Univerzita Hradec Králové
Fakulta informatiky a managementu

DIPLOMOVÁ PRÁCE

2020 **Bc. Milan Kořínek**

Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Generování virtuální scény pro podporu výuky pravidel silničního provozu

Diplomová práce

Autor: Bc. Milan Kořínek
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Bruno Ježek, Ph.D.

Hradec Králové duben 2020

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 29. 4. 2020 Bc. Milan Kořínek

Poděkování

Tímto děkuji Ing. Brunu Ježkovi, Ph.D. vedoucímu mé diplomové práce, za vedení a zájem, který mi věnoval. Dále bych chtěl poděkovat všem, kteří mi pomáhali svými připomínkami, radami i náměty, obzvlášť pak svému bratrovi Michalu Kořínkovi, který se dokáže na problém podívat zcela odlišným způsobem.

Anotace práce

Cílem této práce je navrhnout a následně vytvořit postupy, díky kterým by bylo možné vytvořit aplikaci pro výuku pravidel silničního provozu. V první části práce je objasněno, co je to simulace, k čemu slouží a dále některé důležité informace k silničnímu provozu. Další část práce obsahuje návrh a způsob řešení jednotlivých komponent jako je například vozidlo a jeho chování v křižovatce. Nakonec je uveden způsob převedení návrhu do praxe, v čem bylo nutné udělat změny a možnosti využít vytvořených komponent pro výuku.

Annotation

Title: Generating a virtual scene to support traffic rules learning

The goal of Diploma Thesis is to design and subsequently create procedures that would make it possible to create an application for teaching traffic rules. The first part of the thesis explains what a simulation is, what it is used for and some important information about road traffic. The next part of the thesis contains a proposal and a way of solving individual components such as a vehicle and its behavior at a crossroads. Finally, it is described how the design has been put into practice, what has been changed and how to use the created components for teaching.

Obsah

1 Úvod

Každoročně se na českých silnicích stane více než sto tisíc dopravních nehod a z nich více než pět set končí smrtí. Nehodovost ovlivňuje mnoho faktorů, z nichž některé není možné zcela eliminovat. Mezi takové patří třeba špatné počasí, náhlé poruchy vozidla nebo srážka se zvěří. Jiné faktory však ovlivnit lze. K takovým patří například neohleduplnost řidičů, agresivita, jízda pod vlivem alkoholu nebo drog a také řidičská nezkušenost [11]. Nejen u nás [15], ale i v ostatních státech světa, jsou za celou čtvrtinu až třetinu dopravních nehod zodpovědní mladí řidiči [9] a nejvíce nehod je způsobeno v prvních dvou letech řízení motorového vozidla. Ke všem problémům, se kterými se potýkají ostatní řidiči se u začátečníků přidá právě nezkušenost. Ti mnohdy neznají svoje hranice, neumějí předvídat chování ostatních řidičů a mohou se tak dostat do situací, ve kterých nevědí, jak zareagovat, natož aby zareagovali ve zlomku vteřiny. Jedním z hlavních důvodů je nedostatek praxe. V současnosti je v autoškolách patrný odklon od využívání nákladných trenažerů a také z časových důvodů se co nejdříve přechází k praktické výuce na silnicích. Při té však podle osnov stačí strávit za volantem jen kolem třiceti hodin, a to je pro získání správných návyků pro řešení nenadálých situací nepochybně málo [15]. Problémem nedostatku praxe by mohl být řešen vytvořením aplikace, simulující různé reálné situace v dopravě. Tu by bylo možné nainstalovat doma na osobní počítač žáka a ten by mohl trénovat bez drahého zařízení, dostupného pouze v autoškole. Nesporná výhoda tohoto způsobu praktické výuky by spočívala v tom, že by si jednotliví uživatelé mohli sami individuálně nastavit dobu a délku tréninku, a tak by nebyli limitováni časovými normami, stanovenými v osnovách pro autoškoly. Kromě toho by tento způsob také ušetřil čas učitelům, kteří by jej pak mohli efektivněji věnovat jednotlivým žákům v rámci jejich individuálních potřeb.

Tématem této práce je navržení a vytvoření součástí výše zmíněné aplikace, které budou zajišťovat detekci pravidel silničního provozu. Práce je rozdělena do tří částí. V první části je vysvětleno, co je to simulační model. Je zde popsán způsob vytváření simulace, rozdělení do kategorií, možnosti využití, a nakonec je zde popsáno několik zástupců z řad softwaru, které umožňují vytvářet tyto simulační modely.

Druhá část se zabývá dílčími částmi návrhu a jejich řešením. Jsou zde popsány jednotlivé komponenty a části aplikace. Také jsou zde uvedeny vzniklé problémy, jejich řešení a případné odklonění od původního návrhu. Vše je rozděleno do samostatných kapitol podle logických celků.

V poslední části jsou popsány vytvořené komponenty aplikace a jejich testování. Jsou zde také popsány známé problémy a jedna kapitola je věnována směrům, kterými by bylo možné aplikaci dále rozvíjet.

2 Simulační model

Počítačová simulace je program, který se snaží simulovat, tedy napodobit jiný ať už reálný nebo teoretický systém [8]. Simulace se používá v případech, kdy je potřeba zjistit nějakou skutečnost či získat nějakou informaci, ale není možné nebo je příliš nákladné tuto informaci získat ze skutečného systému.

2.1 Vytváření simulace

Simulace sestává ze tří kroků. V každém z následujících kroků může nastat chyba, která se může a nemusí projevit, nebo se může objevit až při specifickém vstupu. Proto je nutné u simulace určit rozmezí v jakém je model platný [1].

Návrh simulace

Při návrhu simulace je nutné začít tím, co je o modelovaném systému známé. Čím více informací o systému je k dispozici, tím bude simulační model přesnější. Na jednu stranu více informací zvyšuje přesnost simulace, ale na druhou stranu je nutné ponechat pouze informace, které jsou důležité a které mohou ovlivnit požadovaný výsledek. Kdyby byly do simulace zaneseny veškeré známé informace, výpočty by byly příliš náročné a mohlo by se stát, že by na sebelepším stroji simulace běžela pomalu nebo by dokonce nebylo možné takovou simulaci spustit. Informace zanesené do simulace mohou být různé. Může to být jeden fyzikální vzorec nebo to může být mnoho terabytů dat. Při navrhování simulace tedy neznamená, že čím víc informací, tím lépe, naopak je nutné pečlivě zvážit jaká data zanést a která odstranit. Na druhou stranu se však může stát, že na konci procesu je výsledek špatný třeba proto, že byla odstraněna informace, která na první pohled nebyla důležitá, ale ve vytvářené simulaci měla hlubší význam [1].

Vytvoření simulace

Dalším krokem je vytvoření simulace dle připraveného návrhu. I v tomto kroku mohou vznikat chyby kvůli nepřesnostem, které přináší vybraná technologie nebo třeba nesprávné použití informací z návrhu. Po vytvoření modelu je nutná verifikace a validace [2]. Verifikace znamená, že vytvořený model odpovídá návrhu a chová se tak, jak bylo zamýšleno. To však ještě nemusí znamenat, že je model správný, ale pouze to, že byl návrh správně převeden do praxe. Proto je nutná validace, kde se zjišťuje, jestli vytvořený model odpovídá skutečnému systému, tedy jestli byl navržen a vytvořen správně a jestli poskytuje stejné výsledky jaké by dával skutečný systém při stejných počátečních

podmínkách. Validace je velice náročný proces, a zvláště u složitějších modelů není možné zkontrolovat veškeré výstupy pro každou možnou počáteční podmínku. Z toho důvodu se jedná spíše o míru validity modelu.

Experimentování s modelem

Poté co je ověřeno, že se model chová tak jak bylo požadováno, je možné s ním začít experimentovat. Jedná se o zjišťování nových informací na základě vstupních podmínek. Tyto získané informace mohou být dosud neznáme poznatky, které nebylo možné získat jiným způsobem. Získané informace je možné zanést do prvního kroku, tedy do návrhu a vytvořit lepší model s novými dříve neznámými informacemi. Tento proces se může mnohokrát opakovat a simulační model se tak může postupně velice zdokonalit.

2.2 Rozdělení simulací

Simulace je možné dělit z několika hledisek do různých kategorií [1], [18]. Tyto kategorie nejsou striktně dané a mohou se vzájemně prolínat. Dělení může být například následující:

Podle dynamičnosti

- Statická – výsledek simulace je pokaždé stejný
- Dynamická – výsledek se může měnit v čase

Podle času událostí

- Diskrétní – události probíhají skokově a jsou definované pouze určité okamžiky
- Spojitá – simulaci je možné pozorovat v kterémkoliv okamžiku

Podle složitosti

- Jednoduchá – předvídatelné a jasně definované chování
- Komplexní – v simulaci je složité zaznamenat všechny informace a vztahy

Agentové modely

Jednou z možností, jak vytvořit komplexní simulační model, je vytvořit ho jako takzvaný agentový model. Agentové modely se používají v situacích, kdy se jedná o napodobeninu komplexních systémů, kde by nebylo možné popsat všechny zákonitosti a chování. Při tvorbě takového modelu se zformuje prostředí, které má svoje vlastnosti a zákony. Vedle toho se vytvoří „jednoduché“ komponenty – tzv. agenti, kterým je přiřazen relativně jednoduchý popis chování. Poté co jsou tito agenti puštěni do simulace, chovají se v souladu s tímto popisem, ale zároveň komunikují mezi sebou a interagují s prostředím. Díky tomu je možné vypozorovat nově vznikající situace, které nebyly zanesené do modelu a které by někdy ani nemuselo být možné do analytického modelu zanést.

Příkladem takového fungování v reálném světě může být chování hmyzu, kde z pozorování jednotlivce není možné získat takové informace jako při pozorování celku [4].

2.3 Využití

Simulaci je možné využít v mnoha oborech ať už jde o výpočet chování materiálu za určitých podmínek, předpověď ekonomického modelu nebo třeba pouhá vizualizace.

Výcvik

Pro výcvik schopností je možné vytvořit různé trenažery ať už fyzické nebo virtuální. Je mnohem levnější vytvořit pouze část celku a zbytek nasimulovat nebo dokonce vše zobrazit například jen pomocí virtuální reality [7]. Uživatel pro zlepšování svých schopností nepotřebuje vědět co se děje na pozadí. Pro něj je důležité pouze to, aby se vše, s čím přijde do styku, chovalo stejně jako ve skutečnosti. Navíc v případě drahých strojů by jejich opakované ničení, z důvodů nedostatečných zkušeností mohlo být finančně náročné [2].

Jednou z oblastí, kde se hodně využívají simulátory je vojenský sektor. Hlavním důvodem je právě to, že vojenská technika je drahá. Začínajícího pilota není potřeba posadit hned do reálného letadla, ale stačí vytvořit trenažer kokpitu, který obsahuje stejný funkční panel, jaký je ve skutečném letadle. To, že „zbytek“ letadla chybí, není pro uživatele důležité. Všechny ostatní podmínky, jako je například chování letadla za určitého počasí, je možné nasimulovat tak věrně, že pokud uživatel zvládne ovládat stroj v simulátoru, bude mnohem lépe připraven na stroj skutečný [17].

Ve zdravotnictví nejde jen o drahé vybavení, ale také o lidské životy. Lékaři si díky virtuální realitě mohou například procvičit operaci jen za pomocí brýlí, dvou ovladačů a aplikace, která to umožňuje nebo mohou pracovat s reálnými robotickými nástroji na figuríně pacienta. V takovém případě tedy lékař nepracuje s trenažérem, ale operaci provádí skutečným nástrojem na umělém těle, které má nasimulované fyziologické vlastnosti skutečného člověka [25].

Strojírenství

Ve strojírenství se simulace uplatňuje zejména při konstrukci nových výrobků a při automatizaci. V současnosti se prototypy strojů nebo jejich součástek většinou vytvářejí pomocí 3D modelů. Díky tomu, že je model v digitální podobě, je možné pro navržené součástky spustit různé simulace, které odhalí potencionální nedokonalosti za různých podmínek, ať u jde o kvalitu materiálu jeho životnost nebo funkcionalitu v rozdílných prostředích.

Značný prostor pro využití simulací je také v oblasti robotizace. Správné návyky chování robotů, např. trasy pohybu, překážky, nebo umístění skladů komponent se kterými budou pracovat,

se roboti mohou naučit předem v simulovaném prostředí. Po jejich nasazení do reálného prostředí je pak předpoklad, že budou fungovat správně, a tak se předejde potencionálním finančním ztrátám [10].

Předpovědi a možné scénáře

Díky tomu, že simulovaný model může být spouštěn mnohokrát a lze zrychlovat nebo zpomalovat jeho tok času, je možné zjistit, jaký dopad mohou mít určité události ve vzdálené budoucnosti. Tyto výhody lze využít například v oblasti financí, kde se dá díky simulacím podmínek zjistit směr, kterým se bude ubírat cena položek na trhu.

Ve vědě je zase možné simulovat třeba chemické reakce, běh vesmíru nebo vznik světa. To vše díky tomu, že čas v simulaci je relativní a není nutné čekat miliony let. Díky simulaci je také možné vytvořit např. model šíření epidemie nebo paniky a následně dopředu navrhnout ochranná opatření tak, aby se situaci dalo včas zabránit nebo snížit její dopad [22].

U všech těchto globálních, dlouhodobých nebo rychle se měnících scénářů je problém, který byl zmíněný již dříve a to ten, že při vytváření simulace nemusí být dostupné všechny informace. Navíc u takto velkých simulací zasahuje mnoho proměnných, které mohou výsledek někdy významně ovlivnit.

Zábava

Mnohé z předchozích výhod simulací je možné využít také v zábavním průmyslu. Zde lze vytvořit přesnou napodobeninu reálného světa nebo naopak svět čistě imaginární, ve kterém mohou platit dokonce jiné fyzikální zákony. Ačkoliv u těchto simulací nebude potřeba takové přesnosti jako například ve zdravotnictví, stále se jedná o napodobeninu prostředí a situací, ve kterých fungují předepsané vzorce a při určitých vstupních podmínkách vznikají adekvátní výstupy.

Vhodným příkladem pro využití simulace je třeba model, ve kterém se hráč snaží vytvořit fungující a prosperující město. Může však také být obyvatelem takového města a provádět zde každodenní činnosti.

Model města

Model města však může být využit i prakticky, například při plánování jeho dalšího rozvoje, jako je třeba tvorba rozvojových plánů nebo koncepcí budování infrastruktury. V těchto případech simulace umožňuje nejen nalezení optimálních technických řešení, ale i ekonomické vyhodnocení možných alternativ, které bývá při tradičních způsobech plánování nezřídka složité. Simulace města je většinou velmi náročná záležitost. Vychází z velkého počtu parametrů a musí respektovat řadu zákonitostí. Proto je vhodné na vytváření takového modelu použít dříve zmiňované agentové modely.

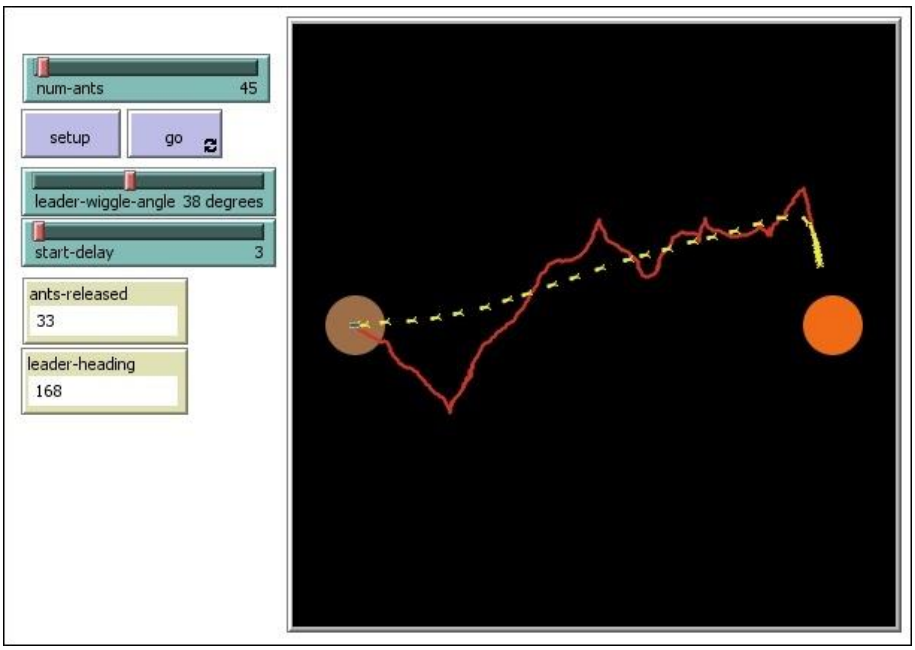
2.4 Software pro tvorbu simulačních modelů

Pro vytváření simulačních modelů existuje velké množství programů. Některé jsou vytvořeny na způsob matematických kalkulátorů, další se zase zaměřují na specifickou oblast, zatímco v jiné oblasti jsou nepoužitelné.

Jednou z možností, jak vytvořit takovou simulaci, je vytvořit ji na způsob počítačové hry. K tomu je vhodné využít, v závislosti na složitosti aplikace, například některý z herních enginů, které jsou pro takové aplikace přímo uzpůsobené. Obsahují mnoho předpřipravených komponent, a proto odstraňují potřebu vytvářet základní herní mechanismy (pohyb, reakce na události, simulace světla a fyziky…) znovu a tvůrce softwaru se může soustředit pouze na svůj problém.

NetLogo

NetLogo je program na vytváření komplexních agentových simulací a jejich vizualizaci. Zároveň je to i programovací jazyk, ve kterém jsou modely vytvářeny. Vznikl z jazyka Logo, který byl určen pro výuku programování a konstruktivního myšlení u dětí. Jedná se o modelovací systém, ve kterém je možné udělat simulace z různých odvětví od modelování davového chování, přes pohyb mravenců (Obrázek 1) až po prosakování materiálu [29].



Obrázek 1 - Ant Lines (NetLogo)

Modelica

Modelica je objektově orientovaný modelovací jazyk, který vznikl v roce 1997. Jedná se o software, který je zaměřený na modelování komplexních systémů pomocí komponent. Využití nalezne například v modelování mechanických, elektrických nebo jiných fyzikálních systémů. Zajímavostí je, že na rozdíl od jiných programovacích jazyků, kde se výraz může vyskytovat pouze na pravé straně přiřazení, v Modelice, je možné mít výraz na obou stranách [28]. Na jeho základě vznikl například open-source software openModelica.

Unity

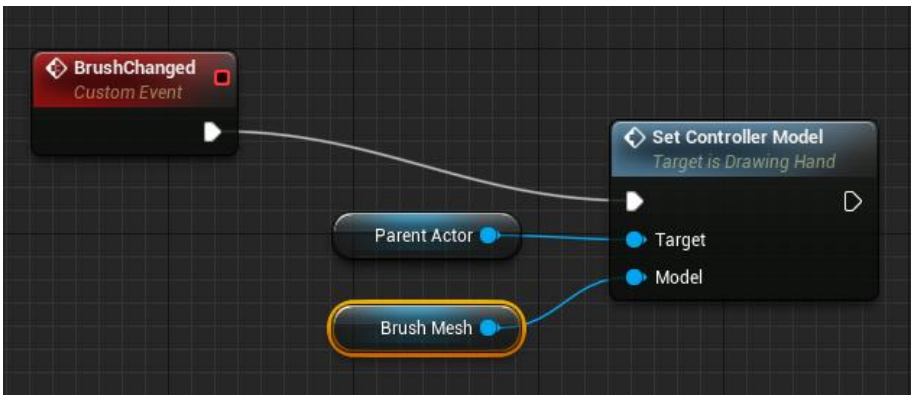
Unity je herní engine vyvinutý v roce 2006 firmou Unity Technologies, původně pouze pro OS X a postupně vylepšovaný a modifikovaný pro různé další platformy. S jeho pomocí je možné vytvářet 2D a 3D hry. Základem vývoje aplikací je vnitřní Unity Script, který je nadstavbou JavaScriptu doplněný o programovací jazyk C#. Podporuje grafické knihovny Direct3D, OpenGL, WebGL a další. Protože patří mezi nejpoužívanější herní enginy, existuje mnoho poznatků, návodů, webových stránek a diskusních fór, kde jeho uživatelé řeší své praktické zkušenosti. Tak vzniklo i mnoho zásuvných modulů a knihoven, které jsou velmi nápomocné při řešení specifických problémů jednotlivých uživatelů [30].

Unreal Engine

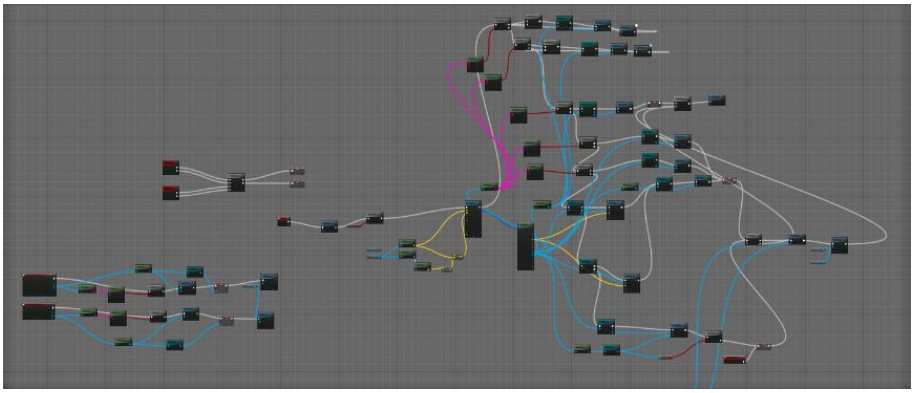
První verze tohoto herního enginu vnikla již v roce 1998 ve firmě Epic Games a původně byla vyvíjena pouze pro „střílečky“. Současná verze je již čtvrtá a její využití je podstatně širší od již zmíněných „střílečekˮ, přes strategické až po masivně multiplayerové hry. Ačkoliv tento engine nebyl původně navržen na vytváření složitých simulací, v dnešní době obsahuje mnoho komponent na takové úrovni, že není problém v něm vytvořit funkční simulaci například silničního provozu nebo funkčního města. Aplikace vzniklé na jeho bázi lze vyvíjet nejen pro osobní počítače s různými operačními systémy a konzole (Xbox, Playstation), ale i pro web či chytré telefony [31].

Tvorba aplikace je možná dvěma způsoby. Jedním z nich je skriptování v Blueprintech. Blueprinty jsou grafické editory znázorňující třídu. Tento způsob funguje na jednoduchém přetahování hotových uzlů na pracovní plochu a jejich následné spojování. Naskriptovaná část pak vypadá jako graf (Obrázek 2). Zároveň odpadá potřeba dlouhého vypisování zdrojového kódu a také se tím současně zajišťuje jistá kontrola, například při přetypování nebo syntaxi. Určitá nevýhoda tohoto způsobu programování se projevuje zejména v případech rozsáhlejší logiky, kdy začne být graf méně přehledný (Obrázek 3). Druhý způsob je klasické programování a psaní zdrojového kódu v jazyce C++. Ten se hodí zejména pro výše zmíněné případy rozsáhlejší logiky. Unreal Engine je

propojen s Visual Studiem od firmy Microsoft, jehož funkce jsou již delší dobu na vysoké úrovni a odhalování chyb se tím značně zjednodušuje.



Obrázek 2 - Jednoduchý graf v Blueprintu



Obrázek 3 - Méně přehledný graf v Blueprintu

2.5 Simulace virtuálního města

Simulační model je vhodné navrhnout dle účelu využití. Pokud bude výsledkem zjišťování vzdáleností a dostupnost mezi jednotlivými místy, není potřeba se zabývat vzhledem a výsledkem bude tabulka čísel. Pokud ovšem bude důležitá vizualizace nebo se bude jednat o aplikaci interaktivní, je nutné celý model rozložit do několika celků. Tyto celky se vzájemně mohou více či méně ovlivňovat.

Město

Při generování města je možné využít mnoha postupů, díky kterým lze získat různě detailní simulace. Jednou z možností je generovat město pomocí strategického plánu rozvoje, tedy tak, jak města skutečně vznikala a jak se dále rozvíjejí [12]. Město může být rozděleno na čtvrtě, kde každá čtvrť bude zaměřena na něco jiného, město může mít základní jádro, od kterého se postupně rozšiřovalo do okolí nebo může být rozděleno na pomyslnou mřížku, kde je jedna buňka jako druhá [19]. Pokud

není důležité, jak město vypadá, nebo kde se nachází jaká budova (například pro simulování dopravy), může se vygenerovat náhodně pomocí šumu nebo jen za pomoci jednoduchých pravidel. Další možností je vytvořit město podle skutečných měst za pomocí leteckých snímků, 3D scanů nebo katastrálních map [5].

Doprava

Pokud se jedná o simulaci skutečně fungujícího města, využije se nejspíš jeden z předchozích postupů. V případě, že je ale potřeba simulovat dopravu, využije se postup opačný. Vytvoří se pravidla pro generování chodníků, ulic, silnic, popřípadě dálnic, a to bude základní vrstva, kolem které se domodeluje některou metodou město jen jako sekundární prvek.

Při vytváření dopravy se mohou uplatnit dva přístupy. Pokud nebude důležitá úplná dynamičnost v chování vozidel, je možné vytvořit statická pravidla, se kterými se budou všechna vozidla chovat stejně nebo s drobnými odchylkami. Budou dodržovat zadané trasy a vše bude umělé. Takové chování je možné simulovat například pomocí celulárních automatů [13]. Tento způsob se hodí například pro dokreslení situace nebo pro jednoduché výpočty pohybu po městě. Takové chování by se dalo využít například v simulátoru řízení auta, kdy je potřeba aby se ostatní vozidla chovala přesně podle předpisů a nezáleží na tom, že jezdí bez cíle.

Druhým přístupem je vytvořit vozidla jako agenty, kteří budou mít jednoduché chování, ale stále budou fungovat jako celek. Přinese to do modelu velkou dynamičnost, ale také i neočekávané chování. Takový model se hodí například v situaci, kdy je potřeba modelovat dopravní špičky kde je variabilita v chování žádoucí [16].

Podobné je to s dopravním značením, kdy je možné ho vynechat a důležitá místa jako rušné křižovatky napevno nastavit. Tím je myšleno, že zde budou jednoduchá pravidla, jako je střídání předností v kruhu nebo určení hlavní a vedlejší silnice a následné střídání předností nebo že každá křižovatka bude mít semafor. Značky zde budou sloužit jako jednoduchý indikátor a nebudou mít hlubší význam. Pro takto „statické“ chování je možné opět využít celulárních automatů, kde je chování simulace diskrétní [6].

Pokud by byla požadována dynamičnost a plné využití agentů, bylo by možné přiřadit značkám pravidla a agenti by na toto značení reagovali. Záleželo by zase na typu simulace, jak vysoká přesnost detekce by byla vyžadována a jestli by bylo dovoleno agentům dělat chyby nebo se rozhodovat. Dále by bylo možné udělat agenty variabilní například tím, že bude každé vozidlo mít trochu jiné chování, bude mít odchylky v rychlost atd.

Chodci

Dalším prvkem, který je možný řešit agentovým modelem je chování chodců. Dalo by se říct, že je to velice podobná oblast jako v případě vozidel. Rozdíl je v tom, že chování chodců je orientováno spíš na skupiny než na jednotlivce. To neznamená, že není možné modelovat jednotlivce, jen je to mnohem náročnější a opět to záleží na detailnosti simulačního modelu. V některých aplikacích ostatní chodci reagují na jednotlivce například pokřikem když běží a vrazí do nich nebo se třeba ohradí proti tomu, že je delší dobu sleduje. Je tedy možné agentům přidat i psychologické a sociální prvky. Naopak pokud je potřeba pouze zjistit, kde by bylo nejlepší umístit nouzové východy, aby se dav dostal co nejdříve z hořící budovy, je možné všechno toto chování ignorovat a modelovat pouze pohyb a třeba reakci na kolizi s ostatními agenty [24].

Je také možnost modelovat dav ne jako jednotlivce, ale jako jeden celek. Odpadnou tedy všechny vlastnosti jednotlivce a bude možnost pozorovat pouze celkový dopad davu na simulaci. Takové modely se používají při simulování početných davů, kde informace o jednom konkrétním člověku nejsou důležité [24].

2.6 Existující řešení simulací silničního provozu

Existuje mnoho aplikací, které jsou vytvořeny jako simulátor pro výuku pravidel silničního provozu a řízení vozidla. Některé se zaměřují na jízdu městem, jiné na samostatné úkony jako je například parkování.

City Car Driving

Simulátorem pro řízení vozidla může být například aplikace City Car Driving. Jedná se o velice realistický simulátor pro řízení auta v různých dopravních podmínkách (Obrázek 4). Je možné připojit některá herní zařízení jako je volant a pedály nebo aplikaci spustit ve virtuální realitě. Recenze na tuto aplikaci jsou velmi kladné, nicméně obsahuje pár nedostatků, jako vzájemně kolidující značky nebo chybné chování ostatních účastníků (ovládaných počítačem) v „nečekaných situacích“. V nastavení je sice možné upravit mnoho těchto situací, ale při vyšších hodnotách se stává, že třeba chodec vběhne do plného provozu nebo auto na dálnici okamžitě zastaví na místě nebo se v případě nehody začnou chovat divně všichni okolo. Aplikace také hlásí uživateli chyby, kterých se dopustil, ale také se stává, že je detekována chyba, která se nestala. Nicméně aplikace nabízí tak velkou variabilitu a tolik možností, že zmíněné nedostatky jsou těmito klady mnohonásobně převýšeny [27].

Obrázek 4 - City Car Driving[1]

[1] https://store.steampowered.com/app/493490/City_Car_Driving/

Car Driving School Simulator

Pro chytré telefony s operačním systémem Android a iOS existuje několik aplikací, které jsou si vzájemně velice podobné. Jednou z nich je Car Driving School Simulator (Obrázek 5). Jedná se o aplikaci, na které se uživatel může učit pravidla silničního provozu. Aplikace obsahuje mnoho úrovní, které jsou zaměřené na různé situace. Vždy, když uživatel vykoná nějakou akci, aplikace zobrazí informace a udělí kladné nebo záporné body. Příkladem takových situací může být zapnutí bezpečnostního pásu, jízda na červenou nebo překročení rychlosti. Aplikace má ale některé situace pevně dané. Příkladem může být křižovatka, kde má uživatel podle scénáře odbočit vpravo. Musí tedy dát znamení o změně směru jízdy vpravo, a to i tehdy, když se předtím nedržel předepsané trasy a ke křižovatce přijel jinou cestou. Ačkoliv nyní odbočuje doleva, musí dát znamení o změně směru vpravo, jinak dostane postih. Pokud si uživatel spustí volnou jízdu bez předpřipravených scénářů, kontrola pravidel neprobíhá [26].

Obrázek 5 - Car Driving School Simulator[2]

[2] https://play.google.com/store/apps/details?id=com.boombitgames.DrivingSchoolParking

3 Návrh řešení

Cílem práce je navrhnout, implementovat a otestovat části aplikace, která bude sloužit k výuce pravidel silničního provozu. V následující kapitole je popsán návrh řešení těchto částí. Nejdříve je popsána aplikace a její chování jako celek. Poté jsou popsány jednotlivé části, do kterých je vhodné aplikaci rozložit.

3.1 Popis aplikace jako celku

Výsledná aplikace, jejíž součástí budou algoritmy popsané v této práci, by měla pomoci uživateli s výukou pravidel silničního provozu a chováním na silnicích. Uživatel se posadí za volant virtuálního vozidla, kterým bude projíždět vygenerovaným prostředím uzpůsobeným k výuce. Aplikace bude obsahovat různé dopravní situace z reálného prostředí, ve kterých se bude muset uživatel chovat podle předpisů tak, aby byl způsobilý k řízení vozidla. Důraz bude kladen na správné vyhodnocení těchto situací. V případě chybného chování bude uživatel na chyby upozorněn způsobem, který mu umožní pochopit, co udělal špatně. Aplikace bude mít také možnost zapnout simulaci okolí, do které patří generování ostatních vozidel, chodců a jejich chování.

Práce se nezabývá celou aplikací, pouze některými částmi. Důraz je kladen hlavně na správné vygenerování míst a situací jako je například křižovatka, omezení rychlosti, chování ostatních řidičů, a také na dodržování pravidel silničního provozu a případně postih za špatné chování.

3.2 Pravidla silničního provozu v ČR

Pravidla silničního provozu v ČR upravuje zákon č. 361/2000 Sb. o provozu na pozemních komunikacích [23].

Značky

Ve vyhlášce č. 294/2015 Sb. [20] se lze dočíst o dopravních značkách a jejich rozdělení do kategorií. Také je zde jejich vzhled a definován jejich význam. Dále jsou zde uvedeny informace například o řízení provozu policistou. Značky je možné rozdělit do dvou hlavních kategorií, na svislé dopravní značky a vodorovné. Každá kategorie má ještě podskupiny, ve kterých jsou značky zařazené podle toho, jakou úpravu silničního provozu představují.

Svislé dopravní značky jsou umístěné na pravé straně vozovky (s možností zopakování na levé straně) nebo nad ní a jsou rozdělené do následujících kategorií:

- Výstražné značky – upozorňují na situaci, kde je potřeba zvýšit pozornost (*zatáčka vpravo, křižovatka, nebezpečné stoupání…*).
- Značky upravující přednost – informují řidiče, o tom, jestli je na hlavní nebo vedlejší komunikaci (*hlavní pozemní komunikace, dej přednost v jízdě, přednost protijedoucích vozidel…*)
- Zákazové značky – zakazují určité chování (*zákaz vjezdu všech vozidel v obou směrech, nejvyšší dovolená rychlost, zákaz vstupu chodců…*)
- Příkazové značky – přikazují určité chování (*přikázaný směr jízdy přímo, sněhové řetězy, stezka pro chodce…*)
- Informativní značky – informují o skutečnostech (*obec, dálnice, měření rychlosti…*)
- Dodatkové tabulky – upravují detaily předchozích značek (*tvar křižovatky, délka úseku, druh vozidla…*)
- Určené symboly – jedná se o symboly vyobrazující skutečnost (*hrad, chodec, čerpací stanic LPG…*)

Vodorovné dopravní značení je vyznačené na vozovce nebo jiné zpevněné části pozemní komunikace. Vodorovné dopravní značení je rozděleno následovně:

- Podélné čáry – oddělují jízdní pruhy (*podélná čára souvislá, podélná čára přerušovaná, vodící čára…*)
- Příčné čáry – určují přerušení komunikace (*příčná čára souvislá, přechod pro chodce, přejezd pro cyklisty…*)
- Šipky – šipky určují směr jízdy (*směrové šipky, předběžné šipky*)
- Označení stání a parkovišť – určují způsob a směr parkování vozidla (*stání kolmé, parkovací pruh, omezené stání…*)
- Označení zastávek – označuje zastávky veřejné dopravy (*zastávka autobusu nebo trolejbusu, zastávka tramvaje*)
- Označení zákazu zastavení a stání – označuje místo zákazu stání (*žlutá klikatá čára, zákaz zastavení, zákaz stání…*)
- Ostatní vodorovné dopravní značení – vodorovné značky s dalšími informacemi (*nápis na vozovce, bílá klikatá čára, bezpečný odstup…*)

Křižovatky a semafory

V zákoně č. 361/2000 Sb. o provozu na pozemních komunikacích [23] se píše mimo jiné o chování na křižovatkách. Je možné se zde dočíst o přednostech na hlavních a vedlejších silnicích, úpravě pravidel světelnými signály a také o přednosti zprava při nestanovení přednosti dopravním značením. Následně jsou vybrány čtyři důležité body, týkající se křižovatek:

- *§ 2 písmena w zákona 361/2000 Sb. Křižovatka je místo, v němž se pozemní komunikace protínají nebo spojují; za křižovatku se nepovažuje vyústění polní nebo lesní cesty nebo jiné účelové pozemní komunikace na jinou pozemní komunikaci.*

- *§ 2 bod 1 zákona 361/2000 Sb. Při řízení provozu na křižovatce se užívá zejména světelných signálů tříbarevné soustavy s plnými signály nebo se směrovými signály.*

- *§ 22 bod 1 zákona 361/2000 Sb. Řidič přijíždějící na křižovatku po vedlejší pozemní komunikaci označené dopravní značkou "Dej přednost v jízdě!" nebo "Stůj, dej přednost v jízdě!" musí dát přednost v jízdě vozidlům nebo jezdcům na zvířatech přijíždějícím po hlavní pozemní komunikaci nebo organizované skupině chodců nebo průvodcům hnaných zvířat se zvířaty přicházejícím po hlavní pozemní komunikaci.*

- *§ 22 bod 2 zákona 361/2000 Sb. Nevyplývá-li přednost v jízdě z ustanovení odstavce 1, musí dát řidič přednost v jízdě vozidlům nebo jezdcům na zvířatech přijíždějícím zprava nebo organizované skupině chodců nebo průvodcům hnaných zvířat se zvířaty přicházejícím zprava.*

Znamená to tedy, že nejvyšší prioritu na křižovatce mají světelné signály. Pokud jsou vypnuté nebo na křižovatce chybí v platnost přijde dopravní značení. Pokud se jedná o křižovatku, která nemá upravenou přednost ani jedním z předchozích způsobů, použije se pravidlo přednosti vozidel přijíždějících zprava.

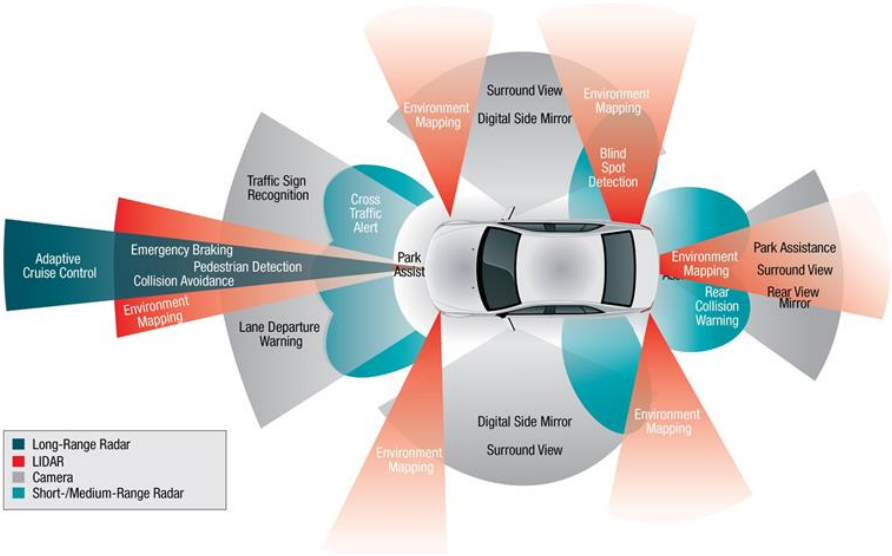
3.3 Autonomní řízení reálných vozidel

Dnes již více typů aut umožňuje alespoň částečně automatizované řízení. K tomu je ale potřeba, aby auto „vidělo“ to, co v klasickém autě vidí řidič. To znamená, že takové auto musí být schopné pozorovat značky, vozovku, okolí, chodce, ostatní účastníky silničního provozu a mnoho dalších prvků. Poté, co auto detekuje určitou situaci, musí rychle a správně reagovat.

Vidění autonomních vozidel

Zjednodušeně by se dalo říct, že vozidlo sleduje okolí podobně jako člověk, jen k tomu používá kamery. Po celém vozidle je namontováno několik kamer a senzorů, které snímají okolí (Obrázek 6). Snímky jsou následně posílány do počítače, který obrazy kontroluje, snaží se v nich detekovat speciální prvky a následně podle nich upravit chování vozidla [14]. Může se však stát, že vozidlo nezareaguje správně, protože vyhodnotí situaci jinak třeba z důvodu snížené viditelnosti. Proto jsou kromě kamer využita ještě další zařízení a těmi jsou radary a lidary. Radary fungují na principu rádiových vln a lidary využívají k detekci laser. Obě zmíněná zařízení se využívají také při výpočtech vzdálenosti objektů [21].

Zatímco v reálných vozidlech může nastat nesprávné vyhodnocení situace, virtuální vozidlo při dotazu na objekt dostane jednoznačnou odpověď. Kapitola (4.8) popisuje, jak byla řešena detekce objektů virtuálním vozidlem.



Obrázek 6 - Detektory na autonomním vozidle[3]

3.4 Komponenty simulace dopravní situace

V chytrých automobilech se informace o okolním prostředí získávají z kamer. Získaný obraz je poté použit jako vstup do algoritmů, které vyhodnotí situaci. Znamená to, že přesnost vyhodnocení závisí na viditelnosti, shody sledovaného objektu s dříve naučenými a na přesnosti algoritmu. Toto při detekci ve virtuální aplikaci odpadá, protože každá komponenta je přesně definována (značka, semafor, přechod, křižovatka…) a pokud se projíždějící vozidlo „zeptá“, dostane jednoznačnou odpověď. U všech dále zmiňovaných dopravních situací bude nutný nějaký způsob detekce. Jednou

[3] https://towardsdatascience.com/how-to-make-a-vehicle-autonomous-16edf164c30f

z možností je, že na vozidle bude muset být umístěna sledovací komponenta, která detekuje prvek v případě přiblížení vozidla. Také by mohl být aplikován opačný postup, kdy dopravní prvek bude detekovat přijíždějící auto a sám se ohlásí a upozorní vozidlo na jistá omezení a úpravy silničního provozu. Stejně jako v předchozím bude zapotřebí detekční komponenta.

Dopravní značení

Na rozdíl od detekce dopravního značení v chytrých autech bude v této aplikaci značku představovat nějaký objekt typu značka, který bude mít v sobě již zabudovanou informaci o tom, jaká je to značka a co to znamená pro právě přijíždějící vozidlo. Při příjezdu ke značce bude auto stoprocentně vědět o jakou značku se jedná nezávisle na tom, jak vypadá. To je výhodou i v případě přenositelnosti aplikace do různých států, kde je značení pro člověka pochopitelné i v případě jiného vzhledu, ale při počítačové detekci z obrazu by mohla vzniknout nejasnost.

Semafory

Dalo by se říct, že semafor je speciálním druhem dopravního značení. Zásadní rozdíl je ale v tom, že toto „značení“ mění svoje chování v čase. Proto bude nutné detekovat, že se chování změnilo. Další věcí, které bude potřeba věnovat pozornost je to, že semafory bývají na křižovatkách ve skupinách. Jednotlivé semafory spolu kooperují a přepínají se podle určitých pravidel.

Křižovatky

Křižovatky jsou jedny z těch složitějších situací, protože se na nich mnohdy kombinují různé prvky upravující pravidla. Při detekci správného projetí křižovatkou bude nutné zkombinovat všechna pravidla dohromady takovým způsobem, aby se vzájemně nevylučovala a aby je bylo možné všechny odchytit. Nejspíš bude nutné udělat nějaký typ prioritního seznamu, do kterého se uloží všechny detekované prvky a následně se začne vyhodnocovat. Další možností by bylo nechat uživatele křižovatkou projet a kontrolu spustit až na konci úkonu, ale tento způsob by snižoval dynamičnost a kvalitu výuky.

Bude nutné vytvořit nějakou základní komponentu nebo předpis, ze kterého se budou křižovatky generovat, aby vznikla jistá variabilita a aplikace neobsahovala pouze několik základních statických dopravních situací.

Detekce vozovky

Aby bylo možné uživatele upozornit nebo penalizovat za nedržení se v jízdním pruhu, je nutné vytvořit komponenty, které dokážou detekovat správnou pozici na vozovce. Jak již bylo zmíněno,

reálná detekce vozovky, stejně jako jiného značení, probíhá z kamerového záznamu. V navrhované aplikace bude opět zjednodušení v tom, že odpadne tato obrazová detekce a vozovka sama o sobě bude mít k dispozici určité informace. Stejně jako u značek bude nutné vytvořit nějaký detektor, který bude kontrolovat správnou pozici na vozovce. Řešením by mohlo být detekování nějaké osy 3D modelu vozovky či vytvoření virtuální křivky, která bude vozovku znázorňovat. Dále by bylo možné vytvořit neviditelné zábrany po stranách vozovky, kdy průjezd vozidla těmito zábranami vyvolá chybové hlášení. Dalším problémem, na který bude nutné se zaměřit, bude spojování silnic a detekce v těchto místech (například odbočování na křižovatce), protože aplikace bude obsahovat pouze základní prvky silnic, které se budou dynamicky propojovat.

Vozidla

Rozhodování vozidel je možné naprogramovat tak, že mají určitý cíl, ke kterému se chtějí dostat a na cestě musí vyřešit mnoho problémů jako jsou značky, překážky a další vozidla, vypočítat nejkratší či nejrychlejší cestu k cíli atd. Také je možné rozložit prostředí na mnoho uzlů a vždy, když se vozidlo dostane na takový uzel, rozhodne se, kam bude pokračovat dál. Tato možnost by nezaručovala logické chování ostatních účastníků, ale to by ani nebylo nutné, protože cílem není zjišťovat, jak se chová vozidlo po celou dobu jízdy, ale pouze jak budou reagovat vozidla ovládaná počítačem na uživatele a na aktuální dopravní situaci.

Ohodnocení chování uživatele

Uživatel by měl být nějakým způsobem ohodnocen, aby věděl, jaký byl jeho výkon. Aplikace by měla rozlišovat závažnost chybného chování. Kontrolní situace a přestupky by měly být rozděleny do kategorií a každá kategorie by měla být ohodnocena dle závažnosti. Je nutné, aby uživatel věděl, že nereagoval správně, ale zároveň by také měl poznat rozdíl mezi drobným přestupkem a závažným porušením předpisů. Příkladem mohou být dvě situace – nedání znamení o změně směru jízdy a jízda na červenou, které jsou obě chybné, ale každá na zcela jiné úrovni. Na konci jízdy by uživatel měl dostat hodnocení, ze kterého by poznal, jak jízdu zvládl. Součástí hodnocení by měl být také výčet přestupků, které udělal a jejich rozdělení do kategorií podle závažnosti.

3.5 Ovládání a hardware

Aplikace je v dnešní době možné ovládat pomocí mnoha zařízení. Ať už se jedná o standardní klávesnici, různé joysticky nebo speciální hardware. Samotnou kategorií je také virtuální realita, která přináší zcela nové možnosti a zážitky. Pro ovládání navrhované aplikace se jeví jako vhodné

tři možnosti. Klávesnice a myš, herní zařízení (volant, gamepad) a virtuální realita. Každá ze zmíněných má svoje výhody a nevýhody a bude nutné tyto možnosti zahrnout do návrhu aplikace.

Pro jednoduché ovládání aplikace je vhodné vybrat tlačítka tak, aby prstoklad byl přirozený a uživatelé s ním neměli problém. Tlačítek by nemělo být moc, aby i uživatel, který není hráč, zvládl aplikaci ovládat a nemusel jednotlivá tlačítka hledat.

Jedná se o aplikaci, jejíž hlavní částí bude ovládání vozidla. K tomu je potřeba několik vstupů. Pokud by se jednalo o nejdůležitější minimum (velká část věcí bude automatizována), bude potřeba ovládat rychlost, brzdu, zpátečku a zatáčení doleva/doprava. Znamená to tedy, že je potřeba minimálně čtyři až pět tlačítek[4].

Pokud by součástí aplikace bylo i rozhlížení v autě, což by bylo vhodné, je nutné přidat ještě způsob ovládání kamery. To by se dalo vyřešit pomocí myši v případě klávesnice a přirozeným pohybem hlavy v případě virtuální reality. Problém by mohl nastat u herních zařízení, kde by ovládání kamery muselo být nastaveno na tlačítka. Toto ovládání by nebylo tak volné jako u předchozích dvou a také ne všechna herní zařízení typu volant mají dostatek tlačítek.

Další problém nastává na klávesnici, kdy tlačítka mají pouze dvě hodnoty – stisknuto/nestisknuto. Nedá se tedy ovládat intenzita. To je problém hned na několika místech. Ve chvíli, kdy uživatel bude chtít zrychlit a přidá plyn, stiskne tlačítko a rychlost bude stoupat. Pokud bude chtít držet konstantní rychlost, bude muset tlačítko pustit a po chvilce opět stisknout. Stejný problém bude i v případě zatáčení a ve všech dalších úkonech, ve kterých je nutné ovládat intenzitu. Tímto problémem naopak netrpí herní zařízení, protože ta mají většinou několik tlačítek, která vrací míru stisknutí a na takovéto úkony se velice hodí.

Vznikající část aplikace nemá učit uživatele, jak ovládat vozidlo, jak řadit nebo jak se rozjíždět. Na to by se hodil jiný typ simulátoru, který by měl totožné hardwarové prvky s těmi ve skutečném vozidle. Tato aplikace se zaměřuje na dodržování pravidel silničního provozu. Čím víc se bude uživatel soustředit na správné mačkání tlačítek, například kvůli dodržování maximální rychlosti, tím méně se bude soustředit na okolní situaci. Kdyby se naopak výsledná aplikace, která by obsahovala vše potřebné pro výuku, připojila do trenažeru, připomínající skutečné vozidlo, uživatel by se mohl učit nejen pravidla silničního provozu, ale také by měl k dispozici skutečné zařízení, které by výuku ještě posílilo [2].

3.6 GUI

Při návrhu uživatelského rozhraní je nutné myslet na to, že uživatel se má soustředit na okolní situace, a ne na zobrazené prvky. Protože se jedná o výukovou aplikaci, bude zároveň potřeba některé

[4] Ve většině závodních her je stejné tlačítko pro brzdu i pro zpátečku

informace uživateli nějakým způsobem zvýraznit, protože virtuální aplikace nebude dokonalou simulací reálného světa a tyto informace by bez zvýraznění mohl přehlédnout. Z toho důvodu bude GUI jedna z mála možností, jak nahradit skutečného učitele či rádce. Protože výuka bude probíhat z velké části prostřednictvím zobrazených informací, bude nutné udělat kompromis mezi oběma požadavky, tedy zobrazit maximum informací při co nejmenším rozptýlení uživatele.

Zobrazovací prvky se budou chovat a vypadat jinak na monitoru a jinak v headsetu virtuální reality. To bude nutné vzít v potaz a návrh by měl už od první chvíle počítat s možnost různého způsobu zobrazení. Ačkoliv budou prvky různé, mělo by jejich chování a zobrazení být vzájemně co nejpodobnější nezávisle na zobrazovacím mediu.

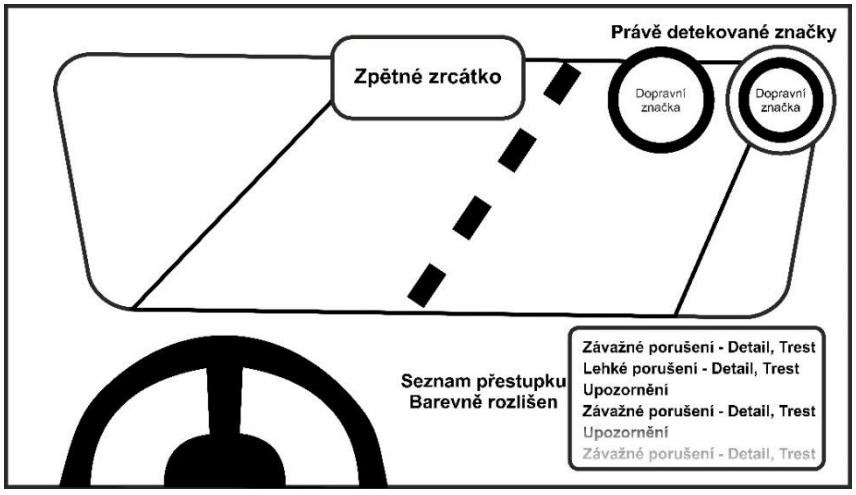
Prvky z uživatelského rozhraní by mělo být možné vypínat v závislosti na zapnutém režimu výuky. Je nutné, aby zapínání či vypínání některých prvků nemělo vliv na ostatní zobrazované prvky, které si musí zachovat svoji pozici nebo velikost v různých výukových režimech.

Jednou ze zobrazovaných informací by měly být dopravní značky. Značky jsou důležitou součástí silničního provozu a z velké části řídí chování na silnicích. Při průjezdu uživatele kolem značky by se mu tato značka měla krátce zobrazit. Tím upozorní uživatele na svoji existenci, a tak se sníží riziko jejího přehlédnutí.

Součástí aplikace by mohla být zobrazena GPS navigace, oznamující cestu, kterou má uživatel pokračovat. To by mohlo být uskutečněno pomocí jednoduchých směrových ikon nebo například pomocí minimapy, která by ovšem zabírala prostor v uživatelském rozhraní.

Důležitými údaji, které bude nutné předávat uživateli jsou informace o porušení předpisů nebo chybách, kterých se dopustil. To by mohlo být zobrazeno opět jako ikona nebo popis. Jinou možností by mohlo být informaci o chybě uložit a ke všem chybám by se uživatel dostal později.

Na obrázku (Obrázek 7) je zobrazeno rozložení nejzákladnějších komponent, které by měly být obsaženy v GUI. Toto je pouze hrubý návrh, který bude upraven, což popisuje podkapitola o zobrazování informací (4.11).



Obrázek 7 - Návrh GUI

4 Konkrétní řešení

Cílem práce nebylo vytvořit kompletní aplikaci pro výuku pravidel silničního provozu, ale navrhnout a vytvořit postupy, které by bylo možné v takové aplikaci využít. Aby se nejednalo o statickou aplikaci, která bude vždy stejná, jednotlivé prvky byly vytvářeny tak, aby se daly ovlivňovat vstupními parametry a vznikaly nové situace při zachování pravidel silničního provozu.

4.1 Výběr vhodného softwaru

Jak již bylo popsáno v úvodní části, ze současně dostupných softwarů se jeví jako vhodné Unity a Unreal Engine 4, protože oba obsahují komponenty pro vytváření her. Po jejich vyzkoušení byl vybrán Unreal Engine 4. Jeho výhodou je zejména větší rozšířenost v různých skupinách uživatelů (studenti, malí vývojáři, ale také velké společnosti jako je například Square Enix nebo Capcom). Další výhodou je snadnější ovladatelnost a lépe navržené a přehlednější vývojové prostředí. Velice dobře je také zpracován způsob programování a skriptování, kde řadu základních programovacích struktur lze vytvořit díky systému Blueprintů pouze několikerým kliknutím a zároveň je možné průchod naskriptovaným grafem sledovat za běhu, což značně usnadňuje hledání chyb.

Jednou z vestavěných komponent Unreal Enginu je také vozidlo, které již obsahuje základní prvky jako chování motoru, ovládání kol nebo třeba změnu rychlosti dle aktuálních otáček. Díky tomu nebylo nutné vytvářet vozidlo od začátku. Komponenta obsahuje pouze základní parametry a chování. Pokud by byla potřebná detailní simulace vozidla, bylo by nutné jej přepracovat nebo vytvořit zcela vlastní.

4.2 Tvorba komponent

Aplikace byla vytvářena s důrazem na rozšiřitelnost a znovupoužitelnost. Bylo nutné jednotlivé prvky aplikace sestrojovat tak, aby se daly seskupovat, byly na sobě nezávislé, ale zároveň aby mezi sebou mohly komunikovat. Z toho důvodu byla navržena rodičovská struktura, kde byla vytvořena základní třída, která obsahuje pouze společné komponenty a jejíž instance se v prostředí aplikace nevyskytuje. Vše funguje na způsob dědičnosti, který se využívá v klasických objektových programovacích jazycích. Díky tomu byla velice zjednodušena detekce komponent a nebylo nutné ověřovat každý prvek zvlášť (značka rychlost, značka přednost, značka město…). Detekce sestávala pouze z dotazů na základní třídy a pokud některý z algoritmů náhodou potřeboval detailní informace, přetypování a následné zjišťování detailů o potomkovi prováděl už sám. Dalším principem, díky čemuž bylo docíleno znovupoužitelnosti, je to, že většina komponent byla vytvářena jako modul,

který fungují i samostatně. Vlastnosti komponent jsou z velké části ovlivňovány parametry. Díky tomu je možné velice jednoduše upravovat jejich chování, a přitom není potřeba vytvářet duplicitní bloky kódu nebo zcela nové třídy. Příkladem může být semafor, který se vyskytuje na křižovatce, ale i na cestě bez křižovatky a stejně tak může a nemusí být jeho součástí přechod. Pokud je semafor součástí křižovatky, křižovatka si obstará, co potřebuje. Existence přechodů závisí pouze na jediném přepínači, jehož sepnutím se lehce změní chování semaforu, ale v základu se jedná o tu stejnou komponentu se stejnou základní funkcionalitou a tou je přepínání světel.

Mezi nejdůležitější a také nejčastěji používané komponenty aplikace jsou vozidlo, vozovka, dopravní značení, semafor a křižovatka. Jejich chování je podrobně popsáno v následujících kapitolách.

4.3 Vozidlo

Základní třída vozidla byla zděděna z existující komponenty pro vozidlo, která je již zabudovaná v Unreal Enginu. Třída vozidlo obsahuje pouze základní chování společné pro všechna vozidla. Mezi prvky, které jsou stejné pro všechny potomky je například vzhled vozidla, chování motoru, proměnné znázorňující rychlost a sepnutí blinkrů. Ke komponentě vozidla je připojen i detekční box (*Overlapper)*, který vždy když do jeho prostoru vstoupí nějaký objekt, spustí událost. Díky tomu je možné detekovat např. dopravní značku nebo blížící se jiné vozidlo. Třída vozidlo původně zpracovávala i veškeré kontroly pravidel silničního provozu, ale nakonec byla pro tuto kontrolu vytvořena samostatná kontrolní třída (*Checker*) a vozidlo vždy, když nastane nějaká zvláštní událost, dá zprávu kontroloru a více se o událost nestará.

Všechny komponenty byly nejdříve vytvářeny pro autonomní vozidlo. Důvod byl ten, že takové vozidlo se, až na prvky náhody, chová pokaždé stejně tak, jak by se měl chovat i skutečný řidič na silnicích. Tedy podle zadaných pravidel. Poté, co vše fungovalo pro autonomní vozidlo, byla s drobnými úpravami vytvořena třída pro uživatele a díky tomu, že obě třídy mají společného rodiče, ostatní komponenty reagovaly na uživatelovo chování velice dobře. Na obrázku je znázorněn diagram této třídy (Obrázek 8).

Autonomní vozidlo

Autonomní vozidlo představuje všechna vozidla, která se vyskytují v prostředí aplikace a nejsou ovládána uživatelem. Slouží pouze k simulaci dopravní situace. Chování autonomního vozidla je značně zjednodušeno. V okamžiku, kdy vznikne instance této třídy, vozidlo se pokusí nalézt nejbližší silnici, po které se začne pohybovat tak, jak mu dovoluje maximální rychlost a aktuální pravidla silničního provozu. Díky kontrolám, které neustále probíhají v připojeném *Checkeru*, vozidlo ví, kdy

může jet a kdy musí zastavit. Momentálně je toto autonomní vozidlo omezeno na striktní dodržování pravidel, takže k jejím porušování nedochází.

Při příjezdu ke křižovatce se vozidlo náhodně rozhodne, kterým výjezdem chce křižovatku opustit. Následně se spustí kontrola dopravní situace v křižovatce a ve chvíli, kdy mu to pravidla dovolí, projede křižovatkou ke zvolenému cíli a napojí se na přilehlou silnici. Informace o zvoleném výjezdu je uložena v proměnné, proto není problém tuto hodnotu změnit z náhodné na požadované číslo výjezdu nebo generovat výjezd dle jiných pravidel. Pokud chce uživatel simulovat konkrétní situaci, může změnit náhodný výběr výjezdu na jím definovaný.

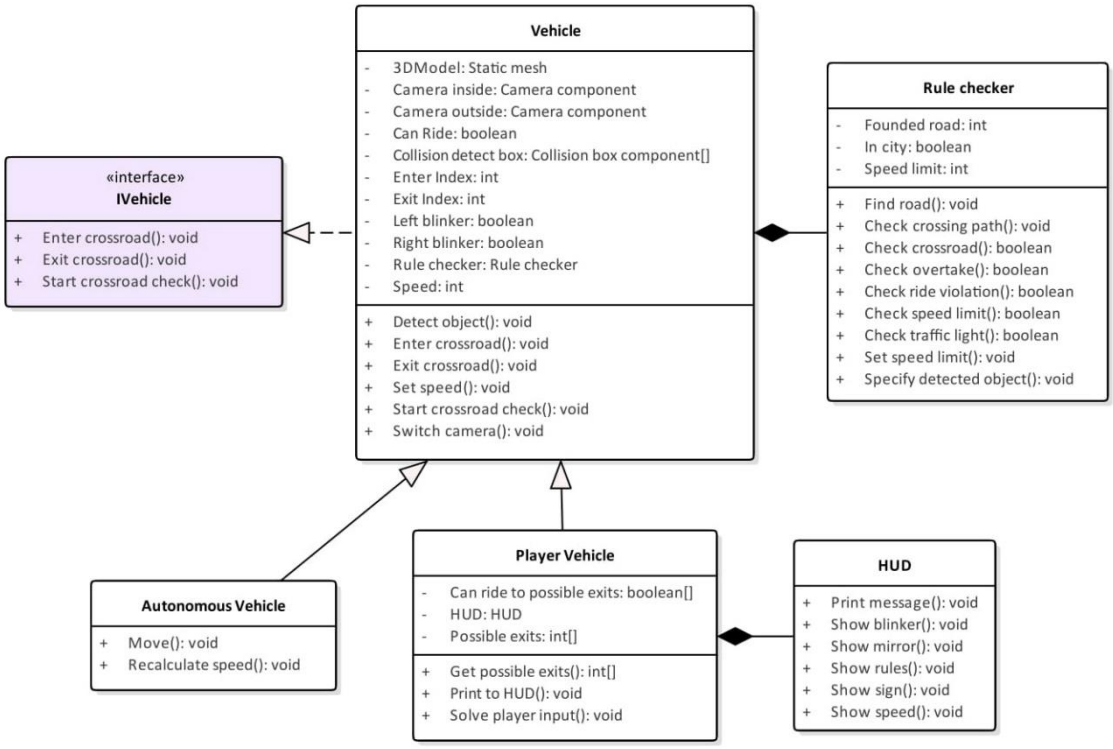
Vozidlo obsahuje funkci *Move*, která se spouští tikem aplikace, což je událost, která se aktivuje s každým snímkem obrazovky. Díky této funkci se vozidlo pohybuje. Nejprve zkontroluje, jestli mu pravidla dovolují jet (proměnná *canRide*), následně přepočítá rychlost dle aktuálních omezení rychlosti a poté se pohne o určitou vzdálenost. Proměnnou *canRide* nastavuje neustále *Checker* dle situace.

Vozidlo uživatele

Vozidlo, které ovládá uživatel, je velice podobné tomu autonomnímu. Stejně jako to, které je řízené počítačem se pohybuje a musí provádět kontroly situací. Rozdíl je ale v tom, že vstup přichází od uživatele, a ani jeho rozhodnutí nejsou předem naplánovaná. Aby vše fungovalo, byla možnost vytvořit zcela samostatnou třídu, která se bude chovat jinak. Bude mít vlastní pravidla, vlastní kontroly a všechno přímo uzpůsobené pro uživatele. Vlastní pravidla by ale musely mít i ostatní komponenty. Veškeré třídy, které pracují s třídou vozidlo by musely mít speciální větev algoritmu přímo pro uživatelovo vozidlo, a ačkoliv by ve výsledku prováděly to stejné, musely by se neustále ptát o jaký druh vozidla se jedná a dle toho spouštět téměř duplicitní kód. Lepším řešením bylo vytvořit třídu pro uživatelovo vozidlo tak, aby se co možná nejvíc podobala vozidlu ovládaném počítačem. Nakonec stejně bylo nutné rozlišit o jaký typ vozidla se jedná, ale v mnohem menší míře. Jednalo se spíš o to, že při porušení předpisů byla spuštěna funkce výpisu na obrazovku nebo se některá funkce nespustila. Ale byly to drobnosti a základ zůstává stejný. Většina tříd nepozná, a ani nepotřebuje vědět, jestli se jedná o vozidlo ovládané uživatelem, nebo o vozidlo ovládané počítačem.

Autonomní vozidlo dostane od *Checkeru* informaci, jestli může jet nebo ne. Tato informace je získaná z okolní situace a například na křižovatce se pracuje s proměnnou obsahující výjezd z křižovatky. Pro jeden vjezd a jeden výjezd je jasně stanovené, jestli vozidlo může jet nebo dává přednost. U uživatele je to složitější v tom, že on nikam nenapíše číslo výjezdu, kterým se chystá vyjíždět. Kdyby to dělal, bylo by vše jednodušší, ale není to dost dobře možné. Jediným vodítkem, jak dát ostatním najevo kam se uživatel chystá jet, jsou ukazatele směru jízdy. Podle jejich sepnutí jsou při příjezdu na křižovatku vypočítané potenciální výjezdy a kontrola probíhá pro každý výjezd. Výjezdy jsou uloženy v poli výjezdů a pro každý prvek samostatně je spuštěna obecná funkce

kontroly, která díky tomu vůbec nepozná, že se na pozadí skrývá nějaké pole. Dostane zadaný vstup a jako výsledek vrátí informaci o tom, jestli auto může jet nebo ne. Stejně jako má autonomní vozidlo proměnnou *canRide*, ve které je uložena informace o tom, jestli může jet, má třída uživatelova vozidla pole proměnných *canRide*, ve kterém jsou uloženy odpovědi pro každou zjištěnou situaci.

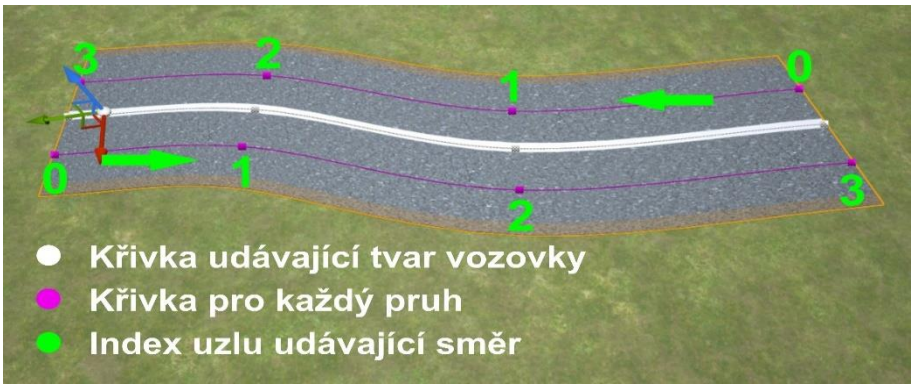


Obrázek 8 - Diagram třídy Vozidlo

4.4 Vozovka

Vozovka je na první pohled jen součást statického prostředí. Ovšem ve vzniklé aplikaci má několik skrytých funkcionalit. Vozovka je vytvořená pomocí *Spline* komponenty, která umožňuje nejen klasickou práci s křivkou, ale také dokáže modifikovat tvar přidružených 3D modelů tak, aby odpovídaly tvarům křivky. Je to ideální způsob, jak vytvořit silnici bez toho, aby bylo nutné vytvářet desítky 3D modelů pro každé zakřivení a nerovnost. Ve skutečnosti pro každou silnici byla vytvořena jedna základní křivka, která určuje tvar a vede středem, a jedna duplicitní křivka pro každý pruh silnice. Tato duplicitní křivka pod každým pruhem určuje směr jízdy a také pozici na vozovce, které se drží autonomní vozidla. Křivka je složena z jednotlivých bodů, které jsou seřazeny a očíslovány. Díky tomu je možné poznat, jestli vozidlo jede po směru nebo v protisměru (Obrázek 9). Vozidlo (přesněji jeho *Checker*) vyhledá nejbližší křivku vozovky a uloží si hodnotu, kterou právě minulo. Pokud se hodnota zvětšuje, jede po směru, pokud se snižuje jede v protisměru. Třída vozovky obsahuje několik přepínačů, díky kterým je možné upravit informace o vozovce. Jedním z přepínačů

je typ vozovky. Vybrat je možné mezi klasickou silnicí, silnicí pro motorová vozidla a dálnicí. Tuto informaci bere *Checker* vozidla v potaz ve funkci přepočítávající maximální rychlost. Dalším přepínačem je možné měnit styl středové čáry. *Checker* díky tomu určuje, jestli bylo možné čáru přejet či nikoliv. V aktuální verzi je možné zvolit mezi čárou plnou a přerušovanou a pravidlo platí pro obě strany. Nicméně nebylo by obtížné rozšířit funkcionalitu o možnost zapnout pro každou stranu jiný typ čáry.

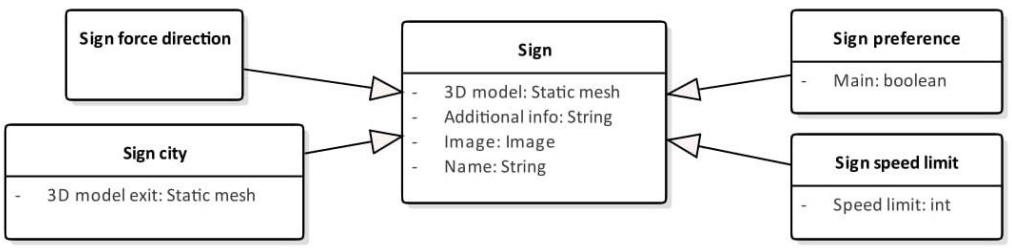


Obrázek 9 - Vozovka

4.5 Dopravní značky

Třídy dopravních značek opět mají rodičovskou třídu, která obsahuje název dopravní značky a vzhled, tedy její 3D model. Později byla rozšířena i o doplňující informaci (např. rychlost) a o obrázek, který se zobrazuje uživateli při jejím detekování na obrazovce.

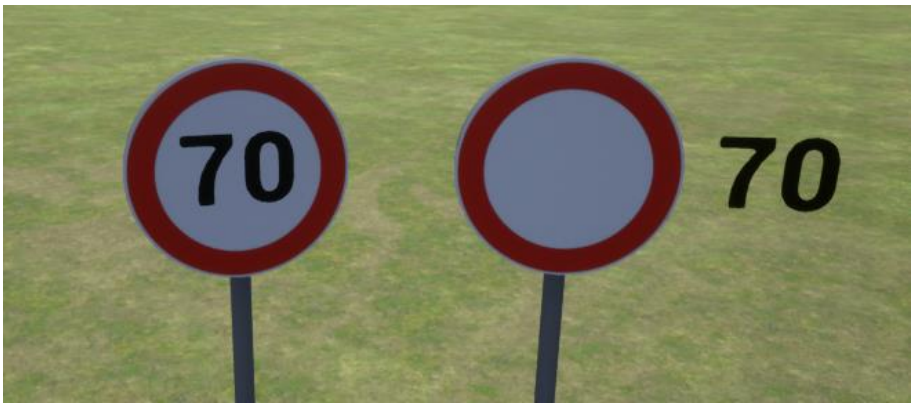
Konkrétní třídy značek byly z počátku rozděleny podle kategorií, které jsou uvedené v zákoně (3.2). To se nakonec ukázalo jako nevhodné, protože například značky upravující rychlost jsou v několika kategoriích. Značky byly následně rozděleny podle své funkcionality (Obrázek 10). Všechny značky upravující rychlost jsou vytvořeny ze stejné třídy. Samostatnou třídu dostala značka oznamující obec. Tato třída obsahuje značky dvě – začátek a konec. V případě detekce této značky se upraví např. rychlost a tato značka je uložena v proměnné vozidla do té doby, než bude detekována značku konce obce. Díky obecnosti této skupiny tříd je možné značky používat nejen samostatně, ale i ve spojení například s křižovatkou.



Obrázek 10 - Diagram třídy Značka

Dynamická značka

Některé značky znázorňují stejné pravidlo s drobnou úpravou. Příkladem může být značka rychlosti nebo značka oznamující začátek a konec obce. Jednou z možností by bylo vytvořit texturu pro každou značku. To není zcela vhodné protože, například pro rychlost by to znamenalo vytvořit minimálně třináct textur v případě, že by rychlost byla po deseti. Kdyby byla vytvořena textura s dostatečným rozlišením pro každou možnou variantu značek, bylo by nutné vymyslet nějakou silnou a kvalitní optimalizační metodu nebo by se zvětšila velikost aplikace. Pokud by aplikace měla umět generovat názvy měst, nebo je brát např. z nějakého přiloženého seznamu a název napojit přímo na texturu, bylo by to sice možné, ale mělo by to dopad na výkon. Tento problém byl vyřešen tak, že textura značky a obrázek značky zobrazující se uživateli na obrazovce je vytvořen pouze ze stejného základu (např. značka maximální rychlosti je značka zákaz vjezdu s číslem). Těsně před značku je následně přidána vestavěná komponenta *TextRenderer*, která je přímo uzpůsobena k tomu, aby zobrazovala text. Díky tomu je ušetřeno mnoho paměti a výkonu a dynamičnost zůstane zachována. Na obrázku (Obrázek 11) je znázorněno oddělení značky a textu.



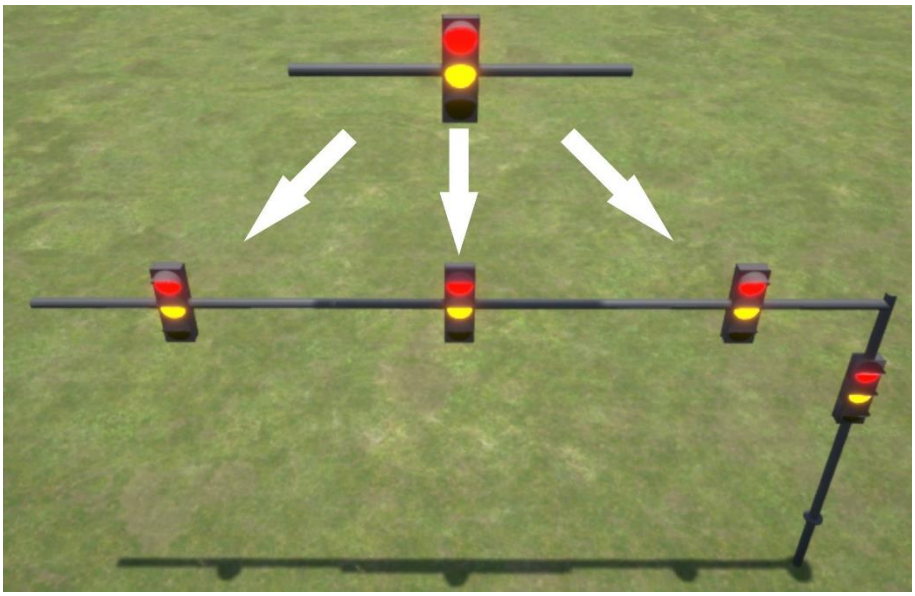
Obrázek 11 – Složená značka rychlosti

4.6 Semafory

Semafory jsou vytvořené na podobném principu jako dopravní značky. Obsahují svůj 3D model a jednoduché obecné chování, a proto je možné je používat nejen ve spojení s křižovatkou, ale i samostatně. Semafor obsahuje proměnnou s číslem pro kolik pruhů je určen. Proto není nutné upravovat třídy nebo dokonce vytvářet nové pro různé počty pruhů na vozovce. 3D model se složen ze stojanu na pravé straně a z menších modelů se světly nad vozovkou. Podle toho, jaké číslo je zadané parametrem se vytvoří příslušný počet semaforů nad vozovkou (Obrázek 12). To stejné platí i pro případný přechod pro chodce, který je také možné vytvořit jen jedním přepínačem. Další proměnnou, která je ve třídě obsažena je pole s časy pro každý stav (zelená, oranžová před zastavením, oranžová před jízdou, červená). Tyto časy slouží časovači pro přepínání stavu semaforu

a jeho barev. Semafor dále obsahuje detekční box a vždy když dojde k přepnutí stavu semaforu, třída informuje všechna vozidla v tomto boxu o novém stavu.

Detekční box má ještě jednu funkcionalitu a tou je detekce opuštění prostoru semaforu. Vždy když vozidlo opustí detekční box, *Checker* vozidla je upozorněn, že k tomu došlo. Podle stavu semaforu *Checker* oznámí uživateli, že měl zpomalit, že vyjel příliš brzy nebo že jel na červenou. Momentálně je semafor zjednodušený oproti skutečným. Proto na křižovatce není žádným způsobem řešena synchronizace a přepínání stavů probíhá tak, že tabulka časů jednotlivých stavů je součtem časů všech semaforů na křižovatce.



Obrázek 12 - Semafor s proměnlivou velikostí

4.7 Křižovatka

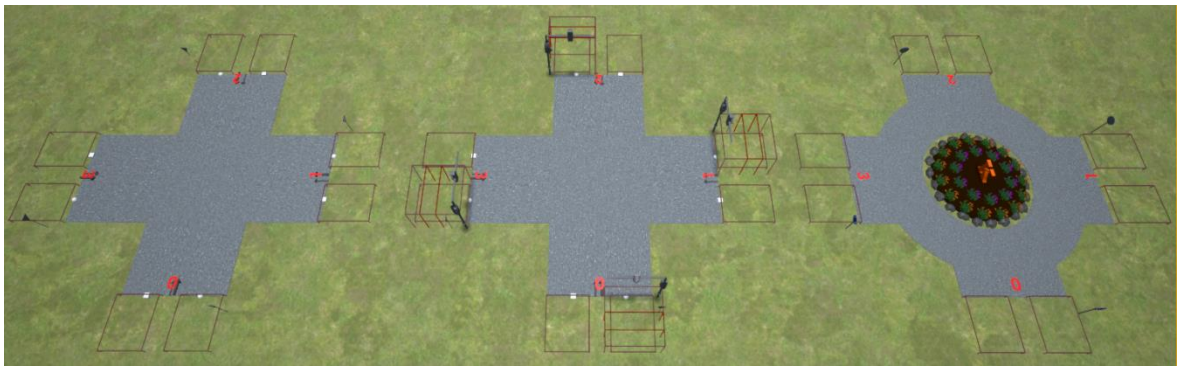
Nejsložitější třídou celého návrhu je křižovatka. Je to proto, že je složena z několika dílčích komponent a také proto, že je zde potřeba detekovat mnoho pravidel silničního provozu. Zároveň musela křižovatka zůstat v obecném tvaru, aby bylo možné měnit její tvar dynamicky (Obrázek 13).

Ke křižovatce je připojena menší komponenta znázorňující vjezd a výjezd. Tato komponenta obsahuje dva detekční boxy v místě, kde končí klasická vozovka a začíná prostor křižovatky. Vždy když vozidlo vjede do prostoru těchto detekčních boxů, spustí se událost v závislosti na typu boxu a následně se spustí překontrolování pravidel všem vozidlům, které stojí na ostatních vjezdech. Každý výjezd je očíslovaný indexem proti směru hodinových ručiček, což se následně používá při kontrole pravidel (vozidlo ví, jaké číslo má jeho výjezd, a tedy jaká pravidla se mají uplatnit při kontrole).

Při vytváření křižovatky je možné přepínačem zvolit, jestli bude obsahovat hlavní silnici a pokud ano, mezi kterými výjezdy povede nebo jestli se má hlavní silnice vygenerovat náhodně.

Zde je rozdíl oproti skutečnosti v tom, že křižovatka a silnice jsou dvě rozdílné věci. Ve chvíli kdy končí silnice, začíná vjezd do křižovatky a obráceně. Z toho důvodu je vjezd do křižovatky tím místem, které obsahuje informaci o hlavní silnici, nikoliv silnice samotná. Podle toho, jestli se jedná o hlavní nebo vedlejší silnici je u vjezdu vygenerovaná příslušná značka automaticky.

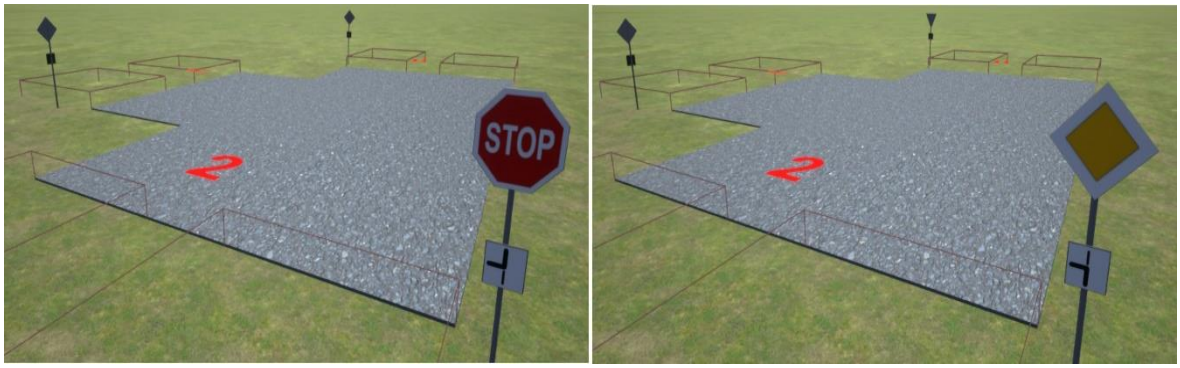
Počet výjezdů je omezen pouze místem v prostoru a jejich pozici je možné změnit. Nicméně základní chování křižovatky obsahuje pravidlo, že pokud je křižovatka složena z více než čtyř výjezdů, je přepnuta na kruhový objezd. Toto chování je možné upravit pomocí přepínačů, díky kterým lze zakázat vytváření kruhových objezdů nebo naopak donutit třídu vytvořit kruhový objezd nezávisle na počtu výjezdů. Velikost kruhového objezdu se mění podle vzdálenosti nejbližšího výjezdu od středu křižovatky. Detailní popis kontroly při průjezdu křižovatkou je rozepsaný v kapitole o kontrolách (4.9).



Obrázek 13 – Stejná křižovatka, různé parametry

Znázornění křižovatky

Protože křižovatka nemá pevně daný tvar, není možné vytvořit standardní texturu znázorňující křižovatku. Pro tento případ byla vytvořena třída dodatkové tabule. Tato tabule je připojena ke značce u vjezdu do křižovatky a pro konkrétní vjezd je vytvořena textura dynamicky (Obrázek 14). Pro každý výjezd je vypočítaný úhel, který svírá s vjezdem, následně je do textury dokreslena čára, jejíž tloušťka je upravena podle toho, jestli se jedná o hlavní nebo vedlejší silnici.



Obrázek 14 - Změna značky a dodatkové tabule dle parametrů křižovatky

4.8 Detekce prvků

Jak bylo zmíněno v kapitole o reálných autonomních vozidlech (3.3), skutečná vozidla se musí spoléhat na kamerové systémy a detektory, díky nimž jsou zachyceny překážky a situace, které se následně vyhodnocují. Tato detekce pomocí „vidění“ ve virtuální verzi vozidel odpadá, protože jakmile je detekován objekt, jsou veškeré informace okamžitě dostupné a nemůže dojít k nepřesnostem nebo zkreslení například z důvodů snížené viditelnosti.

Téměř veškerá detekce probíhá pomocí detekčních boxů, což je komponenta vestavěná v Enginu, která umožňuje detekovat kolizi nebo objekt uvnitř boxu. Vždy když dojde k některé z událostí (*collision, overlap*), může se spustit akce. Tyto detekční boxy jsou využity na mnoha místech. Na vozidle se využívá k detekci všech objektů (další vozidlo, značka, silnice…), dále je využit k tomu, aby vozidlo zjistilo, jestli není příliš blízko překážky a pokud ano, upraví svoji rychlost. Vjezd do křižovatky využívá tento box proto, aby zjistil, jestli zde nečeká nějaké auto a popřípadě mu oznámil nový stav křižovatky. Stejně je tomu i u semaforů, které při změně stavu oznámí novou informaci stojícím vozidlům v prostoru detekčního boxu. Dalo by se říct, že objekty, které získávají informace z okolí, prvky kolem sebe nevidí, ale mohou se jich „dotýkat“.

Jediná detekce, která je vzdáleně podobná té skutečné je v případě uživatelova vozidla, kdy se vypočítá pravděpodobný výjezd podle ukazatele směru jízdy a úhlu, který svírá vjezd, ve kterém uživatel stojí, a výjezdy, které jsou na straně ukazatele směru jízdy (nebo přímo před ním). Další nejednoznačnost může nastat ve chvíli, kdy vozidlo hledá silnici, která není přímo napojená na tu, po které přijel. I v tomto případě se vozidlo „rozhlédne“ a pokusí se nalézt nejbližší vozovku.

Vozidlo, jehož detekční komponenta zjistí přítomnost nějakého prvku se nestará o detaily. Pouze informaci oznámí *Checkeru*, a ten už si objekt a jeho informace zpracovává sám.

4.9 Kontrola pravidel

V prvním návrhu se třída vozidla starala o veškerou detekci a kontrolu pravidel silničního provozu. To se ovšem ukázalo jako ne zcela vhodné a také z důvodu správného objektového přístupu se kontrola pravidel přesunula do samostatné třídy (*Checker*). Stejně jako ve skutečnosti vozidlo pouze jezdí a o vyhodnocení situace se stará člověk nebo nějaká řídící jednotka. Třída vozidla byla upravena tak, že obsahuje detekční box, který snímá okolí a pokud detekuje například značku, oznámí tuto informaci *Checkeru.* Vozidlo samo neví, o jaký objekt se jedná, jen pozná že je to objekt, který má sledovat a referenci na něj odešle na kontrolu do *Checkeru*.

Checker se spouští jednou za čtvrt vteřiny. Bylo by možné spouštět kontroly každý tik, a v reálném vozidle by bylo nutné reagovat co nejdříve, ale v této aplikaci čtvrt vteřiny stačí. Každé

spuštění kontroly provádí nastavení rychlostního limitu, kontrolu přesažení limitu, nalezení nejbližší cesty a na to navazující kontrolu přejíždění do protisměru.

Checker dále obsahuje kontroly, které se spouští pouze pokud nastane nějaká situace. Mezi ně patří detekce kolize a kontrola pravidel v případě, že se vozidlo přiblíží k semaforu nebo křižovatce.

Rychlostní limit

Vozidlo ví, jakou jede rychlostí. *Checker* zase z dostupných informací vypočítá, jaký je aktuální rychlostní limit. Jednoduchým porovnáním zjistí, o kolik byla rychlost překročena a dle výše překročení odešle požadavek na výpis informace uživateli. Aby nebyla informace o překročení rychlosti odesílána při každém testu, je tato možnost po prvním odeslání uzamčena. Odemkne se opět ve chvíli, kdy je rychlost vozidla ve správných mezích.

Diagram nastavení maximální rychlosti je možné vidět v příloze (Příloha č. 2 – Zjištění maximální rychlosti). Následuje zjednodušený popis algoritmu:

- Použije se rychlost omezená dopravní značkou.
- Pokud taková značka nebyla detekována, využije se informace o typu vozovky, po které vozidlo zrovna jede.
- Pokud je vozidlo v obci, rychlost získaná z typu vozovky je upravena pravidlem pro rychlost v obci.

Jízda v protisměru

Díky neustálé detekci vozovky, je známá informace, po které straně vozidlo jede. Vozovka navíc obsahuje i údaj o tom, jestli se na ní může či nesmí předjíždět. Tedy jestli je středová čára plná nebo přerušovaná. Algoritmus vyhledává nejbližší vozovku v určité vzdálenosti, kterou považuje za tu, po které vozidlo jede. Pokud vozidlo přejede středovou čáru, algoritmus detekuje, že nejbližší vozovka je ta v protisměru. Následně vyhodnotí typ středové čáry. Pokud je čára plná nebo uživatel nemá zapnutý blinkr, odešle se požadavek o vypsání informace uživateli.

Detekce kolize

Na vozidle není připnutý pouze jeden detekční box, ale je jich hned několik v různých vzdálenostech. Autonomní vozidlo upravuje rychlost podle toho, který box detekuje kolizi. Tedy podle toho, jak je překážka blízko. Pokud uživatelovo vozidlo narazí do překážky (ať už se jedná o značku nebo jiné vozidlo), je mu na obrazovku vypsána příslušná informace.

Průjezd křižovatkou

Kontrola pravidel začne ve chvíli, kdy vozidlo přijede ke křižovatce a je rozdělena na několik částí.

- Pokud se jedná o křižovatku s kruhovým objezdem, vytvoří se detekční box v blízkosti vjezdu a pokud boxem projíždí vozidlo, je nutné počkat (*canRide* je nastaveno na *false*).
- Pokud se jedná o křižovatku se semaforem, možnost jízdy je upravována aktuálním stavem semaforu.
- Pokud se však jedná o klasickou křižovatku, tedy o většinu případů, je detekce mnohem složitější. Je nutné vědět kterým výjezdem se vozidlo chystá odjet, jestli tam někdo nestojí nebo jestli nikomu nebude křížit cestu.

Nejdříve je zjištěno, jestli se vozidlo, pro které je kontrola spuštěna, nachází na hlavní silnici. Pokud ano, proběhne kontrola výjezdu. V případě, že index výjezdu je menší nebo stejný jako index druhého výjezdu s hlavní, vozidlo může jet bez omezení, protože nikomu nekříží cestu. V opačném případě sice může jet, ale musí si dát pozor, protože by mohlo křížit cestu vozidlu, které přijíždí druhým hlavním výjezdem. V tomto stavu je spuštěna funkce pro detekci křížení.

Pokud vozidlo přijíždí po vedlejší silnici, stejně jako v předchozím případě je zjištěn index výjezdu. Následně algoritmus zkontroluje, jestli je nějaké vozidlo na hlavní silnici. Pokud ano, spustí se funkce pro detekci křížení. V případě, že by se cesty křížily, vozidlo nemůže pokračovat. V opačném případě, nebo pokud na hlavní silnici nikdo není, algoritmus pokračuje dál kontrolou vedlejších silnic. Je-li výjezd první zprava, vozidlo může jet. Pokud ne a na výjezdu po pravé straně čeká vozidlo, je uplatněna přednost zprava a nelze pokračovat. Poslední kontrola zjišťuje, kterým směrem vozidlo pojede. Pokud jede doprava nebo rovně, může jet. Pokud jede doleva, může jet, ale spustí se kontrola křížení. Výše popsaný algoritmus je zobrazen v příloze (Příloha č. 3 – Průjezd křižovatkou).

4.10 Editor situací

Pokud by bylo vytvořené pouze statické prostředí, kde by byla dopravní situace pokaždé stejná, uživatel by se na tom mohl sice učit, ale časem by je znal zpaměti. Také by bylo možné vytvořit velké množství statických situací, které by se měnily. V tom případě by buď muselo být mnoho stejných komponent a možná by musely vznikat i částečně duplicitní třídy, nebo by jich bylo málo a po chvíli by je měl uživatel naučené stejně jako v prvním případě.

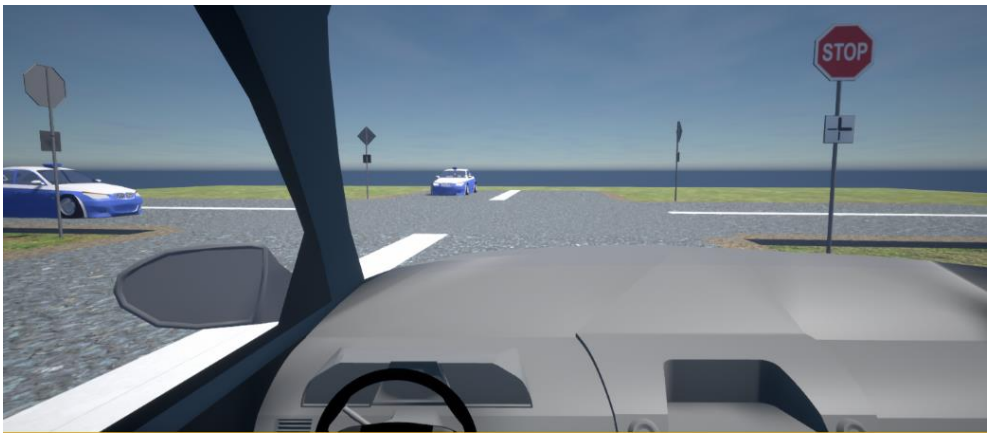
Protož se však již při návrhu počítalo s dynamickým využíváním komponent, jejichž chování se mění pouze pomocí předdefinovaných parametrů, bylo možné vytvořit editor, ve kterém si uživatel

může pospojovat vlastní prostředí, které bude přesně takové, jak bude potřebovat, a zároveň se bude skládat ze standardizovaných prvků.

Typové situace z učebnice

Pomocí tohoto editoru je možné vytvářet i situace z učebnic typu „na křižovatce je následující situace, které auto pojede první“. Uživatel si může připravit situaci dle obrázku (Obrázek 15) a pak projede křižovatkou tak, jak si myslí, že by to bylo správně a hned uvidí výsledek (Obrázek 16). Pokud udělá nějakou chybu, ihned zjistí, co by reálně následovalo, a tak bude účinek větší, než když si jen přečte, že „správná odpověď je za b)“.

Obrázek 15 - Příklad z autoškoly[5]



Obrázek 16 - Příklad vytvořený v editoru

[5] https://www.garaz.cz/clanek/krizovatky-jsou-zaklad-umite-je-21001958

4.11 Zobrazování informací a přestupků

Uživateli se po celou dobu jízdy zobrazují důležité ikony na obrazovce. Mezi ně patří aktuální rychlost a maximální povolená rychlost, pokud je sepnutý ukazatel směru jízdy, tak začne blikat příslušná šipka a v horní části obrazovky je zpětné zrcátko. Pokud je detekována značka, uživatel je upozorněn zobrazenou ikonou značky a jejím popisem v pravé části obrazovky. Zobrazení všech prvků je možné vypnout stiskem klávesy a uživatel může jet bez pomoci.

Pokud uživatel přijede ke křižovatce, zobrazí se na silnici křivka, která znázorňuje pravděpodobný výjezd podle toho, který ukazatel směru jízdy byl sepnutý. Křivka mění barvu podle toho, jestli uživatel může jet (zelená) nebo mu to situace nedovoluje (červená). U autonomních vozidel je tato křivka zobrazena i mimo křižovatku. V hlavním menu je možné zobrazování této barevné křivky vypnout.

Po celou dobu jízdy, se vyhodnocují pravidla silničního provozu. K informování uživatele o chybách, kterých se dopustil, byla vytvořena tabulka s hláškami, která obsahuje popis události. Ve chvíli, kdy *Checker* vyhodnotí situaci jako špatnou, zavolá se funkce pro výpis informací s parametrem chyby, který je klíčem do této tabulky. Tabulka dále obsahuje popis chyby, barvu, kterou se má vypsat a parametr, jestli se má provést i zápis do tabulky přestupků. Například to, že uživatel projede na oranžovou není chyba, ale je na to upozorněn varovnou hláškou, že vyjel příliš brzy, nebo že měl naopak zastavit. Pokud ovšem nedá přednost, výpis bude červený, a navíc bude vytvořen záznam v tabulce přestupků. Text s informacemi je umístěn těsně pod zpětným zrcátkem a po chvíli mizí.

Na obrázku (Obrázek 17) je vidět finální vzhled uživatelského rozhraní. Návrh obsahoval mnoho informací, a proto byly některé prvky přesunuty a okno s přestupky bylo odstraněno. Pro zobrazování informací je určen jeden řádek pod zrcátkem, kde několik vteřin zobrazuje pouze aktuální informaci a pak mizí. Tabulku s přestupky si uživatel může prohlédnout samostatně.

Obrázek 17 - Výsledné GUI

Tabulka přestupků

Pro zaznamenávání přestupků byla vytvořena tabulka, ke které se uživatel dostane jednoduše stiskem klávesy (Obrázek 18). Simulace se pozastaví a uživatel si může v klidu přečíst, jakých přestupků se v průběhu jízdy dopustil. Tabulka přestupků obsahuje paragraf zákona, který byl porušen, zjednodušený popis přestupku, pokutu, kterou je možné dostat a body, o které by takovým chováním mohl uživatel přijít. Díky tomu, že je v tabulce zapsaný i odkaz do zákona si uživatel přesně může najít co bylo špatně a jakému chování by se měl příště vyhnout. Tabulka je vytvořena podle aktuálních znění zákonů, které se však mohou časem změnit. Proto je možné přiložit vlastní soubor ve formátu *csv*, který může obsahovat změny. Aplikace se nejdříve pokusí načíst soubor z disku a pokud není nalezen, načte se předdefinovaná tabulka. To stejné platí i pro tabulku obsahující barevný výpis informací, protože ta s tabulkou přestupků souvisí.

Obrázek 18 - Tabulka spáchaných přestupků

4.12 Ovládání

Jak již bylo zmíněno v části návrhu, bylo nutné rozvrhnout ovládání takovým způsobem, aby se uživatel soustředil na dopravní situaci, a ne na tlačítka. Ačkoliv je možné v Unreal Enginu odchytávat přímo jednotlivé klávesy a tlačítka, vhodnější způsob je vytvořit událost, na kterou se jednotlivá tlačítka namapují. Díky tomu je možné provádět stejný úkon pomocí různých tlačítek nebo různých zařízení. Jak již bylo zmíněno, když uživatel ovládá vozidlo pomocí klávesnice, chybí intenzita stisku. Největší problém nastává při dodržování rychlosti. Z toho důvodu bylo připraveno také ovládání pro herní ovladač Xbox One, který tento problém odstraňuje. Ačkoliv byla navíc pro všechny úkony namapována tlačítka tzv. „univerzálního“ ovladače, neznamená to, že budou všechny herní ovladače rozpoznány.

5 Testování a vyhodnocení

V poslední části práce je popsáno, jakým způsobem byla aplikace testována. Dále jsou zde popsány odhalené chyby, které se nepodařilo ve výsledném řešení odstranit nebo neměly vliv na vytvářené postupy. Nakonec je zde popsán směr, kterým by bylo možné implementovanou aplikaci dále rozvíjet.

5.1 Způsob testování

Testování bylo rozděleno na několik části, z nichž některé běžely současně. Nejdříve se testovaly iterativně jednotlivé komponenty prostředí. Když byla komponenta vytvořena a otestována její funkčnost samostatně bez vlivu okolí, situace byla upravena specificky pro autonomní vozidlo. Pokud vše fungovalo, komponenta byla zobecněna. Nakonec byla zapojena do prostředí, k již hotovým prvkům. Pokud v některém kroku nastala chyba nebo neočekávané chování, bylo nutné se vrátit o krok zpět.

U vozidel probíhalo testování trochu odlišným způsobem. Protože bylo nutné testovat již hotové komponenty, chování vozidla bylo upravováno zároveň s těmito komponentami. V době vytváření komponent mělo vozidlo nepříliš obecné chování, aby bylo možné tyto komponenty testovat. Poté co byly všechny komponenty prostředí dokončeny, testování pokračovalo stejně. V případě, že se třída vozidla chovala dle očekávání, byla zobecněna a následně přidána do celku. Nakonec bylo vytvořeno a testováno vozidlo uživatele, pro které již bylo připravené funkční prostředí, na které muselo správně reagovat. Další část testů vozidel spočívala v tom, že se na uzavřené mapě vytvořily různé dopravní situace (křižovatky, semafory, zatáčky…), následně bylo přidáno několik autonomních vozidel, a nakonec byla celá simulace spuštěna. Toto testování ukázalo, že vozidlo může zvládat určité situace pouze do té doby, než se objeví další účastníci silničního provozu. Také bylo možné odhalit chyby, které se akumulovaly a objevily se až po nějaké době.

Při tomto testování byla odhalena skrytá a nečekaná chyba. Pro kontrolu správné funkčnosti vozidel bylo vytvořeno mnoho kombinací dopravních situací a na jednom místě byl kruhový objezd blízko semaforů. Vozidla, která čekala na semaforu, začala tvořit kolonu, která po chvíli začala zasahovat až do kruhového objezdu a tím v tomto místě zastavila celou dopravu. Testování tedy odhalilo nikoliv chybu algoritmů vozidel, ale nepromyšlený návrh infrastruktury (Obrázek 19).

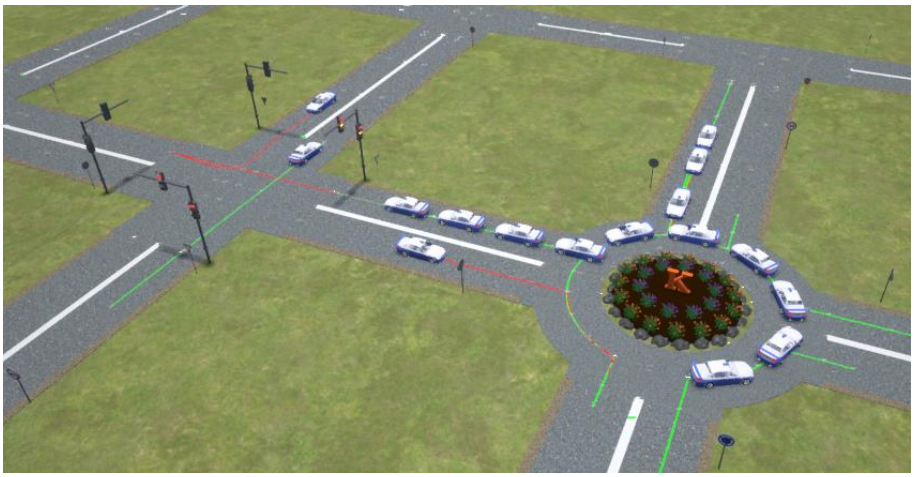
Ve chvíli, kdy byly všechny komponenty dokončeny, bylo nutné, aby aplikaci vyzkoušeli i reální uživatelé. Tento krok byl nezbytný hlavně kvůli zpětné vazbě a také proto, že uživatel, který nevytvářel komponenty, neví, jak se má chovat nebo co nesmí dělat, aby aplikace fungovala správně. Problém při uživatelském testování byl v tom, že aplikace nebyla vytvářena jako celek, ale zaměřuje

se pouze na některé prvky ze silničního provozu. Častým nedostatkem byla právě absence prvků, které by celková aplikace pro výuku obsahovala, ale nejsou předmětem této diplomové práce. Často zmiňovanými nedostatky byl například vzhled ať už prostředí nebo vozidla. Některé poznatky od uživatelů ale byly na místě a na jejich základě došlo k úpravám.

Jednou z úprav prošlo uživatelské rozhraní, které bylo oproti návrhu přeplněné informacemi a uživatel je stejně nezaznamenával, protože jich bylo moc. Úprava byla provedena rozdělením uživatelského rozhraní na dvě části a přesunutím přestupků do samostatné tabulky. Současně přibyla možnost vypnout všechny zobrazované prvky jedním tlačítkem.

Dále byla provedena úprava, ve způsobu otáčení kamery ve vozidle. Původně se uživatel rozhlížel pouze pomocí myši. To ale některým uživatelům nevyhovovalo, protože pohyb nebyl plynulý a museli se neustále soustředit na kameru. Z toho důvodu přibyla v nastavení možnost vypnout ovládání kamery za pomocí myši. Je-li tato funkce vypnuta, uživatel stiskne tlačítko a kamera se plynule otočí určitým směrem. Uživatel sice nemá takovou volnost v rozhlížení, ale je to pro něj pohodlnější.

Zajímavým nápadem bylo to, že by se uživatel mohl přepnout do autonomního vozidla a místo aby ho ovládal, pouze by sledoval, jak se vozidlo chová a jak řeší aktuální dopravní situaci. Protože by to mohl být další způsob výuky, tato funkcionalita byla následně přidána.



Obrázek 19 - Špatně navržená infrastruktura

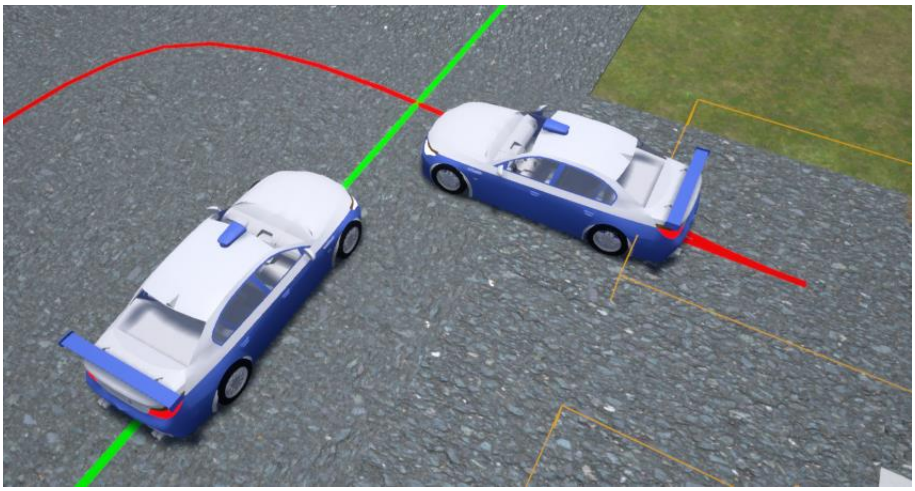
5.2 Známé problémy

Níže jsou popsány známé problémy, na jejich odstranění se při vývoji aplikace pracovalo, ale zatím se je nepodařilo zcela vyřešit. Některé z těchto problémů jsou spíše estetického rázu a na chod aplikace nemají vliv.

Nepřesnost detekce

Někdy se stává, že vozidlo nepřesně detekuje kolizi. Důvodem je, že detekční box je větší než vozidlo, aby byla možná detekce ve větší vzdálenosti než jen v těsné blízkosti. Někdy se ovšem stane (nejčastěji v zatáčce), že vozidla by se ve skutečnosti nesrazila, ale detekční box, který přesahuje, detekuje blížící se vozidlo, a proto zpomalí nebo zastaví. To může mít za následek zpomalení dopravy. Většinou druhé vozidlo projede v pořádku a tím se rozpohybuje i vozidlo pozastavené. Pokud ovšem kolidují vozidla vzájemně, už se z této situace nedostanou.

Při průjezdu křižovatkou kontrolují přednost vozidla, která stojí v detekčních boxech vjezdů. Občas se stane, že vozidlo přijede před křižovatku a vše je v pořádku. Pak toto vozidlo začne vjíždět do křižovatky, ale pokud v okamžiku, kdy ještě není úplně vyjeté z detekčního boxu pro kontrolu, přijede na křižovatku druhé vozidlo, které má přednost, první vozidlo se zastaví. Může se ale stát, že se zastaví v křižovatce a brání ostatním vozidlům v průjezdu (Obrázek 20). To by mohlo být vyřešeno posunutím vjezdů dál od středu křižovatky nebo menším detekčním boxem pro přesnější zachycení vozidel. Problém nepřesných detekčních boxů by mohl být vyřešen například nahrazením boxů za *Tracing*, který by byl přesnější a variabilnější, ale na druhou stranu také výpočetně náročnější.



Obrázek 20 - Blokování průjezdu

Vizualizace

Některé 3D modely jsou zjednodušené a jsou na nich vidět chyby ve struktuře. Jedná se obzvlášť o modely vozidel. Také vozovka je složena pouze z jednoduchého kvádrového *low-poly* 3D modelu a například v zatáčkách může být vidět ostré zalomení místo hladkého oblouku. Tyto modely byly použity pouze za účelem testování.

V některých místech vozovky, obzvlášť tam, kde se překrývají dva 3D modely, může dojít k problikávání textury (Z-fighting). K tomu dochází proto, že oba modely mají stejnou výšku, tím

pádem jsou stejně blízko k uživateli a stroj neví, který model má renderovat. K tomu problému dochází pouze ve chvílích, kdy jsou špatně použity komponenty, které jsou položeny přes sebe. Jedná se pouze o vizuální chybu. Pokud se budou překrývat dvě vozovky, vozidlo nebude mít problém s jejich detekcí.

5.3 Možná rozšíření a využití

Cílem práce nebyla celá aplikace. Proto směrů, kterým by bylo možné se dále ubírat je mnoho. V následujících odstavcích není popsáno pouze rozšíření aplikace, ale i možné využití vytvořených komponent.

Prostředí

Vytvořená aplikace neobsahuje nic z generování okolního prostředí. Na vytváření okolí ať už statického či dynamického existují různé metody a programy, jejichž vytvoření by zabralo spoustu času. Nicméně vytvořené komponenty, byly sestrojeny tak, aby je bylo možné vzít a zapojit je do jakéhokoliv prostředí. Jejich funkcionalita nezávisí na jejich vizualizaci a je možné je využít v jakémkoliv modelu města.

Po celém městě by se mohli procházet chodci, kteří by měli vlastní pravidla chování a předpisy jako vozidla. Také by mohli fungovat na podobném principu (náhodný výběr cest, pevná pravidla, nečekané situace).

Vozidla

Vozidlo uživatele má jednoduché ovládání, které bylo nutné pro testování vzniklých komponent. Obsahuje základní funkcionalitu jako je pohyb, zatáčení a ukazatele směru jízdy. Určitě by bylo vhodné přepracovat vozidlo tak, aby bylo detailnější a více odpovídalo tomu skutečnému. Dalším vylepšení by mohlo být přidání možnosti řazení, zapínání světel, startování motoru atd. Na výběr by mohl být i typ a model vozidla.

U autonomních vozidel by se mohlo zlepšit jejich dynamické chování. Vozidla by měla proměnlivou rychlost, různou „agresivitu“ jízdy a naprogramovaný konkrétní cíl cesty. Místo pevně stanovených pravidel, kterými se momentálně řídí, by se dala do jejich chování zapojit neuronová síť, která by se učila na vznikajících situacích. Stejně tak by se dalo vozidlům přidat chování, které by jim umožňovalo neuposlechnout některé předpisy, čímž by vznikaly nečekané situace, na které by uživatel musel okamžitě zareagovat.

Editor a další části aplikace

Součástí aplikace je i editor, který umožňuje vytvořit si jednoduché situace. Tento editor momentálně nevyužívá všechny možnosti vytvořených komponent. Proto dalším rozšířením by mohlo být zlepšení tohoto editoru. Do editoru by mohla být přidána funkce na automatické generování výukového prostředí. Například uživatel zadá, že chce, aby se zde objevilo tolik a tolik komponent a nechá aplikaci, aby vše dotvořila sama. Kdyby se vytvořené prostředí dalo vyexportovat například do *XML* nebo *JSON* formátu mohlo by se generovat programově a uživatel by nemusel vůbec spouštět editor.

Na přiloženém disku k příručce autoškoly většinou bývají elektronické testy. Součástí aplikace by mohly být právě i tyto elektronické testy. Uživatel by si procházel testy v aplikaci umožňující reálně si situace vyzkoušet. Aplikace by mohla umožňovat například provázání testových otázek s praktickou částí, takže kdyby uživatel chtěl, mohl by se přepnout do vygenerované situace na základě konkrétní otázky.

Funkcionalita

Nejen vozidla, ale i detekce pravidel obsahuje pouze základní funkcionalitu jako ukázku, jak je možné zadaný problém řešit. Celá aplikace by obsahovala mnohem více možností. Jednalo by se například o jízdu mezi pruhy, předjíždění, parkování objíždění překážek a s tím související i další kontroly správného chování dle předpisů. Stejně tak by bylo možné aplikaci rozšířit i o pravidla, která jsou zavedena v jiných státech (rychlost, přednost, jízda vlevo…), což díky univerzálnosti vytvořených komponent by nebyl tak velký problém, protože by se jednalo spíš o modifikaci než o kompletní znovuvytvoření.

V případě obřího města s mnoha vozidly a chodci by aplikace neběžela plynule, protože by veškerý výkon padl na výpočty. Proto by bylo nutné provést na mnoha místech optimalizace. Jednou z optimalizací by musely projít 3D modely, kterým by se zredukoval počet polygonů, a detaily by byly řešeny pomocí textur. Další optimalizace by byla v množství generovaných a zobrazovaných prvků. Prvky a události by se generovaly pouze v blízkosti uživatele a v určité vzdálenosti od něj by bylo vše prázdné. Uživatel by nic nepoznal, protože kolem něj by bylo vždy živo a ušetřilo by se tím mnoho výpočetního výkonu.

Nejlepší výuka je ale vždy s živým učitelem. Proto by mohla přibýt funkce kooperativního režimu, kdy by se mohl k aplikaci připojit další uživatel nebo učitel autoškoly. Ten by pak, stejně jako ve skutečném autě, uživateli dával pokyny, a pokud by uživatel udělal chybu, měl by obratem zpětnou vazbu s detailním vysvětlením, jaké textová tabulka neposkytne.

6 Závěr

Cílem této práce bylo navrhnout, implementovat a otestovat postupy, které by tvořily části aplikace pro podporu výuky pravidel silničního provozu. Požadavkem bylo, aby jednotlivé komponenty byly univerzální s možností obecného využití, snadno rozšiřitelné a aby nebyly přímo závislé na aplikaci a vzhledu. Celá aplikace by potom fungovala jako simulátor skutečné dopravy, na které by se uživatel mohl učit doma bez nutnosti vlastnit drahý trenažer nebo někam docházet.

V úvodu této práce je popsáno, co je to simulace obecně a k čemu se používá, dále jak taková simulace vzniká a do jakých skupin je možné simulace dělit. Je vytvořen přehled vybraných softwarových nástrojů, pomocí kterých je možné vytvářet simulace různých typů. Kapitolu uzavírá výběr existujících aplikací, které se zabývají výukou silničního provozu. Jsou uvedeny některé nedostatky, které jsou ve vlastním navrženém řešení odstraněny.

Následuje část práce, která popisuje výukovou aplikaci jako celek, a navržené komponenty, které by byly součástí této aplikace. Dále se zaměřuje na obecné postupy při návrhu řešení a způsoby, kterými by bylo možné docílit požadovaných výsledků. Mezi navržené komponenty patří dopravní značení, semafory, křižovatka, a také vozidlo, které může být ovládané počítačem nebo uživatelem. Je zde popsáno, na jaké problémy se bude nutné při řešení zaměřit. Kapitola o ovládání popisuje, v čem by mohl být problém v takové aplikaci z hlediska hardwaru, jaké jsou možnosti a jak by bylo možné těmto problémům předejít.

V kapitole konkrétního řešení je popsáno, jak byly části z návrhu převedeny do praxe, jaké problémy při tvorbě vznikly a v čem se konkrétní řešení liší od původního návrhu. Byla řešena detekce správného pohybu vozidla, jak z pohledu jeho umístění na vozovce ve správném pruhu, tak z pohledu dodržování pravidel, například maximální dovolené rychlosti. Je zde uveden způsob kontroly pravidel silničního provozu ve vztahu k ostatním účastníkům. Příkladem změny oproti návrhu je dopravního značení, které bylo rozděleno do skupin dle aktuálního znění zákona, ale pro naprogramování komponent je toto rozdělení zcela nevyhovující. Významnou částí je ukázka algoritmu možného průjezdu křižovatkou, který byl navržen tak, že funguje univerzálně, nezávisí na konkrétním tvaru a typu křižovatky, a proto je možné dopravní situace měnit. Je možné nejen měnit tvar křižovatky a počet výjezdů, ale také lze přidat semafory nebo klasickou křižovatku změnit na kruhový objezd. Je zde uveden i způsob řešení detekce prvků, který se nespoléhá na snímání okolí kamerou, jako je tomu u reálných autonomních vozidel, ale detekce probíhá pomocí speciálních kolizních komponent, díky kterým je získávání informací z okolí mnohonásobně přesnější.

Jedna z podkapitol popisuje funkcionalitu aplikace, která nebyla původně plánovaná, a tím je editor situací. Ten je možné využít pro vytváření vlastních map a dopravních situací, které bývají součástí učebnic autoškoly, a díky tomu je možné ještě víc zlepšit výuku. Editor je výsledkem toho,

co bylo od počátku cílem, tedy vytvořit univerzální komponenty pro kontrolu pravidel a detekci dopravních situací. Tento editor mohl vzniknout právě proto, že všechny komponenty byly vytvářeny obecně, a úprava jejich chování závisí pouze na vstupních parametrech. V editoru je poté možné tyto komponenty používat a ukládat do jednoduchých struktur.

Poslední část práce se věnuje testování vzniklých komponent. Je zde popsáno, co bylo nutné změnit, ačkoliv to na první pohled vypadalo jako správné řešení. Příkladem mohou být prvky, které snižovaly soustředění uživatele. Uživatelské rozhraní bylo zjednodušeno a rozděleno na dva samostatné celky. Ze stejného důvodu i ovládání kamery uvnitř vozidla prošlo patřičnými úpravami a zjednodušením. Jedna z podkapitol se věnuje možným způsobům využití a rozšíření navržených komponent a výsledné aplikace. Současně se zaměřuje na známé potíže, na jejichž eliminace se pracovalo, ale přesto se je nepodařilo odstranit.

Celá aplikace pro výuku začínajících řidičů, by se skládala z mnoha samostatných celků, které by se staraly o prostředí, dynamičnost, reakci na chování uživatele a o mnoho dalších. Některými z těchto celků by mohly být právě komponenty navržené a vytvořené v této práci, obstarávající správné vyhodnocení generovaných situací v silničním provozu.

Zdroje

Literatura

[1] RÁBOVÁ, Zdena. *Modelování a simulace*. 3., přeprac. vyd. Brno: VUT, 1992 [cit. 2020-02-06]. ISBN 80-214-0480-9.

Elektronické zdroje

[2] BACKLUND, Per, Henrik ENGSTRÖM, Mikael JOHANNESSON a Mikael LEBRAM. Games for traffic education: An experimental study of a game-based driving simulator. Simulation & Gaming [online]. 2008, 41(2), 145-169 [cit. 2020-04-28]. DOI: 10.1177/1046878107311455. ISSN 1046-8781. Dostupné z: http://journals.sagepub.com/doi/10.1177/1046878107311455

[3] BANKS, Jerry. *Handbook of simulation: principles, methodology, advances, applications, and practice*. Norcross, Ga.: Co-published by Engineering & Management Press, c1998 [cit. 2020-02-02]. ISBN 978-0471134039.

[4] BONABEAU, E. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences* [online]. 2002, **99**(Supplement 3), 7280-7287 [cit. 2020-01-25]. DOI: 10.1073/pnas.082080899. ISSN 0027-8424. Dostupné z: http://www.pnas.org/cgi/doi/10.1073/pnas.082080899

[5] BRENNER, Claus; HAALA, Norbert. Fast production of virtual reality city models. *International Archives of Photogrammetry and Remote Sensing*, 1998, 32.part 4: 77-84 [cit. 2020-02-20].

[6] BRILON, Werner, Felix HUBER, Michael SCHRECKENBERG a Henning WALLENTOWITZ, ed. *Traffic and Mobility* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999 [cit. 2020-02-23]. DOI: 10.1007/978-3-642-60236-8. ISBN 978-3-642-64316-3.

[7] ÇAKIROĞLU, Ünal a Seyfullah GÖKOĞLU. A Design Model for Using Virtual Reality in Behavioral Skills Training. *Journal of Educational Computing Research* [online]. 2019, **57**(7), 1723-1744 [cit. 2020-01-18]. DOI: 10.1177/0735633119854030. ISSN 0735-6331. Dostupné z: http://journals.sagepub.com/doi/10.1177/0735633119854030

[8] Computer simulation. *http://www.sciencedaily.com* [online]. [cit. 2020-02-06]. Dostupné z: http://www.sciencedaily.com/articles/c/computer_simulation.htm.

[9] DEERY, Hamish A. a Brian N. FILDES. Young Novice Driver Subtypes: Relationship to High-Risk Behavior, Traffic Accident Record, and Simulator Driving Performance. *Human Factors: The Journal of the Human Factors and Ergonomics Society* [online]. 2016, **41**(4), 628-643 [cit. 2020-01-04]. DOI: 10.1518/001872099779656671. ISSN 0018-7208. Dostupné z: http://journals.sagepub.com/doi/10.1518/001872099779656671

[10] DRCHAL, Jan, Jan KOUTNIK a Miroslav SNOREK. HyperNEAT controlled robots learn how to drive on roads in simulated environment. In: 2009 IEEE Congress on Evolutionary Computation [online]. IEEE, 2009, 2009, s. 1087-1092 [cit. 2020-03-09]. DOI: 10.1109/CEC.2009.4983067. ISBN 978-1-4244-2958-5. Dostupné z: http://ieeexplore.ieee.org/document/4983067/

[11] INFORMACE o nehodovosti na pozemních komunikacích v České republice v roce 2019. In: *Policie České republiky* [online]. 8.1.2020 [cit. 2020-01-12]. Dostupné z: https://www.policie.cz/clanek/statistika-nehodovosti-900835.aspx

[12] INGRAM, Rob, Steve BENFORD a John BOWERS. Building virtual cities. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology - VRST '96 [online]. New York, New York, USA: ACM Press, 1996, 1996, s. 83-91 [cit. 2020-03-12]. DOI: 10.1145/3304181.3304199. ISBN 0897918258. Dostupné z: http://dl.acm.org/citation.cfm?doid=3304181.3304199

[13] MAERIVOET, Sven a Bart DE MOOR. Cellular automata models of road traffic. *Physics Reports* [online]. 2005, **419**(1), 1-64 [cit. 2020-02-17]. DOI: 10.1016/j.physrep.2005.08.005. ISSN 03701573. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0370157305003315

[14] MAURER, M., R. BEHRINGER, S. FURST, F. THOMANEK a E.D. DICKMANNS. A compact vision system for road vehicle guidance. In: *Proceedings of 13th International Conference on Pattern Recognition* [online]. IEEE, 1996, 1996, 313-317 vol.3 [cit. 2020-01-18]. DOI: 10.1109/ICPR.1996.546962. ISBN 0-8186-7282-X. Dostupné z: http://ieeexplore.ieee.org/document/546962/

[15] Mladí řidiči. In: *Besip* [online]. 26.8.2019 [cit. 2020-01-20]. Dostupné z: https://www.ibesip.cz/Statistiky/Statistiky-nehodovosti-v-Ceske-republice/Dopravni-nehodovost-v-roce-2019/Mladi-ridici

[16] PARUCHURI, Praveen, Alok Reddy PULLALAREVU a Kamalakar KARLAPALEM. Multi agent simulation of unorganized traffic. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 1 - AAMAS '02* [online]. New York, New York, USA: ACM Press, 2002, 2002, s. 176- [cit. 2020-01-12]. DOI: 10.1145/544741.544786. ISBN 1581134800. Dostupné z: http://portal.acm.org/citation.cfm?doid=544741.544786

[17] RAIVIO, Tuomas, et al. A simulation model for military aircraft maintenance and availability. In: *Proceedings of the European simulation multiconference*, 2001. [cit. 2020-03-05].

[18] ROSEN, Robert. On complex systems. European Journal of Operational Research [online]. 1987, 30(2), 129-134 [cit. 2020-03-12]. DOI: 10.1016/0377-2217(87)90089-0. ISSN 03772217. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/0377221787900890

[19] SUN, Jing, Xiaobo YU, George BACIU a Mark GREEN. Template-based generation of road networks for virtual city modeling. In: *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '02* [online]. New York, New York, USA: ACM Press, 2002, 2002, s. 33- [cit. 2020-01-29]. DOI: 10.1145/585740.585747. ISBN 1581135300. Dostupné z: http://portal.acm.org/citation.cfm?doid=585740.585747

[20] Vyhláška č. 294/2015 Sb., Vyhláška, kterou se provádějí pravidla provozu na pozemních komunikacích [cit. 2020-01-07].

[21] WANG, Heng, Bin WANG, Bingbing LIU, Xiaoli MENG a Guanghong YANG. Pedestrian recognition and tracking using 3D LiDAR for autonomous vehicle. *Robotics and Autonomous Systems* [online]. 2017, **88**, 71-78 [cit. 2020-04-13]. DOI: 10.1016/j.robot.2016.11.014. ISSN 09218890. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0921889015302633

[22] WHITE, S. Hoya; DEL REY, A. Martin; SANCHEZ, G. Rodriguez. Using cellular automata to simulate epidemic diseases. Applied Mathematical Sciences, 2009, 3.20: 959-968 [cit. 2020-03-10].

[23] Zákon č. 361/2000 Sb., Zákon o provozu na pozemních komunikacích a o změnách některých zákonů [cit. 2020-01-07].

[24] ZHOU, Suiping, Dan CHEN, Wentong CAI, et al. Crowd modeling and simulation technologies. *ACM Transactions on Modeling and Computer Simulation* [online]. 2010, **20**(4), 1-35 [cit. 2020-01-18]. DOI: 10.1145/1842722.1842725. ISSN 10493301. Dostupné z: http://portal.acm.org/citation.cfm?doid=1842722.1842725

[25] ZIV, Amitai, Paul Root WOLPE, Stephen D. SMALL a Shimon GLICK. Simulation-Based Medical Education: An Ethical Imperative. *Simulation In Healthcare: The Journal of the Society for Simulation in Healthcare* [online]. 2006, **1**(4), 252-256 [cit. 2020-03-09]. DOI: 10.1097/01.SIH.0000242724.08501.63. ISSN 1559-2332. Dostupné z: http://journals.lww.com/01266021-200600140-00010

Software

[26] Car Driving School Simulator. [online]. Copyright © [cit. 2020-01-28]. Dostupné z: http://play.google.com/store/apps/details?id=com.boombitgames.DrivingSchoolParking

[27] City Car Driving. [online]. Copyright © [cit. 2020-01-29]. Dostupné z: https://store.steampowered.com/app/493490/City_Car_Driving/

[28] Modelica Association — Modelica Association. *The Modelica Association — Modelica Association* [online]. Copyright © [cit. 2020-01-25]. Dostupné z: https://www.modelica.org/

[29] NetLogo Home Page. *The CCL* [online]. Copyright © 1999 [cit. 2020-01-25]. Dostupné z: https://ccl.northwestern.edu/netlogo/

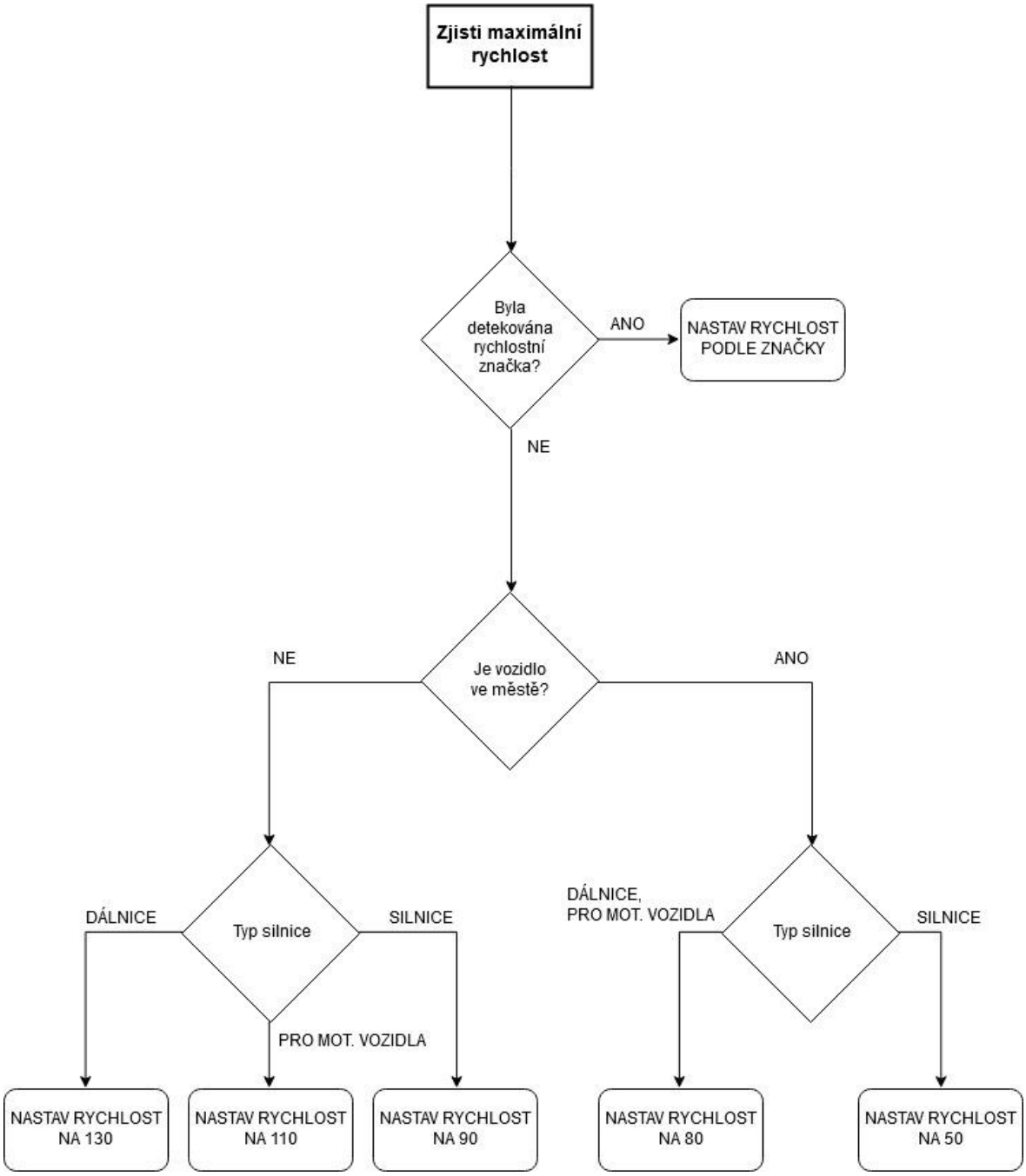
[30] Unity Real-Time Development Platform | 3D, 2D VR & AR Visualizations. *Unity Real-Time Development Platform | 3D, 2D VR & AR Visualizations* [online]. Copyright © 2020 Unity Technologies [cit. 2020-01-24]. Dostupné z: https://unity.com/

[31] Unreal Engine 4. [online]. Copyright © 2004 [cit. 2020-01-24]. Dostupné z: https://www.unrealengine.com/en-US/

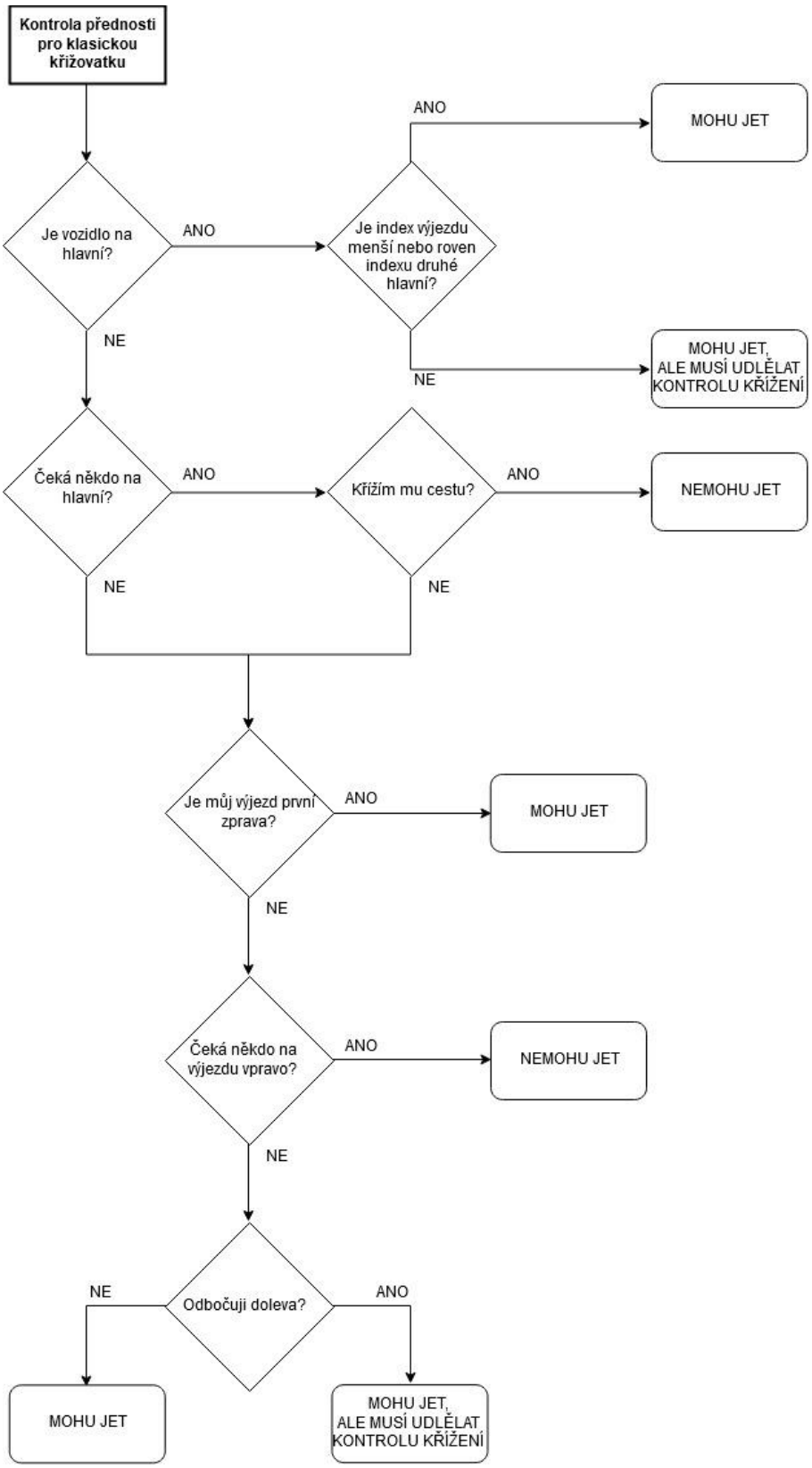
Příloha č. 1 – Obsah přiloženého DVD

- *diplomova_prace.pdf* – Tato diplomová práce ve formátu PDF
- *diplomova_prace.docx* – Zdrojové texty této diplomové práce
- *zdrojove_soubory.zip* – Archiv se zdrojovými soubory demo aplikace
- *vysledna_aplikace.zip* – Archiv výsledné demo aplikace

Příloha č. 2 – Zjištění maximální rychlosti



Příloha č. 3 – Průjezd křižovatkou



Příloha č. 4 – Návod k použití aplikace

V této příloze je popsáno ovládání aplikace a její funkce.

Popis aplikace

Po spuštění aplikace je možné si vybrat z nabídky:

- *Předtvořená mapa* – Obsahuje několik dopravních situací jako jsou křižovatky, kruhové objezdy a semafory. Dále je zde připraveno několik účastníků silničního provozu.
- *Mapa vytvořená v editoru* – Jedná se o mapu, kterou si uživatel vytvořil již dříve za pomocí editoru. Může se jednat o souvislou trasu nebo například o typovou křižovatku z učebnice autoškoly.
- *Editor situací* – Součástí aplikace je i editor, který obsahuje ukázku některých vytvořených komponent. Uživatel si na pevně stanovenou mřížku může naskládat komponenty dle svého uvážení a scénu si následně vyzkoušet. Vše se okamžitě ukládá, takže se nestane, že by uživatel o svoji práci přišel.
- *Nastavení* – V nastavení je možné zapnout ovládání kamery myší, zobrazování kolizních boxů, generování dodatkové tabule pro křižovatku, zobrazování trasy autonomních vozidel a indexy křižovatek.

Aplikace kontroluje, jestli uživatel jezdí podle předpisů. Pokud ne, je o tom informován a do tabulky přestupků je o tom vytvořen záznam.
Při detekci značky je tato značka ukázána uživateli na obrazovce.
Rychlost vozidla je neustále sledována a při překročení je o tom vytvořen záznam.
Přednastavenou tabulku s informacemi o přestupcích je možné nahradit vlastní pomocí souboru ve formátu *csv* ve složce s aplikací.

Ovládání aplikace

Ovládání v editoru:

- Pravé tlačítko myši – pohyb kamery
- Kolečko – zoom
- Levé tlačítko myši – vytvoření komponenty
- R – reset
- Esc – hlavní menu

Je vhodné vložit jedno vozidlo uživatele a všechna autonomní vozidla je nutné položit na silnici.

Ovládání vozidla:

- W, S, A, D – pohyb
- Mezerník – ruční brzda
- Q, E – blinkry
- Tab – přepínání mezi vnitřní a vnější kamerou
- Enter – přepínání mezi vozidlem uživatele a autonomním vozidlem
- Myš/šipky – otáčení kamerou uvnitř vozidla
- P – přestupky
- Esc – hlavní menu

Zadání diplomové práce

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Milan Kořínek**
Studium:	I1800322
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Generování virtuální scény pro podporu výuky pravidel silničního provozu**
Název diplomové práce AJ:	Generating a virtual scene to support traffic rules learning

Cíl, metody, literatura, předpoklady:

Prozkoumat principy a metody Vytvořit přehled Navrhnout implementaci Implementovat Navrhnout a provést testování a srovnání Zhodnotit dosažené výsledky

https://scholar.google.cz

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Bruno Ježek, Ph.D.
Oponent:	prof. RNDr. PhDr. Antonín Slabý, CSc.

Datum zadání závěrečné práce: 14.1.2018