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. Piotr Grobelny

Wrocław 2019

Spis treści

Wstęp

Gry komputerowe już od wielu lat kreują nowe trendy. Istnieją dwa 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 rigowania postaci — ustawienia ruchu postaci w sposób realistyczny — przestaje być kłopotem grafików, oraz animatorów, którzy mogą skupić się na innych aspektach swoich zadań.

Gry typu Indie są 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. W dzisiejszych czasach jest pełno gier, których grafika składa się z pikseli, voxeli czyli sześciokątów oraz w stylu low-poly który zostanie wykorzystany do tego projektu.

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,

— zaprogramować skryptów obsługujących przedsięwzięcie.

1.4. Zakres

Zakres pracy obejmował:

— elementy modeli ze względu na braki funduszy, zostaną stworzone w programie Blender,

— pracę ograniczono jedynie do trzech krótkich poziomów, ze względu na aktualny stan projektu.

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 skala. Zadecydowano więc, że projekt będzie przedstawiał trzy poziomy, w których gracz musi omijać przeszkody aby przejść na poziom wyżej.

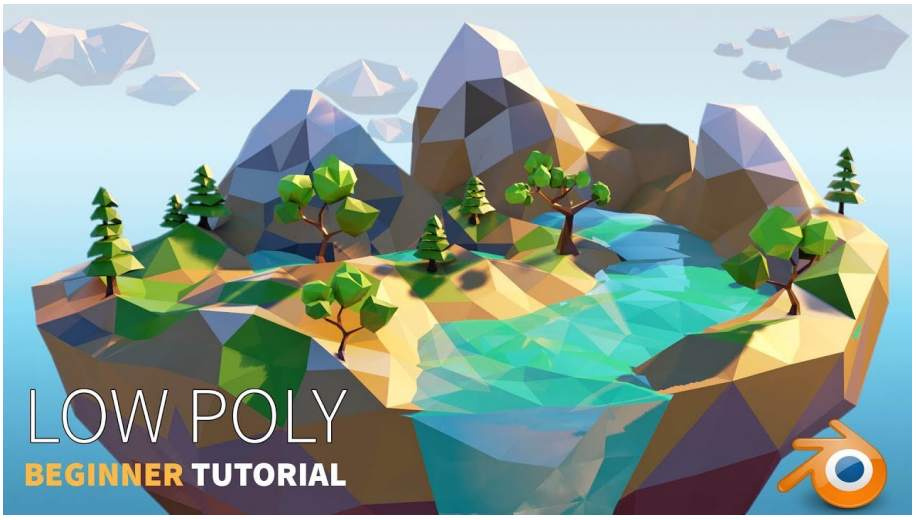
2.1. Blender

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 znajomom nam geometrię, jednakże

nie posiada on idealnie gładkich krawędzi które odwzorcowałyby przedmiot w świecie rzeczywistym.



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. Wybierając Unity kierowałem się własnymi doświadczeniami związanymi z grami komputerowymi i programowaniem. Dużo gier, które miałem okazję zobaczyć było tworzone właśnie na silniku Unity. Unity pozwala programować w dwóch językach skryptów, C# który jest podobny do C++, oraz JavaScript. C# jest dla mnie bardzo intuicyjny ze względu na wielkie podobieństwo do C++ z którym miałem już doczynienia.

3. Metoda zrealizowania pracy

Realizacja pracy będzie się opierać 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

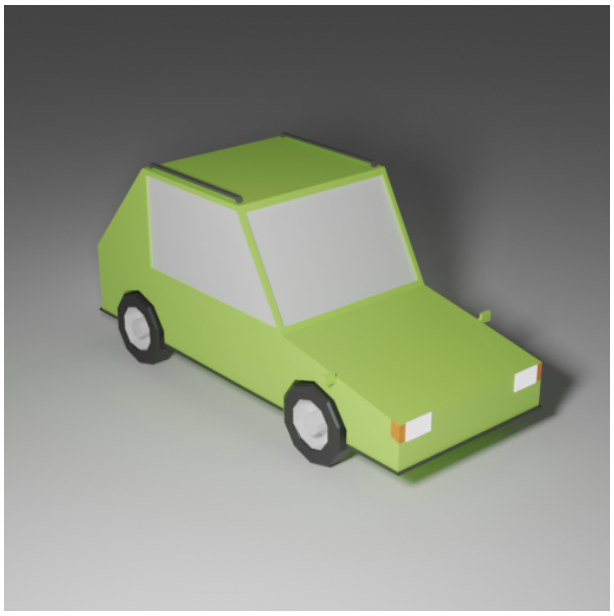
— Pierwszym narzędziem badawczym do stworzenia obiektów, będzie program Blender, który idealnie nadaje się do tworzenia elementów grafiki 3D,
— drugim narzędziem badawczym, które posłuży do stworzenia gry komputerowej będzie środowisko Unity, w którym zostaną zaprogramowane podstawowe elementy rozgrywki,
— 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, zazwyczaj zatrudnieni sią 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 sześciennej, którą następnie modyfikuje się, poprzez przesuwanie vertexów, krawędzi bądź całych ścian obiektu. Istnieje wiele narzędzi w Blenderze, które pomagają uzyskać porządany 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: Grzegorz Dzikowicki

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 umieściłem 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, stworzyłem 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 ŁEVEL 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.

3.4. Programowanie skryptów

Programowanie zacząłem od stworzenia skryptu obsługującego poruszanie się pojazdem. Aby obiekt się poruszał, wykorzystałem komponent Rigidbody umieszczony na samochodzie. Korzystając z gotowych funkcji zawartch 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 sidewayForce = 2000f;

    // Update is called once per frame
    void FixedUpdate()
    {
        //Ustalamy siłe 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(sidewayForce * Time.deltaTime, 0, 0, ForceMode.VelocityChange);
        }

        // JEŻELI a TO ruch w prawo z odpowiednią siłą.
        if (Input.GetKey("a"))
        {
            rb.AddForce(-sidewayForce * 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 3.2: Skrypt PlayerMovement obsługujący poruszanie się pojazdu po poziomie.

Źródło: Grzegorz Dzikowicki

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

```
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 3.3: Skrypt FollowPlayer obsługujący umieszczenie kamery za pojazdem.

Źródło: Grzegorz Dzikowicki

Kolejnym przedsięwzięciem było napisanie skryptu, który będzie wykrywał kolizje z przeszkodami 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 3.4: Skrypt PlayerCollision obsługujący kolizje pomiędzy pojazdem a przeszkodami.

Źródło: Grzegorz Dzikowicki

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 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.

```
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 3.5: Skrypt GameManager zajmujący się restowaniem poziomów.

Źródło: Grzegorz Dzikowicki

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 3.6: Skrypt EndTrigger.

Źródło: Grzegorz Dzikowicki

```
using UnityEngine;
using UnityEngine.SceneManagement;

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

Rysunek 3.7: Skrypt LevelComplete ładujący kolejny poziom.

Źródło: Grzegorz Dzikowicki

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).

Załączniki

4.1. Zał. nr 1 Oświadczenie

Załącznik 1 do Regulaminu funkcjonowania systemu antyplagiatowego
w Wyższej Szkole Zarządzania „Edukacja" we Wrocławiu

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 Zarzadzania „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, ze 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*