

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

1. Wstęp	3
2. Cel pracy, wymagania użytkownika, założenia, zakres i tezy pracy	5
2.1. Cel pracy	5
2.2. Wymagania użytkownika	5
2.3. Założenia	5
2.4. Zakres	5
2.5. Teza pracy	6
3. Badania własne	7
3.1. Program do modelowania 3D	7
3.2. Styl grafiki – Low-poly	10
3.3. Środowisko tworzenia gry	14
4. Metoda zrealizowania pracy	17
4.1. Narzędzia badawcze	17
4.2. Technika tworzenia obiektów 3D	17
4.2.1. Oparcie na zdjęciach referencyjnych	17
4.2.2. Modyfikatory obiektów	20
4.2.3. Wireframe czyli siatka meshowa	23
4.2.4. Tworzenie modelu 3D pojazdu	24
4.2.5. Tworzenie modelu 3D przeszkód	28
4.3. Środowisko Unity	29
4.3.1. Scenariusz rozgrywki	30
4.3.2. Materiał poślizgowy	32
4.3.3. Animacje	33
4.4. Programowanie skryptów	34
4.4.1. Poruszanie pojazdem	34
4.4.2. Obsługa kolizji	37
4.4.3. Ładowanie poziomów	40
4.5. Przygotowanie gry do publikacji	42

5. Podsumowanie	43
Spis literatury	46
Spis rysunków	47
Załączniki	48
Zał. nr 1 Oświadczenie dotyczące praw autorskich i danych osobowych przechowywanych w Systemie Antyplagiatowym	48
Zał. nr 2 Płyta CD	48

1. Wstęp

Pierwsze komputery powstały z myślą o obliczeniach balistycznych, wytwarzaniem broni jądrowej, metrologii oraz badaniu promieniowania kosmicznego. W latach 70. XX wieku, gry komputerowe były produkowane w masowych ilościach ze względu na wzrost popularności automatów do gier oraz pierwszych konsol. 30 lat później, twórcy podzespołów komputerowych produkują sprzęt z uwagą na to, aby najnowsze produkcje gier działały w jak najlepszej rozdzielczości, z jak największą liczbą klatek na sekundę. Prawdziwie prestiżowym wynikiem jest granie w rozdzielczości 4K w 60 klatkach na sekundę.

Gry komputerowe już od wielu lat kreują nowe trendy, przez co powstało wiele różnych rodzajów 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. Obecnie, wielu graczy zmieniło upodobania co do grafiki, jaka pojawia się w grach komputerowych.

Nawiązania do gier sprzed dekady uwarunkowane są sentymentem do tamtego okresu. Gry wtedy były robione z pasji. Aktualnie pojawia się wiele skandalów co do wielkich firm tworzących gry wyłącznie dla pieniędzy. Przykładem może być Electronic Arts, który 8 marca 2012r wydał grę Mass Effect 3, a dzień później udostępnił DLC, czyli zawartość dodatkową za opłatą wynoszącą około 50% ceny gry. Gracze byli oburzeni, dlatego zaczęli masowo zwracać gry do sklepu. Ponieważ liczba zwrotów wynosiła blisko 60% zakupionych wersji gry, EA zdecydowało udostępnić tylko ten dodatek za darmo, następne były już płatne.

W 2019 roku do nagrody „Paszportu Polityki 2019” została nominowana gra Dawida Ciślaka "We. The Revolution" która ukazuje nasze spory wokół sądów i polityki, które dzieją się podczas czasów rewolucji francuskiej. 18 lipca 2019r ukazała się gra Horace stworzona przez niezależne studio 505 Games. Została ona stworzona w stylu pixelowym, z naciskiem na grafikę 2D, która w niektórych scenach zmienia się na 3D. Poprzez takie wymieszanie rodzajów grafik, mieszankę stylów oraz interesującą historią otrzymała przydomek „Najlepszej platformówki 2019 roku”. W dzisiejszych czasach istnieje dużo gier, których grafika składa się z pikseli, voxelów czyli sześciokątów oraz w stylu low-poly, który został wykorzystany do projektu opisanego w tej pracy.

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.

2. Cel pracy, wymagania użytkownika, założenia, zakres i tezy pracy

2.1. Cel pracy

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

2.2. 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.

2.3. 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.

2.4. Zakres

Zakres pracy obejmował:

- elementy modeli, które zostały stworzone w programie Blender,
- pracę ograniczono jedynie do pięciu krótkich poziomów.

2.5. Teza pracy

Na podstawie przeglądu literatury dotyczącego zagadnienia przyjęto następującą tezę pracy:

- do prawidłowego zrealizowania projektu wystarczające jest środowisko składające się z programów Blender, Unity, oraz Inno Setup.

3. Badania własne

Proces badawczy to proces planowego, celowego wykorzystania uzy-skanych 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. Zadecy-dowano więc, że projekt będzie przedstawał pięć poziomów, w których gracz musi omijać przeszkody aby przejść na poziom wyżej.

Przeprowadzając badania własne, postanowiono zapoznać się z ryn-kiem narzędzi służących do tworzenia grafiki 3D. Wybrano 3DS Max, Autodesk Maya, Cinema 4D, ZBrush, Blender oraz Houdini.

Przeszukując możliwe silniki i oprogramowania do tworzenia gier, zdecydowano zaznajomić się z programami RPG Maker, GameMaker, Unreal Engine oraz Unity.

3.1. Program do modelowania 3D

Rynek programów służących do modelowania 3D jest nastawiony na specyficzne projekty. Każde oprogramowanie wybierane jest pod projekt i najczęściej są to programy, które posiadają reputację niezniszczalnych programów, które są niezawodne. Jeżeli firma zajmuje się projektem, który posiada nieprzekraczalny termin dostarczenia, pewnym jest, że wybierze program, który posiada wsparcie konsumenta. Jest to jedna z głównych przyczyn, dlaczego takie programy jak Blender, które są wolnym i otwartym oprogramowaniem, nie są wybierane, ponieważ nie posiadają personelu zajmującego się awariami. Każdy artysta ma swój ulubiony program. Często jest tak, że kierownik projektu z góry narzuca oprogramowanie. Obecnie, do modelowania 3D, animowania, efektów

specjalnych oraz rzeźbienia jest używane przynajmniej 5 programów, które nie każdy może znać. Wywiera to presję na artyście. Nie może on skupić się na pracy, ponieważ musi nauczyć się obsługi oprogramowania, które prawdopodobnie zmieni się przy kolejnym projekcie. Wymóg znajomości większości programów do modelowania 3D stał się priorytetem podczas zatrudniania nowego personelu.

3DS Max jest znany już od dawna. Pierwsza wersja programu ukazała się w 1996r pod nazwą „3D Studio Max 1.0”. Posiada bardzo silną pozycję na rynku i jest jednym z najczęściej wybieranych programów do tworzenia grafiki 3D. Program jest przystosowany do tworzenia animacji postaci, ze względu na zaimplementowany system, Character Studio. To oprogramowanie jest często używane w firmach zajmujących się filmami, animacjami oraz grami.

Pierwsza wersja Autodesk Maya zadebiutowała w 1998r. Jeżeli grafikowi zależy na efektach specjalnych, Maya jest najbardziej profesjonalnym wyborem. Posiada solidny zestaw narzędzi, który jest nieobecny w niektórych programach. Niestety profesjonalizm idzie w parze z kosztami. Maya jest jednym z programów graficznych, który posiada najbardziej wygórowaną cenę, 200£ miesięcznie. Firmy takie jak Pixar, korzystają praktycznie tylko z Maya, bowiem ich produkcje wymagają jak najlepszych efektów. Jest to jeden z programów który jest wychwalany przez artystów oraz grafików, ponieważ posiada niesamowity wachlarz narzędzi. Poprzez ilość opcji, funkcji oraz mechanizmów jest również bardzo skomplikowany, przez co jego nauka jest bardzo trudna. Duża liczba osób zaczynających przygodę z modelowaniem 3D, często sięga po darmową wersję trial, jednakże po pierwszym spotkaniu z interfejsem jak i trudnościami w stworzeniu czegokolwiek, szybko zostawiają to oprogramowanie w spokoju.

Jeżeli artyście zależy tylko na animacjach, idealnym oprogramowaniem będzie Cinema 4D. Oprogramowanie to, jest prawdopodobnie jednym z najstarszych na tej liście, bowiem pierwsza jego wersja ukazała się w 1990r. Prawdopodobnym jest, że w większości filmów zawierających nienaturalne animacje, których aktor nie dał rady wykonać, znajduje się animacja stworzona w Cinema 4D.

ZBrush jest programem który jest nastawiony na Sculpting, czyli rzeźbienie. Tworząc zwykłą kulę a następnie modyfikując ją wszelakimi pędzlami, które dodają, bądź odejmują geometrię, jesteśmy w stanie uzyskać rzeźby identyczne jak w rzeczywistości. Jest to idealny program do tworzenia nieruchomych postaci, które przykładowo pozują. Najczęściej jest wykorzystywany do stworzenia obiektu przypominającego jakiegoś aktora lub postać, który następnie jest użyty w reklamie jako statyczna postać.

Houdini w głównej mierze używany jest do tworzenia obiektów proceduralnych. Są to obiekty, które są tworzone przez funkcje i modyfikatory, które przy różnych ustawieniach wartości mogą ukazać inny wynik. Jest on tańszym wariantem Autodesk Maya, ponieważ graficy używają go do tworzenia animacji na obiektach wcześniej stworzonych w innym programie do tworzenia grafiki 3D. Często Houdini jest też wykorzystywany do animacji obiektów. Niestety nie jest on na tyle dobrą alternatywą dla Autodesk Maya.

Blender z kolei, jest połączeniem każdego ww. oprogramowania. Posiada praktycznie identyczne funkcje 3DS Maxa pod względem modyfikowania siatki meshowej obiektów 3D. Jeżeli chcemy uzyskać delikatne efekty, można spokojnie użyć do tego Blendera, który poradzi sobie dobrze z naszym problemem, chociaż w Maya udałoby się uzyskać ten sam efekt o wiele lepiej. Animacje w Blenderze polegają jedynie na wyobrażni autora. Korzystając z wszelakich transformów oraz modyfikatorów, animacje nie stanowią problemu. Rzeźbienie w Blenderze, przy ustawie-

niach podstawowych jest trudniejsze niż w ZBrush'u. Ponieważ istnieje opcja importowania gotowych pędzli, dopiero po zebraniu dużej ilości narzędzi przystosowanych do jednego celu, jesteśmy w stanie uzyskać praktycznie ten sam wachlarz narzędzi co w ZBrushu. W przypadku tworzenia obiektów proceduralnych, Blender radzi sobie bardzo dobrze. Aby uzyskać efekt jaki możemy dostać w Houdini czy Maya, animator musi się solidnie naprawować, co nie oznacza, że jest to niemożliwe do osiągnięcia.

Do realizacji pracy wybrano Blendera, ponieważ każdy z elementów rozgrywki jaki ukaże się w grze jest możliwy do wykonania w tym programie. Ponieważ obiekty 3D potrzebne do projektu nie są wymagające, a efekty specjalne jak i animacje obiektów są nieobecne, do wykonania modeli wykorzystano program Blender.

3.2. Styl grafiki – Low-poly

Pierwsza gra, która posiadała „prawdziwe” 3D, Descent ukazała się w 1995r i została stworzona przez wydawcę gier Interplay Entertainment. Grafikę skonstruowano z poligonów i jako jedna z niewielu gier tamtych czasów pozwalała na sześć stopni swobody, czyli możliwość poruszania się postacią w każdą możliwą stronę horyzontalnie i wertykalnie.

Graficy, tworząc obiekty 3D używają sformułowań odpowiednich do tworzonego przez nich kształtu. Trójkąty popularnie nazywane trisami oraz prostokąty, czyli poligony, posiadają odpowiednią dla figury geometrycznej liczbę ścian i vertexów. **Vertexy**, są to punkty łączenia krawędzi poligonu. Istnieją również n-gony, czyli figury składające się z liczby ścian większej niż 4. W przypadku, kiedy obiekt nie jest animowany, nie ma żadnych przeciwskań. Należy jednak pamiętać, że w przypadku, gdy obiekt musi być animowany, powinno się unikać takich rozwiązań, ponieważ silniki renderowania grafiki, mają problem z obliczeniem odbicia światła od powierzchni zawierającej więcej niż 4 vertexy.

Oprawa graficzna w grach jest jednym z najważniejszych kryteriów, która wpływa na to, czy ktoś będzie grał w tą produkcję. Gracze są jednym z najgorszych celów marketingowych. Każdy ma inne wymagania oraz upodobania, przez co trudno trafić w większość. Dzisiejsze produkcje twórców zajmujących się produkcjami AAA opierają się na fotorealistyczności. Jak można się domyślić, nie każdemu to pasuje. Spora liczba graczy preferuje gry, które posiadają wygląd przypominający kreskówki, bądź komiksy. Jedną z takich gier jest „The Wolf Among Us” producenta Telltale Games z 2013r. Gra przedstawiona jest w postaci 3D, a styl graficzny jest połączeniem fotorealistycznych modeli z kreskówkowymi elementami. Przedstawia ona świat, w którym są umieszczone postacie z najbardziej znanych baśni, które często ukazują zachowania patologiczne. Główne postacie żyją w Nowym Jorku, jednej z największych metropolii świata.

W przypadku stylu gier komputerowych określających wymiary przestrzeni, możliwym jest wymienienie głównych przedstawicieli. Gry w stylu 3D zachowujące fotorealizm znajdują w rankingach popularności, dla tego też największe studia takie jak Electronic Arts czy Ubisoft skupiają się właśnie na fotorealizmie. Pomniejsze firmy, których produkty nie są rozpoznawalne na całym globie mają inne zdanie. Każdy ich projekt jest innym przedsięwzięciem celowanym w zupełnie inną publicę. Tacy producenci szukają swojego stylu. W przypadku jeżeli ich gra spodoba się sporej liczbie graczy, próbują stworzyć inną grę, która swoją oprawą graficzną będzie przypominać poprzednią produkcję. Nie zmienia to faktu, że duża liczba gier jest ukazana w środowisku trójwymiarowym. Istnieją też zawieści fanatycy konkretnych rodzajów gier.

Gry 2D przeżywają swego rodzaju renesans. Takie gry jak Horace, Stardew Valley, Darkest Dungeon itp. podbijają serca graczy swoją grafiką i historią. Większość z nich posiada bardzo skomplikowane mechaniki, na które 3D nie pozwala, bądź wymagany jest bardzo duży

wkład czasowy. Mechaniki te mają na celu pokazanie odbiorcy, że nie tylko grafika ma ich cieszyć a mechaniki związane z rozgrywką.

Gry dzielą się w głównej mierze na 3D i 2D. Ale nie każda gra jest równa drugiej. Dochodzą przeróżne style graficzne. Możliwym jest wylistowanie kilku z najpopularniejszych. Cartooning, czyli po prostu komiks, jest mniej popularny niż reszta stylów, co nie zmienia faktu, że posiada swój urok. Świecznie wykonane postacie i charakterystyczna czarna kreska tworząca kontur postaci, kolory są bardzo wyraźne.

Flat design, czyli płaskie elementy zawierające kształty geometryczne. Jak każdy ze stylów, posiada fanów oraz krytyków. Spora liczba gier na telefony jest stworzona w stylu flat design. Jego całkowitym przeciwnieństwem jest styl fotorealistyczny 3D. Popularnie uważanym jest, że ten styl zawarty jest tylko w poważnych grach. Zaletą jest poziom detali otoczenia jak i postaci. Minusem jest czas pracy nad poszczególnymi elementami, co również czyni ten styl bardzo kosztownym.

Low-poly jest określeniem siatki modelu 3D, który charakteryzuje się małą ilością wielokątów popularnie zwanych poligonami. Poligony są wykorzystywane do tworzenia obiektów 3D. Siatka modelu tworzona jest poprzez łączenie poligonów w jedną spójną całość. Na przestrzeni lat, patrząc na początki gier komputerowych, dzisiejsze miano low-poly różni się diametralnie od swoich poprzedników z przed dwóch dekad. Ze względu na o wiele większą liczbę poligonów oraz technologię dzięki której tworzenie modeli 3D jest o wiele łatwiejsze, ten styl dalej odbiega od modeli które są nastawione na realizm. Poligony teoretycznie mogą posiadać nieskończoną liczbę boków, jednakże silniki renderowania grafiki często napotykają problem z poligonami, których liczba boków jest większa od pięciu, zwłaszcza przy animacjach ukazujących obiekt w ruchu. 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 cieniąją

poligony i dodają im dodatkowej geometrii. Mapy pomagają uwypatnić detale, których ze względu na ilość poligonów w siatce obiektu, nie da się uwzględnić ręcznie.

Rozwój technologii silników gier umożliwił 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 są detali, które nie pobierają tak dużo mocy obliczeniowej w przypadku głównej sceny, przez co gra działa płynnie [1].

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



Rysunek 3.1: Przykładowy render wyspy stworzonej w stylu low-poly

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

3.3. Środowisko tworzenia gry

Tworząc grę, pierwszą decyzją musi być historia, szata graficzna a następnie wybór silnika oraz środowiska w którym będzie tworzona gra. Ponieważ rynek gier przez ostatnie dwie dekady powiększył się, niektóre silniki postanowiły otworzyć się na świat i stały się wolnymi od opłat oprogramowaniemi. Silnikiem określa się zbiór gotowych funkcji potrzebnych do stworzenia gry, które często posiadają zintegrowane środowisko. Silnik zajmuje się komunikowaniem pomiędzy poszczególnymi elementami projektu.

Istnieje wiele programów oraz silników do tworzenia gier komputerowych. Wiodące w branży gier studia, takie jak Ubisoft, Electronic Arts, CD Projekt RED korzystają z własnych silników. Osoby, które tworzą gry w pojedynkę, korzystają z gotowych silników, bądź próbują pisać własne. Najpopularniejszymi gotowymi silnikami są Unity oraz Unreal Engine.

RPG Maker wydany przez firmę Degica jest jednym z programów, który pozwala tworzyć proste gry komputerowe, nie wymagając znajomości języka programowania. Jeżeli twórca zapoznał się z programem i nie chce go zmieniać, musi nauczyć się JavaScript aby tworzyć gry o bardziej skomplikowanych mechanikach.

W 1998r zadebiutowała gra Unreal wyprodukowana przez Epic MegaGames i Digital Extremes, która powstała na pierwszej wersji silnika Unreal Engine I. W 2006r 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 rozpoczęć przygodę z tworzeniem gier. Spora liczba twórców wybiera Unreal Engine ze względu na możliwość kodowania na klockach, które służą do kodowania wizualnego.

Ten moduł wykorzystuje się głównie do prototypowania gier, co nie zmienia faktu, że można tak stworzyć całą grę nie pisząc kodu.

Podobnym do RPG Maker jest następny program, GameMaker stworzona przez firmę YoYo Games. Pierwsze wydanie ukazało się w 1999r. Jest to środowisko do tworzenia gier i programów komputerowych, w którym nie jest wymagana znajomość języka programowania. Ponieważ program jest przeznaczony dla osób nie znających się na programowaniu, GameManager posiada wiele gotowych funkcji, przystosowanych pod konkretne zachowania rozgrywki. Tak jak Unreal, posiada opcję tworzenia skomplikowanych rozwiązań poprzez łączenie funkcji wizualnie. Środowisko posiada wbudowany język skryptowy GML, tj. Game-Maker Language, który porzypomina JavaScript. Został on stworzony z naciskiem na tworzenie gier [2].

Unity ukazało się w 2005r i początkowo był wykorzystywany w filmach, architekturze i symulacjach. W 2015r trafił do darmowej dystrybucji i szybko stało się konkurencją dla Unreal Engine. Dodatkowo Unity posiada Unity Asset Store, czyli sklep z gotowymi modelami, skryptami a nawet całymi projektami gier. Inspektor, czyli okno akcji każdego obiektu jak i całego projektu, pozwala modyfikować zmienne i dodawać nowe komponenty. Zaimportowany 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. Do pisania kodu, wybrano Microsoft Visual Studio 2017, ponieważ darmowa wersja programu jest dołączana do pakietu instalacyjnego Unity. Aktualnie jedynym językiem programowania jest C#, ponieważ silnik Unity posiada gotowe metody i funkcje, które współpracują wyłącznie z tym językiem.

Unity posiada największą z ww. programów ilość platform do eksportowania gry, konsole takie jak XBOX, PlayStation, komputery stacjonarnych z Windowsem, Mac'iem czy Linuxem a także telefony z iOS bądź Androidem. Do realizacji projektu wybrano Unity, ponieważ pozwala on na stworzenie takiej samej gry jak w Unreal, przy mniejszym wkładzie czasowym.

4. Metoda zrealizowania pracy

Realizacja pracy opiera się na opracowaniu kroków, które należy podjąć aby stworzyć świat gry w stylu low-poly. Aby zaprezentować stworzone elementy, został stworzony projekt gry komputerowej w Unity.

4.1. Narzędzia badawcze

Analizując badania własne, wybrano poniższe 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 zostanie napisany kod do gry, będzie Microsoft Visual Studio 2017.
- ostatnim narzędziem będzie Inno Setup, czyli program do tworzenia plików instalacyjnych.

4.2. Technika tworzenia obiektów 3D

4.2.1. Oparcie na zdjęciach referencyjnych

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, które 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 uwydawnienia 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 projektu użyto zdjęć poglądowych dla beczki, pojazdu, oraz słupka drogowego.



Rysunek 4.1: Zdjęcie poglądowe beczki Castrol.

Źródło: indiamart.com | Castrol Oil Barrel



Rysunek 4.2: Zdjęcie poglądowe samochodu McLaren 570S.

Źródło: kindpng.com | McLaren 570S side view

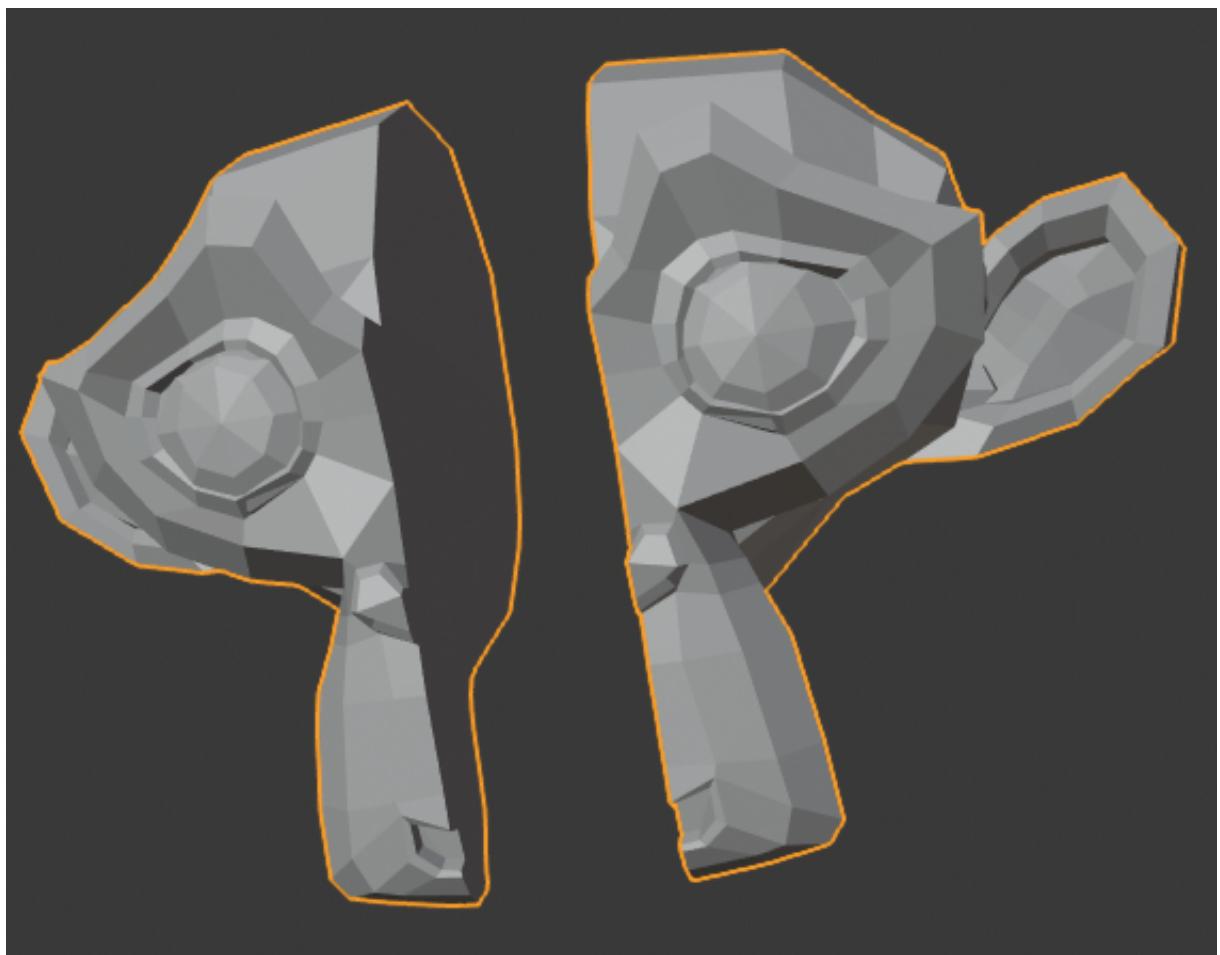


Rysunek 4.3: Zdjęcie poglądowe słupka drogowego.

Źródło: flickr.com

4.2.2. Modyfikatory obiektów

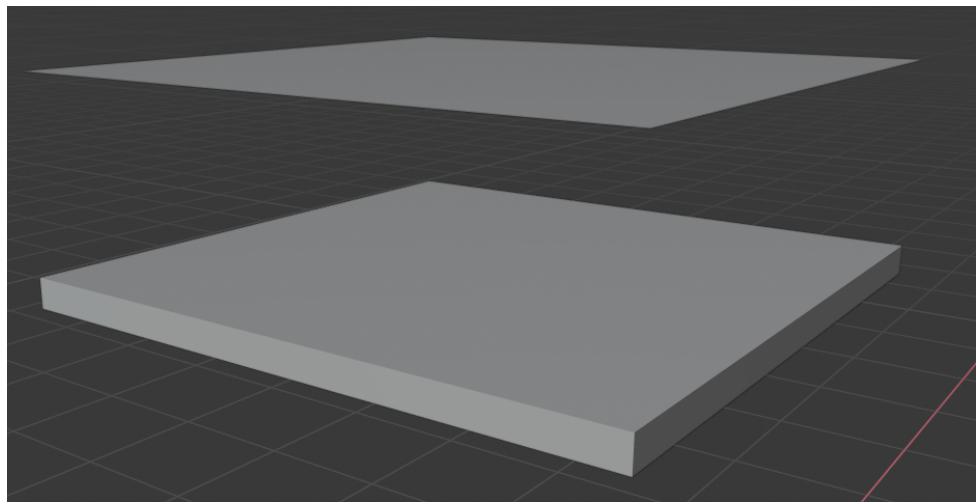
Istnieje wiele narzędzi w Blenderze, które pomagają uzyskać porządkany przez nas efekt. Jednym z nich jest Mirror". Uprzednio usuwając jedną połowę obiektu, dodając ten modyfikator na obiekt, otrzymuje się odbicie lustrzane ukazujące dwie identyczne połówki. Następnie modyfikując jedną stronę obiektu, ukazuje się identyczny wynik po drugiej stronie.



Rysunek 4.4: Przykładowe zastosowanie modyfikatora Mirror.

Źródło: Opracowanie własne

Solidify służy do pogrubiania obiektów bez objętości. Płaskie elementy są pogrubiane dzięki czemu otrzymuje dodatkową geometrię, a dalsze modyfikacje pozwalają uzyskać zupełnie inny wynik w przypadku płaskich obiektów.

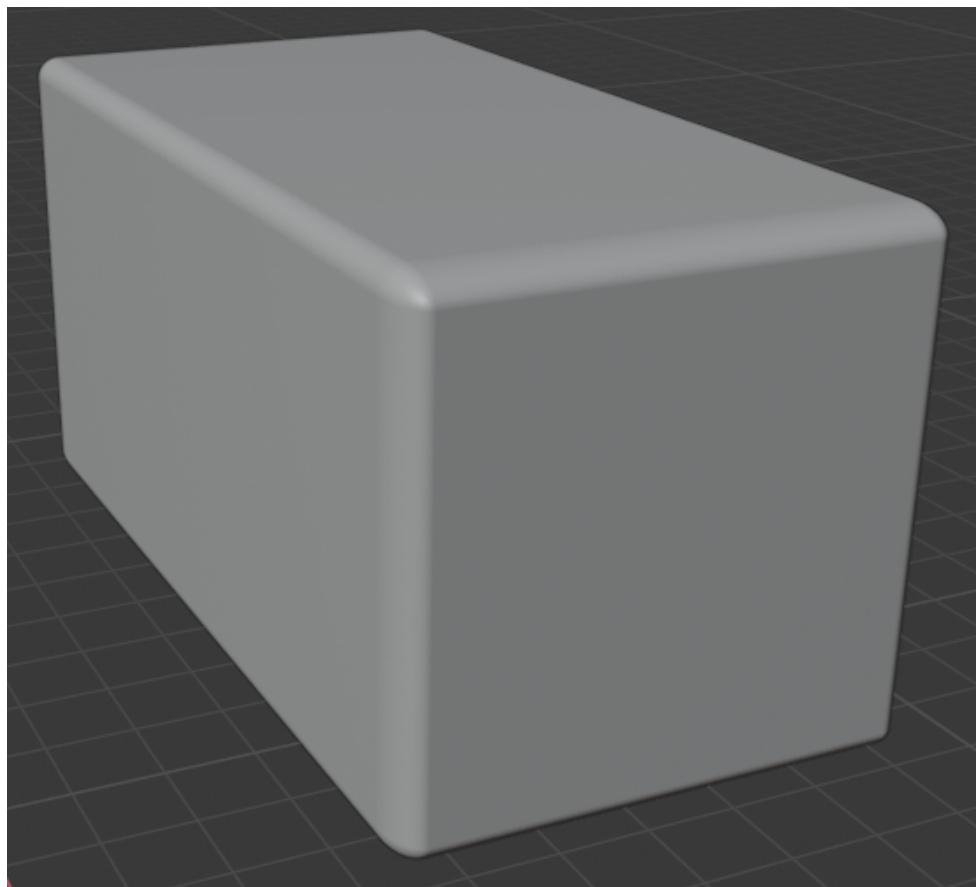


Rysunek 4.5: Na górze kwadrat bez modyfikatora. Na dole z modyfikatorem Solidify.

Źródło: Opracowanie własne

Bevel jest jednym z najlepszych narzędzi umieszczonych w Blenderze.

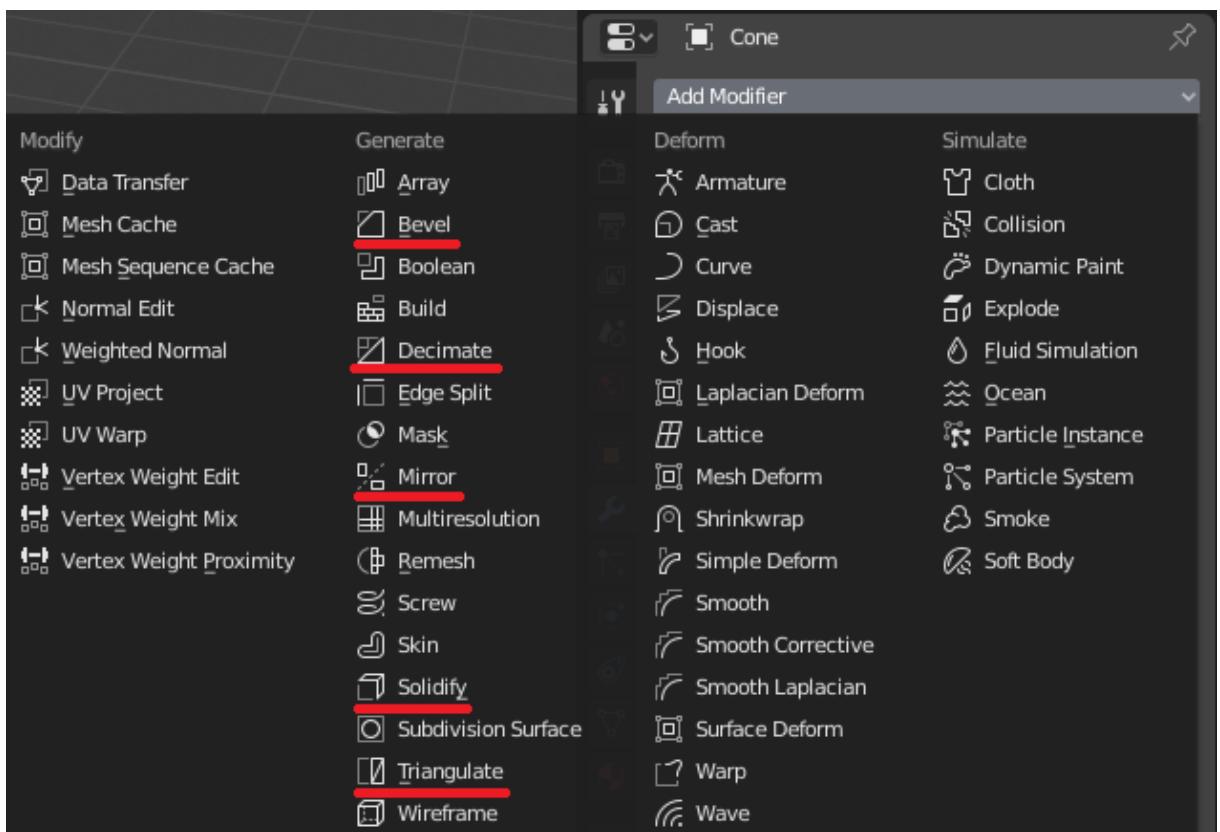
Używając go, jesteśmy w stanie uzyskać efekt ściętych bądź zaokrąglonych krawędzi w zależności od podanej przez nas wartości.



Rysunek 4.6: Przykładowe zastosowanie modyfikatora Bevel.

Źródło: Opracowanie własne

„Decimate” jest jednym z narzędzi pozwalających ograniczyć ilość punktów, czyli vertexów oraz ścian siatki meshowej, przy zachowaniu proporcji oraz kształtu obiektu. Podczas gdy obiekt zawiera kilka tysięcy poligonów, używa się tego modyfikatora aby zmniejszyć liczbę ścian, dzięki czemu uzyskuje się efekt low-poly. Możliwym jest zmniejszenie siatki meshowej o dany procent, ponieważ skala wartości zmian jest określona od 0 do 1 z dokładnością do czterech miejsc po przecinku. Ustawiając zmienną na 0.5 jesteśmy w stanie zmniejszyć liczbę poligonów o dokładnie 50% [3]. Ponieważ pojazd jak i przeszkody zostały zbudowane od podstaw, nie było potrzeby zmniejszać ilości vertexów. W przypadku gdyby zostały użyte gotowe modele zawierające dużą ilość ścian, zostałby użyty modyfikator decimate.

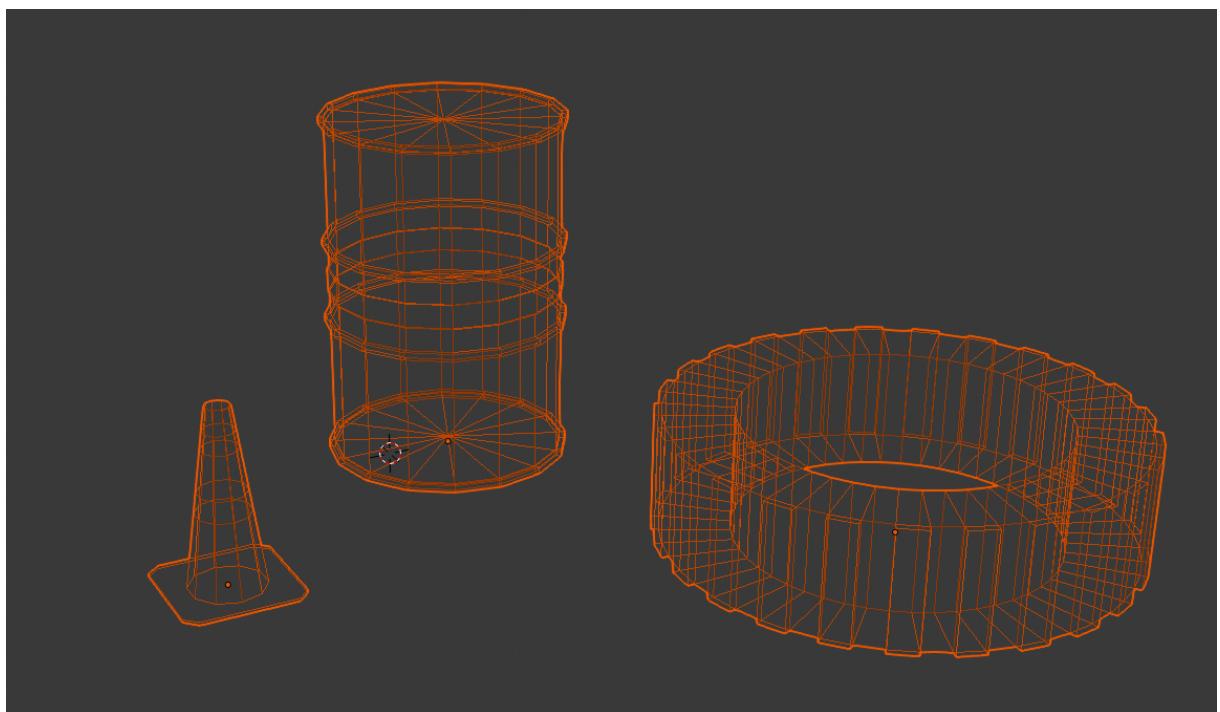


Rysunek 4.7: Zrzut ekranu z listą dostępnych modyfikatorów. Podkreślone zostały modyfikatory, które zostały użyte do stworzenia modeli pojazdu i przeszkód.

Źródło: Opracowanie własne

4.2.3. Wireframe czyli siatka meshowa

Tworząc model który ma być wykonany w stylu low-poly, najważniejszym jest pierwsze spojrzenie. Obiekt o siatce złożonej tylko z poligonów wygląda gorzej niż obiekt składający się z trisów i poligonów. Jeżeli obiekt ma być animowany, dobrym pomysłem jest użycie modyfikatora „Triangulate”. Modyfikuje on siatkę meshową obiektu, przerabiając każdy poligon na dwa trisy zachowując przy tym geometrię obiektu. Podczas animowania obiektów, trisy zachowują się zdecydowanie lepiej od poligonów, które podczas np. przesuwania ręki, tworzą nienaturalne zagięcia. Ponieważ przeszkody jak i pojazd nie posiadają animacji opony i słupek drogowy dalej posiadają siatkę złożoną tylko z poligonów.

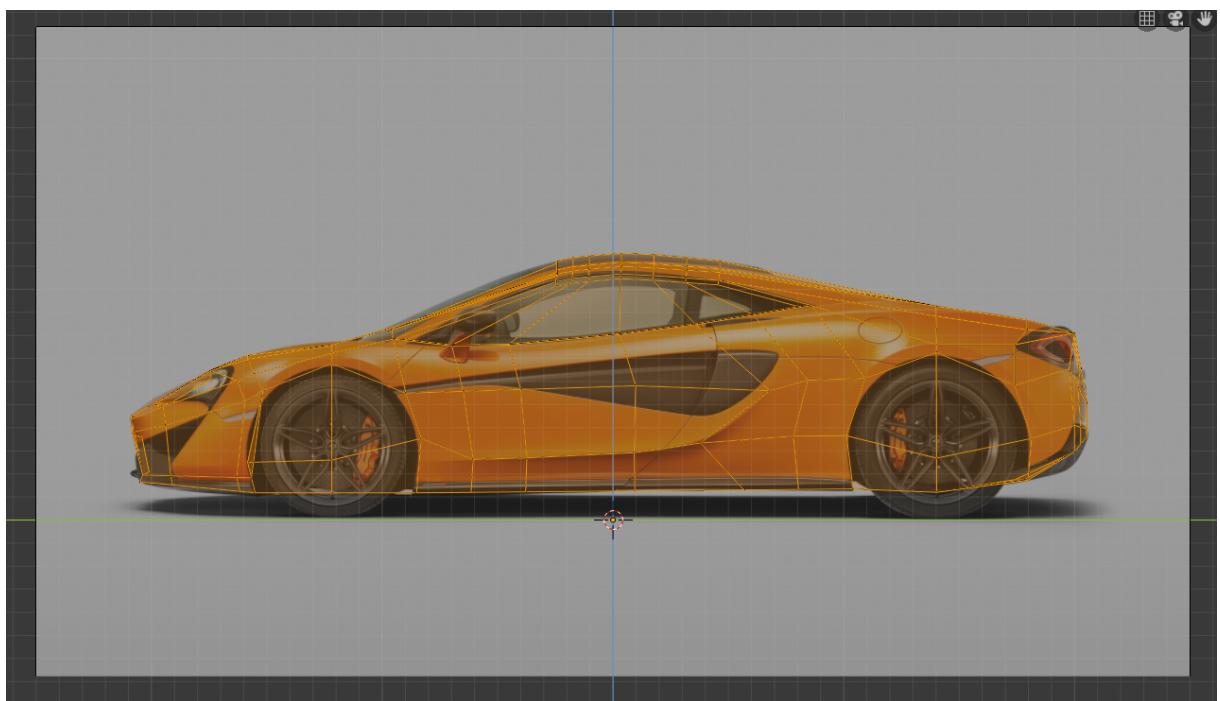


Rysunek 4.8: Siatka meshowa przeszkód.

Źródło: *Opracowanie własne*

4.2.4. Tworzenie modelu 3D pojazdu

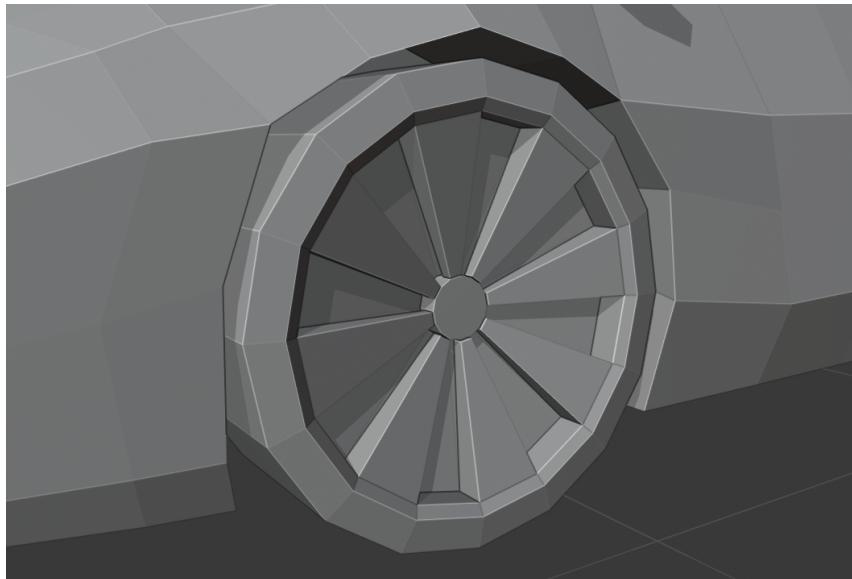
Tworząc obiekt zazwyczaj zaczyna się od zwykłej kostki sześciennnej, którą następnie modyfikuje się poprzez przesuwanie vertexów, krawędzi bądź całych ścian obiektu. Blender posiada niesamowitą umiejętność łączenia funkcji. Przykładowo, korzystając ze skrótów klawiszowych, twórca jest w stanie wysuwać ścianę obiektu a następnie ją skalować nie przerywając wysuwania. **Pojazd** został stworzony korzystając ze zdjęcia poglądowego McLaren'a 570S ustawionego w widoku side view (bocznym) oraz własnej wyobraźni, aby uzyskać kształt przypominający bryłę tego samochodu. Za pomocą narzędzia Extrude dodano dodatkowe ściany, dzięki którym możliwym jest uzyskanie kształtu potrzebnego do odwzorowania modelu rzeczywistego. Jest to jedno z najważniejszych narzędzi, dzięki któremu jesteśmy w stanie uzyskać geometrię, która pozwoli stworzyć dokładniejszy obiekt, który będzie posiadał dodatkowe detale. Następnie można było ustawić krawędzie obiektu nadając mu kształt przypominający ten na zdjęciu.



Rysunek 4.9: Siatka modelu w widoku Wireframe pokazująca ułożenie krawędzi pojazdu na zdjęciu referencyjnym.

Źródło: Opracowanie własne

Ustawiając wszystkie krawędzie, aby pasowały do zdjęcia referencyjnego, umiejscowiono krawędzie siatki w taki sposób, aby możliwym było wysunięcie wszystkich ścian bocznych poza miejscem na koła. W ten sposób za pomocą narzędzia Extrude, uzyskano nadkola pojazdu. Następnym krokiem, było stworzenie koła. W środku nadkola, dodano Circle, czyli okrąg. Bazowo okrąg posiada 32 ściany, lecz zmniejszono ich liczbę do 16 za pomocą narzędzia dodawania okręgu. Następnie dopasowano go do rozmiaru opony, za pomocą narzędzia skalowania. Również przy pomocy narzędzia Extrude, stworzono walec bez ścian zamykających go. Przy pomocy Extrude, którego przesunięcie zostało anulowane poprzez prawy przycisk myszy, stworzono dwie linie, które są umieszczone na płaszczyźnie koła. Następnie zostały one zeskalowane do centralnej linii opony, a przy pomocy Extrude zmniejszono do rozmiarów środka koła i wysunięto na zewnątrz, aby uzyskać efekt prawdziwej felgi.



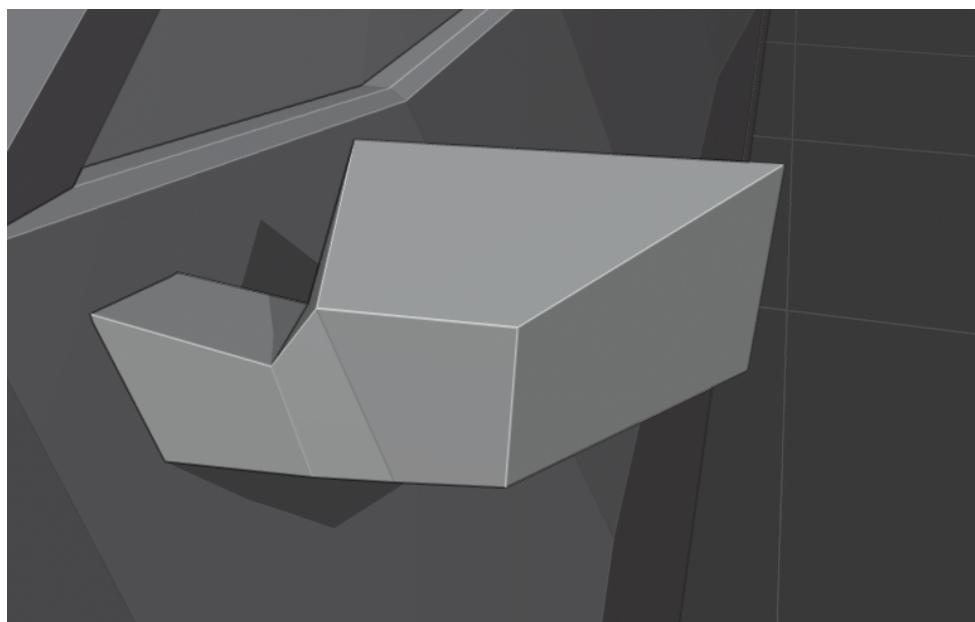
Rysunek 4.10: Ilustracja przedstawiająca koło pojazdu.

Źródło: Opracowanie własne

Wybierając co drugą ścianę felgi, uzyskano 8 elementów, a następnie za pomocą Extrude, którego przesunięcie zostało anulowane oraz narzędzia skalowania, dopasowano uzyskane ramiona do środka felgi, aby imitować połączenie z wystającym walem. Aby uzyskać efekt takiego samego koła po drugiej stronie pojazdu użyto modyfikatora Mirror, który

odbił obiekt na płaszczyźnie Y. Następnie koło z modyfikatorem Mirror, który nie jest jeszcze zaakceptowany, zduplikowano aby dodać koła na tylną część pojazdu.

W każdym pojeździe muszą być lusterka boczne. Aby stworzyć lusterko, wykorzystano plane, czyli płaski kwadrat, który następnie umieszczono pod karoserią drzwi i użyto narzędzia Extrude. Dzięki możliwości przesuwania vertexów oraz skalowania, twórca jest w stanie uzyskać kształt lusterka, który w głównej mierze przypomina prawdziwe lusterko sportowego samochodu.

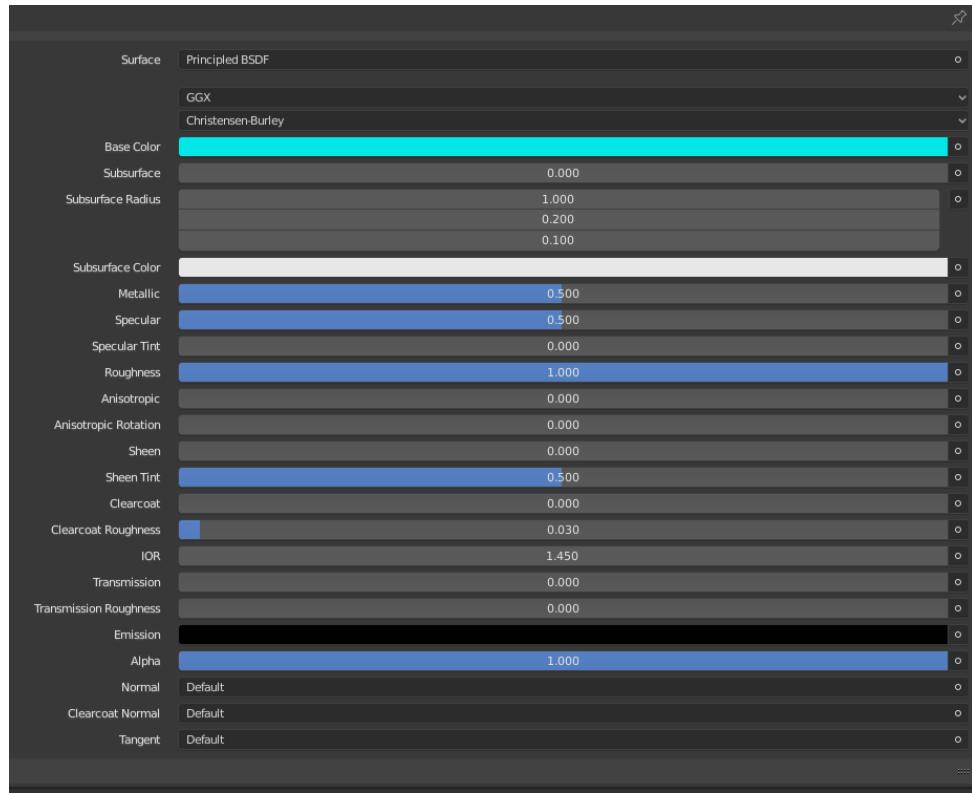


Rysunek 4.11: Ilustracja przedstawiająca lusterko boczne umieszczone na drzwiach pojazdu.

Źródło: Opracowanie własne

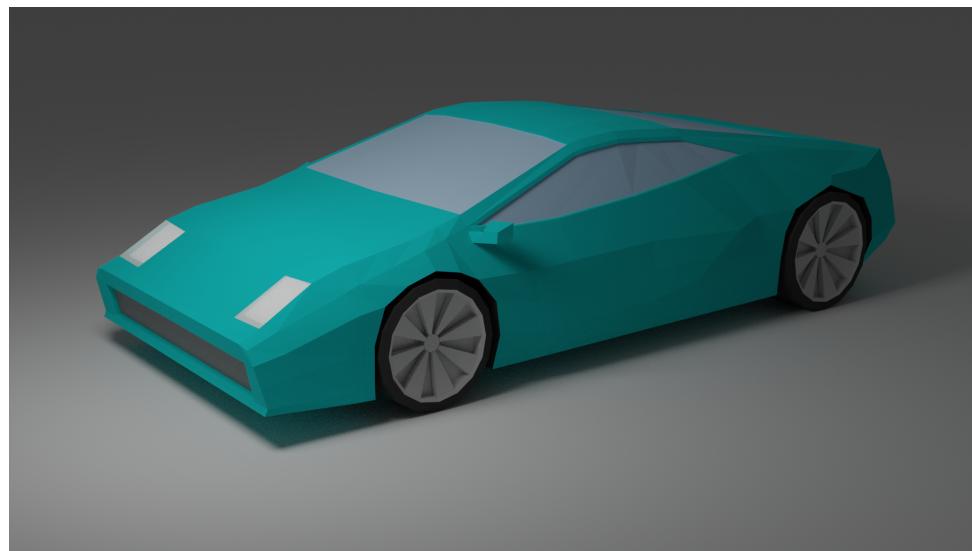
Aby obiekt posiadał kolory, wykorzystano zawartą w Blenderze opcję materiałów. Na dany obiekt, można dodać nieskończonie wielką liczbę materiałów. Każdy z nich może posiadać różne właściwości, takie jak szorstkość, reflektynośc odbić czy też emisja koloru. Zdecydowano, że pojazd będzie posiadał jasny odcień koloru niebieskiego z maksymalną szorstkością oraz wyglądem metalicznym zwiększonym do 0.5. Aby dodać materiał na obiekt, należy wybrać wszystkie vertexy, które tworzą ściany, a następnie zatwierdzić je poprzez przycisk Assign. Przypisuje on danym elementom modelu jeden materiał.

Ponieważ Koła są również połączone z modelem samochodu, automatycznie dostały ten sam kolor. Koniecznym jest powtórzenie poprzedniego kroku, aby dodać inny materiał na opony oraz felgi.



Rysunek 4.12: Okno modyfikacji materiału nałożonego na siatkę pojazdu.

Źródło: Opracowanie własne



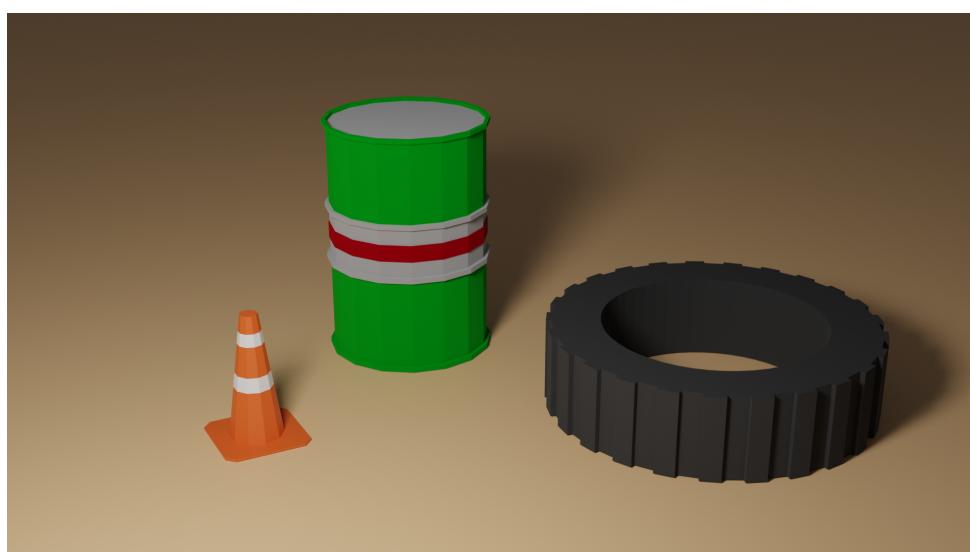
Rysunek 4.13: Render gotowego pojazdu stworzonego na potrzeby projektu.

Źródło: Opracowanie własne

4.2.5. Tworzenie modelu 3D przeszkód

Aby stworzyć **słupek drogowy**, wykorzystano cylinder, którego liczbę ścian ustawiono na 10. Następnie zmniejszono rozmiar górnej płaszczyzny za pomocą narzędzia skalowania, dzięki czemu możliwym było uzyskanie stożkowego wyglądu. Dodając do sceny plane, czyli płaski kwadrat, została stworzona podstawa słupka. Aby uzyskać efekt ściętych rogów kwadratu, użyto modyfikatora Bevel, który zaokrąglą krawędzie. W przypadku płaskiego kwadratu, modyfikator ten nie daje żadnego efektu. Modyfikator Bevel posiada opcję, która pozwala ściąć rogi, czyli z jednego vertexa tworzy dwa i odpowiednio więcej w przypadku zwiększenia ilości cięć. Następnie użyto modyfikatora Solidify, który powoduje pogrubienie płaszczyzny nadając jej potrzebnej geometrii i objętości. Aby wyeksporować modele jako plik .fbx należy zaakceptować obydwie modyfikacje w odpowiedniej kolejności, od góry do dołu. Aby połączyć cylinder z kwadratem użyto skrótu klawiszowego Ctrl + J, który łączy zaznaczone obiekty w jeden.

Tworząc **beczkę** oraz **oponę**, wykorzystano identyczne narzędzia jak w przypadku słupka drogowego. Różnicą był ilość ścian obiektu oraz sposób w jaki były przesuwane krawędzie boczne modelu.



Rysunek 4.14: Przeszkody stworzone w Blenderze.

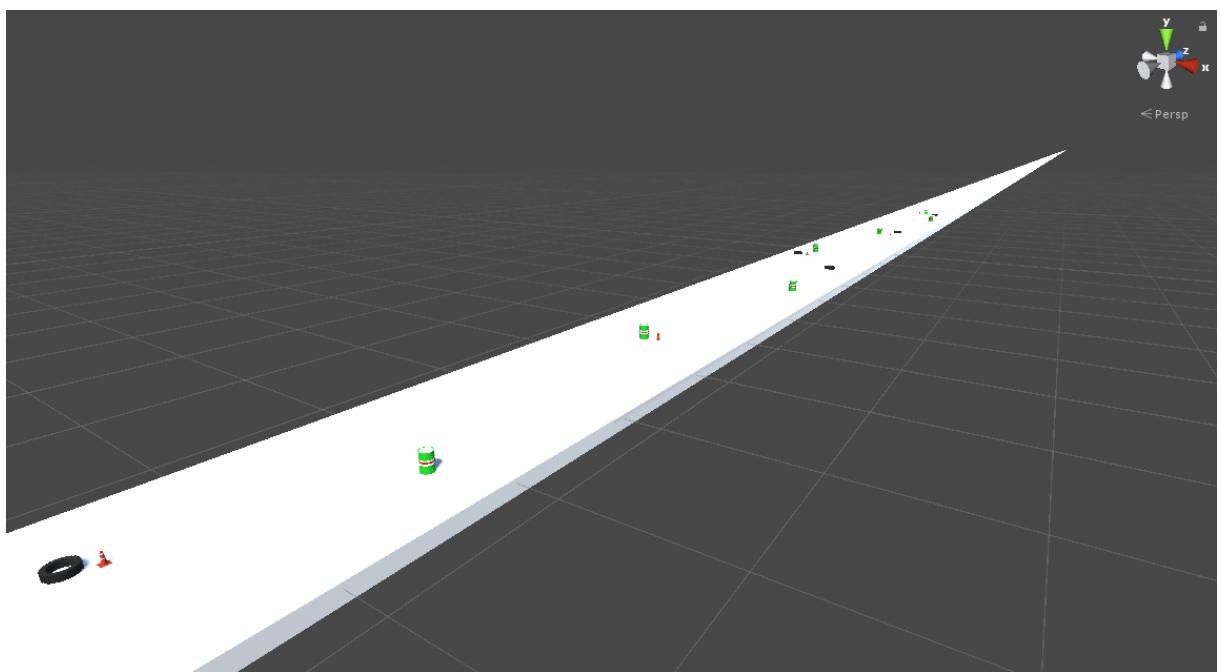
Źródło: *Opracowanie własne*

4.3. Środowisko Unity

Obiekty stworzone w Blenderze, zostały wyeksportowane jako plik typu fbx, który idealnie przechowuje dane obiektu, takie jak siatka meadowa, materiały nałożone na obiekt, oraz animacje. Niestety skala obiektów nie jest zachowana, a modele posiadają skalę 100 jednostek Unity. Aby przeszkody nie wisiały w powietrzu, została stworzona płaska powierzchnia o rozmiarach 15x3x1000 jednostek Unity. Następnie umieszczono wszystkie obiekty w scenie. Korzystając z zawartych w programie komponentów nadano każdemu modelowi komponent Mesh Collider, który odpowiada za wykrywanie kolizji z innymi obiektami, oraz RigidBody będący głównym elementem fizyki obiektów. Aby obiekty posiadały rozsądne rozmiary, zostały zmodyfikowane ich gabaryty. Następnie zmodyfikowano ich masy w komponencie RigidBody tak, aby pojazd się od nich odbijał. Po dokonaniu zmian zostały stworzone prefaby. Prefabem nazywany jest ten sam obiekt, który po ponownym umieszczeniu w scenie posiada te same właściwości. Dodatkowo gotowe prefaby można używać w późniejszych etapach, przykładowo losowo generowany poziom, który w wyznaczonych miejscach ustawia losowy obiekt. Ustawiając obiekty na powierzchni, stworzono 5 różnych poziomów, które różnią się trudnością, odpowiednio od pierwszego do piątego. Każdy z nich zawieranąną liczbę przeszkód ustawionych w innych miejscach.

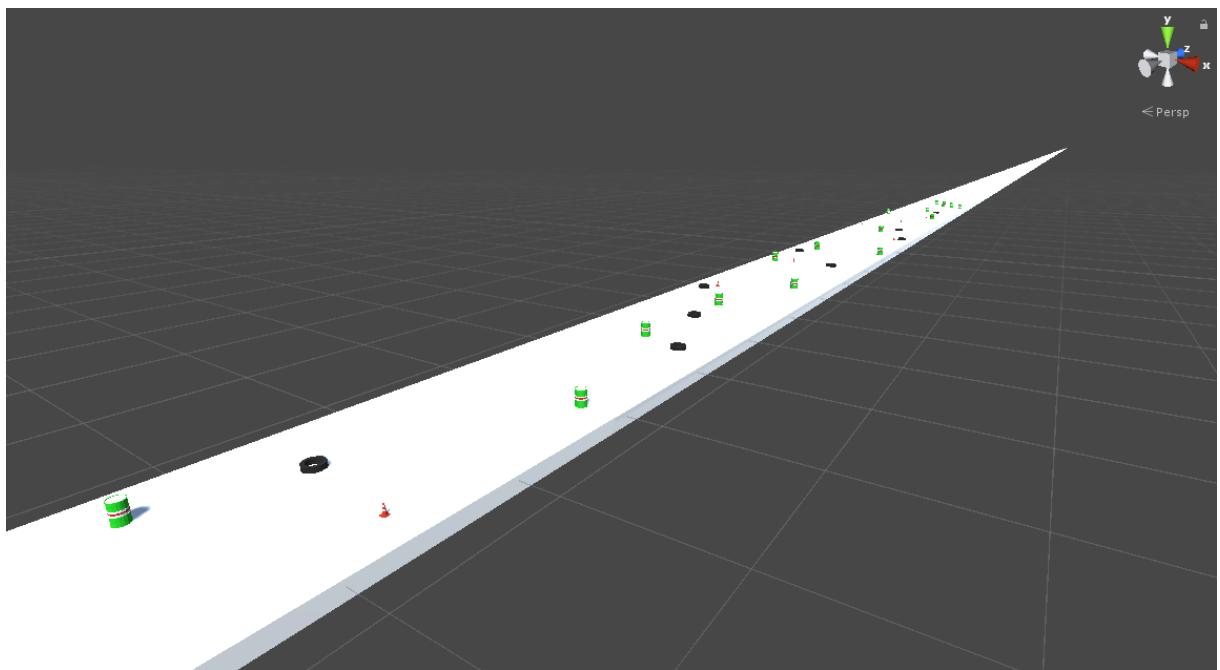
4.3.1. Scenariusz rozgrywki

Po uruchomieniu gry, ukazuje się menu główne na którym znajduje się nazwa gry, przycisk do włączenia gry, oraz do wyjścia z programu. Głównym zadaniem gracza jest przejechanie samochodem przez pięć poziomów bez dotknięcia żadnej przeszkody. W przypadku kiedy pojazd udeży przeszkodę, poziom na którym polegliśmy jest ładowany od nowa z opóźnieniem dwu sekundowym, aby gracz mógł zobaczyć kolizję pojazdu z przeszkodą. Jeżeli graczowi uda się przejeść wszystkie pięć poziomów ukazuje mu się ekran końcowy na którym widnieją gratulacje oraz przycisk do wyłączenia programu.



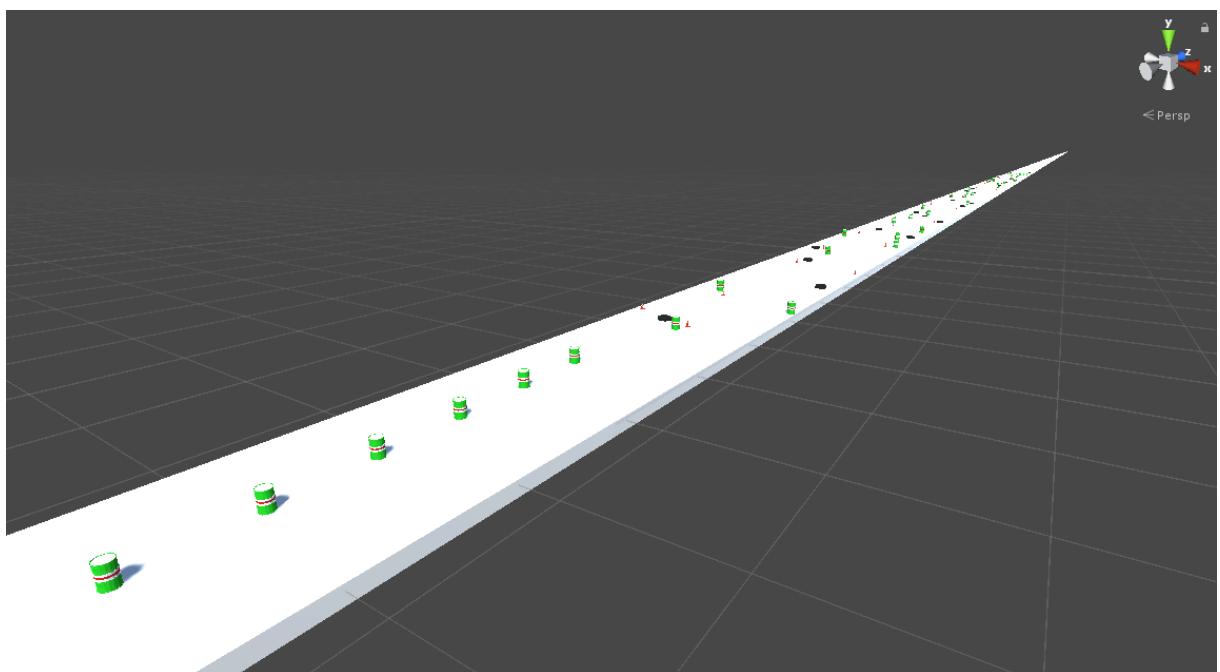
Rysunek 4.15: Ustawienie przeszkód na poziomie 1.

Źródło: Opracowanie własne



Rysunek 4.16: Ustawienie przeszkód na poziomie 3.

Źródło: *Opracowanie własne*

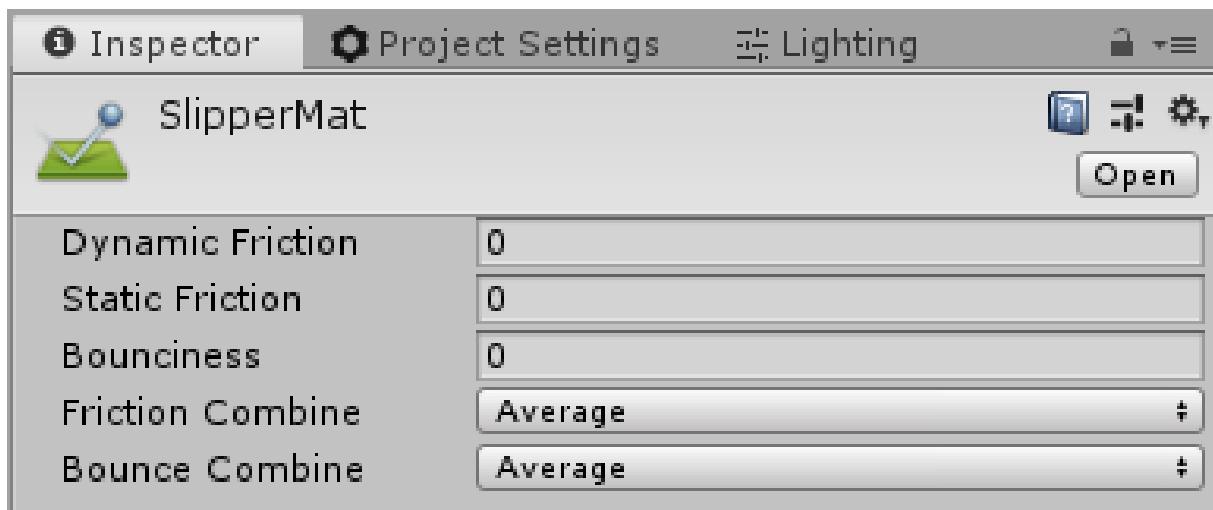


Rysunek 4.17: Ustawienie przeszkód na poziomie 5.

Źródło: *Opracowanie własne*

4.3.2. Materiał poślizgowy

Pojazd podczas poruszania się wykrywa tarcie pomiędzy sobą a powierzchnią planszy. Aby uniknąć nieuchcianych podskoków pojazdu, stworzono w Unity materiał specjalny, który wpływa na fizykę komponentów RigidBody umieszczonych na pojazdzie oraz przeszkodach. Zmienne Dynamic Friction oraz Static Friction zmieniono na 0, a następnie dodano materiał SlipperMat na planszę gry.

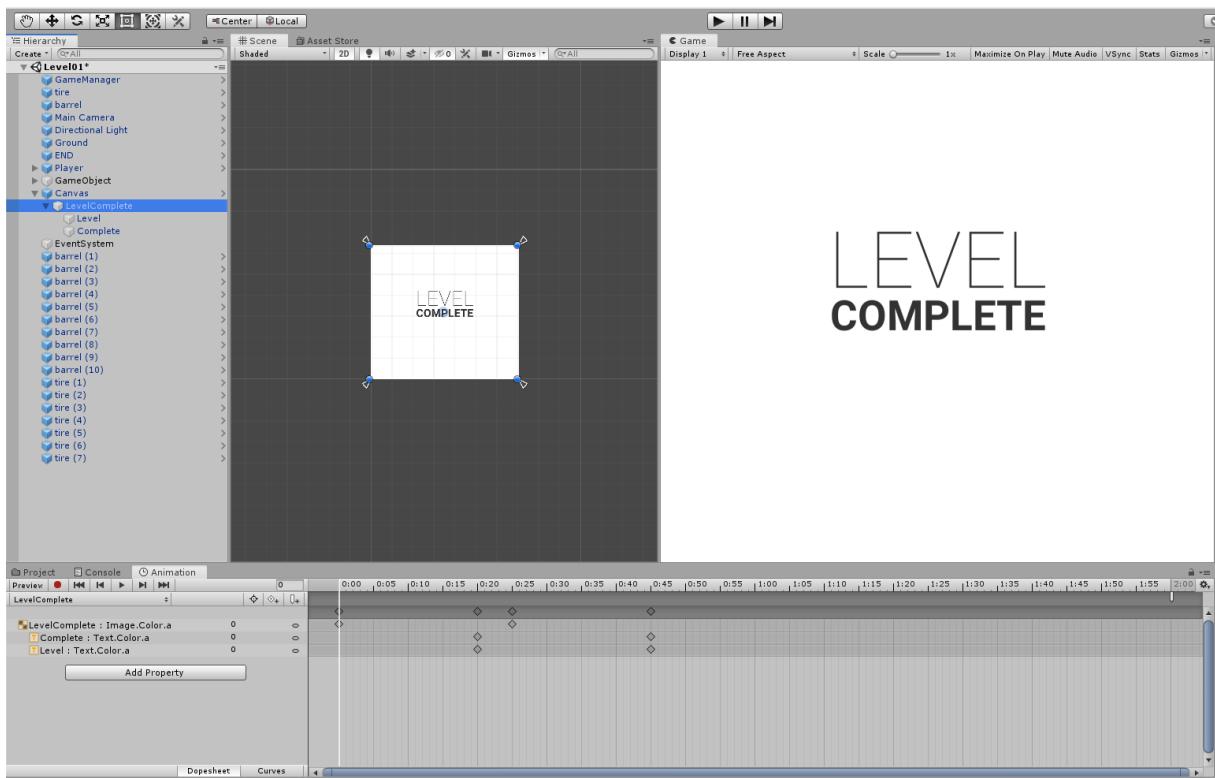


Rysunek 4.18: Materiał SlipperMat, który został dodany na powierzchnię planszy.

Źródło: *Opracowanie własne*

4.3.3. Animacje

Przejścia pomiędzy poziomami będą zawierały krótką i prostą animację wyświetlaną 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 4.19: Zrzut ekranu pokazujący oś czasu oraz wynik animacji.

Źródło: Opracowanie własne

4.4. Programowanie skryptów

4.4.1. Poruszanie pojazdem

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 osi X, Y oraz Z, z określoną mocą zakodowaną w zmiennej *sidewayForce*.

```
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 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(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();
        }
    }
}
```

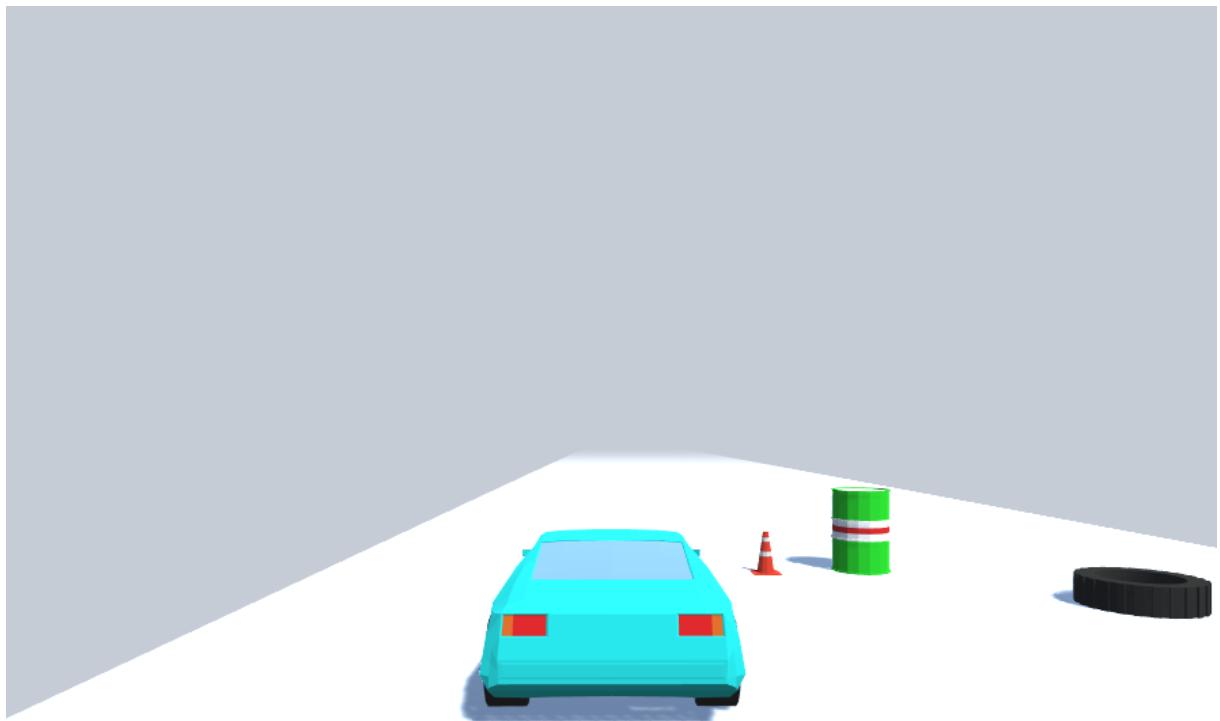
Rysunek 4.20: Skrypt PlayerMovement obsługujący poruszanie się pojazdu po poziomie.

Źródło: Opracowanie własne

Ponieważ nie każdy komputer posiada taką samą prędkość liczenia klatek, do zmiennej określającej prędkość pojazdu dodano *Time.deltaTime*. Wartość ta, określa czas pomiędzy klatkami, które uzyskuje komputer. Dzięki takiemu rozwiązaniu, rozgrywka będzie wyglądać tak samo na komputerze z tego roku jak i na komputera z przed dekadą [4]. Ponieważ na pojazd, został nałożony komponent *RigidBody*, obiekt posiada swoją masę. *RigidBody* jako komponent określający fizykę, oblicza zachowanie każdego obiektu w sposób przypominający w dużym stopniu prawdziwe prawa fizyki. Na każdy obiekt działa siła przyciągania ziemskiego zwiększonej o masę. Takie rozwiązanie wpływa negatywnie na pojazd podczas ruchu w przód oraz skręcania. Zdecydowano dodać czwartą zmienną. *ForceMode.VelocityChange* dodaje natychmiastową zmianę prędkości ciała, pomijając z obliczeń jego masę. Rozwiązanie to przydaje się w przypadku, kiedy na scenie znajduje się wiele obiektów, które poruszają się z tą samą prędkością, ale posiadają inne rozmiary i masy. Okazałoby się, że przykładowy rower, który jedzie z prędkością np. 50 km/h będzie jechał szybciej od dużej ciężarówki, która również porusza się z taką samą prędkością [5].

FindObjectOfType służy do przeszukiwania katalogów gry pod względem tego co zostało napisane w nawiasach ostrokałtnych. Ponieważ w skrypcie *GameManager* została napisana funkcja *EndGame()* *FindObjectOfType* szuka *GameManagera*, aby następnie uzyskać dostęp do funkcji, która musi być ustawiona na public, inaczej nie zostanie uruchomiona, a Unity pokaże błąd.

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



Rysunek 4.21: Zdjęcie ukazujące pojazd, który zboczył z trasy, efekt skryptu ruchu.

Źródło: *Opracowanie własne*

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

Źródło: *Opracowanie własne*

4.4.2. Obsługa kolizji

Kolejnym zadaniem 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 4.23: Skrypt PlayerCollision obsługujący kolizje pomiędzy pojazdem a przeszkodami.

Źródło: *Opracowanie własne*



Rysunek 4.24: Przykładowa kolizja, która wyrzuciła pojazd do góry.

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 która musi być ustalona jako publiczna. Stworzono typ logiczny bool *gameHasEnded*, który przybiera dwie wartości, prawda lub fałsz, którego wartość zmieniana jest przy pierwszym uruchomieniu funkcji, aby upewnić 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, wykorzystano metodę *Invoke* która po dodaniu nazwy funkcji, oraz czasu w sekundach, uruchamia funkcję w określonym przez twórcę czasie. Została stworzona zmienna *restartDelay*, którą twórca może modyfikować z okna Inspektora. Aby przejść na następny poziom, gracz musi przejechać pomiędzy przeszkodami tak żeby ich nie dotknąć. W przypadku porażki poziom zostanie załadowany od nowa.

```
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 4.25: Skrypt *GameManager* zajmujący się restowaniem poziomów.

Źródło: *Opracowanie własne*

4.4.3. Ładowanie poziomów

Poziom jest już możliwy do przejścia, jednakże trzeba załadować kolejny. Na końcu trasy każdego z poziomów umieszczono kostkę o szerokości 15 jednostek Unity, która jest niewidoczna i służy jako przełącznik uruchamiający 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. *gameCompleteUI* włącza animację, która z poziomu Unity została manualnie wyłączona, ponieważ podczas działania programu, gracz nie widziałby planszy, tylko animacje.

```
using UnityEngine;

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

Rysunek 4.26: Skrypt EndTrigger nawiązujący do GameManager.

Źródło: *Opracowanie własne*

Aby gra rozpoczęła się tylko i wyłącznie w momencie kliknięcia przycisku „START” został napisany krótki skrypt. Na scenie Menu został umieszczony przycisk do którego został podpięty wcześniej użyty skrypt LevelComplete, ponieważ będzie miał takie same zastosowanie jak podczas uruchamiania następnego poziomu. Zawiera on kod ładujący następną scenę umieszczoną w Buildzie gry. Build określa kolejność scen poprzez nadawanie im numerów indeksów od 0 wzwyż.

```

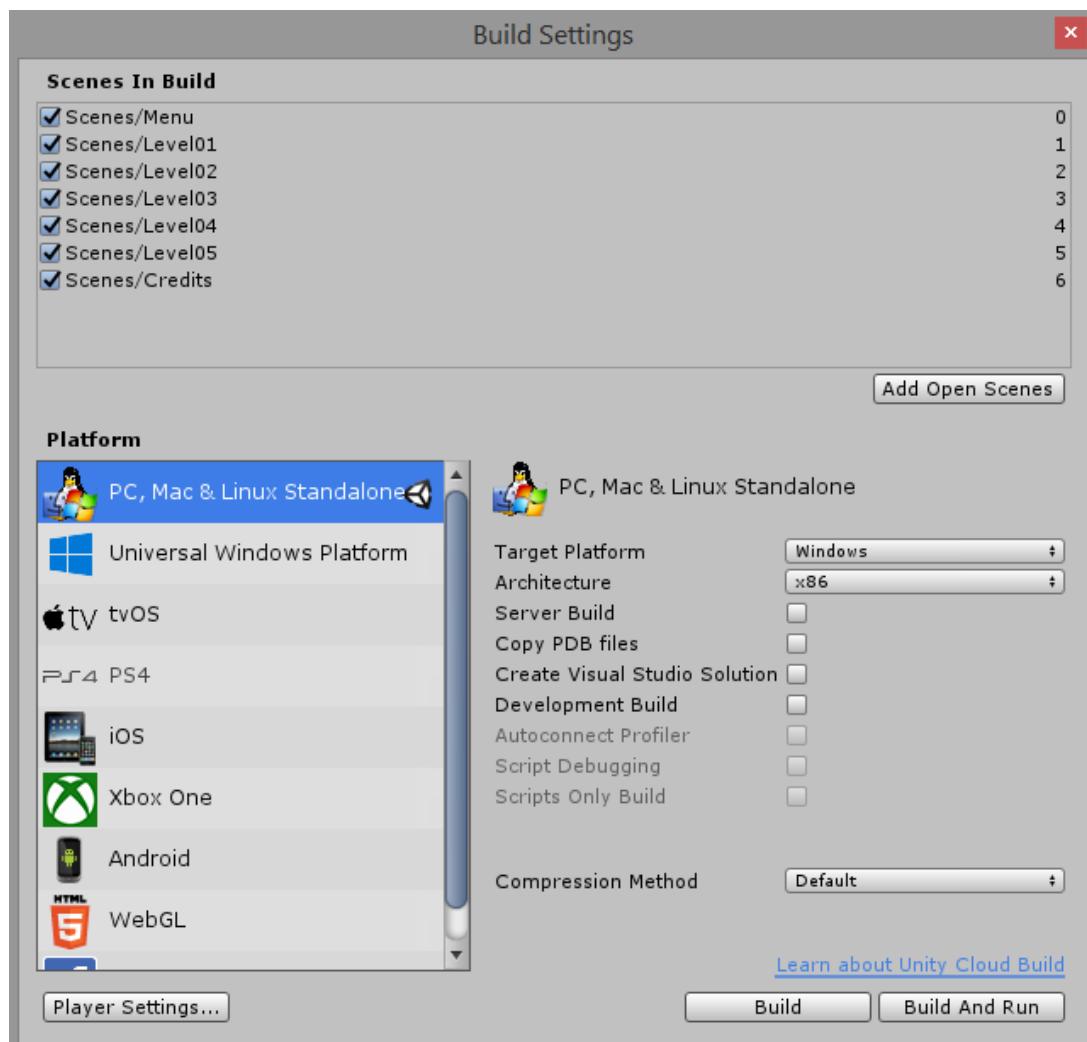
using UnityEngine;
using UnityEngine.SceneManagement;

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

```

Rysunek 4.27: Skrypt LevelComplete ładujący kolejny poziom.

Źródło: Opracowanie własne



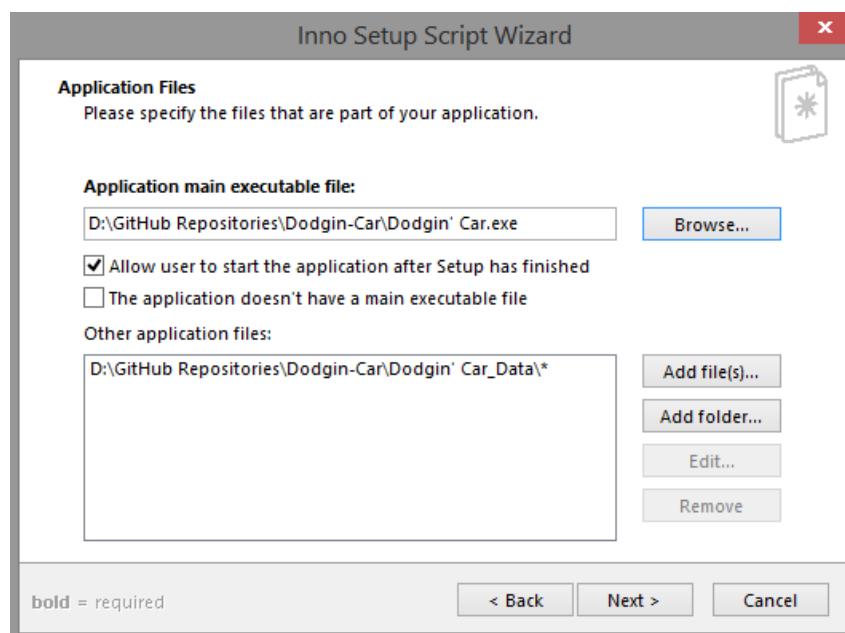
Rysunek 4.28: Okno Build pozwalające określić kolejność poziomów.

Źródło: Opracowanie własne

4.5. Przygotowanie gry do publikacji

Gra jest już gotowa, jednakże nie jest stworzony żaden plik .exe, który będzie uruchamiał program. W oknie build, należy upewnić się, że wszystkie sceny są ustawione w prawidłowej kolejności. Następnie wymaganym jest wybranie platformy na którą chcemy stworzyć grę i potwierdzić przyciskiem Build. Pojawi się okienko w którym musimy wybrać katalog, w którym zostanie stworzona gra, która będzie skończonym produktem.

W przypadku gdyby twórca chciał się podzielić grą z kimś znajomym, wystarczy ją spakować w .zip i wysłać. Aby produkt wyglądał na profesjonalny należy stworzyć plik instalacyjny. Do tworzenia małych plików instalacyjnych, idealnym narzędziem będzie Inno Setup. Jest to oprogramowanie służące do tworzenia plików instalacyjnych oparty na skryptach. Przez cały proces tworzenia pliku instalacyjnego prowadzi właśnie Inno, w jak najprostszy sposób, aby nie zmylić twórcy. W odpowiednim miejscu musimy wskazać ścieżkę do naszego pliku .exe, odpowiednio dla plików schowanych w folderze Data.



Rysunek 4.29: Okno wyboru ścieżki dostępu do pliku exe oraz folderu z danymi gry.

Źródło: Opracowanie własne

5. Podsumowanie

Proces tworzenia gier komputerowych wymaga od twórcy poświęcenia czasu na zapoznanie się z historią stylu graficznego oraz jego wymaganiami. Tworząc grę low-poly należy pamiętać, aby modele 3D posiadały siatkę modelu która będzie wykonana poprawnie. Ważnym elementem tworzenia gier w konkretnym stylu jest zebranie informacji dotyczących wymagań topologii modeli. Charakterystyczne cechy stylu low-poly to siatka modelu składająca się z trisów i poligonów prostokątnych. Obiekt nie może posiadać dużej ilości poligonów, ponieważ jak sama nazwa wskazuje, nie będzie to low-poly. Małe detale trzeba ograniczyć do minimum, bądź całkowicie wykluczyć a kolory obiektów muszą się wyróżniać, aby modele nie nakładały się na siebie.

Tworząc obiekt, którego pierwowzór ma odzwierciedlenie w rzeczywistości, należy pamiętać aby stworzyć go w taki sposób, aby jak najbardziej przypominał tą rzecz, chociażby pod względem kształtu. Jedno z narzędzi Blendera, Extrude, jest najlepszym przyjacielem twórcy. Za jego pomocą jesteśmy w stanie dowolnie rozszerzać siatkę modelu dodając przy tym dodatkowej objętości. Najczęstszym sposobem na modyfikowanie obiektu jest jego skalowanie. Modyfikując poszczególne elementy obiektu, twórca jest w stanie uzyskać nowe krawędzie, które nadadzą modelowi wymaganej geometrii. Modyfikator Mirror, jest przydatny w momentach, kiedy twórca skupia się na jednej stronie modelu, aby przesuwając i modyfikując konkretną część obiektu, uzyskać idealne odbicie lustrzane. W przypadku gdy twórca chce uzyskać ciekawy efekt wizualny obiektu, zamiast tworzyć ręcznie trisy na siatce obiektu, może użyć modyfikatora Triangulate, który zmodyfikuje siatkę meshową modelu,

przerabiając każdy możliwy poligon na dwa trisy. Niestety wiąże się to z konsekwencją zwiększenia ilości poligonów w obiekcie, co może doprowadzić do komplikacji związanych z przewidywaną liczbą ścian w każdym obiekcie.

Umiejętność operowania w danym programie, który służy się do tworzenia gier, również jest przydatna, ze względu na czas. Tworząc obiekty w Blenderze, po wcześniejszym nauczeniu się programu, skrótów oraz dostępnych modyfikatorów, możliwym jest stworzenie modelu w krótkim czasie. Korzystając z silnika gry, który wymaga pisania skryptów, aby uzyskać przeróżne mechaniki rozgrywki, dodatkowo rozwijana jest znajomość języka programowania, użyteczna w następnych projektach. Otrzymanie żądanego efektu może zająć sporo czasu, jednakże podczas powstawania kolejnego projektu, jesteśmy w stanie uzyskać ten sam efekt, bądź lepszy, w o wiele krótszym czasie. Blender jako darmowe oprogramowanie często jest pomijany przez wielkie firmy zajmujące się animacjami oraz modelowaniem 3D. Jest to spowodowane brakiem wsparcia technicznego, które jest wymagane w przypadku, kiedy podczas procesu tworzenia ważnego projektu, wyłączy się program i nie można go uruchomić ponownie, bądź wszystkie dane zostały stracone. Nie zmienia to faktu, że Blender dorównuje swoim konkurentom i przedzej czy później osiągnie status jednego z najlepszych oprogramowań dostępnych za darmo na rynku.

Jak widać w rozdziale **3.4**, aby stworzyć grę wymagana jest podstawowa wiedza o języku programowania, oraz znajomość funkcji silnika Unity. Wszystkie gotowe funkcje i metody zawarte w silniku Unity są udostępnione w Unity User Manual dostępnym na stronie głównej Unity. Podczas pisania kodu ważnym jest aby zwracać uwagę na pisanie pierwszych liter. C# jest językiem programowania, który zwraca uwagę na to, czy funkcja jest napisana z dużej litery czy nie. Standardem jest, aby zmienne zaczynały się z małej litery a metody i funkcje z dużej.

Istnieją też dwie szkoły pisania długich nazw. Aby oddzielić od siebie słowa korzysta się ze snake'a albo camel'a. **to_jest_snake** czyli oddzielanie słów za pomocą znaku podkreślenia, a **toJestCamel**, czyli pisanie pierwszego słowa z małej litery, a następnie zaczynanie każdego następnego słowa z wielkiej. Należy pamiętać o konieczności przystosowania gry do każdego komputera. Do metod które wpływają na rozgrywkę, takie jak poruszanie się, należy dodać *Time.deltaTime* aby sprzęt na którym się gra, nie dawał przewagi. Poprzez *ForceMode.VelocityChange* twórca jest w stanie uzyskać element prawdziwego poruszana się dwóch różnych obiektów o tej samej prędkości. Dodatkowo ten tryb pozwala uzyskać sposób poruszania się obiektów po scenie w stylu gier typu Arcade.

Spełnione zostały wymagania użytkownika, a teza została udowodniona. Gra została stworzona w środowisku Unity w formie 3D, a modele przeszkód i pojazdu zostały wykonane w postaci low-poly. Plik instalacyjny gry został stworzony w programie Inno Setup a następnie nagrany na płytę. Założenia projektu obejmowały stworzenie modeli 3D oraz zaprogramowanie skryptów obsługujących sterowanie i mechaniki gry. Powstały trzy modele przeszkód oraz jeden pojazd. Podczas procesu programowania powstało 6 skryptów zajmujących się rozgrywką, oraz dwa obsługujące przyciski startu oraz wyjścia z gry.

Spis literatury

- [1] Fermentas Inc. Unity technologies: Unity user manual. <https://docs.unity3d.com/Manual/index.html>, 12 stycznia 2019.
- [2] Gammemaker studio 2 user manual. <https://docs2.yoyogames.com/>, 21 stycznia 2019.
- [3] Blender. Blender user guide. <https://docs.blender.org/manual/en/dev/>, 12 stycznia 2019.
- [4] Time.deltaTime. <https://docs.blender.org/manual/en/dev/>, 14 stycznia 2019.
- [5] ForceMode.VelocityChange. Blender user guide. <https://docs.blender.org/manual/en/dev/>, 14 stycznia 2019.
- [6] Bill Wagner. Microsoft: Przewodnik programowania c#. <https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/l>, 8 stycznia 2019.

Spis rysunków

3.1. Przykładowy render wyspy stworzonej w stylu low-poly	13
4.1. Zdjęcie poglądowe beczki Castrol.	18
4.2. Zdjęcie poglądowe samochodu McLaren 570S.	19
4.3. Zdjęcie poglądowe słupka drogowego.	19
4.4. Przykładowe zastosowanie modyfikatora Mirror.	20
4.5. Na górze kwadrat bez modyfikatora. Na dole z modyfikatorem Solidify.	21
4.6. Przykładowe zastosowanie modyfikatora Bevel.	21
4.7. Zrzut ekranu z listą dostępnych modyfikatorów. Podkreślone zostały modyfikatory, które zostały użyte do stworzenia modeli pojazdu i przeszkód.	22
4.8. Siatka meshowa przeszkód.	23
4.9. Siatka modelu w widoku Wireframe pokazująca ułożenie krawędzi pojazdu na zdjęciu referencyjnym.	24
4.10. Ilustracja przedstawiająca koło pojazdu.	25
4.11. Ilustracja przedstawiająca lusterko boczne umieszczone na drzwiach pojazdu.	26
4.12. Okno modyfikacji materiału nałożonego na siatkę pojazdu.	27
4.13. Render gotowego pojazdu stworzonego na potrzeby projektu.	27
4.14. Przeszkody stworzone w Blenderze.	28
4.15. Ustawienie przeszkód na poziomie 1.	30
4.16. Ustawienie przeszkód na poziomie 3.	31
4.17. Ustawienie przeszkód na poziomie 5.	31
4.18. Materiał SlipperMat, który został dodany na powierzchnię planszy.	32
4.19. Zrzut ekranu pokazujący oś czasu oraz wynik animacji.	33
4.20. Skrypt PlayerMovement obsługujący poruszanie się pojazdu po poziomie. .	34
4.21. Zdjęcie ukazujące pojazd, który zboczył z trasy, efekt skryptu ruchu.	36
4.22. Skrypt FollowPlayer obsługujący umieszczenie kamery za pojazdem.	36
4.23. Skrypt PlayerCollision obsługujący kolizje pomiędzy pojazdem a przeskodami.	37
4.24. Przykładowa kolizja, która wyrzuciła pojazd do góry.	38
4.25. Skrypt GameManager zajmujący się restowaniem poziomów.	39
4.26. Skrypt EndTrigger nawiązujący do GameManager.	40
4.27. Skrypt LevelComplete ładujący kolejny poziom.	41
4.28. Okno Build pozwalające określić kolejność poziomów.	41
4.29. Okno wyboru ścieżki dostępu do pliku exe oraz folderu z danymi gry. . . .	42

Załączniki

Zał. nr 1 Oświadczenie dotyczące praw autorskich i danych osobowych przechowywanych w Systemie Antyplagiatowym

Zał. nr 2 Płyta CD

Do pracy dołączono płytę CD z następującą zawartością:

Katalog	Zawartość
Installer	Plik instalacyjny z grą
Praca	Praca inżynierska w formacie PDF
Projekt Unity	Folder zawierający projekt wykonany w Unity
Pliki Blender	Folder zawierający wszystkie pliki .blend

**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 tekstuów, które powstaną w przyszłości.

.....
(miejscowość, data)

.....
(czytelny podpis autora pracy)

*niepotrzebne skreślić