworzeniem gier. Wykonanie każdego obiektu jest o wiele prostsze od modeli 3D przedstawiających realistycznie postać człowieka bądź drzewa. Ten styl graficzny posiada też swój urok, który można nazwać stylem kreskówkowym. Obiekty są wyraźnie zaznaczone i posiadają cechy charakterystyczne, które pozwalają odróżnić je od innych elementów rozgrywki.

# **1. Cel pracy, wymagania użytkownika, założenia, zakres i tezy pracy**

## **1.1. Cel pracy**

Celem pracy jest stworzenie projektu gry komputerowej za pomocą silnika Unity której grafika będzie reprezentowała styl graficzny „low-poly”.

## **1.2. Wymagania użytkownika**

Przez użytkownika (zamawiającego) zdefiniowane zostały następujące wymagania:

- gra musi być stworzona w środowisku Unity,
- projekt musi być przedstawiony w formie 3D,
- gra powinna posiadać świat przedstawiony w postaci low-poly.

## **1.3. Założenia**

Przy realizacji projektu przyjęto następujące założenia:

- do projektu należy stworzyć modele 3D samochodu oraz przeszkód,
- do projektu należy zaprogramować skrypty obsługujące sterowanie pojazdem oraz mechanikę związaną z przechodzeniem poziomów gry.

## **1.4. Zakres**

Zakres pracy obejmował:

- elementy modeli ze względu na braki funduszy, zostały stworzone w programie Blender,
- pracę ograniczono jedynie do pięciu krótkich poziomów, ze względu na aktualny stan projektu oraz komplikacje związane z rozmiarem.

## **1.5. Teza pracy**

Na podstawie przeglądu literatury dotyczącego zagadnienia przyjęto następujące tezy pracy:

— do prawidłowego zrealizowania projektu wystarczające jest środowisko Unity oraz Blender.

## 2. Badania własne

Proces badawczy to proces planowego, celowego wykorzystania uzyskanych informacji oraz własnej wiedzy. Celem głównym pracy jest próba stworzenia projektu świata przedstawionego w stylu graficznym low-poly. Głównym problemem jest rozmiar gry i złożoność poziomów. Zdecydowano więc, że projekt będzie przedstawiał pięć poziomów, w których gracz musi omijać przeszkody aby przejść na poziom wyżej.

W 1998r zadebiutowała gra Unreal wyprodukowana przez Epic MegaGames i Digital Extremes, która powstała na pierwszej wersji silnika Unreal Engine I. Swoją relatywnie prostszą strukturą wygryzła silnik Quake II Engine, który został wykorzystany w grze Quake. W 2006 ukazała się trzecia wersja silnika Unreal do której licencję wykupiło wiele przedsiębiorstw takich jak Electronic Arts, Activision czy też Ubisoft, które dziś są jednymi z najbardziej szanowanych studiów zajmujących się tworzeniem gier. W 2015r, Unreal Engine stał się darmowy dla każdego, kto chciał rozpocząć przygodę z tworzeniem gier.

Unity ukazało się w 2005r i początkowo był wykorzystywany w filmach, architekturze i symulacjach. W 2015r trafiło do darmowej dystrybucji i szybko stało się konkurencją dla Unreal. Dodatkowo Unity posiada Unity Asset Store, czyli sklep z gotowymi modelami, skryptami a nawet całymi projektami gier.

Nawzajem z każdą aktualizacją, obydwa silniki były rozwijane w taki sposób, aby udowodnić że są lepsze i mają więcej możliwości.

### 2.1. Blender

Rozpoczynając pracę związaną z tworzeniem grafiki 3D, postanowiłem dowiedzieć się o możliwych programach do właśnie takich zadań. Z pośród najbardziej popularnych

można wymienić Blendera, Autodesk Maya, Houdini oraz ZBrush. Jednym z kryteriów wyboru była cena zakupu programu. Oczywistym wyborem jest Blender, ponieważ jest on darmowym oprogramowaniem oraz posiada większość opcji z ww. programów. Modele wykorzystane w projekcie muszą być stworzone w stylu low-poly, oznacza to, że wymagania co do złożoności siatki obiektu pod względem topologii nie są wygórowane, wręcz muszą być zaniżone.

## 2.2. Styl grafiki – Low-poly

Low-poly jest określeniem siatki modelu 3D, który charakteryzuje się małą ilością wielokątów popularnie zwanych poligonami. Na przestrzeni lat, patrząc na początki gier komputerowych, dzisiejsze miano low-poly różnie się diametralnie od swoich poprzedników z przed dwóch dekad, ze względu na o wiele większą ilość poligonów oraz technologię dzięki której tworzenie modeli 3D jest o wiele łatwiejsze, jednakże dalej odbiega od modeli które są nastawione na realizm. Wówczas obiekt zachowuje znajomą nam geometrię. Poligony teoretycznie mogą posiadać nieskończoną ilość boków, jednakże silniki renderowania grafiki często napotykają problem z poligonami, których ilość boków jest większą od pięciu, zwłaszcza przy animacjach. Nieoficjalnie ustalono, że siatka modelu musi składać się z trójkątów, bądź prostokątów. Aby obiekt w stylu low-poly posiadał detale prawdziwego obiektu, używa się map normalnych i map wypukłości, które cieniują poligony i dodają im dodatkowej geometrii, aby posiadały detale, których ze względu na rozmiar siatki obiektu, nie da się uwzględnić ręcznie.

Rozwój technologii silników gier przedstawił nowe rozwiązania dla obiektów low-poly. Odkąd silniki oraz technologia obliczeniowa procesorów rozwinęła się w wielkim stopniu, obecne obliczenia są w stanie przedstawić setki tysięcy poligonów w standardowych 25 klatkach. W produkcjach AAA low-poly dalej istnieje, jednakże zostało ograniczone do wykorzystywania mniej detalicznych modeli jako forma zmniejszenia jakości oddalonych obiektów. LOD czyli Level Of Detail, definiuje zasięg z jakim dany obiekt posiada jakość detali. Im dalej znajduje się obiekt, tym gorsze odwzorowanie detali, jednakże nie pobiera tak dużo mocy obliczeniowej w przypadku głównej sceny w pobliżu postaci głównej. [1]

W przypadku kiedy twórca zdecydował, że chce użyć stylu low-poly, nie ma powodu, żeby używać LODa, ponieważ modele 3D w stylu low-poly zawierają wystarczającą liczbę detali jak i nie wymagają wielkiej mocy obliczeniowej.



Rysunek 2.1: Przykładowy render wyspy stworzonej w stylu low-poly

Źródło: *YouTube, CG Geek, Low Poly Island / Beginner / Blender 2.8 Tutorial, 2019.*

## 2.3. Unity

Istnieje wiele programów oraz silników do tworzenia gier komputerowych. Wiele dużych studiów takich jak Ubisoft, EA, CDProjektRED korzystają z własnych silników które sami napisali. Osoby, które tworzą gry w pojedynkę, korzystają z gotowych silników. Najpopularniejszymi są Unity oraz Unreal Engine. Wybierano Unity, ponieważ pozwala on na stworzenie takiej samej gry jak w Unreal, przy mniejszym wkładzie czasowym. Aktualnie jedynym językiem programowania jest C#, ponieważ silnik Unity posiada gotowe metody i funkcje, które współgrają wyłącznie z tym językiem.

Inspektor, czyli okno akcji każdego obiektu jak i całego projektu, pozwala modyfikować zmienne i dodawać nowe komponenty. Zimportowany obiekt można dowolnie zmieniać, tj. dodać materiał, kiedy np. importowany obiekt był tylko siatką meshową. W przypadku kiedy w skrypcie zakodowano zmienną, której nadano przykładową wartość, twórca podczas testowania jest w stanie zmieniać jej wartość nie modyfikując kodu, również podczas procesu testowania, właśnie w Inspektorze.

### 3. Metoda zrealizowania pracy

Realizacja pracy opiera się na opracowaniu kroków, które należy podjąć aby stworzyć świat gry w stylu low-poly. Dodatkowo aby zaprezentować stworzone elementy, zostanie opracowane przedstawienie produktu za pomocą projektu gry komputerowej stworzonej w Unity.

#### 3.1. Narzędzia badawcze

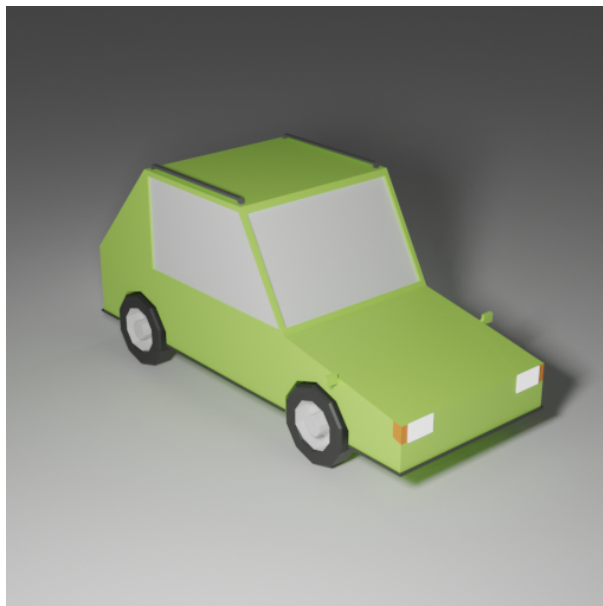
- Pierwszym narzędziem badawczym do stworzenia obiektów, będzie program Blender, który idealnie nadaje się do tworzenia elementów grafiki 3D,
- drugim narzędziem badawczym, które posłuży do stworzenia gry komputerowej będzie środowisko Unity, w którym zostaną ułożone podstawowe elementy rozgrywki, takie jak przeszkody,
- trzecim narzędziem badawczym, dzięki któremu napiszemy kod do gry, będzie Microsoft Visual Studio 2017.

#### 3.2. Technika tworzenia obiektów 3D

Obiekty grafiki 3D o mianie low-poly określa się przedmioty, postacie, świat gry itp. na których widać wyraźne elementy trisów bądź poligonów. Są to wyraźnie zaznaczone krawędzie obiektu, które swoją geometrią przypominają odpowiednik w świecie rzeczywistym, jednakże różnią się stopniem skomplikowania obiektu. Aby stworzyć taki model, najłatwiej jest zacząć od projektów poglądowych. W studiach gier, zatrudnieni są artyści, którzy podczas omawiania aspektów danych postaci bądź elementów świata, rysują proste szkice, które następnie poprawiają w celu uwydatnienia kluczowych aspektów obiektu. W późniejszym stadium rozwoju danego elementu, powstaje tzw. szkic końcowy. Wówczas owy szkic przekazuje się osobom, które zajmują się grafiką komputerową aby stworzyli dany obiekt w środowisku 3D. Do tego projektu użyłem zdjęć poglądowych dla beczki, oraz słupka drogowego.



Tworząc obiekt zazwyczaj zaczyna się od zwykłej kostki sześcienniej, którą następnie modyfikuje się, poprzez przesuwanie vertexów, krawędzi bądź całych ścian obiektu. Istnieje wiele narzędzi w Blenderze, które pomagają uzyskać porządkany przez nas efekt takie jak Mirror. Dodając ten modyfikator na obiekt, otrzymujemy odbicie lustrzane ukazujące dwie identyczne połówki. Korzystając ze zdjęcia referencyjnego ustawionego w widoku side view (bocznym) możemy ustawiać krawędzie obiektu nadając mu identyczny kształt jak na zdjęciu. Samochód który będzie głównym pojazdem w projekcie został stworzony za pomocą zdjęcia poglądowego Poloneza oraz własnej wyobraźni, ponieważ jego proporcje oraz kształt są niczym wyjęte z kreskówki. Obiekty stworzone w Blenderze, zostały wyeksportowane jako plik typu fbx.



Rysunek 3.1: Render samochodu stworzonego na potrzeby projektu

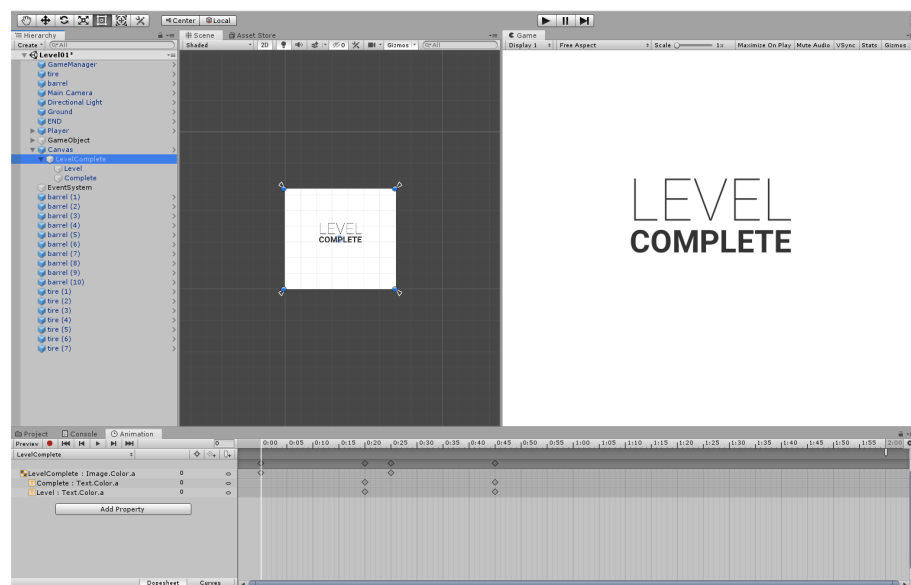
*Źródło: Opracowanie własne*

### 3.3. Środowisko Unity

Po wcześniejszym stworzeniu obiektów w Blenderze, przeszedłem do tworzenia środowiska w Unity. Aby obiekty nie wisiały w powietrzu, została stworzona płaska powierzchnia o rozmiarach 15x3x1000 jednostek Unity. Następnie umieszczono wszystkie obiekty w scenie. Każdy z obiektów posiada komponenty Mesh Collider, który odpowiada za wykrywanie kolizji z innymi obiektami, oraz Rigidbody będący głównym elementem fizyki obiektów. Zostały zmodyfikowane ich masy oraz rozmiary aby stworzyć dla nich "prefab".

Prefabem nazywany jest ten sam obiekt, jednakże po ponownym umieszczeniu go w scenie posiada te same właściwości. Ustawiając obiekty na powierzchni, stworzono 5 różnych poziomów, które różnią się trudnością, odpowiednio od pierwszego do piątego, bardziej skomplikowane ułożenie przeszkód oraz ich ilość.

Przejścia pomiędzy poziomami będą zawierały krótką i prostą animację wyświetlającą wiadomość „LEVEL COMPLETE”. Tworzenie animacji polega na stworzeniu Panelu z tekstem a następnie na osi czasu zmiany wartości alfy, tak aby napis oraz tło pojawiły się. W sekcji programowania znajduje się kod, odpowiedzialny za proces przejścia z jednego poziomu na drugi.



Rysunek 3.2: Screenshot pokazujący LEVEL COMPLETE, poniżej oś czasu z kluczowymi klatkami tworzącymi animację.

*Źródło: Opracowanie własne*

### 3.4. Programowanie skryptów

Programowanie zaczęto od stworzenia skryptu obsługującego poruszanie się pojazdem. Aby obiekt się poruszał, wykorzystano komponent Rigidbody umieszczony na samochodzie. Korzystając z gotowych funkcji zawartych w Rigidbody, mianowicie `AddForce()`, można sprawić aby obiekt poruszał się w danym kierunku z określoną mocą.

```

using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    // Odwołanie do komponentu Rigidbody pojazdu.
    public Rigidbody rb;

    // Dwie zmienne odpowiadające za ruch do przodu i na boki które można zmieniać w Inspektorze Unity.
    public float forwardForce = 2000f;
    public float sidewaysForce = 2000f;

    // Update is called once per frame
    void FixedUpdate()
    {
        //Ustalamy siłę z jaką kostka będzie się poruszać w przód.
        rb.AddForce(0, 0, forwardForce * Time.deltaTime);

        // JEŻELI d TO ruch w prawo z odpowiednią siłą.
        if (Input.GetKey("d"))
        {
            rb.AddForce(sidewaysForce * Time.deltaTime, 0, 0, ForceMode.VelocityChange);
        }

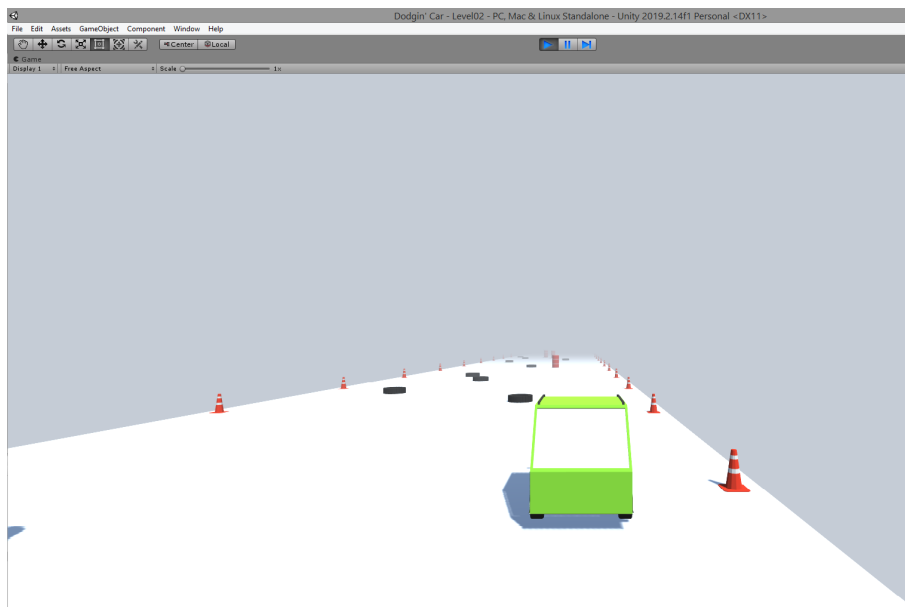
        // JEŻELI a TO ruch w prawo z odpowiednią siłą.
        if (Input.GetKey("a"))
        {
            rb.AddForce(-sidewaysForce * Time.deltaTime, 0, 0, ForceMode.VelocityChange);
        }

        // JEŻELI y spadnie poniżej -1 TO odwołanie do funkcji EndGame w skrypcie GameManager.
        if (rb.position.y < -1f)
        {
            FindObjectOfType<GameManager>().EndGame();
        }
    }
}

```

Rysunek 3.3: Skrypt PlayerMovement obsługujący poruszanie się pojazdu po poziomie.

*Źródło: Opracowanie własne*



Rysunek 3.4: Zdjęcie ukazujące pojazd, który zboczył ze środka trasy.

*Źródło: Opracowanie własne*

Kamera musi śledzić gracza, dlatego napisano skrypt, który ustala pozycję kamery w świecie gry, na taką samą pozycję co pojazd, jednakże dodany jest offset, który podnosi kamerę do góry i przesuwa ją do tyłu, tak aby widok był z perspektywy trzeciej osoby.

Kolejnym zadaniem było napisanie skryptu, który będzie wykrywał kolizje z przeszk-

```

using UnityEngine;

public class FollowPlayer : MonoBehaviour
{
    // Korzystanie z Transforma playera, x y z.
    public Transform player;
    public Vector3 offset;

    // Update is called once per frame
    void Update()
    {
        // Kamera posiada te same wartości x y z co pojazd
        // Dodatkowo offset powoduje widok z perspektywy trzeciej osoby.
        transform.position = player.position + offset;
    }
}

```

Rysunek 3.5: Skrypt FollowPlayer obsługujący umieszczenie kamery za pojazdem.

*Źródło: Opracowanie własne*

dami oraz resetował poziom, jeżeli pojazd spadnie z planszy. Ponieważ platforma po której porusza się pojazd, również jest wykrywana jako kolizja, wszystkie przeszkody zostały otagowane jako Obstacle. W skrypcie PlayerMovement został dodany kod obsługujący resetowanie poziomu, jeżeli wartość y pojazdu spadnie poniżej -1, który odwołuje się do funkcji EndGame w skrypcie GameManager.

```

using UnityEngine;

public class PlayerCollision : MonoBehaviour
{
    // Nawiązanie do skryptu PlayerMovement.
    public PlayerMovement movement;

    void OnCollisionEnter(Collision collision)
    {
        // Jeżeli zderzenie będzie z obiektem o tagu Obstacle.
        if (collision.collider.tag == "Obstacle")
        {
            // Wyłącza wszystkie siły na obiekcie. Przejście do EndGame.
            movement.enabled = false;
            FindObjectOfType<GameManager>().EndGame();
        }
    }
}

```

Rysunek 3.6: Skrypt PlayerCollision obsługujący kolizje pomiędzy pojazdem a przeszkodami.

*Źródło: Opracowanie własne*

Ponieważ w inspektorze nie da się podpiąć skryptu GameManager do prefabu pojazdu, w skrypcie PlayerCollision znajduje się metoda FindObjectOfType która przeszukuje w lokalizacji skryptów plik o nazwie GameManager a następnie korzysta z jego funkcji

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    bool gameHasEnded = false;
    public float restartDelay = 2f;

    // gameCompleteUI oraz CompleteLevel() podpięte pod END point.
    public GameObject gameCompleteUI;

    public void CompleteLevel()
    {
        // SetActive uruchamia animacje ukończonego poziomu.
        gameCompleteUI.SetActive(true);
    }

    public void EndGame()
    {
        if (gameHasEnded == false)
        {
            // Zmiana boola na true, żeby funkcja nie uruchamiała się w nieskończoność.
            gameHasEnded = true;
            // Invoke, uruchamia funkcję po pewnym czasie - restartDelay.
            Invoke("Restart", restartDelay);
        }
    }

    void Restart()
    {
        // Restart tego samego poziomu, którego nie udało się przejść.
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}

```

Rysunek 3.7: Skrypt GameManager zajmujący się restowaniem poziomów.

*Źródło: Opracowanie własne*

która musi być ustawiona jako public. Tworząc boola gameHasEnded, którego wartość zmieniana jest przy pierwszym uruchomieniu funkcji, upewniamy się, że funkcja będzie ładowana tylko jeden raz. Następnie po upadku, bądź kolizji, poziom jest ładowany od nowa, tak więc gameHasEnded ma swoją pierwotną wartość. Aby gra nie resetowała się za szybko, zamiast uruchamiać funkcję EndGame od razu, wykorzystałem metodę Invoke która po dodaniu nazwy funkcji, oraz czasu w sekundach, uruchamia funkcję po chwili. Aby przejść na następny poziom, musimy przejechać pomiędzy przeszkodami tak żeby ich nie dotknąć. W przypadku porażki poziom zostanie załadowany od nowa.

Poziom jest już możliwy do przejścia, jednakże trzeba załadować kolejny. Na końcu trasy każdego z poziomów umieściłem kostkę o szerokości 15 jednostek Unity, która jest niewidoczna i służy jako Trigger, który uruchamia animację zakończonego poziomu, oraz ładuje kolejny. W skrypcie EndTrigger znajduje się jedna funkcja OnTriggerEnter(), która ładuje funkcję CompleteLevel() ze skryptu GameManager odpowiedzialną za włączenie animacji zakończenia poziomu.

```

using UnityEngine;

public class EndTrigger : MonoBehaviour
{
    // Nawiązanie do skryptu GameManager.
    public GameManager gameManager;
    void OnTriggerEnter()
    {
        gameManager.CompleteLevel();
    }
}

```

Rysunek 3.8: Skrypt EndTrigger.

*Źródło: Opracowanie własne*

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelComplete : MonoBehaviour
{
    public void LoadNextLevel()
    {
        // Menadżer scen ładuje następną scenę określoną w Buildzie gry.
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}

```

Rysunek 3.9: Skrypt LevelComplete ładujący kolejny poziom.

*Źródło: Opracowanie własne*

## 4. Wnioski

Realizacja pracy pozwoliła sprecyzować następujące wnioski:

1. do stworzenia gry 3D w stylu low poly wystarczające jest środowisko Unity,
2. Blender jest wystarczająco dobrym środowiskiem do tworzenia obiektów 3D.

Objaśnienie:

**Wnioski:** (lista syntetycznych wniosków)

Wnioski powinny:

1. jednoznacznie określać czy i jak został osiągnięty cel pracy,
2. podsumować w jaki sposób zweryfikowano hipotezy (jeśli były postawione),
3. odpowiadać na pytanie o słuszność tez (czy je udowodniono).

## Spis literatury

- [1] Fermentas Inc. Unity technologies: Unity user manual. <https://docs.unity3d.com/Manual/index.html/>, January 2019.

## Załączniki

### 4.1. Zał. nr 1 Oświadczenie

OŚWIADCZENIE DOTYCZĄCE PRAW AUTORSKICH I DANYCH OSOBOWYCH  
PRZECHOWYWANYCH W SYSTEMIE ANTYPLAGIATOWYM

Ja, niżej podpisany/a

Imię (imiona) i nazwisko .....

autor pracy dyplomowa pt. ....

.....

1. Numer albumu: .....

2. Student/ka, Wydziału: .....  
we Wrocławiu/w Kłodzku\* Wyższej Szkoły Zarządzania „Edukacja”

3. Kierunek i specjalność studiów: .....

4. Oświadczam, że:

- a) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. Nr 24, poz. 83 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- b) nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
- c) nie była podstawą nadania dyplomu uczelni wyższej lub tytułu zawodowego ani mnie ani innej osobie.
- d) jest związana z zaliczeniem studiów w Wyższej Szkole Zarządzania „Edukacja” we Wrocławiu.
- e) że treść pracy przedstawionej przeze mnie do obrony zawarta na przekazywanym nośniku elektronicznym jest identyczna z wersją drukowaną.

5. Udzielam Uczelni prawa do wprowadzenia i przetwarzania w systemie antyplagiatowym pracy dyplomowej mojego autorstwa oraz wyrażam zgodę na przechowywanie jej w celach realizowanej procedury antyplagiatowej w bazie cyfrowej systemu antyplagiatowego. Oświadczam, że zostałem/am poinformowany/a i wyrażam na to zgodę, że tekst mojej pracy stanie się elementem porównawczej bazy danych Uczelni, która będzie wykorzystywana, w tym także udostępniana innym podmiotom, na zasadach określonych przez Uczelnię, w celu dokonywania kontroli antyplagiatowej prac dyplomowych, a także innych tekstów, które powstaną w przyszłości.

.....  
(miejscowość, data)

.....  
(czytelny podpis autora pracy)

\*niepotrzebne skreślić