VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

3D AUTOŠKOLA

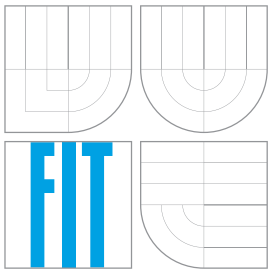
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR LANGR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

3D AUTOŠKOLA

3D DRIVING SCHOOL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR LANGR

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. KUNOVSKÝ JIŘÍ, CSc.

BRNO 2015

Abstrakt

Práce se věnuje tvorbě simulátoru 3D autoškoly a kontrole dodržování pravidel silničního provozu. Je zde představen herní engine Unity 3D, za jehož pomoci byl vytvořen simulátor autoškoly. V práci je představen způsob vývoje her od tvorby modelů po implementaci vlastní hry.

Abstract

This thesis deals with 3D Driving School simulator development, especially with control compliance of traffic rules. Thesis introduce game engine Unity 3D, which was the key factor of creating simulator. Game development in thesis begins with model and ends with scripting object behaviour.

Klíčová slova

Herní engine, Unity 3D, Autoškola, Pravidla provozu, Simulace

Keywords

Game engine, Unity 3D, Driving School, Traffic Rules, Simulation

Citace

Petr Langr: 3D Autoškola, diplomová práce, Brno, FIT VUT v Brně, 2015

3D Autoškola

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Jiřího Kunovského, CSc.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Langr
27. května 2015

Poděkování

Tímto bych rád poděkoval vedoucímu mé práce doc. Ing. Jiřímu Kunovskému, CSc. za vedení mé práce, dále bych rád poděkoval Mgr. Jurovi Jurčovi za poskytnuté odborné rady a své rodině a kolegům za jejich podporu.

Obsah

Kapitola 1

Úvod

V dnešní době je téměř každý vlastníkem řidičského oprávnění. Jde o nepostradatelnou dovednost moderních lidí. Vysoký počet řidičů navyšuje počet dopravních nehod, které mohou skončit zraněním i smrtí. Proto se musíme věnovat zlepšení kvality výcviku budoucích řidičů. V České Republice probíhá školení autoškoly nejprve teoretickou výukou, v níž jsou obvyklým školním způsobem vykládány pravidla provozu na pozemních komunikacích. Dále pak probíhá praktický výcvik, který začíná na automobilových trenažerech. Touto prací se snažíme přispět ke zlepšení školení řidičů tvorbou simulátoru autoškoly pro výcvikové účely.

V druhá kapitola se věnuje vybraným pravidlům, především dopravním značením a křižovatkám. Definujeme zde prioritizovaný seznam implementovaných pravidel, jelikož kontrola všech pravidel je moc obsáhlá úloha. Ve třetí kapitole provedeme analýzu technických nástrojů pro implementaci simulátoru. Dále rozebereme jednotlivá pravidla a jejich možný způsob řešení. Povíme si něco o provedení pohybu ostatních účastníků provozu v simulátoru, kteří hrají velkou roli při průjezdech křižovatkami. V neposlední řadě v této kapitole navrhneme vzhled a rozhraní aplikace. Ve čtvrté kapitole jsou představeny modely použité v simulátoru. Jde o modely města, prostředí a samotných vozidel. Pátá kapitola je věnována implementaci kontroly providel v Unity 3D. V této kapitole se dozvíme o použitých algoritmech kontroly, způsob tvorby města, sestavování komunikací. Dozvíme se jak moc jsou implementovaní boti chytří a jak zvládají úkony jako je dání přednosti v jízdě. V poslední kapitole je zhodnocení výsledného simulátoru a nastíněn další vývoj aplikace.

Kapitola 2

Simulace a kontrola dodržování pravidel silničního provozu

První výcvik začínajících řidičů před započetím povinných jízd probíhá na tzv. trenažeru. Prvotní mechanické trenažery byly vyrobené z kokpitu automobilu, kde nad řidičem byl umístěn průsvitný kotouč s modelem silnice. Tento model se pomocí reflektoru promítal na plátno před žáka. Pohyb vozidla byl proveden rotací tohoto kotouče. Hlavní účel těchto simulátorů je především osvojení ovládání automobilu. Velkou nevýhodou je chybějící zpětná vazba a interakce s ostatními účastníky provozu. Moderní simulátory tyto problémy odstraňují využitím počítačového zpracování a 3D grafiky. Zároveň přinášení větší možnosti v oblasti simulace silničního provozu. Student již nezíská jenom schopnost ovládání vozidla, ale také praktickou přípravu pro zvládání různých dopravních situací jako například průjezd křižovatkou, nebo udržování bezpečné vzdálenosti mezi vozidly. V této kapitole se podíváme jaké simulátory jsou komerčně dostupné, jaká funkcionalita by neměla chybět a definuji seznam pravidel pro můj simulátor autoškoly.

2.1 Simulátor silničního provozu

Hlavním cílem simulátoru autoškoly je poskytnou autoškolám prostředek pro školení studentů před započetím povinných jízd. Až po úspěšném zvládnutí simulátoru je student schopen přestoupit k další fázi. Tyto simulátory by měli obsahovat ovládání vozidla (řazení, znamení o změně směru jízdy apod.) a kontrolu dodržování pravidel silničního provozu.

Jedním příkladem je 3D Driving-School (3D-Fahrschule). Jedná se o simulátor s 3D grafikou, který obsahuje lekce s instruktorem, více zemí a vozidel. Dále je možné si zvolit jízdu za snížené viditelnosti, v obci, nebo na dálnici a jiné další. Uživatel je okamžitě informován o přestupcích kterých se dopustil formou vyskakujících zpráv v horní části okna simulátoru. Na obrázku 2.1 je pohled z vozidla tohoto simulátoru, který také umožňuje pohled z vnějšku vozidla.

Druhým příkladem je Driving School simulator 2014. Jedná se graficky velmi pěkný simulátor. Není zde možnost lekce s instruktorem, který říká směr jízdy, ale na vozovce jsou znázorněny šipky určující další směr jízdy. Navíc tento simulátor využívá hodnotícího systému. Za úspěšné zvládnutí lekce uživatel získá body, které slouží pro zpřístupnění další lekce. Tyto body je také možné získat např. za správné projetí křižovatky apod.. Na druhou stranu je tyto body možné ztratit při porušení pravidel. Samozhřejmostí je kontrola dodržování pravidel silničního provozu a okamžitá zpětná vazba o přestupcích. Další funkčností je

Obrázek 2.1: 3D Driving School

povinnost před každou jízdou zapnout bezpečnostní pásy. Stejně jako předchozí simulátor i tento podporuje pohled z vnitřku a vnějšku vozidla. Na druhou stranu zde není tak silná zpětná vazba o přestupcích. Na obrazovce se pouze zobrazí ztráta bodů a krátký text k tomu.

Obrázek 2.2: Driving School Simulator 2014

2.2 Analýza platných pravidel systému simulátoru

Provoz na pozemní komunikaci podléhá zákonu č. 361/2000 Sb. [16]. Tento zákon definuje práva a povinnosti účastníků silničního provozu a pravidla provozu. V této podkapitole probereme některá platná pravidla a definujeme jejich prioritu implementace do výsledné aplikace autoškoly.

2.2.1 Seznam implementovaných pravidel

Protože implementace kontroly všech platných pravidel zákona je časově náročná, definujeme si prioritizovaný seznam pravidel. Nedílnou součástí simulátoru je vzájemná interakce účastníků provozu. Z tohoto důvodu se práce, mimo kontrolu pravidel, také věnuje imple-

mentaci pohybu ostatních účastníků a jejich dodržování pravidel silničního provozu. Seznam pravidel je tedy následující:

1. Dopravní značení
2. Rychlost jízdy
3. Jízda křižovatkou a odbočování
4. Křižovatka řízená světelnými signály
5. Zastavení a stání

2.2.2 Dopravní značení

Nedílnou součástí pozemní komunikace jsou dopravní značky. Každý řidič je povinnen znát a dodžovat všechny dopravní značky. Podle umístění rozlišujeme značky svislé a vodorovné. Dále rozlišujeme značky stálé a dočasné. Dočasným značením se upravuje provoz například při opravě pozemní komunikace.

Obrázek 2.3: Dopravní značky svislé a vodorovné [16]

Svislé dopravní značky

Svislé dopravní značky jsou umístěny v blízkosti vozovky na viditelném místě. Nesmí dojít k jejich zakrývání. Svislé dopravní značky jsou stálé, proměnné a přenosné. Proměnná svislá dopravní značka je dopravní značka, jejíž činná plocha se může měnit. Přenosnou svislou dopravní značkou se rozumí dopravní značka umístěná na červenobíle pruhovaném sloupku (stojánku) nebo na vozidle [16]. Rozlišujeme následující typy svislých dopravních značek podle jejich účelu:

1. Výstražné značky upozorňují na místa, kde účastníku provozu na pozemních komunikacích hrozí nebezpečí a kde musí dbát zvýšené opatrnosti.

2. Značky upravující přednost stanoví přednost v jízdě v provozu na pozemních komunikacích.

3. Zákazové značky ukládají účastníku provozu na pozemních komunikacích zákazy nebo omezení.

4. Příkazové značky ukládají účastníku provozu na pozemních komunikacích příkazy.

5. Informativní značky poskytují účastníku provozu na pozemních komunikacích nutné informace, které slouží k jeho orientaci, nebo mu ukládají povinnosti stanovené zákonem, nebo zvláštním právním předpisem.

6. Dodatkové tabulky zpřesňují, doplňují nebo omezují význam dopravní značky, pod kterou jsou umístěny.

Vodorovné dopravní značky

Vodorovné značky se nacházení na pozemní komunikaci vyznačené bílými čarami, nebo šipkami. Mohou se vyskytovat společně se svislou značku, kterou zpřesňují. Přechodové vodorovné dopravní značky jsou vyznačeny žlutou, nebo oranžovou barvou.

2.2.3 Rychlost jízdy

Zákon říká, že řidič musí přizpůsobit rychlost jízdy svým schopnostem, technickému stavu a kategorii pozemní komunikace, povětrnostním podmínkám a jiným okolnostem, které by mohli ovlivnit jeho schopnost včas reagovat na vzniklé dopravní situace. Na druhou stranu řidič nesmí náhle snížit rychlost jízdy a omezovat plynulost provozu. Dále musí dbát omezením maximální povolené rychlosti v obci i mimo obec a na dálnici, nebo na silnici pro motorová vozidla. V obci smí jet řidič rychlostí nejvýše 50 km/h a jde-li o dálnici, nebo silnici pro motorová vozidla nejvýše 80 km/h. Mimo obec smí řidič motorového vozidla o maximální přípustné hmotnosti nepřevyšující 3 500 kg jet rychlostí nejvýše 90 km/h.

2.2.4 Jízda křižovatkou

Křižovatka je místo střetu dvou a více pozemních kumunikací. Zákon o jízdě křižovatkou definuje pravidla nutná pro dodržení plynulého a bezpečného provozu. Řízení provozu na křižovatce je buď světelnými signály, dopravnímy značkami a pokud není použit ani jeden ze zmíněných způsobů je provoz řízen implicitní předností vozidel přijíždějících zprava. Pokud je křižovatka řízena světelnými signály, které jsou mimo provoz (blikající žluté světlo), pak je provoz řízen dopravnímy značkami. Nejsou-li ani ty dostupné, pak zde platí pravidlo přednosti zprava.

U křižovatek řízených dopravnímy značkami používáme pojem hlavní a vedlejší pozemní komunikace. Vozidla na hlavní pozemní komunikaci mají přednost v jízdě, pokud neodbočují vlevo, pak platí pravidla ohledně odbočování. Vozidla jedoucí po vedlejší pozemní komunikaci jsou povinna dát přednost v jízdě vozidlům jedoucím po hlavní pozemní komunikaci. Dodatkové tabultky jsou zde použity ke znázornění tvaru a typu komunikací křižovatky.

Řidič přijíždějící ke křižovatce po vedlejší pozemní komunikaci označené dopravní značkou „*Stůj, dej přednost v jízdě!*“ musí zastavit vozidlo na takovém místě, odkud má do křižovatky náležitý rozhled [16]. Dále nesmí vjet do křižovatky, pokud mu aktuální situace nedovoluje pokračovat v jízdě. To neplatí v případě, že řidič odbočuje vlevo.

Obrázek 2.4: Dodatková tabulka E02b

Jedním speciálním typem křižovatky je kruhový objezd, který je značen dopravní značkou „*Kruhový objezd*“ společně se značkou „*Dej přednost v jízdě!*“, nebo značkou „*Stůj, dej přednost v jízdě!*“. V takovém případě musí řidič dát přednost vozidlům jedoucím po kruhovém objezdu a dodržet směr jízdy.

2.2.5 Odbočování

Při odbočování je řidič povinnem dát znamení o změně směru jízdy, dbát zvýšenné opatrnosti a neohrozit řidiče jedoucí za ním. Řidič odbočující v pravo se musí řadit co nejvíce vpravo s ohledem na šířku vozidla, dává při tom znamení o změně směru jízdy v pravo. Odbočuje-li řidič v levo řadí se naopak co nejvíce k levému okraji části vozovky vymezené pro daný směr jízdy, při tom dáva znamení o změně směru jízdy vlevo a přednost protijedoucímu vozidlu. Odbočuje-li protijedoucí vozidlo také vlevo, řidiči se vyhýbají vlevo.

2.2.6 Zastavení a stání

Zastavení znamená uvést vozidlo do klidu na dobu nutnou pro nastoupení či vystoupení přepravovaných osob, nebo naložení čí složení nákladu. Stání znamená uvést vozidlo do klidu nad dobu dovolenou pro zastavení. Řidič smí zastavit a stát:

1. Vpravo ve směru jízdy co nejblíže u kraje vozovky. V jednosměrné ulici smí zastavit a stát i u levého okraje vozovky.

2. Podélně v jedné řadě, nebo řidič vozidla o pohotovostní hmotnosti nepřevyšující 3 500 kg může stát kolmo popřípadě šikmo nedojde-li k omezení a bezpečnosti provozu.

3. Při stání musí zůstat jízdní pruh široký nejméně 3m pro každý směr jízdy. Při zastavení stačí jeden takovýto jízdní pruh pro oba směry.

Řidič nesmí zastavit a stát:

1. Na přechodu pro chodce, nebo na přejezdu pro cyklisty a ve vzdálenosti kratší 5m před i za nimi.

2. Na křižovatce a ve vzdálenosti kratší 5m před hranicí křižovatky a za ní. Vyjímkou je křižovatka typu „T“ kde řidič může zastavit a stát naproti vyúsťující pozemní komunikace.

2.2.7 Řízení provozu světelnými signály

Pro řízení křižovatky světelnými signály se využívá tříbarevné soustavy červené, žluté a zelené. Jednotlivý signály znamenají následující:

1. Signál s plným zeleným kruhovým světlem znamená volno, přičemž řidič musí dodržet ustanovení o odbočování a zároveň musí dát přednost chodcům na přechodě přecházejícím ve volném směru.

2. Signál se žlutým světlem znamená povinnost řidiče zastavit před vodorovnou dopravní značkou „*Příčná čára*“, nebo touto značkou opatřenou nápisem „*STOP*“ případně symbolem „*Dej přednost v jízdě!*“. Pokud není přítomna tato značka musí řidič zastavit před světelným signalizačním zařízením. Je-li řidič při rozsvícení tohoto signálu v takové blízkosti, že není schopen bezpečně zastavit, může pokračovat v jízdě.

3. Signál s červeným světlem znamená „*Stůj!*“ a řidič je povinen zastavit obdobně jako v předchozím případě, bez vyjímky.

4. Signál se současným žlutým a červeným světlem znamená povinnost řidiče k přípravě jízdy.

Obrázek 2.5: Příklad světelných signálů

Kapitola 3

Architektura simulátoru a uživatelského rozhraní

V této kapitole se budeme věnovat návrhu architektury výsledné aplikace autoškoly. Ukážeme si jaké jsou v dnešní době možnosti pro vývoj počítačových her. Na základě těcho možností zvolíme vhodnou cestu vývoje. V závislosti na zvolené technice vytvoříme řešení kontroly dodržování zvolených dopravních pravidel z předchozí kapitoly.

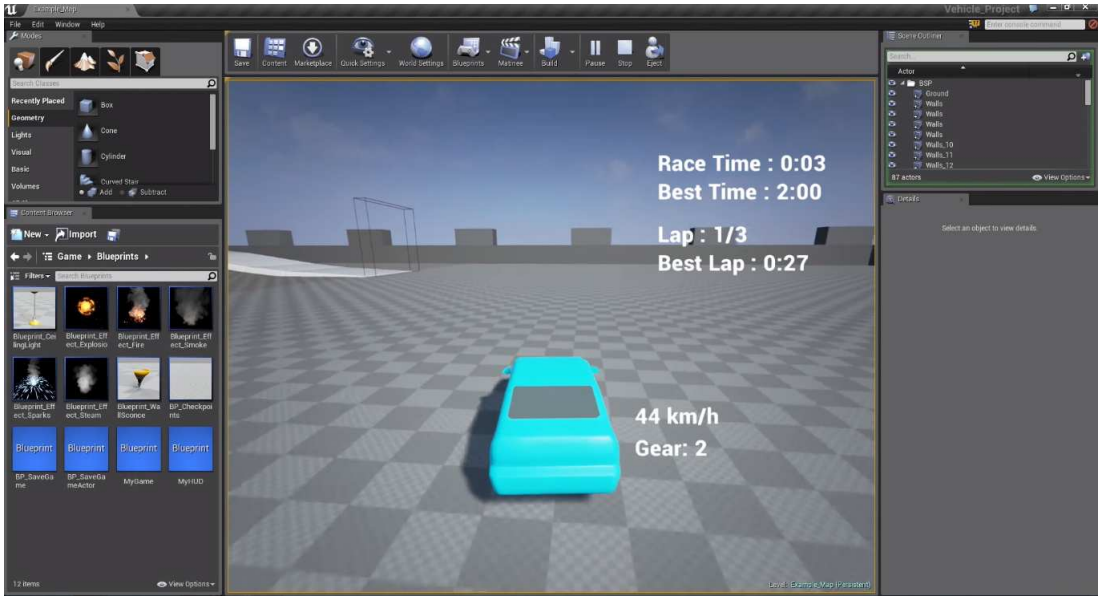
3.1 Herní engine

Herní engine je jádro počítačových her abstrahující moduly pro renderování 3D grafiky, detekci kolizí, přehrávání zvuku a čtení vstupů, tak aby se vývoj her mohl soustředit právě na hru samotnou. Hranice mezi hrou a herním engine je opravdu malá a není přesně vymezený rozdíl. Vznik termínu *herní engine* se datuje do 90. let a spojuje se s populární hrou *Doom* od *id software* [3]. Hlavní cíl herních enginů je znovu použitelnost klíčových komponent. Tyto enginy musejí být vysoce modifikovatelné a proto implementace konkrétních specifik dané hry se provádí pomocí skriptování. Použití herního engine velmi zjednoduší vývoj simulátoru autoškoly a zjednoduší údržbu simulátoru v budoucnu.

3.1.1 Unreal Engine

Tento engine je vyvíjen společností Epic Games. Jedná se o velmi populární herní engine, který je dostupný od roku 1998. V současné době je už jeho čtvrtá verze. V této verzi Epic Games představila nové funkce a nástroje, které podporují rychlý iterativní model vývoje, při kterém vývojář okamžitě vidí výsledek své práce [2]. Vývoj her za pomoci tohoto engine probíhá v programovacím jazyce c++. S užitím Unreal Engine je postavana stovka her a také real-time 3D filmy, tréninkové simulace a vizualizace. Jedna z nových funkci čtvrté verze je podpora rozšířených renderovacích funkcí DirectX verze 11 a 12 jako je full-scene HDR reflections, tisíce dynamických světel scény a fyzikálně založené stínování.

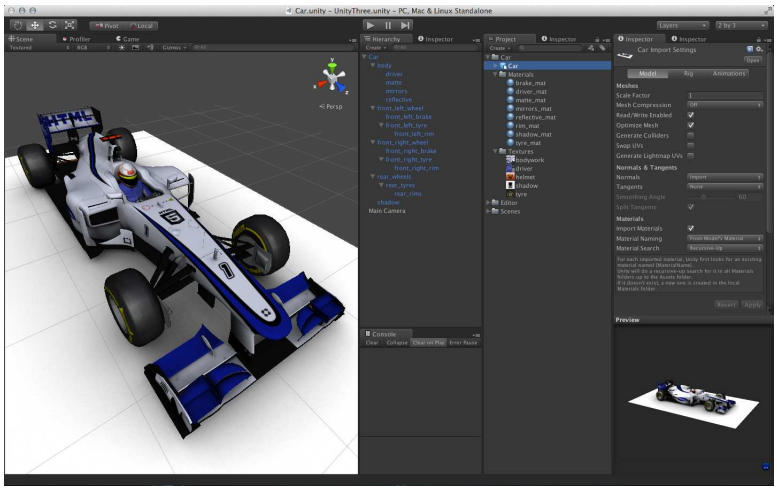
Výhodou tohoto engine je již připravaná implementace pohybu vozidla, které stačí dodat model a konfiguraci fyziky. Kontrola dodržování pravidel také není problém. Unreal Engine disponuje *triggry*, což jsou tělesa hry, které spouštějí událost při interakci s jinými objekty scény. Unreal Engine má velkou komunitu lidí, kteří jsou ochotni pomoci s jakýmkoli problémem. Další výhodou jsou online dostupné lekce pro začátečníky i pokročilé. Podporované platformy jsou Windows PC, Mac OS X, iOS, Android, Linux, SteamOS a



Obrázek 3.1: Unreal Engine

HTML5. Podle článku na officiálních stránkách enginu je od března roku 2015 dostupná volná licence [11].

3.1.2 Unity 3D



Obrázek 3.2: Unity3D editor

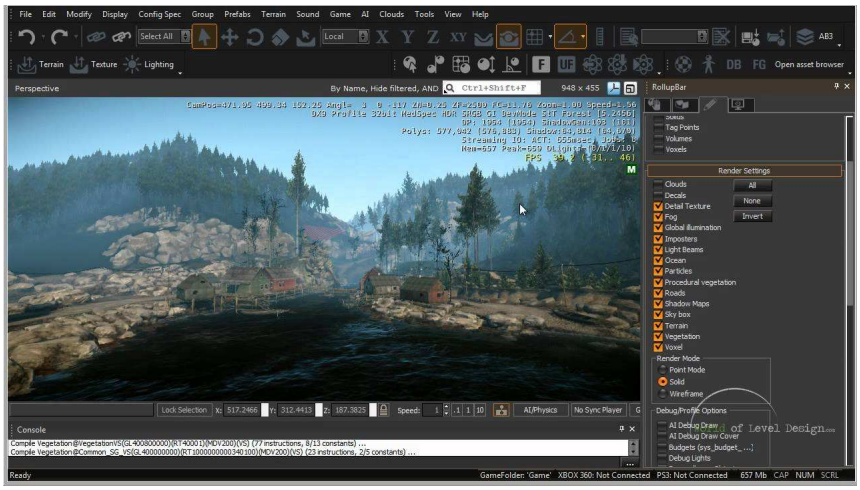
Unity 3D je další výborný engine, jehož první verze vyšla v roce 2005. Jde o opravdu intuitivní plně funkční nástroj pro vývoj her využívající programovacích jazyků JavaScript, C# a Boo. Tento engine je dostupný s editorem pro tvorbu jednotlivých scén aplikace, ve kterém je možné okamžitě hru spustit a otestovat. To umožňuje dynamicky ladit parametry hry, protože nastavení fyzikálních vlastností probíhá právě z tohoto editoru. Další funkcí

editoru je tvorba terénu. Pomocí štětce můžeme vytvářet výškový profil mapy a také nanášet textury, stromy a jiné. Součástí je defaultní editor MonoDevelop pro psaní skriptů. Tyto skripty jsou připojeny k objektům scény jejichž chování ovlivňují. Navíc veškeré veřejné proměnné scripů jsou nastavitelné z Unity editoru. Výslednou hru je možné sestavit pro desktopové platformy Windows, Mac, Linux/Steam, pro mobilní platformy iOS, Android, Windows Phone a pomocí Unity Web Player lze také nahrát hru na web.

Unity 3D má velkou komunitu vývojářů čítající přes 4 milióny v roce 2015 [15]. Také jsou dostupné online lekce téměř na všechny možné úlohy potřebné v jakékoli hře. K dispozici jsou knížky pro začínající i pokročilé vývojáře her v Unity 3D, např. Michelle Menard: Game development with Unity [6]. Unity 3D je volně dostupné pro nekomerční použití.

3.1.3 CryEngine

CryEngine je opravdu silný nástroj vyvíjený společností Crytek. Je navržen pro použití na PC, PlayStation a Xbox. Grafické schopnosti CryEngine předčí Unity, ale jsou na stejné úrovni jako Unreal Engine 4. Využívá k tom fyzikálně založené stínování. Tento engine opravdu umí renderovat nádherné scény z džungle od detailu vegetace po realistické odrazy světla od vody. Pro skriptování je použit jazyk c++. Stejně jako Unreal Engine a Unity 3D i tady bychom našli trigger tělesa, která jsou důležitá pro kontrolu dodržování pravidel.



Obrázek 3.3: CryEngine editor

I když je CryEngine velmi silným nástrojem, je obtížné se naučit jej správně použít. Další nevýhodou je jeho licence. Oproti Unreal Engine 4 a Unity 3D není zdarma, ale pro komerční využití mají lepší nabídku než výše zmíněné. Informace o produktu byli čerpány z [1].

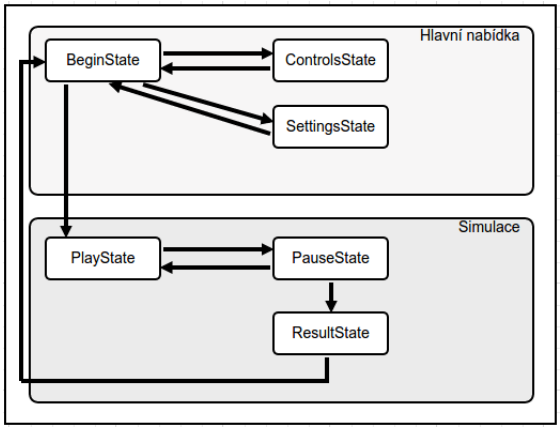
3.1.4 Zvolený engine

Všechny zmíněné enginy by byli dobrou volbou pro simulátor autoškoly. Unreal Engine 4 nám umožní vytvořit graficky realistický simulátor pro různé druhy platforem. Bohužel v době analýzy dostupných technologických prostředků nebyl volně dostupný, což je největší kritérium výběru. CryEngine má také úžasné realistické prvky grafiky, ale není snadné si jej osvojit a navíc není zdarma. Jako vhodný engine se jeví Unity 3D verze 5. Především z

důvodu obrovské komunity lidí používající tento engine, dostupných video lekcí a vzorových příkladů, no a v neposlední řadě jeho licence.

3.2 Architektura simulátoru

Architekturu simulátoru budeme stavět s ohledem na dříve zvolený herní engine Unity 3D. Simulátor rozdělíme na dvě scény, jak je znázorněno na obrázku 3.4. Pro řízené aplikace vytvoříme stavový automat, jehož princip vychází z kapitoly 8 knihy Learning C# by developing games with Unity 3D od Nortona [9]. Výhody použití stavového automatu jsou jednoduché rozšíření do budoucna, čisté a přehledné řízení. Protože Unity 3D nemá funkci pause, ale pouze zastavení pohybu, musíme ošetřit detekci vstupů ve stavech aplikace a přesně k tomu slouží stavový automat.



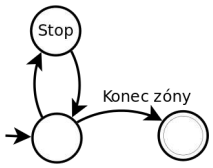
Obrázek 3.4: Stavový automat aplikace

Kontrola pravidel za použití nástroje Unity 3D spočívá v detekci kolizí vozidla s vhodně umístěným kolizním tělesem. Toto těleso bude typu *trigger* (spouštěč), které nevyvolá žádné fyzikální změny, ale pouze událost. Vyvolaná událost bude odchycena a podle typu kolidovaného tělesa se provede příslušná reakce. Způsob zpracování události a umístění kolizního tělese pro jednotlivá pravidla jsou popsány blíže v následujících podkapitolách.

3.2.1 Kontrola dopravních značek

Každá svislá dopravní značka bude opatřena kolizním tělesem *trigger* umístěném tak, aby zabralo celou šířku vozovky. Toto kolizní těleso značky musí zabírat celou šířky vozovky z důvodu možného předjíždění. Pokud ale zabírá celou šířku vozovky, musíme řešit problém s detekovaným směrem. V Unity 3D existují dva typy souřadnic. Souřadnice modelu se středem v bodě [0, 0, 0] a souřadnice místní (local) vztahující se na každý objekt modelu se středem v jeho počátku. Detekci směru příjezdu provedeme v místních souřadnicích značky. Do tohoto souřadnicového systému převedeme souřadnice pozice vozidla a porovnáme hodnotu na ose x. Je-li vozidlo před značkou bude tato souřadnice nabývat záporných hodnot. Pro rozpoznání dané značky bude značka označena příslušným tagem. Tag objektu je způsob označení shodných typů objektů a je možné jej získat při kolizích. Uvedeme si příklad na značce „*Zákaz zastavení*“ umístěné na kraji vozovky. Kolizní těleso dané značky

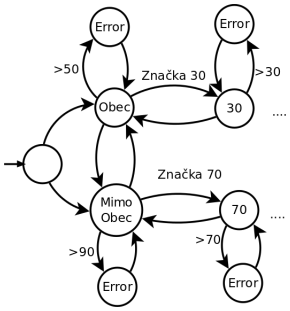
zasahuje do celé šíře vozovky. Projíždí-li vozidlo kolem značky detekujeme kolizi. Z kolidovaného tělese zjistíme, že se jedná o značku „*Zákaz zastavení*“. Zapneme tedy stavový automat, který kontroluje zastavení vozidla. Tento automat je znázorněn na obrázku 3.5. Ukončení platnosti značky je buď značkou „*Konec všech zákazů*“, nebo vzdáleným okrajem křižovatky.



Obrázek 3.5: Stavový automat kontroly zastavení

3.2.2 Kontrola rychlosti jízdy

Kontrola rychlosti jízdy bude provedena pomocí stavového automatu. Tento automat je znázorněn na obrázku 3.6. Protože kontrola rychlosti jízdy zahrnuje kontrolu minimální a maximální povolené rychlosti jsou stavy toho automatu dvojice {min., max.}. Podle počáteční polohy vozidla ve scéně se určí první přechod do jednoho ze stavů *obec* a *mimo obec*. Pokaždé když vozidlo míjí dopravní značku upravující maximální rychlost, přepne se tento automat do nového stavu, ve kterém setrvá do konce křižovatky, do značky ukončující zákaz, nebo do značky „*Obec*“. Pokud řidič jede rychleji, než je aktuální maximální povolená rychlost, automat přejde do stavu „*Error*“. V tomto stavu provedeme výpis hlášky a po dokončení výpisu přepneme automat zpět do předchozího stavu.



Obrázek 3.6: Stavový automat kontroly rychlosti

3.2.3 Kontrola jízdy křižovatkou a odbočování

Pro detekci křižovatky použijeme kolizní tělesa umístěná na jejím okraji v celé šířce vozovky. Tyto kolizní tělesa budou použita k detekci konce platnosti značek a odbočování. Jednotlivá kolizní tělesa okrajů křižovatky budou očíslována, abychom byli schopni určit směr jízdy křižovatkou a detekovat tak nedání znamení o změně směru jízdy. Toto číslování bude obzvlášť důležité pro detekci směru jízdy ostatních účastníků provozu. Jakákoli kolize s ostatními účastníky v rámci křižovatky bude považována za nedání přednosti v jízdě. Na obrázku 3.7a jsou kolizní tělesa označující hranici křižovatky znázorněna zelenou barvou.

Křižovatka může být dále opatřena značkami definující hlavní a vedlejší pozemní komunikaci. Je-li před křižovatkou značka „*Stůj, dej přednost v jízdě!*“, musí řidič zastavit vozidlo na takovém místě, odkud má do křižovatky náležitý rozhled. Toto místo bude definováno kvádrovým kolizním tělesem přiřazeným značce (červěně znázorněno na obrázku 3.7a).

(a) Kontrola jízdy křižovatkou

(b) Kruhový objezd

Obrázek 3.7: Kontrola jízdy křižovatkou a kruhovým objezdem

Kruhový objezd

Kruhový objezd je speciální typ křižovatky označený značkami „*Dej přednost v jízdě!*“, nebo „*Stůj, dej přednost v jízdě!*“ a značkou „*Kruhový objezd*“. Zde označíme jednotlivé výjezdy jako konec křižovatky. Do kruhového objezdu vložíme kolizní tělesa definující správný směr jízdy, jehož princip je popsán v podkapitole 3.2.1. Kruhový objezd je znázorněn na obrázku 3.7b.

3.2.4 Světelná křižovatka a její kontrola

Světelná křižovatka bude implementována jako cyklický přepínač dostupných směrů. Na začátku bude první směr volný a všechny ostatní uzavřený. Řízení křižovatky bude postupně zavírat aktuálně otevřený směr a otevírat následující. Proces uzavření směru je následující. Nejprve se přepne stav signalizačních zařízení daného směru ze zeleného signálu na žlutý a po té na červený. Jakmile je tento směr uzavřen, začne se otevírat další směr. V prvním kroku se rozsvítí červené a žluté světlo najednou signalizující „*Pozor!*“. Poté se rozsvítí zelená světlo „*Volno*“. Křižovatka bude moci mít n směrů, každý obsahující dvě signalizační zařízení.

Při průjezdu vozidla kolem signalizačního zařízení budeme detekovat stav daného směru. Je-li tento směr uzavřený (svítí červené světlo), vypíšeme řidiči zprávu o jízdě na světelný signál „*Stůj!*“. Je-li řidič při rozvícení žlutého signálu v takové blízkosti, že není schopen bezpečně zastavit, může pokračovat v jízdě [16]. Tato situace bude ošetřena rozšířením kolizního tělesa více před signalizační zařízení. Pokud vozidlo vjede do tohoto kolizního tělesa při zeleném signálu a vyjede za žlutého signálu, detekovali jsme tuto situaci a nebude brána jako přestupek.

3.2.5 Kontrola zastavení a stání

Zastavení se bude chápat jako pokles rychlosti na nulu. Řidič nesmí stát na přechodu pro chodce a ve vzdálenosti kratší 5m před i za ním. Za tímto účelem na každý přechod a do dané vzdálenosti vložíme kolizní těleso. Pokud vozidlo zastaví v rámci tohoto tělesa jde o přestupek, který zobrazíme řidiči. Stejný princip použijeme na zákaz zastavení v křižovatce.

3.3 Ostatní účastníci provozu

Do simulátoru zahrneme další účastníky provozu, kteří budou sloužit pro kontrolu průjezdu křižovatkou. Tito účastníci budou nuceni dodržovat pravidla provozu, abychom byli schopni detekovat nedání přednosti v jízdě. Pokud bychom měli takového účastníka, který nedodržuje pravidla, museli bychom řešit detekci vyníka nehody. Všechny kolize s těmito řidiči budou považovány za ohrožení jiného řidiče. Tento účastník bude dodržovat následující pravidla:

1. rychlost jízdy
2. jízdu křižovatkou
3. odbočování
4. světelnou křižovatku

Protože hlavním cílem této práce je kontrola dodržování pravidel a tito účastníci pouze slouží jako pomůcka pro kontrolu dodržování jízdy křižovatkou, budeme tento problém řešit co nejjednodušeji. Tedy ostatní řidiči nebudou schopni sami inteligentně objet překážku. Naopak budou se řadit za sebe a čekat dokud se překážka neuvolní. Při návrhnu města na toto budeme pamatovat. Můžeme tedy říct, že jedinou překážkou bude řidič simulátoru. Zde je potencionální problém, kdy řidič zastaví na okraji vozovky a bude znovu vyjíždět. V tomto případě dává přednost všem vozidlům jedoucích po pozemní komunikaci. V našem případě tito řidiči dávají přednost jemu, respektive čekají na odstranění překážky. Tento nedostatek odstraníme v další verzi, ve které se zaměříme více na ostatní účastníky.

Vlastní řešení pohybu ostatních účastníků provozu bude pomocí předem definované trati. Každý řidič bude mít přiřazenou trasu, kterou bude projíždět stále dokola. Trasa je důležitá pro správné projetí křižovatkami a je definovaná pomocí checkpoint bodů. Po spuštění simulace každý řidič zamíří ke svému prvnímu checkpointu a pokud je dostatečně blízko začne směřovat k dalšímu checkpointu. A tak jede pořád do kola.

Při projíždění trasy se řidič bude dívat dopředu na vzdálenost účinné brzdné dráhy plus povolenou vzdálenost vozidel, takže bude schopen včas reagovat na různé situace. Jednou takovou situací může být pomalu jedoucí vozidlo. V tomto případě sníží rychlost a drží si odstup od vozidla před ním. Tímto způsobem budou detekovány různé značky a křižovatky.

3.3.1 Jízda ostatních účastníků křižovatkou

Jak bylo zmíněno dříve ostatní účastníci pojedou po definované trase a budou se dívat před sebe, čímž budou detekovat různé značky pomocí jejich kolizních těles (viz. podkapitola 3.2). Jízda křižovatkou bude začínat detekcí hranice křižovatky, nebo značkou upravující přednost. V případě že detekuje pouze hranici křižovatky jedná se o křižovatku s předností vozidel zprava. Vozidlo musí dát přednost všem řidičům přijíždějích zprava. Pokud nějaký řidič přijíždí zprava, pak začne vozidlo brzdit, jinak pokračuje v jízdě. Pokud kromě hranice křižovatky vozidlo detekuje značku „*Stůj, dej přednost v jízdě!*“, začne brzdit tak, aby zastavilo před hranicí křižovatky. V okamžiku kdy zastaví začne detekovat přijíždějící vozidla ze všech směrů v křižovatce. Podobný princip bude použit pro značku „*Dej přednost v jízdě!*“ s rozdílem zastavení. Vozidlo při detekci této značky detekuje přijíždějící vozidla. Pokud žádné takové není pokračuje v jízdě.

Při odbočování je řidič povinen dát znamění o změně směru jízdy. Z toho plyne, že vozidlo musí být schopno detekovat změnu své trasy. Za tímto účelem bude jeho trasa muset obsahovat jeden checkpoint uvnitř křižovatky, kde dochází ke změně směru jízdy. Při přijíždění ke křižovatce vozidlo rozpozná vstupní hraniční těleso. Potom provede test, jestli další checkpoint je uvnitř křižovatky s daným checkpointem. Pokud ano, podívá se od tohoto checkpointu směrem k dalšímu checkpointu. Při tom detekuje výstupní hraniční těleso. Pomocí indexů těchto těles určí změnu směru jízdy a dává znamení ostatním řidičům.

3.3.2 Detekce přijíždějících vozidel

Detekce přijíždějících vozidel ke křižovatce provedeme pomocí kolizních těles označujících hranice křižovatky. Přijíždějící vozidlo ke křižovatce detekuje všechny kolizní tělesa v rámci této křižovatky. Jednotlivá kolizní tělesa jsou označena indexem a maximální stupeň křižovatky je 4 vjezdy. Pokud vozidlo požaduje test pouze na řidiče přijíždějící zprava, nalezne tak snadno odpovídající hranici křižovatky. Potom pro každý požadovaný směr vezmeme hraniční těleso a podíváme se jeho směrem. Tento směr zjistíme převedením místní souřadnice osy x hraničního tělesa do souřadnic modelu. Pokud detekujeme přijíždějící vozidlo, další směry není třeba prohledávat.

3.4 Uživatelské rozhraní

Dobré uživatelské rozhraní začíná pochopením uživatelů co mají rádi, proč používají daný software a jak s ním mohou interagovat [14]. Především interakce uživatele se simulátorem bude důležitá, a proto by jsme tomuto tématu měli věnovat více pozornosti.

Prvním krokem v návrhu dobrého rozhraní je definice cíle, kterého chceme dosáhnout. Cílem simulátoru autoškoly je simulovat dopravní situace a připravit tím potencionální budoucí řidiče na reálné situace v dopravním provozu. Důležitou složkou rozhraní je také zpětná vazba například při překročení rychlosti, nebo jízdě na červenou.

Uživatelské rozhraní rozdělíme na tři základní části (tyto části pak mohou být jednotlivé scény programu Unity 3D):

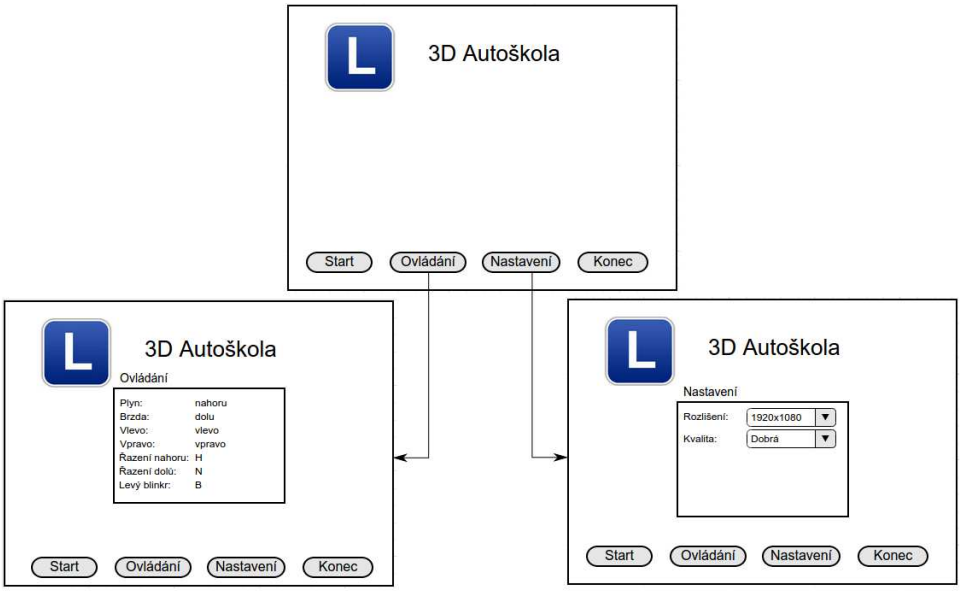
1. hlavní nabídku
2. rozhraní simulace
3. zobrazení výsledků

3.4.1 Hlavní nabídka

Hlavní nabídka bude startovní obrazovka simulátoru. Cílem této části bude rychlá orientace v aplikaci. Uživatel musí být schopen spustit simulaci, provést dodatečnou konfiguraci a přizpůsobit si ovládání simulace jak je zvyklí.

Návrh hlavní nabíky je zobrazen na obrázku 3.8. Nejprve se zobrazí obrazovka s odkazy na spuštění simulátoru, nastavení ovládání, všeobecná nastavení simulace a ukončení aplikace. Po výběru některé z možností se nad těmito odkazy zobrazí panel obsahující požadovaná data. Obrazovka pro nastavení ovládání v aktuální verzi bude obsahovat pouze zobrazení konfigurace ovládání simulace. V další verzi bude tento nedostatek odstraněn a uživatel bude mít možnost upravit ovládání. Další možností hlavní nabídky je nastavení.

Zde je prostor pro různou konfiguraci simulátoru jako je rozlišení, kvalita renderování, nastavení hlasitosti zvuku, režim celé obrazovky a jiná. Výsledný simulátor autoškoly bude obsahovat alespoň nastavení kvality a rozlišení. Veškerá nastavení se v aktuální verzi nebudou ukládat na disk do uživatelského prostoru, ale budou použita pro aktuální spuštěnou simulaci. Při opětovném spuštění aplikace budou použity předdefinovanmé hodnoty.

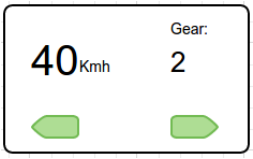


Obrázek 3.8: Hlavní nabídka

3.4.2 Rozhraní simulace

Rozhraní pro interakci uživatele se simulátorem je nejdůležitější část aplikace. Našim cílem je zobrazit uživateli všechny potřebné informace o jízdě a umožnit mu interakci co nejvíce podobnou reálnému stavu. U dříve analyzovaných simulátorů si můžeme povšimnout dvou typů pohledů na simulaci. Prvním je pohled z vozidla a druhým pohled z vnějšku vozidla.

Pohled z vozidla má nejblíže reálnému pohledu. Zde uživatel uvidí na palubní desku vozidla, která bude obsahovat ukazatele rychlosti, otáček motoru, indikátory směru při odbočování. Ostatní indikátory klasické palubní desky jako je ukazatel teploty chladící kapaliny, stavu paliva v nádrži, indikátor nedostatku oleje v motoru a jiné nebudou implementovány, protože zde nehrají žádnou roli. Při návrh rozhraní bereme v úvahu výstupní zobrazení na klasický monitor. Proto je nutné umožnit řidiči simulovat otočení hlavy, aby mohl vidět do zpětného zrcátka, které nejsou dostupné při pohledu přímo z vozidla.

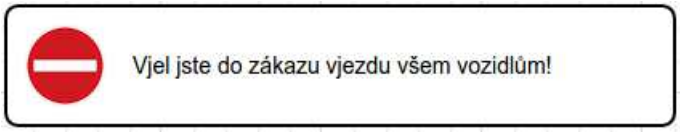


Obrázek 3.9: Externí zobrazení palubní desky

Při pohledu z vnějšku vozidla není palubní deska viditelná a proto musíme provést zobrazení jiným způsobem. Nejvhodnější se jeví použít zaběhlý způsob zobrazení ukazatele rychlosti v dolní části obrazovky. K tomuto ukazateli přidáme indikátor znamení o změně směru jízdy a zařazeného stupně.

Zobrazení zpráv

Informace o porušení pravidel budeme zobrazovat do dialogového panelu v horní části okna simulace. Do tohoto panelu zobrazíme značku, kterou uživatel porušil společně s doprovodným textem o jakou značku se jedná. Bude-li nutné zobrazit více než jednu zprávu, zobrazíme je pod sebe. Panelům nesoucích tyto zprávy přidáme efekt pomalého zobrazení a skrytí. Aby nedošlo k zakrytí výhledu z vozidla omezíme počet současně zobrazených zpráv. Zprávy, které budou navíc se pouze uloží a budou zobrazeny při výpisu jízdy.



Obrázek 3.10: Zpráva o porušení pravidla

3.4.3 Ovládání simulátoru

Nejvhonější ovládání simulátoru by byl externí volant a řadící páka. Bohužel ani tyto prvky by nestačili bez použití více monitorů. Ty jsou potřeba především k výhledu do křižovatky. V navrženém simulátoru musí uživatel být schopen provést následující úkony:

1. rozjet se a zabrzdit
2. změnit směr jízdy
3. řadit převodové stupně, nebo ovládat automatickou převodovku
4. dát znamení o změně směru jízdy
5. rozhlédnout se v křižovatce
6. přepnout pohled

Východní nastavení bude dostupné ke čtení z hlavní nabídky. Při navrhování ovládání pro rozhlížení do křižovatky se zdálo vhodné využít počítačové myši jako vstupu. Jednou rukou by uživatel ovládal pohyb vozidla a druhou by se rozhlížel. Problém je nedostatek dostupných tlačítek přímo na myši. V této variantě chybí kombinace vstupů pro indikaci změny směru a řazení.

Defaultní nastavení ovládání bude následující. Přidání plynu *šipka nahoru*, brždění *šipka dolů* a zatáčení *šipka vlevo* a *šipka vpravo*. Pro řazení použijeme tlačítka *H* a *N*. Pokud je použita automatická převodovka bude možné řadit pouze z *neutrálu* první stupeň, nebo *zpátečku*. Znamení o změně směru jízdy provedeme klávesami *B* a *M*. Vypnutí bude automatické pouze pokud řidič vyjíždí z křižovatky. V ostatních případech bude nutné vypnout znamení manuálně. Rozhlížení se bude provádět pomocí kláves *G* a *J*. Zde nebude uživatel limitován úhlem otočení. Přepnutí pohledu provedeme klávesou *Q*.

3.4.4 Zobrazení výsledků

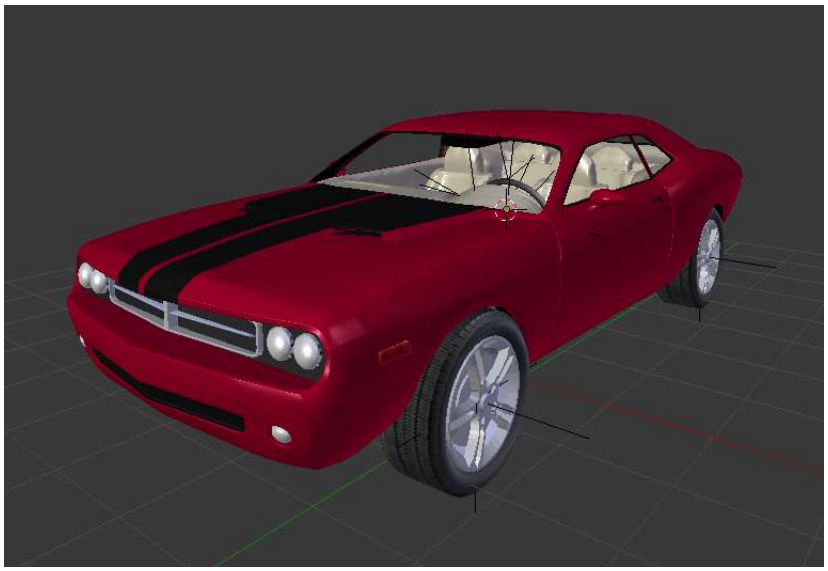
Zobrazení výsledků bude obrazovka před ukončením simulace. Uživatel bude schopen prohlédnout si přestupky, kterých se dopustil během jízdy. Mělo by zde také být vysvětlení ze zákona, čeho se dopustil. Jednoho přestupku se mohl řidič dopustit vícekrát, tudíž zde musí být celkový počet přestupků, nebo seřazený výpis podle doby vzniku. Dobrou funkcí do budoucna by byla možnost se proklikat na zpětné zobrazení dané situace. Především při řešení situací na křižovatkách. Protože pouze "*Nedal jste přednost v jízdě*"není moc konkrétní. V Česku platí bodovací systém řidičů a proto by bylo vhodné vložit počet ztracených bodů.

Kapitola 4

3D modely

V této kapitole se budeme věnovat modelům použitých v implementovaném simulátoru autoškoly. Modely budeme vytvářet případně upravovat pomocí nástroje na editaci 3D modelů Blender. Zde zmíníme podstatný rozdíl zobrazení modelů oproti nástroji Unity 3D, které používá pro zobrazení modelů levotočivý souřadnicový systém. Naproti tomu Blender a mnoho jiných nástrojů jako např. AutoCad používají pravotočivý souřadnicový systém. Z toho důvodu mohou být v Unity prohozené souřadnicové osy a je důležité na toto myslet při implementaci.

4.1 Model vozidla



Obrázek 4.1: Model Dodge Challenger 2008

Jako model vozidla byl zvolen Dodge Challenger model 2008. Použitý model je dostupný z [7]. Tento model obsahuje opravdu moc vrcholů z tohoto důvodu jsou nevyhnutelné úpravy. Nejprve provedeme úpravu kol modelu. Tyto kola jsou v dostupném modelu detailně modelované a je nevyhnutelné vymodelovat kola nová. Aby vypadala reálně, byla na ně nanesena textura. Dalšími nezbytnými úpravami jsou vytvoření oboustranné plochy

karoserie především uvnitř automobilu, oddělení volantu od interiéru, vyhlazení návaznosti interiéru na karoserii a úprava palubní desky. Nakonec převedeme model do levotočivého systému. Správného zobrazení souřadnicových os v Unity 3D docílíme rotacemi objektů modelu v editačním módu nástroje Blender.

4.1.1 Palubní deska

Při zobrazení pohledu z vnitřku vozidla bude palubní deska zabudována do daného modelu, jak je vidět na obrázku 4.2. Proto musíme model automobilu dále upravit. Zarovnáme zadní část palubní desky a přidáme textury pro rychloměr a otáčkoměr. Dále vložíme ručičky pro ukazatel rychlosti a otáček. Pro tyto ručičky nejprve vytvoříme rodičovský objekt, který bude definovat osu rotace tak, abychom později mohli jednoduše provést zobrazení hodnoty. Nakonec v Unity vložíme obrazovky zobrazující aktuální převodový stupeň a indikátory změny směru jízdy.

Obrázek 4.2: Palubní deska

4.2 Model ostatních účastníků

Pro model dalších účastníků provozu bude použit model vozidla z příkladu implementace pohybu závodního automobilu dostupný v Unity Assests store. Jde o model vozidla ve formátu *fbx*, který je dále potřeba upravit pomocí nástroje Blender. Úprava spočívá v mapování materiálů pro směrovky vozidla, protože od těchto účastníků požadujeme správné chování a informování uživatele 3D Autoškoly o svých počinech na křižovatkách. Výsledný model je znázorněn na obrázku 4.3.

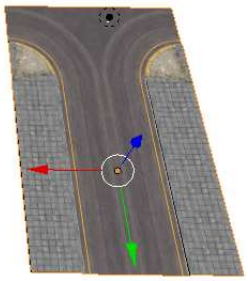
Obrázek 4.3: Model účastníků provozu

4.3 Model města

Jako model města bude použit dostupný model bloku #6 města New York dostupný z Unity assets store [8]. Tento model obsahuje vcelku detailní návrh ulice. Jsou zde dostupné modely tří různých stromů, pouliční světla, světelná zařízení křižovatek, zařízení ulic jako je autobusová zastávka, odpadkové koše a spoustu dalších modelů pro zpříjemnění výsledku simulátoru. Pro model města jsou ale důležité hlavně budovy, proto kompozici ulice necháme na konec práce.

Protože zvolený model města obsahuje světelná zařízení křižovatek, použijeme tyto ve výsledné aplikaci. Bohužel je nutná další úprava z důvodu nekompatibility s evropským stylem. V tomto modelu jsou zařízená stavěná pro umístění do středu křižovatky. V nástroji Blender provedeme patřičnou úpravu umístění světelného zařízení na pouliční sloupek.

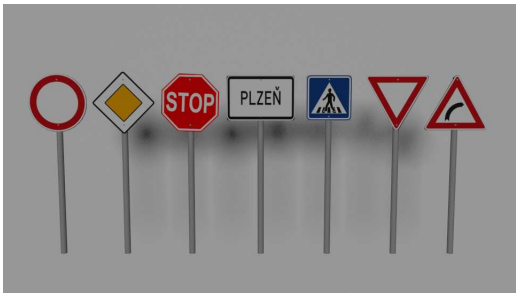
4.3.1 Model vozovky



Obrázek 4.4: Model vozovky

Model vozovky vytvoříme pomocí balíčku *Steet System* [13], který obsahuje díly vozovky pro komplexnější tvorbu. Nalezneme zde textury pro silnici, chodník a jiné. Balíček obsahuje díly pro sestavení ortogonálního silničního systému. V balíčku chybí kruhový objezd. Použijeme tedy nástroj Blender a doplníme sadu o kruhový objezd. Na obrázku 4.4 je znázorněn jeden díl tvořící křižovatku typu „T“.

4.4 Modely značek



Obrázek 4.5: Model svislých značek

Model svislých dopravních značek vytvoříme pomocí nástroje blender. Na obrázku 4.5 jsou znázorněné základní tvary značek. Pro každý tvar připravýme vlastní šablonu modelu. Značku vymodelujeme z tyčky a tvaru konkrétní značky. Protože jeden tvar značky je společný pro různý typ značek, provedeme pouze mapování základních tvarů značek na vzorovou texturu. Pro použití vytvořeného modelu pouze stačí nastavit konkrétní texturu značky. Tyto textury jsou k dispozici ve formátu .ai (formát pro Adobe Illustrator) [4]. Na obrázku 4.5 jsou znázorněny základní tvary svislých značek.

Kapitola 5

Implementace simulátoru

Tato kapitola se věnuje implementaci simulátoru v herním enginu Unity. Zde se budeme věnovat způsobu psaní skriptů, sestavování prostředí simulace a také grafickému uživatelskému rozhraní. Nejprve si rozebereme strukturu projektu do adresářů a základní metody psaní skriptů chování. Dále popíšeme způsob kontroly prioritizovaných pravidel. A nakonec se podíváme na implementaci pohybu a chování ostatních účastníků provozu.

5.1 Struktura projektu a práce s Unity 3D

Práce s Unity 3D je v nativním editoru, který si stáhneme a nainstalujeme. Tento editor obsahuje panely pro náhled hry, editaci scény, hierarchii, konfiguraci objektů a manažer souborů viz. obrázek 3.2. Veškeré soubory potřebné pro sestavení hry jsou umístěny ve složce *Assets* v adresáři projektu. Při vytváření hry sestavujeme jednotlivé obrazovky z připravených modelů. Unity editor pracuje pouze nad konkrétní scénou, pro editaci jiné se musíme přepnout do této scény. Prvním krokem je vytvoření prostředí pro simulaci a potom hlavní nabídku. Scény projektu se nacházejí ve složce *_Scenes*.

Jak bylo zmíněno v podkapitole 3.1.2, v Unity můžeme využít programovací jazyk C#. Důvodem použití tohoto programovacího jazyku je jeho silná dokumentace dostupná online a různé kolekce datových struktur, jako je například *List* a *Dictionary*. Veškeré skripty se nacházejí ve složce *Scripts*. Pomocí skriptů upravujeme chování objektů. Ke každému objektu scény můžeme přiřadit různé komponenty jako je animátor, collider a vlastní skript. V jazyku C# se jedná o třídu, která musí být potomkem třídy `MonoBehaviour`, definující prázdné metody, které Unity provolává na příslušných místech. Základní metody jsou:

1. Awake - Jedná se o metodu, která je volaná při vytvoření objektu, ke kterému je přiřazená.
2. Start - Unity volá tuto metodu nad objektem po inicializaci scény ještě před metodou Update. Typicky se zde provádí inicializace závislostí na jiné objekty.
3. Update - Je volána před každým překreslením okna. Zde se nacházejí veškeré grafické aktualizace scény jako je např. otočení kola apod..
4. LateUpdate - Tato metoda se volá až po metodě Update. Slouží především pro aktualizaci objektu v závislosti na chování jiného objektu.
5. FixedUpdate - Unity volá tuto metodu pro aktualizaci výpočtu fyziky.

Každý objekt scény je v Unity reprezentovám třídou `GameObject`. Tato třída obsahuje definici pozice pomocí proměnné třídy `Transform`, příznak aktivity objektu a jeho *tag*, který jsme zmiňovali v podkapitole 3.2.1. Dále obsahuje také statické metody pro vyhledávání objektů a spoustu dalších.

Veškeré modely 3D Autoškoly jsou umístěny ve složce *Models*. Unity podporuje nativní formát z nástroje Blender, který jsme použili pro úpravu modelů. Po importu modelu do Unity projektu je možné model editovat v nástroji Blender a změny se tak přímo projeví v aplikaci simulátoru. Každý model obsahuje podsložku *Materials* nesoucí veškeré materiály použité na daném modelu.

Pro jednoduché sestavování modelu prostředí jsou použity tzv. *prefab* objekty. Jde o šablony stejných objektů scény. Změny provedené nad touto šablonou se projeví ve všech objektech generovaných pomocí daného *prefab* objektu. Vytvoření *prefab* se provede pomocí drag & drop objektu z okna hierarchie do okna s adresářem *Prefabs*, kde se nacházejí všechny šablony aplikace.

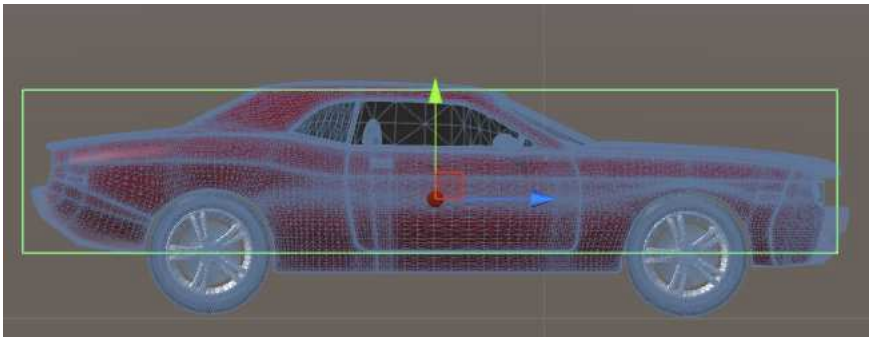
Dále v aplikaci máme složku *Textures* pro všechny použité obrázky. Unity podporuje celou řadu formátů obrázků, které sjednocuje do jedné datové struktury třídy `Texture`. Všechny zvuky jsou v adresáři *Sounds*, animace v adresáři *Animations* a nakonec standardní Unity balíček pro terén se nachází v adresáři *Standard Assets*.

5.2 Fyzikální model automobilu

Pro použití fyziky v Unity slouží hlavní komponenta *RigidBody*. Má-li objekt přiřazenou tuto komponentu, pak se na něj vztahují fyzikální výpočty. V tomto stavu je možné na objekt aplikovat sílu a gravitaci. Vložíme model vozidla do scény a přiřadíme mu *RigidBody* komponentu. V konfiguraci komponenty nastavíme parametr *mass* (z ang. hmotnost) na hodnotu 1882 kg (pohotovostní hmotnost dodge challenger [5]), parametr drag (odpor vzduchu) na hodnotu 0,01 a zaškrtneme parametr *Use Gravity*. Takto nastavené vozidlo by nám po spuštění aplikace propadlo terénem, protože není nastaven výpočet kolizí objektu. Proto přidáme kolizní těleso komponentu typu *Collider*, která způsobí interakci mezi jinými objekty s přiřazeným kolizním tělesem. Unity nabízí různé typy kolizních těles pro přesné definování tvaru, jako je *Mesh Collider*, jehož nevýhodou jsou náročné výpočty kolizí, nebo *Capsule Collider*. U vozidla je použit *Box Collider*, který je dostačující pro definici tvaru automobilu a optimální pro výkon aplikace. Pokud koliznímu tělesu nastavíme parameter *Is Trigger* na hodnotu `true`, pak se jedná pouze o spouštěč události a nikoli o fyzickou kolizi. Automobil nesmý propadnout terénem a proto parameter *Is Trigger* má hodnotu `false`. Takto je hotova základní konfigurace fyzikálního modelu vozidla.

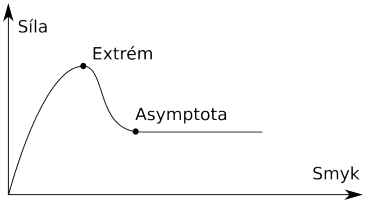
5.2.1 Skript pro ovládání vozidla

Ovládání a pohyb vozidla je implementován ve vlastním skriptu přiřazený vozidlu. Skript obsahuje třídu `CarController` implementující chování objektu. Zde je implementována veškerá interakce uživatele s vozidlem. Pro pohyb vozidla postačí aplikovat sílu daným směrem na *RigidBody* komponentu. Unity problém pohybu vozidla řeší trochu jinak. Jedním typem kolizních těles je tzv. *WheelCollider*. Toto kolizní těleso je speciálně navržené pro kola vozidel. Veškerá logika pohybu, změny směru a tlumičů vozidla je zde přímo implementována. Na druhou stranu je komplikované tento Collider správně nastavit. Dále je v tomto skriptu implementována aktualizace grafických component vozidla, kterými jsou směrovky, světla brzd a palubní deska.



Obrázek 5.1: Kolizní tělesa automobilu

Po incializaci scény se provede inicializace vozidla, ve které se nastavý všechny konfigurovatelné proměnné. Nejprve se vytvoří *WheelCollidery* pro všechna kola. Tento collider se skládá z pružiny tlumičů a dvou křivek tření. U pružiny nastavujeme její sílu, délku, pozici a tlumící sílu. První křivka tření definuje dopředné tření kol, tedy jak moc kolo drží při akceleraci resp. brždění. Druhá křivka tření definuje boční tření, tedy jak moc se vozidlo smýká při změně směru. U těchto křivek se definují dva základní body viz. obrázek 5.2. Prvním bodem je *extrém*, do kterého kolo stále drží na vozovce. Při překročení tohoto bodu



Obrázek 5.2: *WheelCollider* křivka tření

je vozidlo v kontrolovaném smyku. Pokud vozidlo překročí druhý bod křivky nachází se v nekontrolovatelném smyku. Nastavení těchto bodů má velký vliv na ovladatelnosti vozidla. Dalším krokem inicializace je vypnutí animace směrovek vozidla a nastavení rychlostí pro rychlostní stupně jejichž počet je konfigurovatelný z Unity editoru. Maximální rychlost je nastavena na 174 km/h. Z toho rozložení rychlosti na převodové stupně je popsáno v tabulce 5.1.

Stupeň	**Rychlost** $[km/h]$
R	17
N	0
1	17
2	39
3	68
4	131
5	174

Tabulka 5.1: Tabulka rychlostí převodových stupňů

Vozidlo je rozpohybováno pomocí *WheelCollider* komponenty. Nejprve se v metodě `Update` přečtou vstupní parametry pro plyn a směr jízdy pomocí metody `Input.GetAxis()`. Jak bylo zmíněno v kapitole 5.1, metoda `FixedUpdate` slouží pro aktualizaci fyziky. Z toho důvodu se dříve načtené hodnoty aplikují v této metodě. Roztočení kol se provede pomocí parametru *motorTorque*, který představuje kroutící sílu motoru. Chceme-li změnit směr otáčení změníme znaménko kroutící síly. Pro změnu směru vozidla se nastaví úhel natočení kol do proměnné *steerAngle*. Pro snadné ovládání vozidla se při větší rychlosti omezí maximální natočení kol z 25° na 15°.

U vozidla je implementována automatická převodovka, kterou je možné v nastavení změnit na manuální. Rychlost jízdy se získá z parametru *velocity* komponenty *RigidBody*. Výsledek je vektor rychlosti, jehož amplituda je aktuální rychlost v metrech za sekundu.

Nyní je automobil schopen pohybu a je potřeba provést znázornění některých indikátorů jako je např. rotace kol, natočení volantu, rozsvícení brzdových světel a další. Veškeré grafické změny se provádějí v metodě `Update`. Z rychlosti a koeficientu rychloměru se vypočte úhel otočení ukazatele a provede se jeho rotace pomocí komponenty `Transform`, která slouží pro všechny změny pozice a rotace objektů scény. Steným způsobem se natočí volant automobilu. Kolo je modelováno jako dva objekty v rodičovském vztahu, kde rodič slouží pro natočení kola při změně směru jízdy a potomek rotuje v závislosti na rychlosti a směru jízdy. Dále se zobrazuje indikátor změny směru jízdy na palubní desce a modelu vozidla. Palubní deska je implementována pomocí *Canvas* objektu, do kterého lze vložit komponenty uživatelského rozhraní. Směrovky jsou zde obrázky šipek s animátorem, který provádí blikání. Zapnutí těchto směrovek je pouze aktivování komponenty animátor. Zobrazení směrovek z vnějšku vozidla je pomocí materiálu, kterému je přiřazen shader. Tento shader obsahuje vstupní parameter intenzity barvy. Při blikání se pouze mění hodnota této intenzity. Pro každý směr musí existovat vlastní materiál, jinak by se nastavení intezity barvy projevilo všude. Blikání je zajištěno pomocí paralelního vlákna s časovačem.

5.2.2 Implementace kamer simulace

V simulátoru jsou implementované dva různé druhy pohledu. Prvním je pohled na palubní desku vozidla a druhým je pohled na vozidlo z vnějšku. Každý pohled je implementovám vlastní kamerou. Přepínání mezi kamerami je řízeno stavovým automatem aplikace (podkapitola 5.6) a je řešeno aktivováním a deaktivováním jednotlivých kamer. Každou kameru je možné natočit pro lepší výhled do křižovatek.

Palubní deska

Palubní deska obsahuje analogový rychloměr a otáčkoměr, display pro zobrazení převodového stupně a display ukazatelů změny směru jízdy. O aktualizaci údajů palubní desky se stará skript *CarController*.

Tento pohled je řízen skriptem *InsideCamera*. Důležitým atributem skriptu je těžiště vozidla, které kamera následuje. V metodě `LateUpdate` se kontroluje stisknutí tlačítek pro rotaci kamery kolem své osy. Natočení kamery pro výhled z vozidla je definované pomocí kvaternionu, který se při stisku klávesy pootočí. Umístění kamery do scény je v závisloti na pozici těžiště vozidla a předdefinovaného vektoru určujícího posunutí oproti těžišti.

Dalšími prvky výhledu z vozidla jsou zpětná zrcátka (jedno vnitřní zrcátko a dvě vnější). Zrcadlení je provedeno pomocí shaderu a skriptu [10]. Tento skript vytvoří novou kameru, jejíž pozici nastaví podle přiřazeného objektu. Směr pohledu kamery je podél osy $-z$ daného objektu. Klíčovým prvkem zrcadla je `RenderTexture`. Jde o texturu, na kterou vykresluje

Obrázek 5.3: Palubní deska

kamera snímající pohled od zrcadla. Dále zde je shader, který přebere tuto texturu a vykreslí při renderování. Touto kombinací skript, shader je docíleno zrcadlení.

Vnější pohled

Vnější pohled obsahuje display, na který se vykreslují informace shodné s palubní deskou (rychlost a převodový stupeň). Směrová světla jsou zobrazena přímo na modelu vozidla. O aktualizaci těchto informací se stará skript *CarController*.

Pohled je řízen skriptem *OutsideCamera*, který obsahuje referenci na těžiště automobilu. Na základě toho objektu se umístí kamera. Otočení pohledu je implementováno v metodě `LateUpdate`. V tomto pohledu není zpětné zrcátko, ale uživatel má možnost otočit pohled směrem dozadu.

5.2.3 Skript ozvučení vozidla

Pro ozvučení vozidla je použit skript z příkladu dostupného v Unity Assets store [12]. Pomocí nástroje Audacity byli vytvořeny zvukové stopy použité v 3D Autoškole. Daný skript byl upraven na novou verzi Unity a nakonfigurován s využitím vytvořených zvukových stop. Úprava skriptu spočívá v čtení rychlosti a převodového stupně automobilu pomocí skriptu *CarController* a napsání metody pro čekání na nový snímek.

5.3 Tvorba prostředí simulace

Pro tvorbu prostředí simulace byli použity modely popsané v kapitole 4. Tyto modely se nacházejí v adresáři *Models*. Město bylo navrženo tak, aby zahrnovalo místa pro kontrolu prioritizovaných pravidel z podkapitoly 2.2. Návrh města je zobrazen na obrázku 5.4.

Jednotlivé části města jsou vytvořeny pomocí prefab objektů, připravených pouze k umístění ve scéně. Tyto objekty se nacházejí ve složce *Prefabs*. Jsou zde implementovány některé dopravní značky, bloky pro sestavení pozemní komunikace, kompletní křižovatky se značkami, kruhový objezd, řízení světelné křižovatky a další. Sestavení komunikace znamená umístění jednolivých prefab bloků za sebe. Bohužel pomocí těchto bloků lze sestavit pouze ortogonální síť pozemní komunikace.

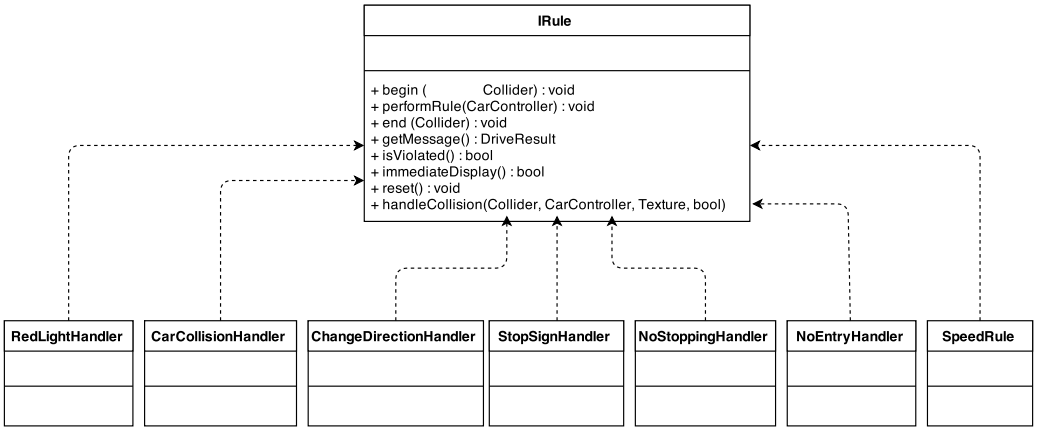
Další součástí prostředí simulace je terén. Unity editor podporuje snadné vytváření terénu pomocí štětce. V editoru terénu se štětcem nanáší výšková mapa a také textury terénu. Ve standardním balíčku se nacházejí různé druhy textur pro zeleň, pěšiny a vodní

Obrázek 5.4: Prostředí simulátoru

plochy. Model terénu je navržen tak, aby se město nacházelo na rovné části pro snadné modelování a v okolí jsou menší kopce.

5.4 Kontrola dodržování pravidel

Kontrola pravidel je implementována jako komponenta automobilu ve skriptu *TrafficRuleController.cs*. Pro kontrolu pravidel jsou použita pomocná kolizní tělesa typy *trigger*. Tyto kolizní tělesa definují začátek a konec platnosti daného pravidla. Kolize s triggery jsou detekovány v metodách `OnTriggerEnter` a `OnTriggerExit`. Pro budoucí rozšíření pravidel byli nalezeny skupinu shodného typu kontroly a implementováno jejich společné rozhraní *IRule* viz. diagram tříd obrázek 5.5.



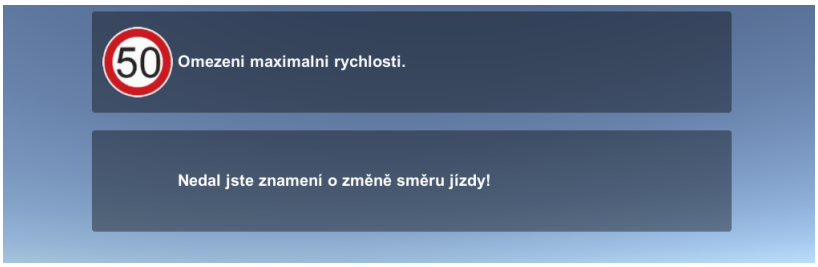
Obrázek 5.5: UML diagram kontroly pravidel

TrafficRuleController nese kolekci pravidel typu `Dictionary`. Jedná se o kolekci dvojic klíč a hodnota. Klíčem je tag označující detekované kolizní těleso a hodnotou je automat kontrolující dané pravidlo. Princip zpracování kontroly pravidel je následující. V systému existuje více druhů platnosti pravidel. Některá pravidla se uplatňují okamžitě při detekování vstupu do kolizního tělesa (např. značka „*Zákaz vjezdu*“). Některá jsou platná neustále, přičemž mohou záviset na detekci různých značek (např. kontrola rychlosti jízdy) a některá jsou platná pouze v rámci kolizního tělesa (např. kontrola zastavení na značce „*Stůj,*

dej přednost v jízdě!“). Pro každé typy existuje ve skriptu vlastní kolekce. Při detekování kolize se z příslušné kolekce vytáhne pravidlo a začne se aplikovat. Jde-li o pravidlo platné v rámci kolizního tělesa, tak se při vstupu aktivuje pravidlo tím, že se vloží do kolekce aktivních pravidel. Tyto aktivní pravidla se pravidelně kontrolují v metodě `Update`. Kontrola jednotlivých pravidel je popsána v následujících podkapitolách.

5.4.1 Zprávy o přestupcích

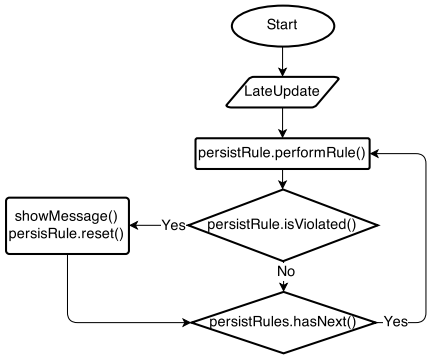
Je-li detekován nějaký přestupek je okamžitě zobrazen pomocí vyskakovacího panelu. Tento panel je obohacen o animaci postupného zobrazení a zmizení. Při detekci začátku pravidla se z dané značky, pokud existuje, vytáhne textura, která se uloží do objektu zpracovávající dané pravidlo. Pro zobrazení se použije šablona zprávy, ve které se nastavý příslušný obrázek a popisek z objektu zpracovávající pravidlo. Takto připravená šablona se vloží do zobrazovacího okna, což způsobí aktivaci animátoru. Po ukončení animace se tento panel odstraní. Pokud okno již obsahuje nějakou zprávu, pak další panel se vloží pod právě zobrazenou zprávu. Aby nedošlo k zahlcení monitoru zprávami je jejich maximální počet omezen na 4 zprávy.



Obrázek 5.6: Výpis přestupků

5.4.2 Způsob kontroly stále platných pravidel

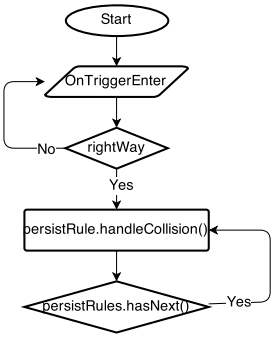
Jedná se o pravidla s nekonečnou platností např. rychlost vozidla. Kontrola těchto pravidel se provádí neustále, mění se pouze vnitřní stav implementovaného stavového automatu. Na obrázku 5.7 je znázorněn vývojový diagram neustálé kontroly. V metodě `LateUpdate` se



Obrázek 5.7: Vývojový diagram kontroly stálých pravidel

prochází pole těchto pravidel a volá se metoda `performRule` pro kontrolu pravidla. Pokud je detekován přestupek, uživateli se okamžitě zobrazí zpráva. Po-té je automat oblushující dané pravidlo nastaven do původního stavu a probíhá kontrola dalšího pravidla.

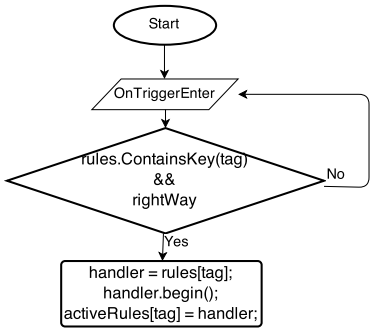
Změny stavu automatu je docíleno dalším procesem, ve kterém se kontrolují detekované značky. Rozhodnutí zda se provede změna stavu je plně ponecháno na implementaci konkrétního stavového automatu. Detekce se provádí v metodě `OnTriggerEnter`, kde se pro všechny pravidla tohoto typu volá metoda pro změnu stavu.



Obrázek 5.8: Vývojový diagram změny stavu stálých pravidel

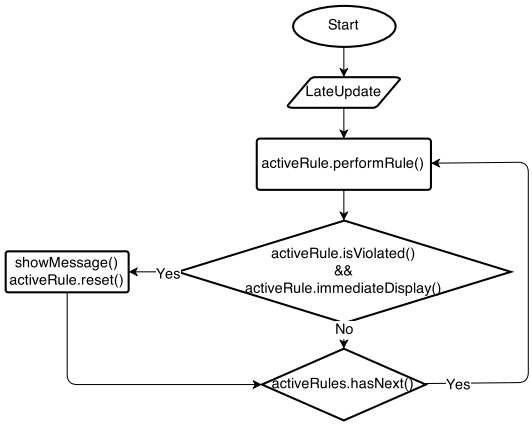
5.4.3 Kontrola pravidel s definovanou platností

Tyto pravidla mají jasně definovanou oblast platnosti např. značka „*Zákaz zastavení*“. Proto zpracování kontroly je rozděleno na tři procesy - začátek, průjezd a konec. Začátek pravidla je implementován v metodě `OnTriggerEnter`. Při detekci kolize se následně zjistí směr průjezdu (viz. 5.4.8) a pokud řidič jede ve správném směru, vybere se implementace automatu zpracovávajícího detekované pravidlo. Pokud takový automat existuje provede se jeho inicializace metodou `begin` a uloží se do aktivních pravidel pro další proces zpracování.



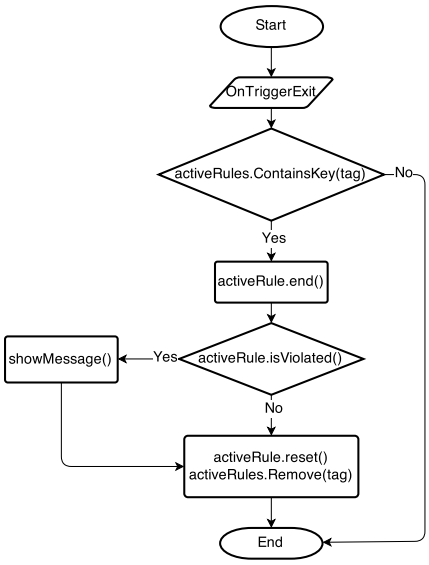
Obrázek 5.9: Vývojový diagram začátku platnosti pravidla

Druhý proces probíhá v metodě `LateUpdate` a provádí se zde kontrola pro všechna nalezená aktivní pravidla. Pokud dojde k porušení pravidla a aktivní automat požaduje okamžité informování, vykreslí se panel s danou hláškou. Takto je např. rozlišena detekce zákazu zastavení a kontroly zastavení na značce „*Stůj, dej přednost v jízdě* “. Zákaz zastavení požaduje okamžité zobrazení, ale povinnost zastavit vozidlo je stále možné dokud řidič neopustí definovanou zónu.



Obrázek 5.10: Vývojový diagram kontroly pravidla

Posledním procesem tohoto typu kontroly pravidel je ukončení aktivního automatu, které začíná v metodě `OnTriggerExit`. Nejprve se zjistí jestli pro dané pravidlo je aktivní automat. Potom se aktivní automat informuje o konci pravidla metodou `end`. Zde už není možné zrušit výpis chyby a pokud k ní došlo je zobrazen panel s hláškou. Následně je aktivní pravidlo nastaveno do původního stavu a odstraněno z aktivních pravidel.



Obrázek 5.11: Vývojový diagram ukončení pravidla

5.4.4 Okamžitá kontrola pravidel při detekci

Pravidla reagující pouze na začátek kolizí. Příkladem je značka „*Zákaz vjezdu*“. K jejich detekci postačí metoda `OnTriggerEnter`. Zde dochází k dvojímu detekování ve správném a v opačném směru. Tímto dvojím detekováním je docíleno kontroly změny směru jízdy. Pouze při detekování kolize ve správném směru dochází k vykreslení zprávy.

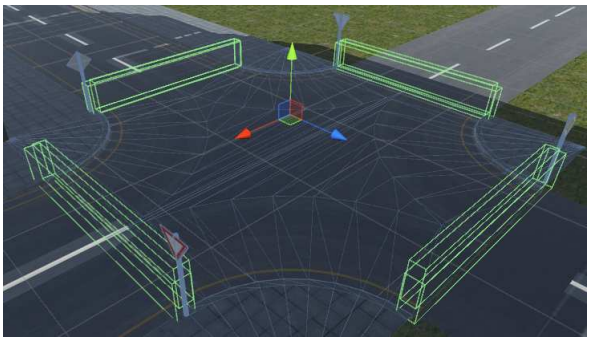
5.4.5 Rychlost jízdy

Kontrola rychlosti jízdy je persistentní pravidlo, které mění svoje nastavení v závislosti na detekovaných značkách. Pravidlo je implementováno ve třídě `SpeedRule`. Klíčovými značkami jsou „*Nejvyšší dovolená rychlost*“, „*Nejnižší dovolená rychlost*“, „*Začátek obce*“ a „*Konec obce*“. Každé persistentní pravidlo se na začátku nastaví podle počáteční konfigurace. V případě kontroly rychlosti se jedná o počáteční stav v obci, nebo mimo obec. Zpracování detekování dopravních značek je ponecháno na tomto pravidlu. Persistentní pravidla provádějí své kontroly v metodě `Update`, tedy v každém vykresleném rámci. Je-li pravidlo porušeno metoda `isViolated` vrací hodnotu `true` a TrafficRuleController okamžitě zobrazí informační panel. Rychlost vozidla se získá pomocí skriptu pro jeho ovládání, který implementuje metodu `getCarSpeedKmh`.

Pokud pravidlo detekuje jednu z uvedených značek změní svůj stav. Značka omezení maximální rychlosti je označena tagem *maxSpeed*. V názvu této značku musí být uvedena maximální rychlost ve formátu *⟨název⟩_⟨rychlost⟩_⟨pořadí⟩*. Z takto pojmenovaného objektu značky se získá maximální rychlost. Ze značky se také získá textura pro zobrazení zprávy o přestupku. Z detekované značky se nalezne potomek s názvem *sign*. Tento potomek obsahuje komponentu *MeshRenderer*, který nese texturu dané značky. Tato textura je použita jako obrázek do zprávy.

5.4.6 Jízda křižovatkou a odbočování

Křižovatka je místo střetu dvou a více cest. Hranice křižovatky jsou označené koliznímy tělesy. Jednotlivé hraniční tělesa jsou označena tagem *crossRoadEnd⟨index⟩*, kde index značí číslo vjezdu do křižovatky. Tento index začíná nulou a po směru hodinových ručiček roste. Za pomoci indexu jsme schopni detekovat jakým směrem vozidlo projíždí křižovatkou. Křižovatka může být dále označena dopravnímy značkami „*Hlavní pozemní komunikace*“ a „*Dej přednost v jízdě!*“, resp. „*Stůj, dej přednost v jízdě!*“. Detekce těchto značek slouží především pro ostatní účastníky provozu (podkapitola 5.5).



Obrázek 5.12: Kolizní tělesa pro určení směru průjezdu křižovatkou

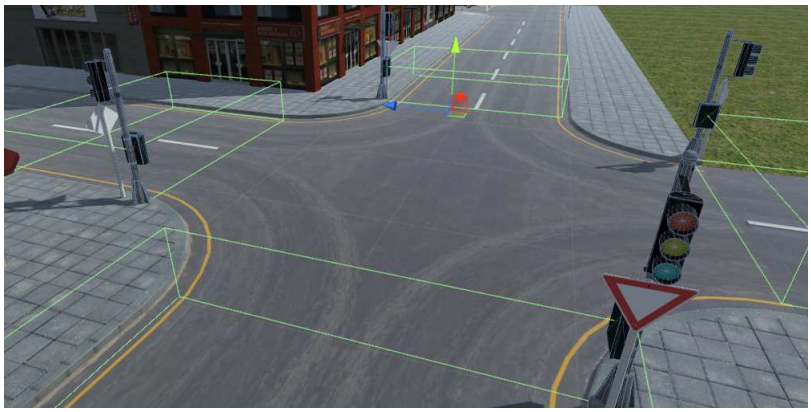
Kontrola znamení o změně směru jízdy je implementována třídou `ChangeDirectionHandler`. Jedná se o pravidlo, které reaguje při vstupu do kolizního tělesa. Nejprve se detekuje vstup do kolizního tělesa značící konec křižovatky v protisměru, tedy jde o začátek křižovatky. Z tagu detekovaného tělesa se vyčte index a uloží do proměnné `from`. Potom se detekuje konec křižovatky ve správném směru. Znovu se provede vyčtení indexu a porovnání s předchozím indexem `from`. Z tohoto porovnání zjistíme směr průjezdu

křižovatkou. Nyní se provede dotaz na vozidlo jestli má zapnutý blinkr daného směru pomocí metod `isLeftTurningOn` a `isRightTurningOn` z komponenty pro ovládání.

5.4.7 Světelná křižovatka

Světelná křižovatka a její řízení je vytvořena jako samostatný prefab objekt. To umožňuje jakoukoli křižovatku opatřit o řízení světly. Prefab je sestaven z hlavního objektu, kterému je přiřazen skript pro řízení světelných signálů *TraficLight.cs*. Potomky tohoto řídícího objektu jsou jednotlivá světelná zařízení křižovatky, která jsou opatřená o kolizní tělesa s tagem *trafficlight(index)*. Zde index je nazávislý na indexu konce křižovatky a slouží pouze pro označení světelných zařízení.

Řízení křižovatky je pro obecný počet směrů, které se postupně otevírají. Každý směr musí být opatřen o světelné zařízení. Ve skriptu existuje pole definující jednotlivé směry. Toto pole je naplněné referencemi na `MeshRenderer` světelných zažízení, proto musí být sudé délky, protože jeden směr je reprezentován dvěmi světly. Po startu aplikace se první takto zadaný směr nastavý jako otevřený. Ve skriptu je každý směr reprezentován vnitřní třídou `LightWay` a pořadí směrů je pole těchto objektů. LightWay objekt nese referenci na MeshRenderer obou světelných zařízení a stav daného směru. Rozvícení světel je vytvořeno pomocí dvou materiálů. První materál nese texturu se zhaslými světly, druhý naopak se světly rozsvícenými. Přepínáním materálů je hotova světelná signalizace.

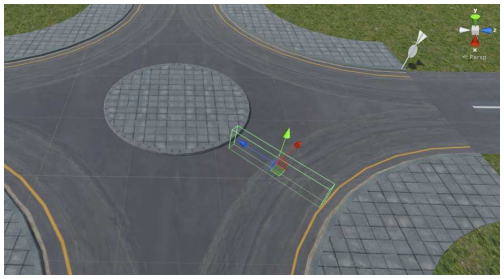


Obrázek 5.13: Kolizní tělesa světelné křižovatky

Vozidlo projíždějící světelnou křižovatkou detekuje světelné zažízení s daným tagem. Komponenta pro kontrolu pravidel *TrafficRuleController* ze své kolekce získá objekt zpracovávající dané pravidlo třídu `RedLightHandler`. Zde se při vstupu do kolizního tělesa získá objekt řízení křižovatky `TraficLight`, z jehož kolekce a daného tagu zjistíme stav právě jedoucího směru. Tento stav si *RedLightHandler* zapamatuje. Při opuštení kolizního tělesa se stejných způsobem zjistí stav směru. Svítí-li zelené světlo vše je v pořádku. Svítí-li ale pouze žluté světlo, provede se porovnání s předchozím stavem. Pokud předchozí stav byl zelený signál, pak řidič nestihl zabrzdit a vše je v pořádku. Ve všech ostatních stavech jde o porušení pravidla a systém zobrazí příslušnou zprávu.

5.4.8 Kontrola směru jízdy na kruhovém objezdu

Kontrola směru jízdy je řešena obecně pomocí kolizního tělesa. Tyto kolizní tělesa jsou vloženy do kruhového objezdu na možná místa vjezdu do protisměru. Detekce směru je řešena na principu přepočítávání souřadnic mezi souřadnicovými systémy. Detekuje-li vozidlo kolizní těleso označené tagem *direction*, provede se přepočet souřadnic vozidla ze souřadnicového systému scény do souřadnicového systému detekovaného kolizního tělesa. Toho je docíleno pomocí metody `InverseTransformPoint()` v objektu `Transform` kolidovaného tělesa. Nyní stačí porovnat výslednou souřadnici osy x s nulou. Jede-li automobil ve směru definovaném směrovým tělesem pak platí $x < 0$. Na obrázku 5.14 je znázorněn souřadnicový systém kolizního tělesa. Červená osa x určujě správný směr jízdy. Jede-li vozidlo v opačném směru k detekci dochází před tímto tělesem po směru osy x.



Obrázek 5.14: Směrové kolizní těleso

5.4.9 Zákaz zastavení

Jedná se o pravidlo s definovanou zónou platnosti. Kontrola zákazu zastavení je založena na aktuální rychlosti automobilu. Pohybuje-li se vozidlo v zóně se zákazem zastavení, provádí se neustálá kontrola rychlosti. Zastavení je považováno za pokles rychlosti pod velmi nízkou hodnotu 0.1 km/h. Zóna pro kontrolu zákazu zastavení je definována kolizním tělesem v celém jejím prostoru. Toto kolizní těleso je označeno tagem *noStopping*.

Obrázek 5.15: Kolizní těleso zákazu zastavení

Umístění kolizního tělesa zákazu zastavení je v blízkosti křižovatek, přechodů a značek „*Zákaz zastavení*“. Pro odlišení zastavení na značce „*Stůj, dej přednost v jízdě*“ a zastavení na zákazu zastavení, musí být tato zóna umístěna kři kraji vozovky tak, aby nezasahovala do zóny, kde řidič naopak musí zastavit.

Kontrola je následující. Komponenta *TrafficRuleController* pomocí detekovaného tagu kolizního tělesa vybere příslušný obslužný objekt. V metodě `Update` nad nalezeným objektem volá metodu `performRule`, ve které se kontroluje rychlost vozidla pomocí ovládací komponenty *CarController*. Pokud vozidlo zastaví, nastavý se proměnná `violated` na hodnotu `true`. Jakmile vozidlo vyjede ze zóny zákazu zastavení detekuje se opuštění kolizního tělesa a pravidlo se zneaktivní.

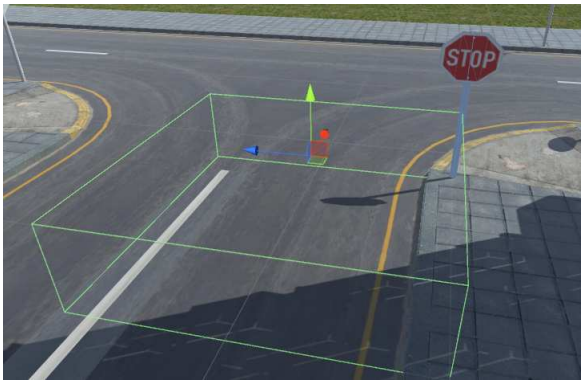
5.4.10 Zákaz vjezdu

Zákaz vjezdu je pravidlo, jehož kontrola je narušena jakmile je detekována kolize s tělesem označené tagem *noEntry*. Kolize je detekována pouze ve správném směru. Kontrola pravidla je zpracována ve třídě `NoEntryHandler`. Po detekování kolize *TrafficRuleController* vybere objekt této třídy a spustí jeho kontrolu. Protože se jedná o typ zpracování pravidla, které reaguje při kolizi, provede se okamžité zobrazení hlášky.

5.4.11 Značka „Stůj, dej přednost v jízdě!“

Na značce „*Stůj, dej přednost v jízdě!*“ se kontroluje zda-li řidič zastavil před vjezdem do křižovatky. Jde o kontrolu pravidla s danou zónou platnosti. Tato kontrola je téměř shodná s kontrolou zákazu zastavení. Provádí se v rámci zóny definované kolizním tělesem označeným tagem *stopSign*. Při vjezdu vozidla do kolizního tělesa se začne pravidelně testovat zastavení vozidla. Tím je považován pokles rychlosti pod velmi nízkou hodnotu 0.1 km/h. Kolizní těleso je umístěno u značky tak, aby zasahovalo do prostoru vozovky, kde je požadováno zastavení vozidla.

Kontrola začíná vstupem do kolizního tělesa. Komponenta *TrafficRuleController* nalezne příslušný objekt pro zpracování pravidla za použití tagu tělesa. Zpracování se provádí v metodě `Update`, kde se kontroluje aktuální rychlost automobilu. Pokud vozidlo zastaví, nastaví se proměnná `stopped` na hodnotu `true`. Při opuštění kolize se kontroluje tato proměnná. Pokud má hodnotu `false` řidič nezastavil a jde o porušení pravidla.



Obrázek 5.16: Kolizní těleso značky „Stůj, dej přednost v jízdě!“

5.4.12 Kolize s ostatnímy účastníky provozu

V simulátoru je důležité také detekovat nehody. Za nehodu je považována srážka vozidla s ostatnímy účastníky. Za nedání přednosti v jízdě je považována nehoda, která byla detekována v rámci křižovatky. Tento druh kolize je odlišný od předchozích typů. Pro kontrolu

pravidel se používají kolizní tělesa nastavené jako spouštěč události. Kolize s ostatnímy účastníky vyžadují zpracování fyziky a proto nemůžou být pouze spouštěče události. Tyto kolize se detekují v metodě `OnCollisionEnter`.

5.5 Účastníci provozu

Součástí zadání práce je také reprezentace pohybu účastníků provozu a jejich vzájemná interakce. Těmito účastníky jsou řidiči automobilů. Pro tyto účastníky je použit model popsán v podkapitole 4.2. V následujících podkapitolách je popsána implementace chování těchto účastníků.

5.5.1 Implementace pohybu

Pohyb automobilů je implementován ve skriptu *AICarController.cs*, který je následně přiřazen k danému modelu. Na model ostatní účastníků se vztahuje fyzika stejně jako na jiné vozidlo, proto musí obsahovat také komponentu *RigidBody*. U této komponty nastavíme parametr mass na hodnotu 1200 kg a parametr drag na hodnotu 0,01. Další potřebnou komponentou je *BoxCollider*, která způsobí kolize vozidel. Vlastní pohyb je implementovám pomocí komponenty *WheelCollider* obdobně jako u předchozího modelu z podkapitoly 5.2.1.

Aby bylo vozidlo schopné pohybu musí mít zadanou dráhu jízdy. Tato dráha je definovaná pomocí pole kontrolních bodů, které vozidlo projíždí jeden po druhém. Komponentě pro řízení vozidla se předá kontejnér těchto kontrolních bodů. Ze získaného kontejneru se vytáhnou všechny komponenty typu *Transform* ve stejném pořadí v jakém byli vloženy a sestaví se pole kontrolních bodů.

Obrázek 5.17: Trať účastníka provozu

Nyní je vozidlo informováno o své trase, ale ještě není schopné se rozjet. Ovládáme-li vozidlo pomocí klávesnice vstupem je precentuální hodnota stisknutí tlačítka v daném směru (čtení hodnot pomocí `Input.getAxis("vertical")`). Touto hodnotou je potom vynásoben výkon automobilu a použit jako vstup pro parametr *motorTorque*. Na stéjném principu funguje pohyb tohoto vozidla. V metodě `LateUpdate` se provádí navigace vozidla. Ta spočívá v přepočtení polohy aktuálního kontrolního bodu do souřadnicového systému vozidla. Při přepočtu nesmíme brát v potaz výšku bodu nad terénem (souřadnice y). Vstupní hodnoty pomyslné „klávesnice“ spočteme následovně (vektor v je přepočtená souřadnice kontrolního bodu):

$$inputSteer = \frac{v_x}{|\vec{v}|} \tag{5.1}$$

$$inputTorque = \frac{v_z}{|\overrightarrow{v}|} \tag{5.2}$$

Pokud vozidlo musí prudce zatáčet hodnota *inputTorque* je podělena dvěmi, aby nedošlo k převrácení vlivem vysoké rychlosti v zatáčce. Až se vozidlo přiblíží ke kontrolnímu bodu na dostatečnou vzdálenost 5m začne směřovat k dalšímu bodu. Implementace jízdy po trase je v metodě `navigateTowardsWaypoint`, která se volá v metodě `FixedUpdate` v řídící komponentě *AICarController*.

Nyní je automobil schopen projíždět svoji definovanou trasu a je zapotřebí ho naučit bezpečné jízdě v provozu. Zastavíme-li s naším automobilem do dráhy automatického vozidla, pak toto vozidlo do nás vrazí. Nebo pokud před ním pojedeme rychlostí menší, pak do nás také narazí. Proto vozidlo sleduje stav provozu před sebou a patřičně podle situace reaguje. Není ale tak chytré aby zvládlo objet překážku. Sledování stavu před vozidlem je pouze do vzdálenosti efektivní brzdné dráhy v aktuální čase plus minimální vzdálenost mezi vozidly. Délku brzdné dráhy vypočteme podle vzorce:

$$s = \frac{r \cdot m}{M} \cdot v^2, \tag{5.3}$$

kde v je aktuální rychlost, m je hmotnost vozidla, M je brzdná kroutící síla a r je poloměr kol. Prohledávání prostoru v Unity je pomocí tzv. paprsků. Tyto paprsky mohou být třech základních tvarů:

1. Ray - obyčejný paprsek představující přímku vyslanou prostorem,
2. Sphere - koule, která se vyšle daným směrem v rámci něhož detekuje kolize,
3. Capsule - spojení dvou koulí vyslané jistým směrem, ve kterém detekuje kolize.

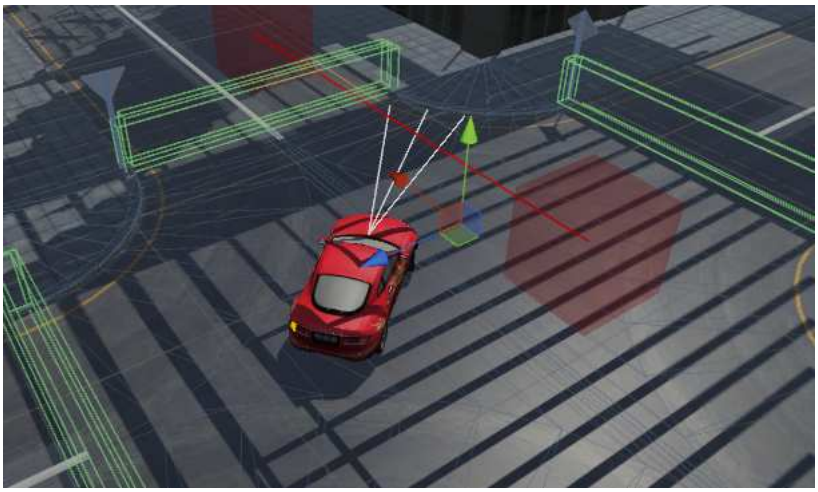
Pro tento účel postačí *Ray* paprsek. Vozidlo tedy sleduje prostor před sebou pomocí tří paprsků otočených po 10° s definovanou délkou. Výsledkem vyslání parpsku je buď pouze první nalezený objekt, nebo pole všech kolidovaných těles, jejichž pořadí není předem určené. Pro detekování vozidla vpředu by postačilo nalézt první kolizi, ale jak je popsáno v následujících kapitolách pořebujeme detekovat veškeré objekty před vozidlem. Pokud jedním z nalezených objektů je automobil, dané vozidlo začne brzdit. Automobil je označen tagem *car*. Implementace kontroly trasy je v metodě `controlWay` a je volána po metodě pro navigování po trase. Proto brždění vozidla je provedeno nastavením proměnné *inputTorque* na hodnotu -1 a parameteru *brakeTorque* předních WheelColliderů na brzdnou kroutící sílu.

5.5.2 Znamení o změně směru jízdy

Pozice kontrolních bodů trasy nutí vkládat dva body pro hladké projetí zatáčky. Toto je podmínkou pro detekci změny směru jízdy při průjezdu křižovatkou nacházející se na trase vozidla. První bod zatáčky musí ležet v křižovatce. Potom spojovací úsečka mezi tímto bodem a následujícím bodem trasy musí protínat kolizní těleso ohraničující křižovatku. Vozidlo přijíždějící ke křižovatce detekuje hranici křižovatky pomocí sledování prostoru před sebou. Pokud ještě nemá jasno jakým směrem se vydá, provede vyslání paprsku od aktuálního kontrolního bodu své trasy směrem k následujícímu bodu. Předchozí podmínka říká, že na této přímce leží kolizní těleso výjezdu z křižovatky. Z tagů detekovaných hraničních těles křižovatky určíme směr průjezdu. Jedním nedostatkem tohoto řešení je křižovatka nacházející se směrem k aktuálnímu bodu, kterou vozidlo projíždí přímo. V tomto případě není aktuální bod uvnitř této křižovatky a vozidlo směřuje rovně. Tento problém je vyřešen tak,

že vozidlo detekující hranici křižovatky testuje vzdálenost mezi aktuálním bodem trasy a hranicí křižovatky. Pokud je vzdálenost pod definovanou úrovní, pak provede detekci směru.

Při jakékoli detekci změny směru se provede aktualizace grafické reprezentace znamení, která je provedena změnou materiálu ploch modelu určených pro směrová světla. Na obrázku 5.18 je znázorněna detekce směru jízdy. Vozidlo již detekovalo hraniční těleso křižovatky *crossroadEnd1* a směr jízdy. Červené kostky představují body tratě a jejich červená spojnice je vyslaný paprsek, který protíná hraniční těleso *crossroadEnd2*. Z rostoucího indexu vozidlo poznalo změnu směru vlevo a dalo znamení o změně směru.



Obrázek 5.18: Detekce směru jízdy křižovatkou

5.5.3 Průjezd křižovatkou

Hlavní úkol tohoto vozidla je průjezd křižovatkou. Jde o průjezd křižovatky označené značkami upravujícími přednost a o křižovatky bez dopravního značení, kde platí přednost přijíždějících vozidel zprava.

Řešení průjezdu křižovatkou označenou dopravními značkami je následující. Vozidlo detekuje jednu ze značek upravujících přednost na křižovatce. Pokud detekuje značku „*Hlavní pozemní komunikace*“, pak má přednost a není třeba dávat přednost. Pokud ale při detekování směru jízdy zjistí, že odbočuje vlevo dává přednost protijedoucím. Proto provede detekci přijíždějících vozidel z tohoto směru (popsáno níže). Pokud nalezne přijíždějící vozidlo začne naplno brzdit, aby včas zastavilo. Brždění a detekce přijíždějícího vozidla probíhá dokud je takové vozidlo nalezeno. Pokud již žádný automobil nepřijíždí vozidlo pokračuje v jízdě. Pokud vozidlo detekuje značku „*Stůj, dej přednost v jízdě*“, nejprve čeká, dokud se nepřiblíží ke značce na vzdálenost odpovídající aktuální brzdné vzdálenosti a potom začne brzdit tak, že zastaví před touto značkou. Opoždění brždění je z důvodu kolizního tělesa značky, které označuje zónu, kde vozidlo musí zastavit a zasahuje tak před danou značku. Až vozidlo zastaví provede detekci přijíždějících vozidel ze všech směrů křižovatky. Detekce probíhá dokud nějaké vozidlo bylo nalezeno a vozidlo tak stále brzdí. Zde může dojít k zablokování (deadlocku), pokud se na křižovatce střetnou dvě tyto vozidla přijíždějící po různých vedlejších komunikacích. Řešení deadlocku ponecháme na dalším vývoji simulátoru. Pokud vozidlo detekuje značku „*Dej přednost v jízdě*“, provede okamžitou detekci přijíždějících

automobilů. V případě, že nebyl detekován přijíždějící automobil, další detekce není nutná a vozidlo pokračuje dále. Je-li nějaký nalezen začne brzdit za stálé kontroly křižovatky.

Detekce přijíždějících vozidel

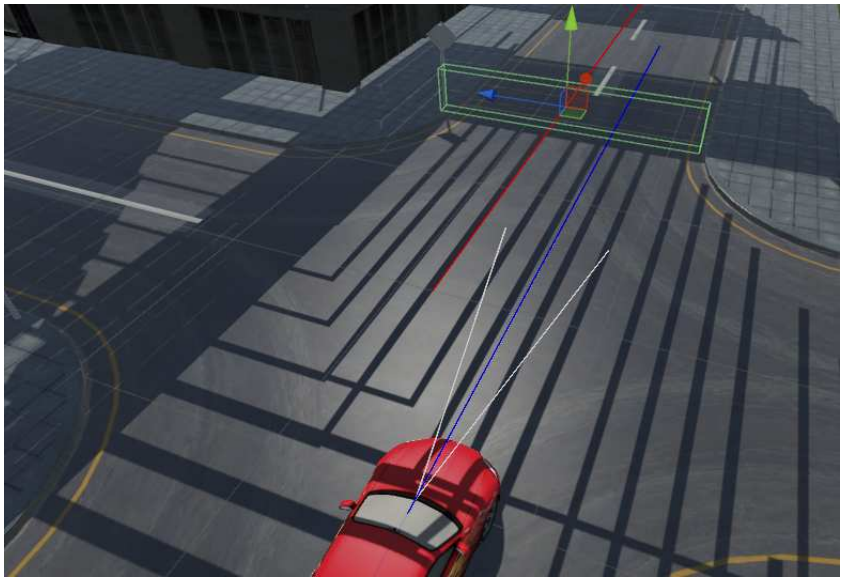
Detekce vozidel je založena na vysílání paprsků pouze specifickou vrstvou. Každé vyslání paprsku může být omezeno maskou vrstev, kterými se má vyslat a ve kterých bude detekována kolize. Všechny vozidla simulátoru mají nastavenou vrstvu *car*, jejíž maska je $2^9 = 512$. Další definovanou vrstvou je vrstva *crossroad*, ve které se nacházejí hraniční tělesa křižovatek. Jak je popsáno dále, tato vrstva slouží pro detekci směrů křižovatky a hodnota masky je $2^8 = 256$.

Potřebuje-li vozidlo nalézt přijíždějící automobily ke křižovatce v daném směru, provede nejprve nalezení příslušného hraničního tělesa křižovatky. Toho docílí pomocí vyslání paprsku tvaru koule v daném směru pouze vrstvou *crossroad*. Tyto směry jsou definované úhly vlevo 30°, vpravo 20° a vpřed 0°. Jakmile jsou získány potřebné hraniční tělesa křižovatky provede se získání směrů z těchto těles. Každé těleso svou souřadnicovou osou x směřuje podél příjezdové silnice. Proto stačí pouze tento vektor převést z místního souřadnicového systému do souřadnicového systému scény metodou `Transform.TransformDirection`. Nyní už víme, jakým směrem vyslat paprsek pro detekci vozidel a odkud je zhřejmé, od pozice daného hraničního tělesa. Takto se tedy vyšle další paprsek tentokrát typu capsule, abychom byli schopni detekovat v célé šířce komunikace. Tento paprsek je poslán pouze vrstvou *car*, proto detekovanými tělesy mohou být pouze vozidla jedoucí po komunikaci. Takto také můžeme detekovat vozidlo vzdalující se od křižovatky. Proto dalším krokem je test směru tohoto vozidla. Způsob tohoto testu byl už nekolikrát v této práci zmiňován. Jde o převod pozice mezi souřadnicovými systémy a porovnání hodnoty jedné z os. Převedeme tedy pozici hraničního tělesa do souřadnic detekovaného vozidla a je-li hodnota na ose x kladná, toto vozidlo směřuje ke křižovatce. Tímto způsobem nalezneme vozidla, která přijíždějí ke křižovatce, ale také vozidla, která parkují při kraji vozovky a jsou pouze natočená směrem ke křižovatce. Což není velký problém pokud uživatel simulátoru neplánuje parkovat delší dobu a automatická vozidla nejsou schopná parkovat při kraji vozovky.

Na obrázku 5.19 je znázorněna detekce přijíždějících vozidel v protisměru. Vozidlo vyslalo modrý paprsek pro nalezení příslušného hraničního tělesa křižovatky, ze kterého potom nalezlo směr vyslání červeného parprsku detekujícího přijíždějící vozidla.

5.5.4 Průjezd světelnou křižovatkou

Průjezd světelnou křižovatkou je implementován pomocí návrhového vzoru *Observer*. Princip je takový, že vozidlo přijíždějící ke světelnému zařízené zjistí stav daného směru pomocí řídící komponenty viz. podkapitola 5.4.7. Je-li tento směr ve stavu zavřeno (svítí červené světlo), vozidlo se přiblíží k zařízení na vzdálenost efektivní brzdné dráhy a začne brzdit. V tomto okamžiku se zároveň registruje jako posluchač změny stavu observeru resp. směru. Každý směr křižovatky je reprezentován objektem třídy `LightWay`, který zároveň nese pole všech posluchačů rozhraní `LightListener`. O každé změně stavu jsou všichni posluchači informováni pomocí metody tohoto rozhraní. Pokud tedy vozidlo čeká na změnu stavu stále brzdí až do doby, kdy obdrží patřičný signál o změně stavu na zelenou. V tom případě odbrzdí a pokračuje dále podle pravidla o odbočování, tzn. při odbočování vlevo mají protijedoucí řidiči přednost v jízdě.



Obrázek 5.19: Detekce přijíždějících vozidel

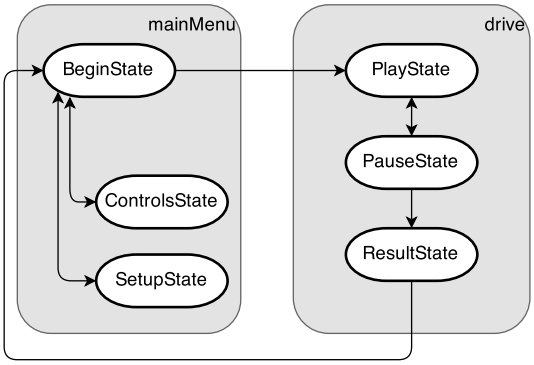
5.6 Řízení aplikace a uživatelské rozhraní

Aplikace je řízena stavovým automatem. Tento stavový automat je implementován skriptem *StateManager.cs*. Simulátor autoškoly se skládá ze dvou scén. Úvodní scénou je hlavní nabídka a druhou scénou je samotná simulace. Řízení spočívá v přepínání těchto scén a v závisloti na vstupu uživatele zobrazovat požadovaná data. Důležitým požadavkem je existence pouze jednoho řízení, které je neustále dostupné. Proto v úvodní scéně s hlavní nabídkou existuje prázdný objekt nesoucí komponentu *StateManager*. Při přepínání scén v Unity se provede odstranění všech objektů aktivní scény a vytvoření nových objektů z druhé scény. Aby nedošlo k odstranění řízení je v metodě `Awake` zavolána metoda `DontDestroyOnLoad`, která způsobý že daný objekt se neodstraní při načítání scén. Bohužel při načtení úvodní scény se vytvoří řízení nové, což způsobí existenci dvou najednou. Toto je vyřešeno pomocí statické proměnné nesoucí referenci na prvně vytvořený objekt řízení. Potom při vytvoření nového se tato proměnná kontroluje. Pokud není `null` tak se nově vytvořený objekt okamžitě odstraní. Tímto je zajištěna existence pouze jednoho objektu řízení. Protože tento objekt existuje stále i během přepínání scén je dobré k němu také navěsit komponentu nesoucí veškerá data. Touto komponentou je skript *GameData.cs* a reference na ni je dostupná z komponenty řízení.

Implementace stavů řízení aplikace je podle návrhového vzoru *Strategy*. Každý stav splňuje kontrakt definovaný rozhraním `IStateBase`. Tento kontrakt říká, že každý stav definuje vykreslení GUI na obrazovku (metoda `showIt`) a zpracování metody `Update`. Při inicializaci řízení aplikace se správce stavů nastaví na první stav *BeginState*.

Stav *BeginState* je prvotním stavem. Zde je implementována obsluha tlačítek dostupných z hlavní nabídky. Každé tlačítko reprezentuje nový stav aplikace. Tudíž v těchto obsluhách dochází k přepnutí stavu a skrytí všech tlačítek hlavní nabídky. V metodě *Update* se kontroluje stisknutí klávesy *escape* pro ukončení aplikace a v metodě pro vykreslení rozhraní se zobrazují tlačíka, aby se při přepnutí do tohoto stavu znovu zobrazily.

Návrh grafického rozhraní obsahuje obrazovku pro kontrolu ovládání aplikace. Tato ob-



Obrázek 5.20: Stavový automat řízení

razovka je řízena stavem *ControlsState* a je vytvořena grafickým panelem, který je umístěn mimo viditelnou část obrazovky. Při vykreslení GUI se tento panel posune zpět na viditelnou plochu. Zároveň se zde generuje obsah tohoto panelu z konfigurace ovládání simulace. Vykreslení obsahu je pomocí statické metody `GUI.Label`, která vykreslí text na definované pozici. Dále je tu tlačítko „zpět“, které vrátí stav *BeginState*.

Dalším stavem je *SetupState*. Tento stav obsluhuje obrazovku pro nastavení aplikace, která je také implementována jako panel schovaný mimo plátno obrazovky. Tento stav při vykreslení panel s nastavením zobrazí a naplní jej políčky formuláře. Jakákoli změna nastavení je okamžitě uložena bez dodatečného potvrzení. Aktuálně je zde nastavení jazyka aplikace a typ převodovky vozidla.

Stavem, který obsluhuje vlastní simulaci, je stav *PlayState*. Zde je důležité zmínit jednu vlastnost při přepínání stavů řízení. Při přepnutí stavu lze definovat, zda jiné komponenty mohou kontrolovat vstup uživatele. Standardně pouze jednotlivé stavy mohou číst vstup. Hlavním důvodem tohoto řešení je pozastavení simulace, protože Unity jej standardně nepodporuje. Jediné jak řešit pozastavení hry je zastavením pohybu nastavením změny času na nulu. V tomto případě je stále možné číst vstup a nastavovat tak interní stav aktuálně zastavených objektů. Proto každému čtení vstupu předchází kontrola dostupnosti užitím komponenty *StateManager*. Dále tento stav přepíná pohledy simulace aktivováním a deaktivováním jednolivých kamer. Při stisku tlačítka *escape* pozastaví simulaci.

Pokud je simulace pozastavená, řízení se nachází ve stavu *PauseState*. Pozastavení simulace docílíme nastavením proměnné `Time.timeScale` na hodnotu 0, která představuje procentuální rychlost pohybů. Pomocí proměnné `AudioListener.volume` jsou vypnuty zvuky. Opětovného zapnutí docílíme nastavení těchto proměnných zpět na hodnotu 1. V tomto stavu se vykreslují tlačítka pro pokračování v simulaci a zobrazení výsledku jízdy.

Pro zobrazení výsledků slouží stav *ResultState*. Všechny přestupky jsou zaznamenány v komponentě *GameData.cs*. V tomto stavu je simulace stále pozastavena a na obrazovku se vypíší veškeré přestupky, kterých se uživatel dopustil. Každý přestupek se skládá z obrázku, názvu přestupku a popisu pravidla. Před každou započatou simulací se promaže kolekce přestupků z předchozí jízdy. Ukončení simulace a přepnutí zpět do hlavní nabídky provádí obsluha tlačítka *Zpět* ve spodní části obrazovky.

5.7 Internacionalizace

V aplikaci jsou veškeré hlášky internacializovány, pro možné rozšíření dalších jazyků a snadnou úpravu textů. Jazyk lze změnit v nastavení aplikace. Při každém spustění aplikace se provede inicializace všech dostupných jazyků. Implementace překladu je užitím kolekce `Dictionary`, kde klíčem je zkratka textu použitá k vyhledání. Implementace čtení hlášek je pomocí návrhového vzoru *Singleton*. Úvodní stav aplikace provede načtení souboru a vytvoření objektu třídy `Lang`, který zpřístupňuje texty. Statická proměnná nese referenci na Singleton objekt a zpřístupňuje tak hlášky zbytku aplikace.

```
<?xml version="1.0" encoding="utf-8"?>
<languages>
  <en>
    <!-- Generic -->
    <string name="menuTitle">Driving School</string>
    <string name="menuStart">Start</string>
  <\en>

  <cz>
    <!-- Generic -->
    <string name="menuTitle">Autoskola</string>
    <string name="menuStart">Start</string>
  <\cz>
<\languages>
```

Obrázek 5.21: Vzorek hlášek aplikace

Kapitola 6

Závěr

V této práci jsme se seznámili s vývojem her a simulátorů. Probrali jsme možnosti implementace a tvorbu modelů. Dali jsme si za cíl vytvořit funkční aplikaci 3D autoškoly, která může sloužit k procvičování jízdy po pozemní komunikaci.

Cíl vytvoření simulátoru se nám podařilo splnit. Výslednou aplikace je možné přeložit na platformy Windows, OS X, GNU/Linux a další. V simulátoru uživatel projíždí městem, ve kterém se nacházejí různé dopravní značky, ale také se dostane mimo obec. Při jeho jízdě dochází ke kontrolování pravidel o zastavení, odbočování, jízdě křižovatkou, světelných signálů, některých dopravních značek a rychlosti jízdě. Na křižovatkách ve městě narazíme na simulovaný provoz pomocí třech automobilů, jedoucích po definované trase. Zároveň si můžeme všimnout dodržování pravidel těchto vozidel.

Dále byli připraveny modely bloků komunikace, které lze různě přeskládat pro tvorbu jiné mapy. Pro dopravní značení a světelné křižovatky byli vytvořeny šablony pro snadné použití. Implementace kontroly pravidel je řešena co nejobecněji, aby bylo snadné aplikaci rozšířit o další pravidla. Výsledkem bylo nalezení tří skupin společného řešení kontroly pravidel. V neposlední řadě byla také vytvořena šablona pro simulované účastníky provozu. Pro další vývoj aplikace bude nutné vyřešit problém se zablokováním těchto vozidel, které může vzniknout na křižovatce. S touto úpravou by se také mohla implementovat větší inteligence vozidel a umožnit jim tak objet překážku.

Chybějící funkčností v implementovaném řešení je možnost zvolit ovládání simulace v nastavení aplikace a v připraveném městě se nenacházejí silnice s více pruhy, které se ve městech často vyskutují.

Pro budoucí rozšíření bude vhodné připravit více modelů vozidla, jízdu s instruktorem, který říká jakým směrem se vydat, nebo pouze naznačení směru šipkami. Dále nesmí chybět možnost jízdy za snížené viditelnosti, nebo jízda po dálnici. Také provést úpravu šablon silnic, protože nyní lze sestavit pouze ortogonální síť pozemní komunikace.

Literatura

[1] Crytek: CryEngine. http://cryengine.com, 2015 [cit. 2015-05-06].

[2] Epic Games: Unreal Engine features. http://www.unrealengine.com/unreal-engine-4, 2015-03-02 [cit. 2015-05-06].

[3] Gregory, J.: *Game Engine Architecture*. A K Peters/CRC Press, 2009, ISBN 978-1-4398-6526-2.

[4] Hanzl, P.: Dopravní značky. http://www.znacenihanzl.cz/ke-stazeni.html, 2011 [cit. 2015-05-11].

[5] LLC, F. U.: Dodge Challenger. http://www.dodge.com/en/challenger/performance, 2015 [cit. 2015-05-11].

[6] Menard, M.: *Game development with Unity*. Course Technology PTR, 2012, ISBN 9781435456594.

[7] Nextwave Multimedia: Blender 3D model of DODGE Challenger. http://nextwavemultimedia.com/blog/3d-animation/free-download-blender-3d-model-of-dodge-challenger, 2010-05-05 [cit. 2015-05-11].

[8] NoneCG: NYC Block #6. https://www.assetstore.unity3d.com/en/content/16272, [cit. 2015-05-11].

[9] Norton, T.: *Learning C# by Developing Games with Unity 3D Beginner's Guide*. Packt Publishing, 2013, ISBN 978-1-84969-658-6.

[10] Pranckevicius, A.: MirrorReflection3. http://wiki.unity3d.com/index.php/MirrorReflection3, 2015-02-25 [cit. 2015-05-19].

[11] Sweeney, T.: If you love something, set it free. http://www.unrealengine.com/blog/ue4-is-free, 2015-03-02 [cit. 2015-05-06].

[12] Technologies, U.: Car Tutorial. https://www.assetstore.unity3d.com/en/content/10, 2015-02-25 [cit. 2015-05-19].

[13] TF3DM: Street System. http://tf3dm.com/3d-model/street-system-v10-48448.html, 2014 [cit. 2015-05-11].

[14] Tidwell, J.: *Designing Interfaces*. O'Reilly Media, 2010, ISBN 978-1-449-37970-4.

[15] Unity Technologies: Unity Game Engine. `http://unity3d.com`, 2015-03-02 [cit. 2015-05-06].

[16] Česká Republika: Zákon č. 361/2000 Sb., o provozu na pozemních komunikacích (o silničním provozu).
`http://www.ibesip.cz/data/web/soubory/legislativa/novela-361-2013.pdf`.

Příloha A

Obsah DVD

- project - adresář obsahující kompletní Unity projekt s modely a zdrojovými kódy
- build - obsahuje přeloženou aplikaci pro Windows x64
- thesis - adresář obsahující zdrojové kódy této práce
- thesis.pdf - text práce
- readme.txt - návod pro instalaci a spuštění aplikace