VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SIMULÁTOR ŘÍZENÍ VOZIDLA
CAR DRIVING SIMULATOR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. David Michalík**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Miroslav Jirgl, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. David Michalík ***ID:*** 173706
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Simulátor řízení vozidla

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je návrh koncepce simulátoru řízení vozidla pro možnost měření a vyhodnocení parametrů lidského operátora a jeho následná implementace využitím vhodného herního enginu.

1. Seznamte se s problematikou hodnocení parametrů lidského operátora.
2. Prostudujte problematiku realizace simulátorů řízení vozidla využitím herních enginů.
3. Navrhněte a popište koncepci simulátoru řízení vozidla pro účely měření a vyhodnocení parametrů lidského operátora.
4. Vyberte vhodný engine a realizujte simulátor.
5. Navrhněte a implementujte vhodné scénáře pro možnost měření parametrů lidského operátora a následně otestujte.
6. Sepište diplomovou práci.

DOPORUČENÁ LITERATURA:

HAVLÍKOVÁ, Marie. Diagnostika systémů s lidským operátorem. Brno, 2008. Disertační práce. Vysoké učení technické v Brně.

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 13.5.2019

Vedoucí práce: Ing. Miroslav Jirgl, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Tato diplomová práce se věnuje tvorbě vlastního simulátoru řízení vozidla a základnímu sběru dat o uživateli pomocí této aplikace. Součástí práce je teoretický úvod do systémů s lidským operátorem, popis funkce a rozhraní herních enginů, rešerše open source herních enginů, popis pracování s Unreal Engine a následné praktické ukázky vytvořeného simulátoru řízení se stručnou analýzou získaných dat o uživateli.

KLÍČOVÁ SLOVA

herní engine, Unreal Engine, simulátor, řízení vozidla, systémy člověk-stroj, sběr dat

ABSTRACT

This master's thesis is mainly focused on creating our own car driving simulator and basic data gathering from the driver's input. To achieve this goal, the thesis includes introduction to man-machine systems and basic information about functions and runtime game engine employs. Research about commonly used open source game engines is also presented with a detailed focus on the engine we chose - Unreal Engine. In conclusion of this thesis, a full version of a car driving simulator is created with gathered data analysis.

KEYWORDS

game engine, Unreal Engine, simulator, car driving, man-machine systems, data acquisition

MICHALÍK, David *Simulátor řízení vozidla*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2019. 81 s. Vedoucí práce byl Ing. Miroslav Jirgl, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Simulátor řízení vozidla“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu své diplomové práce, panu Ing. Miroslavovi Jirglovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Cílem diplomové práce je vytvořit základní verzi simulátoru řízení vozidla, která poskytne potřebné prostředí pro různé scénáře a zisk dat o uživateli - řidiči simulovaného vozidla. Mezi sledované veličiny patří například rychlost reakce testovaného subjektu, míra natočení volantu, sešlápnutí pedálů, dodržení jízdy v pruhu apod. K určení sledovaných veličin, které bude simulátor o operátorovi získávat, je určen teoretický úvod do systémů člověk-stroj v kapitole 1.1. Následně je potřeba se seznámit se strukturou herních enginů a principech jejich fungování. Tomu se věnuje kapitola 1.2 v teoretické části diplomové práce. V ní jsou popsány základy herních enginů, jejich využití a také rozložení do jednotlivých vrstev, které na sebe navazují a pomáhají tak vytvořit aplikační rozhraní pro tvorbu herních aplikací.

Posléze je v kapitole 1.3 provedena rešerše volně dostupných herních enginů pro běžného zákazníka. Není tomu tak dávno, kdy nejvyužívanější enginy byly dostupné pouze pro korporátní sféru za nemalé peníze. V dnešní době však roste podpora tzv. indie vývojářů a s tím se rozšiřují i komunity malých vývojářů. Mezi aktuálně nejvyužívanějších herní enginy na trhu patří CryENGINE, Unity a Unreal Engine. Z této rešerše je následně vybrán nejvhodnější herní engine na realizaci simulátoru a ten je popsán v následující kapitole 1.4 a 1.5.

Praktická část diplomové práce je pak věnována samotné realizaci herního simulátoru na bázi herního enginu Unreal Engine. Cílem je vytvořit fungující prostředí s implementovaným vozidlem a jeho ovládáním, scénáři a systému na sběr dat. V kapitole 2.3 je pak otestován výsledný produkt.

1 TEORETICKÁ ČÁST DIPLOMOVÉ PRÁCE

1.1 Systémy s lidským operátorem

Hlavním cílem této diplomové práce je vytvořit simulátor řízení vozidla, jehož parametry a požadavky vyplynou z teorie jak o vývoji herních aplikací, tak z teorie systémů s lidským operátorem. Simulátor tedy bude poskytovat interaktivní prostředí určené pro sběr dat o uživateli - v tomto případě řidiči, který bude na simulátoru v místnosti (s prvky z reálného vozidla) testován. Je třeba se zaměřit na problematiku systémů s lidským operátorem.

1.1.1 Systémy MMS

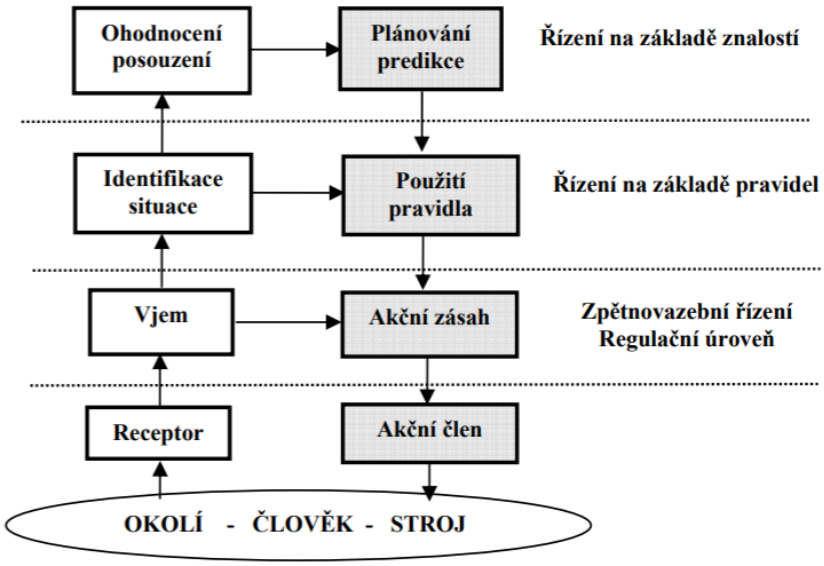
Z teorie řízení je známo, že chceme-li úspěšně a efektivně řídit určitý proces, je potřeba správně navrženého regulátoru. Takovým regulátorem je také člověk a úkony řízení provádí každý den. Neustále řešíme neočekávané situace s různými měnícími se podmínkami a jsme nuceni se jim přizpůsobovat. Z tohoto pohledu je člověk velmi účinným a univerzálním regulátorem. Regulační zásahy v systémech s člověkem se mění na základě zkušeností a kvalita těchto zásahů často závisí na schopnostech lidského operátora. Je tedy vhodné říci, že člověk zastupuje formu učícího se adaptivního regulátoru a jeho regulační vlastnosti jsou analogické s průmyslovými regulátory, jejichž chování lze matematicky vyjádřit a vyhodnotit. Nesmí se však opomíjet důležitá složka, která v průmyslových regulátorech není přítomna - a to je lidské myšlení. Myšlenkové pochody člověka nelze v regulačním procesu odstranit nebo vyloučit. [1]

Stroje či nástroje, které lidé běžně využívají, jsou zkonstruovány tak, aby s nimi člověk manipuloval, ovládal je nebo řídil. Člověk tak vytváří se strojem určitý systém, ve kterém dochází k neustálé spolupráci a vzájemnému působení. V odborné literatuře se tyto systémy označují jak systémy MMS - Man-Machine System, neboli systém člověk-stroj[2]. Příkladem těchto systému mohou být jednoduchá využití ručního nářadí, nebo složitější příklady řízení automobilu či letadla. Pro tuto práci je důležité, že člověk, který řídí auto, vytváří společně s tímto strojem MMS systém, který lze popsat a vyvozovat z něj určité závěry. Tím, že se jedná o komplikovaný systém, který závisí jak na fungování stroje, tak na vstupech od uživatele, není možné předpokládat, že MMS bude vždy pracovat spolehlivě a bez poruchy. Ať už z hlediska lidského ovládání nebo stroje samotného, je potřeba neustále zajišťovat a kontrolovat bezpečnost těchto systémů. Může tedy docházet k závažným poruchám,

které by mohly ohrozit zdraví nebo způsobit rozsáhlé škody. Existencí různých simulací a zisku dat z nich můžeme těmto poruchám předcházet. [1]

1.1.2 Úrovně činností člověka v systémech MMS

Jak již bylo zmíněno v předchozí kapitole, systém člověk-stroj funguje na vzájemné spolupráci těchto dvou prvků a člověk v těchto systémech vykonává různé pracovní a řídící operace. Znalost těchto operací je potřeba pro vytvoření věrného modelu celého systému MMS. Pokud je takovýto model vytvořen, je možné sledovat kritická místa, která jsou nejčastějšími příčinami nebezpečných poruch systému. Pro vytvoření takového modelu je potřeba dokonalá znalost činností člověka a způsob jejich provádění, stejně tak jako znalost komunikace mezi operátorem a strojem. Činnosti operátora jsou závislé na složitosti konkrétního systému a lze je třídit do kategorií podle různých hledisek - časová náročnost, výkonnostní náročnost či složitost. Dle profesora J. Rasmussena jsou tyto činnosti rozděleny dle náročnosti a podle toho, jaké části organismu člověka jsou potřeba k jejich realizaci. Jedná se o tyto úrovně: [2]



Obr. 1.1: Vrstvy řízení systémů MMS [3]

Regulační úroveň

Člověk zastupuje roli regulátoru a jeho činností jsou regulační zásahy, pomoci kterých řídí stroj a tedy i celou soustavu MMS. Akčním a výkonným prvkem lidského regulátoru je jeho pohybový aparát - tedy horní a dolní končetiny. Na tuto úroveň jsou kladeny nejmenší požadavky na zapojení myšlenkových pochodů a člověk

vystupuje a chová se jako výkonný člen. Příkladem těchto činností z pohledu jízdy autem jsou například ovládání volantu, sešlápnutí plynového či brzdového pedálu. [2]

Koordinační úroveň

V této úrovni se nachází všechny činnosti, které jsou aplikované přímo na konkrétní stroj. Lidský operátor rozpoznává různé stavy systému, vyhodnocuje je a následně volí odpovídající činnost tak, aby stav systému odpovídal předem daným pravidlům, normám nebo postupům. Tyto činnosti se musí lidský operátor naučit - nejdříve se tedy musí na množinu stavů natrénovat. Každému stavu odpovídá dané provedení. Při neustálém opakování těchto činností může dojít k postupnému vyřazení myšlení z činnosti a naučení činnosti jako zvyklosti. Zjednodušeně řečeno člověk je již zkušený a vyhodnocování stavů se stává běžnou a automatickou záležitostí. Příkladem těchto činností při řízení automobilu je změna jízdního pruhu, řazení rychlostí (které zpočátku bývá většinou problémové a po určité době již řidič provádí tento úkon automaticky), sledování dopravního značení nebo světelných křižovatek apod. [2]

Kognitivní úroveň

Jinak se této úrovni také říká taktická. Patří do ní činnosti, které jsou spojené s rozhodováním. Nelze se na ně připravit do takové úrovně, jako pro činnosti v koordinační úrovni. Lidský operátor vyhodnocuje náhlé a neočekávané situace nebo stavy systému, pro které ještě není vytvořena konkrétní činnost. Tato rozhodnutí jsou závislá na zkušenostech, dovednostech a schopnostech lidského operátora. Dochází k aktivaci lidského mozku a člověk zapojuje do řízení MMS systému své myšlení. Člověk jako regulátor a jeho myšlenkové postupy jsou originální a jedinečné - tvorba modelu chování lidského operátora na této úrovní je tedy nejobtížnější. [2]

1.1.3 Druhy a způsoby řízení v systémech MMS

Systémy MMS se dělí na dva základní prvky - řízený (stroj) a řídící (lidský operátor - regulátor). Druhy řízení a jeho zásahy v systémech MMS rozdělit do tří kategorií:

Přímé a zpětnovazební řízení

Jedná se o nejnižší úroveň řízení systému MMS. Základem jsou naučené postupy - například rozjezd vozidla, udržování vozidla ve středu jízdního pruhu, udržení povolené rychlosti. Tyto činnosti provádí lidský operátor (řidič) automaticky. Regulátorem je pro tento případ pohybový aparát člověka. [1]

Řízení založené na pravidlech

Tento druh řízení nastává, jakmile je operátor nucen reagovat na vnější vlivy (například reagovat na pohyb jiných vozidel či dopravní značení). Provádí se složitější činnosti, kupříkladu předjíždění, odbočování nebo projíždění křižovatek. Operátor tak uzpůsobuje své činnosti dle platných pravidel a svých naučených dovedností. Vyhodnocuje taktéž chování ostatních přítomných systémů (řidičů ve vozidlech) a na základě analýzy pak dokáže situace rozpoznat a dávat příslušné pokyny v podobě regulačních zásahů na nejnižší úrovni (přímého a zpětnovazebního řízení). [1]

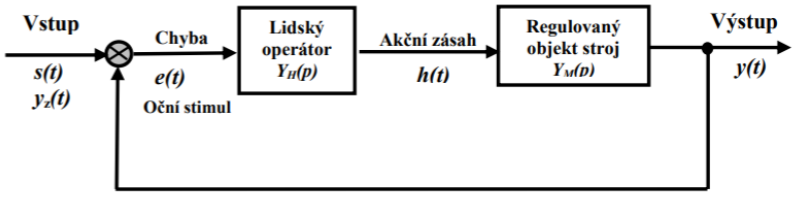
Řízení založené na znalostech

Je nejvyšší úrovní řízení, do které se řadí činnosti, u kterých lidský operátor vychází ze svých znalostí - například plánování trasy, předpovídání chování ostatních řidičů. Operátor pak provádí optimální variantu svých činností. Dá se říct, že se jedná o rozhodování v podmínkách neurčitosti s využitím předchozích zkušeností. Analýza a vyhodnocování situací, na které operátor reaguje, probíhají s daty, ve kterých se ve velké míře projevuje nejistota. [1]

1.1.4 Kompenzační řízení a lidský operátor

Model lidského operátora lze získat za předpokladu, že řízení MMS systému má kompenzační charakter a interní lidské schopnosti jsou společně s kognitivní úrovní potlačeny. Výhodou je také, pokud nejsou v systému přítomny několikanásobné zpětné vazby - lze tak jednoduše pozorovat a měřit hodnoty sledovaných parametrů, které se pak použijí k hodnocení dynamického chování operátora. Nejvýhodnějším postupem pro zisk dat o lidském regulátoru je sledovat způsob jeho řízení a řídící postupy s různými objekty, které mají známý model. Velice důležitým předpokladem je, aby byly vlastnosti a regulační schopnosti operátora po celou dobu měření stálé. Kompenzační řízení je vyobrazeno na obrázku 1.2 a reprezentuje mnoho činností operátora, mezi které, pro případ této práce, patří například udržování polohy vozidla ve stálém jízdním pruhu - operátor (řidič) tak musí neustále sledovat polohu svého vozidla vzhledem k vytyčené trase a reagovat na neustálou změnu směru, která při řízení nastává. Lze konstatovat, že tyto činnosti jsou monotónní a operátor je vykonává automaticky, podpořen svými naučenými reflexy, dovednostmi a schopnostmi. [2]

Na vstupu MMS systému se nachází vstupní signál $s(t)$, který představuje žádanou hodnotu $y_z(t)$. Operátor získává audiovizuální stimuly a vnímá rozdíly mezi požadovaným vstupním signálem $y_z(t)$ a aktuálním výstupním signálem $y(t)$ - jedná se o regulační odchylku (chybový signál $e(t)$). Operátor se snaží minimalizovat tuto



Obr. 21: Kompenzační řízení v systému MMS

Obr. 1.2: Blokové schéma řízení systému MMS [1]

odchylku a provádí akční zásahy $h(t)$ do systému tak, aby byl výstupní signál co nejvíce podobný vstupnímu, za podmínky stability celého MMS systému. Pokud dojde k porušení těchto podmínek, dochází k přenosu systému do nebezpečného stavu. [2]

1.1.5 Měřené parametry operátora

Z teoretického rozboru systému MMS, konkrétně zaměřeného na systémy řidiče a vozidla, vyplývají jevy, které je vhodné analyzovat pro potřebu modelování operátora. Patří mezi ně:

- Sledování vytyčené trasy - simulující jízdu v jízdním pruhu, ve kterém se musí řidič po určitou dobu udržet, přičemž dochází k jemným změnám polohy volantu.
- Analýza reakční doby řidiče - reakce na náhlé změny při jízdě. V reálných situacích se jedná o nepředvídatelné jevy, jakými je například náhlá změna jízdního pruhu vozidla před řidičem, vniknutí cizího předmětu do trasy auta apod. Hlavním smyslem měření je zisk reakční doby řidiče, která odpovídá celému procesu zpracování a vyhodnocení informace s navazující výslednou činností.
- Pozorování stylu řízení - nejlépe na trase s neustálými změnami směru a rychlosti vozidla. Typicky například městské prostředí.

Pro analýzu výše zmíněných jevů je tedy zapotřebí pozorovat různé parametry při řízení: úhel natočení volantu, míra akcelerace nebo brzdění, poloha vozidla, vzdálenost od vytyčené trasy a rychlost vozidla.

1.2 Vývoj simulátoru

Pod pojmem *simulátor* v kontextu počítačových technologií si lze představit 2D/3D virtuální svět, který má svou hlavní postavu pod kontrolou uživatele - ať už se jedná o člověka, zvíře, vozidlo nebo cokoliv jiného. Primární smysl simulátorů je co nejvíce se přiblížit realitě a simulovat proces reálného světa - řízení vozidla, pilotování letadla, vojenské simulátory apod. Simulátory se dají označit jako podskupina herních aplikací, která nabízí hráči určitý druh výzvy, která mu pomáhá se rozvíjet a plní úlohu soutěživé motivace. Tyto herní aplikace se staly jedny z hlavních průkopníků vývoje počítačové grafiky, simulací, umělé inteligence apod. S ohledem na cíl této diplomové práce je vhodné, aby byl vytvořený simulátor koncipován tak, aby uživatele zaujal a přibližoval se co nejlépe reálnému zážitku z jízdy.

V průmyslu či ve výzkumu se pracuje s pojmy jako real-time aplikace, real-time sběr dat apod. Snahou je se co nejvíce přiblížit zisku dat/odezvě v reálném čase - rozlišitelnost až v řádu nanosekund. Avšak tomuto rozlišení není u herních aplikací potřeba. Tyto aplikace se pojí s termínem *soft real-time*. V herním softwaru je vytvářen zjednodušený model naší reality, podle potřeb aplikace. Není zapotřebí zahrnovat každý detail z běžného života - vše pro zmírnění výpočetní zátěže. Na zručnosti vývojáře pak závisí, zdali je/bude tento model světa k nerozeznání od toho reálného, z čehož bude vyplývat kvalita zážitku uživatele při používání aplikace. V průmyslu je většinou kladen důraz na hard real time. Nesmíme připustit chybu či velkou odezvu od nějakého vstupu/výstupu - může dojít k poruše a tím se nám zvýší finanční náklady. U herních aplikací však tento problém nenastává. Pokud dochází k poruše či nahromadění požadavků, sníží se výpočetní výkon - kupříkladu se zmenší počet snímků za sekundu nebo dojde ke zvýšení odezvy mezi vstupem uživatele a výstupem aplikace. Uživateli ani výrobci však není způsobena žádná škoda. Jedním z příkladů real-time u herních aplikací je požadavek aktualizace obrazu na monitoru nebo simulaci fyziky v prostředí aplikace. Ta může vyžadovat až 120 obnovení za sekundu, aby byla aplikace stabilní. [4]

1.2.1 Herní engine

Popis termínu *herní engine* se datuje do poloviny 90. let 20. století, kdy vyšla hra s názvem Doom od vývojáře ID Software. Software byl vyvíjen tak, aby byly jednotlivé důležité komponenty odděleny od sebe - 3D renderování grafiky, detekce kolize, audio, pravidla hry apod. To vneslo nový koncept do vývoje podobného softwaru - programátoři mohli vyvíjet nové produkty s novým designem, herními pravidly,

grafickým zpracováním avšak s minimálním zásahem do jádra softwaru - enginu. Otevřely se tak nové možnosti i nezkušeným programátorům, kteří tento software mohli upravovat bez konkrétnější znalosti jádra a z původních jednoúčelových programů se tak staly mnohoúčelové. Zjednodušeně řečeno, herní engine je software, který lze neustále obohacovat a může být použit jako základ pro spousty druhů aplikací (závodní hry, real-time strategie apod.) bez nutnosti velkých zásahu do jeho struktury/kódu. [4]

V dnešní době mohou vývojáři využívat svůj vlastní engine, nebo použijí licencovaný engine třetí strany. V konečném důsledku se jedná hlavně o ekonomické řešení - je jednodušší a levnější vyvíjet software na již vytvořené platformě s podporou, než vytvářet úplně novou. Není přesně dána hranice mezi herním enginem a vlastním programem. Existuje totiž mnoho enginů, které fungují na odlišné bázi a nabízí programátorům různé možnosti vývoje.

V ideálním případě poskytuje herní engine základ pro všechny druhy aplikací. Vývojář těchto enginů se snaží tomuto ideálu přiblížit, je však velice složité vytvořit univerzální nástroj na všechny druhy aplikací. Většina dostupných herních enginů je zaměřena na určitý druh hardwaru (PC, herní konzole, mobilní hardware) nebo aplikace (hry z pohledu první osoby, real-time strategie, hry pro více hráčů apod.). Čím širší záběr herní engine má, tím těžší je jeho optimalizace na konkrétní produkt a na konkrétní hardware. Vytvoření takového softwaru provází spousta kompromisů - například engine, který je vytvořen pro renderování vnitřních prostor (architektonické programy) bude mít menší výkonnost při renderování vnějšího prostředí. S neustálým vylepšováním hardwaru a renderovacích algoritmů se však tyto rozdíly potlačují. [5]

1.2.2 Specifika závodního simulátoru

Herní aplikace se dají specifikovat dle svého žánru - podle toho také vybíráme herní engine. Mezi typické žánry patří:

- Hry z pohledu první osoby (FPS)
- Bojové hry
- Závodní hry
- Real-time strategie (RTS)
- Sportovní hry
- a další...

Simulátor, který je v rámci této práce vyvíjen, spadá pod kategorii závodních her. Nutno podotknout, že účelem není vytvořit závodní hru s protivníky a tratí, kterou musí uživatel projet za co nejrychlejší čas. Žánr závodních her pod sebou shromažďuje všechny aplikace, jejichž primárním úkolem je řízení vozidla na vytvořené ploše. Žánr má spoustu podkategorií, mezi které patří právě simulátory, které se snaží vytvořit co nejreálnější zážitek řízení vozidla. Patří mezi ně například trenažéry v autoškolách, komerční aplikace nebo simulátory s konkrétní úlohou - příkladem je Carla či AirSim, což jsou simulátory autonomního řízení. Dalším žánrem je takzvaný *arcade* - tyto aplikace upřednostňují zábavu nad reálností. [4]

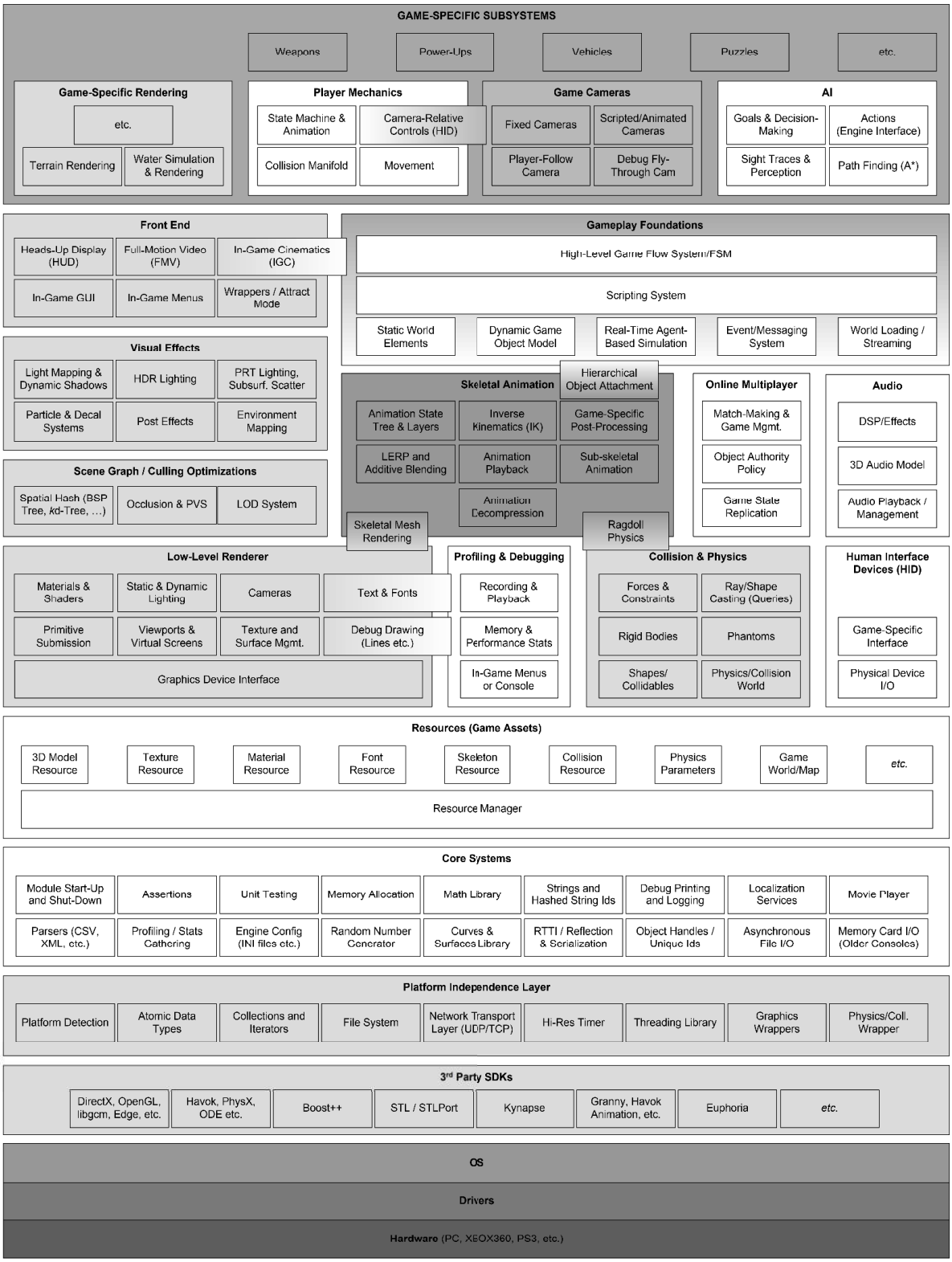
Závodní aplikace je většinou velmi lineární - máme vytvořenou přesně danou trať bez otevřeného světa, kam by mohl uživatel vstoupit. Grafické detaily se soustředí hlavně na model auta a trati, případně okolní prostředí v závislosti na vzdálenosti od hráče. Typické rysy závodních her jsou následující:

- Speciální renderování vzdálených prvků - například 2D modely stromů, budov, použití LOD (Level Of Detail) apod.
- Trať (cesta) je rozdělena na jednoduché 2D oblasti - sektory, které se využívají pro optimalizaci renderování a ke zlepšení mapování (například pro umělou inteligenci, která je využívána pro autonomní řízení).
- Aplikace nabízí pohled z první osoby (uvnitř vozidla, typické pro simulátory) nebo z třetí osoby - kamera následuje auto (typické pro arkády).
- Pokud kamera koliduje s prostředím (například projíždíme tunelem), je důležité těmto situacím předejít.

A samozřejmě spoustu dalšího. Neexistují přesně daná specifika závodních her, avšak výše uvedený seznam poukazuje na rozdíly oproti jiným herním žánrům. [4]

1.2.3 Runtime herního enginu

Herní engine se většinou skládá ze dvou částí - runtime a sady nástrojů. V následující kapitole je popsán obecný runtime herního enginu a jeho propojení se sadou nástrojů. Aby bylo možné simulátor řízení vozidla vytvořit, je potřeba vědět, s jakým softwarem pracujeme a na jaké bázi. Na obrázku 1.3 jsou popsány jednotlivé komponenty herního enginu. Ty jsou rozděleny do vrstev, přičemž vrchní vrstvy jsou závislé na těch spodních. Nutno podotknout, že se nejedná o standardizované rozdělení a trh s herními enginy se neustále vyvíjí. [4]



Obr. 1.3: Architektura runtime herního enginu [4]

Hardware

Úplně nejnižší vrstvou je cílový hardware, na jehož parametry je aplikace vytvořena/navržena. Typické platformy jsou počítačové systémy (Microsoft Windows, Linux, MacOS), mobilní zařízení (iPhone, Android chytré telefony) nebo herní konzole (Playstation 4, Xbox One, Ninendo Wii). Každé z těchto zařízení nabízí odlišný hardware a z toho také odvozený výkon. Je důležité si určit, na jaké z těchto zařízení je aplikace tvořena. Například herní konzole poskytují standardizovaný hardware a výkon (podobně jako Apple iPhone), takže optimalizace je pro vývojáře jednodušší, oproti běžným PC, které se může skládat se široké škály komponentů a výrobců. [4]

Ovladače zařízení

Nízkoúrovňový software, který je dodáván výrobcem HW nebo OS a umožňuje komunikovat a ovládat konkrétní hardware (například ovladače grafické karty, USB portu apod.).

Operační systém

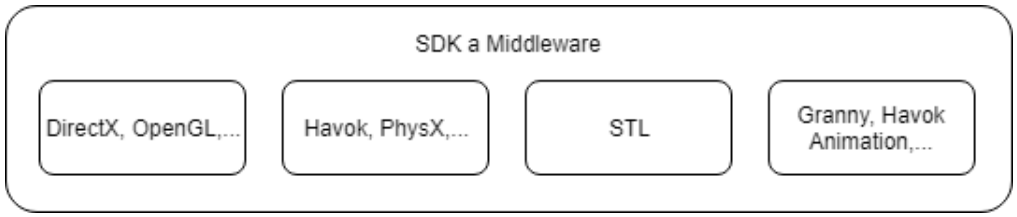
Herní aplikace je stále jen software, který běží na určitém operačním systému. Nejčastěji jsou tyto aplikace vyvíjeny na Microsoft Windows, případně herní konzole, které mají svůj vlastní operační systém. Tím, že se jedná o další aplikaci, není hra přímým uživatelem hardware, tudíž se o něj musí dělit i s ostatními aplikacemi. To v historii neplatilo pro herní konzole, ale ty v dnešní době již mají svůj operační systém (Playstation 3/4 nebo Xbox 360/One), na kterém rovněž běží aplikace (například komunikační platforma, přehrávač hudby apod.), které mohou přerušit herní aplikaci v jejím běhu a odebrat jí tak potřebný výkon.

Software Development Kit / Middleware

Herní enginy využívají pro svůj chod SDK třetích stran. Některé příklady jsou uvedeny na obrázku 1.2.3 a popsány dále v kapitole.

Grafické SDK

DirectX od firmy Microsoft nebo OpenGL jsou grafické SDK, které poskytují aplikační rozhraní (API) pro přímé ovládání (převážně) hardwaru grafických karet. Kromě tvorby počítačových her se využívají pro CAD programy, virtuální realitu či



Obr. 1.4: Software Development Kit vrstva [4]

grafické uživatelské prostředí. [4]

SDK pro zprostředkovávání kolizí a fyziky

Havok, opět od Microsoftu, je middleware, který se kromě výpočtů kolizí a fyziky předmětů věnuje také umělé inteligenci či animacím. PhysX je alternativa od firmy NVidia, která nabízí podobné funkce jako Havok - například dynamiku těla, vozidel, ragdoll, částicové efekty nebo simulace proudění tekutin. PhysX využívá kupříkladu Unreal engine. [4]

STL - Standard Template Library

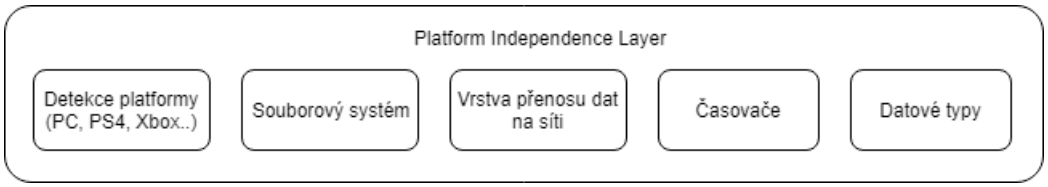
Softwarové knihovny jazyka C++, které seskupují algoritmy a spoustu funkcí. Příkladem může být STLport, Boost, Loki. Otázkou je, zda-li je vhodné tyto knihovny používat v herních enginech. Existují názory, které jsou proti - a to z důvodu nejasné alokace paměti pro výkonné aplikace (dochází k fragmentaci paměti). Použití na PC je určitě vhodné, avšak na konzolích se právě doporučuje vytvořit si své vlastní struktury z důvodu omezené virtualizace paměti. [4]

Animační SDK

Pro animaci se využívají SDK, jako je například Granny - nabízí robustní 3D animace, jednoduchý export pro jiný software, práci s časem a logické rozhraní. V dnešní době jeden z nejpoužívanějších. V předešlých sekcích bylo řečeno, že Havok SDK se věnuje převážně výpočtům kolize a fyziky. V dnešní době však SDK rozšiřují své pole působnosti a jedním z příkladů je právě Havok Animation. Vzhledem k tomu že animace a kolize/fyzika k sobě nemají tak daleko, není takové propojení překvapivé. [4]

Platform Independence Layer

Vzhledem k rostoucí nabídce hardwaru, na který můžeme herní aplikace vyvíjet, je kladen důraz na Platform Independence Layer - vrstva, která je nad HW, ovladači,



Obr. 1.5: Platform Independence Layer [4]

OS a SDK, která zajišťuje potřebné chování na různých platformách pomocí volání správných funkcí, API apod. Největší herní studia vyvíjejí hry na PC a také na konzole (PS4, Xbox One) a tato vrstva zajišťuje, aby na všech těchto zařízeních engine fungoval správně. Součástí je například detekce platformy, souborový systém, síťová komunikace, použití správných datových typů apod. [4]

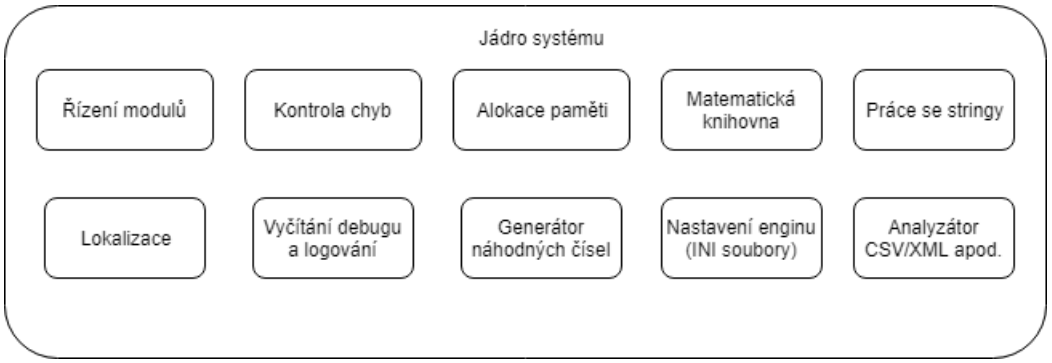
Jádro systému

Každá složitější aplikace vyžaduje své jádro, které poskytuje set důležitých funkcí. Mezi ty například patří:

- *Kontrola chyb* - zachycuje logické chyby v kódu, ve finálním produktu se neobjevuje
- *Správa paměti* - každý herní engine má svůj vlastní systém alokace paměti - je potřeba rychlé alokace a dealokace a eliminovat fragmentaci paměti (kvůli které dochází k vytvořením nevyužitelných bloků prázdné paměti)
- *Matematická knihovna* - herní engine je náročný na matematické výpočty a tyto knihovny nám poskytují širokou škálu matematických operací - rotace, geometrické operace/modely, výpočty rovnic atd.
- *Vlastní datové struktury a algoritmy* - pokud není využita STL (od třetích stran ve vrstvě SDK), je vhodné vytvořit si vlastní datové struktury a algoritmy, které s nimi pracují. Mezi typické datové struktury patří lineární seznam, dynamicky alokovaná pole, binární stromy apod. Algoritmy zastřešují vyhledávání, seřazování (sort) apod. Když je tato vrstva vytvořena přesně na daný engine, dochází k vylepšení a optimalizaci herního enginu na cílových platformách. [4]

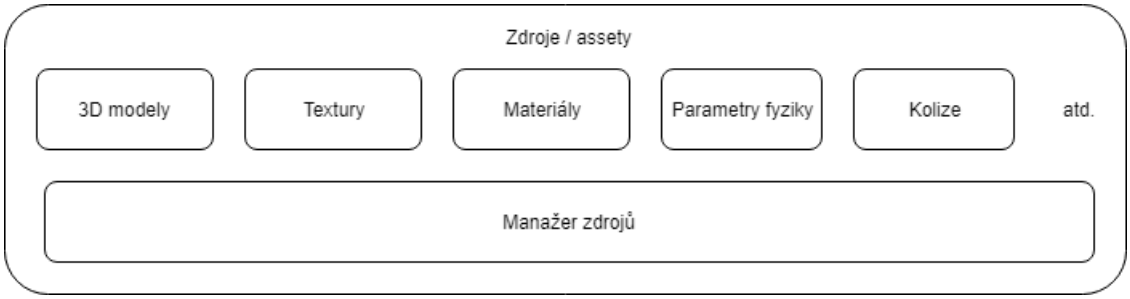
Manažer zdrojů

Tato vrstva zajišťuje přístup a použití různých druhů vstupních dat enginu a assetů, jakými jsou například 3D modely, mesh, textury, materiály, fonty atd. Je několik přístupů, jak implementovat tuto vrstvu. Kupříkladu Unreal engine vytváří balíčky,



Obr. 1.6: Jádro systému - příklady [4]

které zastřešují všechny typy (zobrazené na obrázku 1.7) assetů. Naopak některé herní enginy nechávají přístup k těmto souborům čistě na programátorovi, který přistupuje k surovým datům a pak je zpracovává dle svého uvážení. [5]



Obr. 1.7: Manažer zdrojů [4]

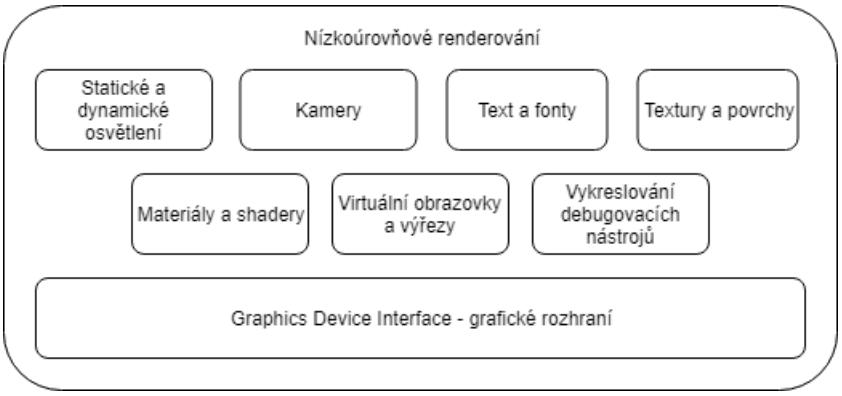
Renderovací engine

Renderovací engine je jedna z nejdůležitějších a zároveň nejsložitějších komponent herního enginu. Může být navržena mnoha způsoby, avšak základ je pro všechny podobný (závisí hlavně na grafickém hardwaru, se kterým pracuje) - v následující kapitole si jej popíšeme.

Nízkoúrovňové renderování

Tento modul pod sebou zastřešuje veškeré základní renderování herního enginu. Je zaměřen hlavně na renderování jednoduchých geometrických obrazců s důrazem na rychlost a kvalitu. Vůbec nebere v potaz scénu, kterou hráč zrovna zobrazuje.

Graphics Device Interface má na starosti propojení a inicializaci grafických zařízení pro grafické SDK (DirectX/OpenGL). Ostatní komponenty spolupracují tak,



Obr. 1.8: Nízkoúrovňové renderování [4]

aby došlo k vytvoření jednoduchých geometrických útvarů - *meshů* (spojení hran a ploch pro vytváření 3D modelů), vektorů, bodů apod. Každý z těchto útvarů má pak daný materiál/texturu a nasvícení, které pak určují dalším elementům herního enginu jejich vykreslování. [4]

Optimalizace scény

Renderování celého prostředí je výkonově náročné, z toho je zapotřebí nástroje, který renderování optimalizuje. Vrstva optimalizace scény limituje počet útvarů, které jsou vykreslovány nižší vrstvou v závislosti na určení scény/pohledu hráče. Kupříkladu v menších herních světech stačí pouze odstranit objekty, které záběr scény nevidí. U větších je potřeba komplikovanějších algoritmů, určujících prostor pro vykreslování. Zajišťuje také různou kvalitu vykreslovaných objektů, která závisí na vzdálenosti od uživatele v prostředí aplikace (popisováno pojmem LOD - Level Of Detail). [5]

Vizuální efekty

Modul, který je zaměřen na renderování speciálních efektů ve spolupráci s nízkoúrovňovým renderováním. Některé prvky mohou být součástí nižší vrstvy, záleží na architektuře daného enginu. Příklady vizuálních efektů jsou následující:

- Částicový (particle) systém - kouř, oheň...
- Obtiskový (decal) systém - stopy...
- Mapování osvětlení a prostředí
- Dynamické stíny
- Celoobrazové efekty - aplikují se po vyrenderování 3D scény, příkladem je HDR, anti-aliasing, úpravy barev [5]

Front end

Modul, který komunikuje s uživatelem a má za úkol zobrazení videí/animací, nastavení a ovládání aplikace. Můžeme si jej představit jako 2D grafické rozhraní na 3D scéně. Součástí jsou například:

- HUD (Heads-Up Display) - informuje uživatele o aktuálním stavu, při závodních hrách kupříkladu o rychlosti vozidla
- Hlavní menu - slouží k nastavení aplikace, výběru herních scénářů nebo ukládání pozic
- GUI (Graphical User Interface) - složitější rozhraní pro manipulaci s herním světem (např. inventář hráče)
- FMV (full-motion video) - přehrává video soubory
- IGC (in-game cinematics) - slouží k přehrávání videí renderovaných v herním enginu (v 3D) [4]

Nástroje pro debugování

Podobně jako některá IDE mají své debugovací nástroje, tak i herní enginy mají své. Aplikace vytvořené enginy jsou brány jako real-time systémy a je potřeba pro vývojáře i uživatele mít co nejrychlejší zpětnou vazbu z důvodu optimalizace a vylepšení výkonu aplikace. Herní enginy nabízejí nástroje typu: kontrola kódu, zobrazení statistik na hrací plochu, ukládání dat (využití paměti, stisknutí klávesy, využití CPU...) do logovacích souborů, vykreslování debugovacích obrazců (z renderovací vrstvy) apod. [4]

Kolize a fyzika

Detekce kolizí je jedna z nejdůležitějších věcí v každé herní aplikaci. Bez ní by nemohl uživatelský objekt jakkoliv zasahovat do vytvořeného virtuálního světa. Většina her využívá určitý stupeň realismu (některé hry jsou arkádové, není potřeba takové míry realismu, oproti simulátorům). Vyhodnocení kolizních situací obstarává fyzikální systém a jeho logika. Pro tyto výpočty se využívá software třetích stran (PhysX, Havok), viz kapitola o SDK. [4]

Animace

Pokud jsou součástí herní aplikace objekty, u nichž je očekáván pohyb - žijící stvoření, vozidla apod., je potřeba využití animačního modulu. V herních aplikacích se

využívá několik základních typů animace:

- Animace textur
- Animace tuhých těles/těl
- Animace koster (skeletal animation)
- Vektorová animace

Nejvyužívanějším druhem animace je skeletal animation. Objekt animace a jeho model je navázán na kostru. Pokud se hýbou kosti kostry, tak se pohybuje i model. Například objekty představující lidské osoby obsahují kostru podobnou té lidské. Animační systém produkuje pozice pro každou kost a tyto pozice poté posílá renderovacímu enginu jako soubor matic. Tomuto procesu se také říká skinning. Animace jsou také provázány s fyzikálním systémem, který simuluje pohyb objektu - určuje tedy pozice různých částí objektu (kostry). [4]

HID - Human Interface Devices

Pro aktivní zapojení uživatele ve virtuálním světě jsou potřebná vstupní zařízení: klávesnice, myš, gamepad, volant, pedály a spoustu dalších. Tato zařízení pak posílají informace o svých aktuálních vstupech aplikaci, která podle nich koná různé úkony dané její strukturou. Aplikace však může také informace či data posílat hráči zpátky, například formou force feedback při řízení nebo vibrací gamepadu. V dnešní době jsou již vstupní/výstupní zařízení komplikovaná zařízení se spoustou funkcí, které kladou důraz na vrstvu HID - řeší se například konflikty současně zmáčknutých kláves, míra natočení páčky na gamepadu, nástupné/sestupné hrany stisknutí kláves apod. Tato vrstva tedy umožňuje mapování vstupních zařízení na prvky v aplikaci (spouštění funkcí, pohyb postavy, natočení kamery,...). [5]

Audio

Rovněž zvuková stopa herní aplikace potřebuje renderování. Jakýkoliv objekt na scéně, případně samotné prostředí, může generovat zvukové signály, které je potřeba renderovat s rozdílnou kvalitou či prioritou. Je důležité brát v potaz umístění postavy nebo jakéhokoliv objektu v herním světě, aby došlo ke správnému nasměrování zvukového signálu a také přehrávání ve specifikovaném audio systému. Některé herní enginy mají svůj vlastní audio engine (například Unreal Engine 4), jiné využívají engine třetích stran, kupříkladu XACT, XAudio2 či X3DAudio. Pro dobrý zážitek a optimalizaci aplikace je vhodné věnovat renderování zvuku nemalou část

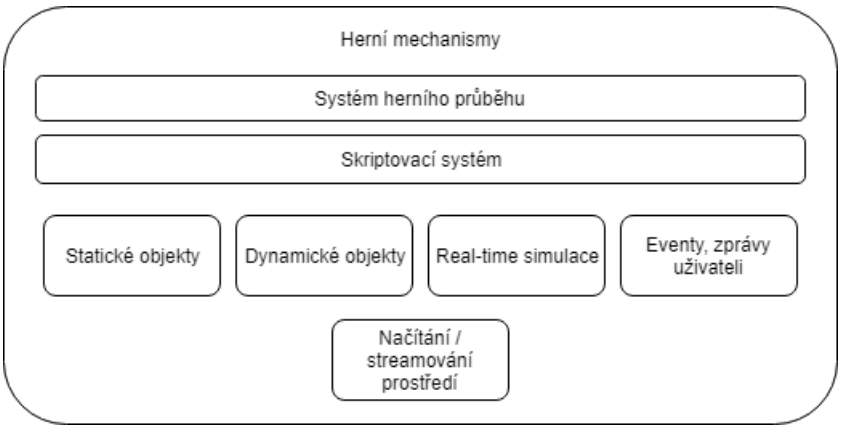
práce na projektu. [5]

Síťová vrstva

Spousta herních aplikací nabízí hru pro více hráčů, pro účel simulátoru řízení však tato možnost není potřebná. Lze však využít komunikační API, které umožňuje komunikaci s jinou aplikací spuštěnou na síti. Tím je možné přenášet důležitá data či příkazy, například pro spuštění různých skriptů nebo scénářů.

Herní mechanismy

Pod pojmem herní mechanismus (v anglickém jazyce *gameplay*) je myšlen soubor pravidel, podle kterých se řídí prostředí v aplikaci. Jaké možnosti má hráč nebo objekty, se kterými interaguje. Určují cíle hry a možnosti, jak jich dosáhnout. U závodních her lze uvést příklad herního mechanismu (neboli módu). Herní mód "time attack"zastřešuje typ hry, ve které existuje přesně daná trasa s bránami, které musí uživatel projet s tím, že je mu vždy při průjezdu měřen čas. Na poslední bráně se uživateli zobrazí výsledný čas a může tak soupeřit s jinými uživateli o nejlepší výsledek. Herní mechanismus je napsán v původním jazyce enginu nebo ve vyšším, skriptovacím (Blueprintový systém v Unreal Engine nebo Flow Graph v CryENGINE). Pro komunikaci mezi herním mechanismem a moduly enginu na nižší úrovni je využívána vrstva na obrázku 1.9. [5]



Obr. 1.9: Vrstva herních mechanismů [4]

Herní prostředí a modely objektů

Vrstva je vyobrazena na diagramu 1.9. Její součástí jsou objekty ve světě, které mohou být buď statické nebo dynamické. Mezi typické objekty patří:

- Statické objekty na pozadí - budovy, cesty, povrchy ...
- Dynamické objekty - mohou na ně být použity fyzikální výpočty a uživatel s nimi může interagovat (nábytek, kamení, věci z běžného života)
- Model hráče (Player Character) - nejčastěji člověk, vozidlo apod.
- Model nehrající postavy (Non-Player Character)
- Projektily
- Osvětlení
- Kamera
- Spousta dalších...

Eventy

Další části této vrstvy je systém na vyřizování eventů/přerušení. Tyto eventy jsou dány herním mechanismem a může se jednat například o funkci, která nám umožní vzít předmět do ruky při zmáčknutí klávesy, nebo zobrazení času na obrazovce při projetí určitého úseku mapy. Při splnění podmínky pro daný event dojde k zavolání funkce s potřebnými parametry. Toto volání funkce je pak předáno event systému, který funguje podobně jako systém přerušení a může jednotlivé eventy řadit do fronty, či jim přiřazovat prioritu, se kterou budou vyřizovány. [5]

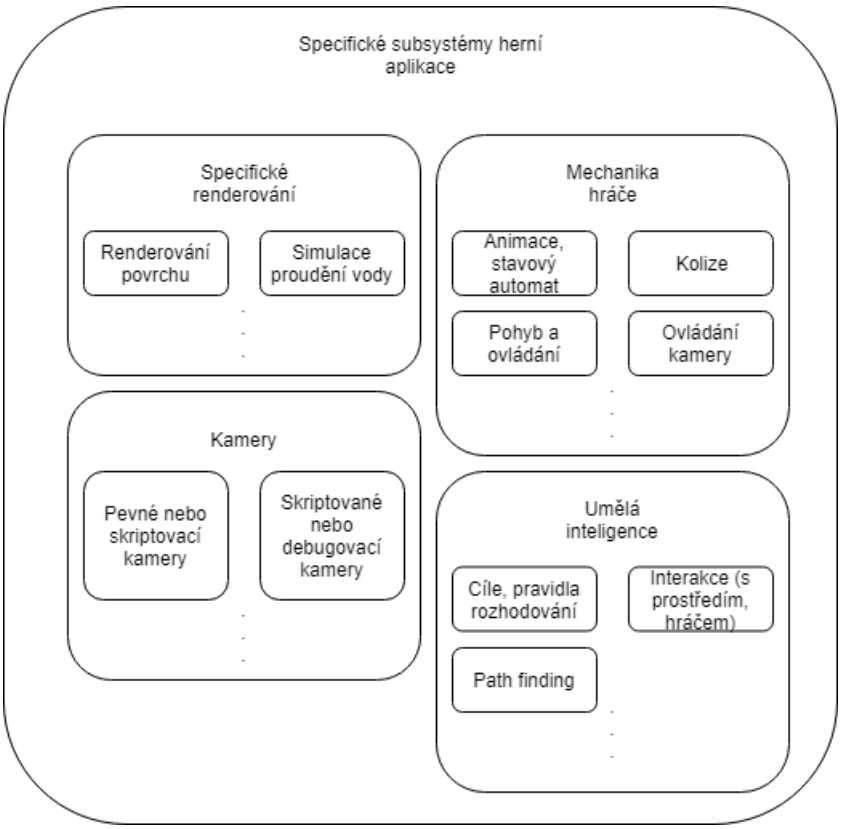
Skriptovací systém

Většina nejvyužívanějších herních enginů v sobě má zakomponovaný skriptovací systém, který určuje pravidla chování virtuálního světa a umožňuje je rychle měnit za vývoje aplikace. Pokud by skriptovací systém nebyl součástí, bylo by nutné všechny nižší vrstvy a jejich chování kódovat na nižší úrovni a vše by tak pro vývojáře bylo složitější. Příkladem je výše zmíněný Flow Graph nebo Blueprintový systém. [5]

Specifické subsystémy herní aplikace

Úplně nejvyšší vrstva zastřešuje specifika vyvinuté herní aplikace - čím se odlišuje od ostatních na trhu a co jsou její hlavní (silné) stránky. Nutno podotknout, že se celkově nejedná o standardizovaný model, takže součástí této vrstvy může být cokoliv specifického pro danou aplikaci.

Některé aplikace jsou orientovány na velký, otevřený svět - v této vrstvě tedy budou mít nejspíše specifický modul renderování světa, povrchu apod. Jiné se zase zaměřují na interakci hráče s prostředím, budou mít tedy vyvinutější umělou inteligenci pro chování NPC a tak dále. Více podobných příkladů je uvedeno na obrázku 1.10. [4]



Obr. 1.10: Vrstva specifických subsystémů [4]

Nástroje a assety

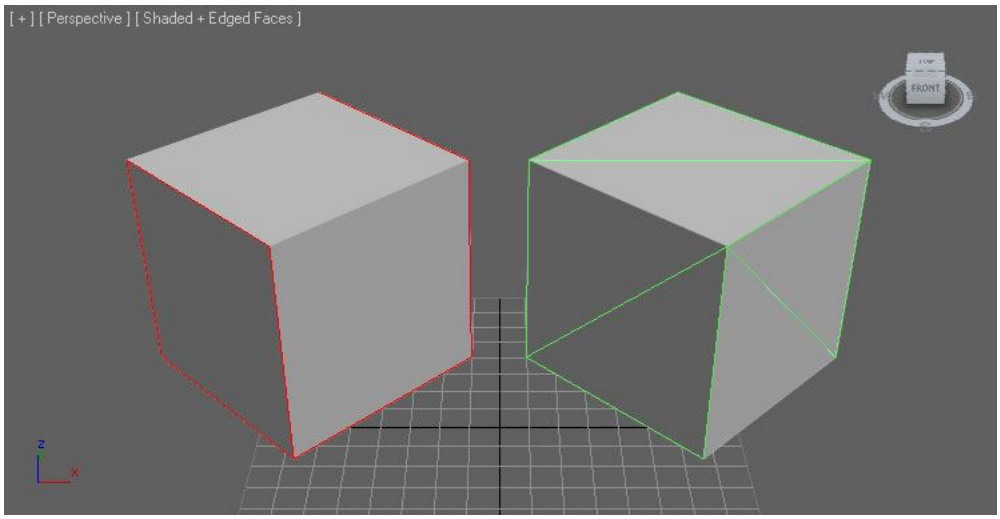
K vytvoření herní aplikace pomocí herního enginu je potřeba velkého množství souborů, objektů, textur apod., které jsou sjednoceně nazývány "assety". Pokud je potřeba vytvořit model objektu, je lepší využití softwaru třetí strany, který je na to určen (například program Blender či Maya) a potom vytvořený objekt exportovat do herního enginu. Herní enginy některé z těchto nástrojů mají implementovány, avšak plně nenahradí externí software na danou problematiku určený. V následující kapitole jsou popsány některé druhy assetů, bez kterých by aplikace nemohla vzniknout.

Digital Content Creator

DCC je druh aplikace, který se zaměřuje na vytvoření konkrétního typu objektu nebo dat. Příkladem může být Maya nebo 3ds Max pro vytváření 3D modelů (mesh), Adobe's Photoshop pro vytváření textur, SoundForge pro vytváření zvukových stop a spousta dalších. Nejčastějším problémem, který musí herní engine řešit, je ten, že výstupní soubory z DCC aplikací obsahují až příliš mnoho zbytečných a detailních informací, přičemž herní engine využívá pouze část z nich. To způsobuje zbytečný nárůst poptávky výkonu a zpomalení aplikace. DCC aplikace ale umožňují export dat ve zjednodušených, standardizovaných formátech, se kterými si již herní engine poradí. [4]

3D model - mesh

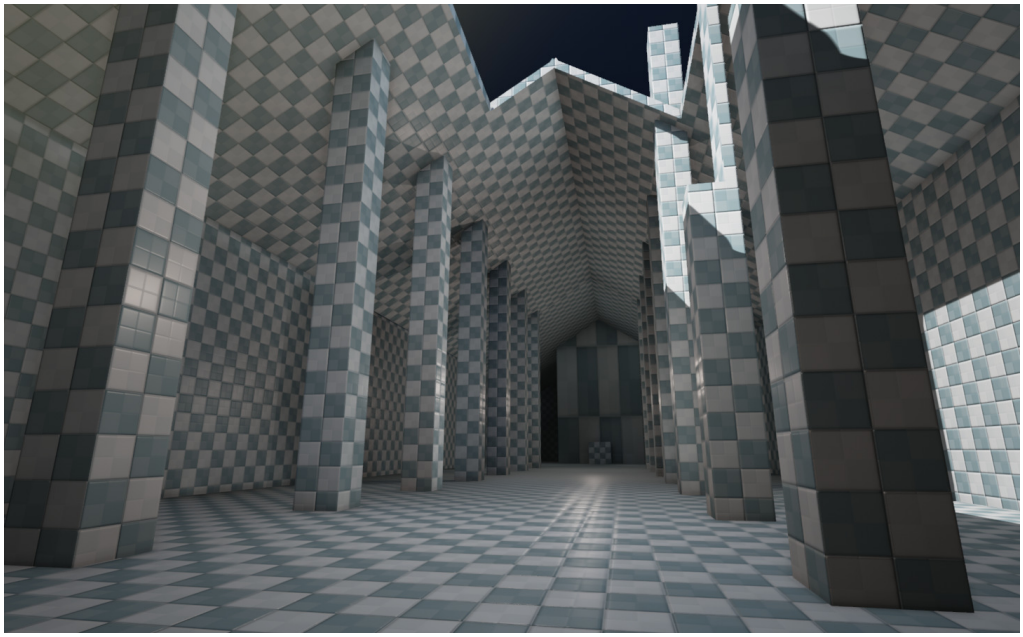
Model, se kterými herní engine pracuje v editoru, se nazývá *mesh*. Je tvořen geometrickými útvary, nejčastěji trojúhelníky, případně vyššími řády (čtverce atd.). V dnešní době se však využívají primárně trojúhelníkové obrazce, kvůli optimalizaci výkonu a výsledného renderování. Na mesh se aplikují materiálové objekty pro poskytnutí potřebného vzhledu (barva, odraz, textura apod.). Meshe se vytvářejí pomocí 3D modelových programů - 3ds max, Maya, Blender, SoftImage, ZBrush aj. [4]



Obr. 1.11: Zobrazení mesh a jeho rozdělení na trojúhelníky [6]

Alternativou mesh je *brush geometry*. Tyto modely jsou definovány jako soubor spojených ploch, což lze pozorovat na obrázku 1.12. Využívají se pro rychlé vytvoření jednoduchých modelů nebo prostor. Většinou jsou implementovány v herním enginu. Jejich nevýhodou je nízké rozlišení a složitost při vytváření komplexnějších

objektů. Také, oproti mesh objektům, nepodporují animace. Většinou se pro statické objekty používají static mesh. [4]



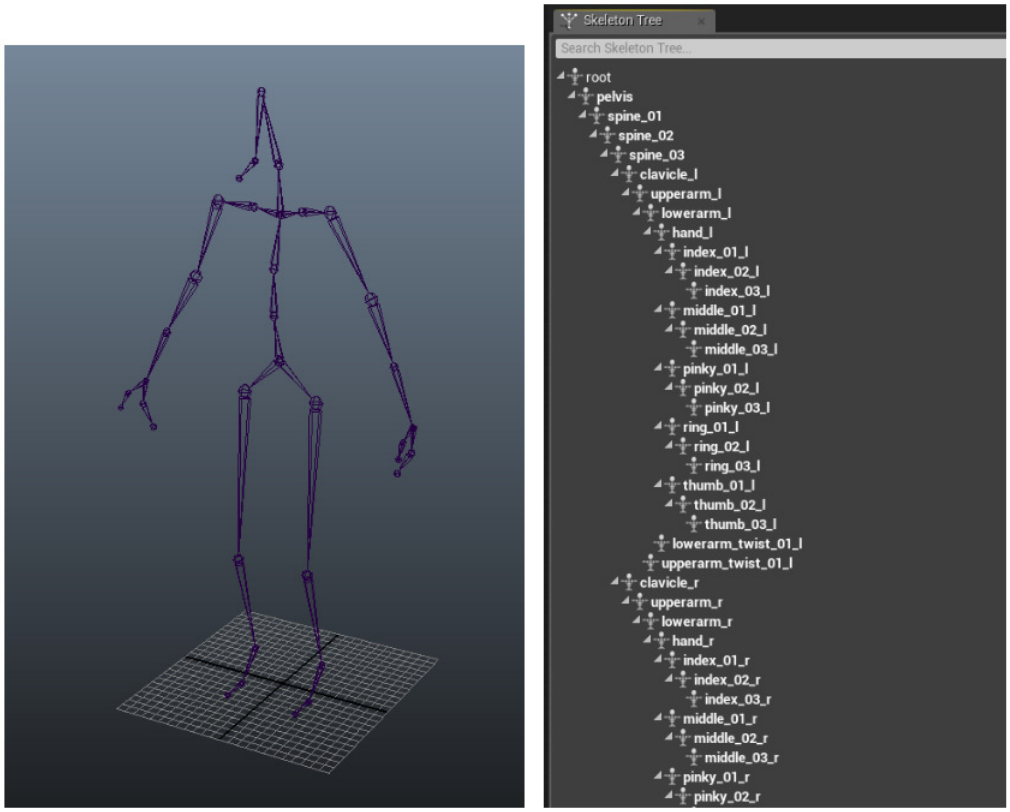
Obr. 1.12: Brush geometry pro vytvoření prostředí (levelu) [7]

Skeletal animation data

Pokud má být objekt animován, je potřeba, aby měl tento model přiřazenou kostru. Tento model se pak stává tzv. skeletal mesh. Samotnému modelu, který je na kostru připojen, se říká *skin*, neboli kůže. Všechny termíny si můžeme přiblížit na modelu člověka, viz obrázek 1.13. Aby bylo možné renderovat skeletal mesh, jsou potřeba tři různé objekty: samotný mesh, hierarchie kostry (viditelná v pravé tabulce obrázku 1.13) a animační data. Hierarchie kostry popisuje názvy kloubů, vztahy mezi jednotlivými částmi kostry a základní pozici kostry, na kterou byl mesh přiřazen. Animační data zase specifikují, jaké kosti mají v čase měnit svou pozici. Objekt skeletal mesh lze exportovat s přímo přiřazenou kostrou v jednom souboru. Pokud však nastane situace, kdy na jeden mesh existuje více druhů koster (třeba různě komplexních), je vhodnější kostry exportovat zvlášť oproti modelu. [4]

1.3 Rešerše herních enginů

V následující kapitole jsou popsány nejvyužívanější open source herní enginy současnosti. Kromě níže zmíněných jich existuje na trhu mnoho, avšak jedním z hlavních



Obr. 1.13: Model kostry a jednotlivých komponent [8]

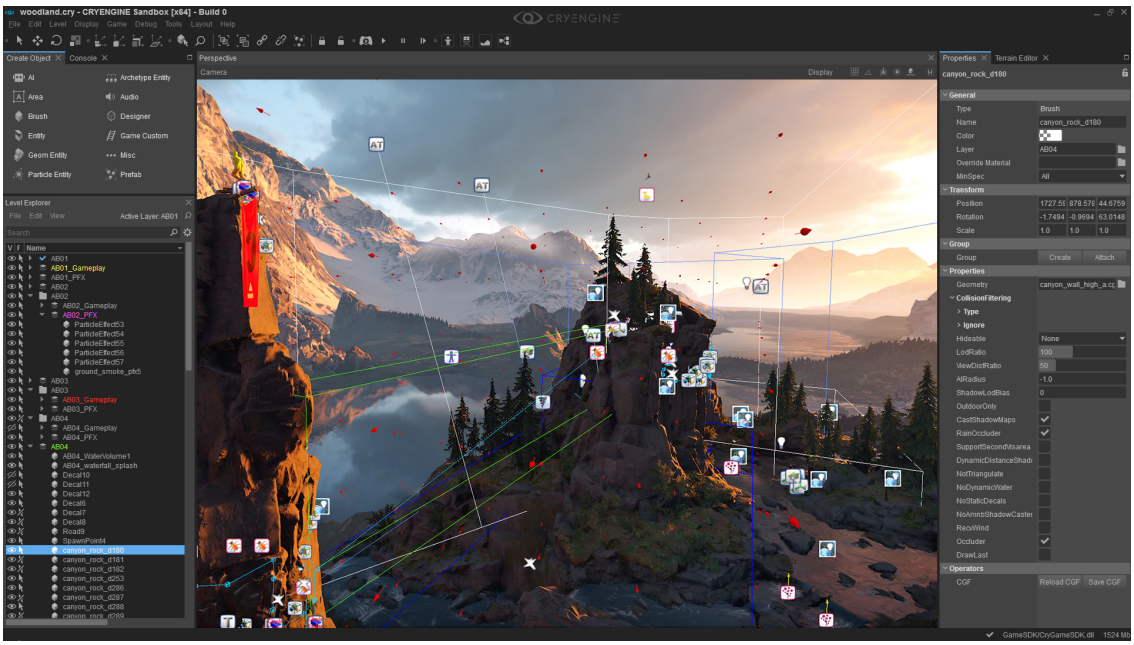
faktorů, podle kterých je provedena rešerše, je dostupnost assetů a podpora vývojářů. Herní aplikace vytvářejí týmy o několika desítkách vývojářů několik let. Tento vývoj není primárním cílem diplomové práce, musí se tedy využít volně dostupné assety (například modely aut, mapy, textury...) a soustředit se primárně na sběr dat o uživateli a vytvoření dobrého prostředí a herní mechaniky.

Požadavky na herní engine jsou následující:

- Vývojové prostředí (editor)
- Možnost programování/skriptování v programovacím jazyce
- Integrované rozhraní pro výpočet fyziky/kolizí
- Rozhraní pro editaci modelů objektů
- Pokročilé grafické možnosti - vytváříme simulátor řízení a chceme, aby se co nejvíce přibližoval realitě
- Dostupnost assetů třetích stran
- Podpora od výrobce

1.3.1 CryENGINE V

CryENGINE V je poslední řada herních enginů od společnosti Crytek. Jeho první verze vešla na trh v roce 2004 při vydání hry Far Cry. V dnešní době se jedná o jeden z graficky nejpokročilejších herních enginů, nabízející kompletní využití nového grafického hardwaru a technologií. Předností je CRYENGINE Sandbox editor - nástroj, který umožňuje vytvářet, editovat a okamžitě vyzkoušet herní aplikaci (WYSIWYP - what you see is what you play). Engine v reálném čase řeší konverzi a optimalizaci assetů. Od roku 2016 je jeho plný zdrojový kód dostupný a engine přešel do režimu "pay what you want". Pro studenty je tedy volně dostupný bez nutnosti poplatků. Velkou nevýhodou, oproti jiným enginům, je malé množství assetů, na které jsem přišel při prozkoumání databáze vývojářů a také nízká rozšířenost v komunitě herních vývojářů. Také dokumentace k enginu není zcela kompletní a klade tak velké nároky na znalosti vývojáře v problematice herního vývoje. Programování v enginu je realizováno v programovacím jazyce C++, v novější verzi i C#, případně pomocí technologie *Flow Graph*, kterou si lze představit jako propojené funkční bloky se vstupy a výstupy. Je podobné programování pomocí Blueprintů v Unreal Engine. Vzhledem ke zmíněným nedostatkům jsem shledal CryENGINE nevyhovujícím pro realizaci simulátoru. [10]

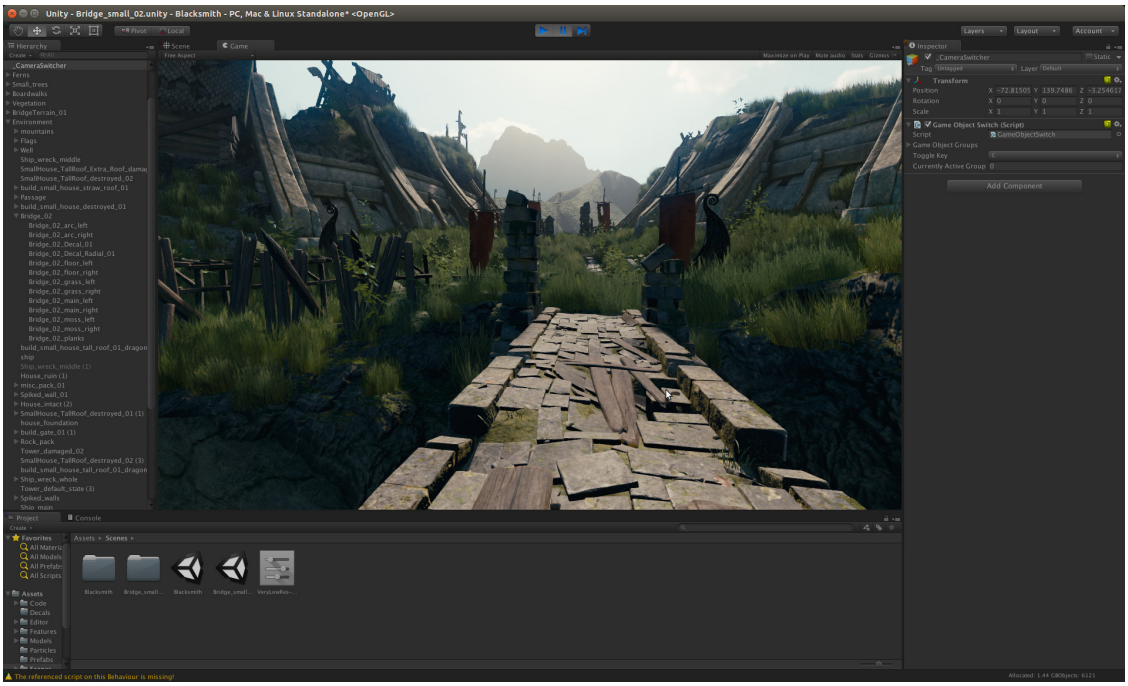


Obr. 1.14: Vývojové prostředí CryENGINE V [9]

1.3.2 Unity

Engine byl vydán v roce 2005, původně určen k vytváření aplikací pro platformy od společnosti Apple. V průběhu let se však rozrostl na jeden z nejvyužívanějších herních enginů současnosti. Dle stránek výrobce je přes polovina her na trhu vyvíjena právě pomocí Unity. Mezi hlavní přednosti Unity patří:

- Všestranný editor podporující vývoj na Windows a Mac
- Podpora 2D a 3D aplikací
- Programování v jazyce C#
- Implementace AI - unity obsahuje navigační systém pro NPC
- User interface - modul pro rychlou tvorbu uživatelského rozhraní pro hráče
- Podpora fyzikálních enginů - Box2D, NVIDIA PhysX
- Vlastní nástroje - rozšíření editoru vlastními nástroji nebo nástroji třetích stran, které jsou dostupné v asset databázi Unity
- Podpora komunity - propojená komunita s dostatečným množstvím návodů, dokumentace a informací přímo ve vývojovém prostředí [11]



Obr. 1.15: Vývojové prostředí Unity [12]

Unity se jeví jako ideální kandidát pro tvorbu simulátoru řízení. Nakonec jsem však tento simulátor pro realizaci nevybral z důvodu existence jiných simulátorů vozidel, jejichž assety a řešení mohu využít ve své diplomové práci. Tyto simulátory

(kupříkladu Carla, AirSim, do budoucna NVIDIA Drive) jsou určeny pro simulaci autonomního řízení a jsou vytvořeny na platformě Unreal Engine.

1.3.3 Unreal engine

Unreal Engine od výrobce Epic Games je výborným prostředníkem mezi CryENGINE, který nabízí pokročilé grafické možnosti, a Unity, který zase nabízí velkou komunitní základnu a podporu od výrobců. Unreal Engine kombinuje skvělé grafické zpracování a dobrou vývojářskou základnu a společně s dostupnými assety, které jsou přímo určené pro aplikace využívající vozidla, nám poskytuje perfektní prostředí pro vývoj simulátoru řízení. Důležitý je také fakt, že engine má implementovanou PhysX Vehicle SDK, která obstarává chování vozidla a jeho programování v prostředí editoru.

1.4 Vývojové prostředí Unreal engine

Následující kapitola je věnována detailnímu popisu práce ve vývojovém prostředí Unreal Engine. Obsahuje rozbor ovládacích prvků, popis různých druhů programování a nástrojů, které engine nabízí.

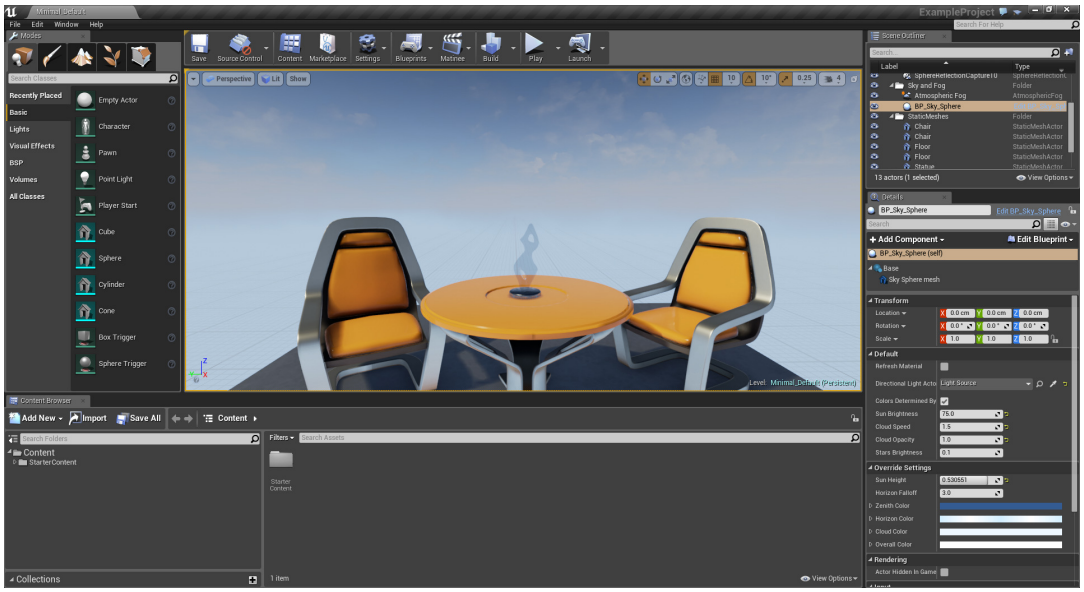
1.4.1 Editor

Základním prvkem Unreal Engine je editor, ve kterém dochází k vývoji aplikace. Tento popis vychází z použitých prostředků při vývoji aplikace, která je součástí této diplomové práce - *Car Driving Simulator*. Unreal Engine je komplexní software, který obsahuje mnoho nástrojů, například na editaci mesh nebo skeletonů, vytváření animací, kolizních modelů apod. V této práci je nelze popsat všechny, tak se zaměřuji pouze na ty, které jsou důležité a které využívám.

Editor má několik částí:

Hlavní panel

Umožňuje uživateli ukládat projekt, zálohovat jej pomocí source control, přistupovat do databáze objektů a souborů v projektu či na marketplace (online obchod) s assety. Lze zde také měnit celkové nastavení projektu, jako je například název projektu, defaultní mód spuštění, nastavení vstupních zařízení apod. Dále zde můžeme



Obr. 1.16: Vývojové prostředí Unreal Engine 4 [13]

vybírat možnosti kompilace projektu, kvalitu renderování (například osvětlení) a případně spouštět/exportovat projekt.

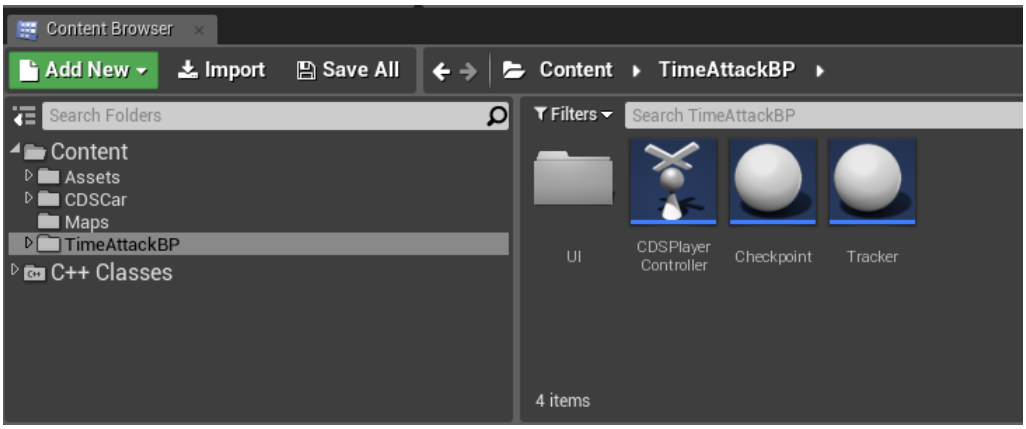
Obr. 1.17: Hlavní panel UE4

Content browser

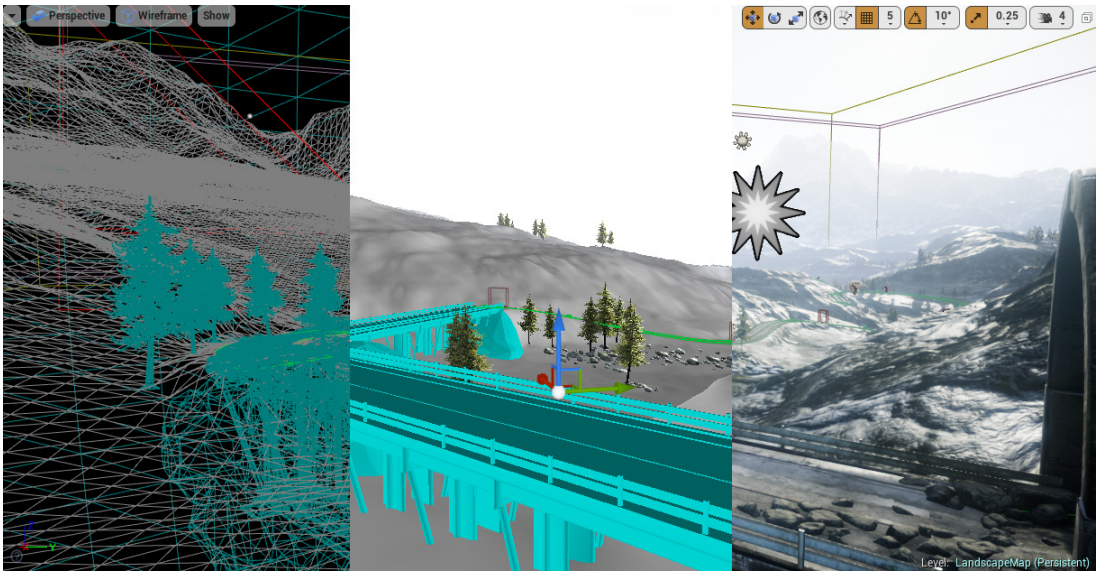
Tato část zobrazuje projektovou složku včetně všech souborů a funkcí, se kterými je možné pracovat a aplikovat je v editoru. Umožňuje přístup k assetům, objektům, souborům vycházejících z DCC (mesh, skeleton...) nebo k blueprintům či vytvořeným C++ třídám, které lze přiřazovat objektům v editoru.

World editor

Zobrazuje vývojáři celé prostředí herní aplikace, do které je možné vkládat objekty a manipulovat s nimi - měnit jejich pozici, rotaci, vlastnosti apod. Pohyb vývojáře v editoru se ovládá pomocí kláves W,A,S,D a myší. Umožňuje několik druhů zobrazení a pohledů (zobrazení z různé perspektivy, mesh geometrie, kolizních zón objektů apod.). Na obrázku 1.19.



Obr. 1.18: Content browser UE4



Obr. 1.19: World editor UE4, pohledy zleva: mesh (geometry), kolizní zóny, celkový pohled

World outliner

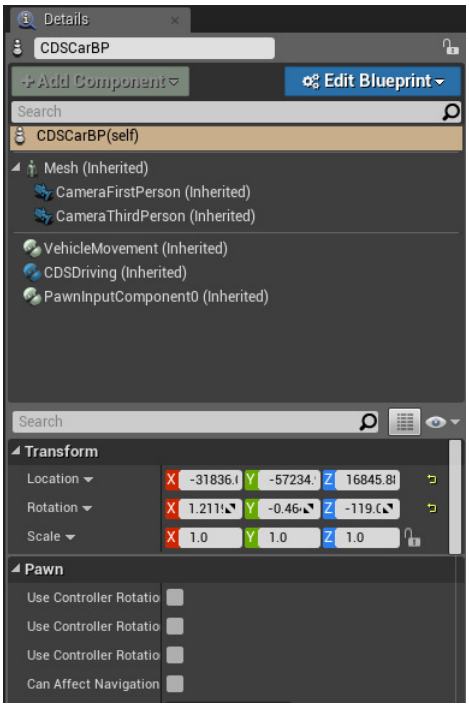
Sekce world outliner obsahuje soupis všech objektů, které se nachází v editovaném prostředí. Můžeme je shlukovat do složek dle typu objektů, případně rozkliknout a editovat.

Modes

Obsahuje databázi základních objektů (static mesh, brush...), které lze vkládat do world editoru. Také umožňuje upravovat povrch prostředí, vytvářet mapy, nanášet textury nebo měnit tvary objektů ve world editoru.

Details

Sekce *details* obsahuje detailní informace a parametry objektu. Kupříkladu objekt hráče (pawn) má parametry dané jeho pozicí a natočením ve světě a také dědí některé C++ třídy nebo komponenty (VehicleMovement, Mesh Component, vytvořené funkce apod.). Odtud je možné k těmto komponentům přistupovat a editovat je. Kupříkladu VehicleMovement je komponenta, zajišťující pohyb objektu (vozidla) pomocí PhysX Vehicle SDK.



Obr. 1.20: UE4 editor - sekce Details

1.4.2 Programování v Unreal Engine

Vytvářet objekty (a tím pak celou aplikaci) lze v Unreal Engine dvěma způsoby - pomocí programovacího jazyka C++ nebo skrze vizuální skriptování - Blueprint programming. Programátoři C++ pracují ve svém vývojovém prostředí (například Visual Studio, které má přímo integrovanou podporu Unreal Engine), zatímco pro využití Blueprintů má Unreal Engine integrovaný Blueprint editor. Oba tyto přístupy jsou navzájem provázány a ve skutečnosti je nanejvýš vhodné, aby byly využívány pospolu - ideální propojení je takové, kdy C++ programátoři vytváří funkční bloky a herní designéři tyto bloky využívají k vytvoření herního prostředí.

Programování v C++

Prvním krokem k vytvoření vlastních funkcí v C++ pro Unreal Engine je výběr rodičovské třídy (parent class). Při výběru této třídy dojde k vytvoření .cpp a .h souborů s předepsaným vzorem. Typy rodičovských tříd jsou následující:

- Žádná - prázdná C++ třída s defaultním konstruktorem a destruktorem.
- Actor - základní třída popisující objekt, který může být vložen do herního světa. Může defaultně obsahovat set komponent, který určuje například způsob pohybu, renderování aj.
- Actor Component - Actor komponenta, kterou může používat jakýkoliv jiný Actor a udává mu určitý set vlastností/funkcí
- Pawn - druh Actor třídy, který může být ovládán a dostávat vstupní data od uživatele či umělé inteligence.
- Character - druh Pawn třídy, který má svůj mesh, kolizi a pohyb. Reprezentují hráčskou postavu a umožňují uživateli interagovat s prostředím a herními prvky či objekty.
- Game Mode Base - třída zodpovídající za pravidla, nastavení a cíle hry.
- Player Controller - komponenta Pawn zastřešující jeho ovládání.
- ... a obrovské množství dalších tříd. [14]

Jakmile doje k vytvoření nové třídy, může programátor vytvářet svůj objekt aplikace. Výše zmíněné rodičovské třídy slouží k ulehčení práce a jako template pro vytvoření dané funkce. Aby však mohl programátor využívat všech knihoven a funkcí, které mu Unreal Engine nabízí, je potřeba se seznámit s jejich dostupností a hlavně syntaxí, jelikož Unreal Engine používá svou vlastní pozměněnou syntaxi jazyka C++. Pravidla syntaxe jsou následující:

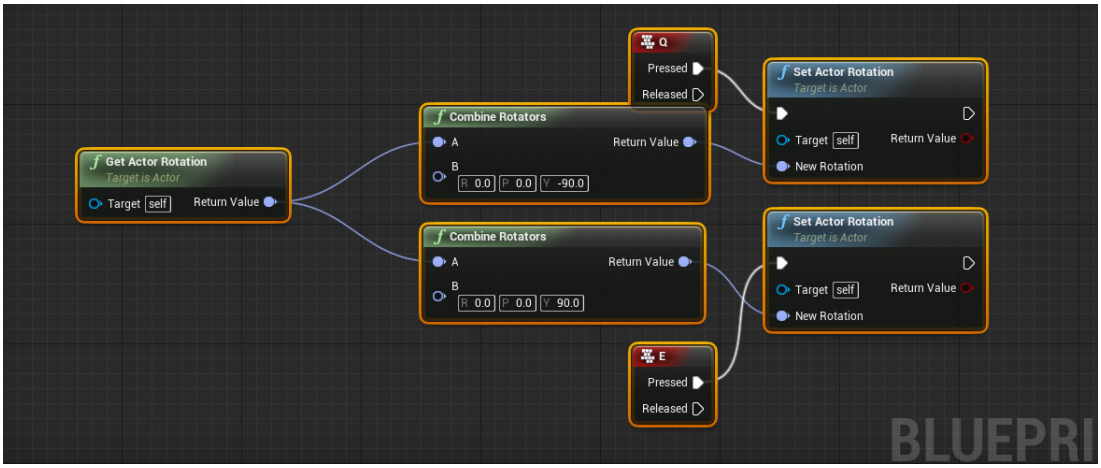
- Třídy odvozené od rodičovské třídy Actor mají předponu "A", například AController
- Třídy odvozené od rodičovské třídy Object mají předponu "U", například UComponent
- Enum datový typ má předponu "E", např. EFortificationType
- Třídy rozhraní mají předponu "I", např. IAbilitySystemInterface
- Šablony užívají předponu "T", např. TArray
- Třídy odvozené od rodičovské třídy SWidget mají předponu "S", např. SButton
- Vše ostatní používá předponu "F", např. FVector [14]

Podobně jako jsou pravidla pro třídy, také proměnné mají své odvozené názvy -

int8(16,32,64), uint8(16,32,64), FString, FText, TCHAR aj.

Programování pomocí Blueprintů

Celému systému se přezdívá *Blueprints Visual Scripting system*. Jedná se o skriptovací systém na úrovni herních mechanismů architektury herního enginu (viz kapitola 1.2.3). Tento systém definuje objektově orientované třídy nebo objekty v herním enginu. Umožňuje herním designerům využívat celou škálu nástrojů, které byly v minulosti dostupné právě pouze pro programátory. Odpadává tak nutná znalost přímého kódování a designéři se tak mohou soustředit na tvoření herních mechanik a logiky. [15]



Obr. 1.21: UE4 editor - příklad použití Blueprintů [16]

Systém je rozdělený na moduly (node), eventy, funkce a proměnné, které se vizuálně propojují. Existují následující typy blueprintů:

- Level Blueprint - globální blueprint zastřešující celou instanci (daný level) aplikace.
- Blueprint Class - paralelní k třídám vytvořeným v C++ a popsaných v kapitole 1.3.2, mají stejné rodičovské třídy (Actor, Pawn, Character ...).
- Data-Only Blueprint.
- Blueprint Interface - jedná se o set funkcí (bez implementace), který můžeme přiřadit jiným blueprintům a umožnit jim tak vzájemnou komunikaci a posílání dat.
- Blueprint Macro Library - podobně jako makra v programování reprezentují části kódu, které jsou zapouzdřené v modulu a můžou se tak využít v jiných

blueprintech. [15]

Příklad použití blueprintů je uveden na obrázku 1.21. Jedná se o Blueprint Class, konkrétně odvozená od rodičovské třídy Actor Component. Umožňuje rotaci objektu zmáčknutím klávesy Q nebo E. Modul *Get Actor Rotation* získá aktuální pozici Actor objektu, jehož je blueprint součástí jako komponenta (Target: self). Návratovou hodnotou je rotace objektu, která je pomocí modulů *Combine Rotators* pozměněna o kladnou či zápornou hodnotu v jedné ose. Moduly *Set Actor Rotation* nastaví při stisknutí daných kláves novou hodnotu natočení objektu. [15]

1.4.3 Fyzikální model vozidla

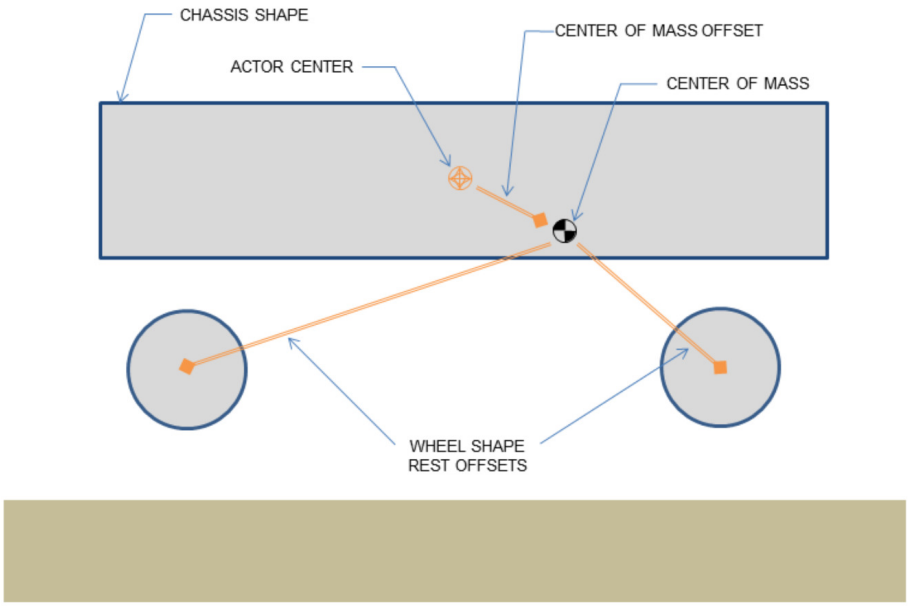
Za chování vozidla a jeho fyzikální model odpovídá PhysX Vehicle SDK od společnosti NVIDIA. Nutno podotknout, že se jedná o knihovnu a následující sekce popisuje fungování této knihovny pouze z pohledu dokumentace. Vozidlo je v jádru soubor několika odpružených hmot (viz obrázek 1.23), které zastupují spojení mezi vozidlem a koly (zavěšení). Každá odpružená hmota má svá asociovaná kola a informace o pneumatikách. Tento soubor je pak celkově reprezentován "tuhým"modelem vozidla, jehož hmotnost, těžiště či moment setrvačnosti odpovídají rozložení odpružených hmot. Vzájemný vztah mezi tuhým modelem vozidla a odpruženými hmotami je pak reprezentován pomocí rovnic[21]:

$$M = M_1 + M_2 \tag{1.1}$$

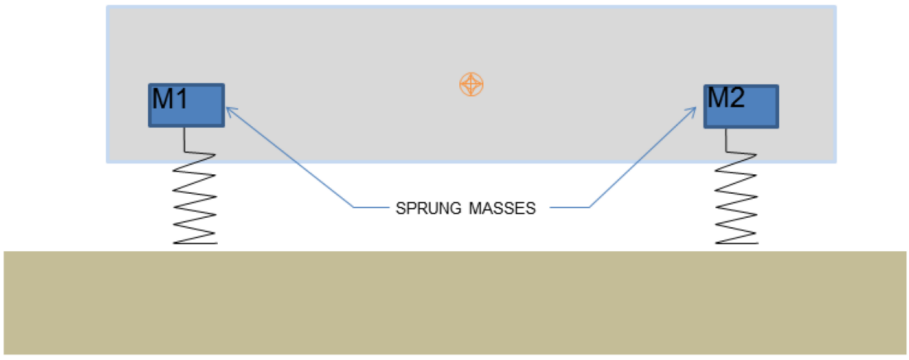
$$X_o = (M_1 * x_1 + M_2 * x_2)/(M_1 + M_2) \tag{1.2}$$

Přičemž M_1 a M_2 značí odpružené hmoty, x_1 a x_2 jsou souřadnice polohy odpružených hmot a X_o je posunutí (offset) těžiště tuhého tělesa (modelu) vozidla. [21]

PhysX Vehicle SDK obsahuje několik funkcí, z nichž ta nejhlavnější je *update* - díky modelu s odpruženými hmotami vypočítává síly, které na ně působí v závislosti na podložce. Následně souhrn těchto sil aplikuje na tuhé tělo vozidla ve formě modifikované a úhlové rychlosti. Interakce mezi tělem vozidla s ostatními objekty a aktualizace pozice vozidla je pak řešena skrze celkovou knihovnu PhysX SDK. Síla z každé pružiny je vypočtena a přidána do celkové síly, která se aplikuje na tuhé tělo vozidla. Také určuje množství síly, která působí na kola a zpomaluje tak jejich pohyb. Kromě toho je také síla z kol ovlivněna několika dalšími faktory - jejich úhel natočení, tření, rychlost rotace kol, hybnost vozidla aj. [21]

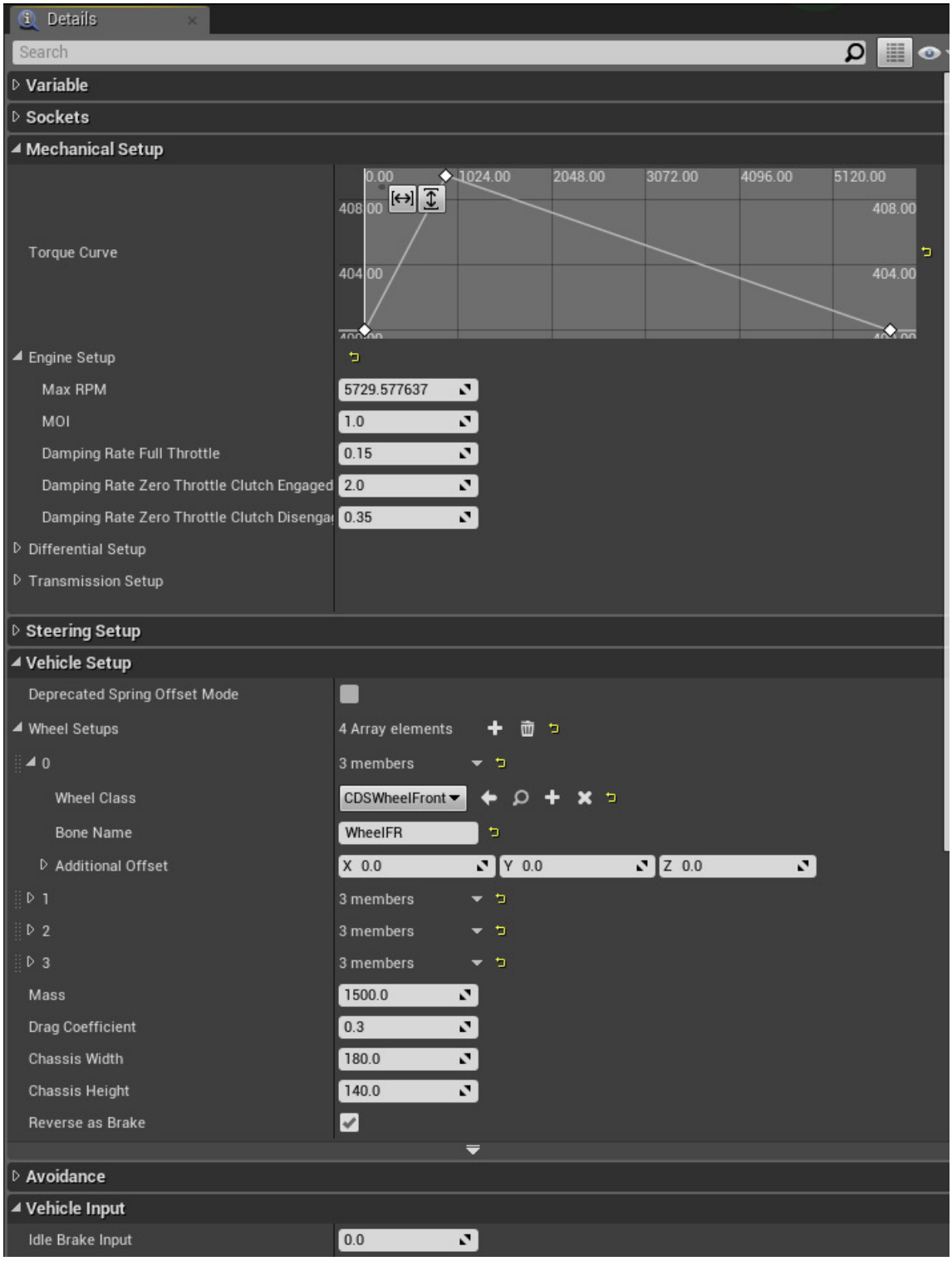


Obr. 1.22: Model vozidla PhysX Vehicle SDK [21]



Obr. 1.23: Odpružené hmoty modelu PhysX Vehicle SDK [21]

Dále také PhysX Vehicle SDK podporuje různé modely řízení - základem je torzní spojka, která mezi sebou spojuje jednotlivá kola a motor. Na jedné straně spojky je motor, který má výkon daný podle plynového pedálu. Na druhé straně máme řadící systém, diferenciál a jednotlivá kola. Otáčky motoru jsou převáděny skrze spojku, řadící poměr a diferenciál na otáčky kol. V obrázku 1.24 lze pozorovat nastavitelné parametry vozidla, které pak ovlivňují jeho chování. [21]



Obr. 1.24: Nastavení chování vozidla třídy WheeledVehicle

2 REALIZACE SIMULÁTORU

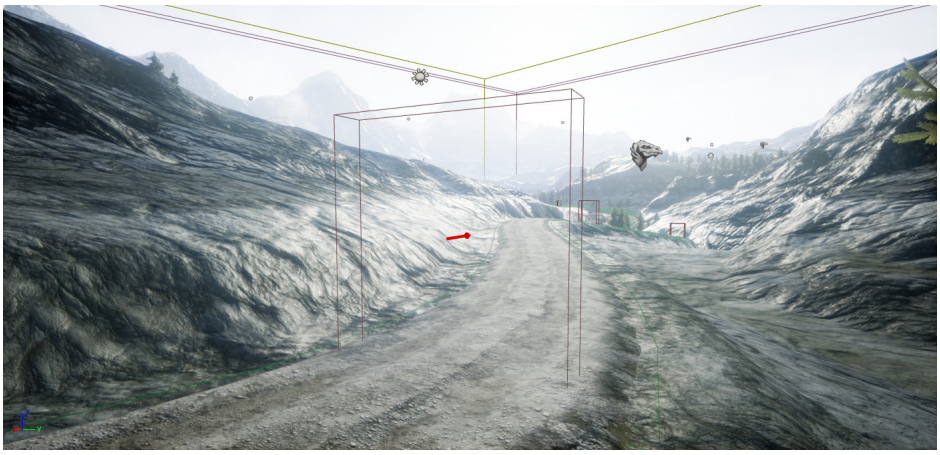
Cílem praktické části diplomové práce je vytvoření aplikace simulátoru řízení vozidla. Následně provést test vytvořené aplikace a analyzovat výsledná data. Požadavky jsou určeny dle teoretického rozboru.

2.1 Aplikace Car Driving Simulator

Vývoj simulátoru byl rozdělen na dvě části, přičemž v první fázi došlo k vytvoření testovací aplikace CDS Showcase a následně v druhé fázi k vytvoření samotného simulátoru Car Driving Simulator.

2.1.1 CDS Showcase

Smyslem aplikace CDS Showcase bylo v rámci semestrálního projektu naučit se základům vývoje v Unreal Engine a otestovat, zda-li tento herní engine vyhovuje požadavkům na vytvoření simulátoru. CDS Showcase byla vytvořena v Unreal Engine v4.18. Základem je volně dostupná mapa z dílny Epic Games (vývojář Unreal Engine) horského pohoří. Model auta je součástí simulátoru AirSim od společnosti Microsoft. Jedná se o simulátor autonomního řízení pro auta a drony. V aplikaci byl vytvořen nový blueprint typu *WheeledVehicle* pro otestování rozpohybování vozidla v Unreal Enginu. Na obrázku 2.2 lze pozorovat model auta bez textur a materiálů. Uživatel má možnost vybrat si ze dvou pohledů. Pohled z první osoby, který vidíme na obrázku 2.2 ve vrchní části. Ten simuluje pohled řidiče auta. Druhý pohled z třetí osoby je umístěn za autem a ukazuje model auta v prostředí.



Obr. 2.1: CDS Showcase - checkpoint

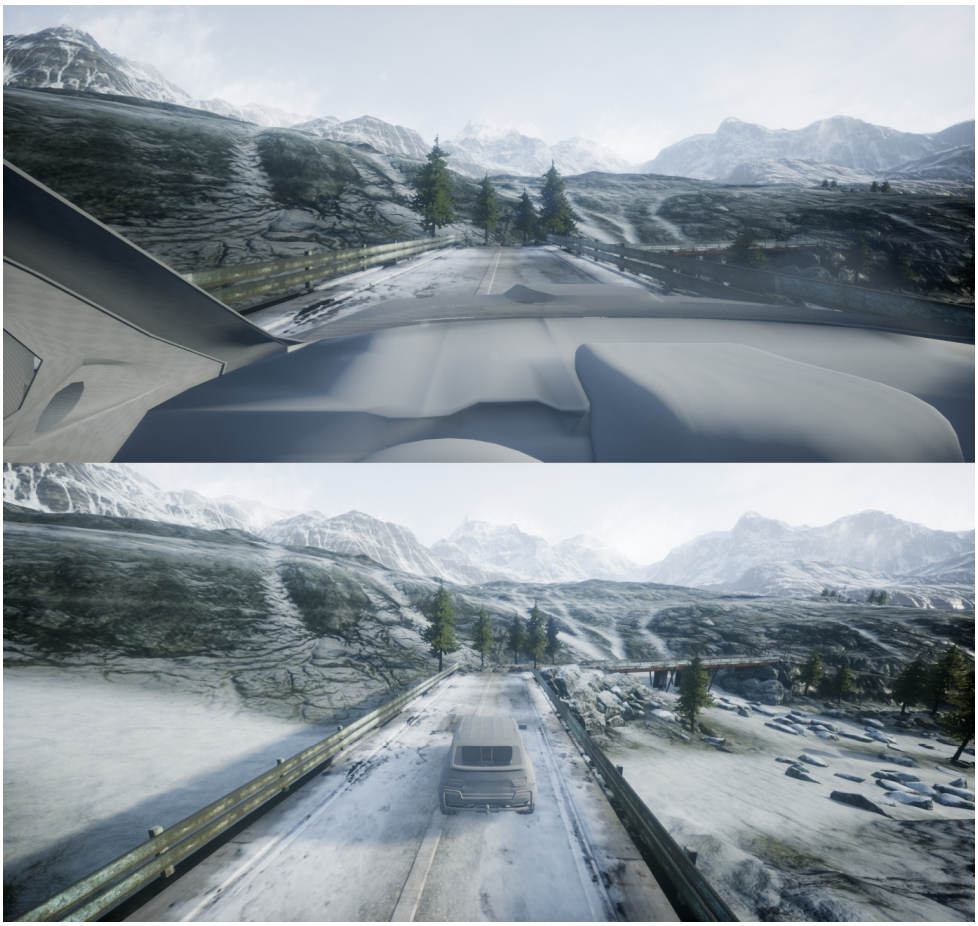
Time attack scenář

Kromě otestování vozidla a jeho pohybu byl také v aplikaci CDS Showcase vytvořen herní mód, jehož smyslem je získávat údaje o časech, ve kterých hráč projel s vozidlem danou bránu (zobrazena na obrázku 2.1) na trase vyznačené na mapě. Čas se začíná měřit ihned po spuštění aplikace a při každém projetí bránou se hráči ukáže aktuální čas a dojde k jeho zápisu do .csv souboru. Jakmile hráč projede celou trasu a skončí u posledního checkpointu, hra se restartuje a danou trasu může projíždět znovu. Celkovým výsledkem aplikace je čas, za který hráč projede danou trasu. Jedná se o základní typ scénáře, kterému se přezdívá *time attack*. Na tomto scénáři byla úspěšně otestována možnost aplikace v Unreal Enginu získavat a ukládat data o uživateli.

2.1.2 Koncepce simulátoru

Aby bylo možné přijímat data o uživateli a následně je zpracovávat, je nutné, aby byl k tomu simulátor uzpůsoben. Na obrázku 2.3 jsou vyobrazeny nejdůležitější části simulátoru:

- Vstupní/výstupní zařízení - pro tento simulátor se jedná o volant, pedály, řadící páku, monitor a případně o klávesnici a myš. Úhel natočení volantu, míra stlačení pedálů, rychlost řazení - to všechno jsou data, která se mohou o uživateli zpracovávat. Krom toho lze také z těchto zařízení sledovat rychlost odezvy uživatele na podněty. Zpětnou vazbu dostává uživatel vizuální pomocí monitoru.
- Unreal Engine - herní engine, který poskytuje jádro pro aplikaci simulátoru.

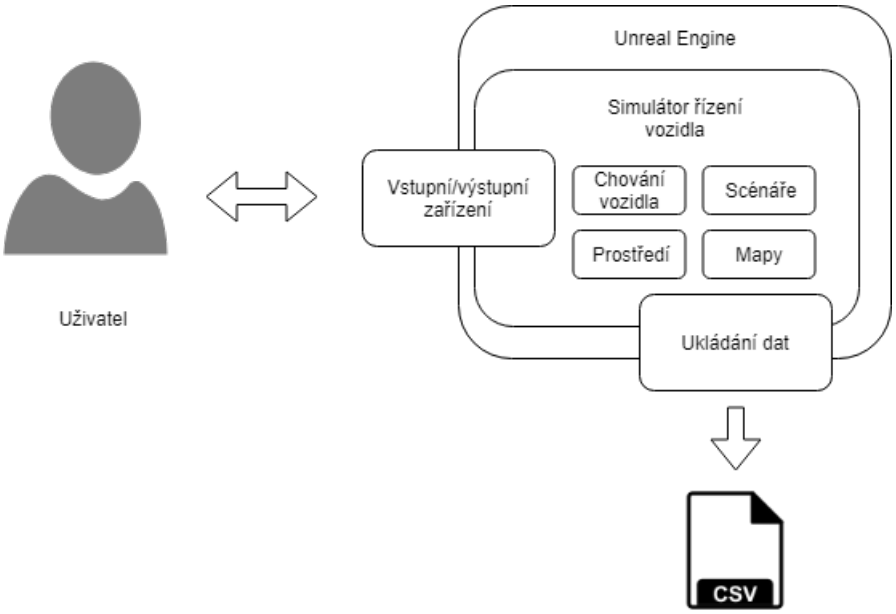


Obr. 2.2: CDS Showcase - kamery

- Simulátor řízení vozidla - samotná aplikace, která poskytuje zpracování dat, různé druhy scénářů a map, chování vozidla (hlavně jeho fyzika) a okolní prostředí.
- Ukládání dat - výstupní funkce, která má za úkol ukládat zpracovaná data simulátoru do souboru.

Aby byl učel aplikace simulátoru naplněn, je zapotřebí soustředit se na několik základních aspektů aplikace, které jsou následně v praktické části popsány:

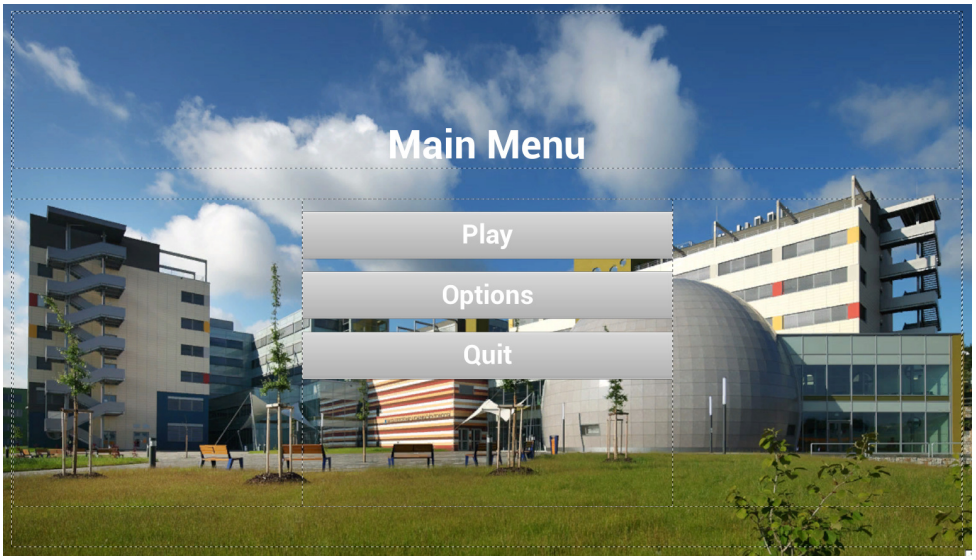
- Vytvoření reálného prostředí
- Implementace modelu auta včetně jeho ovládání (propojení s HW)
- Herní mechanismy - scénáře
- Sběr dat o uživateli



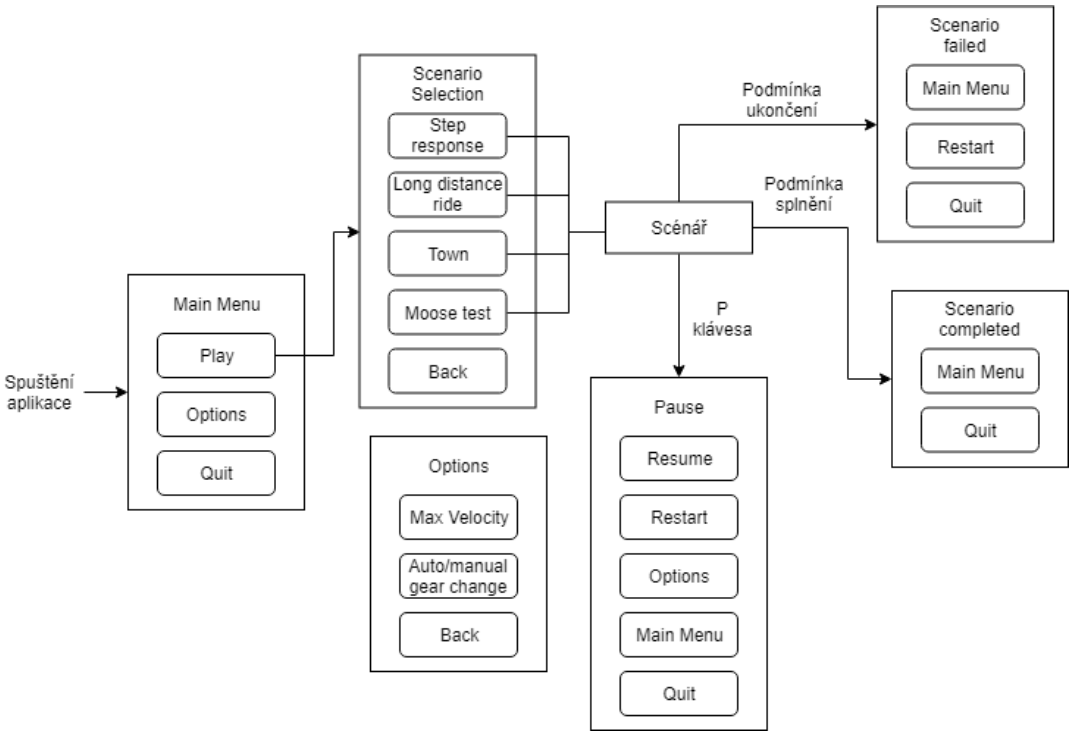
Obr. 2.3: Car Driving Simulator - koncept

2.1.3 Navigace aplikace - menu

Diagram navigace uživatele v aplikaci je vyobrazen na obrázku 2.5. Jakmile je aplikace spuštěna, uživatel se ocitne v hlavním menu (obrázek 2.4), ve kterém si může nastavit parametry vozidla v sekci *Options*, jakými jsou třeba maximální povolená rychlost nebo způsob řazení (automat/manuál).



Obr. 2.4: Hlavní menu Car Driving Simulator

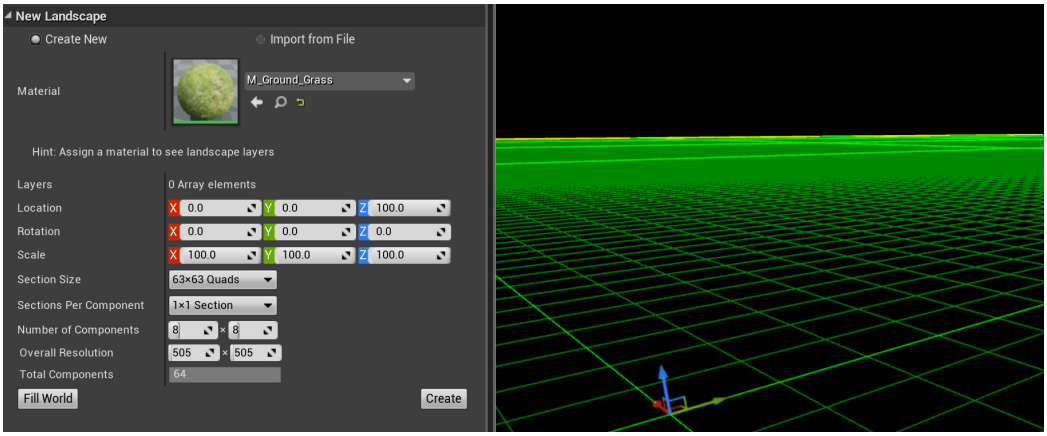


Obr. 2.5: Navigace v aplikaci Car Driving Simulator

Stiskem tlačítka *Play* si uživatel vybírá, jaký scénář chce spustit. Jakmile je vybráno, vybere uživatel soubor, do kterého chce ukládat data aplikace a následně vykonává daný scénář. Během jízdy může kdykoliv simulátor pozastavit stiskem tlačítka P, změnit nastavení, restartovat scénář či se vrátit do hlavního menu. Pokud dojde ke splnění podmínky správného ukončení scénáře, scénář je hotov. Pokud však dojde k porušení některých z podmínek, scénář je přerušen a uživatel je nucen jej restartovat, případně si zvolit jiný.

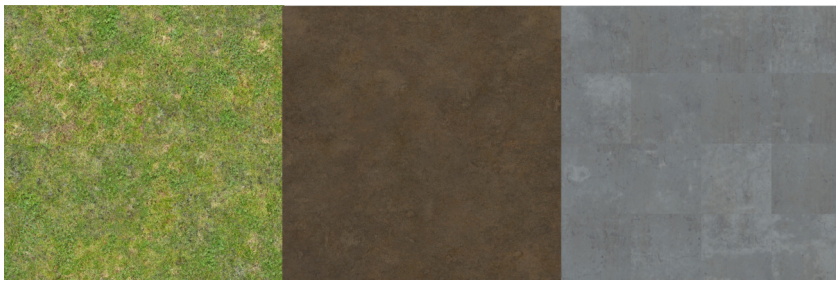
2.1.4 Vytvoření prostředí

Proces tvorby mapy vyžaduje několik po sobě jdoucích kroků. Vytvoří se nová komponenta *level*, která je však zpočátku úplně prázdná. Nemáme v prostředí žádné světlo či plochu. Prvním krokem je tedy vytvořit povrch. Ten se vytváří pomocí nástroje v Unreal Enginu, ve kterém si můžeme zvolit rozdělení povrchu na části. Čím více částí, tím detailněji může být povrch modelován, avšak zvyšuje se náročnost jeho vykreslování a tím pádem nemůžeme vytvářet rozlehlé povrchy. Ukázka nastavení tohoto povrchu je na obrázku 2.6. Jakmile dojde k vytvoření takového povrchu, je možné jej pomocí různých nástrojů modelovat a vytvářet tak hory, kopce, doliny apod.



Obr. 2.6: Vytvoření povrchu

Jakmile máme vytvořený povrch, je potřeba na něj nanést vrstvy. Tyto vrstvy určují, jakou texturu bude daná část povrchu vykreslovat. V Car Driving Simulator jsou použity 3 typy povrchů: tráva, hlína a beton (vyobrazeny na obrázku 2.7). Rozdílné vrstvy pak lze pozorovat na obrázku 2.10. Na každou z vrstev je možno navázat různé objekty, jak je tomu v případě vykreslování vrstvy trávy. Tato vrstva má na sobě navázány 3D modely trávy a květin, takže kdekoliv na povrchu je tato vrstva, dochází k vykreslování daných objektů s danou hustotou, viz obrázek 2.8.



Obr. 2.7: Textury vrstev povrchu. Zleva: grass, mud, concrete_tiles

Neméně důležitou součástí prostředí je samozřejmě osvětlení. To je zajištěno čtyřmi různými typy objektů, které musí být ve virtuálním světě přítomny:

- Atmospheric Fog - simuluje rozmístění světla na obloze, vytváří tak mlhu v pozadí dle barvy directional light
- Directional light - zastupuje funkci Slunce jako zdroje světla
- Sky light - rozprostírá světlo vně i uvnitř objektů

- BP_Sky_Sphere - kombinuje všechny prvky dohromady a vytváří oblohu závislou na denní době. Přidává do světa mraky a hvězdnou oblohu.

Obr. 2.8: Detail vykreslování trávy

Jakmile je vytvořen tento základ, do prostředí je možno přidávat objekty. Pro vytvoření obrazu přírody jsou do prostředí přidány modely stromů pomocí nástroje *foliage*. Na pozadí jsou vykreslovány objekty hor, aby mapa působila pro uživatele věrohodně. V aplikaci jsou vytvořeny dvě základní mapy, které jsou poté upravovány pro potřeby scénářů.

Dálnice

V této mapě je vytvořeno horské prostředí, skrze které vede dlouhá dálnice. Uživatel se svým autem může jezdit pouze v omezeném pravém pruhu této dálnice, která je přes 8 kilometrů dlouhá. Mapa tak nabízí využití ve scénářích, které vyžadují dlouhou jízdu - scénáře *step response* či *long distance ride*.

Město

Tato mapa simuluje projížďku městským prostředím. Nabízí spoustu delších i kratších cest a zatáček. Oproti mapě s dálnicí, na které se spíše sledují dlouhodobé jízdy s minimálními změnami, na této městské mapě je uživatel nucen být neustále ve

Obr. 2.9: Mapa dálnice

střehu a manipulovat s vozidlem tak, aby byla jízda plynulá - neustálé řazení, zvyšování/snižování rychlosti a točení volantem.

Obr. 2.10: Městská mapa

2.1.5 Implementace modelu auta

Druhou důležitou částí je implementace modelu auta, jeho chování v simulátoru a ovládání. Cílem aplikace je, aby se vozidlo v rámci dostupných možností chovalo na vozovce reálně, aby byl pohled uživatele zevnitř vozidla a aby bylo možné auto ovládat skrze pedály, řadící páku a volant.

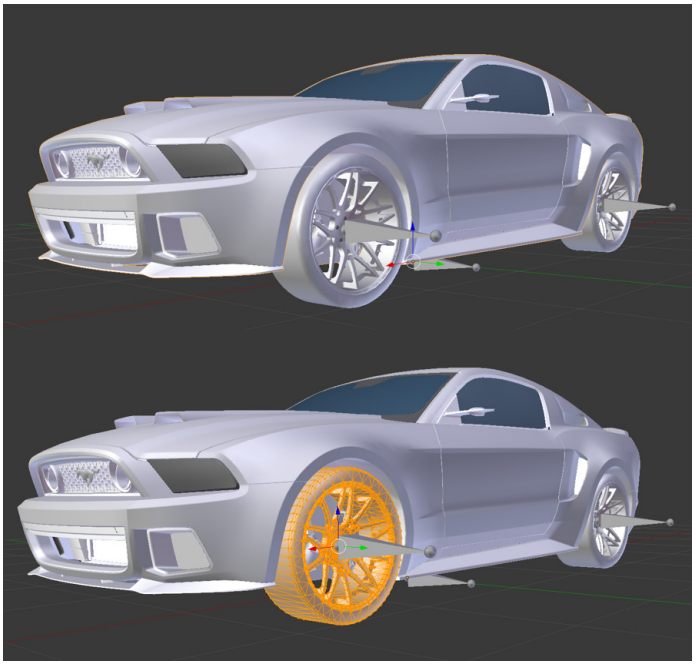
Obr. 2.11: Model vozidla

Pro mesh vozidla je použit model auta volně dostupný s royalty free licencí ze stránek *turbosquid.com*. Tento model již obsahuje vložené textury a materiály (kromě materiálu skla, který je pozměněn v editoru tak, aby odpovídal průhlednému materiálu). Před importováním tohoto modelu do aplikace je však nutné k němu vytvořit prvek armature.

Vytvoření armature

Aby Unreal engine zvládl zpracovat chování modelu v prostředí, je potřeba pro importovaný mesh vytvořit armature. Jedná se o vytvoření kostry modelu, jak je popisováno v kapitole *Skeletal animation data* v teoretickém úvodu. Tato kostra byla vytvořena v prostředí programu Blender. Celkově má auto 5 důležitých částí - kostí, se kterýmu engine manipuluje. Lze je pozorovat na obrázku 2.12 jako trojúhelníkové útvary na modelu vozidla. Je přesně dáno, jak se musí jednotlivé kosti jmenovat a jaká část modelu se na ně musí navázat. Jedná se o:

- Root - základní kost, na kterou je navázán celý model vozidla.
- FL - Front Left - kost, na kterou je navázáno přední levé kolo (část modelu, která je na kost navázána, je zobrazena žlutě na obrázku 2.11).
- FR - Front Right - pravé přední kolo.
- RL - Rear Left - levé zadní kolo.
- RR - Rear Right - pravé zadní kolo.



Obr. 2.12: Model vozidla - armature

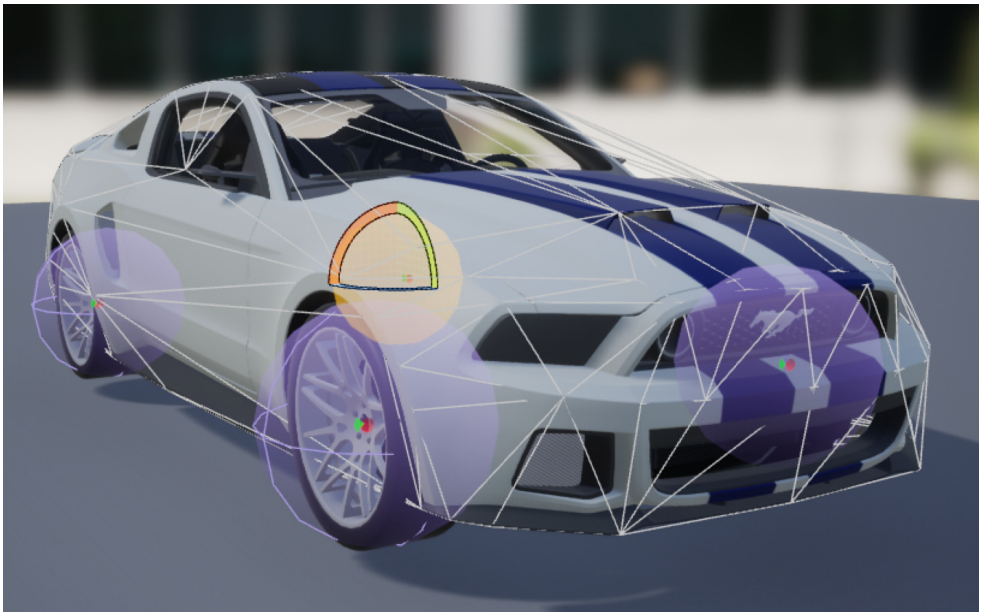
Blueprint vozidla

Jakmile je vytvořen model auta se svou navázanou kostrou, lze jej importovat do enginu. Model se exportuje ve formátu *.fbx* a do enginu je importován jako skeletal mesh - model s kostrou. Aby však vozidlo tvořilo plnohodnotný objekt ovládaný uživatelem, musí se zapouzdřit do blueprintu. Je tedy vytvořen blueprint s názvem *cds_vehicle_BP*, který v sobě zahrnuje:

- Mesh - model auta
- Skeleton - kostra pro animace a fyziku vozidla
- Animation blueprint - zajišťuje otáčení a natočení kol (pomocí funkce Wheel Handler)
- Physics blueprint - odpovědný za chování vozidla v simulátoru, včetně pohybu či kolizí s jinými objekty

- Ovládání auta - namapování vstupů (klávesy, volant, pedály, řadící páka apod.), nastavení parametrů vozidla
- Odesílání hodnot do globálních proměnných - rychlost vozidla, aktuální zařazenou rychlost či úhel natočení volantu
- VehicleMovement komponenta - obstárává chování vozidla v prostředí, vyplývá z PhysX Vehicle SDK

Na obrázku 2.13 vidíme kolizní model vozidla, který byl vytvořen a navázán na jednotlivé kosti (a tím na různé části modelu). Tělo vozidla obsahuje komplexnější model tak, aby reálně odpovídal rozměrům vozidla. Jiné kolizní modely (sféra či ovál) způsobily přesah, takže ke kolizi s objekty docházelo, aniž by se samotný model objektu dotkl. Kola pak obsahují jednoduchý kolizní model sféry. Vozidlo neobsahuje deformační model, tím pádem při kolizi s objekty (například s ohraničením dálnice) nedochází k jeho poškození.



Obr. 2.13: Kolizní model auta

Vstupní zařízení

Aplikaci Car Driving Simulator lze ovládat pomocí volantu a pedálů Logitech G920 (obrázek 2.14). Volant má rozsah otáčení 900 stupňů od dorazu k dorazu a zpětnou odezvu se dvěma motory. Brzdový pedál je nelineární - odpovídá tak více skutečnému pedálu. Součástí je také řadící páka Driving Force Shifter. Každý převodový stupeň se chová jako zmáčknuté tlačítko (takže když je zařazen první převodový stupeň,

tlačítko k tomu přiřazené má na výstupu neustále 1. Mapování vstupů je následující:

- Plynový pedál - zajišťuje akceleraci vozidla a jeho pohyb dopředu.
- Brzdový pedál - brzda.
- Úhel natočení volantu - ovládání natočení kol vozidla do stran
- Spojkový pedál - spojka
- Klávesa P nebo tlačítko Home na volantu (X) - pozastavení aplikace

Obr. 2.14: Volant a pedály Logitech G920 [17]

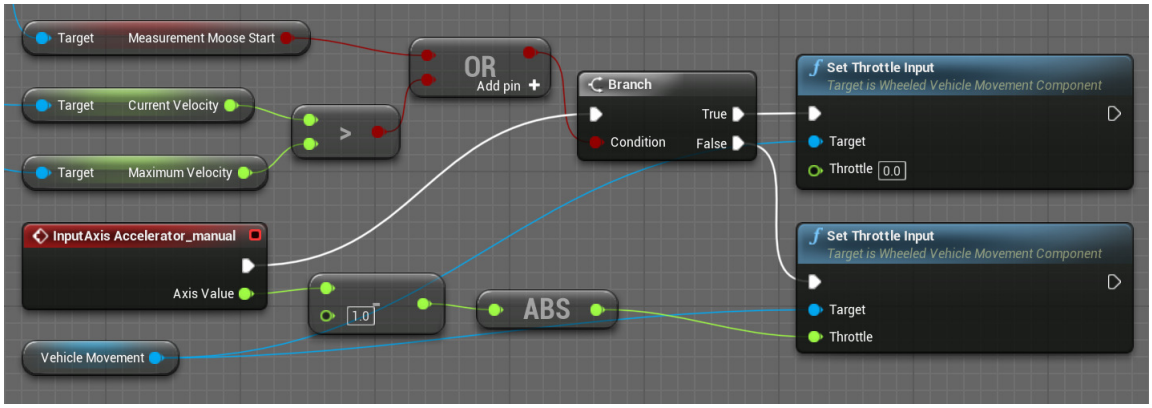
Vstup z pedálů plynu, brzdy a spojky nabývá hodnot 0 až 1 s tím, že 0 odpovídá stlačenému pedálu. Je tedy nutné hodnoty přepočítat, jelikož Vehicle Movement komponenta potřebuje hodnoty inverzní - tak, aby byla 1 na sešlápnutém pedálu. Podobně je tomu u volantu, který má na vstupu hodnoy 0 až 1, přičemž 0,5 odpovídá volantu v prostřední (neutrální) poloze. Vehicle Movement komponenta vyžaduje, aby vstup volantu byl v rozsahu od -1 do 1, opět je tedy nutný přepočet na správný interval. Na obrázku 2.15 lze vidět způsob tohoto mapování a přepočtu pro pedál plynu. Je zde také řešena podmínka spuštění měření a maximální rychlosti vozidla, kterou uživatel nastavuje v menu Options. Pokud je maximální rychlost překročena, dojde k vynulování vstupu pro akceleraci vozidla. Podobně jsou řešeny vstupy z brzdy, spojky či volantu, avšak už bez podmínky maximální rychlosti. Stlačení spojky je řešeno ve funkci *Change Gear Control* na obrázku 2.16, přičemž se kontroluje podmínka dostatečného stlačení tohoto pedálu na minimální hodnotu 0,5.

Detekce vstupu z volantu Logitech G920, jeho pedálů a řadící páky probíhá s využitím pluginu Raw Input. Tento plugin umožňuje získávat vstupní data ze zařízení, která nejsou ovladatelná skrze rozhraní XInput (plugin pro mapování gamepadů). Používá se nejčastěji právě pro volanty či joystick (pro letecké simulátory). Tlačítka

a osy těchto zařízení mohou být mapovány na ovládací mechanizmy aplikace. Identifikace zařízení probíhá pomocí vendor ID a product ID, což jsou hexadecimální hodnoty určené ovladačem zařízení. Obvykle se mapuje až 8 os a 20 tlačítek zařízení s tím, že každá osa či tlačítko mohou být vypnuty. [19]

Ovládání vozidla

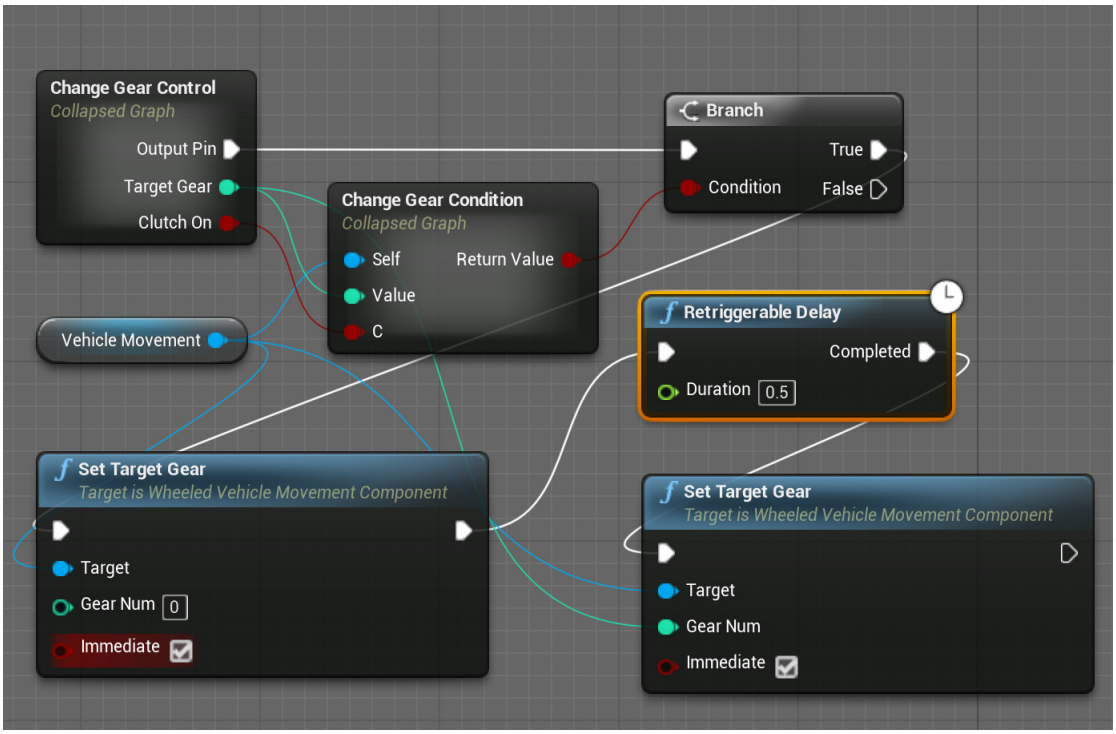
Vehicle Movement komponenta, která je součástí *cds_vehicle_bp*, zabezpečuje pohyb vozidla v prostředí dle nastavitelných hodnot - parametrů motoru, otáčení, převodu rychlosti a dalších (viz kapitola 1.5). Hodnoty pro nastavení chování vozidla jsou odvozené od reálného auta. Vozidlo má přední náhon, maximální počet otáček za minutu (RPM) 7000, hmotnost 1720 kg, nastavené přechody mezi rychlostmi podle automatické převodovky, torzní křivku apod.



Obr. 2.15: Navázání vstupů na funkce Vehicle Movement

Uživatel má možnost vybrat si mezi automatickým či manuálním převodem rychlostí. Změnu může provádět v menu Options, ke kterému se dostane pozastavením hry nebo z hlavního menu. Automatický převod řeší komponenta Vehicle Movement, ve které lze nastavit parametry tohoto převodu - poměr mezi rychlostmi, otáčky při podřazení/zařazení či například prodlevu mezi řazením. Manuální převod je poté řešen pomocí kódu v blueprintu na obrázku 2.16.

Zařazená rychlost je kontrolována v zapouzdřené funkci *Change Gear Control* (v rámci úspory místa není zobrazen celý kód, který zastřešuje mapování vstupů). Tato funkce také řeší podmínku, zda-li při změně rychlostního stupně byla stlačena spojka. Posléze dochází k vykonání funkce *Change Gear Condition*, která kontroluje,



Obr. 2.16: Kód pro změnu rychlosti

zda-li došlo k zařazení správné rychlosti. Není možné při nízké rychlosti vozidla zařadit vysoký rychlostní stupeň, auto by pak nemělo dostatečný výkon na svůj pohyb. Pokud jsou všechny tyto podmínky splněny, dochází ke změně stupně na neutrální (simulující přeřazení) a následné zařazení požadované rychlosti (proměnná *Target Gear*).

HUD

Blueprint vozidla je také zodpovědný za HUD (Heads Up Display) pro uživatele, který mu zobrazuje aktuální informace o rychlosti, aktuálním stupni rychlosti, cílovým stupněm rychlosti a vzdálenosti od čáry (důležitá součást scénářů). Vzdálenost je vyobrazována blíže středu obrazovky tak, aby se mohl uživatel zároveň soustředit na jízdu samotnou. Zbytek informací se nachází v pravém horním rohu obrazovky (viz obrázek 2.17).

2.1.6 Herní mechanizmy

Každá mapa, neboli level, má při svém spuštění navázány defaultní objekty typu player actor, game mode, player controller apod. Při načítání nové mapy tedy aplikace ví, jaký objekt bude primárně určen pro hráče či jaký je herní mechanismus

Obr. 2.17: Pohled z vozidla a HUD

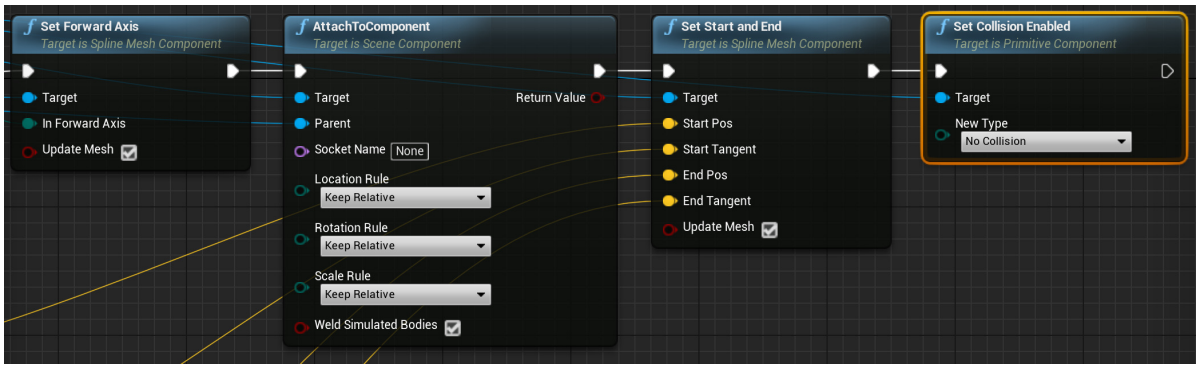
pro uživatele. Ten je dán kombinací objektu typu *GameModeBase* a blueprintu levelu (úrovně). Kromě základního módu hlavního menu, ve kterém se nic speciálního neděje, jsou vytvořeny další 4 aplikační módy, které korespondují se scénáři:

- Highway - step response
- Highway - long distance ride
- Town
- Moose test

Náplň scénářů a data, která jsou o operátorovi ukládána, vycházejí z teoretického rozboru o lidském operátorovi v MMS systémech. Ve scénářích je primárně pozorována odchylka operátora od trasy, například v reakci na skokovou změnu či udržení požadované trasy (jízdního pruhu) v delším časovém intervalu.

Line blueprint - vykreslování čáry

Unreal Engine nabízí možnost využití vykreslování spline - různě orientovaných čar, na které lze navázat objekty. Vykreslování těchto objektů (mesh) však není součástí, musí být proto vytvořeno zvlášť. Tuto situaci řeší kontrukční skript vytvořeného blueprintu *spline_bp*. Zabezpečuje, že mesh se bude vykreslovat mezi dvěma body na spline objektu a multiplikovat tak, aby nedocházelo k porušení konstrukce. Část skriptu lze vidět na obrázku 2.18. Blok *Set Forward Axis* zabezpečuje správnou orientaci objektu, který vykreslujeme - určuje, v jaké ose bude směřovat podél spline.



Obr. 2.18: Část konstrukčního algoritmu pro vykreslování čáry

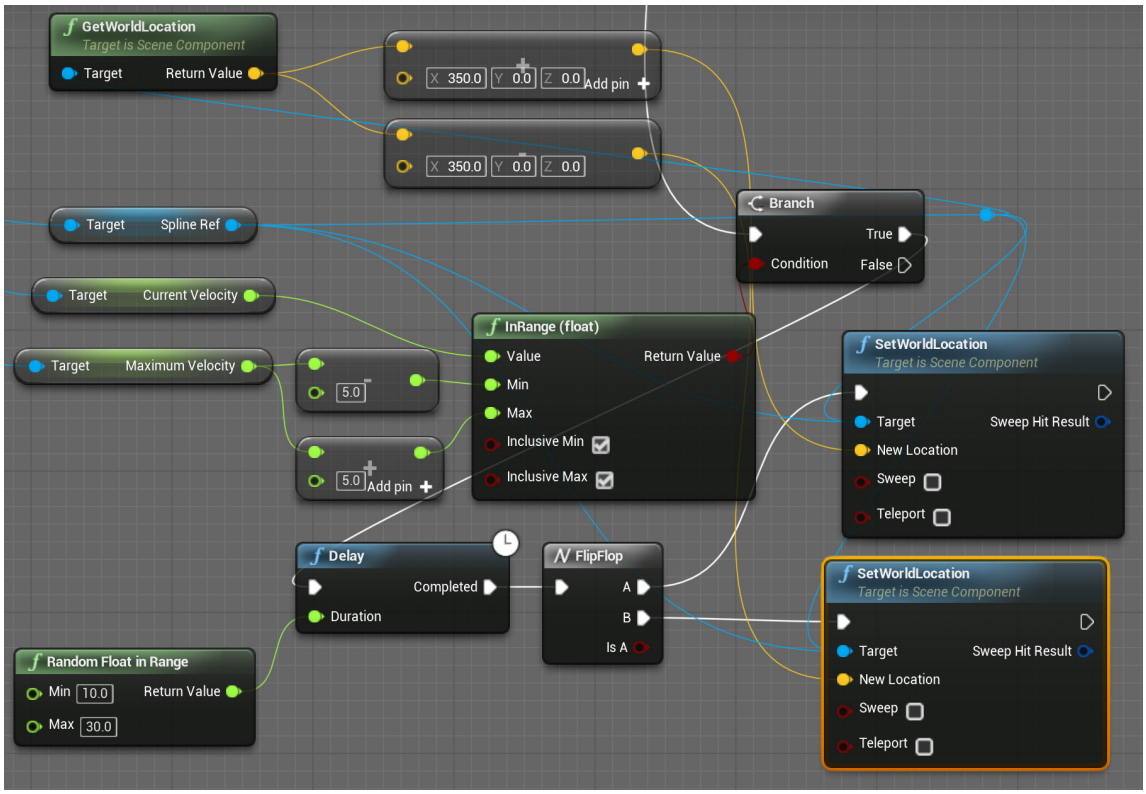
Attach To Component blok přiřazuje objekt (mesh) k spline, přičemž hned následující blok *Set Start And End* určuje počáteční a koncový bod vykreslování objektu (mesh) - pro vykreslování čáry je použit objekt laserového paprsku.

Scénář Highway - step response

Obr. 2.19: Scénář Highway - step response

Scénář obsahuje mapu dálnice, která je dlouhá přes 8 kilometrů a neobsahuje žádné zatáčky. Jedná se tedy o dlouhou a rovnou trasu. Uživatel, který si vybere tento scénář, se se svým vozidlem objeví na dálnici a jeho úkolem je následovat čáru, která se mu zjeví při zapnutí scénáře. Je na uživateli nebo správci simulátoru, jakou maximální rychlost tomuto vozidlu zvolí (změna je možná buď v hlavním menu nebo

při pozastavení aplikace). Jakmile dojde k dovršení této maximální rychlosti (v intervalu +- 5 km/h), tak dochází ke změně polohy čáry, kterou musí uživatel následovat. Tato změna je podmíněna dodržením intervalu v okolí maximální rychlosti a náhodně generovaným zpožděním přesunu čáry mezi 10 až 30 sekundami. Polohy čáry jsou 2 možné a přesně dané, mezi kterými se po splnění podmínek přepíná (obrázek 2.20). Kód je řešen v blueprintu herního módu cds_mode_highway_step_response. Ukončení scénáře je dáno projetím vozidla skrze trigger volume, neboli částí prostoru mapy. Toto ukončení je řešeno v blueprintu úrovně a uživatel se tak dostane na obrazovku úspěšného ukončení scénáře.



Obr. 2.20: Kód na změnu polohy čáry

Scénář Highway - long distance ride

Tento scénář je opět tvořen dlouhou dálnicí, která však obsahuje dlouhé zatáčky. Nejedná se tak o čistě rovnou trasu. Smyslem tohoto scénáře je měřit chování řidiče během dlouhé jízdy a jeho schopnost udržovat pozornost a stále stejný směr. Hlavním měřeným údajem je vzdálenost od čáry, kterou musí uživatel udržovat co nejmenší. Oproti scénáři step response však nedochází ke změně polohy této čáry.

Obr. 2.21: Scénář Highway - long distance ride

Ukončení scénáře je podmíněno projetím trigger volume na konci trasy.

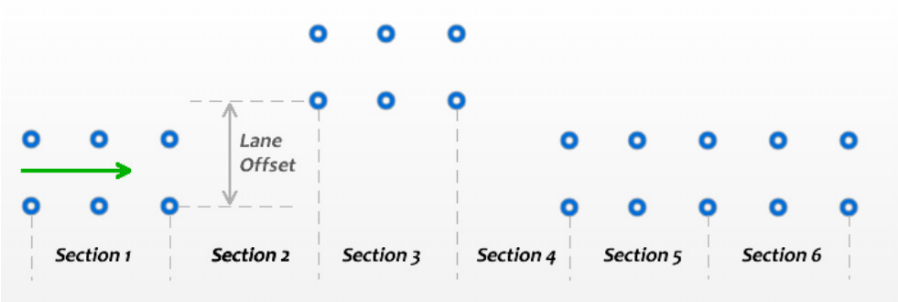
Scénář Town

Městská mapa (lze ji vidět na obrázku 2.10) obsahuje spoustu zatáček a kratších či delších cest. Zatím obsahuje mapa pouze pár budov z důvodu nedostatku volně dostupných assetů budov s texturami. Hlavním smyslem scénáře je, aby si uživatel vyzkoušel ovládání vozidla a byl nucen neustále měnit rychlosti, přidávat plyn či brzdit. Během scénáře je ukládána trasa, kterou uživatel projíždí, která se dá výsledně porovnávat s jeho minulými jízdami. S ostatními výsledky měření tak dokážeme vytvořit ideální trasu, kterou by měl uživatel zvládnout projet. V budoucnu lze tento scénář využít s algoritmy pro autonomní řízení a porovnat trasy, které vypočítal algoritmus s těmi, které projeli skuteční řidiči a určit tak spolehlivost algoritmu či efektivitu řidiče. Po projetí celé trasy je scénář opět ukončen trigger volume na konci trasy.

Scénář Moose test

Jedná se o scénář, který vychází z reálného měření chování vozidel na krátkých trasách, který má za cíl otestovat stabilitu vozidla. Průběh a provedení testu vychází z úmluvy výrobců automobilových vozidel, kteří se shodli na jednotném testování vozidel tak, aby pak mohly být výsledky těchto testů jednoduše porovnatelné a unifikované. Test je definován ISO 388-2:2011 - definuje rozměry testovací tratě (na

obrázku 2.22 a v tabulce 2.1), prováděný manévr, potřebnou dynamiku vozidla a přilnavost povrchu. Tento test je možné aplikovat na osobní auta definovaná dle ISO 3833 a lehká komerční vozidla do maximální hodnoty 3,5 tuny. Lane offset na obrázku 2.22 určuje posunutí kuželů 3. sekce oproti 1. a 5. sekci. [20]



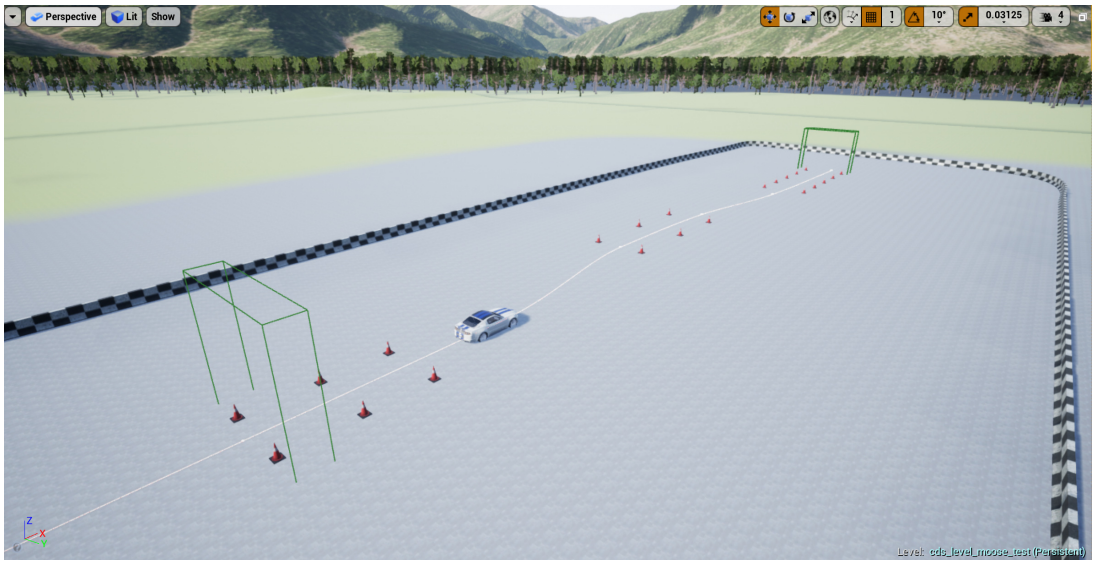
Obr. 2.22: Vizualizace testovací tratě [20]

Sekce	Délka sekce [m]	Šířka sekce [m]
1	15	1,1 * šířka vozidla + 0,25 m
2	30	1,1 * šířka vozidla + 0,25 m
3	25	1,2 * šířka vozidla + 0,25 m
4	25	1,2 * šířka vozidla + 0,25 m
5	15	1,3 * šířka vozidla + 0,25 m

Tab. 2.1: Tabulka rozměrů testovací tratě [20]

Měření času začíná ve vstupní sekci a končí při odjezdu vozidla z výstupní sekce. Vstup z plynového pedálu je při projížďce mezi vstupní a výstupní sekcí vypnut, takže auto projíždí testovací měřenou tratí bez akcelerace. Během testu nesmí dojít k dotyku vozidla s ohraničujícími kužely, jinak je měření neúspěšné. Test se provádí opakovaně se zvyšující rychlostí vozidla. Začíná se na 60 km/h a s každou iterací tato rychlost narůstá. Konec testu nastává tehdy, když při měření dojde ke smyku auta, dotyku s kužely nebo otočení vozidla. Nejčastěji k těmto jevům dochází při rychlostech 70 až 80 km/h. Test je prováděn bez jakékoliv pomocné stabilizace vozidla. [20]

Ve scénáři se nachází dva spouštěcí prostory (trigger volume) - vstupní a výstupní, lze je pozorovat na obrázku 2.23. Při projetí vozidla vstupním prostorem dochází k sepnutí měření času a zapisování informací o vozidle a jeho trase do souboru. Je také vypnut vstup z plynového pedálu (viz obrázek 2.15). Pokud během



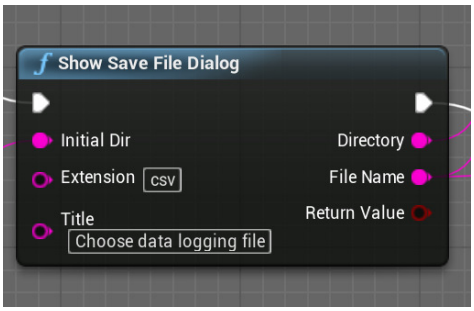
Obr. 2.23: Scénář moose test

testu dojde ke kolizi mezi vozidlem a hraničními kužely, scénář je neplatný a je ihned ukončen. Pokud auto projede trasu bez dotyku a úspěšně projede výstupním prostorem, scénář je ukončen úspěšně. Smyslem tohoto scénáře je vyzkoušet chování vozidla v simulátoru a také implementovat reálné měření, které se využívá pro testy aut.

2.1.7 Sběr a ukládání dat

Sběr dat probíhá pouze při spuštěném a běžícím scénáři. Při výběru scénáře z hlavního menu se uživateli zobrazí dialogové okno, ve kterém si vybere soubor, do kterého chce ukládat naměřená data. Tuto funkci zabezpečuje vlastní BlueprintFunction třída *Show Save File Dialog* (na obrázku 2.24). Po výběru souboru dochází k měření dat dle daného scénáře. U Highway - step response, long distance ride a Town dochází k zápisu do souboru ihned po spuštění scénáře. Ve scénáři Moose test je tento zápis podmíněn projetím vozidla vstupní řadou. Samotný zápis do souboru je řešen pomocí další, vlastní naprogramované BlueprintFunction třídy, kterou můžeme vidět na obrázku 2.25.

Výstupními hodnotami *Show Save File Dialog* funkce jsou *Directory* a *Filename*. *Directory* určuje adresář (včetně názvu souboru), kam se soubor bude ukládat, defaultně adresář projektu (nebo adresář s uloženou aplikací). *Filename* určuje název souboru, kterému funkce automaticky dodá koncovku *.csv* tak, aby s tímto souborem mohl nakládat například Microsoft Excel a jednoduše tak vložit data do tabulkového



Obr. 2.24: Vytvořený blueprint node pro volbu adresáře a souboru

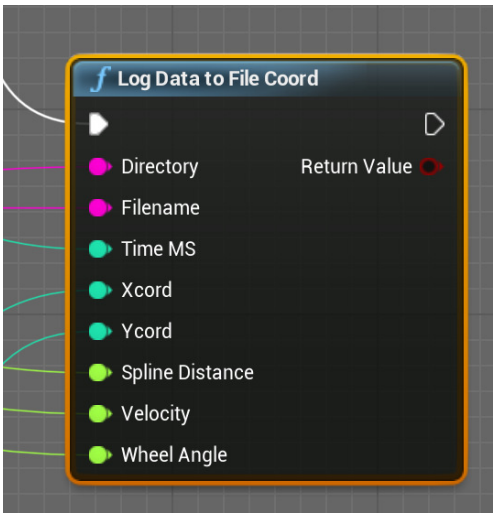
souboru.

Následně dojde k využití funkce *Log Data To File Coord*, která zapisuje naměřené hodnoty do souboru vybraného funkcí *Show Save File Dialog*. Parametry, které se ukládají jsou:

- TimeMS (Int32) - čas zápisu v milisekundách, určený aktuálním tickem aplikace. Jelikož aplikace vytvořená v Unreal Enginu je soft real-time, perioda vzorkování je tedy určena aktuálními dostupnými zdroji. Z naměřených dat lze však usoudit, že přesnost zápisu se pohybuje v rámci desítek milisekund.
- Xcord (Int32) - souřadnice osy X polohy vozidla ve světě
- Ycord (Int32) - souřadnice osy Y polohy vozidla ve světě
- Spline Distance (Float) - vzdálenost mezi vozidlem a čárou, kterou musí uživatel následovat (v metrech)
- Velocity (Float) - rychlost vozidla (v km/h)
- Wheel Angle (Float) - míra natočení volantu. Z kapitoly o popisku ovládání je známo, že tato proměnná nabývá hodnot od -1 (natočení úplně vlevo) do 1 (natočení úplně vpravo). 0 určuje neutrální polohu volantu.

Z výpisu hlavičkového souboru *cds_logdata_BP_function* (o pár řádků níže) lze vyčíst syntax, kterou využívá Unreal Engine. UFUNCTION označuje funkci, která je tvořena pro engine. Parametr BlueprintCallable určuje, že tuto funkci lze volat jako node (blok) v blueprintovém programování. Další parametr pak určuje její zařazení do kategorie, v tomto případě do *CDS Custom*. Celkově jsou vytvořeny tři funkce:

- LogDataOnScreen - funkce, která zobrazuje aktuální ukládaná data na obrazovku, byla využita pro debugging
- LogDataToFile - jedná se o okleštěnou verzi LogDataToFileCoord, která neukládá informace o poloze vozidla
- LogDataToFileCoord



Obr. 2.25: Vytvořený blueprint pro ukládání dat

```
#pragma once

#include "Engine.h"
#include "CoreMinimal.h"
#include "Kismet/BlueprintFunctionLibrary.h"
#include "UnrealString.h"
#include "cds_logdata_BP_function.generated.h"

/**
 *
 */
UCLASS()
class CAR_DRIVING_SIM_API Ucds_logdata_BP_function : public
    UBlueprintFunctionLibrary
{
  GENERATED_BODY()

  UFUNCTION(BlueprintCallable, Category = "CDS Custom")
  static bool LogDataOnScreen(float SplineDistance, float
      Velocity, float WheelAngle);

  UFUNCTION(BlueprintCallable, Category = "CDS Custom")
  static bool LogDataToFile(FString Directory, FString Filename,
       INT32 TimeMS, float SplineDistance, float Velocity, float
      WheelAngle);

  UFUNCTION(BlueprintCallable, Category = "CDS Custom")
  static bool LogDataToFileCoord(FString Directory, FString
      Filename, INT32 TimeMS, INT32 Xcord, INT32 Ycord, float
```

```
        SplineDistance, float Velocity, float WheelAngle);

};
```

Následuje výpis .cpp souboru *cds_logdata_BP_function*. Jedná se pouze o část celého souboru, která je zde využita pro názornou ukázku programování v Unreal Engine. Pro vytvoření adresářů cesty a souboru je využita třída IPlatformFile a pro následný zápis do souboru pak třída FFileHelper. Obojí jsou třídy dodávané v Unreal Engine. *CreateDirectoryTree* vytváří adresář a vrací hodnotu true, pokud byl adresář vytvořen nebo již existuje. *FileExists* kontroluje, zda-li soubor již není vytvořen. Pokud ne, tak třída *FFileHelper* tento soubor vytvoří funkcí SaveStringToFile a uloží do něj hlavičku a první data. Parametr *EFileWrite::FILEWRITE* určuje, zda-li má být vytvořen soubor nový, nebo má dojít k přepisu původního souboru, nebo k dodatečnému přidávání (append) na konec souboru. V případě vytvoření hlavičky tedy nejdříve dojde k vytvoření prázdného souboru a posléze při zápisu dochází k postupnému přidávání jednotlivých stringů, obsahujících informace o proměnných z aplikace. Tímto je zabezpečeno, že při výběru stejného souboru nedojde ke ztrátě dat.

```
bool Ucds_logdata_BP_function::LogDataToFileCoord(FString
    Directory, FString Filename, INT32 TimeMS, INT32 Xcord, INT32
     Ycord, float SplineDistance, float Velocity, float
    WheelAngle)
{


  IPlatformFile& PlatformFile = FPlatformFileManager::Get().
      GetPlatformFile();

  if (PlatformFile.CreateDirectoryTree(*Directory))
  {
    // Gets absolute file path
    FString FilePath = Directory + "/" + Filename;
    FString Header = "Time(ms);X pos;Y pos;SplineDistance;
        Velocity;WheelAngle\n";
    FString StringToSave = FString::FromInt(TimeMS) + ";" +
        FString::FromInt(Xcord) + ";" + FString::FromInt(Ycord) +
         ";" + FString::SanitizeFloat(SplineDistance) + ";" +
        FString::SanitizeFloat(Velocity) + ";" + FString::
        SanitizeFloat(WheelAngle) + "\n";

    // If file does not exist, creates header. Otherwise appends
         data at the end of the file
    if (!PlatformFile.FileExists(*FilePath))
```

```
    {
      FFileHelper::SaveStringToFile(Header, *FilePath,
          FFileHelper::EEncodingOptions::AutoDetect, &
          IFileManager::Get(), EFileWrite::FILEWRITE_None);
      FFileHelper::SaveStringToFile(StringToSave, *FilePath,
          FFileHelper::EEncodingOptions::AutoDetect, &
          IFileManager::Get(), EFileWrite::FILEWRITE_Append);
    }
    else
    {
      FFileHelper::SaveStringToFile(StringToSave, *FilePath,
          FFileHelper::EEncodingOptions::AutoDetect, &
          IFileManager::Get(), EFileWrite::FILEWRITE_Append);
    }
  }
  return false;
}
```

2.2 Testování simulátoru a zpracování naměřených dat

Aplikace simulátoru byla otestována na PC s operačním systémem Windows. Vzhled celého simulátoru s komponenty lze pozorovat na obrázku 2.26. Soustava obsahuje následující HW:

- Procesor: AMD Ryzen 5 2600 šestijádrový procesor s taktem 3,40 GHz
- Grafická karta: Radeon RX 590
- HDD: SSD Disk Crucial MX500 500GB
- Operační paměť: HyperX 16GB DDR4 3200MHz
- Základní deska: MSI B450 TOMAHAWK
- Monitor: 49"Samsung C49HG90
- Volant a pedály: Logitech G920
- Řadící páka: Driving Force Shifter od společnosti Logitech

V tabulce 2.2 je vyobrazena část uložených dat ze simulátoru, která se dají posléze analyzovat dle potřeb. Zaokrouhlení hodnoty natočení volantu je ponecháno v takovém rozlišení záměrně, jelikož při řízení může docházet k malým změnám natočení volantu, které je potřeba registrovat.

Obr. 2.26: Vzhled simulátoru

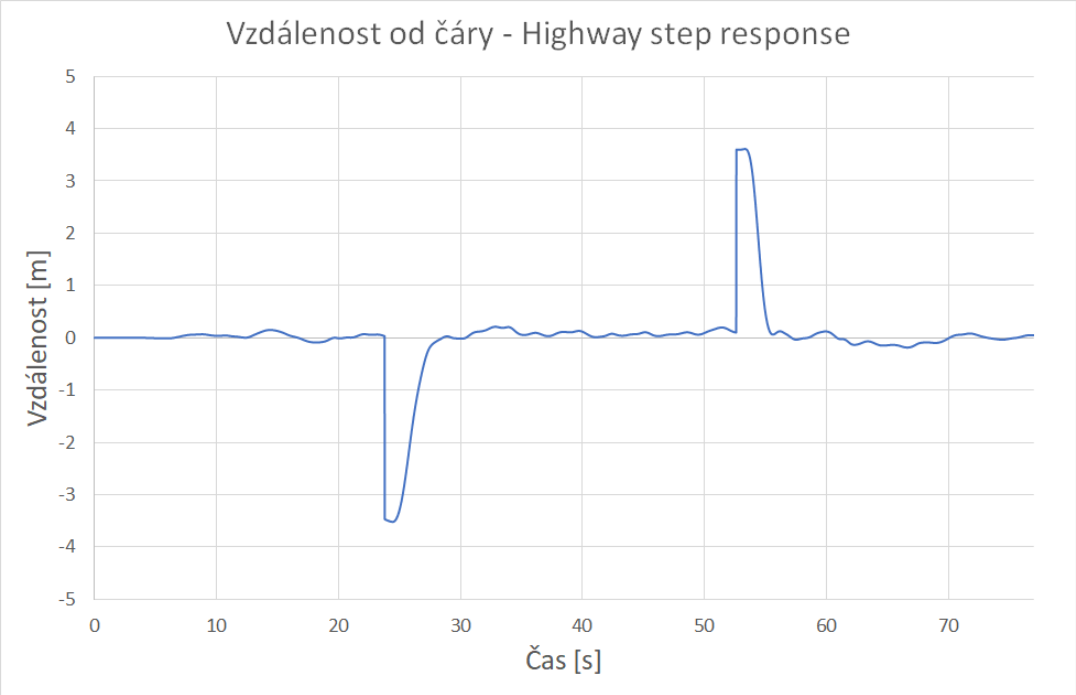
Získaná data ze scénáře Highway - Step response

Scénář Highway - Step response umožňuje získat informace o reakční době uživatele. V grafu 2.27 lze vypozorovat skokové změny polohy čáry. Uživatel byl nucen na tuto změnu reagovat změnou polohy volantu (a tím i vozidla), vyobrazené na grafu 2.28. Záporné hodnoty v grafu natočení volantu (i v dalších grafech) určují natočení doleva, kladné pak doprava. Záporné hodnoty v grafu 2.27 určují vzdálenost zprava, přičemž kladné hodnoty určují vzdálenost zleva od sledované čáry.

Reakční dobu lze vyjádřit z porovnání dvou sledovaných parametrů, a to změny polohy čáry a změny natočení volantu. Časový rozdíl mezi skokovou změnou polohy a natočením volantu správným směrem o určitou minimální hodnotu je úměrný reakční době řidiče. Tato hodnota odpovídá u řidiče přijetí informace (o změně po-

Čas [ms]	Souřadnice X [-]	Souřadnice Y [-]	Vzdálenost od čáry [m]	Rychlost [km/h]	Natočení volantu [-]
.	.	.	.	.	.
.	.	.	.	.	.
9937	22	6752	0.22	78	0.19145
9949	22	6778	0.22	78	0.16289
9961	22	6805	0.22	78	0.13038
9973	22	6831	0.22	79	0.10255
9986	23	6858	0.22	79	0.07142
.	.	.	.	.	.
.	.	.	.	.	.

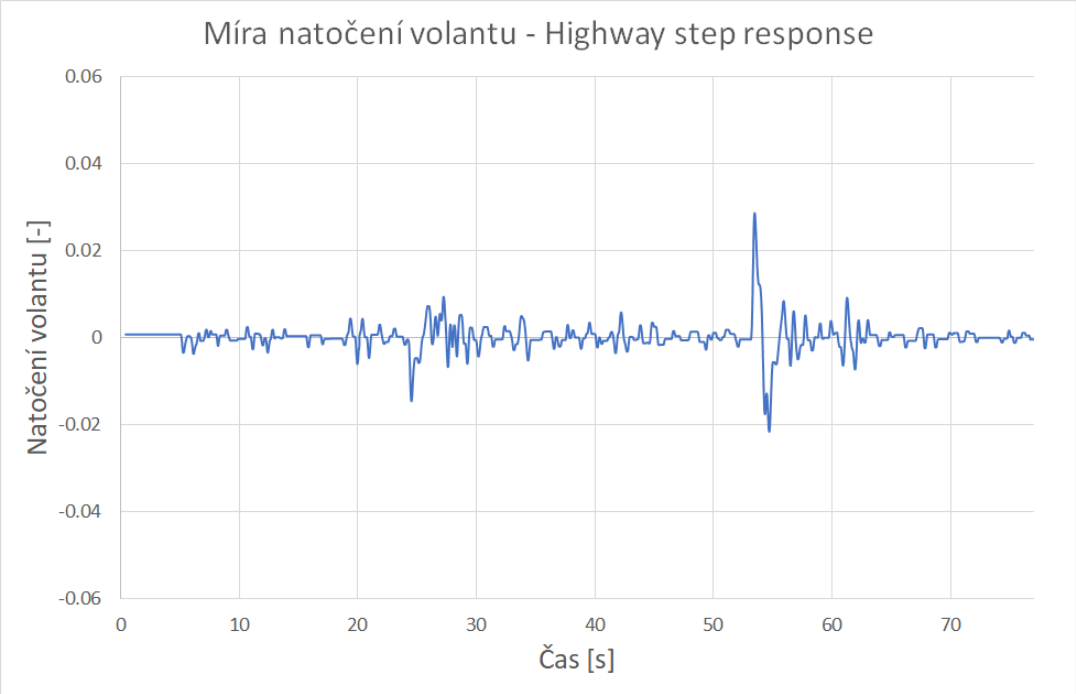
Tab. 2.2: Ukázka naměřených dat ze simulátoru



Obr. 2.27: Graf vzdálenosti vozidla od čáry ve scénáři Highway - Step response

lohy), vyhodnocení situace a následnou reakcí (natočení volantu). Hodnoty reakčních dob byly měřeny s jedním uživatelem při dvou jízdách a celkově jedenácti změnách polohy sledované čáry. Zobrazeny jsou v následující tabulce 2.3:

Z nichž průměrná hodnota reakční doby řidiče činí **670 ms**.



Obr. 2.28: Graf natočení volantu ve scénáři Highway - Step response

Reakční doba [ms]	572	616	679	957	580	950	593	594	676	658	496

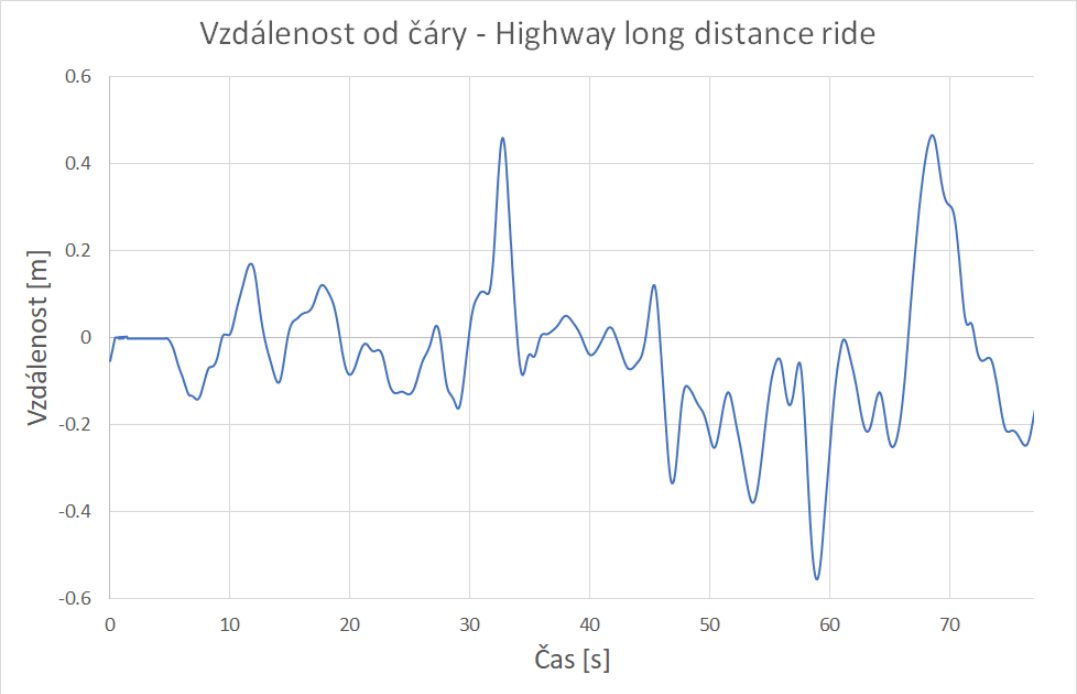
Tab. 2.3: Reakční doba řidiče na skokovou změnu polohy čáry

Získaná data ze scénáře Highway - Long distance ride

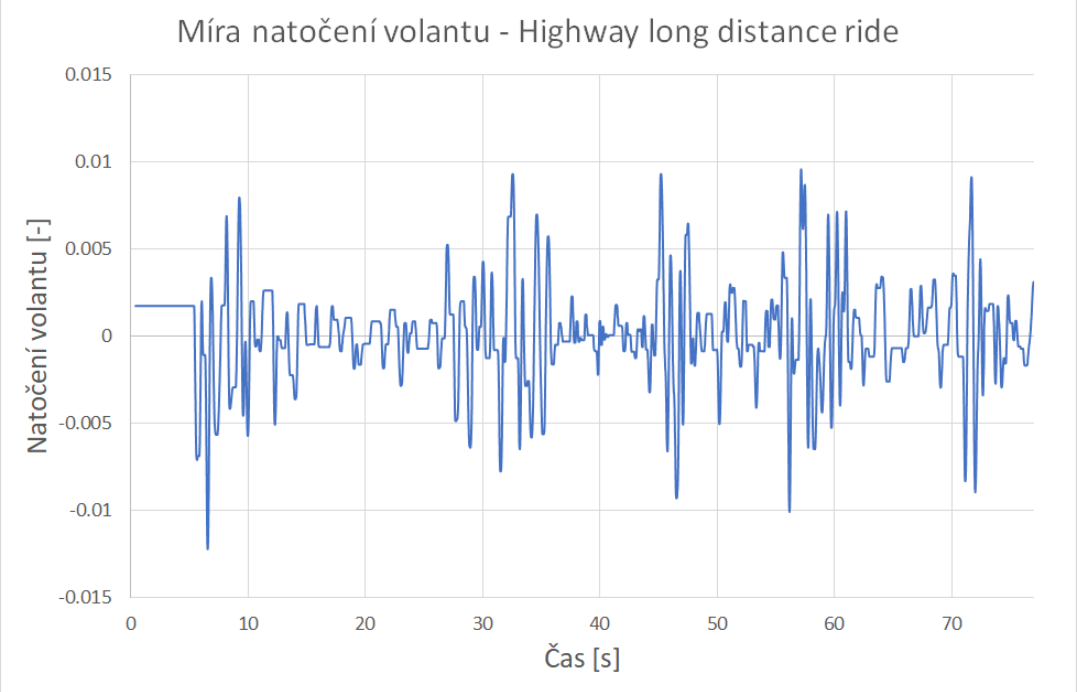
V grafu 2.29 lze pozorovat změnu vzdálenosti v čase ze scénáře Highway - Long distance ride. Poloha sledované linie se již nemění skokově, ale postupně. Jsou tak simulovány zatáčky na dálnici. Uživatel má za úkol udržovat vzdálenost mezi čarou a vozidlem co nejmenší, přičemž ji lze porovnat s grafem natočení volantu (na obrázku 2.30). Jsou pozorovatelné drobné impulsy řidiče pro udržení stabilní vzdálenosti. Z těchto dat je možné posléze analyzovat fyzický stav řidiče - jakým způsobem se v jeho řízení projevuje například únava nebo nepozornost.[1]

Získaná data ze scénáře Town

Ze scénáře Town lze kromě polohy získat také rychlost vozidla a natočení volantu. Na grafech 2.31 a 2.32 lze tyto veličiny v čase pozorovat, přičemž jejich průběh odpovídá vlastnostem trati, kterou musí uživatel projet. Neustále dochází ke změně rychlosti, brzdění, natočení volantu kvůli zatáčkám apod. Z těchto hodnot lze vhod-

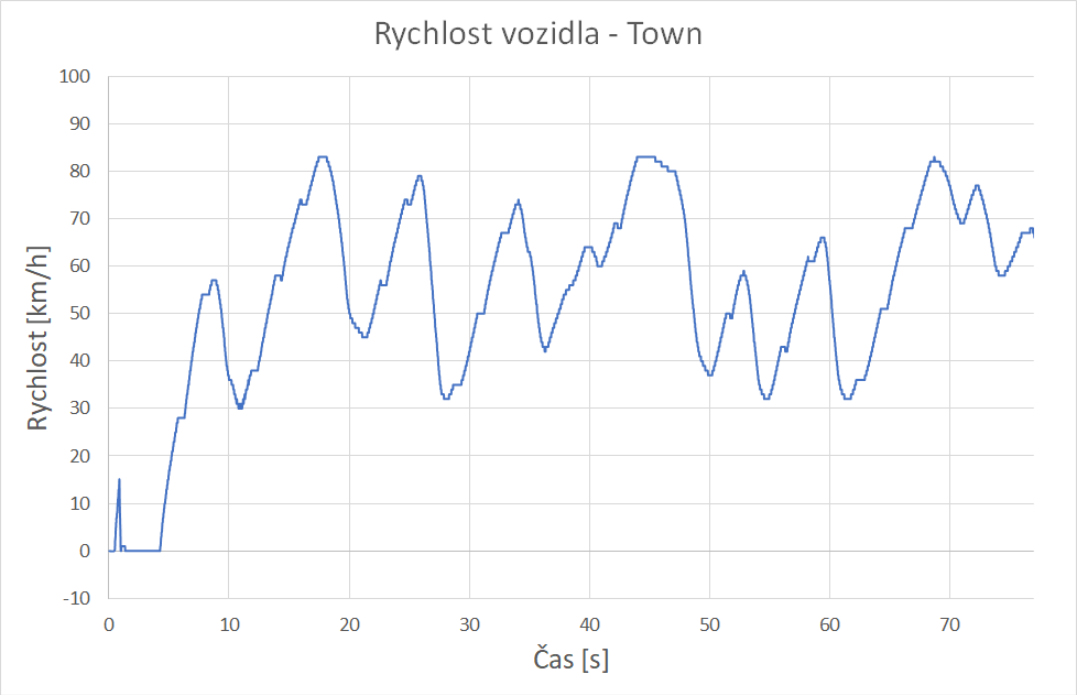


Obr. 2.29: Graf vzdálenosti vozidla od čáry ve scénáři Highway - Long distance ride

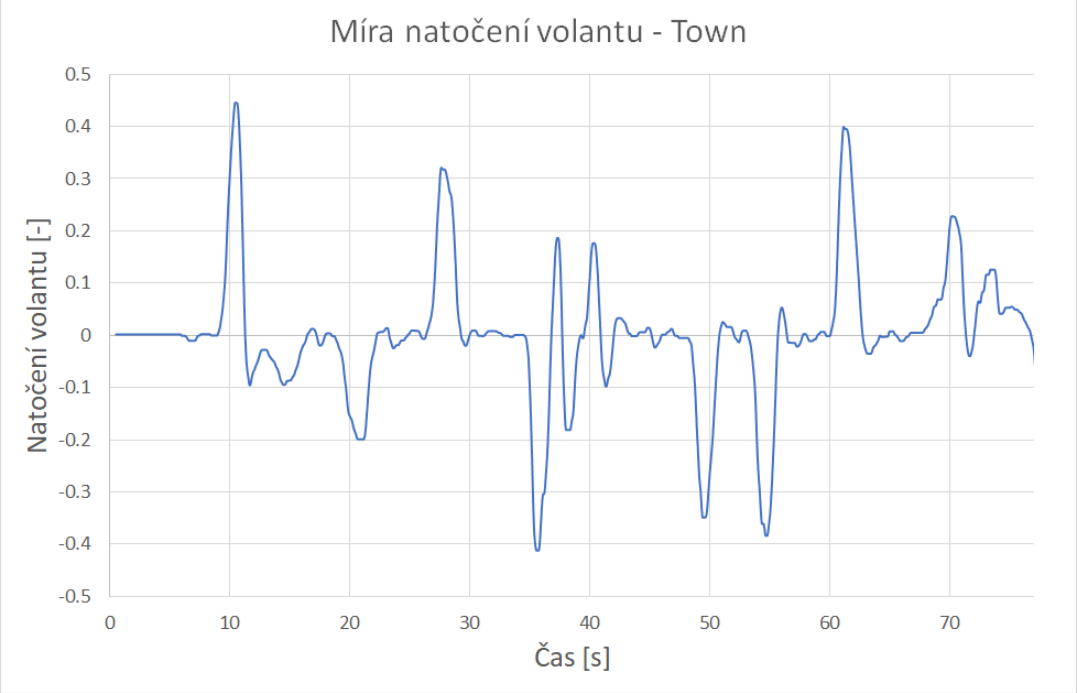


Obr. 2.30: Graf natočení volantu ve scénáři Highway - Long distance ride

nou analýzou získat informace o stylu řízení operátora (řidiče) - kupříkladu jedná-li se o agresivního řidiče s náhlým otáčením volantu a brzděním/přidávání plynu, či naopak o řidiče s pomalým otáčením volantu a pozvolným přidáváním/ubíráním plynu.



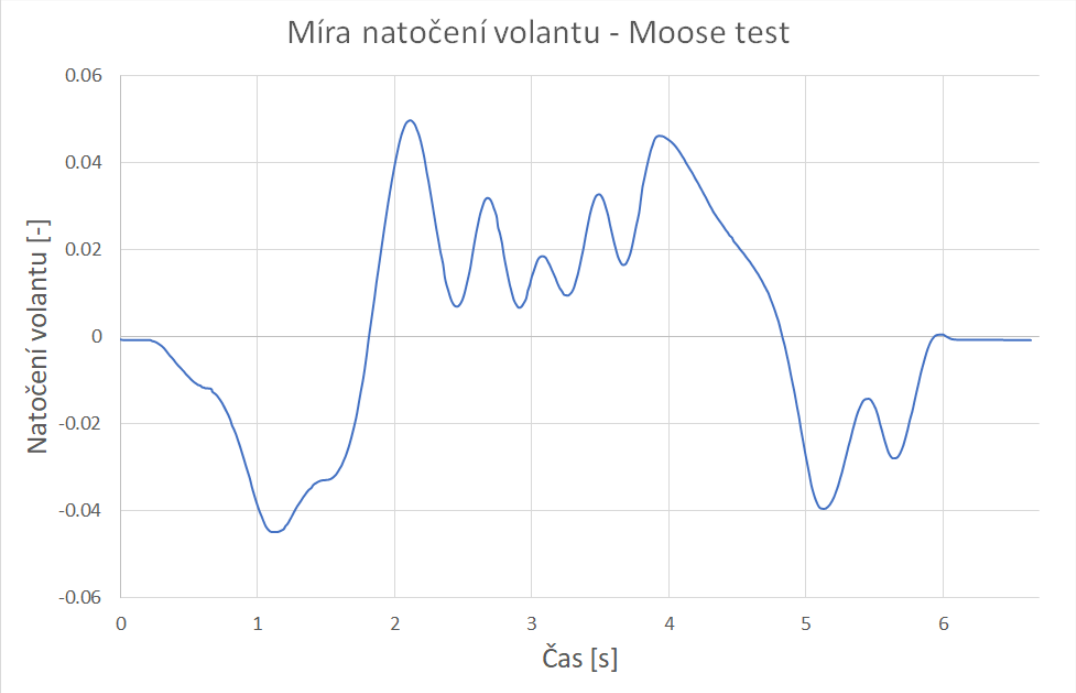
Obr. 2.31: Graf rychlosti vozidla ve scénáři Town



Obr. 2.32: Graf natočení volantu vozidla ve scénáři Town

Získaná data ze scénáře Moose test

Scénář Moose test vyplývá z reálného měření, které se používá na určení vlastností vozidla - přilnavost k povrchu, dynamika jízdy apod. Podobně je tomu tak i u tohoto scénáře, na kterém je primárně testováno chování vozidla a tím i jeho fyzikální model v prostředí Unreal Enginu. Můžeme však také získávat data o uživateli, například tehdy, je-li nucen ovládat vozidlo bez použití plynového pedálu. Jediným vstupem je tedy pouze natočení volantu, jehož znázornění vidíme na grafu 2.33. Záporné hodnoty odpovídají natočení volantu doleva, kladné pak natočení doprava. Z těchto dat lze posléze vyvodit úroveň řidičských schopností uživatele - jak reaguje na náhlé změny trasy (způsob otáčení volantu, jestli je pozvolný či sekaný). [1]



Obr. 2.33: Graf natočení volantu vozidla ve scénáři Moose test

3 ZÁVĚR

V diplomové práci byl proveden stručný teoretický rozbor hodnocení parametrů lidského operátora, ze kterého vzešly požadavky na pozorované jevy, které byly posléze aplikovány v simulátoru - sledování jízdního pruhu (čáry), polohy vozidla, míry natočení volantu a rychlosti. Následně byla popsána architektura a funkce herních enginů s navazující rešerší dostupných open source herních enginů. Vybrán byl Unreal Engine z důvodu dostupnosti assetů, dobře zpracované dokumentace, nápomocné komunity a implementovaným PhysX Vehicle SDK, který vytváří potřebný model vozidla.

V praktické části je navržena koncepce simulátoru zahrnující 4 hlavní předpoklady - reálné prostředí, vhodná implementace vozidla s funkčním ovládáním, scénáře ke sběru dat a samotný systém sběru a ukládání dat. Následně je v prostředí Unreal Engine vytvořena aplikace Car Driving Simulator, která výše uvedené body obsahuje a umožňuje tak zisk dat o operátorovi. Simulátor nabízí 4 scénáře, z nichž každý dodává různé druhy informací - reakční dobu uživatele na skokovou změnu, drobné změny polohy volantu při dlouhodobé jízdě v pruhu, styl jízdy řidiče nebo informace o chování modelu vozidla. Do budoucna je možné tyto scénáře obohatit nebo rozšířit o další, které opět umožní simulaci jiných procesů a zisk jiných druhů dat.

Vytvořená aplikace Car Driving Simulator bude v budoucnu využívána k výzkumu a vytváření modelu operátora. Hlavní předností této aplikace je její rychlá a relativně jednoduchá úprava při dostatečné znalosti fungování Unreal Engine. Může tak být rozšířena o potřebné funkce, které vyplynou z požadavků výzkumu. Je také vhodné soustředit se na zdokonalení aktuální verze simulátoru - správným ozvučením, přidáním kvalitnějších modelů a dotvořením prostředí, vylepšit chování vozidla na vozovce nebo zahrnout force feedback operátorovi.

LITERATURA

[1] HAVLÍKOVÁ, M.: Diagnostika systémů s lidským operátorem. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2009.

[2] HAVLIKOVA, M.; SEDIVA, S.; BRADAC, Z.; JIRGL, M.: A Man as the Regulator in Man-Machine Systems. Advances in Electrical and Electronic Engineering [online]. VSB-Technical University of Ostrava, 2014, 12(5), 469-475 [cit. 2019-05-06]. DOI: 10.15598/aeee.v12i5.1121. ISSN 1336-1376. Dostupné z: https://doaj.org/article/783b048069864f06850888311e43a29b

[3] RASMUSSEN, J.: Information Processing and Human-machine Interaction. An Approach to Cognitive Engineering, New York: North-Holland, 1985.

[4] GREGORY, J.: Game Engine Architecture, Second Edition. Second edition. A K Peters, 2014. ISBN 1466560010.

[5] TRISTEM, B.; PATTUZZI, S.: Unreal Engine C++ Developer: Learn C++ and Make Video Games. Udemy [online]. Aktualizováno 12/2018. [vid. 20.12.2018]. Dostupné z: https://www.udemy.com/unrealcourse

[6] FBX Static Mesh Pipeline | Unreal Engine. [online]. Copyright © 2004-2019, Epic Games, Inc. All rights reserved. [cit. 28.12.2018]. Dostupné z: https://api.unrealengine.com/INT/Engine/Content/FBX/StaticMeshes/index.html

[7] Geometry Brush Actors. [online]. Copyright © 2004-2019, Epic Games, Inc. All rights reserved. [cit. 28.12.2018]. Dostupné z: https://docs.unrealengine.com/en-US/Engine/Actors/Brushes

[8] Skeleton Assets. Object moved [online]. Copyright © 2004-2019, Epic Games, Inc. All rights reserved. [cit. 28.12.2018] Dostupné z: https://docs.unrealengine.com/en-us/Engine/Animation/Skeleton

[9] CRYENGINE Features. CRYENGINE | The complete solution for next generation game development by Crytek [online]. Copyright © 2019 Crytek GmbH. All rights reserved. [cit. 28.12.2018]. Dostupné z: https://www.cryengine.com/features

[10] CRYENGINE Help & FAQs. CRYENGINE | The complete solution for next generation game development by Crytek [online]. Copyright © 2019 Crytek GmbH. All rights reserved. [cit. 28.12.2018]. Dostupné z: https://www.cryengine.com/faq

[11] Products - Unity. Unity [online]. Copyright © 2019 Unity Technologies [cit. 30.12.2018]. Dostupné z: https://unity3d.com/unity

[12] bridge-6 – Unity Blog. Unity Blog - A glimpse inside Unity Technologies... [online]. Copyright © 2019 Unity Technologies [cit. 28.12.2018]. Dostupné z: https://blogs.unity3d.com/2015/07/01/the-state-of-unity-on-linux/bridge-6/

[13] Get Started with UE4. [online]. Copyright © 2004-2019, Epic Games, Inc. All rights reserved. [cit. 28.12.2018]. Dostupné z: https://docs.unrealengine.com/en-us/GettingStarted

[14] Introduction to C++ Programming in UE4. Object moved [online]. Copyright © 2004-2019, Epic Games, Inc. All rights reserved. [cit. 28.12.2018]. Dostupné z: https://docs.unrealengine.com/en-us/Programming/Introduction

[15] Introduction to Blueprints. [online]. Copyright © 2004-2019, Epic Games, Inc. All rights reserved. [cit. 28.12.2018] Dostupné z: https://docs.unrealengine.com/en-US/Engine/Blueprints/GettingStarted

[16] Simple movement blueprint - UE4 AnswerHub. [online]. Copyright © 2004-2019, Epic Games, Inc. All rights reserved. [cit. 29.12.2018]. Dostupné z: https://answers.unrealengine.com/questions/107146/simple-movement-blueprint.html

[17] Volanty a pedály Logitech G920 a G29 Driving Force. 301 Moved Permanently [online]. Copyright © 2019 Logitech. Všechna práva vyhrazena [cit. 30.12.2018]. Dostupné z: https://www.logitechg.com/cs-cz/products/driving/driving-force-racing-wheel.html

[18] S. Shah and D. Dey and C. Lovett and A. Kapoor, AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. Field and Service Robotics [Online] 2017. [cit. 28.12.2018]. Dostupné z: https://arxiv.org/abs/1705.05065

[19] RawInput Plugin. Object moved [online]. Copyright © 2004-2019, Epic Games, Inc. All rights reserved. [cit. 1.5.2019] Dostupné z: https://docs.unrealengine.com/en-US/Gameplay/Input/RawInput

[20] Car Engineer - Automotive engineering website [online]. Copyright © 2018 Car-engineer.com. All rights reserved. [cit. 1.5.2019] Dostupné z: http://www.car-engineer.com/the-moose-test-or-vda-test

[21] Vehicles — NVIDIA PhysX SDK 3.4.0 Documentation. NVIDIA Developer Documentation [online]. Copyright © Copyright 2008 [cit. 06.05.2019]. Dostupné z:

https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/-Manual/Vehicles.html

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

VR – virtuální realita

LOD – Level Of Detail - úroveň detailu renderování objektů

SDK – Software Development Kit

API – Application Programming Interface

STL – Standard Template Library

HUD – Heads-Up Display - informace na obrazovce pro hráče

GUI – Graphical User Interface

FMV – Full-Motion Video

IGC – In-Game Cinematics

IDE – Integrated Development Environment - programovací vývojové prostředí (př. Visual Studio)

DCC – Digital Content Creator - software třetí strany pro vytvoření assetů

NPC – Non-Player Character - charakter v herním prostředí, který interaguje s hráčem, avšak nemůže být ním přímo ovládán

SEZNAM PŘÍLOH

A OBSAH DVD

/.....Kořenový adresář
- Diplomová práce - Simulátor řízení vozidla.pdf
- /Car Driving Simulator/.....Aplikace Car Driving Simulator