Oświadczenie kierującego pracą

Oświadczam, że praca dyplomowa inżynierska studenta Kirila Milosha, pt „**Projekt i implementacja interaktywnej animacji sterowania samochodem w Unreal Engine 4 z zachowaniem praw fizyki w wybranych komponentach pojazdu**” została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego inżyniera.

Data .....

.....

podpis kierującego pracą

Oświadczenie autora pracy

Świadomy odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa inżynierska pt. „**Projekt i implementacja interaktywnej animacji sterowania samochodem w Unreal Engine 4 z zachowaniem praw fizyki w wybranych komponentach pojazdu**” została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami ustawa z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2006 r. nr 90, poz. 631 z późniejszymi zmianami).

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w szkole wyższej.

Oświadczam ponadto, że niniejsza wersja jest identyczna z wersją elektroniczną umieszczoną w systemie APD.

Data .....

.....

podpis autora

UNIWERSYTET

PRZYRODNICZO – HUMANISTYCZNY

W SIEDLCACH

WYDZIAŁ NAUK ŚCISŁYCH I PRZYRODNICZYCH

Kierunek: Informatyka

Kiril Milosh

PROJEKT I IMPLEMENTACJA INTERAKTYWNEJ ANIMACJI STEROWANIA SAMOCHODEM W UNREAL ENGINE 4 Z ZACHOWANIEM PRAW FIZYKI W WYBRANYCH KOMPONENTACH POJAZDU

Praca inżynierska
napisana w Instytucie Informatyki
pod kierunkiem
dr. Andrzeja Salamończyka

Siedlce, 2023

W pracy zaprojektowano oraz zaimplementowaną interaktywną animację sterowania samochodem z wykorzystaniem silnika Unreal Engine. Zaimplementowano mechanizm symulacji zachowania samochodów z prostą konfiguracją w zależności od specyfikacji poszczególnych modeli pojazdów. Sterowanie pojazdem opiera się na współdziałaniu jego elementów ze sobą, takich jak silnik, skrzynia biegów i przekładnia, a opracowany algorytm działa cyklicznie, aktualizując obliczenia wszystkich parametrów pojazdu co klatkę. Praca zawiera opis fazy projektowania, implementacji oraz testowania aplikacji. Zawiera również przegląd podobnych rozwiązań i technologii do ich wykonania.

Słowa kluczowe: Unreal Engine, technologia Blueprints, 3D model samochodu, symulacja zachowania samochodów.

DESIGN AND IMPLEMENTATION OF AN INTERACTIVE CAR CONTROL ANIMATION IN UNREAL ENGINE 4 WITH THE LAWS OF PHYSICS IN SELECTED VEHICLE COMPONENTS

In this work designing and implementing an interactive car control animation using the Unreal Engine process is described. A mechanism for simulating the behavior of cars with a simple configuration depending on the specifications of individual vehicle models has been implemented. The vehicle control is based on the interaction of its elements with each other, such as the engine, gearbox and transmission, and the developed algorithm works cyclically, updating the calculations of all vehicle parameters every frame. The thesis contains a description of the phase of designing, implementing and testing the application. It also contains an overview of similar solutions and technologies for their implementation.

Keywords: Unreal Engine, Blueprints technology, 3D car model, car behavior simulation.

Spis treści

1. Wstęp

Tworzenie nowoczesnych fizycznie realistycznych gier komputerowych jest dość złożonym procesem. Twórcy gier muszą stale kontrolować równowagę pomiędzy fizycznymi i mechanicznymi właściwościami postaci w grze ponieważ mechanika i fizyka w takich grach musi oddawać realizm wszystkiego, co się dzieje, aby sama rozgrywka była intuicyjna, zrozumiała, a jednocześnie ciekawa dla graczy.

Nie ma wątpliwości, że każda gra posiada silnik fizyczny, w przeciwnym razie obiekty oddziaływałyby na siebie losowo lub nie reagowałyby w ogóle. Takie systemy, jak Havok, BPL, ODE i PhysX sprawiają, że środowisko i obiekty oddziałują na siebie w sposób zbliżony do praw fizycznych panujących w prawdziwym świecie.

Jednak realistyczna mechanika i fizyka dla twórców gier to ogromne przedsięwzięcie, których implementacja w dużych projektach gier może trwać latami. To bardzo ważny element współczesnych gier, który potrzebuje sporo czasu.

Poza tym fizyka to mało popularna rzecz, która jest uważana za oczywistość. Przeciętny gracz w pierwszej kolejności będzie podziwiał piękny obraz, a potem być może zainteresuje się interakcją obiektów. Wydawanie zasobów na rzeczy, które mogą minąć uwagę gracza nie jest rozsądne, dlatego twórcy wolą zwiększać rozdzielczość tekstur, liczbę wielokątów i efektów wizualnych, a realizm zostawić na później, w wyniku czego, upraszcza się wszelkie interakcje z obiektami gry.

W dodatku jakość fizyki w grach komputerowych jest dziś postrzegana jako norma, [6]. A jeśli coś odbiega od realizmu, to od razu jest to zauważane. Dlatego fizyka dla twórców gier komputerowych to zawsze wyzwanie, trzeba osiągnąć maksymalny realizm gry, jednocześnie nie tracąc zainteresowania grą przez graczy, a także maksymalnie wszystko uprościć i zoptymalizować, aby końcowy produkt był wydajnym.

W fizyce w grach komputerowych trudno jest zrównoważyć i znaleźć "złoty środek", ponieważ wymaga to od deweloperów poważnego wysiłku, wiedzy i przede wszystkim czasu. Tylko dobrej jakości silnik gry i dobre umiejętności programistyczne mogą osiągnąć ten niezbędny balans.

1.1 Cel i zakres pracy

Celem pracy jest zaprojektowanie i implementacja fizycznego modelu sterowania samochodami oraz jego wizualizacja w postaci interaktywnej wizualizacji. Otrzymany algorytm powinien być wydajny i równocześnie symulować działanie wszystkich podstawowych komponentów samochodów takich jak: zawieszanie, tarcie opon, silnik, skrzynia biegów, przeniesienie napędu i hamulce. Użytkownik będzie mógł zaimportować dany model do swojego projektu i skonfigurować swój samochód o pożądanych charakterystykach. Do konfiguracji będą wykorzystywane takie parametry, jak moc i wykres momentu obrotowego silnika, prześwit i środek ciężkości nadwozia, typ napędu, siła momentu obrotowego hamulców. Będzie można w modelu dostosować wartości przełożenia dla każdego biegu i dostosować współczynnik tarcia dla różnych typów nawierzchni. Oprócz tego istnieje możliwość śledzenia aktualnych wartości komponentów za pomocą graficznego interfejsu i dodatkowych trybów do debugowania. Otrzymany samochód będzie można przetestować na wirtualnym placu testowym i w razie potrzeby skorygować parametry komponentów modelu.

W aplikacji będzie można w zależności od stanu elementów sterujących pojazdem, takich jak pedał przyspieszenia, siła hamowania, skręt kierownicą lub wybrany bieg, wyprowadzać siły wypadkowe na każde koło pojazdu, jak również pewne wartości pośrednie, takie jak prędkość obrotowa kół, prędkość obrotowa silnika itp.

1.2 Struktura pracy

Rozdział pierwszy zawiera motywacje do napisania pracy. Jest w nim sformułowany cel pracy oraz jej struktura.

W drugim rozdziale została umieszczona analiza fizyki w świecie gier oraz zaprezentowane istniejące modele do symulacji zachowania samochodów w grach komputerowych.

Trzeci rozdział zawiera w sobie charakterystykę zastosowanych technologii, przedstawia istniejące na rynku konkurencyjne narzędzia i przeprowadza porównanie pomiędzy przedstawionymi technologiami.

W czwartym rozdziale umieszczona została informacja o projekcie systemu. Zostaną przedstawione wymagania funkcjonalne i niefunkcjonalne, przypadki użycia i schematy struktur z domyślnymi wartościami używane w danym projekcie.

Piąty rozdział prezentuje zaimplementowany system wraz ze strukturą samego projektu, wyjaśnieniem wywołania odpowiednich funkcji i tłumaczeniem działania poszczególnych komponentów w modelu.

W szóstym rozdziale zostaną przeprowadzone testy sprawdzające działanie zrealizowanych komponentów i zostanie sprawdzona zgodność otrzymanego modelu fizycznego z postawionymi wymaganiami.

Ostatni siódmy rozdział zawiera podsumowanie pracy, są w nim sformułowane dalsze kierunki prac.

10:1131467460

2. Analiza problemu i przegląd podobnych rozwiązań

Tworzenie gier i rozwijanie do nich modułów to złożony i czasochłonny proces. Istnieje wiele rozwiązań, które mają ułatwić ten proces. Niektóre silniki specjalizują się tylko w grafice i zarządzaniu (na przykład XNA), inne w obliczeniach fizycznych (Nvidia PhysX), jeszcze inne w czymś innym. Fizyka w grach jest często traktowana jako coś oczywistego, jeśli postać skacze, musi wylądować. Oczekujemy, że obiekty w grze będą zachowywać się jak w prawdziwym życiu. Jednak napisanie fizyki gier za pomocą osobnego silnika fizycznego (takiego jak Havok lub PhysX) może wymagać napisania do kilku milionów linii kodu. Tak na przykład, PhysX pozwala na wykonywanie podstawowych obliczeń fizycznych, dlatego do tworzenia fizyczno-realistycznych gier i ich poszczególnych modułów wymagane jest głębokie zrozumienie procesów mechaniki, aerodynamiki i hydrodynamiki. Dlatego fizyka gier wideo jest złożonym obszarem, w którym programiści nieustannie szukają równowagi między realizmem a ograniczeniami mocy obliczeniowej. Jednak różnego rodzaju uproszczenia i silniki fizyczne, takie jak Unity i Unreal Engine, ułatwiają tworzenie i testowanie fizyki gier.

2.1 Fizyka samochodów w świecie gier

W tej części zostaną przedstawione niektóre strategie tworzenia mechaniki pojazdów, a także sposób, w jaki te podejścia do implementacji uzupełniają lub kolidują z główną ideą projektu.

Istnieją dwa główne podejścia do implementacji zachowania samochodów:

1. Implementacja polegająca na odtworzeniu rzeczywistej fizyki samochodu.
2. Realizacja w stylu „Arcade” (ze znacznymi uproszczeniami).

2.1.1 Realizacja oparta na odtworzeniu rzeczywistej fizyki samochodu

Podejście to polega na symulowaniu kół jako niezależnych obiektów fizycznych z własną masą, tarciem i zawieszeniem. Samochód rozpędza się, przykładając moment obrotowy bezpośrednio do kół i dopóki koła dotykają podłoża, możemy użyć silnika fizycznego, aby przekształcić te siły w ruch modelu samochodu. Często silniki gier oferują gotowe modele implementacji prostych ruchów w świecie gry. Na przykład Unity oferuje wbudowany komponent o nazwie WheelCollider, który oferuje już model tarcia do symulacji poślizgu opony, a także dość ciekawy system zawieszenia do samochodów. Należy jednak wziąć pod uwagę, że takie komponenty w większości przypadków są bardzo uproszczone i mają jedynie charakter poglądowy i edukacyjny.

Prawie każda gra wyścigowa poprawia prowadzenie samochodu poprzez ograniczenie maksymalnego kąta skrętu przednich kół w zależności od tego, jak szybko samochód się porusza. Na przykład przy niskich prędkościach możemy pozwolić graczom na skręcanie kół do maksymalnie

czterdziestu pięciu stopni, ale przy dużych prędkościach możemy ograniczyć je do maksymalnie dwudziestu stopni. Duże kąty skrętu przy dużych prędkościach sprawią, że koła będą zachowywać się bardziej jak hamulce, przez co opona natychmiast wpadnie w poślizg i nie będzie jasno odzwierciedlać intencji gracza. Podobnie moment przyspieszenia zmienia się dynamicznie w zależności od aktualnej prędkości, więc przyspieszamy się znacznie szybciej, gdy nasza prędkość jest bliższa prędkości minimalnej, niż gdy zbliżamy się do prędkości maksymalnej.

Niestety podejście oparte na fizyce dodaje kilka poziomów pośrednich między regulowanymi zmiennymi a rzeczywistym wynikiem, jaki one mają na ekranie. Na przykład, jeśli układ kierowniczy po prostu nie działa prawidłowo, może to być wynikiem dowolnej kombinacji zmiennych, takich jak masa, moment obrotowy silnika, prędkość maksymalna, wartości zawieszenia, wartości poślizgu opon, rozstaw osi, wymiary pojazdu itp. W rezultacie rozwiązywanie problemów może być bardzo czasochłonne i żmudne. Zmienne masy, ciąg i przyczepność mogą czasami osiągać niebotyczne rozmiary, dlatego warto testować wartości w różnych skalach i wykorzystywać rzeczywiste wartości, kiedy tylko można je uzyskać.

Z punktu widzenia projektowania gier podejście oparte na fizyce generalnie prowadzi do większej złożoności, zarówno pod względem implementacji, jak i umiejętności graczy. Wiele gier wyścigowych (na przykład Mario Kart) po prostu nie prosi graczy o myślenie o przyczepności opon, równowadze, pędzie itp., ale wiele z tych elementów są wymagane w implementacji opartej na fizyce. Jednak nawet jeśli nie tworzymy symulatora wyścigów, nadal możemy dodać te elementy do naszej gry. Mogą urozmaicić liczbę wyzwań, które angażują graczy, polegając na mechanice jazdy.

Jednak implementacja oparta na zaawansowanych obliczeniach fizycznych zwykle zwiększa złożoność fazy projektowania i wdrażania projektu. Obliczenie sił fizycznych, uwzględniające dużą liczbę zmiennych, znacznie obniża wydajność produktu końcowego w porównaniu z projektem wykorzystującym uproszczone obliczenie niezbędnych zmiennych. Należy również wziąć pod uwagę fakt, że rozgrywka oparta na czysto symulacyjnych obliczeniach wymaga od graczy zrozumienia podstawowych metod prowadzenia samochodu, co zwykle prowadzi do nieoczekiwanego, ale niezawodnego zachowania się samochodu podczas manewrów. W rezultacie zachowanie samochodu okazuje się złożone i trudne do przewidzenia dla graczy, którzy początkowo oczekiwali prostej i intuicyjnej mechaniki.

Jednym ze sposobów rozwiązania tego problemu jest dodanie funkcji podobnej do autopilota, takiej jak dodatkowi „asystenci", których można znaleźć w większości symulatorów wyścigów. Takie „pomocnicy" interpretują intencje gracza na podstawie ich danych wejściowych, a następnie modelują prawidłowy sposób prowadzenia samochodu w danym kontekście. Jeśli jednak

pozostajemy wierni realizmowi, istnieje również możliwość „oszukania” gracza poprzez przyłożenie dodatkowych sił do pojazdu i dynamiczną zmianę właściwości pojazdu, aż pewne manewry staną się łatwe do wykonania.

2.1.2 Realizacja w stylu „Arcade”

Dzięki realistycznej implementacji opartej na fizyce zazwyczaj zaczynamy od zaprojektowania i symulacji wymaganej mechaniki, a następnie przechodzimy do optymalizacji lub wyeliminowania obliczeń matematycznych wymagających dużej ilości zasobów. Jednak w podejściu w stylu arkadowym zaczynamy zasadniczo od zera, a następnie budujemy niestandardową mechanikę, aż do uzyskania pożądanego rezultatu.

W arkadowym podejściu wystarczy zacząć od pojedynczego BoxCollidera, który rozpędza się, przykładając siły, gdy dotyka ziemi, a potem możemy zdecydować, czy potrzebujemy mechaniki poślizgu samochodu, systemu przyczepności opon, czy obu. Możemy wtedy skupić się na uzyskaniu fizyki skoku, a następnie zbudowaniu systemu stabilizacji samochodu podczas lądowania.

Jednak stosując takie podejście, może się okazać, że będziemy musieli ukryć wiele z tych mechanizmów za pomocą animacji lub efektów wizualnych, które uczynią grę bardziej atrakcyjną i zrozumiałą dla gracza. Na przykład, jeśli gra nie potrzebuje mechaniki zawieszenia, to możemy przynajmniej sfałszować zawieszenie, korzystając z wizualnych efektów samochodu na podstawie aktualnego pędu i przyspieszenia.

2.1.3 Wniosek

Pomimo tego, że podejścia te opierają się na zupełnie przeciwnych zasadach, granica między nimi jest bardzo rozmyta. Nietrudno wyobrazić sobie model hybrydowy, który łączy w sobie cechy tych dwóch modeli. Istnieje sporo gier i projektów, które zapewniają dobrą równowagę między fizyczną interakcją a uproszczoną jazdą, na przykład seria Grid firmy Codemasters lub Forza firmy Microsoft. Podczas rozwijania naszej gry ważne jest, aby stale sprawdzać poszczególne mechaniki pojawiające się w naszej symulacji oraz wydajność całego algorytmu, aby dalej decydować, czy je zachować, czy uprościć.

Na przykład fizyka w Grid często pozwala samochodom ledwo wytracać prędkość podczas wchodzenia w poślizg. Zamiast przerabiać to zjawisko, firma Codemasters zdecydowała się wykorzystać je jako element rozgrywki swojego produktu, co z kolei przypadło do gustu przeważającej liczbie graczy.

Bez względu na to, jakie podejście przyjmiemy, osiągnięcie dopracowanej implementacji fizyki samochodu, która zapewnia stałą przyjemność z jazdy, będzie wymagało znacznej ilości pracy i wielu iteracji.

Planując i realizując architekturę tego projektu zdecydowano się na wykorzystanie hybrydowego modelu fizycznego samochodów (fizyka Simcade), gdyż realizacja pełnoprawnej symulacji wymaga skoordynowanej pracy całego zespołu składającego się z projektantów, twórców modeli 3D, animatorów, inżynierów dźwięku, programistów i testerów. Ponadto opracowanie zaawansowanej gry opartej na fizyce wymaga znacznie więcej czasu, zarówno na opracowanie samego modelu fizycznego, jak i na debugowanie i testowanie wszystkich jego poszczególnych elementów.

Jednym z głównych celów przy tworzeniu tego projektu było pokazanie w praktyce złożoności tworzenia gier i modułów dla ich rozbudowania. Głównym problemem fizyki samochodowej jest połączenie różnych sił, które działają na samochód w określonym momencie. Obejmuje to tarcie opon, rozkład masy, opór aerodynamiczny i wiele innych czynników. Nowoczesne potężne komputery nauczyły się obliczać całą tę złożoną fizykę, którą producenci samochodów chętnie wykorzystują w swoich projektach, ale takie systemy fizyczne są zbyt uciążliwe dla gier. Jak dostosować je do różnych konfiguracji komputerów i konsol? Jak zrealizować wyścigi dla dwudziestu uczestników, którzy również regularnie się ze sobą zderzają?

Według portalu [13], programiści upraszczają fizykę pojazdów i oddzielają fizyczny model pojazdu od modelu wizualnego, co pomaga usunąć wiele czynników z systemu. Przykładowo, w wielu grach wyścigowych spojlery na bagażniku to czysto wizualny tuning, który nie wpływa na osiągi samochodu, choć w rzeczywistości takie akcesoria poważnie zmieniają aerodynamikę auta.

Pozostałe siły fizyczne są często dla wygody dzielone przez programistów na siły podłużne i poprzeczne. Siły wzdłużne to wszystko, co popycha samochód do przodu lub do tyłu: przyspieszenie, hamowanie, opór powietrza. Boczne to skręty kierownicy i kół, ślizgające się koła, a w najbardziej zaawansowanych symulatorach samochodowych ostre podmuchy wiatru. Następnie do fizyki samochodu dodawane są warunki zewnętrzne i wewnętrzne, takie jak rodzaj nawierzchni czy ustawienia zawieszenia. Na koniec twórcy łączą wszystkie te czynniki w jeden system. Przyjrzyjmy się bliżej jego elementom.

Silnik i przyspieszenie

Gdy gracz wciska wirtualny pedał gazu, silnik i skrzynia biegów są obsługiwane według mniej lub bardziej skomplikowanych równań, czyli samochód nabiera prędkości zgodnie z prawami fizyki.

Ale sama kombinacja silnika, skrzyni biegów i skrzyni transferowej nie jest symulowana, a twórcy opisują ją zazwyczaj jednym umownym parametrem - przyspieszeniem.

Przyspieszenie pojazdu w grze nie jest stałe i zazwyczaj zależy od momentu obrotowego kół. To z kolei zależy od mocy silnika i włączonego biegu. A nawet jeśli gracz nie zmienia biegów ręcznie, silnik gry zwykle robi to za kulisami. Twórcy muszą jedynie określić moc silnika i dostosować biegi.

Poważniejsze symulatory samochodowe używają skomplikowanych równań dla biegów i obliczają je inaczej dla różnych samochodów. Na przykład w grze Project Cars niektóre klasyczne amerykańskie samochody bardzo wolno wychodzą z pierwszego biegu, podczas gdy samochody sportowe przeskakują go z łatwością.

Jednak programiści nie ustalają maksymalnej prędkości samochodu - silnik sam ją oblicza na podstawie innych danych. Tak długo, jak siła pociągowa silnika przewyższa siły przeciwne, takie jak opór powietrza, samochód porusza się. Wraz ze wzrostem prędkości rosną również siły przeciwne i w pewnym momencie wyrównują się z przyspieszeniem samochodu. Jest to punkt, w którym silnik ustanawia swoją maksymalną prędkość.

Takie uproszczenia stają się wyraźnie zauważalne przy najwyższych prędkościach i nachyleniach. Na przykład opór powietrza jest mniejszy w górach niż na nizinach, ale nawet skrupulatne symulatory nie próbują symulować tego efektu. A realistyczne przyspieszenie zwykle zaczyna nie zgadzać się z realnymi wartościami, gdy igła prędkościomierza przechodzi ponad 100 km/h - około tej wartości zarządzanie trakcją i przyczepnością jest trudniejsze, ale wiele gier wyścigowych, wręcz przeciwnie, starają się jak najbardziej uprościć sterowanie na dużych prędkościach.

Prowadzenie poślizgu i kierowanie

W nowoczesnych grach wyścigowych pokonywanie zakrętów w kontrolowanym poślizgu stało się jedną z głównych mechanik rozgrywki. Jednak poślizg w grze ma niemal więcej konwencji i uproszczeń niż reszta fizyki gry. Na przykład w wielu częściach Need for Speed gracz może bez problemu driftować[1] na samochodach z napędem na przednie koła, choć tradycyjnie to jest możliwe tylko dla samochodów z napędem na tylne koła.

Jak działa driftowanie w samochodach z napędem na tylną oś: w zakręcie kierowca zwalnia przepustnicę i naciska hamulec, cięższa przednia część samochodu z silnikiem jest dociskana do

[1] Technika jazdy autem w kontrolowanym poślizgu.

ziemi, a tylna oś przeciwnie - unoszona. W tym momencie koła napędowe tracą przyczepność i możliwy staje się driftowanie.

Ważna dla kontrolowanego poślizgu jest również przyczepność - tarcie, które zapobiega niekontrolowanemu poślizgowi opony. Projektanci gier zazwyczaj dostosują to tak, aby samochód łatwiej wchodził w poślizg lub pozostawią te ustawienia graczowi. The Crew 2 pozwala na przykład na oddzielne ustawienia przyczepności przedniej i tylnej opony - jeśli gracz chce przejść każdy zakręt w poślizgu, może celowo poluzować przyczepność tylnej osi.

W fizycznych parametrach driftowania deweloperzy mogą wykazać się wyobraźnią, bo często nie mają na czym się oprzeć. Inżynierowie prawdziwych samochodów starają się za wszelką cenę uniknąć poślizgu. Na przykład równomiernie rozkładają ciężar pomiędzy przednią a tylną osią, dzięki czemu koła napędowe nie tracą przyczepności.

Na prawdziwym torze wyścigowym nawet kontrolowany poślizg może być poważnym problemem. Podczas driftu prędkość spycha samochód na bok zamiast do przodu, co może kosztować kierowcę cenne sekundy lub nawet pierwsze miejsce. W grach driftowanie nagradza zręcznych graczy, ale rzadko karze za niepowodzenia: zwykle utrata prędkości spowodowana niedokładnym dryftem jest minimalna i łatwo ją zrekompensować.

Drugim ważnym aspektem poślizgu jest sterowanie, ponieważ kontrolowany dryft również wymaga odpowiedniego kąta jazdy. Często gry wyścigowe znacznie ułatwiają sterowanie, aby nie psuć radości z gry graczom z gamepadem i klawiaturą. Na przykład seria Forza Motorsport łagodnie podchodzi do ostrych skrętów kierownicy, choć w rzeczywistości kierowca nie jest w stanie błyskawicznie przekręcić koła ze skrajnej lewej strony na skrajną prawą.

Symulatory inspirowane iRacing, przeznaczone dla dedykowanych kontrolerów, są pod tym względem bardziej realistyczne. Zwykle ograniczają się one do podstawowych "elektronicznych asystentów" - pozwalają np. ograniczyć sterowanie, by gracz nie driftował niechcący tam, gdzie nie jest to konieczne.

Opony i kontakt z drogą

Złożoność fizyki opon polega na tym, że silnik musi ją obsługiwać dla wszystkich czterech kół osobno. W końcu samochód stojący w całości na asfalcie będzie zachowywał się zupełnie inaczej niż taki, który ma dwa koła na zarośniętej ziemi. A to tylko interakcja z drogą: trzeba też wziąć pod uwagę obciążenie kół, ciśnienie w oponach, deformację opon i inne czynniki.

Dlatego programiści starają się przynajmniej częściowo wykorzystać gotowe parametry opon. Autorzy Forza Motorsport 4 współpracowali więc z producentami opon - przedstawiciele Pirelli, Toyo, Goodyear dostarczyli deweloperom niezbędne dane, które bezpośrednio trafiły do gry. Później twórcy porównali oficjalne dane z wynikami swoich symulacji fizycznych i okazało się, że parametry są niemal takie same.

Temperatura opon to bardziej skomplikowany parametr, który jest ściśle monitorowany przez profesjonalnych rajdowców - jednak większość graczy nie przywiązuje do niego wagi. W grach zręcznościowych praktycznie jej nie widać, natomiast w bardziej realistycznych wyścigach temperatura opon jest zazwyczaj prezentowana jako opcja - jeśli gracz chce się martwić o rozgrzanie opon, może aktywować specjalne pole wyboru w ustawieniach.

I tak w najnowszych częściach F1 programiści z Codemasters wraz z inżynierami Formuły 1 dostosowali opony w grze tak, aby ich temperatura nie tylko wpływała na rozgrywkę, ale także była obliczana na kilku poziomach. Silnik gry obsługuje wewnętrzne i zewnętrzne ciepło opony oddzielnie: rozgrzanie zewnętrznej warstwy opony i przygotowanie jej do wyścigu jest łatwe, ale rozgrzanie warstwy wewnętrznej wymaga bardziej intensywnej jazdy - a to może spowodować rozerwanie opony.

Zachowanie opony w dużym stopniu zależy od nawierzchni drogi i nie chodzi tu tylko o wszelkiego rodzaju drogi gruntowe i mokre - nawet zwykły suchy asfalt będzie miał inny wpływ na jazdę samochodu, w zależności od tego, czy jest zimny, czy gorący w słońcu. Na drodze możemy spotkać oba rodzaje asfaltu - na przykład oświetlony i nagrzany odcinek wokół zakrętu zostanie zastąpiony przez zacieniony odcinek w lesie. W takich sytuacjach deweloperzy korzystają z porad profesjonalnych kierowców.

Zawieszenie

W dzisiejszych samochodach dobre zawieszenie jest częściej potrzebne dla komfortu pasażerów niż dla osiągów. Wyjątkiem są samochody rajdowe, które są zobowiązane do prawidłowego radzenia sobie ze wszystkimi wybojami, nierównościami i skokami. W grach fizyczna obsługa zawieszenia jest konieczna, aby gracz mógł lepiej odczytać powierzchnię pod samochodem, a samochód - zareagować na wszelkie nierówności, dlatego twórcy często biorą za podstawę samochody rajdowe. Choć nie jest to też pozbawione uproszczeń.

Pełna symulacja zawieszenia ze wszystkimi elementami jest rzadkością, nawet w zaawansowanych symulatorach samochodowych, ale są wyjątki, takie jak BeamNG.drive. Twórcy

tego symulatora samochodowego zaprojektowali zawieszenie w grze nie jako kompletny system, ale jako zbiór poszczególnych fizycznych komponentów.

Rozwiązanie BeamNG.drive nie pasuje jednak do wszystkich gier. Z jednej strony jest to dość proste, bo każda część zawieszenia w grze jest podstawowym obiektem fizycznym, który funkcjonuje samodzielnie, bez dodatkowych linijek kodu do tuningu. Z drugiej strony pojawiają się problemy ze skomplikowanymi elementami, takimi jak hydraulika, które muszą się rozszerzać i kurczyć. W dodatek takie elementy muszą być ponownie dostosowany do każdego pojedynczego pojazdu.

Zawieszenie w grze jest również wrażliwe na częstotliwość odświeżania silnika, która wpływa na to, jak szybko wirtualny samochód uświadamia sobie, co znajduje się pod jego kołami. Na przykład w serii F1 wysoka częstotliwość odświeżania jest potrzebna, aby samochody F1 szybko reagowały na krawężniki wzdłuż toru.

Arkady, symulatory i asystenci kierowcy

Wszelkiego rodzaju rozwiązania jak elektroniczny asystent znajdują się we wszystkich nowoczesnych samochodach, skąd zapożyczyły je gry wideo. W rzeczywistości asystenci służą bezpieczeństwu, aby roztargniony lub śpiący kierowca nie zranił siebie, pasażerów i innych użytkowników dróg. W grach okazały się świetnym sposobem na zrównoważenie złożoności, ponieważ mogą bezpośrednio wpływać na prowadzenie samochodu.

Na przykład, twórcy Project CARS 3 starali się, aby gra była zarówno przystępna dla nowicjuszy, jak i ciekawa dla rajdowców. Autorzy nie chcieli upraszczać modelu fizycznego, więc popracowali nad elektronicznymi pomocami: jeśli podkręcimy je do maksimum, Project CARS 3 zamienia się w lekką grę zręcznościową, jeśli wyłączymy je całkowicie – w realistyczny symulator jazdy.

2.2 Przegląd podobnych rozwiązań

Opisane poniżej projekty specjalizują się w symulacji zjawisk fizycznych działających na pojazd podczas jego ruchu. Nie należy jednak traktować tych projektów jako pełnoprawnych gier, gdyż większość wysiłków twórców skupia się na obliczaniu sił fizycznych, a nie na elemencie rozgrywki.

2.2.1 Vehicle Physics Pro

Vehicle Physics Pro (w skrócie VPP) to zaawansowany zestaw do symulacji pojazdów dla Unity 3D, który zapewnia wydajne, w pełni realistyczne i dokładne symulacje pojazdów dla niemal wszystkich typów i konfiguracji pojazdów.

Realistyczny model opony

Część teoretyczna oparta jest głównie na formułach Hansa B. Pacejka (ekspert w dziedzinie fizyki pojazdów i układów kołowych). Stosuje się również równania relaksacji podłużnej i poprzecznej oraz (opcjonalnie) równania odkształceń sprężystych. Przy tym wyszukują się wszystkie punkty styku opony z powierzchnią drogi, a nie tylko jeden (jak w przypadku Wheel Collider), ale można to wyłączyć, gdy używamy trybu arcade (tj. z uproszczoną fizyką i mniejszym wykorzystaniem zasobów).

Realistyczny model zawieszenia, układ napędowy

W VPP symulowane jest zawieszenie, zachowanie amortyzatorów (zarówno na ściskanie, jak i wyciąganie). Zawiera również realistyczne modele sprzęgła, mechanizmu różnicowego, kół zębatych i zębników.

Elementy zewnętrzne

Wbudowana jest funkcjonalność świateł hamowania, deski rozdzielczej. Tablica przyrządów może być dowolnie skonfigurowana, można wyświetlić prędkościomierz, obrotomierz, tachometr, ABS, skrzynię biegów itp.

Dźwięk

Dźwięk silnika odtwarza wszystkie rodzaje odgłosów takich jak silnik, zawieszenie, skrzynia biegów, opony, uszkodzenia, poślizgi, tarcie, jazda po piasku, trawie i w terenie. Hałas różni się również w zależności od modelu pojazdu.

Ułatwienie prowadzenia pojazdu

Nie blokujące się hamulce, system stabilizacji, kontrola trakcji, tryb treningowy. Tryb Arcade, to znacznie upraszcza model fizyki, ułatwiając sterowanie, ale zmniejszając realizm.

Używanie VPP

Vehicle Physics Pro może być używany wszędzie tam, gdzie działa Unity3D. Sam silnik jest napisany w C# i dostarczany z kodem źródłowym.

Silnik działa również na platformach mobilnych (iOS i Android) i działa z akceptowalną prędkością.

Wiele parametrów można skonfigurować, a jeśli nie masz pożądanego ustawienia, możesz wprowadzić zmiany w kodzie źródłowym. Vehicle Physics Pro posiada własną stronę internetową[2], na której szczegółowo opisane są wszystkie funkcje modelu fizycznego, a także dokumentacja i instrukcje jak z niego korzystać.

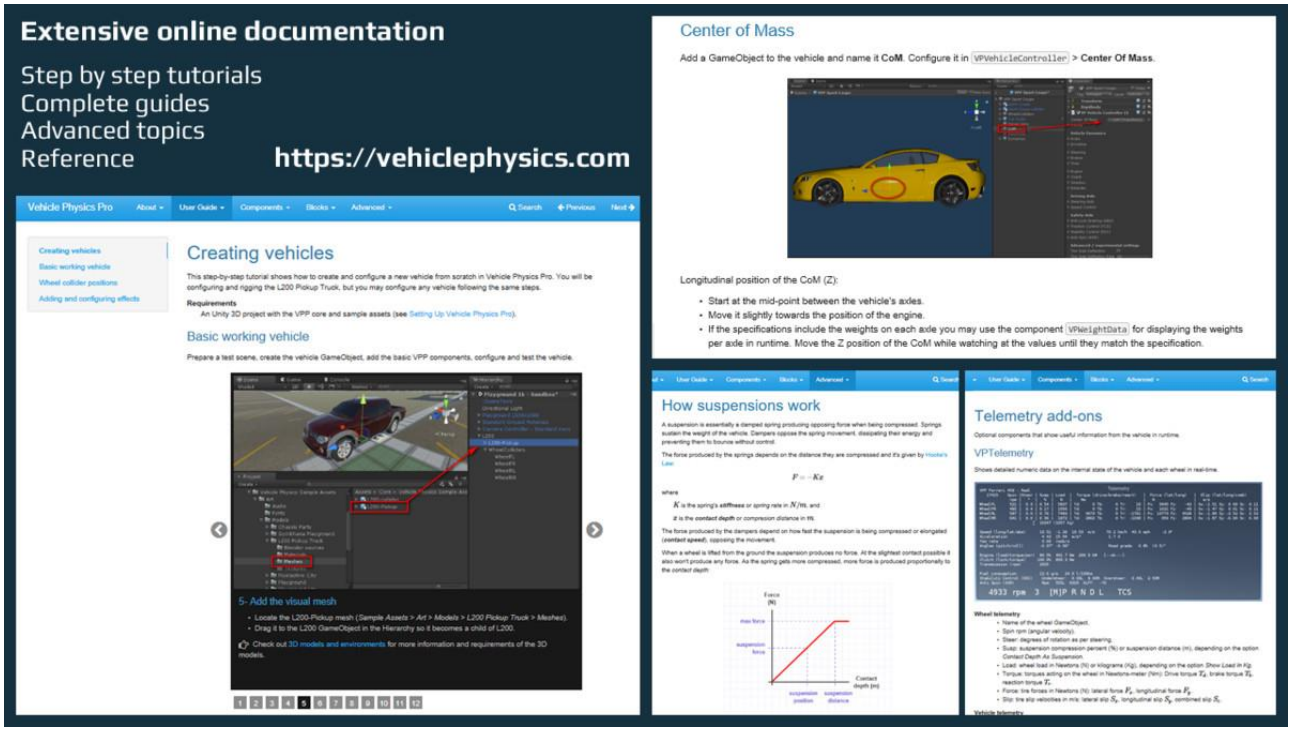
Features	Community	Professional	Enterprise
Full-featured Vehicle Controller and Wheel Collider components	✔	✔	✔
Essential components: Telemetry, Audio, Anti-roll bar, keyboard input, visual effects, rolling friction, aerodynamics	✔	✔	✔
Example locations and vehicles	✔	✔	✔
Wheels per vehicle	4	1-20	1-unlimited
Simultaneous vehicles in each scene	1	unlimited	unlimited
Supported platforms	Desktop	All	All
Multiple ground materials with tire effects (tire marks, smoke...)		✔	✔
Custom extensions support: custom add-ons, custom vehicles, custom parts		✔	✔
Frame-exact replay system with pause, forward/reverse playback, flashback...		✔	✔
Vehicle damage with mesh deformation and degradation of handling		✔	✔
Advanced telemetry tools: real-time performance charts, weight distribution		✔	✔
High-quality locations and vehicles *(WIP)*		✔	✔
Mobile touch-based controller *(WIP)*		✔	✔
DirectInput controller with force feedback *(Windows only)*		✔	✔
XBox gamepad support (360 and One) *(Windows only)*		✔	✔
Support by email. Other support options		individual	team
Number of developers		1	up to 10
Full source code included			✔
Advanced simulation features: multi-body vehicles, solid and liquid cargo components			✔
Advanced suspension features: damper bump & rebound, dynamic suspension, bump stop			✔

Rysunek 1. Zdjęcie przedstawiające tabelę porównawczą trzech wersji VPP. Źródło: https://vehiclephysics.com/about/licensing/

[2] https://vehiclephysics.com/

Istnieją trzy wersje produktu: Community, Professional, Enterprise. Pierwsza wersja jest darmowa i ma znacznie ograniczoną funkcjonalność. Na rysunku 1 znajduję się porównawcza trzech wersji VPP:

Również Vehicle Physics Pro posiada własną stronę internetową, na której znajduje się szczegółowa dokumentacja i opis wszystkich składników projektu, zestaw podstawowych samouczków oraz szereg innych materiałów pomocniczych przydatnych podczas korzystania z VPP (patrz rysunek 2).



Rysunek 2. Zdjęcie przedstawiające materiały pomocnicze dla VPP. Źródło: https://assetstore.unity.com/packages/tools/physics/vehicle-physics-pro-community-edition-153556

Na rysunku 3 przedstawione są prawdziwe samochody modelowane przy użyciu Vehicle Physics Pro w różnych projektach stron trzecich. Wszystkie pojazdy wykorzystują cechy prawdziwego modelu. Wiele z nich jest wykorzystywanych w certyfikowanych symulatorach szkolenia kierowców. Zdjęcia pochodzą z realnych scen testowych.

2.2.2 BeamNG.drive

BeamNG.drive to symulator pojazdów i miękkich ciał w czasie rzeczywistym opracowany i opublikowany przez startup BeamNG i działający w systemie Microsoft Windows. BeamNG.drive oferuje oryginalne sterowanie dla tzw. soft-body physics, fizyki, która może symulować reakcje "miękkich" materiałów, takich jak tkanina czy skręcona stal. BeamNG.drive to nie tyle gra, co demonstracja niedoskonałości współczesnych projektów gier z udziałem samochodów wszelkich odmian.

Rysunek 3. Zdjęcie przedstawiają możliwości VPP w realizacji różnego typu pojazdów. Źródło: https://vehiclephysics.com/about/showcase/.

Projekt ten jest rozwijany od końca maja 2012 roku i trwa do dziś, co kilka miesięcy pojawiają się nowe wersje. Wprowadzają do projektu dodatkową zawartość, naprawiają błędy i powoli, ale skutecznie zmieniają prototyp w mniej lub bardziej gotowy produkt (patrz rysunek 4).

Rysunek 4. Zdjęcie przedstawiające działanie zawieszenia samochodu w BeamNG.drive. Źródło: opracowanie własne.

Realizm

Twórcy postawili na realizm zniszczeń. Zderzenia ze skałami i drzewami sprawiają, że samochody zamieniają się w akordeon, z którego wylatują zderzaki, maski i drzwi. Twórcy uzyskali ten efekt dzięki silnikowi Torgue 3D, który został zmodyfikowany przez programistów BeamNG. Rezultatem jest doskonała fizyka miękkiego ciała i całkowicie unikalny silnik (patrz rysunek 5).

Rysunek 5. Zdjęcie przedstawiające uszkodzenia samochodu w BeamNG.drive. Źródło: opracowanie własne.

Gracz jest w stanie przetestować zarówno drogie i nowoczesne super samochody, jak i ciężarówki oraz ciągniki na wielu różnych torach, od pustyni po serpentyny górskie. Jest okazja, by po prostu obserwować upadek modelu z góry i jego stopniowe niszczenie.

Sterowanie

Sterowanie w BeamNG Drive jest również realistyczne, więc samochód będzie często wpadać w poślizg na zakrętach, jeśli droga jest pokryta lodem lub asfalt jest mokry. W grze przewidziane są warunki pogodowe: śnieg, deszcz, a nawet grad.

Gracz może sterować pojazdem, zarówno w trybie trzecioosobowym[3], jak i z kokpitu. Wszystkie modele pojazdów są bardzo dobrze zaprojektowane i szczegółowe. Każdy samochód ma inne i niepowtarzalne odczucia. Samochody są wrażliwe na nierówności drogi - na dużych wybojach drżą, a nawet przewracają się.

Nierówności drogi, nagłe przeszkody mogą całkowicie zniszczyć samochód. Wypadki można oglądać z różnych perspektyw - dostępne są kamery i tryby specjalne.

[3] Określenie perspektywy graficznej w grach komputerowych (zwykle o grafice trójwymiarowej), w których gracz obserwuje świat przedstawiony zza pleców kierowanej postaci.

3. Przegląd technologii do realizacji zadania

Podczas realizacji tego projektu wykorzystano następujące oprogramowanie: do modelowania i dalszego przygotowania modelu samochodu użyto 3Ds Max, a późniejsza implementacja modelu zachowania fizycznego została przeprowadzona w silniku Unreal Engine 4. W tej sekcji dokonamy przeglądu oprogramowania i porównamy je z istniejącymi odpowiednikami.

3.1 3Ds Max

3Ds Max[4] to oprogramowanie do modelowania, animacji i renderingu 3D zaprojektowane i stworzone z myślą o grach i wizualizacji projektów. Program wchodzi w skład kolekcji oprogramowania multimedialnego i rozrywkowego oferowanego przez firmę Autodesk. Jest także częścią kolekcji Autodesk w architekturze, projektowaniu i budownictwie i stanowi jedno z narzędzi w wytwarzaniu produktów 3D.

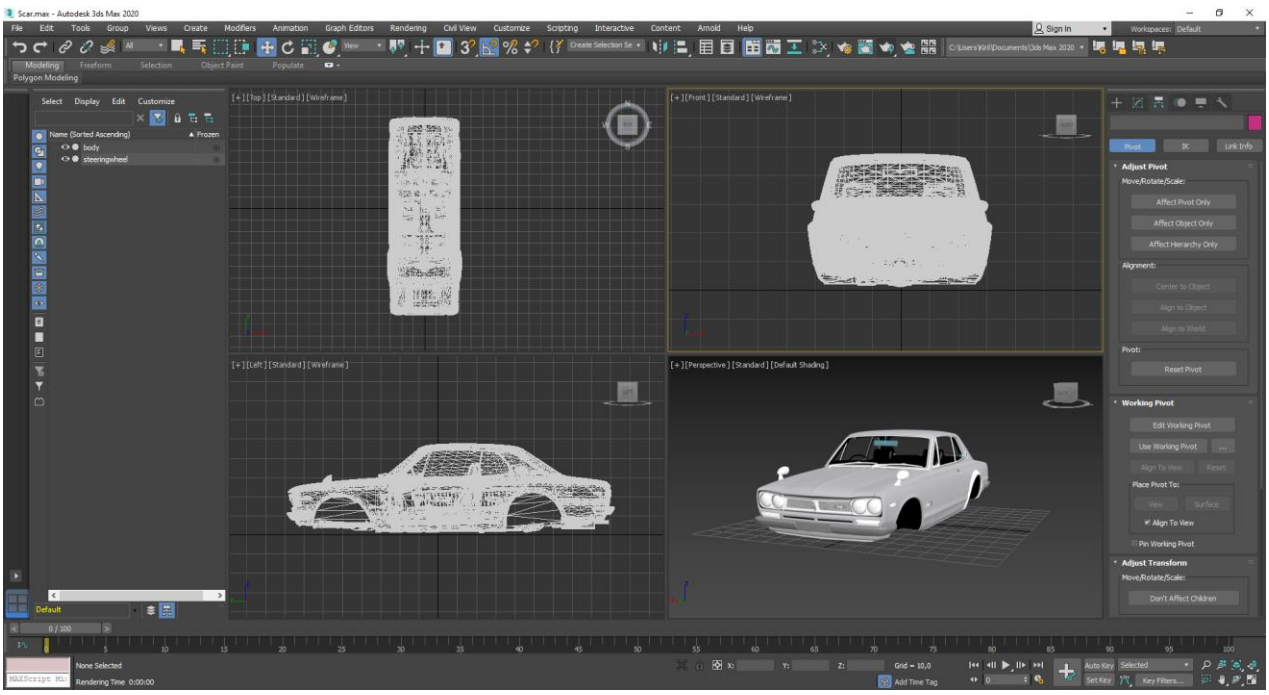
3Ds Max jest używany przez artystów i specjalistów od efektów wizualnych w przemyśle filmowym i telewizyjnym, a także przez twórców i projektantów do tworzenia gier w rzeczywistości wirtualnej. Oprogramowanie jest bardzo przydatne w projektowaniu budynków, infrastruktury i konstrukcji, a także w rozwoju produktu i planowaniu produkcji. Ponadto 3Ds Max pomaga użytkownikom tworzyć masywne światy gier, szczegółowe postacie, dostosowywać środowiska budowlane, tworzyć sceny z dużą ilością ludzi i symulować właściwości fizyczne płynów, takich jak woda, olej i lawa. Ponadto 3Ds Max posiada kontrolery animacji, które użytkownicy mogą tworzyć, modyfikować i udostępniać. Oprogramowanie posiada również funkcje renderowania 3D, takie jak możliwość symulacji rzeczywistych ustawień kamery. Ponadto oferuje bibliotekę zasobów, która umożliwia użytkownikom łatwe wyszukiwanie kontentu 3D. 3Ds Max oferuje również funkcje do modelowania 3D, teksturowania i efektów. Dzięki temu użytkownicy mogą tworzyć i animować geometrię na wiele sposobów, a także stosować modelowanie powierzchniowe i siatkowe.

3DS Max to międzyplatformowa aplikacja do tworzenia modeli, scen, animacji, materiałów i wszystkiego, co związane ze światem 3D. Wszystkie efekty specjalne w filmach, prezentacje nowych modeli maszyn, ubrań, samochodów - wszystko to są wcześniej zaprojektowane modele 3D, które następnie zamienia się w realne obiekty. Profesjonaliści wybierają 3Ds Max z kilku powodów, jednym z nich jest obsługa dużej ilości istniejących skryptów i pluginów. Dla specjalisty najważniejszy jest nieograniczony potencjał i możliwości, a prostota i użyteczność są na drugim miejscu.

[4] https://www.autodesk.pl/products/3ds-max/overview

3Ds Max oferuje użytkownikom wiele sposobów i solidnych narzędzi do tworzenia i edycji animacji (patrz rysunek 6). Potrafią tworzyć trójwymiarowe animacje komputerowe i efekty, które można zastosować w grach komputerowych, filmach, audycjach, ilustracjach lub prezentacjach. Oprogramowanie posiada kilka kontrolerów animacji, które służą do przechowywania kluczowych wartości i ustawień proceduralnych, obsługując wszystko, co użytkownicy animują za jego pomocą. 3Ds Max umożliwia również łączenie obiektów ze sobą. W rezultacie można tworzyć hierarchie lub łańcuchy, za pomocą których zestawy obiektów mogą być animowane jednocześnie, co upraszcza proces.

Według [5], wydawnictwo Ascent - Center of Technical Knowledge, jednym z najczęściej używanych narzędzi w 3Ds Max jest Material Editor, który pozwala użytkownikom na tworzenie i edycję materiałów i map w ich scenach, nakładanie kreatywnych tekstur i symulowanie załamań, odbić i innych efektów podczas przypisywania materiałów do obiektów. 3Ds Max to jeden z najwcześniejszych programów do pracy z grafiką 3D. Można by wysnuć analogię między 3Ds Max a After Effects. Oba są bardzo „ciężkie", trudne do zrozumienia, ale potrafią wytworzyć produkty najwyższej jakości. Większość bibliotek, presetów, pluginów, jest napisanych dla nich, co od razu eliminuje konkurencję - nawet jeśli użytkownik będzie chciał migrować z 3Ds Max na inne oprogramowanie, to pozostanie bez swoich ulubionych narzędzi.



Rysunek 6. Zdjęcie przedstawiające interfejs graficzny programu 3DS Max. Źródło: opracowanie własne.

W momencie premiery nie miała ona swojego odpowiednika. Zanim ukazał się pierwszy analog minęło sporo czasu, więc oprogramowanie otoczyło się warstwą dyskusji, wieloma wątkami

na forach oraz tonami poradników i tutoriali. Aby którykolwiek z konkurentów mógł go przewyższyć lub pokonać, musiałby cofnąć się o 20 lat i zacząć tworzyć własne oprogramowanie.

Jednak obecnie oczywiście istnieje wiele innych programów podobnych do 3Ds Max. Jedne lepiej nadają się do szkicowania, inne do projektowania produktów i schematów, albo na przykład do tworzenia animacji i kompletnych filmów. Przyjrzyjmy się trzem najważniejszym konkurentom:

Blender jest zaskakująco prosty i lekki, a dzięki niewielkim rozmiarom mieści się nawet na pendrivie. Standardowy pakiet nie zawiera wielu bibliotek i presetów. Funkcjonalność programu mimo to jest naprawdę imponująca. Potrafi wszystko to, co jest założone w fundamentach 3Ds Max, w dodatku jest w stanie podłączyć skrypty do automatyzacji typowych działań. Co warto zauważyć Blender jest darmowy w porównaniu z 3Ds Max, którego miesięczna subskrypcja kosztuję ponad 1000 złotych, Jednak potencjał Blendera nie jest aż tak duży. Ci, którzy z nim pracowali, często nazywają go emulatorem 3Ds Maxa, ponieważ wszystkie skrypty i pluginy muszą być ponownie skonfigurowane i przekonwertowane do nowego formatu. Niemniej jednak dziś jest to silny konkurent, który z roku na rok coraz bardziej wysuwa się na czoło rynku.

AutoCAD to kolejne oprogramowanie CAD 3D firmy Autodesk, które jest przeznaczone do projektowania produktów, projektowania budynków, planowania produkcji, infrastruktury cywilnej i budownictwa.

Jest częścią kolekcji oprogramowania CAD 3D firmy Autodesk, z którego korzystają zespoły opracowujące produkty, firmy produkcyjne, branża medialna i rozrywkowa, inżynierowie, architekci, nauczyciele i studenci, przedsiębiorcy, przedstawiciele zawodów medycznych i wielu innych.

Ponadto AutoCAD służy do tworzenia rysunków 2D, dokumentów, modeli 3D i wizualizacji. Jego funkcje rysowania, kreślenia i adnotacji 2D obejmują możliwość kontrolowania wyglądu tekstów, automatycznego tworzenia stylów i wymiarów, łączenia i aktualizowania danych między arkuszami kalkulacyjnymi i tabelami programu Microsoft Excel w rysunkach oraz pracy z blokami dynamicznymi.

MAYA od dawna była stawiana w przeciwieństwie do swojego głównego konkurenta na rynku oprogramowania – 3Ds Max. Posiada wszystko co potrzebne do tworzenia grafiki 3D. Maya jest w stanie przeprowadzić przez wszystkie etapy tworzenia 3D - od modelowania i animacji po teksturowanie, kompozycję i warstwowe renderowanie. Ten trójwymiarowy edytor może symulować fizykę ciał stałych i miękkich, obliczać zachowanie tkaniny, emulować efekty płynów, pozwala na szczegółowe dopasowanie włosów postaci, tworzenie suchego i mokrego futra, animowanych włosów itp. Wizytówką oprogramowania jest moduł PaintEffects, który daje

możliwość rysowania wirtualnym pędzlem takich trójwymiarowych obiektów jak kwiaty, trawa, trójwymiarowe wzory i inne. Program jest dość trudny do opanowania, co jest rekompensowane przez dużą liczbę tutoriali na temat tego edytora. Ponadto, ten edytor jest wykorzystywany przez największe studia, takie jak Pixar, WaltDisney, Dreamworks i inne.

Wniosek

Grafika trójwymiarowa to cała nauka, dziedzina, w której można pogłębiać swoją wiedzę i umiejętności przez całe życie. Dlatego wymienienie różnic w narzędziach oferowanych artystom 3D przez poszczególne edytory 3D jest zadaniem obszernym i zajęłoby więcej niż jeden artykuł. Niemniej jednak każde oprogramowanie charakteryzuje się specyficznym zestawem narzędzi, co determinuje obszar, w którym wygodnie jest zastosować edytor 3D. Do renderingu architektonicznego, oczywiście, najlepiej nadaje się 3Ds Max wraz z jego kompatybilnością z innymi aplikacjami Autodesk, takimi jak AutoCad, obszerną biblioteką materiałów architektonicznych i elastycznym graficznym interfejsem.

Niekwestionowanym liderem w branży filmowej jest Maya. Program ten specjalizuje się w tworzeniu animacji, wizualnych efektów 3D i najlepiej nadaje się dla artystów 3D i animatorów.

3.2 Unreal Engine

W rozwoju każdego produktu, kwestia silnika gry jest zawsze jedną z najważniejszych. Nad projektem gry pracują zwykle różni specjaliści - programiści, artyści, projektanci, dźwiękowcy i inni eksperci, których w branży gier jest dziś wielu. Aby stworzyć wysokiej jakości produkt, potrzebują zestawu narzędzi - silnika gry. Gdy nie było tak szerokiego dostępu do popularnych silników jak Unity, Unreal Engine, CryEngine i innych, ludzie siadali i pisali wszystko sami. Jest to dość skomplikowane, czasochłonne i dość ryzykowne, ponieważ nie wszystkie zespoły mają wiedzę techniczną i umiejętności, aby to zrobić.

Celem firm tworzących gry jest wydanie jednej lub więcej gier, więc bardziej praktyczne jest zlecanie tej części pracy dużym firmom, które mają zasoby, aby opracować zestaw narzędzi, i skupić się na samej grze. Szczególnie jeśli mówimy o małym zespole lub indie-developerze[5].

Jednym z najpopularniejszych obecnie silników jest Unreal Engine (UE), opracowany i utrzymywany przez firmę Epic Games. Unreal Engine to pakiet narzędzi do tworzenia gier, który

[5] Specjalista, który tworzy gry samodzielnie lub jest częścią małego zespołu deweloperskiego, który produkuje gry komputerowe i mobilne bez wsparcia finansowego dużych firm.

posiada szeroki zakres funkcji, od tworzenia gier 2D na urządzenia mobilne po projekty AAA na konsole i komputery.

Jest kilka powodów, dlaczego UE jest tak popularny:

- UE obsługuje dużą liczbę funkcji, dzięki czemu można w nim stworzyć niemal każdą grę.
- Unreal Engine ma wbudowany system skryptów wizualnych, który pozwala nawet nowicjuszom budować logikę gry bez zbyt wielu przeszkód.
- Silnik może być używany za darmo: umowa licencyjna Unreal Engine mówi, że można tak postępować dopóki gra nie przyniesie więcej niż 1 000 000 dolarów - po tym czasie musimy zapłacić 5% od przychodu.
- Możliwe jest stworzenie gier, która będą działać na wszystkich popularnych platformach: PlayStation, Xbox, Switch, PC, iOS, Android.
- Silnik posiada ogromną społeczność użytkowników, którzy tworzą tutoriale, dzielą się ze sobą doświadczeniami i pomagają rozwiązywać problemy. Dodatkową zaletą dużej społeczności jest wiele assetów (gotowy mechanizm lub funkcja) dostępnych do swobodnego wykorzystania we własnym projekcie.

Dużą zaletą Unreal Engine jest jego uniwersalność i dostępność - mogą z niego korzystać zarówno doświadczeni programiści, jak i nowicjusze, którzy po raz pierwszy podejmują się tworzenia gier. Rzecz w tym, że UE domyślnie obsługuje dwa języki programowania: tekstowy C++ oraz wizualny język Blueprints, gdzie logikę gry buduje się za pomocą połączonych ze sobą bloków. Takie podejście pomaga uczynić programowanie bardziej przejrzystym i łatwiejszym do zrozumienia. Dużą zaletą Blueprints jest to, że można z nich stosunkowo szybko złożyć podstawowy gameplay dla gry.

Chociaż Blueprints są łatwiejsze do zrozumienia, takie skrypty są prawie tak funkcjonalne jak C++ - istnieje tylko kilka rzadkich wyjątków, w których Blueprints jest nieco ograniczony.

UE może być również wykorzystywany w rozwoju projektu przez jedną osobę. Często twórcy indie korzystają z assetów innych deweloperów - pozwala to na szybsze ukończenie projektu, gdyż implementacja pewnych mechanik może zająć dużo czasu. W sklepie zapewnionym przez Epic Games są komponenty wykonane zarówno w Blueprints jak i C++.

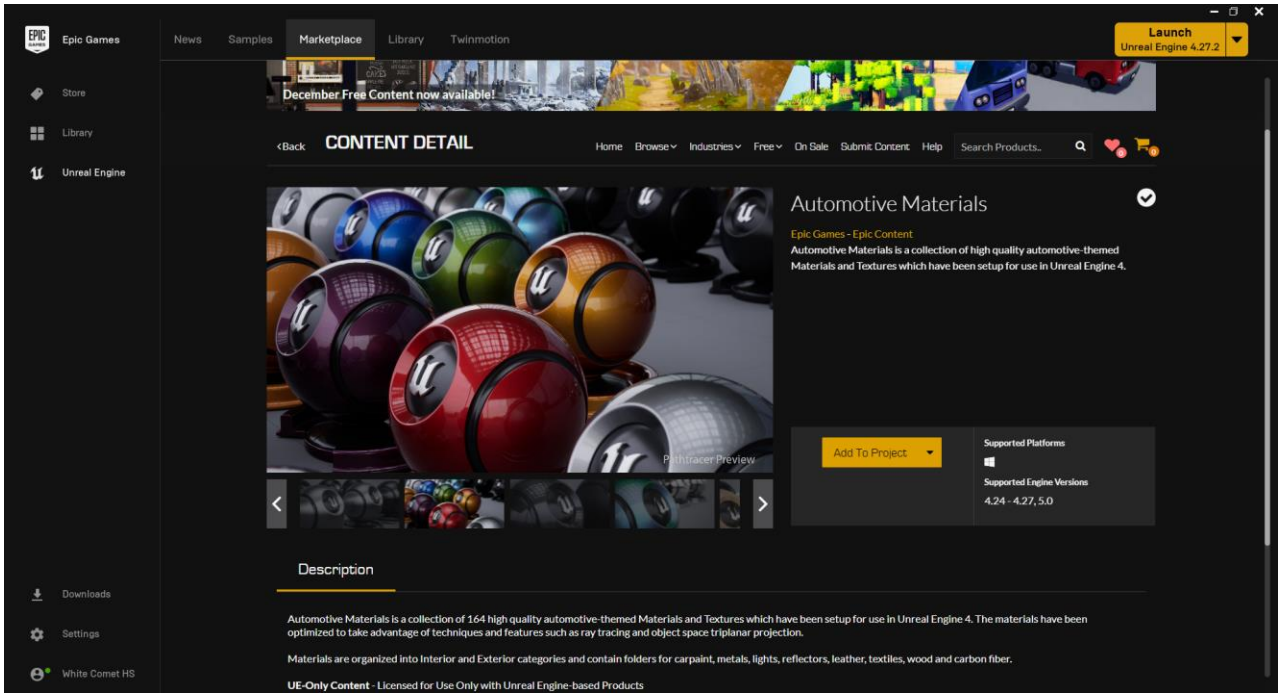
Możliwości zastosowania Unreal Engine już dawno wykroczyły poza ramy tworzenia gier. UE4 jest szeroko wykorzystywany podczas tworzenia filmów, reklamy, wizualizacji architektonicznej i symulacji szkoleniowych. Sekretem tej różnorodności jest wygoda silnika, jego ciągły rozwój i bezpłatność.

Epic Games, twórcy silnika Unreal Engine, śledzą i wspierają inne firmy oraz integrują ich rozwiązania w globalny produkt. Filozofia Epic Games ma na celu przełamanie barier dla kreatywności, a to przekłada się zarówno na ogólną politykę firmy, jak i na intuicyjność jej produktów.

Próg wejścia dla Unreal Engine 4 jest dość niski: nawet artysta bez doświadczenia w programowaniu będzie w stanie wykorzystać go na własne potrzeby. Nawigacja i ustawienia są pod wieloma względami podobne do innych programów.

Cenną cechą Unreal Engine jest szybkie prototypowanie gry. W prototypie szczególnie ważne jest natychmiastowe przekazanie atmosfery. UE oferuje wiele narzędzi do tego celu, od symulacji zjawisk naturalnych poprzez system cząstek, aż po postprocessing.

Ponadto, [2], Epic Games nawiązali współpracę z Quixel, rozszerzając bibliotekę Megascan i udostępnili ją za darmo przy użyciu w UE4. Z tego wynika biblioteka darmowych, bardzo szczegółowych skanów zintegrowanych z silnikiem z rozdzielczością od 2 do 8K (patrz rysunek 7).



Rysunek 7. Zdjęcie przedstawiające bibliotekę Megascan udostępnioną w internetowym sklepie Epic Games. Źródło: opracowanie własne.

Artyści mają w zasięgu ręki podstawowe elementy otoczenia, takie jak skały, zwalone drzewa, ziemię, trawę i wszelkiego rodzaju drobne rekwizyty (patrz rysunek 8). Dzięki temu szybciej i łatwiej można zaprezentować swój zasób, a także procesy związane z tworzeniem poziomu rozgrywki lub wizualizację architektoniczne.

Rysunek 8. Zdjęcie przedstawiające realistyczne środowisko leśne z wykorzystaniem silnika Unreal Engine 4 i Megascans. Źródło: https://www.unrealengine.com/marketplace/en-US/product/the-forest-v1.

Jeśli potrzebujemy czegoś konkretnego, prawie zawsze możemy znaleźć elementy kodu, blueprinty i gotowe komponenty w sklepie Epic Games.

W dodatku niedawno pojawiła się kolejna generacja silnika Unreal Engine (Unreal Engine 5). Główną różnicą w tej wersji są dwa nowe systemy Lumen i Nanite.

Pierwszy system, Lumen, to globalny model oświetlenia, który odbudowuje się w czasie rzeczywistym. Jednak trudno jeszcze powiedzieć, jak elastyczny i interaktywny jest ten system i jakie narzędzia będą dostępne dla programistów. Drugi system, Nanite, pozwoli na przeniesienie do gry 3D obiektów o bardzo wysokiej jakości.

Jednak patrząc w [15] można powiedzieć, że piąta wersja silnika od Epic Games nie ma charakteru rewolucyjnego, a raczej ewolucyjny. Unreal Engine 4 miał również kilka aktualizacji z dużymi systemami - Chaos Physics i efekty Niagara, są one również dołączone do UE5.

Dodatkowo twórcy zapowiedzieli wsteczną kompatybilność projektów na UE 4 i UE 5, co oznacza, że rdzeń silnika nie uległ znaczącym zmianom i system podstawowych klas, na którym opiera się rozgrywka, pozostanie taki sam.

Wniosek

Korzystając z książki [1], opanowanie Unreal Engine otwiera przed programistą wiele możliwości. Może zrobić grę sam, albo dołączyć do zespołu. Przy tym wcześniejsze doświadczenie

nie ma aż tak dużego znaczenia - początkujący programista nie będzie miał problemu z nauczeniem się Blueprints, aby stworzyć swój własny pierwszy projekt. Będzie to doskonała podstawa do dalszej pracy w C++, której znajomość pozwoli jeszcze bardziej zagłębić się w tworzenie wewnętrznej architektury gry.

Unity vs Unreal Engine

Rynek gier indie był kiedyś zdominowany przez Unity i do dziś jest silny w swoim segmencie, ale powoli traci grunt na rzecz Unreal Engine. Kiedy Unity pojawiło się na rynku, większość poważnych silników do tworzenia gier była silnikami płatnymi. Kiedy więc świat ujrzał pełnoprawną, darmową alternatywę, wielu postanowiło przełamać swój strach i wejść w rozwój swojego własnego projektu. Unity ewoluowało, oferując coraz ciekawsze rozwiązania zarówno dla gier 3D, jak i 2D.

Dziś największym konkurentem Unity jest Unreal Engine, który wyewoluował w kolejne darmowe i przyjazne użytkownikowi środowisko. Oba silniki posiadają rozbudowane zestawy narzędzi, w tym edytor krajobrazu, symulację fizyki, animację, ulepszone oświetlenie, wsparcie dla VR i wiele innych. Ostatnio jednak można zauważyć, że wielu deweloperów z małymi i średnimi projektami wykonanymi w Unity zaczyna przesiadać się na produkt Epic Games. Niektórzy próbują nawet stworzyć swoje pierwsze pełnoprawne gry na Unreal Engine 4.

W dzisiejszej pogoni za lepszą grafiką bardzo trudno zdziwić graczy dobrą grafiką, więc każdy produkt z małym budżetem i bez artystów z 20-letnim doświadczeniem jest uznawany za co najmniej prosty i nie bardzo atrakcyjny. Dlatego też, przystępując do tworzenia gry, programiści myślą o tym, jak osiągnąć fajne efekty w jak najprostszy sposób i tutaj Unreal Engine 4 zdecydowanie wygrywa. Po pierwszym uruchomieniu daje przysłowiowy „efekt wow" dzięki pięknemu oświetleniu, szczegółowym modelom i potężnym narzędziom graficznym.

W Unity łatwo można zobaczyć surową, przestarzałą scenę. Aby osiągnąć dobry efekt podobny do UE4, będziemy musieli się napracować: zmienić system renderingu z normalnego na HDRP, wymienić światła i skybox na ładniejsze. Ale nawet po transformacji prawdopodobnie nie uzyskamy tego samego rezultatu. Wynika to głównie z faktu, że Epic Games, jako twórca silników od lat 90-tych, zebrał mnóstwo doświadczeń i za każdym razem tworzył coś nowego, podczas gdy inni musieli nadrabiać zaległości. Unity jest bardziej nastawione na mniejsze projekty, wykonane głównie w 2D i na urządzenia mobilne.

4. Projekt systemu

Fizyczny model zachowania pojazdu powinien być modułem łatwo rozszerzalnym i intuicyjnym dla osób z niego korzystających. Moduł może służyć jako podstawa do dalszego rozwoju pełnej własnej gry lub być wykorzystany jako moduł dodatkowy (asset) w istniejącym projekcie do realizacji symulacji pojazdów kołowych. Możliwość dalszej rozbudowy pozwoli na wprowadzanie ulepszeń i różnych modyfikacji dla twórcy modelu fizycznego, natomiast intuicyjna struktura samego projektu pozwoli na samodzielną konfigurację i rozbudowę modelu fizycznego w zależności od potrzeb użytkownika.

4.1 Wymagania funkcjonalne

Kluczową cechą projektu jest możliwość zmiany, usunięcia lub dodania funkcjonalności do modelu fizycznego w zależności od wymagań i celów użytkownika. Nie jest więc możliwe precyzyjne określenie stałej grupy funkcjonalności dla modelu fizycznego. Przedstawione poniżej wymagania funkcjonalne stanowią podstawę danego modelu i tworzą równowagę pomiędzy realizmem a wydajnością.

- [WF1] Mechanizm symulujący działanie zawieszenia samochodowego.
- [WF2] Możliwość sterowania główną kamerą i kątem ustawienia kół.
- [WF3] Mechanizm tarcia kół.
- [WF4] Możliwość debugowania sił działających na każde osobne koło samochodu.
- [WF5] Realizacja silnika pojazdu.
- [WF6] Realizacja skrzyni biegów pojazdu.
- [WF7] Realizacja transmisji pojazdu.
- [WF8] Realizacja hamulców pojazdu.
- [WF9] Rodzaj powierzchni i współczynnik tarcia.
- [WF10] Efekty dźwiękowe i wizualne.

4.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne umożliwiają stworzenie uniwersalnego, łatwo rozszerzalnego i zoptymalizowanego modułu symulacji zachowania pojazdu. Główne wymagania obejmują:

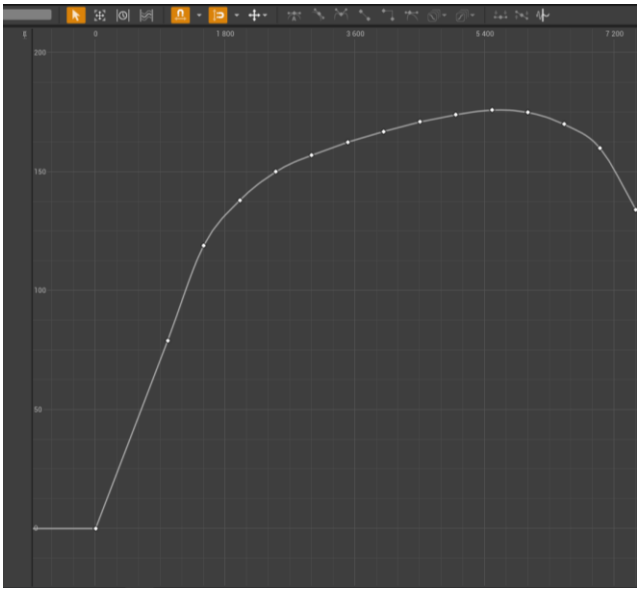
- [WNF1] Optymalizacja wykorzystania zasobów systemowych do obliczenia wymaganych zmiennych.
- [WNF2] Wysoka wydajność w przypadku zastosowania w gotowym projekcie.

Dodatkowo oczekiwane cechy to:

- Dokładność i niezawodność modelu fizycznego.
- Możliwość zintegrowania modułu z projektami gier mobilnych i konsolowych.
- Możliwość konfiguracji modelu fizycznego dla modeli pojazdów o różnych specyfikacjach, liczbie osi, rodzaju napędu i wymiarach.
- Intuicyjny i łatwy proces konfiguracji modelu fizycznego.
- Prosty i bogaty w informacje interfejs z wyprowadzeniem aktualnych informacji i ruchu wybranego pojazdu.

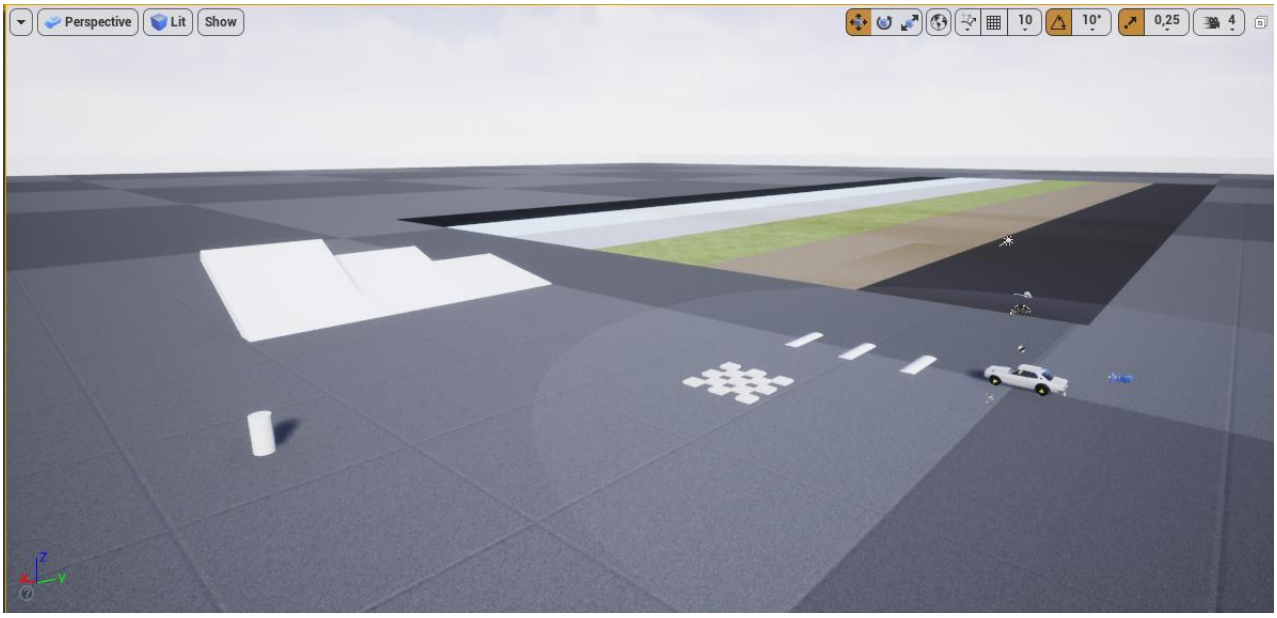
4.3 Przypadki użycia

Głównym i najbardziej kluczowym sposobem wykorzystania danego modelu fizycznego jest użycie go jako assetu (modułu dodatkowego) w projekcie gry. Sam model fizyczny może być dystrybuowany poprzez platformę dla wirtualnych aktywów wspierany przez Epic Games. Asset jest w pełni kompatybilny z wersjami 4 i 5 silnika gry, dzięki kompatybilności wstecznej wprowadzonej w piątej generacji silnika. Pierwszym krokiem jest zaimportowanie przez użytkownika modułu do swojego projektu. Import odbywa się za pomocą wirtualnej platformy po dodaniu aktywa do swojej biblioteki. W drugim kroku użytkownik konfiguruje sam pojazd, korzystając z przygotowanych wcześniej modeli 3D. Sam proces konfiguracji obejmuje wymianę modeli 3D nadwozia i kół, wybór punktów odniesienia dla amortyzatorów, regulację rozstawu kół i prześwitu samochodu oraz regulację środka ciężkości nadwozia. Kolejnym krokiem jest dogłębne dostosowanie parametrów technicznych poszczególnych elementów pojazdu. Użytkownik może regulować liczbę biegów, przełożenia skrzyni biegów przekładni głównej, jak i dla poszczególnych biegów, a także ustawić czas wymagany do zmiany biegu. Możliwa jest regulacja wykresu momentu obrotowego silnika (patrz rysunek 9), a także ustawienia napędu pojazdu (FWD, RWD, 4WD).



Rysunek 9. Zdjęcie przedstawiające wykres momentu obrotowego silnika. Źródło: opracowanie własne.

Po szczegółowym dostosowaniu charakterystyki samochodu, użytkownik może przetestować gotowy pojazd na placu testowym i w zależności od uzyskanych wyników ponownie przeprowadzić ostateczną regulację parametrów samochodu (patrz rysunek 10).



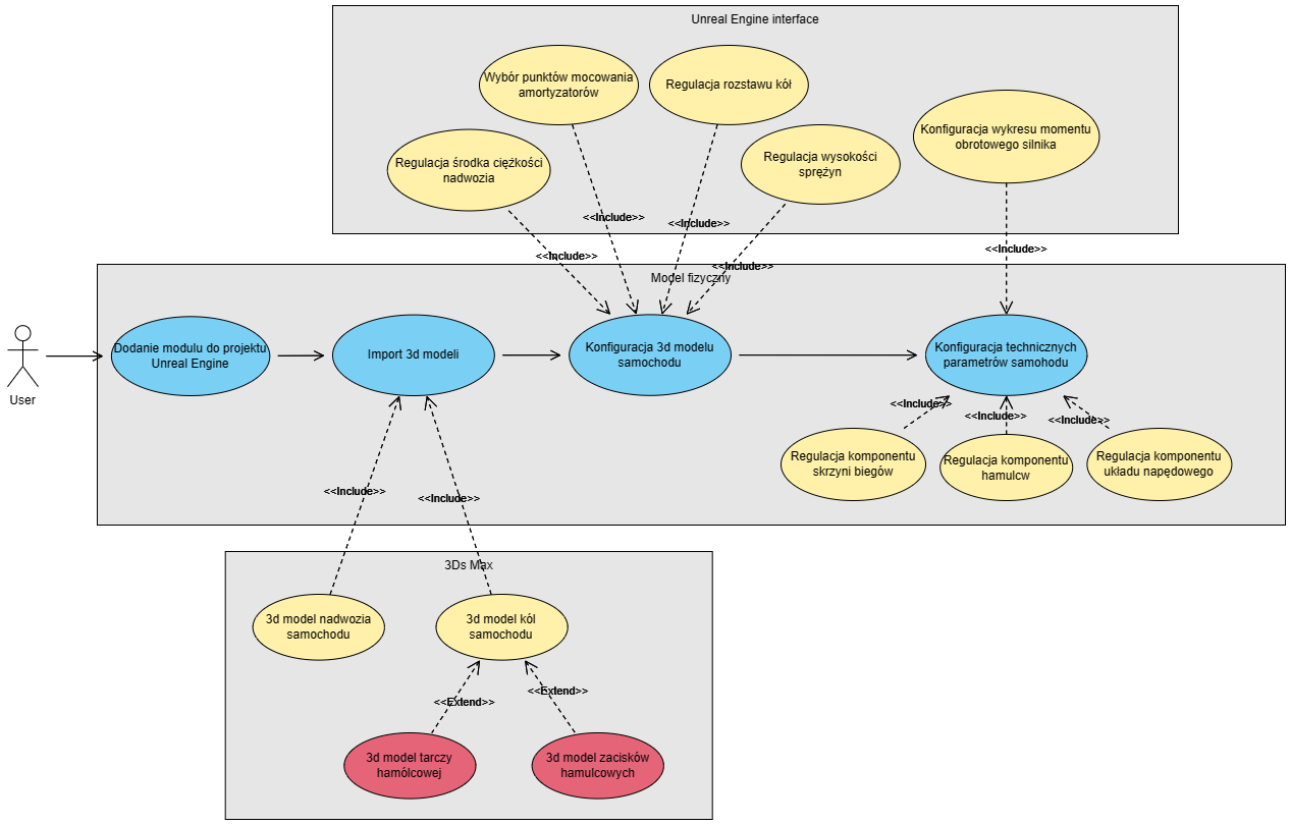
Rysunek 10. Zdjęcie przedstawiające plac testowy. Źródło: opracowanie własne.

Podczas testów wszystkie kluczowe parametry samochodu są wyświetlane za pomocą przygotowanego interfejsu (patrz rysunek 11), co w efekcie daje możliwość użytkownikowi bardziej szczegółowo przeanalizować otrzymany samochód.

Rysunek 11. Zdjęcie przedstawiające interfejs do debugowania. Źródło: opracowanie własne.

Na koniec wszystkich ustawień użytkownik otrzymuje swoją klasę pojazdu z wymaganymi cechami, gotową do użycia w swoim projekcie. W razie potrzeby zawsze można wprowadzić modyfikacje i zmiany w tej klasie pojazdu.

Na rysunku 12 przedstawiony jest schemat użycia modelu fizycznego jako oddzielnego podprojektu (assetu):



Rysunek 12. Schemat przedstawiający przypadek użycia modelu fizycznego. Źródło: opracowanie własne.

4.4 Struktury komponentów

Jednym ze sposobów definiowania typów użytkownika w Blueprints jak i w języku C++ jest wykorzystanie struktur. Metoda ta została odziedziczona z języka C. Struktura to narzędzie do łączenia wielu zmiennych w nowy rodzaj zmiennej.

Struktury mogą zawierać zmienne, tablice i inne struktury, ale struktury nie mogą zawierać funkcji. Dzięki strukturom program jest bardziej zwarty i elastyczny na wprowadzanie zmian.

W dalszej części pracy zostaną przedstawione przykładowe struktury oddzielnych komponentów wykorzystane podczas obliczeń sił działających na pojazd.

W tabeli 1 zostały umieszczone parametry struktury „*SuspensionStruct”* z krótkim opisem każdego parametru i jego wartością domyślną:

Tabela 1. Schemat struktury „SuspensionStruct". Źródło: opracowanie własne.

Struktura komponentu zawieszenia samochodowego			
Nazwa parametru	**Typ**	**Opis**	**Wartość domyślna**
restLength	Float	Długość spoczynkowa sprężyny	50 cm
travel	Float	Zakres wahań sprężyny	10 cm
stiffness	Float	Stopień sprężystości	500 kg/s^2
damper	Float	Stopień absorpcji wahań sprężyny	10 kg/s
force_min	Float	Minimalna siła sprężyny	-2000 kg*cm/s^2
force_max	Float	Maksymalna siła sprężyny	6000 kg*cm/s^2

Tabela 2 przedstawia parametry struktury „*WheelStruct"* z krótkim opisem każdego parametru:

Tabela 2. Schemat struktury „WheelStruct". Źródło: opracowanie własne.

Struktura komponentu koła samochodowego		
Nazwa parametru	**Typ**	**Opis**
Radius	Float	Promień koła [cm]
Mass	Float	Masa koła [kg]
CornerStiffness	Float	Sztywność kątowa koła [0-1]
LongStiffness	Float	Sztywność wzdłużna koła [0-1]
Mesh	Static mesh	3D obiekt koła

Scale	Vector 2D	Rozmiar koła

Rysunek 13 przedstawia domyślne wartości struktury *WheelStruct*:

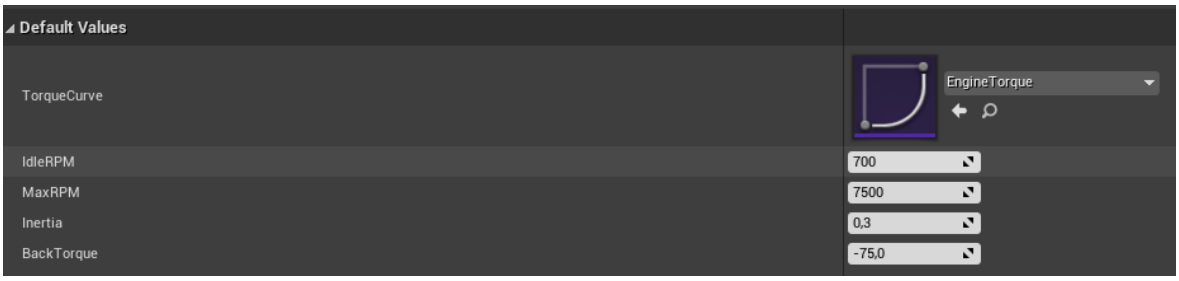
Rysunek 13. Zdjęcie przedstawiające domyślne wartości parametrów struktury 'WheelStruct'. Źródło: opracowanie własne.

W tabeli 3 zaprezentowane zostały parametry struktury „*EngineStruct"* z krótkim opisem każdego parametru i jego wartością domyślną:

Tabela 3. Schemat struktury „EngineStruct". Źródło: opracowanie własne.

Struktura komponentu silnika		
Nazwa parametru	**Typ**	**Opis**
TorqueCurve	Curve Float	Wykres momentu obrotowego silnika
IdleRPM	Integer	Obroty silnika w stanie czekania
MaxRPM	Integer	Maksymalne obroty silnika
Inertia	Float	Moment bezwładności silnika
BackTorque	Float	Odwrotny moment obrotowy silnika

Rysunek 14 przedstawia domyślne wartości struktury *EngineStruct*:



Rysunek 14. Zdjęcie przedstawiające domyślne wartości parametrów struktury „EngineStruct". Źródło: opracowanie własne.

Tabela 4 przedstawia parametry struktury „*BrakeStruct”* z krótkim opisem każdego parametru:

Tabela 4. Schemat struktury „BrakeStruct”. Źródło: opracowanie własne.

Struktura komponentu hamulców		
Nazwa parametru	**Typ**	**Opis**
BrakeRotorMesh	Static Mesh	3D obiekt tarczy hamulcowej
BrakeCaliperMesh	Static Mesh	3D obiekt zacisków hamulcowych

W tabeli 5 zaprezentowane zostały parametry struktury „*SurfaceStruct”* z krótkim opisem każdego parametru i jego typem:

Tabela 5. Schemat struktury „SurfaceStruct”. Źródło: opracowanie własne.

Struktura komponentu powierzchni		
Nazwa parametru	**Typ**	**Opis**
surface	String	Typ powierzchni
friction	Float	Współczynnik tarcia
roll_resistance	Float	Współczynnik oporu toczenia
roughness	Float	Współczynnik chropowatości powierzchni
bias	Float	Pionowe wgłębienie do symulacji luźnych powierzchni

Rysunek 15 przedstawia domyślne wartości struktury *SurfaceStruct*:

Rysunek 15. Zdjęcie przedstawiające domyślne wartości parametrów struktury „SurfaceStruct”. Źródło: opracowanie własne.

Tabela 6 przedstawia parametry struktury „*AxisDataStruct”* z krótkim opisem każdego parametru i jego typem:

Tabela 6. Schemat struktury „AxisDataStruct”. Źródło: opracowanie własne.

Struktura komponentu osi pojazdu		
Nazwa parametru	**Typ**	**Opis**
Location	Vector	Pozycja osi
SteeringMult	Float	Współczynnik ustawienia kół na osi
BrakeRotorOffset	Float	Odstęp tarczy hamulcowej
BrakeCaliperLocation	Vector	Pozycja zacisków hamulcowych
TorqueMult	Float	Współczynnik rozdziału momentu obrotowego między osiami

Rysunek 16 przedstawia domyślne wartości struktury *AxisDataStruct*:

Rysunek 16. Zdjęcie przedstawiające domyślne wartości parametrów struktury „AxisDataStruct”. Źródło: opracowanie

5. Implementacja systemu

5.1 Struktura projektu

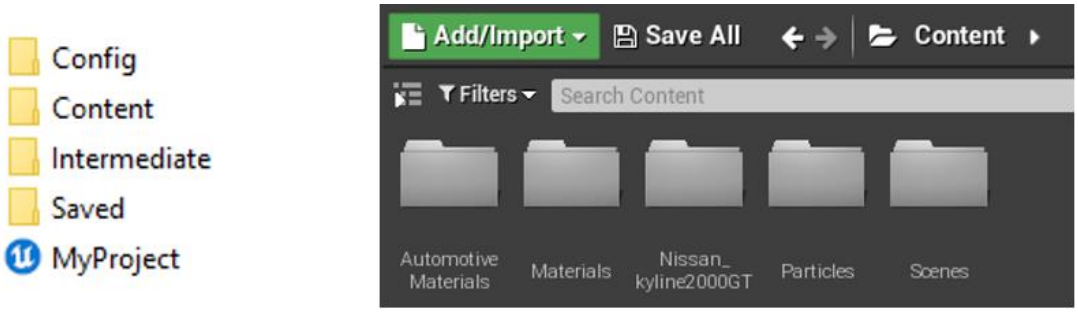
Podczas tworzenia projektu opartego na blueprintach przy użyciu Epic Games Launcher, edytor kopiuje kolekcję folderów i plików instalacyjnych do wskazanej przez użytkownika lokalizacji na komputerze. Po utworzeniu projektu w folderze pojawią się następujące sekcje: Config, Content, Intermediate, Saved oraz plik .uproject (patrz rysunek 26 po lewej stronie).

- Config zawiera instalacyjne pliki .ini, które przechowują domyślne ustawienia edytora i projektu.
- Content przechowuje wszystkie assety projektu zaimportowane przez użytkownika lub utworzone bezpośrednio w edytorze.
- Intermediate przechowuje ustawienia robocze projektu w plikach .ini i pliki konfiguracyjne, wraz z plikiem CachedAssetRegistry.bin.
- Saved zawiera pliki autozapisu i kopii zapasowe edytora, folder Collections dla assetów, zorganizowanych w kolekcje w przeglądarce projektów, ustawienia Config projektu dla platform docelowych, pliki z logami zapisów edytora oraz obrazek projektu w formacie .png, który jest wyświetlany w Launcherze.

Wszystkie pliki z rozszerzeniem .ini są plikami konfiguracyjnymi, które przechowują ustawienia edytora, silnika i gry. Wszelkie zmiany dokonane w projekcie lub ustawieniach są wyświetlane w tych plikach.

W folderze Content edytor przechowuje wszystkie zaimportowane i utworzone treści projektu. Struktura folderu Content z kolei zawiera następujące sekcje (patrz rysunek 17 po prawej stronie):

- AutomotiveMaterials zawiera kolekcję wysokiej jakości materiałów i tekstur, które są dystrybuowane za darmo w Epic Games Store.
- Materials zawiera własne materiały typu nawierzchni, takie jak żwir, piasek, lód i śnieg.
- Nissan_Skyline2000GTR zawiera wszystkie niezbędne modele 3D, skrypty, struktury i pliki do wyświetlania i poprawnego symulowania samochodu marki Nissan.
- Particles zawiera pliki systemu cząsteczek potrzebne do tworzenia efektów wizualnych.
- Scenes zawiera scenę testową dla testów pojazdów.



Rysunek 17. Zdjęcie przedstawiające strukturę projektu (po lewej stronie) i strukturę folderu Content (po prawej stronie). Źródło: opracowanie własne.

5.2 Przypisanie klawiszy

Użycie mapowań klawisz daje możliwość mapowania wielu klawiszy do tego samego zachowania za pomocą jednego wiązania i również pozwala na interpretację klawiszy wejściowych, które nie są wejściami osiowymi (na przykład A/D dla skrętu w lewo lub w prawo).

Aby stworzyć możliwość sterowania pojazdem w edytorze Unreal Engine, stworzono osie i przypisano odpowiednie klawisze z mnożnikiem wartości wynikowej po ich naciśnięciu (patrz rysunek 18).

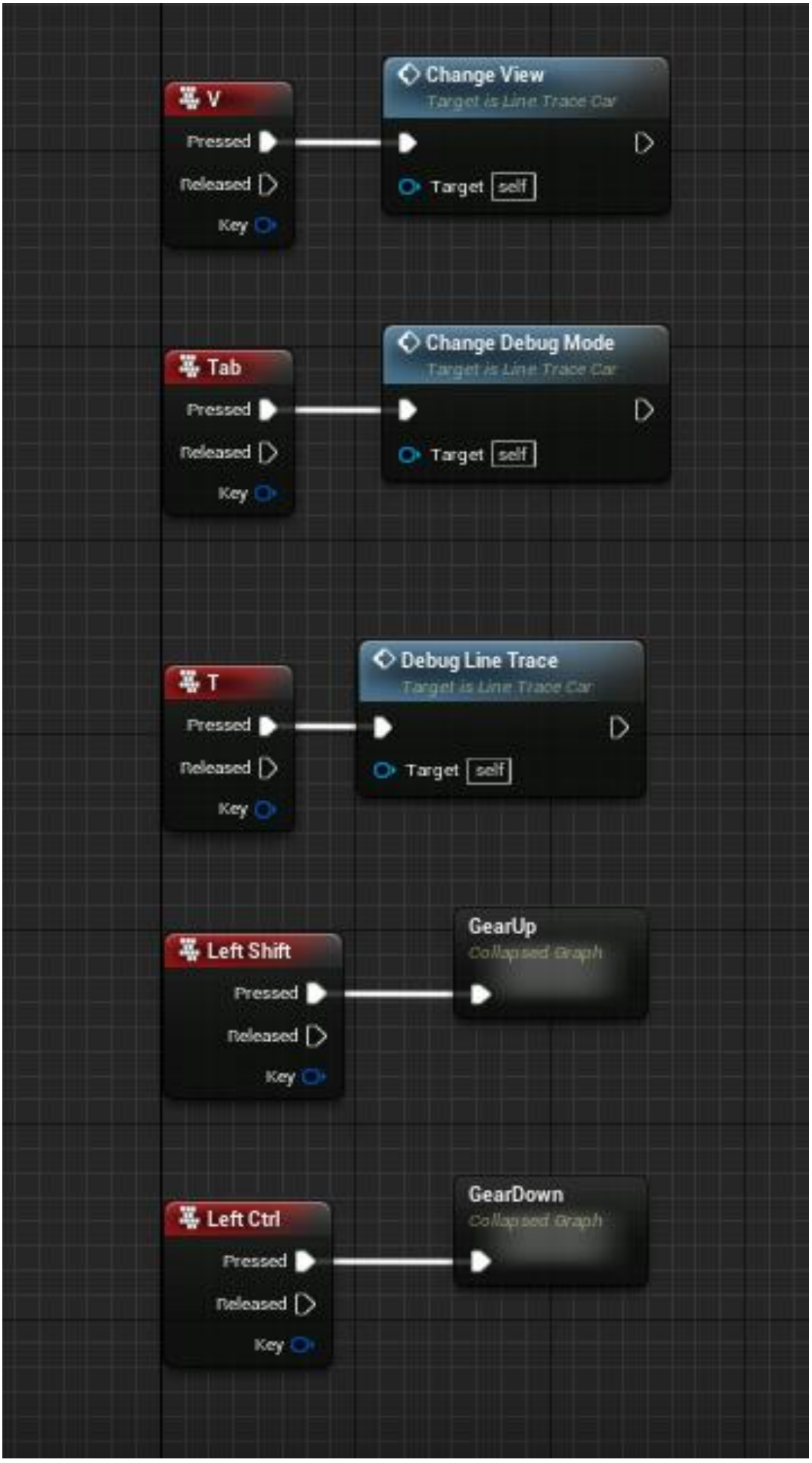
Rysunek 18. Zdjęcie przedstawiające stworzone osi. Źródło: opracowanie własne.

Oś Right odpowiada za obracanie kół za pomocą klawiszy A i D.

Oś Throttle odpowiada za stopień nacisku na pedał gazu w samochodzie (klawisz W).

Oś Brake odpowiada za stopień nacisku na pedał hamulca (klawisz S).

Oś HandBrake odpowiada za używanie hamulca ręcznego (klawisz W).



Rysunek 19. Zdjęcie przedstawiające mapping klawiszy oraz funkcje za ktre one odpowiadają. Źródło: opracowanie własne.

Na rysunku 19 przedstawiony schemat klawisz wraz z funkcjami obsługującymi ich wciśnięcie.

- Klawisz *V* odpowiada za zmianę kamery pojazdu.
- Klawisz *Tab* włącza i wyłącza siły działające na koła pojazdu.
- Klawisz *T* ukrywa koła pojazdu, aby przetestować mechanizm LineTrace.
- Klawisze *lewy Shift* i *lewy Ctrl* odpowiednio zwiększają i zmniejszają prędkość pojazdu.

Dodatkowo istnieje możliwość zmiany kamery podążającej za pojazdem (kamera w widoku z trzeciej osoby i w widoku z pierwszej osoby), możliwość włączenia trybu testowania mechanizmu LineTrace i sił, działających na każde koło pojazdu oraz opcja zmiany biegów pojazdu. Dane funkcjonalności zostały zrealizowane na podstawie zdarzeń akcji (Action events).

Główną różnicą pomiędzy zdarzeniami osi i zdarzeniami akcji jest to, że zdarzenia osi są wywoływane co klatkę, przekazując aktualną wartość osi, podczas gdy zdarzenia akcji (Action events) mają wyjścia Pressed i Released, gdy odpowiednie klawisze są wciskane.

5.3 Komponenty

Sam model fizyczny zachowania pojazdu składa się z dużej liczby funkcji i poszczególnych zmiennych, które można podzielić na 6 głównych elementów: zawieszenie, koła, silnik, skrzynia biegów, przekładnia i hamulce. Każdy z elementów realizuje określoną grupę funkcji i w każdej chwili może być udoskonalony lub uproszczony w zależności od potrzeb użytkownika.

5.3.1 Zawieszenie

Zawieszenie jest bardzo ważnym elementem podczas realizacji algorytmu zachowania pojazdu, ponieważ jego prawidłowe działanie decyduje o tym, jak pojazd będzie się zachowywał na zakrętach i na nierównej nawierzchni. Jednak pomimo tego, że w inżynierii samochodowej zawieszenie jest bardzo złożonym i krytycznym elementem, Unreal Engine pozwala na stworzenie stosunkowo prostego mechanizmu, który symuluje działanie zawieszenia w prawdziwym samochodzie. Na listingu 1 przedstawione zostały funkcje zapewniające działanie komponentu zawieszenia.

Podstawą komponentu zawieszenia jest mechanizm *LineTrace* oparty na liniowym, punktowym sprawdzaniu obecności przeszkód. Funkcja ta wypuszcza skierowany wektor z punktu początkowego do punktu końcowego i zwraca jako wynik pozycję punktu, w którym wektor zderza się z obiektem.

Listing 1. Kolejność metod odpowiadających za komponent zawieszenia. Źródło: opracowanie własne.

Dlatego kluczowymi punktami w tym mechanizmie są dwa punkty, *LineStart* - punkt, wktórym zaczyna się wektor i *LineEnd* - punkt, w którym nastąpiła kolizja wektora z przeszkodą (patrz rysunek 20). Jako punkt wyjścia używana jest pozycja pustego komponentu o nazwie *TopLink*. *LineEnd* jest z kolei obliczany według wzoru:

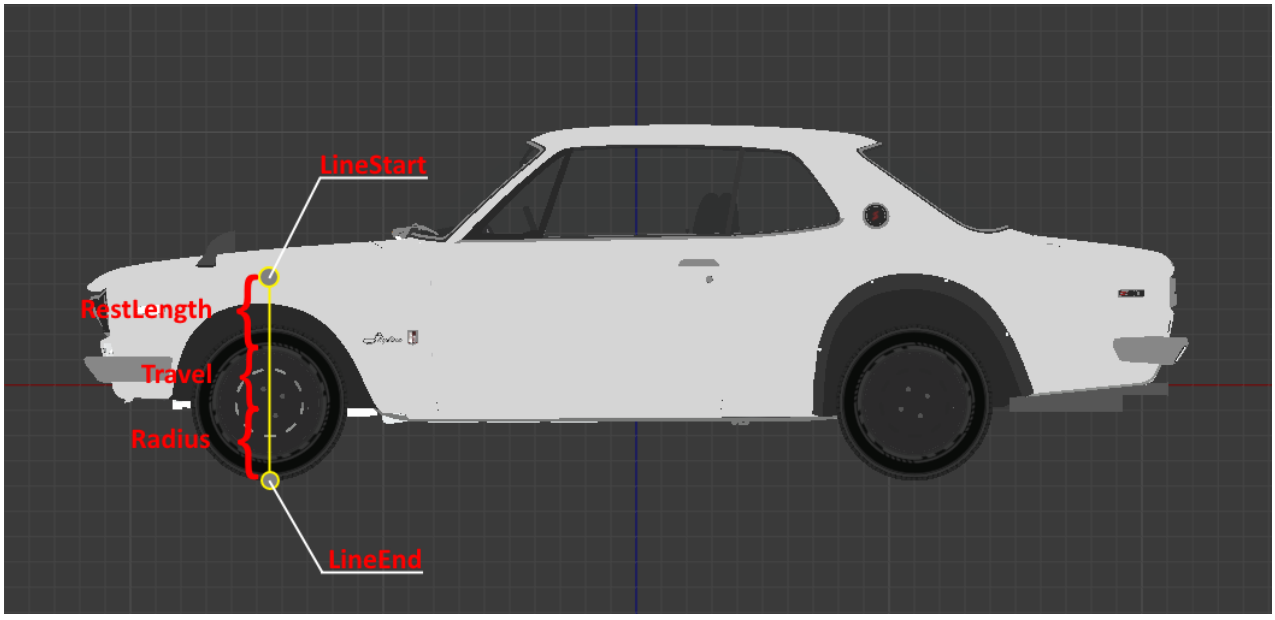
$$LineEnd = TopLink\ World\ Location - TopLink\ Up\ Vector\ (MaxLength + Radius), \quad (1)$$

gdzie $MaxLength = RestLength + Travel$

W przypadku kiedy wektor styka się z obiektem, to długość sprężyny obliczamy według wzoru:

$$Length = Vector\ Length\ (TopLink\ World\ Location - (HitRes\ Location + TopLink\ Up\ Vector * Radius)), \quad (2)$$

w przeciwnym razie długość sprężyny jest pobierana z parametru maksymalnej długości sprężyny: $Length = MaxLength$.



Rysunek 20. Zdjęcie przedstawiające z czego składa się długość wektora w mechanizmie LineTrace. Źródło: opracowanie własne.

Teraz, gdy znamy już długość samej sprężyny, pozostaje tylko obliczyć siłę, z jaką sprężyna będzie wywierać nacisk na nadwozie samochodu. Na podstawie tego wzoru można wyznaczyć siłę sprężyny: $Q = k * (L_0 - L)$, gdzie:

k – współczynnik sprężystości sprężyny,

L_0 – długość sprężyny w stanie swobodnym,

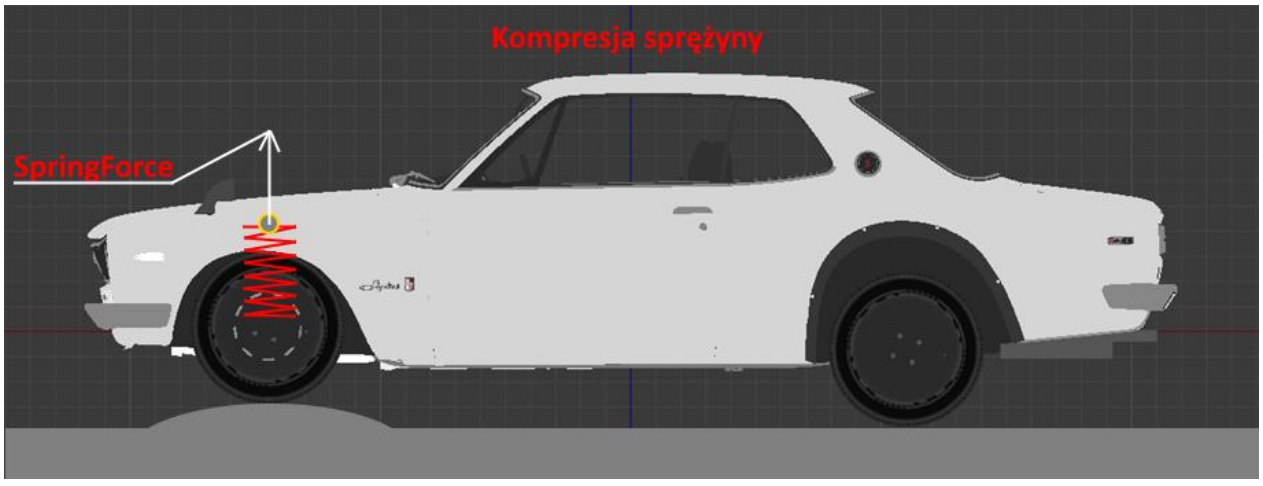
L – długość sprężyny po obciążeniu masą m.

Jeśli dostosujemy ten wzór do naszych warunków i parametrów, otrzymamy:

$$SpringForce = Stiffness * (RestLength - Length), \quad (3)$$

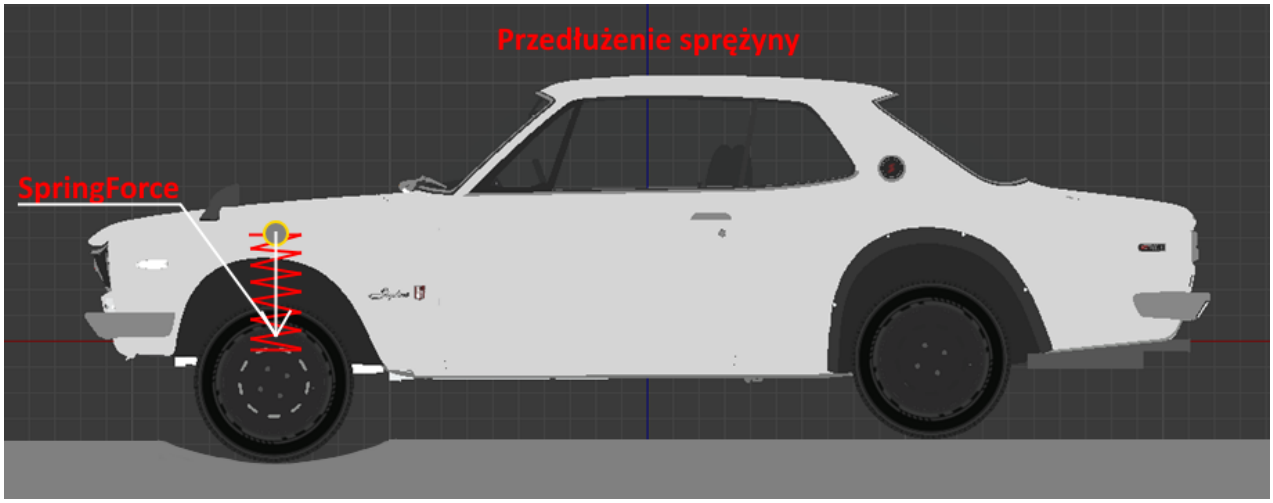
Siła sprężyny w rzeczywistości działa w obu kierunkach, tzn. zarówno na korpus, jak i na koła. Biorąc jednak pod uwagę, że masa koła w porównaniu z masą ciała jest dość mała, siłę sprężyny działającą na koło można pominąć na potrzeby optymalizacji i wydajności całego algorytmu. Dlatego w poniższych obliczeniach uwzględniono jedynie siłę sprężyny działającą na nadwozie pojazdu, a ruch samych kół realizowano za pomocą przyspieszenia bezwzględnego.

Należy zauważyć, że jeśli wartość $RestLength - Length$ jest dodatnia, obserwujemy efekt kompresji sprężyny, natomiast jeśli wartość jest ujemna, obserwujemy efekt wydłużenia. Zgodnie z pierwszym przypadkiem koło popycha nadwozie samochodu do góry (patrz rysunek 21).



Rysunek 21. Zdjęcie przedstawiające zjawisko kompresji sprężyny. Źródło: opracowanie własne.

Natomiast w drugim przypadku koło ciągnie nadwozie samochodu w dół, co można zauważyć na rysunku 22.



Rysunek 22. Zdjęcie przedstawiające zjawisko przedłużenia sprężyny. Źródło: opracowanie własne.

Ponadto żadne zawieszenie samochodu nie obywa się bez amortyzatorów, których głównym zadaniem jest zmniejszenie drgań sprężyny. Sprężyny pojazdu są wahadłami, które mechanicznie oscylują (wracając do położenia równowagi). W efekcie bez amortyzatorów pojazd miałby wrażenie, że unosi się na fali, co jest niezwykle szkodliwe dla prowadzenia.

Aby obliczyć siłę amortyzatora, należy skorzystać ze wzoru:

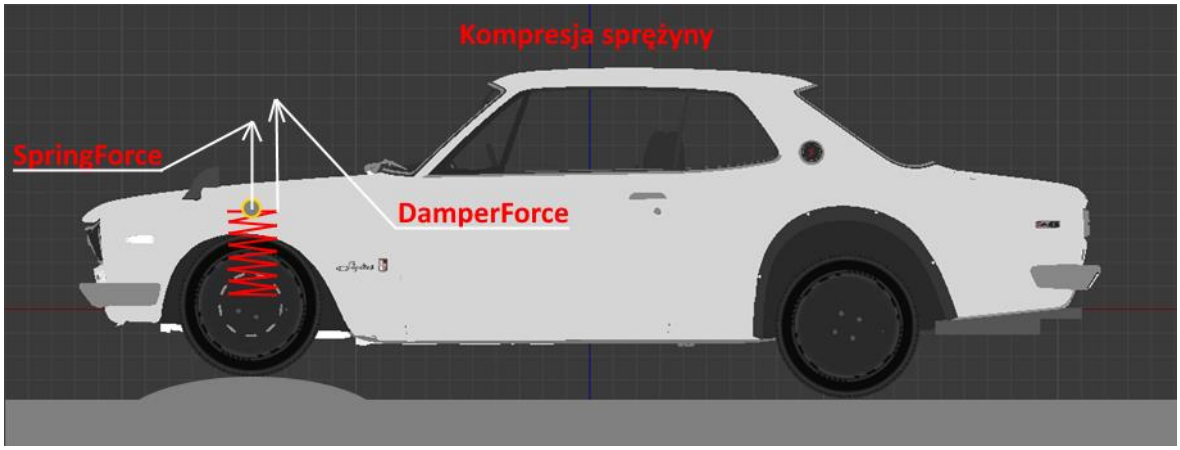
$$DamperForce = damper * SuspensionSpeed, \quad (4)$$

$$SuspensionSpeed = (LastLength - Length)/deltaTime, \quad (5)$$

gdzie: *LastLength* - długość sprężyny na ostatniej klatce,

Length - rzeczywista długość sprężyny dla danej klatki,

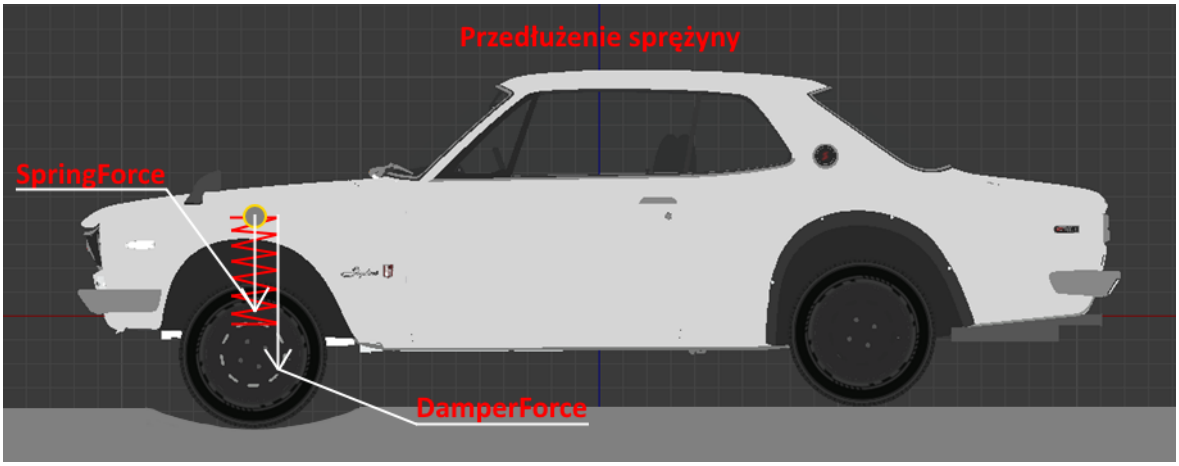
deltaTime - odstęp czasu między poprzednią a obecną klatką.



Rysunek 23. Zdjęcie przedstawiające działanie amortyzatora przy kompresji sprężyny. Źródło: opracowanie własne.

Na rysunku 23 możemy zauważyć, że gdy sprężyna jest ściskana, amortyzator tworzy opór, próbując rozciągnąć sprężynę, zmniejszając w ten sposób siłę *SpringForce* i zapobiegając dalszej oscylacji.

Sytuacja jest podobna, gdy sprężyna jest rozciągana, amortyzator daje opór ściskając sprężynę i zmniejszając siłę działającą na nadwozie samochodu (patrz rysunek 24).



Rysunek 24. Zdjęcie przedstawiające działanie amortyzatora przy przedłużeniu sprężyny. Źródło: opracowanie własne.

Znając siłę sprężyny i amortyzatora, można łatwo uzyskać całkowitą siłę działającą na nadwozie samochodu:

$$Fz = (SpringForce + DamperForce) * 100, \qquad (6)$$

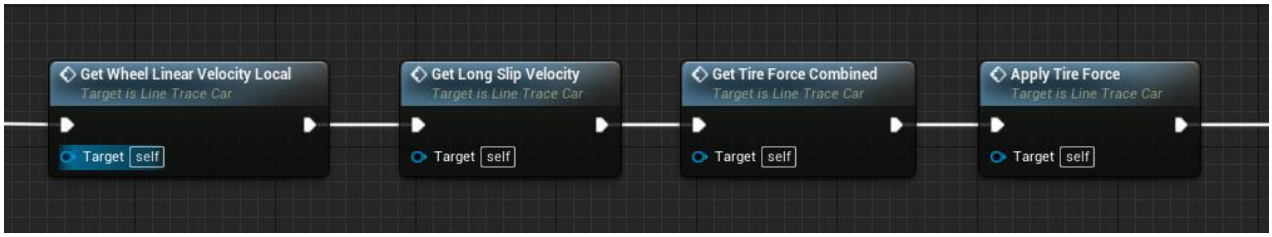
gdzie *Fz* - zmienna lokalna przechowująca wektor siły pionowej działającej na nadwozie w danym momencie. Należy również pamiętać, że mnożenie przez 100 jest konieczne, aby uzyskać prawidłową wartość siły w kg*cm/s^2, ponieważ jednostką długości w Unreal Engine jest centymetr (patrz listing 2).

Listing 2. Mnożenie zmiennej Fz przez 100 dla uzgodnienia jednostki siły. Źródło: opracowanie własne.

Pionowa siła sprężyny jest obliczana dla każdego koła indywidualnie, przy użyciu tablicy z referencjami dla każdego punktu odniesienia sprężyny koła. Takie podejście umożliwia symulację pracy zawieszenia pojazdu z dużą wiernością i realizmem.

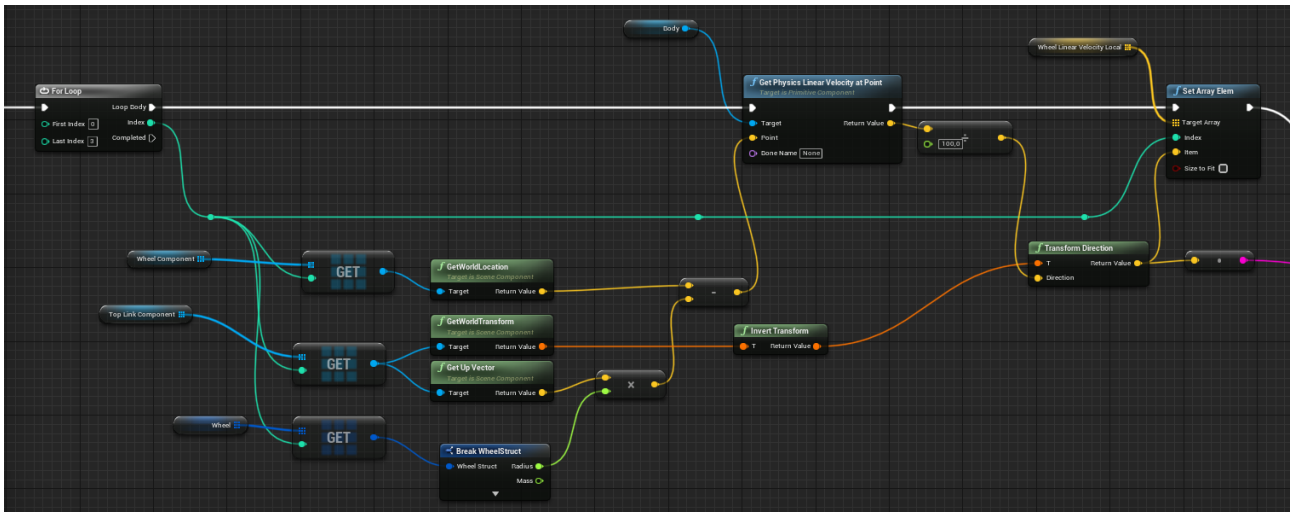
5.3.2 Koła i siła tarcia

Komponent koła w danym modelu fizycznym jest kluczowym elementem, ponieważ fizyczny model tarcia koła określa, jak pojazd hamuje, skręca i przyspiesza. Im bardziej zaawansowany model tarcia, tym dokładniej i bardziej realistycznie zachowuje się pojazd, ale im więcej parametrów oblicza model tarcia, tym większe jest obciążenie obliczeniowe całego modelu fizycznego, gdyż jak i w przypadku zawieszenia obliczenia wykonywane są dla każdego koła indywidualnie. Dlatego tak ważne jest znalezienie równowagi pomiędzy realizmem a wydajnością dla modelu tarcia koła. Kolejność wykonywanych funkcji została zaprezentowana na listingu 3.



Listing 3. Kolejność metod obliczających siłę tarcia opony. Źródło: opracowanie własne.

Zanim zaczniemy obliczać tarcie dla koła, musimy znaleźć jego prędkość liniową w przestrzeni lokalnej. Pierwszą rzeczą, którą należy zrobić, jest znalezienie prędkości liniowej koła we współrzędnych świata w punkcie jego kontaktu z drogą za pomocą funkcji *Get Physics Linear Velocity at Point*. Funkcja zwraca wektor prędkości liniowej we współrzędnych światowych, który może być przetłumaczony na współrzędne lokalne za pomocą funkcji *Transform Direction* (patrz listing 4).



Listing 4. Proces konwersji wektora o współrzędnych światowych na współrzędne lokalne. Źródło: opracowanie własne.

Oprócz prędkości liniowej należy obliczyć moment bezwładności koła ze wzoru:

$$Inertia = \frac{mr^2}{2}, \quad (7)$$

gdzie: m - masa koła,

r - jego promień.

Znając moment bezwładności, można otrzymać przyspieszenie kątowe koła:

$$AngularAcceleration = \frac{DriveTorque}{Inertia}, (8)$$

gdzie *DriveTorque* jest momentem obrotowym przekazywanym do koła przez przekładnię, a mając przyspieszenie kątowe, można znaleźć prędkość kątową koła:

$$AngularVelocity \mathrel{+}= AngularAcceleration * deltaTime. \quad (9)$$

Wzory w ruchu kątowym są podobne do wzorów dla ruchu liniowego, a więc:

$$DriveTorque = Inertia * AngularAcceleration \quad (10)$$

$$AngularAccelaration = \frac{DriveTorque}{Inertia} (11)$$

$$AngularVelocity \mathrel{+}= AngularAccelaration * deltaTime \ (12)$$

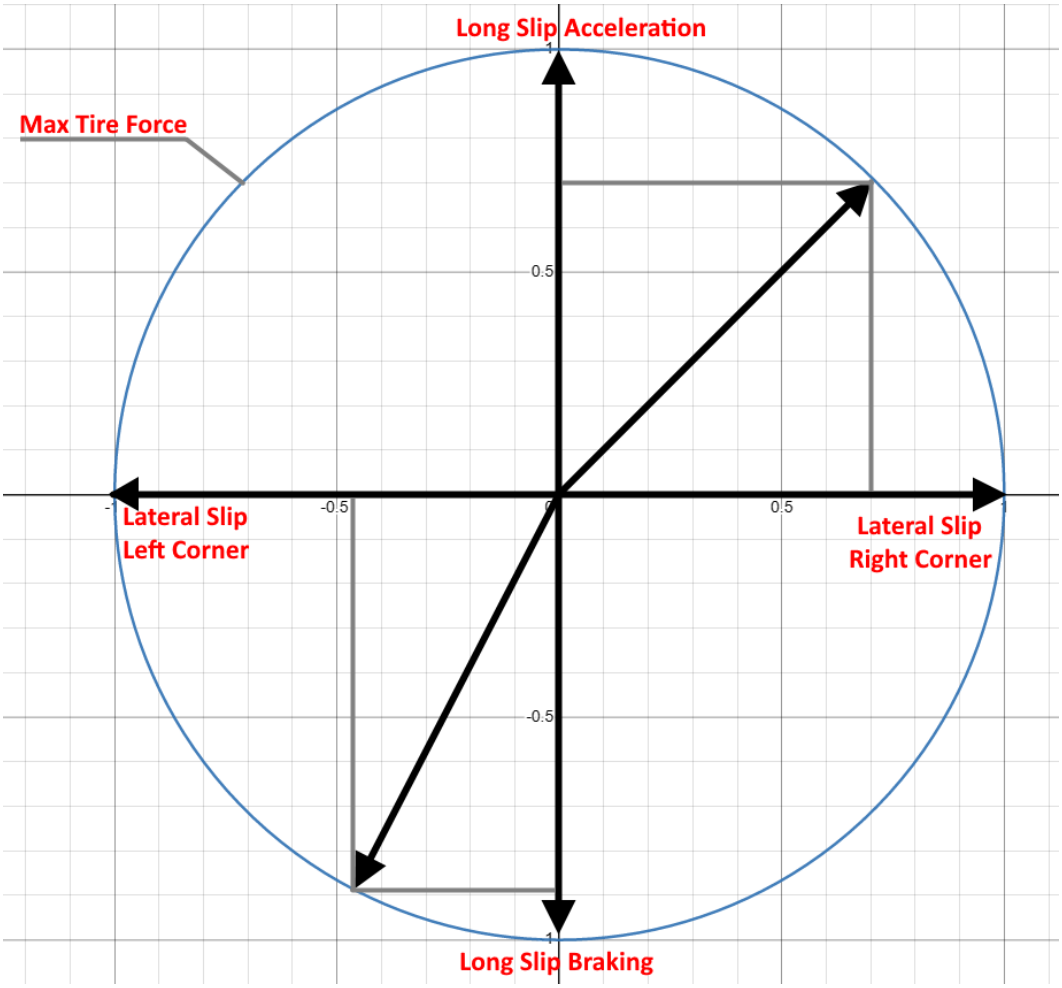
Teraz, gdy znamy prędkość kątową i prędkość liniową koła, można zaimplementować model tarcia. Sam model oparty jest na kombinowanym poślizgu i teorii koła tarcia. Poślizg kombinowany oznacza, że poślizg koła może być nie tylko wzdłużnym lub poprzecznym. Przykładowo, podczas jazdy samochodem z napędem na przednie koła po ciasnym łuku, kierowca mocno naciska na pedał gazu, co oznacza, że przednie koła są poddawane znacznym obciążeniom, ponieważ oprócz poślizgu poprzecznego, koło poddawane również poślizgu wzdłużnemu, przyspieszając samochód, w wyniku czego powstaje silna podsterowność, a samochód jest w stanie prawie pozbawionym sterowności. Z kolei teoria koła tacia ogranicza maksymalną możliwą siłę tarcia opony poprzez uniemożliwienie wyjścia wektora siły tarcia poza obwód umownego koła (patrz rysunek 25).

Najpierw obliczmy poślizg wzdłużny i poprzeczny opony. Wzór na obliczenie poślizgu wzdłużnego jest następujący:

$$LongSlipVelocity = WheelSpeed - CarSpeed, \quad (13)$$

gdzie: $WheelSpeed = AngularVelocity * Radius$

$$CarSpeed = LinearLocalVelocity.x$$



Rysunek 25. Zdjęcie przedstawiające teorię kręgu tarcia opartą na kombinowanym poślizgu. Źródło: opracowanie własne.

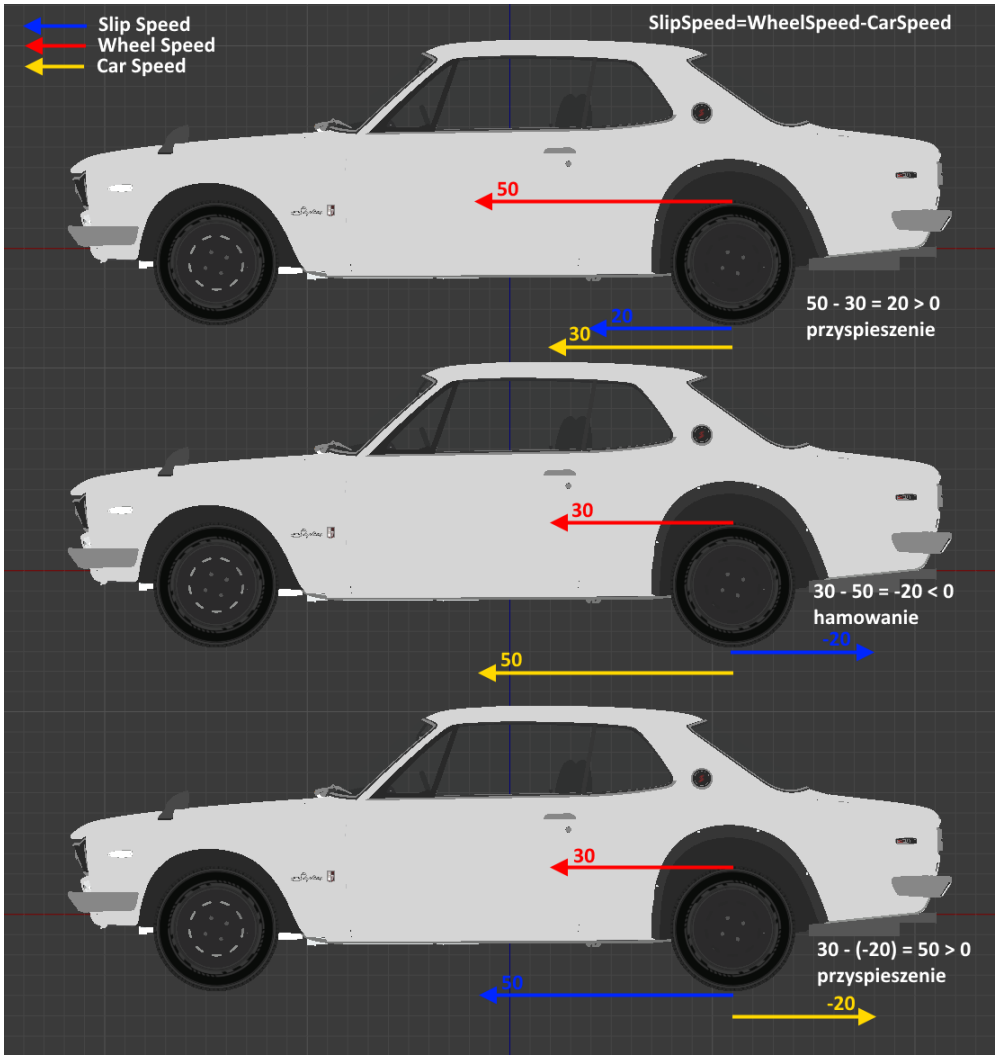
Do obliczenia poślizgu wzdłużnego stosuje się następującą logikę, jeżeli wektory prędkości *SlipSpeed* i *CarSpeed* wskazują ten sam kierunek (*SlipSpeed* * *CarSpeed* > *0*) to przyspieszenie wzdłużne jest obliczane na podstawie momentu obrotowego przyłożonego do koła

$$Traction = \frac{DriveTorque}{Radius}, \qquad (14)$$

w przeciwnym razie *SlipSpeed* działa jako przyspieszenie wzdłużne. Mając zatem poślizg wzdłużny i poprzeczny za pomocą funkcji *Make Vector 2D* otrzymujemy wektor poślizgu kombinowanego.

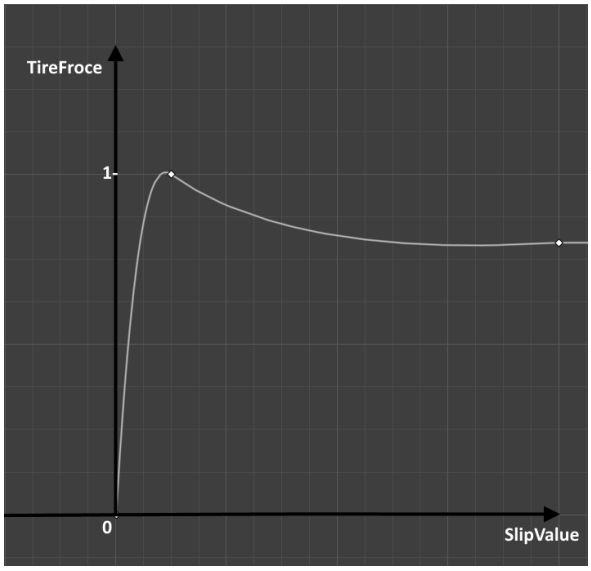
Na podstawie wektora poślizgu kombinowanego możliwe jest określenie mnożnika dla znormalizowanej siły nacisku na oponę za pomocą wykresu siły nacisku na oponę w stosunku do siły poślizgu. Jako podstawa został wykorzystany wykres empirycznego modelu autorstwa holenderskiego naukowca Pacejki, określony w literaturze jako "magiczna formuła" [21].

Na rysunku 26 przedstawione są trzy możliwe typy poślizgu wzdłużnego.



Rysunek 26. Zdjęcie przedstawiające rodzaje poślizgu wzdłużnego. Źródło: opracowanie własne.

Wykres na rysunku 27 przedstawia zależność między siłą tarcia koła a wektorem poślizgu Ponieważ sam model koła Pacejki uwzględnia ponad 20 różnych zmiennych, do celów optymalizacji wykorzystuje się jedynie przybliżoną kopię wykresu.



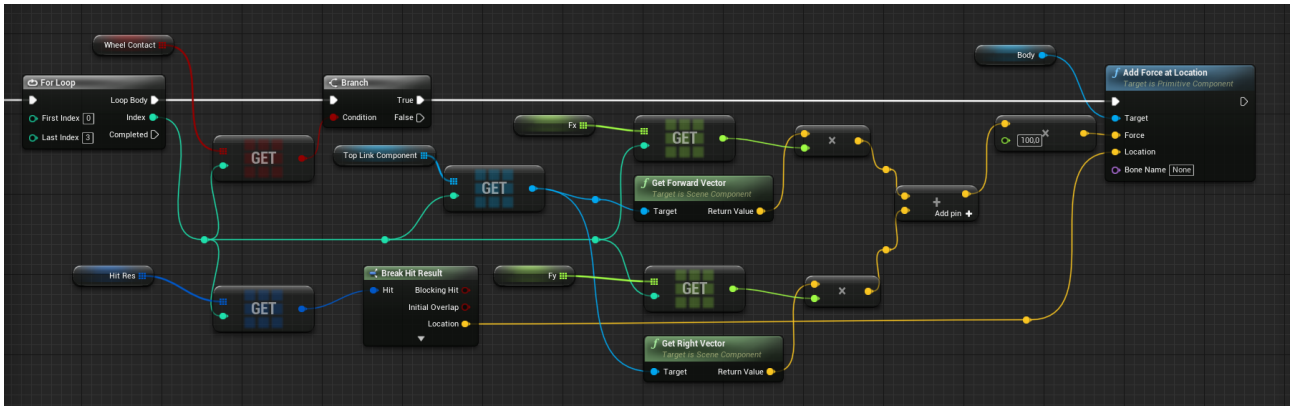
Rysunek 27. Zdjęcie przedstawiające przybliżony wykres oparty na modelu Hans B. Pacejki. Źródło: opracowanie własne.

Ponieważ wartość wektora siły mieści się obecnie w przedziale od 0 do 1, musimy pomnożyć ten wektor przez maksymalną siłę tarcia:

$$TireForce = TireFroceNormalized * MaxFriction. \quad (15)$$

Wektor ten jest następnie dzielony na dwie współrzędne *Fx* i *Fy*.

Teraz, gdy znane są siły tarcia wzdłużnego (*Fx*) i poprzecznego (*Fy*) opony, można je przyłożyć w punkcie styku koła z nawierzchnią drogi za pomocą funkcji *Add Force at Location* (patrz listing 5).

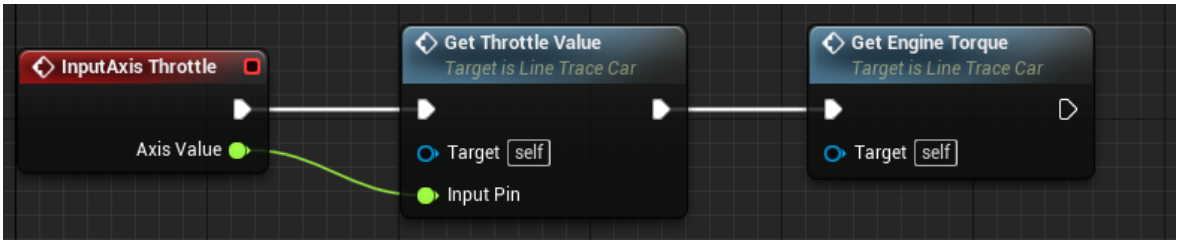


Listing 5. Przełożenie siły tarcia wzdłużnego i poprzecznego w punkcie styku koła z nawierzchnią drogi . Źródło: opracowanie własne.

5.3.3 Silnik

Liczy się, że silnik jest najważniejszym elementem w samochodzie, ale jeśli potraktować samochód jako zbiór różnych systemów współpracujących ze sobą, to od razu widać, że silnik jest tak samo ważny jak każdy inny element w samochodzie. Zgodnie ze swoim przeznaczeniem silnik jest źródłem energii mechanicznej niezbędnej do poruszania się samochodu. Aby uzyskać energię mechaniczną, w silniku pojazdu przekształcana jest inna forma energii (energia spalania lub energia elektryczna). Źródło energii musi znajdować się bezpośrednio na pojeździe i musi być okresowo uzupełniane. Silnik zamienia energię cieplną na moment obrotowy, który za pośrednictwem wału korbowego i skrzyni biegów przekazywany jest na oś napędową z kołami.

Zasadę działania samego podzespołu silnika można podzielić na dwa etapy: odbieranie od użytkownika wartości pedału gazu oraz generowanie momentu obrotowego (patrz listing 6).



Listing 6. Kolejność metod odpowiadających za komponent silnika. Źródło: opracowanie własne.

Podczas uzyskiwania wartości pedału gazu stosowane jest podejście filtrujące, które symuluje płynną reakcję pedału przepustnicy i stopniowo zmienia wartość zmiennej *ThrottleValue* od 0 do 1. Możliwe jest również wyłączenie filtru po zmianie parametru *ThrottleFilter* na *false*.

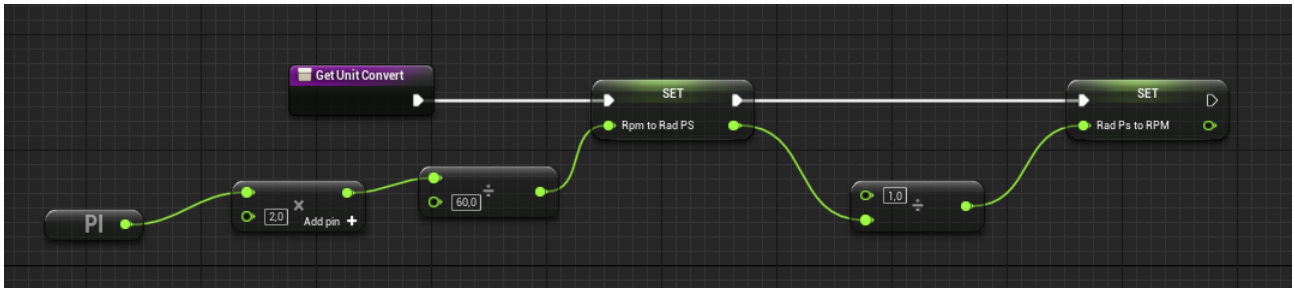
W funkcji generowania momentu obrotowego silnika wykorzystujemy funkcję interpolacji liniowej Lerp do uzyskania wartości momentu obrotowego silnika na podstawie wykresu zależności momentu obrotowego od prędkości obrotowej silnika oraz wartości zmiennych ThrottleValue i EngineRPM. Następnie możemy znaleźć przyspieszenie kątowe, ponieważ silnik generując moment obrotowy sam siebie rozkręca, czyli im wyższa jest wartość ThrottleValue, tym silniejsze są obroty silnika. Przyspieszenie kątowe oblicza się według wzoru:

$$AngularAcceleration = \frac{EngineTorque}{Inertia}, \quad (16)$$

gdzie *Inertia* – moment bezwładności dla samochodowych silników.

Znając przyspieszenie kątowe, znajdziemy prędkość kątową silnika, uprzednio ograniczając ją do minimalnej i maksymalnej możliwej prędkości obrotowej.

W ostatnim kroku, zaktualizujemy obroty silnika (EngineRPM) poprzez konwersję jego prędkości kątowej z radianów na sekundę na obroty na minutę (patrz listing 7).



Listing 7. Funkcja do przeliczania prędkości kątowej z radianów na sekundę na obroty na minutę. Źródło: opracowanie własne.

5.3.4 Skrzynia biegów

Skrzynia biegów jest nieodłączną częścią każdego pojazdu z silnikiem spalinowym. Zadaniem skrzyni biegów jest przenoszenie i przekształcanie momentu obrotowego z silnika na koła

oraz dostarczanie mocy do napędu innych urządzeń i akcesoriów. Proces ten umożliwia optymalne przyspieszenie, prędkość pojazdu, a także jazdę do tyłu. Ponadto pomaga oddzielić wał korbowy silnika od kół napędowych, co pozwala pojazdowi pracować na biegu jałowym lub całkowicie się zatrzymać.

Sposób działania skrzyni biegów opiera się na koncepcji "złotej zasady mechaniki". To znaczy, że kiedy zyskujemy w mocy (moment obrotowy), to tracimy w prędkości i odwrotnie. Im niższy bieg, tym łatwiej samochód przyspiesza, a im wyższy bieg, tym większą prędkość możemy osiągnąć.

Strukturalnie, układ skrzyni biegów można przedstawić w dość prostej formie. Mechanizm skrzyni biegów składa się z zespołu kół pasowych o różnych średnicach, które znajdują się na wale napędowym, koła te połączone są z wałem silnika za pomocą paska. W zależności od warunków jazdy pasek jest przesuwany z jednego koła pasowego na drugie za pomocą specjalnego mechanizmu. Dzięki temu możliwa jest zmiana momentu obrotowego przekazywanego na koła napędowe. Ten prosty mechanizm jest wykorzystywany nie tylko w motoryzacji, na przykład rowery wykorzystują ten sam mechanizm do zmiany biegów.

Element skrzyni biegów jest jednym z najprostszych elementów w fizycznym modelu zachowania pojazdu, gdyż jego realizacja wymaga stworzenia mechanizmu wyboru biegu i odpowiedniego dla niego przełożenia. Utworzona tablica *GearRatio* przechowuje przełożenia dla każdego z pięciu biegów pojazdu, w tym biegu jałowego i wstecznego. Gdy użytkownik wciśnie *lewy Shift*, wartość zmiennej *Gear* przechowującej wartość aktywnego biegu jest zwiększana o jeden i zapisywana do zmiennej tymczasowej *GearTarget*. Następnie aktywny bieg jest ustawiany na neutralny i przytrzymywany przez czas określony w zmiennej *GearChangeTime* za pomocą metody *Delay*.

Należy pamiętać, że przełożenie całkowite składa się z dwóch wartości: przełożenia wybranego biegu oraz przełożenia głównego mechanizmu różnicowego:

$$\text{TotalGearRatio} = GearRatio[Gear] * MainGear, \qquad (17)$$

Dlatego po upływie wymaganego czasu w zmiennej *Gear* bieg jest ustawiany na kolejny i w *TotalGearRatio* zapisywane jest jego przełożenie pomnożone przez przełożenie biegu głównego (*MainGear*).

W taki sposób symuluje się czas, potrzebny skrzyni biegów do zmiany aktywnego biegu. Podobnie działa funkcja do redukcji biegów, gdzie użytkownik musi nacisnąć *lewy Ctrl*, aby zmniejszyć bieg.

5.3.5 Przeniesienie napędu

Przeniesienie napędu w budowie maszyn odpowiada za przeniesienie energii z jej źródła (silnika) na koła napędowe. W pojeździe, niezależnie od jego typu, skrzynia biegów to układ elementów i mechanizmów, które przenoszą ruch obrotowy, postępowy z silnika na koła.

W zależności od tego, które koła otrzymują moment obrotowy, przekładnie można podzielić na trzy rodzaje: FWD (napęd na przednie koła), RWD (napęd na tylne koła) i AWD (napęd na wszystkie koła). Dlatego najpierw należy zainicjować komponent transmisji. Zmieniając wartość zmiennej *DriveType*, użytkownik dostosowuje procentowy udział momentu obrotowego przekazywanego na osie pojazdu w tablicy *TorqueRatio*. W tej tablicy 1 odpowiada za sto procent momentu obrotowego przekazywanego na oś samochodu pod odpowiednim indeksem, 0 za brak momentu obrotowego, a w przypadku napędu na wszystkie koła moment obrotowy jest rozdzielany między osie na podstawie zmiennej *TorqueBias*, gdzie wartość 0,25 oznacza, że przednia oś otrzymuje 25 procent całego momentu obrotowego, a tylna 75 procent (1 - 0,25 = 0,75).

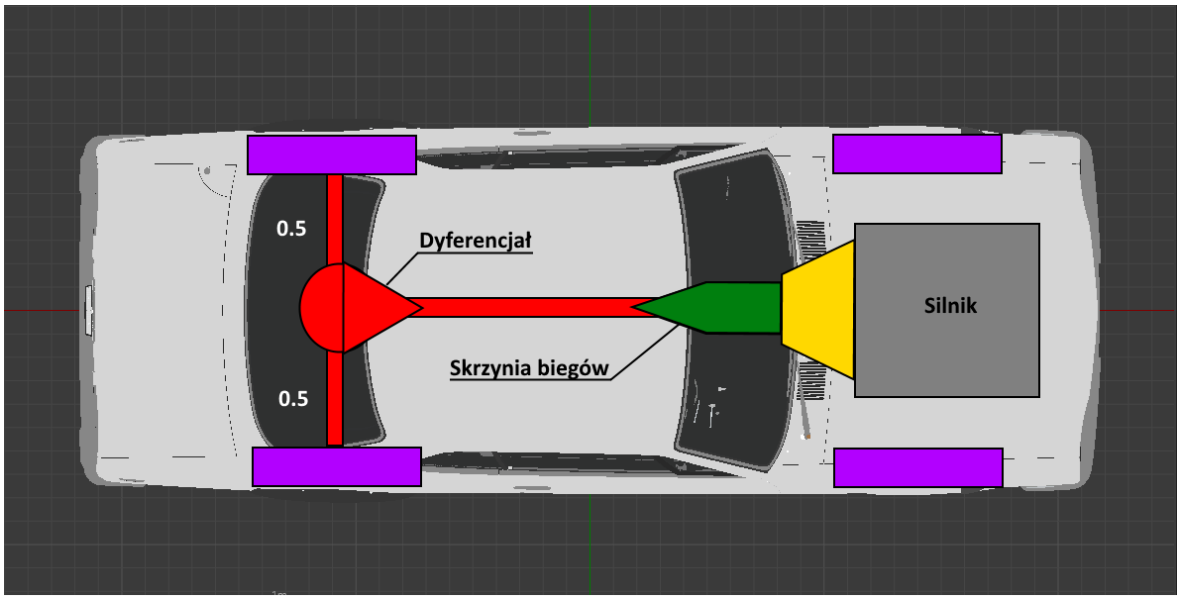
Teraz, gdy tablica *TorqueRatio* została zainicjalizowana, można obliczyć moment obrotowy przyłożony do kół napędowych (*DriveTorque*), określić prędkość kątową kół i rozpocząć obracanie kół. Kolejność postępowania została umieszczona na listingu 8.

Listing 8. Kolejność postępowania dla przekazania momentu obrotowego do kół. Źródło: opracowanie własne.

Aby obliczyć zmienną *DriveTorque* należy pomnożyć moment obrotowy silnika przez całkowite przełożenie uzyskane w komponencie skrzyni biegów (*TotalGearRatio*), otrzymaną wartość mnożymy przez współczynnik podziału momentu obrotowego pobrany z tablicy *TorqueRatio* dla odpowiedniego indeksu koła. Bardzo ważne jest pomnożenie końcowego wyniku przez 0,5, ponieważ w tym momencie moment obrotowy jest całkowity dla całej osi. To właśnie mechanizm różnicowy dzieli moment obrotowy pomiędzy lewym i prawym kołem często w oparciu

o kąt ustawienia kół przednich. Jednak implementacja takiego mechanizmu byłaby bardzo zasobożerna, dlatego moment obrotowy jest dzielony równo między lewym i prawym kołem. Na rysunku 28 przedstawiony jest schemat rozdzielenia momentu obrotowego pomiędzy lewym a prawym kołem.

$$DriveTorque = EngineTorque * TotalGearRatio * TorqueRatio * 0.5, \quad (18)$$



Rysunek 28. Zdjęcie przedstawiające schemat rozdzielenia momentu obrotowego pomiędzy lewym a prawym kołem. Źródło: opracowanie własne.

Przy obliczaniu prędkości kątowej kół, zasada jest taka sama jak przy uzyskiwaniu prędkości kątowej silnika. Najpierw znajdziemy przyspieszenie kątowe, dzieląc wynikowy moment obrotowy koła przez jego bezwładność:

$$AngularAcceleration = \frac{DriveTorque}{WheelInertia}, \quad (19)$$

Znając przyspieszenie kątowe, można znaleźć prędkość kątową, dodając do już istniejącej prędkości kątowej przyspieszenie kątowe pomnożone przez czas między dwiema klatkami:

$$AngularVelocity \mathrel{+}= AngularAcceleration * deltaTime, \quad (20)$$

Z kolei na podstawie prędkości kątowej możemy obracać koło samochodu za pomocą funkcji *AddLocalRotation*, najpierw konwertując wartość zmiennej *AngularVelocity* z radianów na stopnie, gdyż jednostką miary kątów w Unreal Engine są stopnie.

5.3.6 Hamulce

Hamulce są niezbędnym elementem każdego pojazdu, służą do zwalniania, zatrzymywania i zapobiegania ruchowi po zatrzymaniu. Bezpieczeństwo jazdy zależy od dostępności i jakości układu

hamulcowego. Zasada działania każdego układu hamulcowego jest prosta. Pedał hamulca kierowcy przenosi siłę poprzez szereg urządzeń na koła zębate, które z kolei oddziałują na tarcze hamulcowe, dociskając do nich klocki i w ten sposób zatrzymując ich obrót, a w konsekwencji cały pojazd.

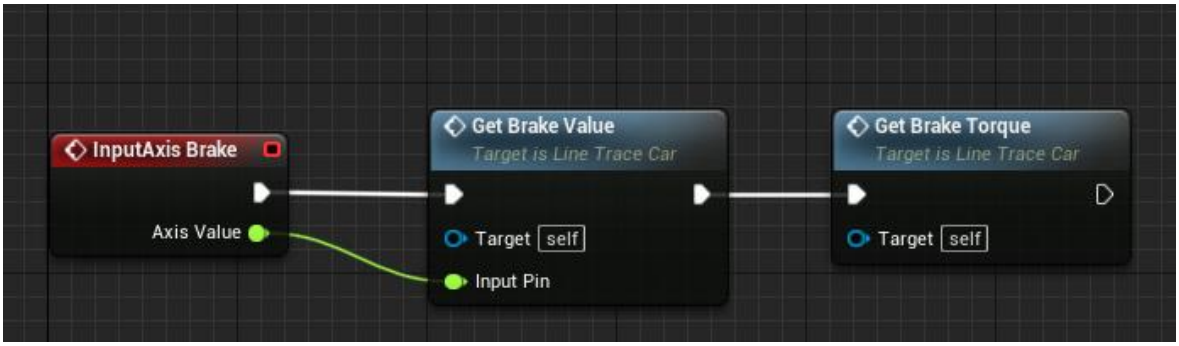
Pierwszą czynnością jest inicjalizacja tablicy *BrakeRatio* na podstawie wartości zmiennej *BrakeBias*, która odpowiada za rozkład siły hamowania pomiędzy osiami. Wartość *BrakeBias* wskazuje procent siły hamowania działającej na przednią oś (np. 0,6). Odpowiednio do tylnej osi stosuje się 1 - 0,6 = 0,4, czyli 40 procent.

Należy również zaimplementować mechanizm, który pozwoli uzyskać informację o wciśniętym przez użytkownika przycisku hamowania. Funkcja ta jest realizowana w taki sam sposób jak otrzymywanie informacji o wciśniętym przycisku pedału gazu w komponencie silnika. Implementacja polega na dodaniu filtra, który płynnie zmienia wartość zmiennej *BrakeValue* z 0 na 1. Moment hamowania może być następnie obliczony na podstawie informacji o wciśniętym przycisku hamulca i rozkładzie siły hamowania na osiach.

$$BrakeTorque = BrakeRatio[WheelIndex] * BrakeStrength * BrakeValue, \quad (21)$$

gdzie *BrakeStrength* jest odpowiedzialny za siłę hamowania.

Podobnie obliczany jest moment hamowania po wciśnięciu przycisku hamulca ręcznego.



Listing 9. Kolejność postępowania przy naciśnięciu przycisku hamowania. Źródło: opracowanie własne.

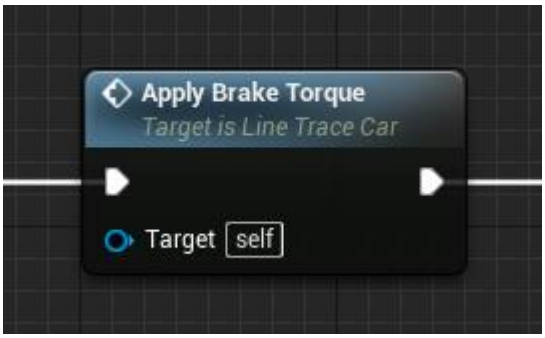
Teraz pozostaje obliczyć prędkość kątową koła podczas hamowania i dodać ją do obecnej wartości prędkości kątowej (patrz listing 10). Niezwykle ważne jest, że moment hamujący musi być przeciwny do znaku prędkości kątowej koła, w przeciwnym razie prędkość kątowa koła będzie rosła, a nie malała. Dlatego wartość *BrakeTorque* mnożymy przez przeciwny znak zmiennej *WheelAngularVelocity*. Następnie obliczamy przyspieszenie kątowe korzystając ze wzoru:

$$WheelAngularAcceleration = \frac{BrakeTorque}{WheelInertia} \quad (22)$$

Na podstawie przyspieszenia kątowego obliczamy prędkość kątową i dodajemy tę wartość do aktualnej wartości prędkości kątowej koła:

$$WheelAngularVelocity \mathrel{+}= WheelAngularAcceleration * deltaTime \qquad (23)$$

Ważne jest również sprawdzenie znaku uzyskanej prędkości kątowej oraz znaku wektora prędkości w poprzedniej klatce. Tylko wtedy, gdy znak odpowiada znakowi nowej prędkości kątowej, wartość prędkości kątowej koła zostanie zaktualizowana. Mechanizm ten pozwala na pozbycie się wirowania kół podczas stania samochodu w miejscu.



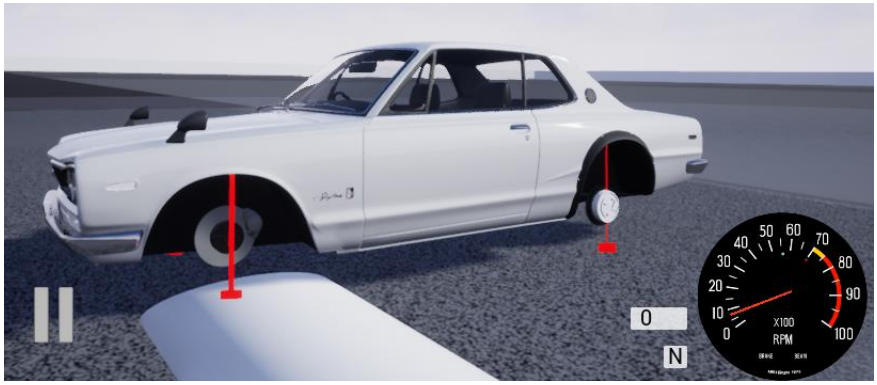
Listing 10. Funkcja odpowiadająca za przyłożenie momentu obrotowego hamowania do kół samochodu. Źródło: opracowanie własne.

6. Testowanie

W tej części zostaną przedstawione testy stworzonego modelu fizycznego w celu weryfikacji jego poprawnego działania. Poniższe testy potwierdzą również, że powstały model fizyczny spełnia wymagania funkcjonalne i niefunkcjonalne.

Algorytm zawieszania (LineTrace):

Głównym zadaniem elementu zawieszenia jest amortyzacja niewielkich nierówności na drodze przy jednoczesnym utrzymaniu kontaktu opon z nawierzchnią. Aby to zrobić, długość wektora LineTrace musi być stale aktualizowana. Jak widać na rysunku 29, wektor LineTrace jest przerywany dokładnie w miejscu, w którym styka się z powierzchnią. Również w zależności od ukształtowania terenu długość wektora maleje lub rośnie, co w pełni odzwierciedla działanie sprężyn w rzeczywistym pojeździe.



Rysunek 29. Zdjęcie przedstawiające działanie mechanizmu LineTrace. Źródło: opracowanie własne.

Rysunek 30 z kolei ilustruje, jak na podstawie długości wektora LineTrace zmienia się położenie koła pojazdu:

Rysunek 30. Zdjęcie przedstawiające zmianę położenia koła w zależności od napotkanej przeszkody. Źródło: opracowanie własne.

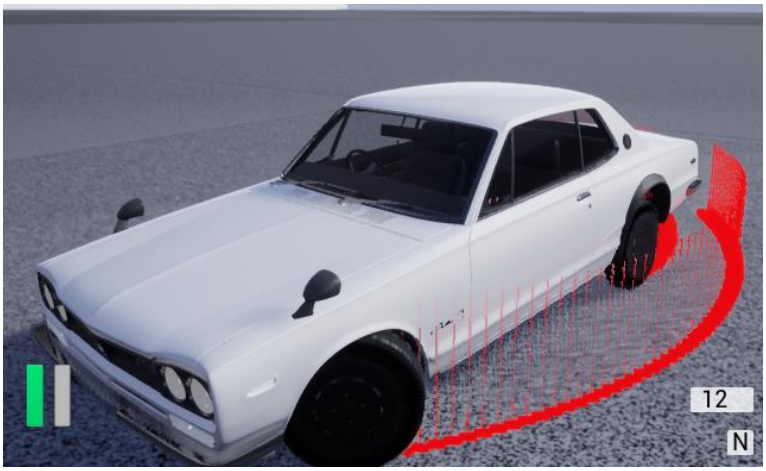
Zastosowanie algorytmu opartego na LineTrace wynika z dobrej symulacji zachowania kół pojazdu na nierównościach, przeniesienia obciążenia pomiędzy wszystkimi kołami, co daje widoczne przechyły nadwozia pojazdu podczas pokonywania zakrętów oraz stosunkowo słabego obciążenia obliczeniowego. Dzięki temu element zawieszenia jest w pełni zgodny z [WF1].

Sterowanie kamerą i obrót kół:

Możliwość zmiany pozycji kamery jest niezwykle przydatną funkcją podczas testowania modelu fizycznego, ponieważ pozwala na bardziej szczegółowy podgląd poszczególnych elementów pojazdu. Z kolei skręt kół jest istotnym elementem zachowania pojazdu, gdyż to właśnie skręt kół pozwala na kierowanie pojazdem. Zmieniając wartość osi *Right* za pomocą klawiszy A i D, użytkownik reguluje stopień skrętu kół. Na rysunku 31 przedstawiono maksymalną sterowność stojącego pojazdu.

Rysunek 31. Zdjęcie przedstawiające maksymalny kąt skrętu kół samochodu. Źródło: opracowanie własne.

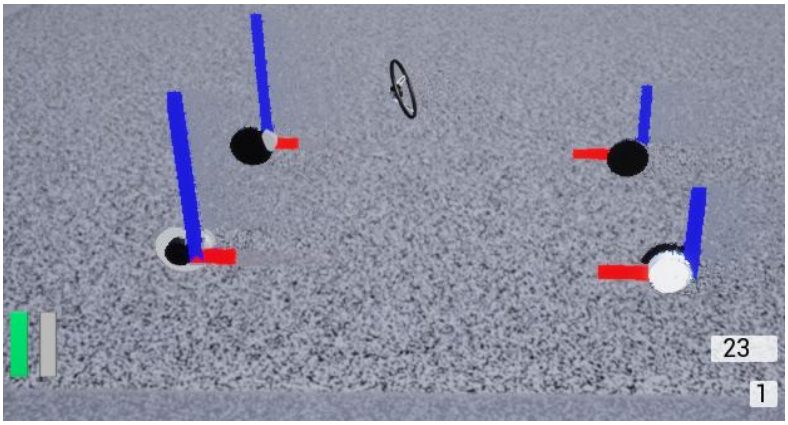
Rysunek 32 demonstruje sterowanie pojazdem przy skręconych kołach, potwierdzając wymaganie funkcjonalne [WF2].



Rysunek 32. Zdjęcie przedstawiające skręcanie samochodu z prędkościa. Źródło: opracowanie własne.

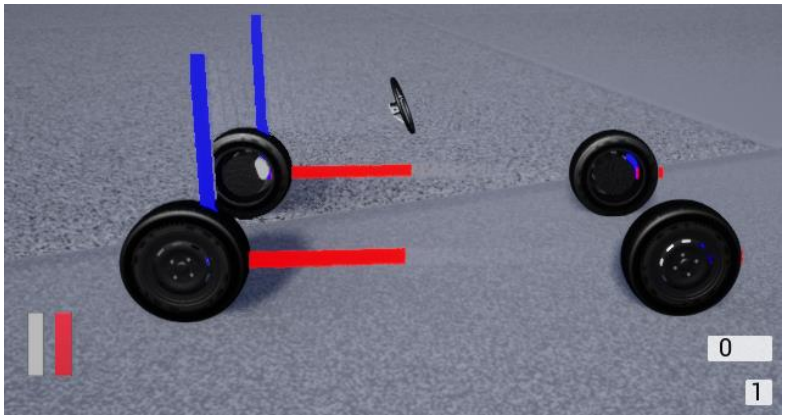
Mechanizm tarcia kół:

Mechanizm tarcia kół jest kluczowym elementem w realizacji fizycznego zachowania samochodu. To właśnie dzięki tarciu samochód przyspiesza, skręca i hamuje. Na rysunku 33 przedstawiono na czerwono siłę wzdłużną działającą na każde z kół samochodu podczas przyspieszania. Ponieważ przedstawiony samochód posiada napęd na tylne koła, siła wzdłużna kół tylnych jest współosiowa z kierunkiem jazdy samochodu. Natomiast siła wzdłużna kół przednich jest skierowana w kierunku przeciwnym do ruchu samochodu, ponieważ nie jest na nie przenoszony moment obrotowy silnika i odczuwają one jedynie siłę tarcia. Koła samochodu zostały specjalnie ukryte, aby lepiej zaprezentować działające na nie siły.



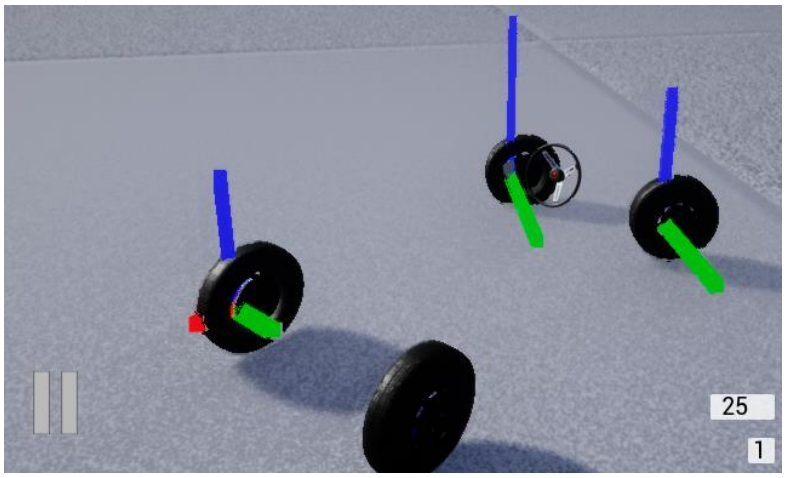
Rysunek 33. Zdjęcie przedstawiające siły działające przy przyspieszaniu samochodu. Źródło: opracowanie własne.

Na rysunku 34 przedstawiono sytuację hamowania samochodu. Ponieważ bilans sił hamowania jest przesunięty na oś przednią w stosunku 60:40, siła wzdłużna działająca na koła przednie jest znacznie większa niż siła na koła tylne. Występuje również silne przesunięcie siły sprężyn działających na nadwozie samochodu. Siła pionowa działająca na nadwozie pojazdu przedstawiona jest w kolorze niebieskim. Takie przemieszczenie siły sprężyny jest naturalne dla prawdziwych samochodów podczas ostrego hamowania.



Rysunek 34. Zdjęcie przedstawiające siły działające przy hamowaniu samochodu. Źródło: opracowanie własne.

Rysunek 35 z kolei przedstawia działanie siły poprzecznej na koła podczas skręcania samochodu.



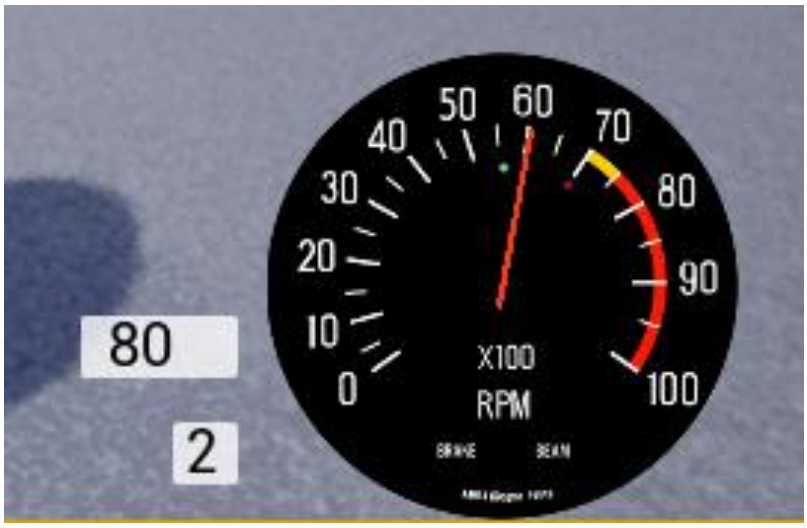
Rysunek 35. Zdjęcie przedstawiające siłe działające przy skręcaniu samochodu. Źródło: opracowanie własne.

Na przedstawionych wyżej rysunkach dla wizualizacji sił działających na koła został wykorzystany tryb debugowania, który ukrywa nadwozie samochodu i rysuje wartości zmiennych *Fx*, *Fy* i *Fz* jako osi współrzędnych w wymiarze 3D.

Po przeprowadzonych testach można z pewnością powiedzieć że wymagania funkcjonalne [WF3] i [WF4] zostały spełnione.

Zrealizowane komponenty samochodu:

Komponent silnika generuje moment obrotowy na podstawie wykresu zależności momentu obrotowego od obrotów silnika. Na rysunku 36 przedstawiony jest element graficznego interfejsu symulujący analogowy tachometr w samochodzie.

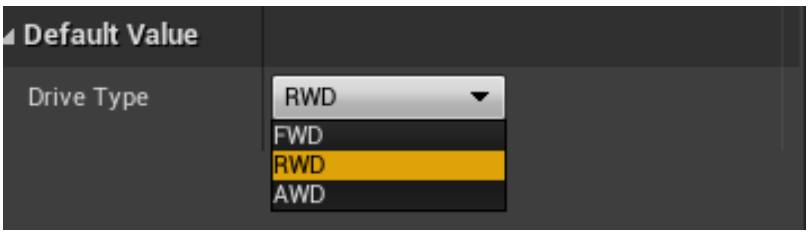


Rysunek 36. Zdjęcie przedstawiające elementy interfejsu do wyświetlenia obrotów silnika, prędkości i aktualnego biegu samochodu. Źródło: opracowanie własne.

Na rysunku 36 również jest przedstawiony interfejs komponentu skrzyni biegów z aktualnym biegiem i prędkością samochodu. Zmieniając bieg samochodu dostosowujemy moment obrotowy dostarczany do osi napędowej. Tak, na przykład, na trzecim biegu samochód rozpędza się znacznie wolniej niż na pierwszym.

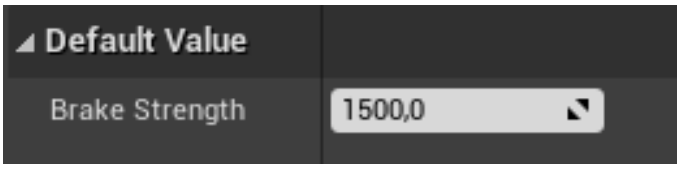
Komponent transmisji odpowiada za bezpośrednie dostarczenie momentu obrotowego silnika pomnożonego przez przełożenie wybranego biegu oraz przełożenie głównego mechanizmu różnicowego. Należy pamiętać że jedno koło dostaje tylko połowę momentu obrotowego przeznaczonego na całą oś.

Rysunek 37 przedstawia możliwość wyboru typu napędu dla samochodu i w zależności od wybranego typu (RWD, FWD, AWD) moment obrotowy zostanie przekierowany na odpowiednią oś.



Rysunek 37. Zdjęcie przedstawiające wybór typu napędu samochodu. Źródło: opracowanie własne.

Hamulce służą do zatrzymania lub zmniejszenia prędkości jadącego samochodu. Jak było pokazane na rysuku 34, hamulce posiadają możliwość przesunięcia momentu hamowania pomiędzy osiami samochodu. Również istnieje możliwość zmiany siły hamowania za pomocą zmiennej *BrakeStrength* (patrz rysunek 38).



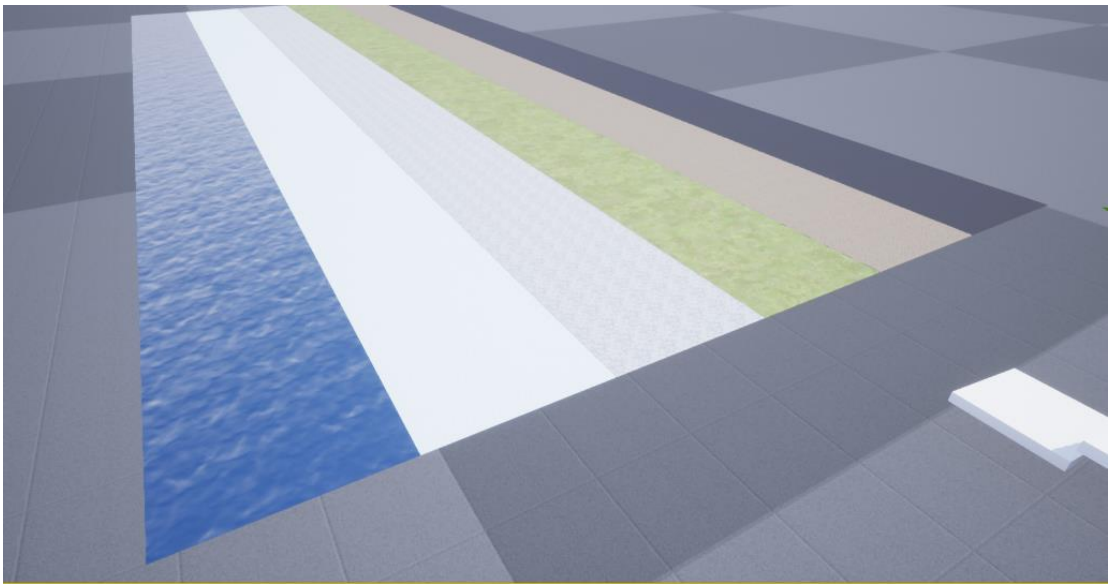
Rysunek 38. Zdjęcie przedstawiające domyślną wartośc siły hamowania. Źródło: opracowanie własne.

Z przedstawionych testów i przykładów wynika że wymienione wyżej komponenty w pełni spełniają wymagania funkcjonalne [WF5-8].

Typy powierzchni:

Oprócz modelu fizycznego opracowano kilka typów powierzchni, które posiadają własne współczynniki tarcia oraz stopień nierówności. Rodzaj nawierzchni ma bezpośredni wpływ na trakcję opon, a także, w zależności od szorstkości, utrudnia rozpędzanie samochodu. Na rysunku 39

przedstawiono obszary testowe o różnych powierzchniach, a mianowicie wodę, lód, śnieg, trawę, piasek i asfalt.



Rysunek 39. Zdjęcie przedstawiające typy powierzchni. Źródło: opracowanie własne.

Na rysunku 40 przedstawiono sposób symulacji nierówniści powierzchni:

Rysunek 40. Zdjęcie przedstawiające poruszanie się po luźnej powierzchni. Źródło: opracowanie własne.

Wydajność:

Dodatkowo został przeprowadzony test wydajności całego modelu. Na stanowisku testowym w oparciu o ten model fizyki stworzono 15 identycznych pojazdów. Zmierzono liczbę klatek produkowanych przez silnik Unreal Engine podczas symulacji jednego samochodu i 30 samochodów na jednej scenie (patrz rysunki 41 i 42).

Rysunek 41. Zdjęcie przedstawiające ilośc klatek na sekundę przy symulacji jednego samochodu. Źródło: opracowanie własne.

Na rysunku 42 przedstawiony jest pomiar wydajności fizycznego modelu z użyciem 15 samochodów:

Rysunek 42. Zdjęcie przedstawiające ilośc klatek na sekundę przy symulacji 30 samochodów. Źródło: opracowanie własne.

W efekcie spadek liczby klatek na sekundę 8 FPS (Frames Per Second) wynosi tylko 13%, co jest dość dobrym wskaźnikiem wydajności. Można więc stwierdzić, że wymagania dotyczące wydajności [WNF1] i [WNF2] zostały spełnione przy tworzeniu danego modelu fizycznego. Dany test był przeprowadzony na laptopie firmy Acer z kartą graficzną marki NVIDIA GTX 1050Ti i processorem Intel(R) Core(TM) i7-8750H.

W wyniku końcowym, można powiedzieć, że zbudowany model fizyczny w pełni realizuje główne komponenty rzeczywistych samochodów, wykonuje obliczenia dla każdego z kół samochodu

osobnie, uwzględnia zależność przyczepności opon od typu powierzchni, zapewnia śledzenie zmiennych w czasie rzeczywistym i równocześnie jest dość wydajnym przy symulacji wielu samochodów opartych na danym modelu fizycznym. Rysunek 43 przedstawia zrzut ekranu końcowego projektu:

Rysunek 43. Zdjęcie przedstawiające zrzut ekranu końcowego projektu. Źródło: opracowanie własne.

7. Podsumowanie

Zaprojektowanie i implementacja modelu opisującego zjawisko fizyczne nie jest łatwym zadaniem. Konieczne jest jasne zrozumienie, jakie czynniki wpływają na dany proces fizyczny. Spośród nich należy wskazać te, które są niezbędne oraz te, które można i należy pominąć. Uproszczenie lub całkowite zaniedbanie pewnych procesów jest po prostu konieczne, uwzględnienie wszystkiego po prostu jest niemożliwe i bezsensowne. Na przykład modelując upadek kamienia na ziemię z wysokości jednego metra, możemy oczywiście pominąć zmianę przyspieszenia siły ciężkości z wysokością, a także przyciąganie księżyca i inne czynniki. Przy modelowaniu przypływów nie można jednak pominąć grawitacyjnego przyciągania Księżyca, gdyż jest ono głównym czynnikiem.

Podczas modelowania fizyki w grach należy zadbać o to, aby model fizyki nie zużywał zbyt wielu zasobów komputera. Nawet rzeczy, które na pierwszy rzut oka są niepożądane, będą musiały zostać uproszczone, a niektóre czynniki mogą nawet zostać zastąpione przez równoważne. Podstawowym podejściem w tworzeniu modelu fizycznego nie jest odtworzenie rzeczywistości, ale stworzenie czegoś podobnego. Również fizyka w grze powinna podkreślać pewne cechy. Jeśli robimy zabawną grę zręcznościową, fizyka powinna być odpowiednia. Można, a czasem nawet należy, wymyślać własne czynniki, które w życiu nie istnieją. Jeśli dążymy do realizmu w grach, to trzeba oszukiwać użytkownika wymyślając uproszczoną fizykę zbliżoną do prawdziwej. Rzeczywistość jest niepowtarzalna, a my jako twórcy naszego modelu fizycznego musimy jedynie uchwycić podstawowe cechy rzeczywistości i wcielić ich podobieństwo.

Głównym celem pracy było stworzenie zaawansowanego mechanizmu symulacji zachowania pojazdu, który jest wydajny i łatwo modyfikowalny do wykorzystania w innych projektach. Model fizyczny musi odnosić się do wszystkich głównych elementów pojazdu, implementując je w sposób optymalny i efektywny.

Cel pracy został w pełni zrealizowany. Model fizyczny jest w pełni zgodny ze wszystkimi wymaganiami, również opracowano system efektów wizualnych dla jazdy po piasku i trawie, system dźwięków opon, silnika oraz jazdy po różnych rodzajach nawierzchni. Stworzony interfejs pozwala na monitorowanie zmian w parametrach pojazdu, bezpośrednio podczas samego testowania. Testy potwierdzają pełną funkcjonalność stworzonego modelu fizycznego oraz pokazują dość dobrą wydajność samej symulacji.

Mówiąc o możliwej rozbudowie, można wyróżnić dwa główne kierunki. Pierwszy z nich opiera się na zbudowaniu pełnoprawnej gry opartej na danym fizycznym modelu, włączając w to tworzenie i symulację lokacji w grze, dodanie różnych modeli samochodów, opracowanie systemu personalizacji, ulepszania i dostosowywania samochodów przez użytkownika i wiele więcej. Drugie podejście polega na rozbudowie istniejącej symulacji, na przykład, dodaniu zależności trakcji kół od ciśnienia w oponach, układu zużycia paliwa i zużycia opon, a nawet zależności mocy silnika od

ilości dopływającego do niego powietrza. Dodanie różnych czynników zewnętrznych, znacznie zwiększy dokładność symulacji i rozszerzy sam model fizyczny. Zatem zakres rozbudowy tego projektu jest bardzo duży i w większości przypadków zależy od potrzeby i czasu przeznaczonego na rozwój.

Literatura

1. Cookson Aram, Dowling Soka Ryan - *Unreal Engine 4 Game Development in 24 Hours*, 496 stron
2. Joanna Lee - *Learning Unreal Engine Game Development*, 274 stron
3. Brenden Sewell - *Blueprints Visual Scripting for Unreal Engine*, 188 stron
4. Pradeep Mamgain - *Autodesk 3ds Max 2020: A Detailed Guide to Modeling, Texturing, Lighting and Rendering, 2nd Edition*, 648 stron
5. Ascent - Center of Technical Knowledge - *Autodesk 3ds Max 2020 Fundamentals*, 688 stron
6. Robert Nystrom - *Game Programming Patterns*, 354 stron
7. Fisher Gordon - *Blender 3D Basics*, 468 stron
8. Andrew Gahan - *3ds Max Modeling for Games*, 480 stron
9. Dokumentacja narzędzia *Unreal Engine 4*, https://docs.unrealengine.com/4.27/en-US/ (data dostępu 28.12.2022)
10. Dokumentacja narzędzia *3Ds Max 2020*, https://help.autodesk.com/view/3DSMAX/2020/ENU/ (data dostępu 28.12.2022)
11. Strona z teorytyczną częścią modelowania fizyki samochodów, https://asawicki.info/Mirror/Car%20Physics%20for%20Games/Car%20Physics%20for%20Games.html (data dostępu 28.12.2022)
12. Strona z teorytycznymi materiałami do modelowania fizyki samochodów, https://nccastaff.bournemouth.ac.uk/jmacey/MastersProject/MSc12/Srisuchat/Thesis.pdf (data dostępu 28.12.2022)
13. Strona z teoretycznymi materialami o dynamice pojazdów wyścigowych, https://www.racecar-engineering.com/tech-explained/racecar-vehicle-dynamics-explained/ (data dostępu 28.12.2022)
14. Strona z opisem nowych funkconalności Unreal Engine 5, https://logicsimplified.com/newgames/unreal-5-nanite-and-lumen-technology-for-next-gen-games/ (data dostępu 28.12.2022)
15. Dokumentacja narzędzia Unreal Engine 5, https://docs.unrealengine.com/5.0/en-US/ (data dostępu 28.12.2022)
16. Dokumentacja narzędzia Unity, https://docs.unity3d.com/Manual/index.html (data dostępu 19.01.2022)
17. Dokumentacja narzędzia Blender, https://docs.blender.org/manual/en/latest/index.html (data dostępu 19.01.2022)
18. Seria poradników do tworzenia gier w Unreal Engine 4, https://www.kodeco.com/663-unreal-engine-4-blueprints-tutorial (data dostępu 19.01.2022)
19. Wprowadzenie do inżynierii gier, https://www.studytonight.com/3d-game-engineering-with-unity/introduction (data dostępu 19.01.2022)

20. Podstawy modelowania 3D, https://www.instructables.com/Intro-to-3D-Modeling/ (data dostępu 19.01.2022)
21. Dynamika opon i pojazdów, https://ftp.idu.ac.id/wp-content/uploads/ebook/tdg/TERRAMECHANICS%20AND%20MOBILITY/epdf.pub_tyre-and-vehicle-dynamics-second-edition.pdf (data dostępu 19.01.2022)

Spis rysunków

Spis tabel

Spis listingów