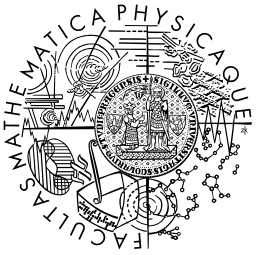


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Eliáš Cizl

Efektivita centralizovaného plánování křižovatek

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jiří Švancara, Ph.D.

Studijní program: Informatika

Studijní obor: Programování a softwarové systémy

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Rád bych poděkoval své rodině a svému blízkému okolí, kteří mě na této cestě a obzvlášť v jejím závěru podporovali. Nemalý dík také zasluhuje můj vedoucí RNDr. Jiří Švancara, Ph.D., především za jeho neumírající trpělivost.

Název práce: Efektivita centralizovaného plánování křižovatek

Autor: Eliáš Cizl

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jiří Švancara, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Navigace vozidel po silnicích je komplexní problém, který se pravděpodobně dočká řešení zapojením umělé inteligence pro klíčové role. Dnes existují auta schopná samostatné jízdy, která jsou ale závislá na staré infrastruktuře zahrnující především křižovatky určené pro lidské řidiče. Tento text otvírá novou kapitolu v problematice autonomního řízení křižovatek (AIM). Většina dosavadních výzkumů se zabývala pouze tím, jak co nejlépe naimplementovat řešení pro jednu křižovatku. My jsme vytvořili simulaci běžící v reálném čase, kde se objevuje až několik desítek křižovatek vedle sebe. V práci provádíme experimenty, kde testujeme různá zapojení autonomních algoritmů spolu se světelnými křižovatkami. Autonomní křižovatky svou efektivitou jasně vítězí a ve větších městech se nejvíce vyplácí je nasazovat na nejvytíženějších uzlech.

Klíčová slova: Inteligentní křižovatka Plánování cest Simulace Unity Autonomní vozidla

Title: The Effectiveness of Centralized Planning for Intersections

Author: Eliáš Cizl

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Jiří Švancara, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Vehicle navigation on roads is a complex problem that will probably be solved by using artificial intelligence in key roles. Today, there are cars capable of autonomous driving, but they are dependent on an old infrastructure that primarily includes intersections designed for human drivers. This thesis opens a new chapter in the area of autonomous intersection management (AIM). Most research to date has only looked at how best to implement a solution for a single intersection. We have created a simulation that runs in real time, where up to several dozen intersections appear side by side. In this work, we conduct experiments where we test different layouts of the autonomous algorithms along with traffic lights. Autonomous intersections clearly win with their efficiency, and in larger cities it's most advantageous to deploy them at the busiest intersections.

Keywords: Intelligent intersection Route planning Simulation Unity Autonomous vehicles

Obsah

Úvod

Většina z nás každý den řeší problém, jak se dopravit z místa A na místo B. Pokud žijeme ve městě, máme rychlou možnost v podobě hromadné dopravy, ale ta ne vždycky dokáže pokrýt všechny naše nároky. Mnoho měst v USA a Kanadě (a jistě i dalších zemích světa, ale v těchto je to obzvlášť znatelné) disponuje velmi chudou veřejnou dopravou, a tedy jsou tam lidé odkázáni na přepravu osobními automobily. Navíc existují zaměstnání, v rámci kterých je potřeba přepravovat věci mezi různými místy a pomocí hromadné dopravy je momentálně uskutečňovat nejde.

Ať už je naše potřeba a volba jakákoli, realitou zůstává, že silniční komunikace jsou dnes hojně používaným způsobem dopravy. Dává smysl zabývat se jejich problematikou, a to dokonce z mnoha hledisek: bezpečnost, efektivita, dopad na životní prostředí, ... Bohužel v mnoha ohledech nejsou automobilní sítě ani nejbezpečnější (Liu a Moini, 2015), ani nejrychlejší, ani nejšetrnější k planetě. O to více bychom se měli snažit tyto jejich aspekty zlepšovat.

Jedním z řešení, které se před lety objevilo, je možnost nechat vozidla řídit umělou inteligenci. Dnešní pokrok v technologiích umožňuje komerční automobily osadit mnoha senzory, díky kterým mohou monitorovat své okolí a v zásadě si neustále udržovat velice přesný model reality. Když se do nich vloží software, který umožní tyto dispozice využít, jsou auta schopná naprosto samostatné navigace. Autonomní vozidla pak zvládají mnohem rychleji reagovat na změny prostředí, neřídí pod vlivem alkoholu, netelefonují za jízdy, ... Zkrátka bezpečná a spolehlivá cesta jsou vždy jejich první prioritou.

Kvůli malému zastoupení jsou dnešní chytrá vozidla ještě dost zaměřená na sebe – pomáhají jen svým řidičům s jízdou a řeší své vlastní okolí a trasu. Jenže díky výbavě a funkcionalitě, kterou disponují, už zbývá pouze malý krůček k tomu nechat autonomní auta spolupracovat mezi sebou. Vzájemná koordinace za použití bezdrátových technologií má potenciál umožnit ještě větší bezpečnost a efektivitu, než je tomu nyní.

Silniční komunikace bychom mohli rozdělit do více kategorií – dle hustoty, dle běžných rychlostí, dle potenciálních kolizí apod. V existujících výzkumech (Huang a kol., 2008) se ukazuje, že ke kolizím dochází nejčastěji na křižovatkách (a obecně na místech s hustší dopravou jako jsou města). Proto se touto tématikou bude zabývat i naše práce. Díky dostupným technologiím, které zajišťují autonomitu vozidel a vzájemnou komunikaci můžeme začít přicházet s novými a efektivnějšími řešeními, než jsou dnešní světelné křižovatky (které v práci budeme častěji označovat kratším pojmem *semafory*), případně prosté značky „Stůj, dej přednost v jízdě!“ (které budeme nazývat *stopky*).

Nejsme první, kdo jeví zájem o problematiku křižovatek s autonomními vozidly. Ta byla již dříve (Dresner a Stone, 2008) zasazena do oboru multiagentního plánování, kde agenti jsou jednotlivá vozidla. Vzniklo už mnoho výzkumů, které navrhují nové systémy pro navigaci vozidel skrze křižovatky. Tyto práce mívají různé cíle (bezpečnost, efektivita i šetrnost k prostředí) a vývoj si různými způsoby ulehčují (zjednodušený model vozidel, zákaz zatáček, neomezená doba pro výpočty, ...).

My bychom do této oblasti rádi vnesli trošku jiný pohled. Představme si situ-

aci, kdy už jsou všechna vozidla autonomní a schopná mezi sebou komunikovat a přizpůsobit se novým mechanismům. Akorát jsme se „zasekli“ na staré architektuře křižovatek, tudíž jsou všude pouze semafory či stopky s omezenou propustností. My bychom nyní jakožto designéři města měli navrhnout nový systém pro jednotlivé křižovatky. Tento proces bude v realitě asi probíhat ještě za přítomnosti lidských řidičů, kde hybridní křižovatky budou poskytovat víc systémů naráz pro různé typy řidičů. My si situaci zjednodušíme a budeme počítat s tím, že vozidla jsou řízena chytrou umělou inteligencí a zpomaluje stará infrastruktura silničních komunikací.

V tomto díle chceme prozkoumat interakce mezi řidiči a křižovatkami, kdy zde není žádný všeobjímající systém a jednotlivé prvky jsou na sobě co nejvíce nezávislé. Předpokládáme, že nové křižovatky jsou nějaké samostatné (centralizované) systémy a když k nim přijede vozidlo, doručí mu nějakým způsobem (pravděpodobně bezdrátovou komunikací) informace, jak má pokračovat na své cestě – jestli a kdy zpomalovat/zrychlovat (to samé vlastně dělají ve velice omezené formě i existující řešení).

Klademe si předpoklad, že nasazení takovéto chytré křižovatky by nebylo úplně levné a vyžadovalo by řadu zásahů do infrastruktury, proto nás bude především zajímat, *jak efektivní tato změna je* a *na kterých křižovatkách se nejvíce vyplatí*. Chceme provádět experimenty, kdy zapojíme více křižovatek vedle sebe a porovnáme různé konfigurace jejich *algoritmů* – tak budeme říkat systému, který křižovatku ovládá (stopka/semafor/autonomní křižovatka). Výsledkem by mělo být zjištění, jaké zapojení je nejvýhodnější.

Cílem této práce není přijít s ultimátním algoritmem, který bude ještě lepší než ty, které už byly dříve vymyšleny. My se chceme na problém podívat z jiného úhlu pohledu a porovnat chování několika autonomních křižovatek ve větším systému. Opakujeme, že ale ani nechceme přicházet se speciálním navigačním systémem pro celé město. Křižovatky a vozidla jsou na sobě nezávislá (samozřejmě kromě situací, kdy přicházejí do blízkého kontaktu).

Nový chytrý algoritmus křižovatky, který jsme pro toto dílo připravili, byl při vývoji ovlivněn několika cíly specifickými pro tuto práci. Věděli jsme, že nás čeká řada testování s městy, které mohou mít až desítky najednou běžících křižovatek. Potřebovali jsme systém, který běží alespoň dostatečně efektivně, aby šlo experimenty vykonávat v omezeném čase na běžných domácích zařízeních. Plná věrnost realitě pro nás nebyla tak důležitá, protože testy probíhaly se stejnými parametry pro všechny typy křižovatek a našim cílem je hlavně porovnat algoritmy mezi sebou.

Samotný text je členěn do několika hlavních oddílů. V první kapitole (1) se dopodrobna podíváme na problematiku autonomních křižovatek a rozebereme si existující výzkum. V druhé kapitole (2) si detailně popíšeme řešení a systém, s kterým jsme přišli my v rámci této práce. V třetí kapitole (3) využijeme vyhotovenou simulaci k provedení experimentů s různými algoritmy křižovatek a pokusíme se objevit nejvýhodnější konfigurace.

Jelikož většina z existujících prací byla napsaná v anglickém jazyce a obecně je to v našem oboru velice hojně používaný jazyk, budeme některé pojmy nechávat anglicky. Stejně tak veškerý kód a software dodávané simulace byly vytvořeny v angličtině.

1. Analýza souvisejících prací

V této kapitole se podíváme na existující vědecké práce, které se zabývají problematikou autonomních křižovatek. Projdeme si témata, na která je potřeba se zaměřit při zkoumání tématu a při tvorbě vlastního systému.

1.1 Autonomní křižovatky

Jedna z prvních a jistě nejrozšířenější práce na toto téma je článek Dresnera a Stona (Dresner a Stone, 2008). Práce zavádí pojmy jako *ITS* (*Intelligent Transportation Systems* – inteligentní přepravní systémy) či *AIM* (*Autonomous Intersection Management* – autonomní správa křižovatky). Zároveň už text pokládá nějaké základní cíle, kterých by bylo dobré dosahovat:

- **Autonomita** – každé vozidlo by mělo být nezávislým agentem. Kontrolovat vše centrálně by bylo příliš složité a náchylné na chyby.
- **Nízká komunikační komplexita** – ideálem je držet počet zpráv mezi prvky na minimu.
- **Realistický model senzorů** – v simulacích bychom neměli využívat neexistující senzory, které nejsou v realitě aplikovatelné.
- **Standardizace protokolu** – komunikace by měla zahrnovat nějaký standardizovaný protokol, na který se dá spolehnout.
- **Vyhnutí se zaseknutí** – systém by neměl dojít do mrtvého bodu, nebo nechávat nějaká vozidla nekonečně čekat.
- **Inkrementální nasaditelnost** – autonomní křižovatky by měly být nasaditelné postupně do systému a také na jednotlivých křižovatkách by měla být podpora ze začátku i pro lidské řidiče.
- **Bezpečnost** – nikdy by uvnitř křižovatky nemělo docházet ke kolizím vozidel kromě naprosto nevyhnutelných případů (například úmyslné nabourání).
- **Efektivita** – auta by měla cestovat skrze křižovatku co nejrychleji s co nejmenším zpožděním.

Od té doby vzniklo mnoho dalších textů, ve kterých jsou nějaké principy dodrženy a v jiných jsou prozkoumávány další možnosti. Naše práce se také zaměřuje na některé a jiné může účelně porušit v cestě za nějakým dalším cílem.

Různých článků vzniklo tolik, že se objevily i práce, které se snaží zpřehlednit to, co bylo dříve vydané, jako například Khayatian a kol. (2020) nebo Zhong a kol. (2020) a další. S pomocí těchto textů si můžeme udělat přehled o tom, jaká různá řešení se v problematice vyskytují a co už bylo dříve vyzkoušeno.

Dá se říct, že většina řešení funguje na nějakém principu plánu či rezervací. Vozidla mají projet skrz křižovatku a je potřeba určit, kdo kdy bude v jakém bodě. Proto se většinou vytvoří seznam (či jiná reprezentační struktura) možných

kolizních míst a následně rozvrh, který zajišťuje, že vozidla správně projedou. Otevírá se tu však hodně možností, jak jednotlivé části řešit, takže se na ně pojďme detailněji podívat.

1.1.1 Distribuované přístupy

Základní otázkou je, jak vozidla v křižovatce budou komunikovat, přesněji kdo s kým. Jedním z možných přístupů, je takzvaný distribuovaný. To znamená, že jednotliví agenti spolu komunikují pouze navzájem a nepotřebují vnější entitu. Výhodou je, že se zde nemusí zavádět žádná nová infrastruktura (pouze nová vozidla, ale to je něco, co se v průběhu času stejně děje). Tyto systémy jsou použitelné hlavně v méně vytížených místech. Kdybychom se je pokusili nasadit na velkou frekventovanou křižovatku, prudce by stoupl počet zpráv v systému a organizace by se velice zkomplikovala.

Často se uplatňuje přístup, že vozidla si vzájemně posílají informace a následně jedno zaujme pozici „vůdce“, která se pak předává dál (například Li a Wang (2006)). Tato řešení často pro zjednodušení uplatňují princip seskupování, kdy si sousední vozidla vyměňují informace mezi sebou a jedno z nich pak reprezentuje tuto skupinu. Hassan a Rakha (2014) to aplikují tak, že skupina vzniká pro každý pruh. Přijíždějící vozidla se přidávají do skupiny svého pruhu a skupiny mezi sebou komunikují skrz svá první vozidla.

Dále například Au a kol. (2015) rozděluje oblast křižovatky do různých zón – pozorující zóna, zóna, kde probíhá plánování a optimalizace, kontrolní zóna, kde si vozidlo hlídá dodržení dříve naplánované trasy a udržuje bezpečné vzdálenosti, a nakonec slučující zóna, kde vozidla projedou křižovatkou. Tento přístup počítá s reálnými limity, které se týkají komunikace na dlouhou vzdálenost.

Velkou výhodou tohoto přístupu je snadná škálovatelnost, absence jediného bodu selhání a za předpokladu přítomnosti autonomních vozidel (což je zatím silný předpoklad) možnost nasazení takřka kdekoli.

1.1.2 Centralizované přístupy

Dalším řešením, které se možná jeví jako přímočařejší a snazší pro designéry komunikací, je umístit do prostoru křižovatky centrální entitu, se kterou musí všechna přijíždějící vozidla komunikovat. Sice to znamená, že systém má jeden bod, na kterém může selhat, ale na druhou stranu je to i hlavní bod, o který je potřeba se starat a tím se vlastně snižují nároky na údržbu (a požadavky na jednotlivá vozidla). Navíc, pokud je systém dobře zabezpečen, nemůže se například stát to, že by se někdo tvářil jako další z vozidel, které komunikuje, a začal do systému zasévat chaos.

Tyto přístupy se dají rozdělit na dvě podkategorie: *query-based*, kdy sama vozidla navrhují rezervaci, a *assignment-based*, kdy vozidla pouze oznámí své údaje a křižovatka sama jim hledá rezervaci.

Query-based

Query-based, což bychom mohli přeložit jako „na základě dotazů“. Na tomto principu je založena i práce Dresner a Stone (2008). Vozidlo při přiblížení se ke křižovatce uvede svůj odhad času a rychlosti příjezdu a křižovatka mu buď potvrdí

možnou rezervaci, nebo zamítne a pak musí vozidlo zpomalit, čekat a zkoušet to znovu. Jin a kol. (2013) používá podobný přístup, ale vozidla jsou zde seskupována (komunikují spolu navzájem) a požadavek křižovatce posílá pouze vůdce skupiny.

Assignment-based

Assignment-based přístupy neboli „na základě přiřazení“ přesouvají více zodpovědnosti na křižovatku samotnou. Vozidlo při registraci pouze poskytne své současné hodnoty a čeká, že křižovatka mu naplánuje rychlost a čas, kterých má dosáhnout v době průjezdu. Za cenu vyšší výpočetní složitosti tato řešení mohou dosahovat větší propustnosti než *query-based* alternativy.

Například v práci Khayatian a kol. (2019) vozidla posílají své hodnoty a cílovou destinaci. Může zde však nastat zpoždění při odesílání a přijímání zpráv a na to se uvedený text soustředí. Navrhují systém synchronizace časů, aby tím plánování získalo na deterministočnosti.

1.1.3 Detekce kolizí

Dalším důležitým prvkem je způsob, jak si systém křižovatku virtuálně rozčlení, aby předcházel kolizím. Při plánování v počítači nemůžeme přenést veškerou spojitou informaci z reálného světa, a proto se musíme uchýlit k nějakým zjednodušením. Je potřeba zavést reprezentaci, pomocí které se zabrání potenciálním kolizím, ale zároveň nebude způsobovat velké snížení celkové účinnosti.

Nejčastější přístupy uvádíme v dalších podsekcích. Občas se ale nějaká studie vymkne zaběhlým způsobům – například Li a kol. (2018) ve své práci zkoumají možnosti libovolných (často nestandardních) cest skrz křižovatku, kde trajektorie vozidla může vést kdekoli skrz křižovatku za účelem vyhnout se kolizím s dalšími vozidly.

Mapa obsazenosti

Mapa obsazenosti znamená, že prostor křižovatky se virtuálně rozdělí do (typicky čtvercových) políček a následně plánování probíhá pro jednotlivá políčka. Projíždějící vozidlo vždy zabírá určitý počet polí. V rozvrhu nikdy nesmí být pole obsazené více různými vozidly v ten samý okamžik. Tento přístup využili už Dresner a Stone v původním díle (Dresner a Stone, 2008). Mohli bychom to připodobnit dnešním rastrovým obrazovkám, které také spojité (vektorové) obrázky musí zobrazit v čtvercové mřížce.

Nevýhodou tohoto přístupu je, že autor musí nějak určit granularitu mřížky – tedy počty políček na šířku a délku. Příliš velká granularita znamená vysokou přesnost, ale i vysokou komplexitu (s tím, že používanost jednotlivých polí se pak dost liší). Naopak příliš málo polí sice vede k rychlým výpočtům, ale propustnost může klesat – toto už objevili autoři původní práce (Dresner a Stone, 2008) společně s tím, že sudý počet může být efektivnější než vyšší lichý počet.

Škopková a kol. (2020) dokonce dělí křižovatku na mřížku a následně určí, že přesně na jednom políčku může být přesně jeden agent (a agent nezabírá více polí). Práce dále provádí experimenty v tomto zjednodušeném módu a dokonce zahrnuje statistiku zamítnutých požadavků na rezervaci, kdy vozidlo ve výsledku vůbec neprojelo.

Většina novějších výzkumů se od tohoto způsobu detekce kolizí dle nových studií (Gholamhosseinian a Seitz, 2022) spíše odklání v preferenci následujícího řešení.

Trajektorie cest

Jedná se o přístup, který si uvědomuje, že ne všechna místa na křižovatce jsou stejně důležitá. Typicky se nejdříve spočítají trajektorie všech průjezdů skrze křižovatku a následně se naleznou křížení těchto trajektorií. Přístup má i svoje nevýhody – například, že vozidla zabírají větší prostor než jen křivku trajektorie. Tomuto přístupu se také říká *Conflict point based* – pracuje na základě konfliktních bodů (Levin a Rey, 2017).

Dále podobný způsob používá např. Lu a Kim (2016), kde vyvinuli speciální koncept *DTOT* (*Discrete-Time Occupancies Trajectory*), tedy sekvence přítomností vozidel seřazená dle času. Při plánování tedy stačí porovnávat, že dva záznamy v sekvenci se v ten samý čas nepřekrývají.

1.1.4 Scheduling policy

Při přijíždění a plánování dalších a dalších vozidel se musíme zamyslet, jak má systém pracovat, když by teoreticky kolize mohla nastat. Kvůli tomu je potřeba rezervace nějakého vozidla posunout a obecně průjezdy začít nějak řadit. O tuto část se stará takzvaná *scheduling policy*, což by se dalo přeložit jako „plánovací přístup“ – necháváme zde anglický název, protože máme pocit, že v češtině neexistuje dobrý ekvivalent slova „policy“.

V dalších podsekcích uvidíme nejčastější přístupy. Objevují se i nestandardní postupy jako například určování priority pomocí aukce (Carlino a kol., 2013). Tento přístup s sebou ale nese určité morální otázky, tudíž se jedná spíše o experimentální než praktické dílo. Pojďme se podívat na příklady, které spíš u prvních v praxi nasazených systémů budou použity.

FCFS

FCFS je označení, které se často vyskytuje v existujících pracích. Tato zkratka znamená *First Come First Served*, což bychom mohli přeložit klasickým příslovím „kdo dřív přijde, ten dřív mele“. Jednoduše pořadí příjezdů zajišťuje i pořadí odjezdů (či alespoň pořadí konfliktních rezervací). Tento přístup má výhodu férovosti – nemůže se stát, že by se nějaké vozidlo zaseklo a nikdy nedostalo rezervaci. Nevýhodou naopak je zjevná potřeba pro čekání, aby se zajistilo správné pořadí.

Tento přístup byl použit už v práci Dresner a Stone (2008), ale najdeme ho pochopitelně i v dalších obměnách. Například Jin a kol. (2013) jej aplikuje na celou skupinu („platoon“) vozidel, která jedou spolu.

Optimalizace

Optimalizační přístupy mají za cíl najít ten nejlepší možný plán. Snaží se snížit vytvořené zpoždění na minimum. Toto řešení má značnou nevýhodu v tom, že v reálném světě můžou nová vozidla přijíždět kdykoli, takže vždy musí dojít

k přepočtu. Navíc měnění dříve vytvořených rezervací znamená zátěž na komunikační složku. Dalším aspektem je samozřejmě to, že se jedná o složitý problém, kdy máme mnoho možností a spolehlivě najít tu nejlepší znamená testování potenciálně velkého množství situací. Z těchto důvodů je možné, že se tomuto řešení praktičtěji orientované výzkumy vyhnou.

Některé práce (například Emami a kol. (2018)) koordinují autonomní vozidla s lidskými řidiči a pomocí principu *SPaT* (signál, fáze a časování) plánují těm autonomním rychlosti tak, aby se správně synchronizovaly se zelenou fází zbylých aut.

Dále se můžeme setkat s výzkumy (jako třeba Jin a kol. (2012)), kde byl problém převeden na instanci lineárního programování, kde účelovou funkcí je minimalizovat celkové časy průjezdů a jako podmínky jsou použity nutná bezpečnostní omezení.

Heuristiky

Poslední přístup v sobě zahrnuje vlastně několik různých řešení. Typicky se pohybují někde mezi dvěma dříve zmíněnými přístupy s tím, že se ho pak snaží nějak chytře a rychle vylepšit. Heuristiky nemusí nutně splňovat žádný konkrétní cíl jako férovost či optimalita, ale přináší dobré zlepšení za nízkou cenu.

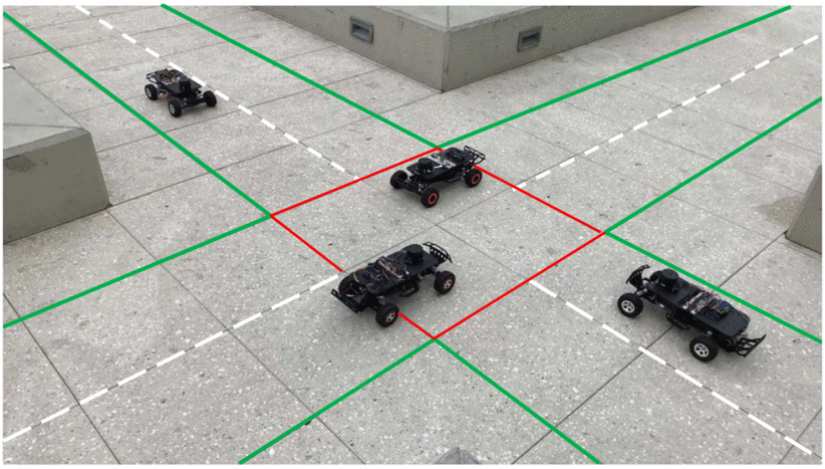
Například práce Tachet a kol. (2016) přichází s poměrně přímočarou heuristikou a to je systém BATCH, který vždy za nějaký časový úsek vezme vozidla v dávce a těm naplánuje optimální průjezd.

1.2 Dynamika vozidel

Důležitým prvkem, obzvlášť pokud se jedná o simulovaný výzkum, je fyzikální model simulace. Týká se to především samotných vozidel, ale je dobré si říct, že vliv na výsledek může mít i prostředí – materiál cest, viditelnost, náklon komunikačních ploch, vliv počasí, ... Většinou tyto aspekty práce ignorují a my se jimi také nebudeme zabývat, ale je dobré si uvědomit jejich přítomnost a reálná implementace by jim měla věnovat prostor při návrhu.

Co se týče fyzikálního modelu vozidel samotných, mnoho studií přistoupilo k jednodušším řešením, kdy se zabývají jen dopřednou pozicí vozidla. Složitější přístup představuje 2D model, kdy je do výpočtů zahrnuta druhá dimenze včetně úhlu kol a zatáčení (používají jej například Dresner a Stone (2008), Khayatian a kol. (2019) či Khayatian a kol. (2018)). Dá se jít i dál a posuzovat další vlastnosti jako odpor vzduchu, úhel plochy a váhu vozidla. Zesložiťování modelu však i zvyšuje komputační složitost, a proto je tato volba spíše výjimečná (například Bichiou a Rakha (2018)).

Existují práce, které se odchýlily od přístupu čistě skrze virtuální prostředí a experimenty prováděly s fyzickými modely. V takových případech je ale potřeba počítat s vyššími náklady a náročností provedení a situace jako mnohoproudá křižovatka s desítkami aut jsou v součastnosti spíše nerealizovatelné. Khayatian a kol. (2018) vytvořili jednu křižovatku s modýlky v desetinnové velikosti (na obrázku 1.1). Práce Fayazi a kol. (2019) používá skutečné vozidlo v zapojení „vehicle-in-the-loop“, tedy fyzické auto napojené na simulační prostředí.



Obrázek 1.1: Experimenty s fyzickými modely. Zdroj: Khayatian a kol. (2018)

1.3 Technické aspekty

Přestože většina výzkumů pracuje s částečně (ale většinou úplně) simulovaným prostředím, měli bychom se podívat na to, jak přenositelné jsou jednotlivé části do reality.

Předpokládá se, že většina vozidel je vybavena nějakým zařízením, které zjišťuje jejich polohu – typicky GPS. Ale právě GPS může být dost nepřesná, obzvlášť v situaci, kdy potřebujeme přesné hodnoty metrů a rozhodovat se v malých jednotkách vteřin. Často může dávat smysl vybavit samotnou křižovatku různými senzory, které potvrzují hodnoty dodané vozidlem, to ale zvyšuje náklady řešení.

Poměrně málo studií řeší časovou synchronizaci. V simulovaném prostředí je velmi snadné zajistit jeden čas, na který se všichni spoléhají. V realitě se můžeme setkat s nesynchronizovanými hodinami, zpožděním v komunikaci a dalšími překážkami. Ale existují i výzkumy, které se vypořádávají s toutou zábranou (například Khayatian a kol. (2019)).

Komunikace mezi vozidly a křižovatkou (či vozidel mezi sebou) by mohla tvořit kapitolu sama o sobě. Je potřeba počítat s potenciální ztrátou zpráv, zpožděním, rušením, mylnými zprávami atd. Souvisí s tím to, že se může stát, že simulace počítá s nějakým modelem vozidla, ale to ve skutečnosti vypadá jinak.

Je také třeba si uvědomit, že v simulaci či v prostředí, kde jsou nám v rámci testování dostupné téměř neomezené zdroje, nemusí proběhnout správné otestování potřebného výkonu hardware. Simulace si může dovolit libovolně zastavovat a zpomalovat čas, dokud se vše potřebné nespočítá. V realitě však musíme myslet i na to, aby bylo možné dosáhnout výsledku v poměrně krátkém čase bez nutnosti použití drahých (vysoce výkonných) strojů. Kvůli tomu přístupy jako optimalizační scheduling policy nemusí moc dobře fungovat. Bohužel je to část, která se velice těžko porovnává napříč pracemi.

Další problém, který se pojí především se skutečnou podobou, tkví v existenci neautonomních vozidel. Dnes je stále drtivá většina řidičů lidských. I kdybychom měli přejít na čistě autonomní řešení, určitě bude fáze, v které se budou na silnicích pohybovat oba typy vozidel. Proto dává smysl hledat tzv. hybridní algoritmy, které hledají spojení současných systémů a těch budoucích. Tuto problematiku řeší už Dresner a Stone (Dresner a Stone, 2008), kdy vlastně má křižovatka pro

dva typy vozidel různé protokoly (AIM pro autonomní, semafory pro lidmi řízená). Zjištění bylo, že už při malém počtu lidských řidičů se tím snižuje efektivita a zvyšuje zpoždění, protože autonomní agenti za nimi musí čekat. Au a kol. (2015) se snaží tuto nevýhodu řešit zapojením tzv. semi-autonomních vozidel, které umožňují zapojení lidských řidičů do rezervačních systémů s nutností vykonávat jen několik základních akcí.

1.4 Simulátory měst

Existují už hotové systémy pro zkoumání a simulování autonomních vozidel, křižovatek a pohybu po silnicích. Jelikož je nějaké výzkumy použily, několik jich tu zmíníme.

Mezi nejrozšířenější systém patří SUMO (Lopez a kol., 2018), který poskytuje open-source 2D simulaci pro zkoumání městského pohybu. SUMO nabízí mnoho měřících nástrojů pro záznam statistik. Systém se používá i v realitě pro odhady provozů na skutečných úsecích komunikací.

VISSIM (Fellendorf a Vortisch, 2010) nabízí pokročilejší simulaci ve 3D a řada výzkumů jej využívá. Systém však není tak otevřený a z oficiálních stránek se dá získat pouze demo verze.

Dalším 3D simulátorem, který najde využití je Gazebo (Koenig a Howard, 2004), které umožňuje vysoce přesnou reprezentaci vozidel včetně senzorů, což se ale velice silně projeví na výkonu, a tedy je běh těchto simulací velice hardwarově náročný.

Hodně výzkumníků se rozhodlo vyvinout si vlastní simulační systém, ve kterém si vše otestovali. Velkou výhodou je, že autoři mají vše pod kontrolou. Nevýhodou je pak samozřejmě horší spolupráce s jinými řešeními a neexistence společného rozhraní (což se ale stane i při použití více než jednoho existujícího programu).

Obrázek 1.2: Křižovatka v SUMO (Lopez a kol., 2018)

1.5 Kooperace křižovatek

Zde už bohužel existující výzkum pokulhává, neboť se většina prací zaměřuje pouze na jednu křižovatku. Hausknecht a kol. (2011) zkoumají zapojení 4 křižovatek s použitím algoritmu A*, kde odhalili výskyt Braessova paradoxu (Braess, 1968), který způsobuje, že přidáním cest se celková propustnost sníží, protože všechna vozidla sobecky soupeří o nejvýhodnější cestu a nikdo nechce použít tu méně vytíženou, protože vychází jako pomalejší. To vede k tomu, že se silnice ucpou a nakonec jsou celkové časy horší, než by mohly být. Odebráním cesty se tedy propustnost zvýší. Článek Hausknecht a kol. (2011) se zabývá právě vlivem tohoto paradoxu v prostředí autonomních křižovatek.

Wuthishuwong a Traechtler (2013) zkoumají spolupráci autonomních křižovatek na vyšší úrovni a ve své práci se snaží zabránit zahlcení systému pomocí chytrého plánování. Nejdřívě objevují hustotu, kdy spojením může cestovat co nejvíce vozidel za hodinu a následně se snaží jí dosáhnout.

2. Simulační prostředí

V předchozí kapitole jsme si popsali mnoho různých přístupů k problémům, které se v našem díle vyskytují. V této kapitole si povíme, jaké z nich byly použity v této práci.

Problém simulujeme čistě ve virtuálním prostředí. Jedná se o dostupný a efektivní způsob provádění škály experimentů. Jako základ simulačního software, který tato práce představuje, byl použit engine Unity (Unity Technologies, 2023a). Výhody této volby jsou:

- Připravený základní fyzikální systém
- Připravený 3D renderovací systém
- Použití moderního jazyka `C#` (Microsoft, 2023)
- Objemná dokumentace a rozsáhlá komunita
- Možnost využití předpřipravených modulů a knihoven (Unity Technologies, 2023b)
- Přizpůsobitelnost všech částí – teoreticky žádné limity toho, co bude výsledný systém umět

Samozřejmě s sebou tento přístup nese i určité nevýhody jako je absence specializace na problematiku řízení provozu (jako poskytuje například systém SUMO (Lopez a kol., 2018)) či související pravděpodobně nižší znalost v komunitě, která se problematice věnuje. Ve výsledku se jedná o řešení, které přináší vlastní implementaci, což není až tak neobvyklé (Khayatian a kol., 2020). Do finálního rozhodnutí zasáhly i fakty jako předchozí zkušenosti autora práce s tímto systémem a kladný vztah k programovacímu jazyku `C#`.

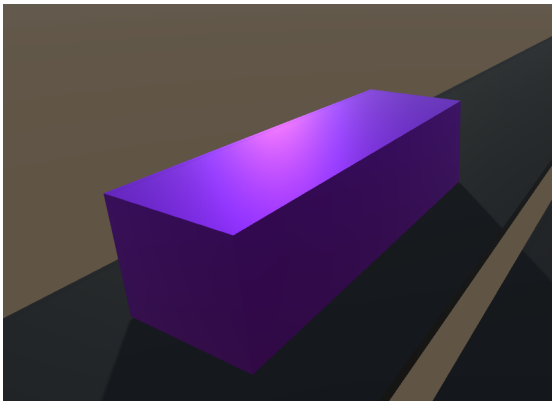
Tyto stavební kameny nám umožňují přizpůsobit vývoj systému jasnému účelu: spuštění co nejvíce experimentů za co nejkratší dobu. Proto ne všechny části kódu a vnitřku simulace reflektují jednotlivé komponenty, které by byly v reálném prostředí potřeba. Například vozidla o sobě vědí díky informacím v kódu, nikoli za použití simulovaných senzorů, které by představovaly snímání okolí auta. Toto však nevadí, jelikož uživatel vnitřní implementaci nepozná a navenek se simulace chová věrně. Její běh (a také vývoj systému) tedy nebyl zpomalen zabýváním se těmito detaily.

V simulaci předpokládáme autonomitu veškerých vozidel. Tedy že jejich řidičem není člověk, ale umělá inteligence, která je schopná přímo používat elektronické systémy a komunikovat v mnohem vyšší rychlosti. Na tomto základu stavíme natolik, že jsme se rozhodli ignorovat potenciální (byť malou) chybovost. V simulaci tedy vystupují ideální autonomní řidiči, kteří se vždycky chovají dle připraveného kódu. Dojde-li k nějaké nepředpokládané situaci (například kolizi dvou vozidel), je zaviněna právě chybou v kódu, nikoliv nějakým nenadálým vlivem prostředí, který by mohl nastat v reálném světě.

Většinou budeme v textu používat pojem *vozidlo* nebo *auto*, případně *agent*. Použijeme-li slovo *řidič*, také tím myslíme autonomní objekt. Kdyby se mělo jednat o lidského řidiče, bude to explicitně zmíněno.

2.1 Model vozidel

Nejprve je potřeba si uvědomit, že skutečný automobil je velice sofistikované zařízení s mnoha součástmi. Stejně tak silnice, po kterých se auta pohybují, nejsou jenom rovné plochy. Cílem této práce je však zkoumat obecnější části provozu, proto došlo k určitým zjednodušením. Na modelu vozidel se to projevilo nejspíš nejvíce. Byla implementována jen níže popsaná funkcionalita. Výsledek však dostatečně dobře aproximuje realitu a pro cílové měření dat nám tento model postačí.



Obrázek 2.1: Model auta v simulaci

Jako vozidla byly použité prosté kvádry s barevným odlišením (příklad na obrázku 2.1). Simulace tím jistě pozbývá na realističnosti, ale hlavní myšlenkou je, že rozměry kvádru jsou nastavené tak, aby se do něj typické auto vešlo. Použitý model je jakýmsi horním odhadem běžného auta. Rozměry jsou nastavené následovně: výška = 1.4 m, šířka = 1.7 m a délka = 4.6 m. Simulace je teoreticky připravená na jiné tvary vozidel – základní délka kvádrů se dá nastavovat v konfiguračním souboru (A.2.7).

Vozidla zrychlují a zpomalují lineárně. Zrychlení je tedy konstantní. V konfiguraci se hodnoty dají nastavit. V základu je brždění nastaveno na $10\,\mathrm{m/s^2}$ a akcelerace na $5\,\mathrm{m/s^2}$. Jedná se o poměrně vysoké hodnoty, ale náš systém se snaží simulovat situaci, která v realitě nastane nejspíš až za desítky let. Předpokládáme, že přítomná vozidla schopná plně samostatné navigace disponují kvalitním motorem s vysokou akcelerací a decelerací. Když chce vozidlo změnit rychlost, používá k tomu vždy nejvyšší hodnoty, aby se změna odehrála co nejrychleji.

Auta se otáčejí kolem středu svého těla (geometrický střed kvádru). Nastavení úhlu předních kol (resp. jeho simulační ekvivalent) probíhá instantně. Toto samozřejmě neodpovídá realitě. Pro účely zjednodušení simulace bylo přistoupeno k tomuto rozhodnutí. Ve výsledku to má tu odchylku, že vozidla se pohybují po jiných drahách, než by se pohybovaly ve skutečnosti. Stačí jim na začátku zatáčky nastavit správný úhel a na jejím konci se opět vrátit do přímočarého pohybu. V praxi by auto muselo začít zatáčet dřív a na konci zatáčky začít vracet úhel kol zpět dopředu také dřív.

Skutečná vozidla nejsou schopná projet zatáčku stejnou rychlostí, jako by to byla rovná cesta. Oproti dřívějším zjednodušením jsme se na tento fakt zaměřili a ujistili se, že toto omezení bude zachováno. Pokud chce auto vjet do ostré zatáčky, musí nejdříve zpomalit. Do programu to bylo zavedeno tak, že podle

míry zatáčení je na daném úseku omezená maximální rychlost. Vozidla zpomalují předem, aby při vjezdu do zatáčky už měla správnou rychlost.

Většinu běhu simulace kontrolujeme, zda nedošlo ke kolizi, ale samotná vozidla nemají žádný fyzikální model ve smyslu, že by kolize měla vliv na jejich cestu. Pokud k ní dojde, auta zkrátka projedou skrz sebe, jakoby tam to druhé nebylo.

2.2 Náhodné generátory a seedy

Naše simulace na mnoha místech (generování mapy, algoritmů, provoz, ...) používá náhodně generované hodnoty. Jedná se pochopitelně o pseudonáhodné generátory, které je potřeba nastavit *seedem* (dalo by se přeložit jako „semínko“, ale čeština nám zde obecně moc nepomůže). Seed započne sekvenci generovaných čísel. To znamená, že dva stejné seedy vygenerují tu samou sekvenci.

Veškeré náhodné generování v simulaci má svůj seed. Dokonce se jedná o několik seedů, aby každá část šla nastavit zvlášť. Některé seedy jsou zabudované v konfiguraci a jiné jde nastavit v menu – spolu s rozhodnutím, zda se mají seedy automaticky generovat, nebo zůstát pevné. Pokud pustíme příslušný kód (generování mapy/běh simulace/...) dvakrát po sobě se stejným seedem, vygenerované hodnoty budou stejné (vzhled mapy/naplánované cesty/...).

2.3 Generování města

Důležitou částí, kterou je potřeba nasimulovat, je samotné prostředí, kde se vozidla pohybují. Zde se také setkáváme s mnoha možnostmi (dopodrobna se tím zabývá například Shi a kol. (2009)). Pro hladký průběh simulování si opět musíme nastavit mantinely a zjednodušit problém dostatečně, aby se dal efektivně testovat. Naše řešení se zaměřuje na jeden z nejčastějších případů – město s „klasickými“ křižovatkami s 4 vstupními/výstupnimi směry. S tím, že samozřejmě ne všechny směry musí být na každé křižovatce skutečně využité. Dodávaný generátor oplývá schopností napojit několik takovýchto křižovatek vedle sebe a vytvořit systém – mřížku (viz obrázek 2.2).

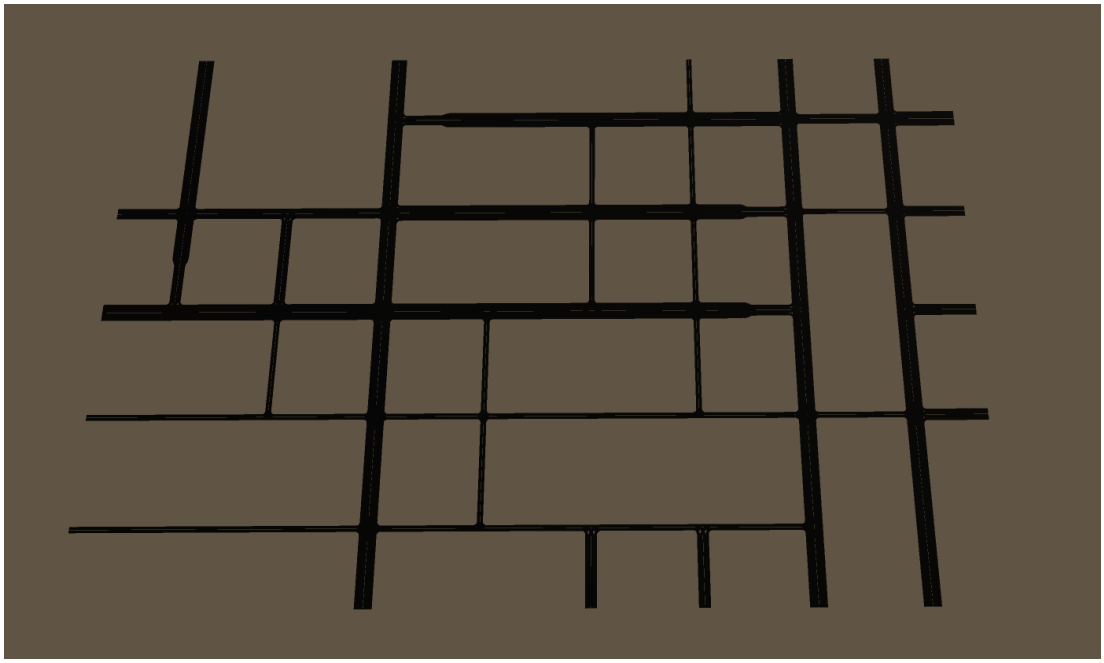
Mnoho částí generování se dá nastavit (detailní popis všech možností lze najít v uživatelské dokumentaci: A.2.2). Mřížka může zahrnovat jen jednu křižovatku nebo naopak až vyšší jednotky v obou směrech. Menu poskytuje dostatek možností, jak si systém nakonfigurovat.

2.3.1 Hlavní a vedlejší silnice

Generátor rozlišuje dva typy cest, které se dají nastavit – hlavní a vedlejší. Toto nastavení má vliv *pouze* v rámci generování. Jakmile je město postavené a simulace běží, auta kromě místa, kde se objeví, nedokáží rozlišit, jaký je typ silnice. Uživateli simulace to však může pomoct.

Hlavní silnice se generují první a mají pevně daný počet pruhů. Buď jsou náhodně rozmístěné, nebo rozložené ve městě s rovnoměrnými rozestupy. Mohou být jednosměrné, nebo obousměrné.

Vedlejší silnice se generují až po hlavních. Buď se napevno doplní mřížka tam, kde hlavní silnice nejsou (s pevným počtem pruhů), nebo se generují náhodně



Obrázek 2.2: Vygenerované město

(pak lze i počet pruhů na každé silnici náhodně generovat v určitém rozmezí). Náhodné cesty můžou být jednosměrné.

Náhodné generování

Výše jsme psali, že silnice mohou být generovány náhodně – týká se to jak hlavních, tak vedlejších s tím, že všechny hlavní se vygenerují jako první a vedlejší jako druhé. Způsob generování je ale stejný (je-li zvoleno náhodné generování pro oba typy).

Na vstupu je vždy vyžadovaný *počet* generovaných silnic. Algoritmus 1 popisuje chování generátoru. Když algoritmus mluví o otevírání a uzavírání bodů na mřížce, jedná se o místa, kde se cesty budou protínat (a následně zde budou umístěny křižovatky), a konce mřížky. Každý takový bod (kromě krajů) má vlastně 8 nezávislých „podbodů“ – pro každý ze 4 hlavních směrů 1 začátek a 1 konec. Každý z těchto bodů-směrů je na začátku v prázdném stavu, následně může být otevřen a poté i uzavřen.

Obrázek 2.3 zachycuje průběh generování jednosměrných cest. Body na mřížce jsou barevně zvýrazněny včetně směru, kterým výsledná silnice směřuje. Červené body již byly uzavřeny (o tom svědčí již znázorněná silnice). Zelené jsou otevřené, tudíž jsou připravené na to přijmout nový stav. Modré body jsou prázdné, takže tudy nejprve musí nějaká silnice procházet, než se mohou otevřít. Každá křižovatka má tedy 8 takovýchto bodů a 2 jsou na všech okrajích mřížky (na obrázku vpravo). Na obrázku je nezajímavější nejspíš pravá dolní křižovatka, kde najdeme všechny tři typy bodů. Zelené jsou otevřené proto, že kdyby tu vznikla cesta, tak už jde napojit na některou z dříve přítomných. Modrý bod má však ještě prázdný stav, protože kdyby v něm měla končit silnice, nejde ji napojit na žádnou další.

Pokud hlavní silnice nejsou vygenerovány náhodně, v místech, kde vedou, se stejně před generováním vedlejších otevřou začátky a konce stejně, jako by

Algoritmus 1 Generování náhodných silnic

```
Označ krajní (vstupní a výstupní) body mřížky za otevřené začátky a konce
for Každá silnice, která se má vygenerovat do
    Najdi otevřený začátek
    if Otevřený začátek neexistuje then
        Skonči
    end if
    Najdi otevřený konec, do kterého jde přivést silnice ze začátku ▷ Díky tomu, jak algoritmus funguje, je garantované, že takový konec existuje
    Spoj začátek s koncem novou silnicí
    if Silnice mají být obousměrné then
        Spoj začátek v opačném směru v bodu konce s koncem v opačném směru v bodě začátku
        Následující příkazy vykonej pro oba směry
    end if
    Uzavři začátek a konec
    Otevři neuzavřené body na mřížce, které silnice protíná, ve směrech kolmých na přidávanou silnici
    if Začátek není na kraji mřížky then
        Otevři všechny konce v tomto bodě kromě toho, který sousedí se začátkem
    end if
    if Konec není na kraji mřížky then
        Otevři všechny začátky v tomto bodě kromě toho, který sousedí s koncem
    end if
end for
```

náhodně generované byly.

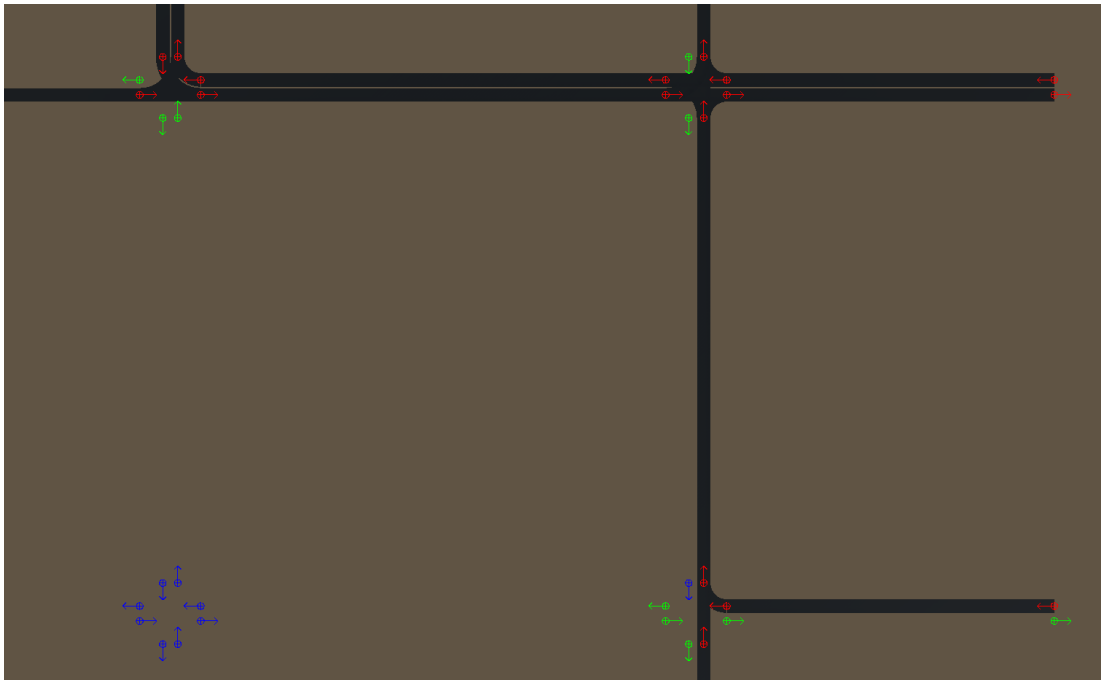
Pokud jsou hlavní silnice jednosměrné, začátky a konce se uzavírají, jako by byly obousměrné. To znamená, že jakmile je někde jednosměrná hlavní silnice, nemůže už se vygenerovat žádná další silnice v opačném směru. U vedlejších cest to však neplatí. Generují-li se náhodně a někde se umístí jednosměrná, stále je opačný směr otevřen a dají se sem generovat nové silnice (nemusí být stejně dlouhé jako ta původní).

2.3.2 Křižovatky

Generátor vytváří „klasické“ křižovatky se 4 hlavními směry potom, co už byly vygenerovány cesty. Přítomna může být i libovolná podmnožina vstupních a výstupních pruhů, pokud „dává smysl“, tzn. pokud každý vstupní pruh vede do nějakého výstupního a do každého výstupního nějaký vstupní.

Teoreticky by šlo nastavovat libovolné množství pruhů, ale v simulaci je to omezeno na 0 až 3. Více jich na většině městských křižovatek nepotkáme. Ať už je počet jakýkoli, pro spojování vstupních a výstupních pruhů platí následující pravidla. Předpokládejme, že pro nějaký směr existuje alespoň 1 vstupní pruh, pak platí:

- Pokud existuje výstupní pruh po zahnutí doleva, nejlevější takový je napo-



Obrázek 2.3: Generování náhodných silnic. Modré body – prázdný stav, zelené – otevřené, červené – uzavřené

jen na nejlevější vstupní pruh.

- Pokud existuje výstupní pruh po zahnutí doprava, nejpravější takový je napojen na nejpravější vstupní pruh.
- Pokud existují nějaké výstupní pruhy ve směru dopředu, jsou napojeny na vstupní pruhy, které jsou naproti nim – od středu, tedy od nejlevějších pruhů. Například 2 vstupní se 2 výstupními. Počty se přepočítávají a případně upravují (viz další odstavec). V případě, kdy by byly naproti sobě 2 výstupní a 1 vstupní, propojí se vstupní s nejlevějším výstupním a ještě musí existovat vstupní pruh v jiném směru, který se napojí na ten druhý výstupní.

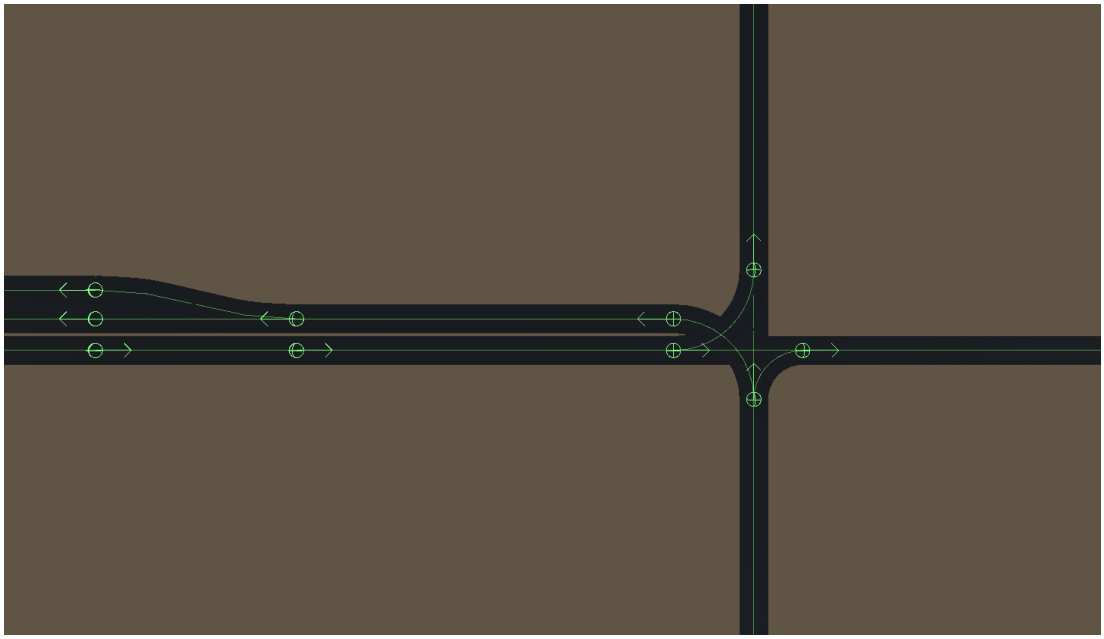
Počty pruhů se přizpůsobují mezi křižovatkami. Pokud by do nebo z křižovatky měla vést cesta, na které bude pruh, který nepůjde nikam napojit, tak se mezi 2 křižovatkami počet pruhů sníží, aby k tomuto problému nedošlo.

Možný výsledek je vidět na obrázku 2.4. Pruhy, které šly napojit, byly napojeny. Zároveň si všimněme, že 2 výstupní pruhy by vlevo z křižovatky nemohly vycházet (nejsou vstupní pruhy, které by se na ně napojily), takže se na tento počet rozšíří až v prostoru mezi křižovatkami.

Generátor také zahrnuje možnosti pro výběr algoritmů křižovatek – více v části 2.5.

2.3.3 Spawnování aut

Pojem *spawnování* pochází z prostředí počítačových her. Typicky se jím popisuje moment, kdy se nějaká postava/objekt objeví v prostředí a začne fungovat jako jeho součást. Jelikož Unity je také herním enginem, bylo toto slovo v práci



Obrázek 2.4: Vygenerovaná křižovatka

použito, neboť nejlépe vystihuje příslušnou událost v simulaci. Jednotliví agenti, než začnou jednat, se musí *spawnout* v simulačním prostředí. To znamená, že vozidla jsou umístěna na silnici a spustí se veškeré jejich ovládání.

Distribuce spawnování a počet celkových aut se dá nastavit v menu (A.2.2). Celkový počet znamená, kolik se vždy může pohybovat vozidel v simulaci *najednou*. Simulace se snaží spawnovat další a další (je-li na ně prostor), dokud tohoto čísla aktivních řidičů nedosáhne.

Město končí otevřenými cestami, které představují potenciálně navazující jinou část dopravy. V tomto bodě se ale v simulaci právě auta uměle spawnují, nebo naopak despawnují (opak spawnování – tzn. zmizení), pokud jsou na konci cesty. Po tom, co je auto odstraněno ze simulace, sníží se tím počet aktivních vozidel, a tím pádem se vytvoří prostor pro nový spawn dalšího auta.

2.4 Navigace

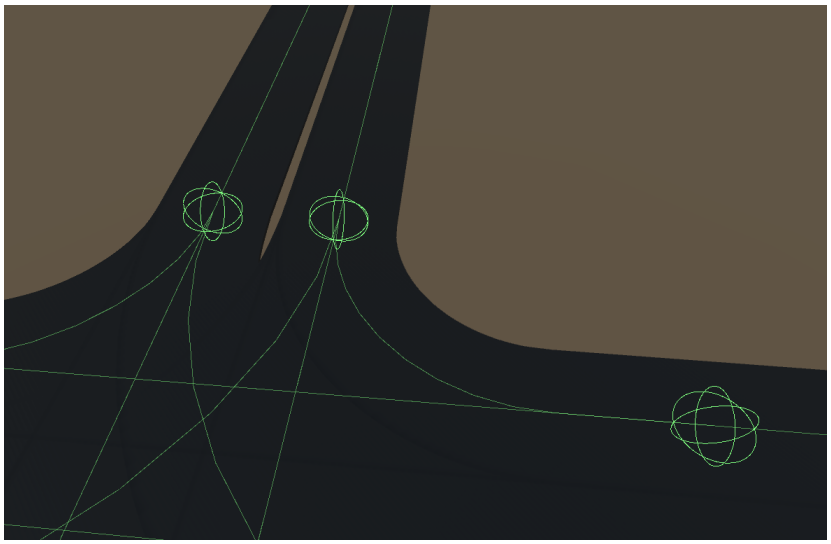
Autonomní agenti jednotlivých vozidel kromě základního ovládání potřebují i nějaký systém, pomocí kterého bude probíhat navigace po silnicích. Veškeré cesty jsou reprezentovány virtuálními konstrukty – vrcholy, tzv. *waypointy* a hranami (*edge*).

Waypoint reprezentuje bod. Každá *hrana* začíná a končí waypointem. Waypoint zahrnuje směr, kterým skrz něj mají vozidla projíždět.

Hrana představuje spojení dvou *waypointů*. Auta následně jezdí přímo po hranách. To znamená, že hrana v sobě zahrnuje i geometrii.

Hrany jsou navržené tak, aby vždy spojily dva waypointy cestou skrze jejich směry, po které lze jet. To znamená, že se dynamicky spočítají po vygenerování mapy. Hrana nemusí být nutně rovná a spojení dvou waypointů může být odlišné podle jejich natočení (směru průjezdu). Auta (středy kvádrů) se následně pohybují po trajektoriích hran. Waypointy slouží především jako předělové body

- při vstupu do prostoru křižovatky, při registraci před křižovatkou, pro začátek a konec cesty apod.



Obrázek 2.5: Waypointy (drátěné koule) a hrany (spojující linky)

Skrz systém hran je implementována i kontrola okolí aut. Hrany vědí, jaká vozidla se na nich pohybují a v jakém pořadí a tyto informace autům předávají, aby si mezi sebou mohla udržovat bezpečné vzdálenosti. K výpočtu bezpečné vzdálenosti můžeme přistupovat dvěma způsoby – *naprosto bezpečným* a *hodně nebezpečným*. *Naprosto bezpečný* znamená, že auto udržuje takovou vzdálenost, aby kdykoli bylo schopno bez nárazu zabrzdit, i kdyby vozidlo před ním instantně zastavilo. *Hodně nebezpečný* naopak počítá s tím, že auta mají stejné vlastnosti, tudíž když předchozí vozidlo začne brzdit, auto začne brzdit také a stačí mu udržovat minimální vzdálenost.

Hodně nebezpečný přístup v simulaci funguje, ale výsledek je dosti komický. Auta jedoucí za sebou drží stejné vzdálenosti a zrychlují a brzdí „spolu“. V realitě by toto řešení určitě nefungovalo, protože změny rychlostí můžou být dosti náhlé (například kvůli nárazu). *Naprosto bezpečný* přístup je zase příliš pesimistický. Auta typicky nezastavují instantně a obzvlášť autonomní řidiči disponují rychlými reakcemi a mnoha možnostmi, jak se vyhnout nebezpečí.

Náš systém volí střední cestu – dá se říct, že přesně. Auta udržují vždy polovinu naprosto bezpečné vzdálenosti. K této vzdálenosti se ještě přičítají napevno 2 m, které jsou dodržovány i když vozidla stojí.

Tento systém je důležitý i proto, že ho přímo využívají autonomní křižovatky (2.5.2).

2.4.1 Přejezdy mezi pruhy

Práce počítá s tím, že auta mohou jezdit takřka odkudkoliv kamkoliv (v rámci klasických silničních pravidel). Proto je potřeba implementovat nějaký způsob změny pruhů, aby auta mezi křižovatkami jela tam, kam potřebují.

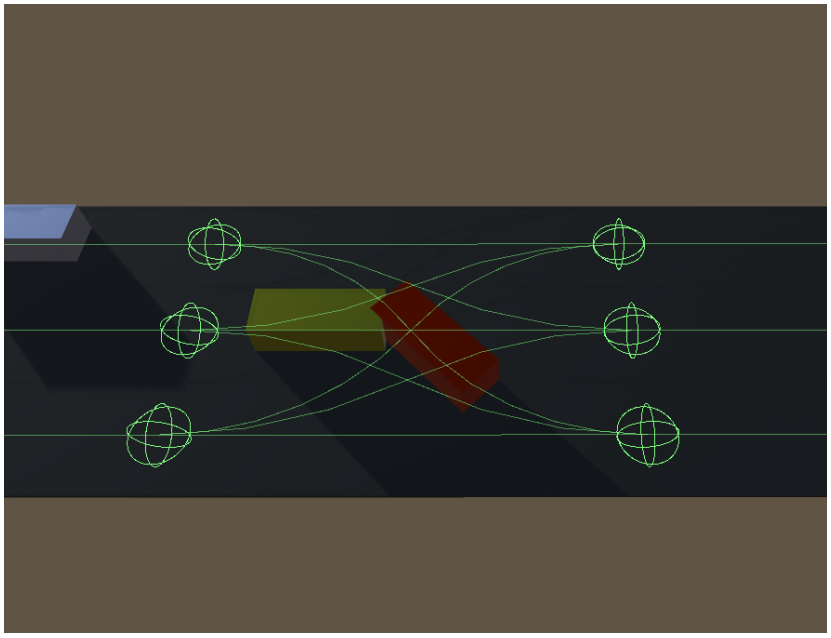
Řešení přejezdů mezi pruhy by vystačilo na celou samostatnou práci. Problematika zahrnuje mnoho malých částí, které je potřeba promyslet a vyřešit.

Vyzkoušeli jsme několik primitivnějších přístupů, které ale všechny měly svoje nevýhody.

Jeden ze způsobů, který jsme zkusili, bylo úsek přejezdů mezi pruhy považovat za samostatnou křižovatku a přidat nový algoritmus, který nechává projíždět vozidla jedoucí paralelně, ale pokud se jejich cesty slučují nebo kříží, utvoří se čekací fronta jako na stopkové křižovatce (2.5.1). Toto však vedlo k tomu, že vozidla úplně zastavovala a čekala na plný průjezd toho, kdo je blokoval. Při vyšší zátěži se začínaly vytvářet nesmyslné fronty, které celkový systém dost zasekaly, takže jsme toto řešení museli vyřadit.

Nakonec jsme dospěli k závěru, že plnohodnotná změna pruhů by byla příliš náročná a zbytečně tím zatěžovala rozsah práce. Jednoduchá řešení (jako zmíněný příklad) by zase měla silný negativní dopad na efektivitu a došlo by k zánosu naměřených dat.

Pro změnu pruhů je mezi každými dvěma křižovatkami speciální úsek, kde vozidla mohou přejíždět z libovolného „vstupního“ pruhu do libovolného „výstupního“. V tomto úseku se vypne kontrola kolizí a je možné, že zde auta projedou skrz sebe. Jakmile úsek vozidlo opustí, kontrola kolizí, respektive udržování vzdálenosti se opět zapne a mezi vozidly se utvoří mezery (pokud například vyjely v sobě). Vozidla se i v úseku snaží vzájemně co nejvíce respektovat – pokud jedou dvě auta za sebou, tak se zde nestane, že by do sebe narazila. Ke srážkám může dojít pouze, pokud se dva pruhy kříží, nebo slučují (viz. obrázek 2.6).



Obrázek 2.6: Změna pruhu na 3 pruzích s „přijatelnou“ kolizí

Výsledkem tohoto řešení by měla být co nejlepší (jednoduchá) aproximace toho, co se při přejezdech děje v realitě. Tedy že přejíždění probíhá organicky a příliš nezpomaluje celkový provoz. Typicky při změně pruhu dojde pouze k dočasnému zpomalení. Na průběh uvnitř křižovatky nemá přejíždění téměř žádný vliv. Těchto efektů je dosaženo i v naší práci, proto jsme s výslednou podobou spokojení.

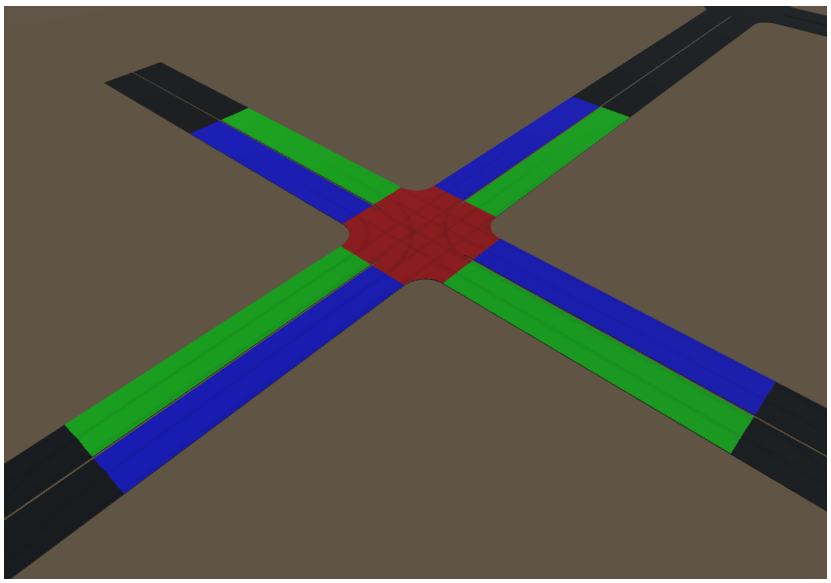
2.4.2 Hledání cest

Autu se při spawnu vyberou náhodné počáteční a koncové body na okrajích města (dle uživatelem zadané distribuce). Před během simulace se pro všechny takovéto dvojice spočíta nejkratší cesta pomocí algoritmu $A*$ (Hart a kol., 1968). Hodnotící funkcí pro jednotlivé hrany byla doba, kterou auto stráví jízdou přes tuto hranu nejvyšší možnou rychlostí. Jakmile se vozidlu přiřadí trasa městem, tak už po celou svou jízdu tuto cestu následuje.

Práce nezahrnuje žádný pokročilejší systém navigace, který by například používal vytíženost v určitých úsecích, aby autům našel jinou cestu. V realitě takové systémy samozřejmě existují (Skog a Handel, 2009), ale do rozsahu naší práce nespadají, proto nebyly implementovány.

2.5 Algoritmus křižovatky

Algoritmem křižovatky se rozumí řídící systém křižovatky. Ve skutečném světě se většinou setkáváme se semafory. Dostatečně malé křižovatky jde vyřešit i velice prostým řešením jako je stopka. Naopak pro opravdu velké uzly často vzniká samostatná infrastruktura – dálnice se kříží pomocí tzv. cloverleafů (porovnání efektivity nabízí např. Galletebeitia (2011)), kde nemůže dojít ke kolizi aut v různě jedoucích směrech. Podobné prvky ale naše simulace neobsahuje, neboť právě kolizi (a jejímu předejití) se věnujeme nejvíc. Navíc ve městech po světě se nachází mnoho frekventovaných křižovatek, které cloverleafem nahradit ani nejde (byť se jedná o efektivní řešení, vyžaduje velké množství prostoru). Proto musíme zkoumat rozličné algoritmy, které fungují na libovolně malém prostoru.



Obrázek 2.7: Zóny křižovatky: červená = hlavní kolizní zóna, modrá = zóna vjezdu, zelená = výjezdní zóna

Křižovatky zahrnují hlavní kolizní prostor (na obrázku 2.7 v červené). Jedná se o nejdůležitější část, kterou je potřeba hlídat algoritmem, protože se tu kříží vícero cest v různých směrech.

Dále kolem sebe veškeré křižovatky v simulaci mají určitou zónu, o které vědí (několik desítek metrů před samotnou křižovatkou – na obrázku 2.7 v modré). Jakmile vozidlo vjede do této zóny, křižovatka ho zaregistruje a převezme kontrolu nad řízením – dá autu příkazy, kterými se musí řídit.

Veškeré algoritmy si nějakým způsobem hlídají, aby potenciální fronta aut za křižovatkou nezasahovala do (červeného) prostoru, kde by mohlo dojít ke kolizi. Křižovatka tedy sleduje počet aut v úsecích odjezdů (na obrázku 2.7 zeleně). Jedná se o zjednodušení. V realitě by si nejspíš tento fakt auta musela hlídat sama (u „hloupějších“ křižovatek určitě). Necháváme stranou fakt, že mnoho lidských řidičů tento jednoduchý požadavek není schopno splnit (více se lze dočíst např. zde: Simopt, s.r.o. (2023)). Ve výsledném chování simulace však není poznat, kdo je zdrojem rozhodnutí nevjet do křižovatky, pokud je výjezd plný.

Vzhledově křižovatky nejsou odlišeny (například přítomností fyzických semaforů nad/vedle cest). V aplikaci je pouze malá ikonka, která uživateli prozradí použitý algoritmus.

2.5.1 Jednoduchá řešení

Jedná se o řešení, která se dnes běžně používají a jejich obecné fungování bude známé i nezasvěcenému čtenáři. Pro úplnost zde ale popis uvádíme – včetně konkrétního fungování v rámci simulace.

Stop sign

Stop sign neboli stopka („Stůj, dej přednost v jízdě“) v realitě představuje takřka bezúdržbový systém, který nepotřebuje kromě pouličního osvětlení žádnou speciální elektroniku.

V našem řešení stopky fungují tak, že auta, která vjedou do zóny křižovatky se zařadí do určité virtuální fronty, která je nezávislá na směru, odkud přijela, a čekají, až na ně přijde řada. První auto ve frontě projíždí skrz křižovatku a další může vyrazit, až když je kolizní prostor volný. Nikdy se v něm tedy nenachází více než 1 vozidlo. V realitě by provoz vypadal o něco chaotičtěji, protože některá auta by křižovatkou projížděla současně. Křižovatka by díky tomu byla o něco propustnější. Na efektivitě by ale i pozbyla, protože skutečná auta typicky zpomalují při příjezdu, i když je křižovatka prázdná nebo jsou první, kdo bude projíždět (v simulaci první auto nikdy nezastaví).

Byť není tento algoritmus zajímavý pro frekventovanější uzly (očividně je pak příliš pomalý), dává smysl se na něj podívat alespoň v elementárních situacích a zahrnout ho do simulace pro více možností.

Traffic light

Traffic light neboli semafory (světelná signalizační zařízení) jsou v praxi nejspíš nejběžnějším řešením pro řízení křižovatky. Přesto si ho pro jistotu pojďme popsat. Na první pohled se to nemusí zdát, ale existuje zde několik rozhodnutí, které je potřeba při návrhu učinit – počet jednotlivých fází, délka fází, které pruhy jezdí naráz, apod.

V našem systému fungují světelné křižovatky následovně:

- Zelená fáze trvá 10 vteřin. Následně 1 vteřinu trvá žlutá fáze. Následně se zelená fáze přepne pro další možný směr ve směru hodinových ručiček. Tyto hodnoty se dají nastavit v konfiguraci (A.2.7).

- Při zelené fázi mohou jezdit všechna auta v pruzích aktivního směru a auta z následujícího směru (ve směru hodinových ručiček), která jedou do aktivního směru. Na typické čtyřsměrové křižovatce to znamená, že pokud mají zelenou vozidla jedoucí z jihu, auta ze západu zahybající doprava mají také zelenou. Zelenou navíc má zjednodušeně ten průjezd, který se s průjezdy z aktivního směru nekříží.

- Při žluté fázi nesmí žádná nová auta vstupovat do kolizního prostoru křižovatky – jen ta, co by nestihla zabrzdit. Očekává se, že v této době se vnitřek křižovatky vyčistí pro vozidla z nových směrů. Doprava se tím na chvilku přeruší i pro průjezd, který by jel dvakrát po sobě (typicky v situaci, kdy nejdřív zatáčala jen auta jedoucí doprava, ale pak se směr zaktivnil, a tak se odbočka doprava spustila znovu spolu s ostatními průjezdy).

- Červenou fázi zde nepopisujeme, neboť vždy má nějaký směr zelenou nebo žlutou. Červenou mají všichni ostatní, výše nepopsaní. Pokud nebylo uvedeno jinak, vozidla do křižovatky nevjíždějí.

Obrázek 2.8: Průběh světelné křižovatky – auta zprava jedou všemi směry, auta zezdola pouze doprava

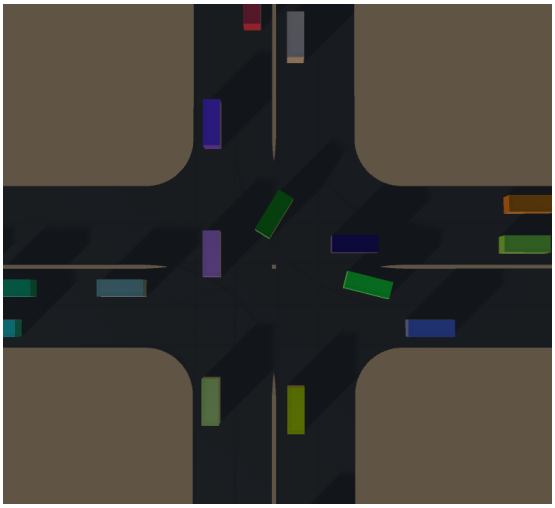
Takto fungují jednotlivě všechny křižovatky. Jenže dalším důležitým bodem je **synchronizace**. Při testování (3.5) jsme zjistili, že pokud jsou všechny křižovatky nastavené stejně, tedy zelené fáze začnou ve stejnou chvíli a pro stejné pruhy, dost

se tím sníží propustnost. V realitě se často navazující křižovatky synchronizují (Hillier a Rothery, 1967), aby vznikla tzv. „zelená vlna“ a auta mohla projet. Pro rozsah naší práce by takováto funkcionalita vyžadovala vynaložení značného úsilí a testování, a proto jsme zvolili jednodušší řešení, které ale stále výrazně pomůže celkové efektivitě.

Jednotlivým křižovatkám se náhodně nastaví první směr a náhodně se nechá křižovatka čekat v intervalu $[0, \text{délka zelené fáze})$. Pro stejnou vygenerovanou mapu jsou tato náhodná čísla stejná. Používá se *seed*, který je uložený v konfiguraci (A.2.7). K porušení dojde pouze při manuální změně algoritmů.

2.5.2 AIM

AIM = *Autonomous Intersection Management*, neboli autonomní řízení křižovatky představuje hlavní pilíř této práce. Součástí práce je algoritmus, který je připraven na různé kombinace počtů cest a směrů, které se v simulaci vyskytují. Hlavním cílem je průjezd všech vozidel skrz křižovatku bez kolize. Sekundárním cílem je uskutečnit tyto průjezdy co nejefektivněji, aby auta v kolizním prostoru křižovatky trávila co nejmenší čas.



Obrázek 2.9: Průběh autonomní křižovatky

Algoritmus používá systém přiřazení rezervací (1.1.2). Pokusí se ji pro auto najít v momentě, kdy se zaregistruje křižovatce. Pokud však rezervaci nenalezne, je možné, že ji dříve získá jiné vozidlo. Stejně tak pořadí získání rezervací nutně neurčuje pořadí průjezdů (později zarezervované vozidlo může z křižovatky vyjet před dříve zarezervovaným). Širší podrobnosti lze nalézt na následujících stránkách.

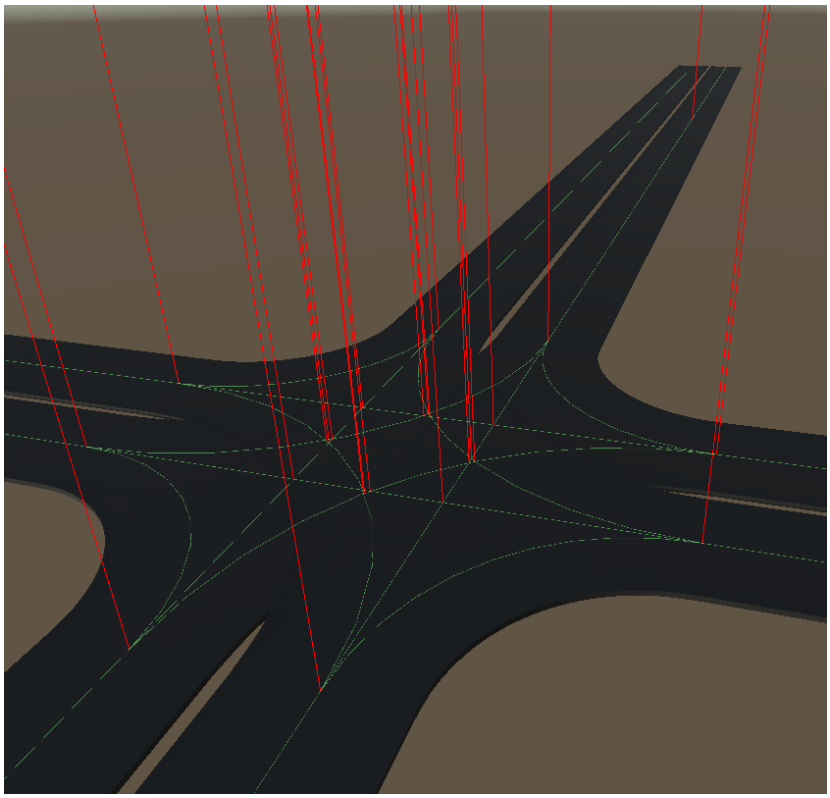
Plánování probíhá centrálně (1.1.2). Vozidla komunikují se systémem a ten jim dává další pokyny. Budeme-li v následujících řádcích mluvit o křižovatce nebo algoritmu myslí se tím právě tento systém, ovládání křižovatky. Samotná vozidla zde nečiní žádná rozhodnutí. To samé by šlo teoreticky říct o výše zmíněné implementaci semaforové křižovatky, ale zde chceme tento fakt opravdu zdůraznit. Pokud nějaké vozidlo poruší plán, může dojít k nepředvídatelným následkům (nejspíše ke kolizi) a systém v simulaci na takové výpadky není připraven. Proto ani žádné umělé výpadky nejsou přítomny.

Pro hledání potenciálních kolizí se používají trajektorie cest skrze křižovatku (1.1.3). Ty se spočítají dopředu za použití systému *waypointů* a *hran* (více popsáno dříve v 2.4). Na místech, kde se cesty rozdělují, spojují či kříží se vytvoří tzv. *konfliktní body*.

Konfliktní body jsou jediná místa (může jich být hodně), kde může dojít ke kolizi dvou vozidel. Pokud 2 cesty vedou velice blízko sebe, ale neprotnou se, nelze takové potenciální místo kolize odhalit. Jediný způsob, jak se takovýmto situacím vyvarovat, je předejít jim už při generování mapy. Při plánování rezervace je potřeba zařídit, že když vozidlo bude projíždět skrz daný konfliktní bod, nenastane situace, že by tím samým konfliktním bodem v ten samý časový úsek projíždělo i jiné vozidlo.

Konfliktní body existují na každém místě, kde se potkávají dvě cesty skrze křižovatku. Pokud se jich protíná víc na tom samém místě, pro každou dvojici je vytvořen vlastní konfliktní bod. Výjimkou jsou místa, kde se jedna cesta rozděluje, nebo se jich několik slučuje (vjezdy či výjezdy z křižovatky), pak je zde pouze jeden konfliktní bod. Pro konzistentní fungování je vytvořen i konfliktní bod na odchozím pruhu kousek za křižovatkou. K tomu se cesta plánuje, ale vozidla si už v tomto úseku přebírají sama kontrolu.

Jakmile vozidlo vjede do blízkosti křižovatky (do modré zóny na obrázku 2.7), zaregistruje se algoritmu. Ten se mu okamžitě pokusí najít plán skrze konfliktní body na jeho trase.



Obrázek 2.10: Konfliktní body na typické křižovatce (červené čáry). Na obrázku lze vidět i vzdálený konfliktní bod.

Samotný algoritmus

Pseudokód 2 popisuje základní chování algoritmu na vysoké úrovni. Nechceme čtenáře zatěžovat složitým kódem ve specifickém jazyku. Některé části nemusí být z tohoto zjednodušeného přepisu zřejmé, takže je dopodrobna vysvětlujeme ve vlastních sekcích.

Důležité je zmínit fakt, že oproti jiným řešením mají vozidla dovolené měnit rychlost v průběhu průjezdu skrz křižovatku. Přesněji jde o úseky mezi dvěma konfliktními body. Vždy ale mohou jen buď jednorázově zrychlit, nebo jednorázově zpomalit (a samozřejmě jet předtím a potom konstantní rychlostí). Oproti jednorázové změně rychlosti to pomáhá hlavně v situacích, kdy je na cestě úsek s omezenou maximální rychlostí – vozidlo se na zbylé trase pohybuje co nejrychleji.

Pokud algoritmus selže (pokusí se jít na předchozí bod z prvního bodu), vozidlo musí čekat (viz 2.5.2). V opačném případě se nalezená finální okna uloží na konfliktních bodech do plánovače.

Algoritmus 2 Hledání rezervace

```
1: Auto se zaregistruje křižovatce
2: Najdi konfliktní body na cestě auta křižovatkou
3: for Každý konfliktní bod (úsek před ním) v pořadí, jak je auto bude projíždět do
4:     if Další konfliktní bod neselhal then
5:         Najdi maximální rychlost ▷ Maximální rychlost může být omezena možnostmi auta nebo dalším úsekem silnice
6:         Zkus naplánovat co nejrychlejší projetí úseku
7:         Najdi okno, kdy vozidlo bude cestovat skrze konfliktní bod ▷ Okno je úsek daný v čase, který na konfliktním bodě má zarezervováno pouze jedno vozidlo
8:         if Okno nekoliduje s žádným existujícím then
9:             Úspěch, pokračuj na další konfliktní bod
10:        else
11:            Najdi okno pro konkrétní čas(Koncový čas okna, s kterým došlo ke kolizi)
12:            if Okno se podařilo nalézt then
13:                Pokračuj na další konfliktní bod
14:            else
15:                Jdi na předchozí bod se zjištěnou informací o nutném časovém posunu
16:            end if
17:        end if
18:    else
19:        Najdi okno pro konkrétní čas(Nový začátek po vyžadovaném časovém posunu)
20:        if Okno se podařilo nalézt then
21:            Pokračuj na další konfliktní bod
22:        else
23:            Jdi na předchozí bod se zjištěnou informací o nutném časovém posunu
24:        end if
25:    end if
26: end for
27: if Došlo k úspěchu u všech kontrolních bodů then
28:    Ulož nalezená okna do jednotlivých konfliktních bodů.
29:    Předej vozidlu pokyny k průjezdu
30: end if
```

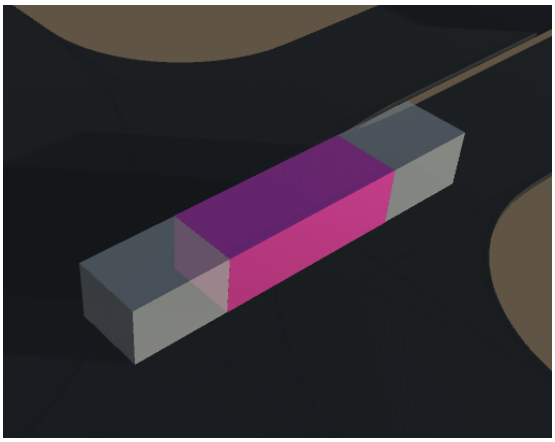
Algoritmus 3 Pomocná funkce

```
function Najdi okno pro konkrétní čas(Informace o čase)
    Spočítej okno, které začíná na daném čase s co nejvyšší konečnou rychlostí
    ▷ Jedná se o sadu matematických a logických funkcí, jejichž výsledkem je nové okno
    if Okno se nepodařilo spočítat then
        Vrať, o kolik času později by auto mělo odjíždět z předchozího konfliktního bodu
    else if Okno nekoliduje s žádným existujícím then
        Úspěch, vrať nalezené okno
    else
        Najdi okno pro konkrétní čas(Konec okna, s kterým došlo ke kolizi)
    end if
end function
```

Buffery

V mnoho existujících pracích (Khayatian a kol., 2020) se setkáme s pojmem *buffer*. Český překlad „vyrovnávací paměť“ by mohl být dosti zavádějící, proto zde ponecháváme původní slovíčko.

Při plánování se neuvažuje pouze bezrozměrná pozice bodu samotného, ale i blízké okolí. Rezervace na konfliktním bodu tedy není jen jeden bod v čase, ale celý úsek, který zde nazýváme *okno* – má začátek a konec. Velikost tohoto okna je určena vícero věcmi. V základu je to čas, kdy se vozidlo pohybuje přes konfliktní bod – tedy kdy se část vozidla nachází nad bodem.



Obrázek 2.11: Auto a jeho teoretický buffer velikosti 1

Toto je ale příliš úzký odhad, protože vozidla do sebe mohou narazit, i když se ani jedno nedotýká konfliktního bodu samotného. Proto bylo potřeba do simulace přidat umělé *buffery*, které rezervační okna zvětšují, aby zde byl dostatečný prostor navíc (jak křižovatka uvažuje o autu s jeho bufferem k náhlednutí na obrázku 2.11). Velikost bufferů se může lišit. Každý buffer má velikost alespoň *1*, neboť se touto hodnotou násobí celková délka vozidla a tu rozhodně nechceme nižší než násobek 1. V následujících hodnotách tuto jedničku odečítáme, aby bylo

vidět, jakou část okna *přidáváme*. Zde jsou velikosti bufferů pro různé konfliktní body:

- 1,5 pro vstupní konfliktní body, tedy ty, kde se cesta z pruhu rozděluje na vícero cest vedoucích do odchozích směrů.
- 0,5 až 2 pro konfliktní body uvnitř křižovatky, kde se cesty kříží. Výsledná hodnota lineárně závisí na úhlu křížících se cest. Pokud jsou cesty kolmé, nebo vedou podél sebe (úhel menší než 90 stupňů), hodnota se rovná 0,5. Pokud naopak jdou proti sobě (téměř opačným směrem), hodnota se blíží 2.
- 1,6 pro výstupní konfliktní body, tedy ty, kde se několik cest slučuje do jedné, která vede pryč z křižovatky v jednom pruhu.
- 0,5 pro vzdálené body za křižovatkou, kde už je pouze jeden pruh (jediná cesta vedoucí skrze bod).

Je vidět, že hodnoty nemusí být úplně dokonalé. Například buffery pro vstupní a výstupní body, kde se rozděluje/slučuje více cest hodně záleží na konkrétním vzhledu těchto cest. Také jsou buffery vždy oboustranné (čas kdy je vozidlo svým středem přesně na konfliktním bodu je přesně uprostřed rezervace), což může být právě v tomto případě na škodu. Pro stranu, kde se nachází několik rozdělených cest vedle sebe, je to potřeba, zatímco pro tu, kde jsou cesty sloučené (tedy je zde jen jedna cesta) by stačil mnohem menší buffer.

Algoritmus používá ještě jeden speciální, méně důležitý buffer, a to časový. Pro jasně nekolidující návaznost rezervací se mezi nimi vytváří drobná mezírka (v základu 0,01 s).

Pevné hodnoty lze nastavit v konfiguraci (A.2.7). Výsledná velikost bufferů záleží na hodně parametrech, které by bylo obtížné všechny spočítat v simulaci, a proto jsou nastaveny napevno. Hodnoty, které zde uvádíme, jsme vypozorovali po důkladném testování.

Hledání okna

V algoritmu popisujeme jakési magické „hledání okna pomocí matematických funkcí“ (2.32). Jedná se o vcelku prosté řešení, které ale funguje o dost efektivněji než pracné procházení všech možností.

Část, kde se počítá, jak vozidlo může chytře zrychlovat či zpomalovat rozebíráme později (2.5.2). Jde vlastně o nadstavbu nad onou „matematikou uvnitř“. Jak tedy tyto výpočty fungují?

Pojďme si nejprve připomenout kontext. Chceme spočítat cestu vozidla ke konfliktnímu bodu. Máme situaci, kdy známe čas, v kterém vozidlo vyjelo z předchozího bodu (respektive současný čas, pokud se jedná o první konfliktní bod). Víme, jakou rychlosí odtamtud auto vyjíždí. Známe vzdálenost k dalšímu konfliktnímu bodu a dostali jsme pokyn o čase, kdy má začít rezervace na současném konfliktním bodu (buď kvůli kolidující rezervaci, nebo díky dříve spočítané rezervaci a informaci z dalšího konfliktního bodu o nutném časovém posunu). Algoritmus si samozřejmě načte i základní hodnoty vozidla jako rozměry, zrychlení, brzdové zpomalení, apod. S těmito informacemi lze pomocí poměrně snadných

matematických úprav (dělení, násobení, kvadratické rovnice) spočítat, jak moc má vozidlo zrychlit/zpomalit, aby dojelo do dalšího bodu v přesně určený čas.

Řešení tohoto problému může existovat hned několik – kdyby auto mohlo na cestě libovolně zrychlovat a zpomalovat. Tento problém také řešíme více v další části (2.5.2), ale zde zmíníme základní fungování při hledání. Algoritmus počítá s tím, že změna rychlosti proběhne *co nejdříve* a zbylou část vozidlo cestuje konstantní rychlostí. Tedy například nejprve auto zrychlí a poté bude pokračovat konečnou rychlostí až do konfliktního bodu. Pokud má vozidlo měnit rychlost pouze jednou na začátku, existuje vždy maximálně jedno řešení. Snadno si rozmyslíme, že dvě rozdílná řešení by nutně vyústila v různé celkové časy. Pamatujme, že auta používají maximální hodnoty akcelerace/decelerace, aby změna rychlosti proběhla co nejdříve.

$$\frac{1}{2 * a}x^2 - (t + \frac{v_0}{a})x \ + \frac{v_0^2}{2 * a} + d \ = 0, \tag{2.1}$$

kde:

x ... cílová rychlost,
a ... zrychlení vozidla,
t ... čas, za který má vozidlo urazit vzdálenost,
v_0 ... počáteční rychlost vozidla,
d ... vzdálenost vozidla od vjezdu do konfliktního bodu.

Vzoreček 2.1 popisuje kvadratickou rovnici, která se používá pro výpočet cílové rychlosti při plánování každého okna. Vzdálenost ve vzorečku tedy odpovídá vzdálenosti vozidla od vjezdu do konfliktního bodu (kdy se buffer vozidla dotkne bodu), příp. vzdálenost průjezdu předchozím konfliktním bodem od tohoto místa. Algoritmus počítá podobný výpočet i pro případ zpomalení (vzoreček 2.2).

$$\frac{1}{2 * b}x^2 + (t - \frac{v_0}{b})x \ + \frac{v_0^2}{2 * b} - d \ = 0, \tag{2.2}$$

kde:

b ... brzdové zpomalení vozidla,
ostatní proměnné stejné jako u předchozího vzorečku.

Po výpočtech algoritmus zkontroluje, které výsledné hodnoty dávají smysl a podle toho určí, jestli se jedná o zrychlení, nebo zpomalení. Pro nalezení vzorců jsme pracovali se základními pravidly pro vzdálenost, lineární zrychlení, atd.:

$$v = v_0 + (a * t),$$
$$d = \frac{v + v_0}{2} * t.$$

Může se stát, že okno spočítat nejde (nezvladatelné požadavky). Pak samozřejmě výpočet nemá řešení a tato skutečnost se zjistí. Algoritmus spočítá čas, o kolik by potřeboval začínat později, a nahlásí ho předchozímu bodu. Případ,

kdy by vozidlo nebylo schopné v úseku dostatečně zrychlit a dostat se do bodu včas (který nejde vyřešit žádostí o pozdější začátek), nastat nemůže. Hledání okna pro konkrétní čas totiž probíhá vždy až potom, co už bylo nalezeno okno, které kolidovalo, nebo nekolidující okno, u kterého ale následující konfliktní bod zažádal o pozdější čas.

Zrychlování mezi konfliktními body

Průjezd samotným konfliktním bodem vždy probíhá konstantní rychlostí – část, kdy se auto (nebo jeho buffer) skutečně nachází na konfliktním bodu (tedy doba rezervace). Je to z toho důvodu, že kdyby auto měnilo rychlost tady, výpočet rezervace by byl mnohem složitější a není jasné, co by pak vlastně rezervační okno představovalo. Do programu jsme však přidali i drobné vylepšení. Pokud má vozidlo na dalším úseku zrychlovat, může zrychlení začít už v rámci (druhé poloviny) cesty skrze konfliktní bod. Rezervace pak zůstane stejná, ale vozidlo může konfliktní zónu opustit o něco rychleji. Proto také nehrozí, že by došlo k nepředvídatelné srážce – auto **dříve** zmizí z nebezpečné oblasti. Podobné vylepšení funguje i v případě, kdy auto do rezervace zpomaluje (tudíž do konfliktní zóny může přijet později). Při hledání okna pro konrétní čas se ale nepoužívá, protože výpočet počítá s tím, že v rámci rezervací na konfliktních bodech se auta pohybují konstantní rychlostí.

Při prvotním odhadu co nejlepšího průjezdu (2.6) — není zde časové omezení, snažíme se o co nejrychlejší průjezd — platí následující:

- Pokud je startovní rychlost vyšší než konečná maximální rychlost, vozidlo musí v úseku zpomalit. Ideální je zpomalit pouze na maximální rychlost. Aby průjezd byl co nejkratší, zpomalení proběhne co nejpozději.
- Pokud je konečná maximální rychlost vyšší než startovní rychlost, vozidlo se pokusí okamžitě zrychlit na co nejvyšší rychlost (ta stále nesmí překročit tu maximální). Zrychlování ale může probíhat jen před vjezdem do okna rezervace.
- Pokud jsou hodnoty rychlostí stejné, rychlost se nezmění.

Následující řádky budou rozebírat případy, kdy už došlo k nalezení výsledku (popsáno v 2.5.2), ale ten se ještě dá vylepšit. Kontext je tedy takový, že víme, že výpočet našel plán, jak auto může projet úsek za daných podmínek, akorát nepočítal s některými dalšími vlastnostmi či omezeními.

- Pokud jsou konfliktní body příliš blízko u sebe (rezervační okna se překrývají), nedochází ke změně rychlosti.
- Pokud je nalezená výsledná rychlost vyšší než maximální konečná rychlost, jedná se o situaci, kdy auto zpomaluje. Tento fakt nemusí být hned zřejmý. Jenže při hledání okna se pracuje s nějakým pozdějším časem, než který by byl dosažen co nejvyšším zrychlením. A jelikož odhad nejvyššího zrychlení s maximální rychlostí pracuje a nepřekračuje ji, existuje řešení, ve kterém ani tento pomalejší výsledek maximální rychlost nepřekročí. Auto tedy zpomaluje, jenom ne dostatečně. Řešením je, že zpomalovat musí začít

až později. Tudíž delší dobu stráví jízdou rychlejší rychlostí a musí zpomalit o to víc. Začne zpomalovat až v moment, kdy to přesně vyjde, aby při vstupu do prostoru konfliktního bodu mělo maximální rychlost. Opět o tom, že to určitě jde, svědčí fakt, že dříve už byl učiněn pokus o průjezd s nejvýše maximální rychlostí, akorát kolidoval s jiným oknem.

- Pokud má auto zrychlovat a nalezená výsledná rychlost je nižší než maximální rychlost, řeší se situace vlastně podobně jako v předchozím případě. Auto začne zrychlovat co nejpozději (aby nepřekročilo nejvyšší rychlost), tudíž stráví delší dobu pomalejší rychlostí a bude ke konci úseku muset zrychlit o to víc.

- Pokud je výsledná rychlost nižší než minimální povolená rychlost pro průjezd autonomní křižovatkou (více v 2.5.2), nelze takové okno potvrdit a algoritmus se musí vrátit na předchozí konfliktní bod, nebo nechat auto čekat (jedná-li se o první bod).

Speciální první konfliktní bod

Během testování jsme si všimli, že algoritmus funguje dobře – ke kolizím v křižovatce nedochází, ale vozidla do sebe na chvilku vjíždí před ní. Tento problém jsme přikládali povoleným kolizím při přejezdech mezi pruhy, ale po podrobnějším průzkumu se ukázalo, že se skutečně jedná o chybu plánování.

Problém byl v tom, že křižovatka si hlídala kolize pomocí konfliktních bodů – na nich a mezi nimi skutečně k nárazům nedocházelo. Jenže vůbec nezaručovala, že auta jedoucí za sebou si nenaplánují cestu „přes sebe“. Například by mohlo jedno vozidlo chtít na začátku jet pomalu a pak zrychlit na vysokou rychlost a vjet do křižovatky, zatímco auto jedoucí za ním by jelo střední rychlostí po celou dobu, tudíž by do prvního narazilo ve chvíli, kdy by bylo ještě pomalé (pak by se rozjelo a na konfliktních bodech už by vše probíhalo v pořádku).

Řešení jsme se rozhodli aplikovat pouze při plánování cesty do prvního konfliktního bodu, kde jsou ještě auta jen za sebou (nikde jinde ani k problému nedocházelo). Jak jsme psali o odstavec výše, úskalí tkvělo v tom, že některá vozidla zrychlovala až ke konci, zatímco jiná měla od začátku poměrně vysokou rychlost. Rozhodli jsme se to vyřešit tím, že *všichni* budou na cestě k prvnímu konfliktnímu bodu nejdříve zpomalovat a pak zrychlovat.

Tento přístup se může jevit jako pošetilý, ale pokud existuje (dříve popsaný) způsob, jak vozidlo může v daný čas dorazit do konfliktního bodu, často se dá najít plán, kdy hned na začátku vozidlo zpomalí na spočítanou rychlost a pak hned zrychlí a díky tomu projede bodem rychleji, než kdyby k tomuto procesu nedošlo. Opět je potřeba zapojit elementární výpočty rychlostí, časů apod. Ne vždy plán poskytuje dostatek času, aby k tomuto manévru došlo (pak se neaplikuje, ale probíhá kontrola popsaná v odstavci níže). Většinou má však vozidlo nějaký čas navíc, protože konfliktním bodem projede s větší mezerou než jeho předchůdce.

$$v_{tm} = \min\left(\sqrt{v_0^2 + 2d\ a}, v_{max}\right), \tag{2.3}$$

$$(a+b)x^2 - 2v_{tm}(a+b)x\ + b\ v_{tm}^2 + a\ \left(-v_0^2 + 2(v_{tm}v_0 + b\ (d - t\ v_{tm}))\right) = 0, \tag{2.4}$$

kde:

v_{tm} ... teoretická maximální rychlost,
v_0 ... počáteční rychlost vozidla,
d ... vzdálenost vozidla od vjezdu do konfliktního bodu,
a ... zrychlení vozidla,
v_{max} ... maximální povolená rychlost po průjezdu konfliktním bodem,
b ... brzdové zpomalení vozidla,
x ... rychlost, na kterou se zpomalí uprostřed,
t ... čas, za který má vozidlo urazit vzdálenost.

Celkový výpočet lze vidět v předchozích vzorečcích. Nejprve se spočítá maximální rychlost, kterou teoreticky vozidlo může na konci cesty ke konfliktnímu bodu dosáhnout (2.3). Pak se pomocí kvadratické rovnice (2.4) počítá, na jakou rychlost má vozidlo v průběhu zpomalit, aby následně zrychlilo na teoretickou maximální rychlost, kterou projede konfliktní bod v určený čas. Pokud je tato hodnota z výpočtu nižší než 0, nastaví se na 0, naplánuje se krátké zastavení a čekání do okamžiku, kdy naplánované zrychlení přesně časově vyjde.

Když si to shrneme, naplánovaný průběh vypadá následovně: auto začne okamžitě zpomalovat, dokud se nedostane na cílovou rychlost „uprostřed“. Pokud je tato nulová, může se stát, že auto chvilku čeká zastavené. Následně vozidlo začne zrychlovat na finální rychlost, která se rovná dříve spočítané teoretické maximální, případně co nejvyšší nižší, pokud vozidlo nestihne zrychlit na tu maximální. Pokud vozidlo na maximální rychlost stihne zrychlit, pokračuje jí zbytek cesty ke konfliktnímu bodu.

Další překážkou bylo to, že některá vozidla nemají prostor zpomalit a zase zrychlit – zkrátka potřebují většinu času jet co nejrychleji. Při plánování jejich cest je potřeba spočítat, zda nenarazí do předchozího vozidla v bodě, kde ještě nezrychlilo (resp. kde toto vozidlo jede ještě rychle). Pokud by mohlo v takovém místě dojít ke kolizi, plánované vozidlo automaticky nedostane rezervaci a musí čekat na další možnost.

Toto řešení se ukázalo jako funkční – ke kolizím už nedochází nikde (až na přejezdy mezi pruhy). Co nás překvapilo, byl fakt, že autonomní křižovatky díky tomuto vylepšení získají na propustnosti. Přestože nemůžou „podvádět“ a nechávat vozidla najíždět do sebe. Zlepšení přikládáme snaze o vyšší rychlosti skrze první konfliktní bod (a tudíž většinu dalších).

Obecně se jedná o další složitější část celé problematiky. V realitě by si nejspíš vozidla trasu a dodržování rezervací hlídala sama. V naší simulaci musíme každou takovou funkcionalitu sami zajistit a z designu algoritmu bylo mnohem snazší nechat tuto část na plánování.

Nastavení hodnot vozidlu

Komunikační protokol byl pro účely rychlého běhu simulace zjednodušen. Komunikace vozidla a křižovatky probíhá přímo pomocí funkcí v kódu. Neexistuje žádný speciální konstrukt pro zprávy mezi auty a křižovatkou.

Jakmile křižovatka najde možný průjezd pro vozidlo, pošle mu pro každý úsek mezi dvěma konfliktními body následující údaje:

- Jak dlouho má čekat před změnou rychlosti.
- Na jakou rychlost má zrychlit/zpomalit. Tento údaj není potřeba doplnit časovou hodnotou, neboť si ji křižovatka při plánování dokáže z údajů o zrychlení/zpomalení auta dopočítat. Vozidlo samotné také nepotřebuje znát dobu změny rychlosti – stačí mu vědět, v jaký okamžik ji má začít měnit.
- Jak dlouho má čekat po změně rychlosti.

Jedná se o seznam, který si autonomní řidič postupně prochází, tudíž ani nepotřebuje vědět, že časy souvisí s konfliktními body. Teoreticky by se navazující hodnoty čekání daly spojit, ale pro jednoduchost, srozumitelnost a rozšiřitelnost to bylo ponecháno takto.

Hodnoty jsou velmi specifické a fungují díky tomu, že v simulaci nedochází k odchylkám a je zde perfektní komunikace mezi vozidlem a křižovatkou. V realitě/důkladnější simulaci by bylo třeba na tyto fakty myslet a zabývat se jejich řešením.

Optimalizace minimální rychlosti

Výše jsme zmiňovali minimální povolenou rychlost v křižovatce. Bez té by algoritmus fungoval skvěle pro nižší počty vozidel. Jakmile se však křižovatka začne zahušťovat a přijíždí stále další a další, algoritmus jim vždy najde průjezd, ale kvůli existujícím rezervacím a bufferům se musí jednat o průjezd o něco pomalejší. To znamená, že vozidla stráví více času v křižovatce a zaberou delší rezervace v konfliktních bodech. Pokud stále přijíždějí nová, tak se celkově křižovatka stále zpomaluje a zpomaluje. Nakonec se může stát, že se systém zasekne, protože potřebné hodnoty jsou příliš nízké, než aby se s nimi dalo rozumně počítat v číslech s plovoucí desetinnou čárkou.

Tento problém jsme vyřešili jednoduchým trikem - zakázali jsme průjezdy, kde by kdykoli rychlost klesla pod minimální hodnotu nastavenou v konfiguraci (v základu 5 m/s). To má pochopitelně za následek, že autu, které přijede později, může být přiřazen průjezd dřív, než autu, které přijelo dříve. Také je potřeba vyřešit, co se tedy má stát s autem, kterému se nepodařilo najít rezervaci. O tom si povíme v následující sekci.

Čekací fronta

Vozidlo, kterému nebyla nalezena rezervace, se pustí čekání po dobu dle konfigurace (v základu 0,25 vteřiny). Po tuto dobu se auto chová podobně, jako kdyby mělo červenou na světelné křižovatce s tím rozdílem, že se zastaví dále od křižovatky, aby mělo dostatečný prostor na rozjezd. Po uplynutí čekací doby se algoritmus pokusí najít novou rezervaci. Pokud se to povede, je autu přiřazena, jinak se opět zapne stejné čekání.

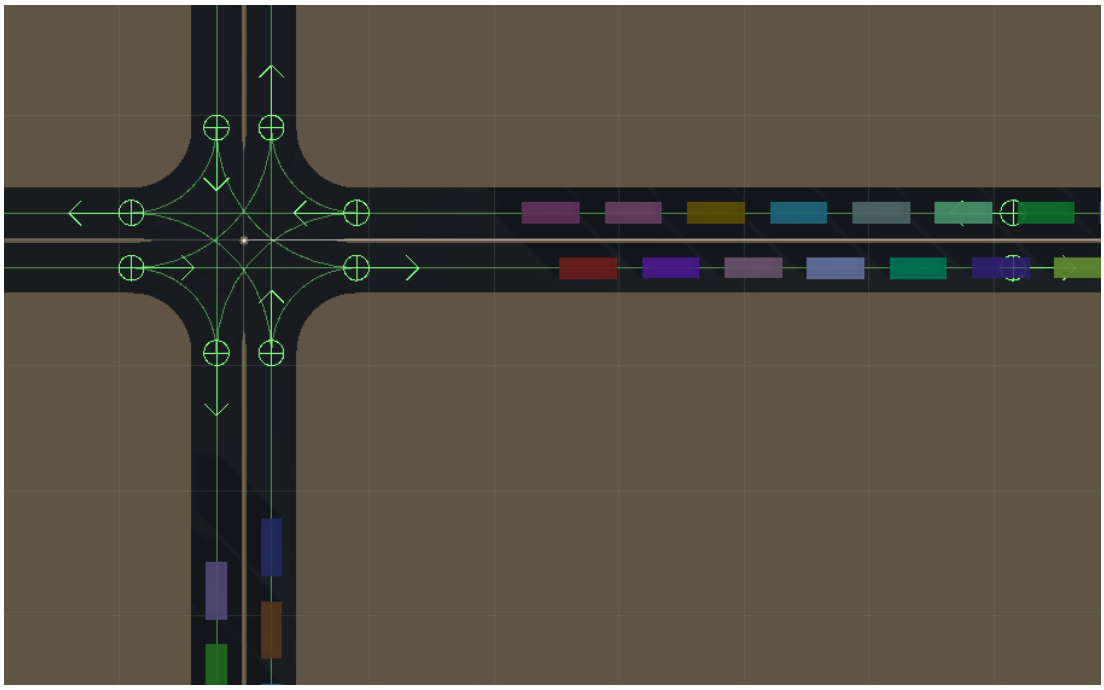
Samozřejmě se může stát, že do pruhu, kde se nachází čekající auto, vjedou další vozidla. Systém ví, že není možné jim najít rezervaci, proto se o to nepokouší a pouze je přidá do *fronty* za první auto. Tento krok spustí předčasný pokus o nalezení rezervací pro celou frontu. Pokud se to nepovede pro všechna z aut, pokračuje pro ta zbylá dříve spuštěný časovač. Vždy po uplynutí času se systém

pokusí postupně najít rezervace všem autům z fronty na daném pruhu. Pokud to u jednoho nevyjde, tak se opět pouští časovač jako u jednoho auta.

Výsledkem řešení je to, že často se rozjede celá fronta aut v pruhu za sebou. Tento přístup může porušovat férovost, ale zvyšuje celkovou efektivitu. Jedná se tedy o jakousi heuristiku (1.1.4).

Je tu ještě jedna možnost, kdy auto *nikdy* nemůže vjet do křižovatky, a proto zůstává také ve frontě. Jsou to případy, kdy je odjezdový pruh plný. Auto v takovou chvíli nepřeplánovává svou trasu (v realitě by mohlo), ale prostě čeká, dokud se místo neuvolní. Tato situace může být způsobena tím, že křižovatka sousedí s další, která je hodně vytížená, nebo je na ní použit neefektivní algoritmus.

Ve výsledku v základní podobě simulace obě „fronty“ končí zhruba ve třetině příjezdového či odjezdového úseku – jako je to vidět na obrázku 2.12. Velice jednoduchým vylepšením pro zahlcené systémy, který jsme ale v simulaci nevyužili (a v realitě by jen tak také nešel využít), je prodloužit úseky mezi křižovatkami – především ty odchozí. Pak vznikne prostor pro víc aut a křižovatka jich může víc naplánovat a nemusí čekat, aby se ujistila, že se vejdou.



Obrázek 2.12: Fronta před a za AIM křižovatkou

3. Experimenty

Tato kapitola by měla být tou nejdůležitější v práci, neboť přináší výsledky z experimentů prováděných ve vyhotovené simulaci.

Podíváme se na to, jak efektivně fungují jednotlivé algoritmy i na to, jak co nejlépe zapojovat více algoritmů vedle sebe.

Vzhledem k množství experimentů jsme jich většinu prováděli pouze jednou. V rámci vnitřního testování jsme si ověřili, že při použití stejných seedů pro generování provozu dostáváme výsledky s minimální odchylkou. Pokud se seed změní, může to lehce ovlivnit i výsledná data, ale i zde jsme si ověřili, že celkový vliv je zanedbatelný. Navíc většinu experimentů budeme pouštět s tím samým seedem a jen budeme porovnávat různé výběry algoritmů křižovatek. Proč stejné seedy nedají stejné výsledky a další omezení popisujeme v části 3.3.

Budeme-li mluvit o čase, budeme tím myslet čas uběhlý v rámci simulace (kterou jsme běžně pouštěli zrychleně).

3.1 Data ve výsledcích

V následujících experimentech se budou vyskytovat různá data ve sloupečcích výsledků. Ne v každé tabulce budou zahrnuta všechna – často budeme zkoumat rozdílné vlastnosti v závislosti na tom, co nás bude v daném experimentu zajímat. Pro přehlednost se však hodí mít všechny výstupní hodnoty popsané na jednom místě, a proto zde předkládáme seznam všech možných:

- **Počet vozidel** značí počet aut, který byl nastaven před během simulace. Této hodnoty simulace nemusela dosáhnout, pokud se vozidla do systému nevešla, ale garantovaně nikdy v simulaci nemůže v jednu chvíli být více aut, než je tento počet.
- **Algoritmus** vypovídá o zvoleném systému řízení křižovatky (jedná-li se o experiment s pouze jednou křižovatkou). „Stopky“ vyjadřují algoritmus typu *Stop Sign* (2.5.1), „Semafory“ algoritmus typu *Traffic Light* (2.5.1) a „AIM“ autonomní algoritmus (2.5.2).
- **Průjezdy** udávají počet, kolikrát vyjelo nové auto do systému. Většinu této hodnoty tvoří vozidla, která svou cestu i dokončila, ale číslo zahrnuje i ty, které konec simulace zachytila někde „za běhu“.
- **Celková ujetá vzdálenost** odpovídá součtu všech vzdáleností, které vozidla v rámci běhu simulace urazila.
- **Průměrná rychlost** se vypočítá z celkové ujeté vzdálenosti vydělené celkovým časem, který všechna vozidla strávila v simulaci. Do výsledku je tedy zahrnut i čas, který vozidla vyplnila čekáním.
- **Maximální hustota**. Každý běh má nastavený počet vozidel. Simulace se pokouší přidávat do systému další a další auta, dokud to jde, nebo dokud nedosáhne této hodnoty. Pro první případ, kdy se právě nepodaří toto číslo

„naplnit“ měříme *maximální hustotu*, tedy kolik aut se v systému maximálně naráz pohybovalo. Hustota se měří i v průběhu simulace (v základu každou minutu).

- **Celkový čas** popisuje součet všech časů, které auta strávila buď v systému, nebo na jednotlivých křižovatkách – vždy vysvětleno v poznámce.
- **Ideální čas** vypovídá o době, kterou vozidla mohla trávit průjezdem. Počítá se jako součet časů na jednotlivých úsecích, pokud by se vozidlo pohybovalo maximální rychlostí úseku. Jedná se o velice optimistický dolní odhad. Opět bude typicky popsáno v poznámce, o jaký případ konkrétně jde (souučet, průměrná hodnota, ...). Měří se pro křižovatky samostatně.
- **Zpoždění**. Zpožděním se myslí doba, o kolik vteřin byl skutečný průjezd horší než „ideální“ průjezd. Typicky budeme představovat průměrné zpoždění. Důležité je porovnávat zpoždění jednotlivých algoritmů mezi sebou, protože dosáhnout nulového zpoždění je téměř nemožné. Měří se pro křižovatky samostatně.
- **Čekání** představuje dobu, kterou vozidlo strávilo stáním na místě bez pohybu. Měří se na úrovni celého systému, tedy pro každý průjezd (ne pro jednotlivé křižovatky).

V tabulkách jsou čísla většinou zaokrouhlená na 2 desetinná místa, což považujeme za dostatečnou přesnost v případě jednotek metrů, sekund i metrů za sekundu.

Data použitá v práci jsou dostupná v rámci přílohy. Ne vždy se práce odkazuje na všechny parametry, pokud nejsou zajímavé, ale samotné soubory s výsledky obsahují veškeré zaznamenané hodnoty. Jen poznamenáváme, že při testování se ukázalo, že by se mohlo hodit, kdyby data obsahovala i nějaké celkové výsledky navíc, takže jsme pak výstup o tyto části rozšířily, ale starší experimenty nutně neupravovali. Hodnoty, z kterých se tato nová čísla spočítala, jsou však vždy přítomny, a tedy jde výstup navíc teoreticky dopočítat.

3.2 Základní parametry při testování

Pokud není uvedeno jinak, byly veškeré experimenty spouštěny po dobu 1 simulační hodiny.

Simulace se stejnou mapou byly pouštěné se stejným seedem pro provoz. To znamená, že posloupnost naplánovaných cest bude stejná.

Data výsledků, ze kterých jsme čerpali, jde nalézt v příloze – včetně vstupního souboru pro uživatelské nastavení. V souboru se objevuje hodně hodnot a ne vždycky jsou všechny v běhu simulace využité. Podstatné hodnoty jsou však vždy zadané ve vstupech po načtení nastavení (viz A.2.3).

3.3 Limitace simulace

Při testování jsme pochopitelně (asi jako s každým počítačovým programem) narazili na nějaké překážky. Pokud šlo o věci snadno opravitelné zásahem do

kódu, tak jsme opravy učinili. Některé skutečnosti ale lehce vyřešit nešli, proto je zmiňujeme tady. Stejně tak v několika případech jsme už měli naměřená data a uvědomili jsme si, že pro další experimenty by se hodilo tato data nějak rozšířit nebo více zpracovat. To jsme také učinili, ale soubory s hodnotami prvních experimentů to nemusí reflektovat, pokud nová data nebyla potřeba.

Pochopitelně je simulace stále jen nedokonalý počítačový program. Byť jsme se veškerými silami pokusili ji připravit tak, aby ke kolizím nedocházelo, může nastat nějaká zákeřná nepředvídatelná situace a ke kolizi dojde. Nejpravděpodobnějším důvodem je nesprávně (ručně) odhadnutá bezpečnostní hodnota nastavená v konfiguraci. Zvýšení těchto hodnot by však znamenalo zpomalení celého systému. Proto jsme do výstupu umístili počet kolizí, který za běhu nastal. Tuto hodnotu jsme sledovali – typicky zůstávala na 0 nebo se pohybovala v nízkých jednotkách, což považujeme za výsledek nedokonalosti systému. Pokud by však došlo k většímu počtu kolizí, určitě to tu zmíníme. Na samotný běh simulace kolize nemají vliv – systém kolizi pouze zaznamená, ale vyignoruje ji, jakoby auta projela skrz sebe.

Poslední nevýhodou je nedeterminismus Unity. Jedná se o diskrétní simulaci. Hodnoty časů, po kterých se mají odehrávat aktualizace, jsme sice nastavili fixně, ale do systému se mohly dostat nepřesnosti způsobené tím, že skripty jednotlivých objektů se pouštějí v předem nespecifikovaném pořadí. To znamená, že se například může lišit, které vozidlo vjede první do prostoru křižovatky a tím se změní průběh. Důsledkem celé záležitosti je to, že pustíme-li ten samý experiment dvakrát po sobě, budou výsledná data velice podobná, ale ne úplně stejná. Rozhodli jsme se tuto nedokonalost přikládat standardní nespolehlivosti měření.

Také si uvědomujeme, že napojení pruhů v křižovatkách nemusí být dokonalé. Například v křižovatce, kde jsou ve všech směrech 3 pruhy, zatáčí doleva i doprava pouze 1 pruh, zatímco rovně pokračují všechny 3. To může způsobit, že krajní pruhy jsou vytíženější než ten střední, a tím i klesne propustnost. Tento problém se však týká všech algoritmů, takže ho nepovažujeme za překážku při porovnávání.

3.4 Jedna křižovatka

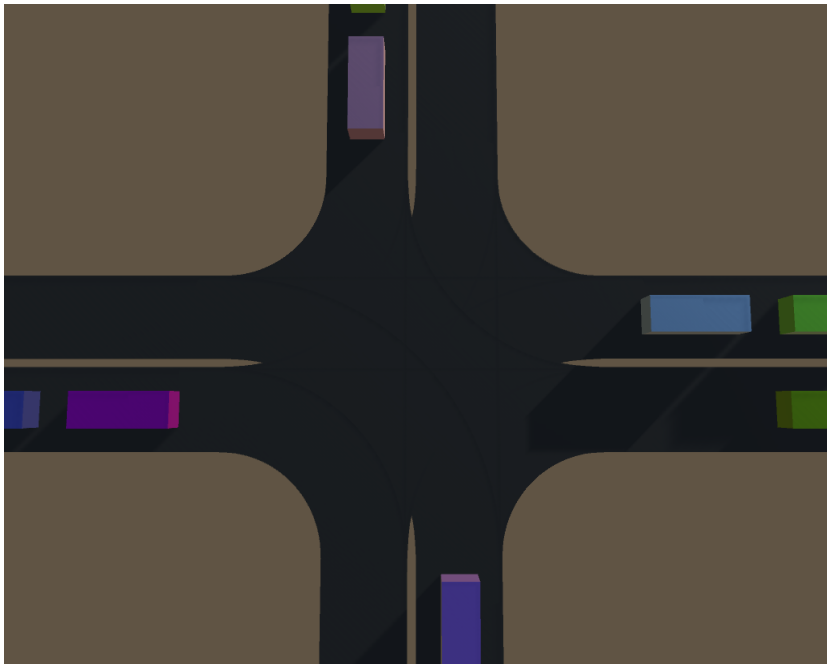
Bude následovat popis experimentů prováděných pouze na jedné křižovatce. Tématem práce je především porovnat různá nasazení autonomních křižovatek ve větším propojeném systému, ale je důležité si nejdříve otestovat jednotlivé algoritmy na nejjednodušším příkladu a ujistit se, že odpovídají základním očekáváním.

Budeme zkoumat křižovatky se všemi 4 směry s 1, 2, a 3 vstupními i výstupními pruhy. Pro představu na obrázku 3.1 ukazujeme verzi s 1 pruhem.

3.4.1 Malá křižovatka

Tabulka 3.1 ukazuje výsledky pro nízké počty vozidel na křižovatce s 1 vstupním a 1 výstupním pruhem ve všech směrech.

V horní části vidíme, že při opravdu nízké zátěži jsou semafory spíše přítěží. AIM takřka neztrácí žádný zbytečný čas. Stopka funguje podobně, ale když si pustíme průběh simulace, tak se několik aut na křižovatce potká na začátku (proto o malinko méně průjezdů) a následně se víceméně střídají. Toto chování



Obrázek 3.1: Základní křižovatka

Počet vozidel	Algoritmus	Průjezdy	Celková ujetá vzdálenost (m)	Průměrná rychlost (m/s)
5	Stopky	1143	237717,8	13,22
5	Semafory	657	136394,4	7,58
5	AIM	1145	238018,8	13,24
10	Stopky	1272	263924,2	7,34
10	Semafory	1270	263691,3	7,33
10	AIM	2285	475203,4	13,23
20	Stopky	1281	264542,3	3,68
20	Semafory	2481	514307,7	7,16
20	AIM	4526	941053,6	13,11

Tabulka 3.1: Malé počty vozidel na malé křižovatce

úplně neodpovídá realitě, kde typicky auto zastaví (nebo zpomalí), i když je na křižovatce samo. Napodobit zde přesně skutečné chování by bylo složité, proto budeme pokračovat s tímto „efektivním“ modelem.

Při 10 vozidlech už se stopky potkávají hodnotami se semafory. Autonomním křižovatkám se daří v podstatě stejně jako při 5 autech (zhruba dvojnásobné hodnoty). Na 20 vozidlech vidíme, že pro stopky je zhruba 1300 průjezdů za hodinu maximum. Naopak když jednotlivě zanalyzujeme AIM a semafory, uvidíme, že jejich propustnost v podstatě lineárně roste.

Pro zajímavost zde zmíníme ještě hodnoty celkového čekání, které jsme v tabulce neuváděli. Na semaforech a stopkách jsou to očekávatelné hodnoty, které se pohybují kolem tisíců či dokonce nižších desetitisíců (při větší hustotě provozu) vteřin. Náš autonomní algoritmus se však v pokusech s 5 vozidly drží na čisté 0, u 10 vozidel téměř také – 0,28 vteřin celkového čekání. U 20 vozidel dosáhlo celkové čekání zhruba 13 vteřin, což je vzhledem k celkovému naježděnému času

71768 vteřin (na 941054 metrech) úctyhodný výkon.

Hodnoty počtu průjezdů a celkové ujeté vzdálenosti zde velice úzce korelovali, neboť jsou všechny průjezdy zhruba stejně dlouhé. V experimentech pro pouze 1 křižovatku tedy od této chvíle uvádíme pouze 1 z těchto hodnot. U pozdějších experimentů se však budou tyto hodnoty potenciálně dost lišit, takže určitě rozdělení opět zavedeme.

V následujících experimentech se pokusíme zvýšit počet vozidel na maximum – stále na té stejné malé křižovatce s 1 pruhem. Výsledky zachycuje tabulka 3.2.

Počet vozidel	**Algoritmus**	**Průjezdy**	**Max. hustota**[a]	**Čas**[b] **(s)**	**Ideální čas**[b] **(s)**
35	Stopky	1295	35	102701,7	10340,33
35	Semafory	3609	35	91319,5	29338,64
35	AIM	6792	35	73565,9	55461,35
50	Semafory	4043	50	118376,9	32765,91
50	AIM	6709	50	96726,9	54657,83
75	Semafory	4035	66	126187,3	32626,53
75	AIM	6732	63	98966,5	54780,82

Pozn: [a] Maximální počet aut, který se najednou pohyboval v simulaci

Pozn: [b] Jedná se o celkový (ideální) čas všech aut na území křižovatky

Tabulka 3.2: Testování zátěže malé křižovatky

Vidíme, že stopky se stále drží na zhruba 1300 průjezdech. Zbylá auta tráví čas čekáním a systém se nezlepšuje, proto už pro vyšší počty výsledky ani neuvádíme.

Zajímavější je sledovat vývoj autonomních křižovatek a semaforů. Vidíme, že u nich se efektivní kapacita zasekla zhruba na 50 (u AIMu dokonce na 35) vozidlech – každý z algoritmů se svou vlastní propustností. Vícero aut výsledky v zásadě nezlepšuje. Žádnému z algoritmů se nepovedlo prostor křižovatky a sousedící silnice naplnit více než 66 vozidly. Nemá tedy smysl pokračovat s vyššími počty.

Tentokrát místo průměrné rychlosti uvádíme informace o čase. Nejpravější sloupeček udává celkovou hodnotu, za kterou vozidla, která projela křižovatkou mohla tuto cestu dokončit. Jedná se pochopitelně o nedosažitelnou hodnotu, jak jsme si popisovali výše. Zajímavé je však porovnání se sousedícím sloupcem, kde je součet časů, které vozidla skutečně strávila cestováním skrz prostor křižovatky (místo, kde křižovatky ví o vozidlech – stejné pro všechny algoritmy). Zde AIM jasně dominuje, neboť je nejvíce ani ne dvakrát horší než ideální čas. Semafory se drží zhruba na necelém čtyřnásobku.

V dalších experimentech se budeme více zaměřovat právě na 2 podstatné hodnoty: propustnost křižovatky (počet průjezdů za čas) a průměrné zpoždění (skutečný čas oproti ideálnímu).

3.4.2 Větší křižovatka

Nyní provedeme velice podobné experimenty, ale na křižovatce, která má o 1 pruh v každém směru navíc. To znamená, že do křižovatky ve všech 4 směrech přichází i odchází 2 pruhy. Tabulka 3.3 ukazuje naměřené hodnoty.

Počet vozidel[a]	Algoritmus	Průjezdy[b]	Průměrné zpoždění (s)[c]	Průměrné čekání (s)
20 (20)	Stopky	1137	47,39	41,53
20 (20)	Semafory	2587	12,28	10,12
20 (20)	AIM	4580	0,28	0
50 (50)	Semafory	5818	14,79	12,09
50 (50)	AIM	10523	1,5	0,17
75 (75)	Semafory	7167	18,48	17,07
75 (75)	AIM	11886	4,57	2,83
100 (100)	Semafory	7492	22,11	25,15
100 (100)	AIM	11812	5,89	6,24
130 (150)	Semafory	7438	24,02	32,28
114 (150)	AIM	12031	6,14	6,32

Pozn: [a] Maximální dosažený počet vozidel, v závorce nastavený počet vozidel

Pozn: [b] Tentokrát pouze dokončené průjezdy skrz prostor křižovatky

Pozn: [c] Zpoždění oproti „ideálnímu“ času

Tabulka 3.3: Středně velká křižovatka

Stopky se opět zasekly na ani ne 1200 průjezdech (což dává smysl – nikdy u nich nesmí být více než 1 auto v křižovatce a navíc je teď křižovatka o něco větší), a proto je u vyšších počtů vozidel už neuvádíme.

Upozorňujeme, že zatímco čísla průjezdů a průměrného zpoždění se tentokrát týkají pouze křižovatky samotné (dokončené průjezdy skrz prostor křižovatky a zpoždění průjezdu skrz prostor křižovatky), doba průměrného čekání se vztahuje na celou simulaci, tedy je do ní zahrnut i čas, který vozidlo trávilo ve frontě před křižovatkou, ještě než došlo k registraci. V systémech s vícero křižovatkami nám už tato hodnota nepomůže s analýzou jednotlivých křižovatek, ale nyní ji ještě můžeme využít – především v porovnání se zpožděním.

Je dobré zmínit, že autonomní křižovatky užitečnost hodnoty čekání trochu „obcházejí“, protože se stává, že vozidla před rozjezdem uvnitř křižovatky jedou velice pomalu a čekají na moment, kdy mohou zrychlit. Nestojí u toho ale vždy na místě, tudíž se může stát, že se jim nezapočítá žádný čas čekání. V datech vidíme, že čekání tvoří velkou část zpoždění právě u semaforů a stopek. Je to dáno tím, že vozidla při použití těchto algoritmů jsou v jednom ze dvou módů: bez omezení jedou, nebo stojí. V autonomní křižovatce se setkáváme se situacemi, kdy vozidlo zpomalí během průjezdu nebo obecně neprojíždí maximální rychlostí.

Vidíme, že náskok autonomních křižovatek se s přibývajícím počtem aut snižuje (poměrově). Na hustotě kolem 125 vozidel se však zaseknou oba algoritmy a můžeme si všimnout, že zde v podstatě ani nedošlo ke zlepšení propustnosti oproti 100 vozidlům.

3.4.3 Největší křižovatka

Ještě pro úplnost přidáváme data z největší možné křižovatky – 3 pruhy v každém ze 4 (8) různých směrů. Výsledky nalezneme v tabulce 3.4. Typy sloupečků jsme ponechali stejné jako v předchozím případě.

Počet vozidel[a]	Algoritmus	Průjezdy[b]	Průměrné zpoždění (s)	Průměrné čekání (s)
20 (20)	Semafory	2653	11,66	9,58
20 (20)	AIM	4562	0,38	0
50 (50)	Semafory	6124	13,65	10,99
50 (50)	AIM	10770	1,22	0,09
100 (100)	Semafory	8384	20,4	21,46
100 (100)	AIM	13704	5,91	5,54
150 (150)	Semafory	8619	25,66	36,94
150 (150)	AIM	13923	7,22	10,48
172 (200)	Semafory	8535	26,06	38,63
154 (200)	AIM	14108	7,35	10,46

Pozn: [a] Maximální dosažený počet vozidel, v závorce nastavený počet vozidel
Pozn: [b] Dokončené průjezdy skrz prostor křižovatky

Tabulka 3.4: Největší křižovatka

Můžeme vidět, že autonomní křižovatka exceluje při nižších zátěžích. Sice nad semafory vede i při 100 vozidlech, ale zlepšila se průjezdy oproti 50 vozidlům jen 1,27krát, zatímco semafory si polepšily 1,37krát. Což není až takový rozdíl, ale vyplývá nám z toho stále informace, že semafory samy sebe dokáží o něco lépe zlepšovat při rostoucím počtu vozidel než AIM. Stejně tak celkovou maximální kapacitou semafory vedou – zde se jim povedlo dosáhnout až 172 vozidel naráz, zatímco autonomní křižovatky se zasekly na 154. Semafory jsou schopné zaplnit systém trošku víc, což je dané tím, že se u nich fronty tvoří na větší ploše – u autonomních křižovatek auta zastavují už kus předem, aby měla prostor na rozjezd.

Průměrným zpožděním však AIM stále dominuje nad semafory. Všimněme si také, že semafory dokonce od 100 vozidel (a AIM od 150) převyšují zpoždění časem čekání. Je to dané skutečností, kterou jsme popisovali výše – auta čekají frontu, která se táhne i mimo prostor křižovatky. Na území křižovatky už tolik času nestráví, a proto je zpoždění skrz ni nižší než celkové čekání.

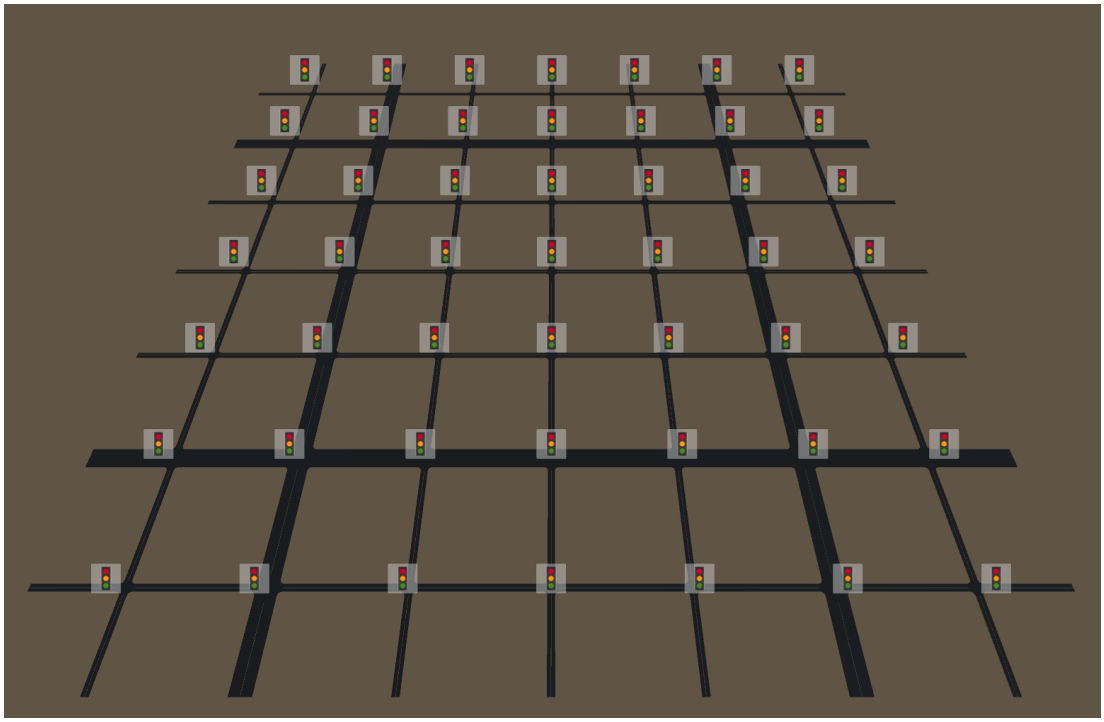
3.4.4 Závěry z měření jedné křižovatky

Na jedné křižovatce jsme si položili pár základních kamenů k dalším měřením – nejvíce nás bude zajímat propustnost a průměrné zpoždění. Zároveň jsme objevili určité limity kapacity, ale ty budou ve městě přísnější, protože čekání na jedné křižovatce bude moct zaseknout ty další.

Co se týče naměřených statistik, autonomní řešení dopadlo vždy nejlépe. Na těch úplně nejmenších zátěžích s ním soupeřily stopky, které tam poskytují nižší zpoždění než čekání na semafor. Na vyšších počtech vozidel AIM stále vede, ale rozdíl oproti semaforům už není tak markantní. Také jeho maximální potenciální kapacita je o něco nižší než u světelných křižovatek.

3.5 Standardní město

Pod pojmem *standardní město* rozumíme vygenerované prostředí, které bude sloužit jako základ pro všechny následující experimenty. Snahou bylo co nejvíce se přiblížit kusu nějakého skutečného města (ale stále dost „obecného“). Nerozebírejme však pouze hodnoty z uživatelského rozhraní, ale pojďme si vysvětlit, jak ten systém v praxi vypadá. Pro začátek si ho můžeme prohlédnout na obrázku 3.2.



Obrázek 3.2: Standardní město

Základ standardního města tvoří 4 pravidelně rozmístěné obousměrné hlavní silnice. Zbytek mřížky je doplněn vedlejšími silnicemi. Dohromady silnice dávají mapu 7x7 křižovatek.

Všechny experimenty budou používat ty stejné *seedy*. To znamená, že simulace se pokusí pouštět vozidla do města ve stejných místech se stejnými cestami. Také, pokud bude potřeba generovat část mapy náhodně, bude pro více experimentů použit ten stejný seed, aby město vypadalo stejně.

Co se týče distribuce spawnování, byly nastaveny hodnoty, aby se půlka vozidel objevovala na vjezdech hlavních silnic – a 80 % z nich končilo ve výjezdu také hlavní silnice. Z druhé půlky, která bude začínat na vjezdech vedlejších silnic, 60 % pojede do hlavního výjezdu. Zbylých 20 % či 40 % tedy pochopitelně směřuje do výjezdu vedlejší silnice. Očekáváme proto, že na hlavních silnicích se bude odehrávat větší část provozu.

Pro začátek jsme město vybavili pouze semafory, což je bod, od kterého se budeme chtít odpíchnout a získat základní měření. Tabulka 3.5 ukazuje řadu naměřených hodnot, které si v dalších odstavcích dopodrobna rozebereme.

Nejprve si pro jistotu popišme jednotlivé sloupce. *Počet vozidel* opět udává nastavený maximální počet aut, které se mohou ve městě pohybovat. Maximální dosaženou hustotu tentokrát neuvádíme, protože simulace vždy dosáhly nastavené hodnoty. Toto téma je však zajímavé a za chvilku se k němu vrátíme. *Po-*

Počet vozidel	Poznámka	Celková ujetá vzdálenost (m)	Průměrné zpoždění (s)	Průměrný čas (s)
250		5854321	92,83	165,10
250	2 hodiny	11724260	94,18	167,54
250	4 hodiny	23499650	94,38	168,14
250	jenom stopky	4615021	136,68	208,67
250	většina stopky	4855184	126,72	198,92
250	chytré stopky	7333096	60,04	132,92
250	synchronizace	3799890	180,26	251,78
500		11384240	96,98	169,08
500	chytré stopky	5543634	268,20	338,47
750		16561800	101,74	173,74
750	4 hodiny	66766220	103,76	177,63
1000		20682350	113,11	185,13
1000	2 hodiny	40418510	120,10	193,27
1000	4 hodiny	16150450	871,28	942,41
1250		20053740	163,74	234,76

Tabulka 3.5: Standardní město se semafory

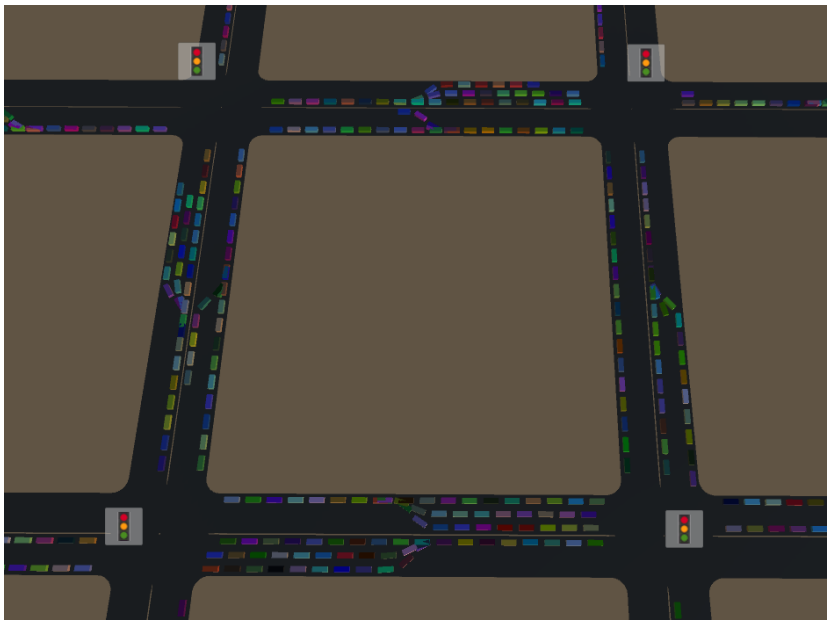
známka doplňuje, jestli byl experiment něčím speciální – na ni se odkazujeme v nadcházejícím textu. *Celková ujetá vzdálenost* bude naší měřenou hodnotou pro propustnost systému. Na počet průjezdů se totiž nelze tak dobře spolehnout, protože můžou být dost různě dlouhé (ale po zprůměrování by nejspíš šlo také o poměrně kvalitní metriku). *Průměrné zpoždění* udává rozdíl času průjezdu oproti ideálnímu času. Je potřeba si uvědomit, že ideální čas udává nejoptimističtější odhad – předpokládá, že vozidlo celou trasu jede maximální rychlostí. Proto kromě zpoždění uvádíme i *Průměrný čas*, který popisuje, jak dlouho pro představu průměrný průjezd trval. Zpoždění tedy v nadcházejících experimentech budeme muset porovnávat hlavně mezi sebou.

Pokud to není uvedeno v poznámce, experiment trval 1 hodinu. U některých hodnot jsme jej pustili i s delšími časy, abychom zjistili, jestli se po delší době něco nezmění. Vidíme, že při 250 a 750 vozidlech zůstávají hodnoty při změnách doby běhu podobné (ujetá vzdálenost jsou násobky, zpoždění se minimálně liší). 1000 vozidel už však stojí na hraně. Při 1 či 2 hodinách funguje vše v pořádku. Ale vidíme, že po přepnutí na 4 hodiny se dokonce propustnost snížila oproti jedné hodině, protože došlo k zaseknutí.

Zaseknutí znamená, že se ve městě nacházelo tolik aut, že si navzájem zablkovala výjezdy (příklad lze vidět na obrázku 3.3). Někde se utvořil cyklus, který už nebylo možno rozpojit. Vozidla na sebe vzájemně čekají a nikdo se nehýbe. Dá se očekávat, že když se do systému pokusíme nacpat příliš mnoho agentů, na něčem to selže. Výsledkem je právě zaseknutí. Částečně je způsobeno specifickým vzhledem a nastavením naší simulace. V reálném světě bychom se možná setkali s podobnými zátěžemi, ale ty by pokrývaly jen hlavní cesty a vytížené křižovatky by mezi sebou měly více prostoru. Případně by zde existovala speciální cesta (magistrála/dálnice/...), kde je povolená vyšší rychlost.

Zaseknutí z dat odhalíme díky měřeným dokončeným průjezdům za minutu,

dojde při něm k výraznému poklesu a následně se toto číslo zastaví na 0. Na 1000 vozidlech dochází k zaseknutí, necháme-li simulaci déle běžet. V rámci testování jsme zkoušeli i další podobné zátěže (1050, 1075, 1100, . . .), ale většinou se zasekly ještě dřív. Zajímavé bylo, že například při 1250 vozidlech k zaseknutí došlo až poměrně pozdě – což uvádíme v tabulce, ale stále jde z průměrů zpoždění vyčíst výsledek. Máme tedy horní limit pro testování hustoty křižovatky – 1000 vozidel. Běžně ale budeme testovat pro „malou“, „střední“ a „velkou“ zátěž, tedy s 250, 500, a 750 vozidly.



Obrázek 3.3: Příklad zaseknutí – cyklus ve čtverci vozidel, které chtějí zahnout doleva (a doprava v opačném směru)

Při měření malého počtu aut jsme si všimli, že část křižovatek zůstává většinu času poměrně prázdná. Napadlo nás tedy zkusit do standardního města k semaforům zasadit i nějaké stopky – jak víme z experimentů na jedné křižovatce, dokážou být pro malé počty efektivnější. Výsledky vyčteme také z tabulky 3.5 u 250 vozidel. „Jenom stopky“ znamenalo, že stopky byly přítomné místo semaforů úplně všude. „Většina stopky“ znamenalo, že semafory byly nasazeny jen na 4 hlavních uzlech velkých silnic. Nakonec „chytré stopky“ představovaly model, kde veškeré křižovatky na hlavních silnicích byly světelné a pouze na kříženích vedlejších silnic jsme nasadili stopky. Prvním dvěma alternativám se moc nedařilo, ale právě poslední nastavení stopek dokonce předčilo základ se semafory všude. Zároveň tím odpovídá i reálnému světu, kde typicky jen zatíženější křižovatky bývají pokryté semafory.

Zkusili jsme toto nejlepší rozmístění použít i při větší zátěži (500 vozidel), ale zde už bohužel došlo k výraznému poklesu efektivity. Proto, budeme-li tyto experimenty porovnávat s dalšími algoritmy, budeme pro 250 vozidel používat nastavení s „chytrými stopkami“, ale pro zbytek použijeme výsledky z celosemaforových měst.

Ještě bychom chtěli vysvětlit význam řádku s poznámkou „synchronizace“. Jedná se o výsledek staršího experimentu dřívější verze simulace, kde místo náhodných intervalů mezi spuštěním jednotlivých semaforových algoritmů a místo

náhodných počátečních směrů začly všechny křižovatky se zelenou fází ve stejnou chvíli pro stejný směr. Tento přístup dosáhl mnohem nižší efektivity.

3.6 Plošný AIM

V testech jedné křižovatky jsme zjistili, že ve všech scénářích autonomní křižovatky vítězí nad semafory. Nyní by nás zajímalo, jak se AIMu daří na větším městě – začneme tím, že ho nastavíme úplně všude. Následně se podíváme na odlišné možnosti rozdělení města na různé algoritmy, ale nejdříve musíme změřit teoreticky nejlepší výsledky. To nám vytvoří rámec, v kterém se nejspíš budeme pohybovat.

Naměřené výsledky pro základní zátěže si můžeme prohlédnout v tabulce 3.6. Druhý a třetí sloupec uvádí celkovou ujetou vzdálenost a průměrné zpoždění. Na první pohled tyto hodnoty - obzvlášť zpoždění, když si připomeneme, jak „drsný“ je odhad ideálního času, působí velice dobře. Zajímavější jsou však poslední dva sloupce, které uvádějí zlepšení v příslušných kategoriích oproti dříve naměřeným hodnotám semaforových křižovatek.

Počet vozidel	**Celková ujetá vzdálenost (m)**	**Průměrné zpoždění (s)**	**Zlepšení vzd. (m)**	**Zlepšení zpoždění (s)**
250	12949210	2,40	5616114	57,64
500	25538420	3,31	14154180	93,67
750	36586690	6,76	20024890	94,98
1000	14695720	185,31	−5986630	−72,20

Tabulka 3.6: Výsledky plošného AIMu

Mohlo by v nás nastat znepokojení, když se podíváme na výsledky při 1000 vozidlech. Z dat však můžeme vyčíst, že zde došlo k zaseknutí v necelé polovině času experimentu. Simulace pak výsledky spočítala pro celý běh. Víme tedy, že 1000 vozidel už je na autonomní křižovatky moc. Připomeňme si, že se u nich fronty vytváří dále než v případě semaforů, tudíž je celková kapacita nižší. Měření na 1000 vozidlech tedy dále provádět nebudeme.

Pojďme se ještě vrátit k hodnotám zlepšení, které jsou pro nás nejzásadnější. Jednotlivé řádky v tomto případě nejde porovnávat mezi sebou, protože vyšší zátěž znamená jiné výsledky i u semaforů. Čekali bychom však, že rozdíl mezi 250 a 500 vozidly bude ve zlepšení vzdálenosti zhruba dvojnásobný (nebo horší kvůli škálování). Autonomní křižovatky však dominují mnohem víc (násobek 2,52). Je to způsobené tím, že semafory jsme na 250 vozidlech vylepšili pomocí stopek, a proto dosahují v této kategorii lepších výsledků. Čísla se nám srovnají, když porovnáváme 500 a 750 vozidel.

Máme tedy pro každou kategorii (zátěž) zvlášť výsledky. V budoucích experimentech však nečekáme pokrytí všech křižovatek v systému AIMem. Hodilo by se tedy nějak měřit, jak si systém v experimentu vede *na jednu křižovatku*. Přesně tato data nám přináší tabulka 3.7.

Uváděná čísla představují, kolik průměrná křižovatka v dané konfiguraci přinesla oproti čistě semaforovému řešení. například při 250 vozidlech je plošný AIM

Počet vozidel	Počet AIM křižovatek	Zlepšení vzdálenosti na křižovatku (m)	Zlepšení zpoždění na křižovatku (s)
250	49	114615	1,176
500	49	288861	1,912
750	49	408671	1,938

Tabulka 3.7: Zlepšení plošného AIMu na křižovatku

v celkové ujeté vzdálenosti lepší o 5616114 m, což znamená, že jedna z jeho 49 křižovatek v průměru přinesla zlepšení o 114615 m (= 5616114/49). Vyšší číslo je tedy lepší. Podobně to funguje s průměrným zpožděním, kde sice obecně nižší číslo je lepší, ale my zde uvádíme hodnoty zlepšení, které jsou pozitivní, tudíž i zde je vyšší číslo lepší. Zmíněných 250 vozidel má na AIMu oproti semaforům průměrnou cestu kratší o 57.64 s, což značí 1.176 s na průměrnou vylepšenou křižovatku.

Zlepšení na křižovatku jsou tedy hodnoty, kterým se budeme dále nejvíce věnovat (ty další jdou zpětně dopočítat, případně se dají nalézt v přílohových datech). Následně můžeme sledovat, zda došlo k velkému zvýšení (tedy se jedná o dobrou konfiguraci), nebo se naopak čísla drží kolem stejných hodnot (tedy příliš nezáleží na počtu AIM křižovatek), nebo dokonce klesnou (tedy je konfigurace efektivně horší).

3.7 Náhodná volba

Jeden z nejsnazších přístupů je vybrat autonomní křižovatky naprosto náhodně. Simulace tuto možnost obsahuje. Stačí nám tedy vyzkoušet různé hodnoty poměrů algoritmů. Výsledky nalezneme v tabulce 3.8. Přidali jsme k nim i hodnotu z plošného testování pro porovnání.

Upozorňujeme, že generátor v simulaci funguje na bázi poměrů (procent). Takže byť 20 % ze 49 není přesně 8, generátor s daným seedem dorazil na tuto hodnotu (musíme počítat s rozptylem).

Když se podíváme na experimenty s 250 vozidly, vidíme, že několik prvních výsledků dokonce přineslo zhoršení (záporná čísla). To je dané tím, že semafory v tomto scénáři používají jako doplňky stopky, zatímco při náhodném generování jsou autonomní křižovatky doplněny pouze semafory. Při vyšším množství AIM křižovatek se už dostaneme do plusu, ale stále vychází nejlépe nasadit autonomní křižovatky plošně.

U 500 vozidel už dostáváme lepší výsledky. Sice žádný nepředčil plošný AIM, ale například zlepšením ohledně průměrného zpoždění už jsou na tom velice podobně.

Experiment se 750 vozidly a 75 % nedoběhl do konce kvůli zaseknutí. Při nižších procentech tato konfigurace produkuje slabé výsledky. Na 50 % je dokonce po stránce vzdálenosti horší než 500-vozidlový, což by se vzhledem k vyšším výsledkům u plošného AIMu dít nemělo.

Obecně se náhodný výběr autonomních křižovatek nezdá jako moc šťastná volba. Propustností na přidanou křižovatku jsou na tom všechny případy hůř než

Počet vozidel	Poznámka	Počet AIM křižovatek	ZVNK[a] (m)	ZZNK[b] (s)
250	10% AIM	5	−279894	−6,19
250	20% AIM	8	−131998	−2,74
250	30% AIM	13	−59955	−1,187
250	50% AIM	27	41960	0,665
250	75% AIM	38	79397	1,017
250	100% AIM	49	114615	1,176
500	10% AIM	5	24222	0,314
500	20% AIM	8	96878	1,339
500	30% AIM	13	105522	1,344
500	50% AIM	27	184545	1,886
500	75% AIM	38	217395	1,861
500	100% AIM	49	288861	1,912
750	10% AIM	5	33996	0,348
750	20% AIM	8	127881	1,216
750	30% AIM	13	144717	1,318
750	50% AIM	27	162747	1,330
750	100% AIM	49	408671	1,938

Pozn: [a] Zlepšení vzdálenosti na křižovatku oproti semaforům

Pozn: [b] Zlepšení zpoždění na křižovatku oproti semaforům

Tabulka 3.8: Náhodně vybrané autonomní křižovatky

plošný AIM. Dobrých hodnot u zpoždění jsme dosáhli pouze u 500 vozidel na vyšších počtech AIM křižovatek.

3.8 Design města

V dalších pokusech využijeme design města jako takového. Dříve jsme popisovali, že zde lze nalézt silnice dvou typů: hlavní a vedlejší. V našem standardním městě vedou 4 hlavní silnice, které se spolu kříží a zároveň se potkávají i s těmi vedlejšími. Jelikož zde dostáváme hodně možností, ukážeme si data pro různé zátěže postupně.

Výsledky pro 250 vozidel nalezneme v tabulce 3.9. Popis jednotlivých případů nemusí být na první pohled jasný. Máme na mapě celkem 3 typy křižovatek: křížení hlavních cest (ty jsou celkem 4), křížení hlavních cest s vedlejšími (těch je dohromady 20) a ostatní (pouze vedlejší – najdeme jich 25). Dříve jsme už prováděli experimenty, kde jsou všechny typy obsazené buď semafory, nebo AIMem. Nyní chceme ještě otestovat zbylých 6 kombinací mezi nimi. V případě 250 vozidel místo semaforů na „ostatní“ křižovatky umisťujeme stopky, aby to lépe odpovídalo semaforovým testům.

Spodní řádek tabulky nám opět připomíná „nejlepší“ možné výsledky (pro plošný AIM) po stránce zlepšení na průměrnou křižovatku oproti čistým semaforům. Vidíme, že nasazením autonomních křižovatek pouze na 4 hlavní křižovatky se podařilo překonat obě tyto hodnoty. Obzvlášť ve zpoždění je rozdíl markantní,

Hlavní	Hl. a vedl.	Vedl.	Počet AIM křižovatek	ZVNK (m)	ZZNK (s)
Sem.	Sem.	AIM	25	13533	0,239
Sem.	AIM	Stopky	20	99488	1,420
Sem.	AIM	AIM	45	84591	1,009
AIM	Sem.	Stopky	4	116675	1,962
AIM	Sem.	AIM	29	35369	0,567
AIM	AIM	Stopky	24	97872	1,340
AIM	AIM	AIM	49	114615	1,176

Tabulka 3.9: Autonomní křižovatky dle designu města pro 250 vozidel

ale musíme si uvědomit, že se jedná pouze o 4 křižovatky, tudíž celkové zlepšení je v průměru „jenom“ asi 8 vteřin. Přidávat AIM na další křižovatky už se nevyplácí tolik – minimálně u těch vedlejších dochází k velmi malému zlepšení celkových čísel. Naopak když přidáme AIM na křížení hlavních a vedlejších cest, dostáváme poměrně slušné výsledky. V propustnosti nepomáhají tolik jako plošný AIM, ale v hodnotách zpoždění na křižovatku jsou na tom lépe.

Nyní se podívejme na výsledky pro 500 vozidel v tabulce 3.10.

Hlavní	Hl. a vedl.	Vedl.	Počet AIM křižovatek	ZVNK (m)	ZZNK (s)
Sem.	Sem.	AIM	25	139744	1,570
Sem.	AIM	Sem.	20	171357	1,930
Sem.	AIM	AIM	45	229100	1,781
AIM	Sem.	Sem.	4	220110	2,930
AIM	Sem.	AIM	29	164958	1,710
AIM	AIM	Sem.	24	226986	2,263
AIM	AIM	AIM	49	288861	1,912

Tabulka 3.10: Autonomní křižovatky dle designu města pro 500 vozidel

Tentokrát už se žádnému nastavení nepodařilo předčít plošný AIM po stránce zlepšení vzdálenosti, ale v případech, kde jsou autonomní křižovatky na hlavních cestách, vidíme opět lepší čísla po stránce zpoždění.

Nyní se podívejme na výsledky pro 750 vozidel v tabulce 3.11.

Verzi s 45 křižovatkami se obecně daří po stránce zlepšení propustnosti, ale je to dané tím, že autonomních křižovatek už je skoro tolik, co při plošném AIMu. V takové situaci by bylo nejlepší doplnit i poslední 4. Po stránce zpoždění se nejlépe vede opět při nasazení na 4 hlavní křižovatky, žádná jiná konfigurace dokonce tentokrát nepřekonala cílové hodnoty.

3.9 Vytíženost křižovatky

Poslední hlavní způsob umísťování algoritmů, který prozkoumáme, je s výpočtem vytíženosti jednotlivých křižovatek. Ten totiž probíhá hned na začátku,

Hlavní	Hl. a vedl.	Vedl.	Počet AIM křižovatek	ZVNK (m)	ZZNK (s)
Sem.	Sem.	AIM	25	156970	1,298
Sem.	AIM	Sem.	20	221987	1,813
Sem.	AIM	AIM	45	322869	1,797
AIM	Sem.	Sem.	4	284805	2,637
AIM	Sem.	AIM	29	162974	1,306
AIM	AIM	Sem.	24	224152	1,737
AIM	AIM	AIM	49	408671	1,938

Tabulka 3.11: Autonomní křižovatky dle designu města pro 750 vozidel

kdy uživatel zadá hodnoty do spawnovacích distribucí. Systém spočítá, kolik přes které křižovatky teoreticky vede cest a podle toho jim přidělí hodnotu vytíženosti. V praxi také často existují data, z kterých lze vyčíst používanost jednotlivých křižovatek ve městě.

Při prohlížení simulace s generováním dle tohoto nastavení se ukazuje, že vybrané křižovatky dost odpovídají těm z předchozí sekce. Nejvytíženější jsou ty, kde se kříží hlavní silnice. Následuje křížení hlavních silnic s vedlejšími. Akorát „ostatní“ křižovatky se dělí na 2 typy – ty na krajích jsou vytíženější než devítka uprostřed. Z předchozích experimentů však víme, že přidávat AIM na tyto křižovatky (vedlejší/méně vytížené) moc nepomáhá. Naopak máme mocný nástroj a chceme ho použít co nejvýhodněji. Z těchto důvodů v následujících experimentech nepůjdeme do takové hloubky, jako kdybychom předchozí výsledky neměli. Například nedává smysl testovat umístění autonomních křižovatek na nejméně vytížené uzly.

Tabulka 3.12 představuje výsledky pro 3 různé zátěže. „Prvních N AIM“ znamená, že vždy na N nejvytíženějších křižovatek byl nasazen AIM, zatímco jinde byly umístěny semafory. Výjimkou je systém s 250 vozidly, kde posledních 25 křižovatek (které jsou na čistě vedlejších silnicích) bylo pokryto stopkami, jako tomu bylo u původních experimentů se semafory. „Prostředních 14 AIM“ pak znamená, že 10 nejvytíženějších křižovatek bylo pokryto semafory, dalších 14 autonomními algoritmy a zbylých 25 opět semafory (stopkami u 250 vozidel).

Výsledky dopadly veskrze dle dřívějších očekávání. Čím víc pokrýváme od nejvytíženějších po ty méně vytížené křižovatky, zvyšuje se zlepšení vzdálenosti na křižovatku, ale klesá zlepšení ve zpoždění. Na pokusech s 250 a 750 vozidly, se dokonce konfigurace „Prvních 40 AIM“ ukázala v obou číslech výhodnější než plošný AIM. U 250 k celkové ujeté vzdálenosti chybělo jen 111550 m, abychom dostali naměřené maximum. Po stránce (celkového) průměrného zpoždění bylo prvních 40 jen o 0,66 s horší než plošný AIM. Musíme si ale uvědomit důvod – nacházíme se ve specifické situaci, kdy posledních 9 křižovatek jsou ty nejméně zatížené, na kterých byly použity stopky, které při nízké zátěži dosahují stejných výsledků jako AIM. I tak je to ale pro náš výzkum zajímavé zjištění. Znamená to, že vylepšování některých křižovatek bude mít velmi malý vliv.

Ve statistice občas zahlídneme výkyvy. Například první řádek pro 250 vozidel má lepší hodnotu ve zlepšení vzdálenosti než druhý řádek. Nebo že „Prvních 20 AIM“ pro 750 vozidel má horší výsledky než sousedé. To je dané tím, že v sys-

Počet vozidel	Poznámka	Počet AIM křižovatek	ZVNK[a] (m)	ZZNK[b] (s)
250	Prvních 5 AIM	5	127650	2,142
250	Prvních 10 AIM	10	100045	1,602
250	Prvních 20 AIM	20	103362	1,451
250	Prvních 40 AIM	40	137614	1,424
250	Plošný AIM	49	114615	1,176
250	Prostředních 14 AIM	14	72490	1,125
500	Prvních 5 AIM	5	198328	2,63
500	Prvních 10 AIM	10	217855	2,666
500	Prvních 20 AIM	20	221640	2,343
500	Prvních 40 AIM	40	283406	2,105
500	Plošný AIM	49	288861	1,912
500	Prostředních 14 AIM	14	149023	1,835
750	Prvních 5 AIM	5	283710	2,663
750	Prvních 10 AIM	10	290622	2,539
750	Prvních 20 AIM	20	247986	1,964
750	Prvních 40 AIM	40	415874	2,172
750	Plošný AIM	49	408671	1,938
750	Prostředních 14 AIM	14	181611	1,619

Tabulka 3.12: Autonomní křižovatky dle vytíženosti

tému je přítomných několik různých typů (které jsme testovali v předchozí fázi). Jinými slovy, křižovatky by se daly rozdělit do určitých skupin vytíženosti a pokud „špatně“ odhadneme, při jakém počtu se skupina mění, dostaneme takovéto výkyvy ve statistice. Pro znázornění se můžeme podívat na výsledky v předchozí sekci, kde vidíme, že 4 nejvytíženější křižovatky mají o dost lepší hodnoty než 5 křižovatek, které jsme měřili zde. Je to právě tím, že první 4 jsou si velice podobné, zatímco pátá už se liší.

Součástí těchto testů byl pokus o jiné pokrytí než od nejvytíženějších v podobě „14 prostředních“. Ten má však ve všech kategoriích nejhorší výsledky, a tedy jsme si potvrdili hypotézu, že se nejedná o správnou cestu výběru.

3.9.1 Změna distribuce spawnování

Určitým problémem předchozího experimentu bylo to, že výsledky byly velice podobné, jako při výběru dle designu. Proto jsme se rozhodli pro standardní město změnit hodnoty distribucí spawnování. Nastavili jsme, aby se na vstupech hlavních silnic objevovalo pouze 10 % všech aut a jen půlka z nich se vrátí do hlavního výstupu. Z 90 % na vedlejších vstupech také 90 % končí zpátky na výjezdu vedlejší silnicí. Výsledky představuje tabulka 3.13. Experiment jsme tentokrát pouštěli pouze na střední zátěž v podobě 500 vozidel. Jelikož město vypadá jinak, uvádíme v tabulce i hodnoty plošných semaforů a plošného AIMu spolu s pevnými výsledky pro celkovou ujetou vzdálenost a průměrné zpoždění.

Vidíme, že pokud jsou AIM křižovatky umístěny na křížení hlavních křižo-

Poznámka[a]		Vzdálenost[b]	Zpoždění[c]	ZVNK	ZZNK
Semafory	0	10968340	103,11	—	—
Křížení hlavních	4	11415020	96,56	111670	1,638
Všechny hlavní	24	13618830	69,53	110437	1,399
Všechny vedlejší	25	16580150	44,34	224472	2,351
Prvních 5 AIM	5	12182710	85,71	242874	3,48
Prvních 10 AIM	10	12838590	78,09	187025	2,502
Prvních 20 AIM	20	15897160	49,16	246441	2,698
Prvních 40 AIM	40	22051740	15,33	277085	2,194
Plošný AIM	49	25463880	3,54	295827	2,032

Pozn: [a] Popis konfigurace a počet AIM křižovatek

Pozn: [b] Celková ujetá vzdálenost v metrech

Pozn: [c] Průměrné zpoždění v sekundách

Tabulka 3.13: Vytíženost křižovatky pro 500 vozidel s jinou distribucí spawnování

vatek nebo křížení hlavních s hlavními i vedlejšími („Všechny hlavní“), tak jsou výsledky dost chudé. Naopak pokrytí jen křížení vedlejších silnic („Všechny vedlejší“) dává tentokrát mnohem lepší čísla.

Ověření jsme provedli tím, že jsme také zkusili algoritmy umístit dle vytíženosti křižovatek – teď už to pochopitelně nebyly znovu křížení hlavních, ale právě některé z vedlejších. Zde se vracíme na již dřívě ověřená data. Design města tedy nemá až takový vliv jako skutečná vytíženost.

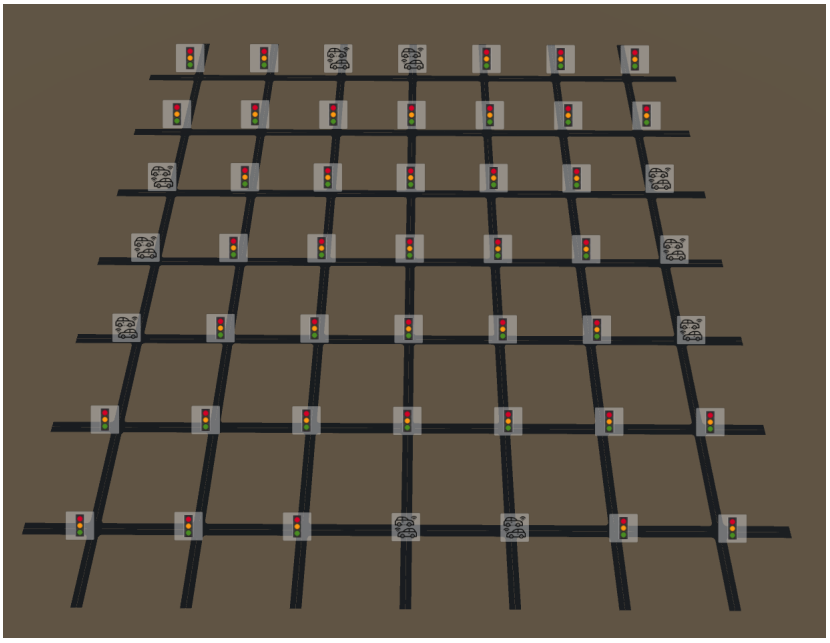
Zajímavé je, že se stále můžeme potkat s nějakými výkyvy. „Prvních 10 AIM“ má horší výsledky na křižovatku než oba jeho sousedé.

3.9.2 Alternativní město

Dává smysl podívat se na efektivitu výběru autonomních křižovatek dle zátěže i v trošku odlišném prostředí. Vygenerovali jsme pro tento účel nové město, kde jsme zakázali hlavní silnice. Vytváří se mřížka pouze 7x7 vedlejších silnic o 2 pruzích, což by mělo poskytovat větší propustnost. Proto jsme tentokrát experimenty pouštěli s 500, 1000 a 1500 vozidly. Tabulka 3.14 zachycuje všechny výsledky.

Všechny 3 zátěže nabízí poměrně podobné výsledky – s rostoucím počtem autonomních křižovatek se zvyšuje celková ujetá vzdálenost a průměrné zpoždění klesá. Hodnoty na křižovatku pak fungují jako jsme viděli už dříve – vyšší počty AIM křižovatek dávají lepší ujetou vzdálenost na přidanou křižovatku, ale zhoršující se zlepšení ve zpoždění.

Zajímavostí této sekce je počet 10 autonomních křižovatek, který se chová podobně na všech 3 zátěžích. V každé sadě totiž sice stále dochází ke zvýšení propustnosti a snížení celkového zpoždění, ale když se koukneme na čísla na křižovatku, vidíme, že je tento test horší než oba sousedi. V případě 1500 vozidel dokonce šlo o jediný z experimentů, který se v průběhu zasekl (proto záporná čísla – asi polovinu simulace už neprobíhal žádný pohyb). Toto chování pro „prvních 10“ už jsme zahlédli dříve na křižovatce 7x7. Můžeme tedy předpokládat, že v této mřížce jde u tohoto čísla o propad.



Obrázek 3.4: Alternativní město s 10 nejvytíženějšími křižovatkami nastavenými na AIM

3.10 Další experimenty

Abychom nemuseli provádět miliony potenciálních experimentů, museli jsme mnoho hodnot v nastavení pevně ukotvit (např. v rámci *standardního města*) a hýbali jsme jen s částí. Nyní se pokusíme v krátkosti pokrýt pár příkladů, které nám přijdou zajímavé a u kterých jsme moc neprozkoumali alternativní možnosti.

Z předchozích experimentů víme, že nejlépe funguje nasazení AIMu na vytížených křižovatkách, příp. kříženích hlavních cest (pokud jsou používané).

3.10.1 Střídání algoritmů

Jedno z rozložení, které jsme dosud neměli možnost vyzkoušet, je aby autonomní křižovatky a semafory byly v mřížce „na střídačku“. Vytvořili jsme proto speciální město 8x8 silnic, které všechny mají 2 pruhy, ale 4 v obou směrech byly nastaveny jako hlavní, tudíž se střídají a prokládají s těmi vedlejšími (lze vidět na obrázku 3.5). V tabulce 3.15 nalezneme porovnání s dalšími typickými přístupy. „Na střídačku“ představuje ono střídání algoritmů a „Lehce na střídačku“ je verze, kde je síť AIM křižovatek řidší a jen každá druhá ze střídavé mřížky je autonomní.

Ukazuje se, že ani jedno z nových rozložení bohužel neposkytuje žádnou velkou výhodu. Naopak, pokrytí nejvytíženějších křižovatek stále funguje nejlépe a generuje vysoká čísla, zatímco oba nové přístupy jsou na tom hůře.

3.10.2 Náhodné město

Další možností, co se nabízí, je nepoužívat stále plnou mřížku, ale zkusit vygenervoat trochu děravé město. Použitý příklad si můžeme prohlédnout na obrázku 3.6. Tabulka 3.16 předkládá naměřené hodnoty.

Počet vozidel	Počet AIM křižovatek	Vzdálenost	Zpoždění	ZVNK	ZZNK
500	0	11411570	95,42	—	—
500	5	12471520	81,83	211990	2,720
500	10	12871760	77,19	146019	1,824
500	20	15813540	49,79	220098	2,282
500	40	21184840	19,08	244332	1,909
500	49	25755960	2,84	292743	1,890
1000	0	21915170	102,23	—	—
1000	5	23831200	88,19	383206	2,807
1000	10	24548120	83,97	263295	1,826
1000	20	30398560	54,06	424170	2,408
1000	40	41148460	21,38	480832	2,021
1000	49	49885968	5,02	570833	1,984
1500	0	29196870	122,59	—	—
1500	5	29633140	119,88	87254	0,542
1500	10	11041930	417,70	−1815494	−29,511
1500	20	38525608	76,16	466437	2,321
1500	40	58181920	26,61	724626	2,399
1500	49	64859480	16,49	727808	2,165

Tabulka 3.14: Vytíženost křižovatky pro různé zátěže s jiným městem

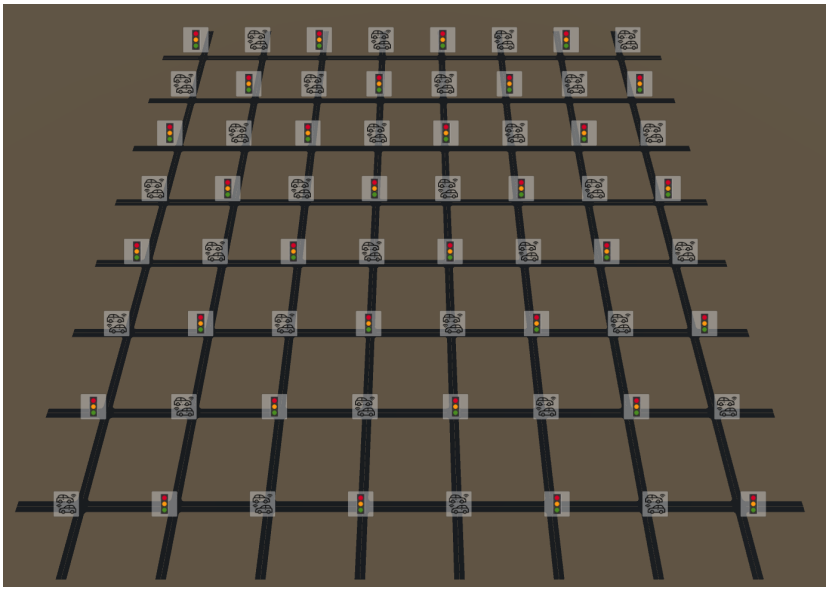
Tentokrát statistiku naprosto jasně dominuje nasazení dle vytíženosti – dokonce je v obou hodnotách na křižovatku lepší než plošný AIM. V předchozích experimentech se stávalo, že „Prvních 10 AIM“ zaostávalo za sousedy. Zde je tomu však obráceně – tato konfigurace je zdaleka nejlepší. Sice se stále vyplatí přidávat další autonomní křižovatky v celkové statistice, ale výhodností na křižovatku vede 10 křižovatek. Tím se také ukazuje, že toto číslo není nijak „zakleté“. Zkrátka každé nové město má jiné parametry a nejefektivnější počet se pohybuje jinde.

Chtěli bychom ještě upozornit, že 500 vozidel bylo pro toto město dost limitních. Na datech o průjezdech za minutu je vidět, že některé běhy simulace už se ke konci začínaly zasekávat (semafory, „všechny hlavní“, „všechny vedlejší“).

3.10.3 Jednosměrky

Jednou neprozkoumanou možností jsou také jednosměrné silnice. Vygenerovali jsme jedno takové město s náhodnými jednosměrkami (obrázek 3.7). Výsledky několika typických konfigurací při 500 vozidlech si můžeme prohlédnout na tabulce 3.17.

Vzhledem k neobvyklému vzhledu města v těchto experimentech může docházet k lehce vyššímu počtu kolizí – na obou hlavních typech křižovatek. Tyto případy jsou dané tím, že systém nebyl podroben tak důkladnému testování v těchto konfiguracích. Šlo by to vyřešit zvýšením velikostí bufferů (pro autonomní křižovatky) či prodloužením světelných fází (u semaforů). Jelikož se jedná o pevné



Obrázek 3.5: Město se střídajícími se algoritmy

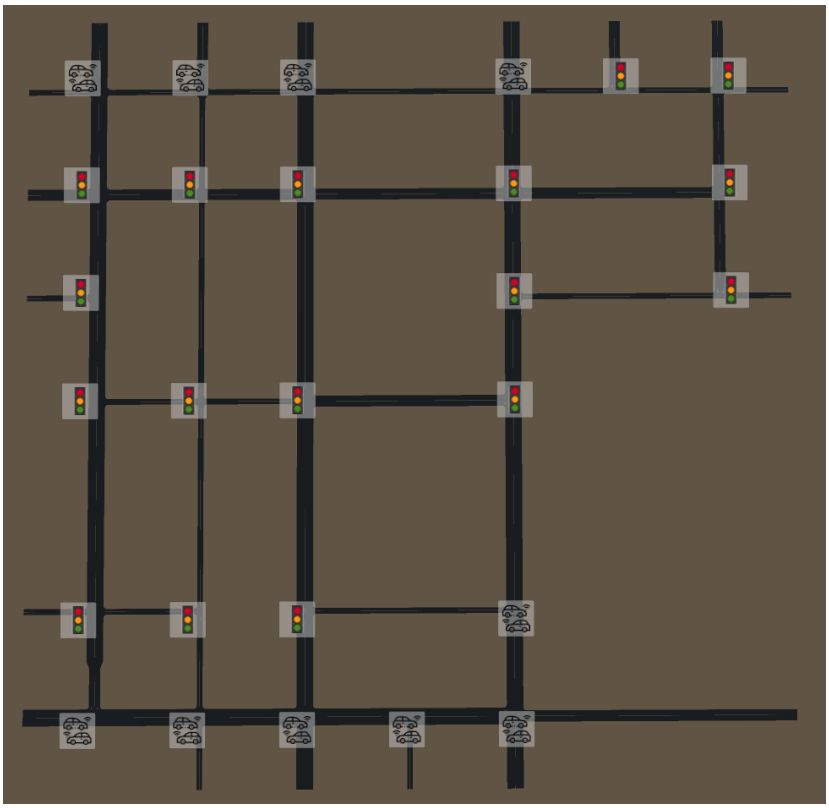
Poznámka		Vzdálenost	Zpoždění	ZVNK	ZZNK
Semafory	0	22375710	110,88	—	—
Lehce na střídačku	16	26233140	83,18	241089	1,732
Na střídačku	32	30801960	59,28	263320	1,613
Prvních 5 AIM	5	23835930	99,55	292044	2,266
Prvních 10 AIM	10	25052720	90,87	267701	2,001
Prvních 20 AIM	20	29361810	66,22	349305	2,233
Prvních 40 AIM	40	35460872	40,96	327129	1,748
Plošný AIM	64	50841080	4,09	444771	1,669

Tabulka 3.15: Křižovatky na střídačku pro 1000 vozidel

hodnoty, tak jsme je neupravovali, abychom tím neovlivnili jiné části systému. Kolize při milionech naježděných metrů se pohybují nejvýše na nižších desítkách.

Zvolená zátěž 500 vozidel byla pravděpodobně moc vysoká, protože u několika konfigurací AIM křižovatek došlo k zaseknutí (poslední 3 řádky). Když jsme pustili simulaci s autonomními algoritmy na křižovatkách hlavních silnic, systém se nezasekl, ale došlo ke zhoršení statistik oproti čistým semaforům.

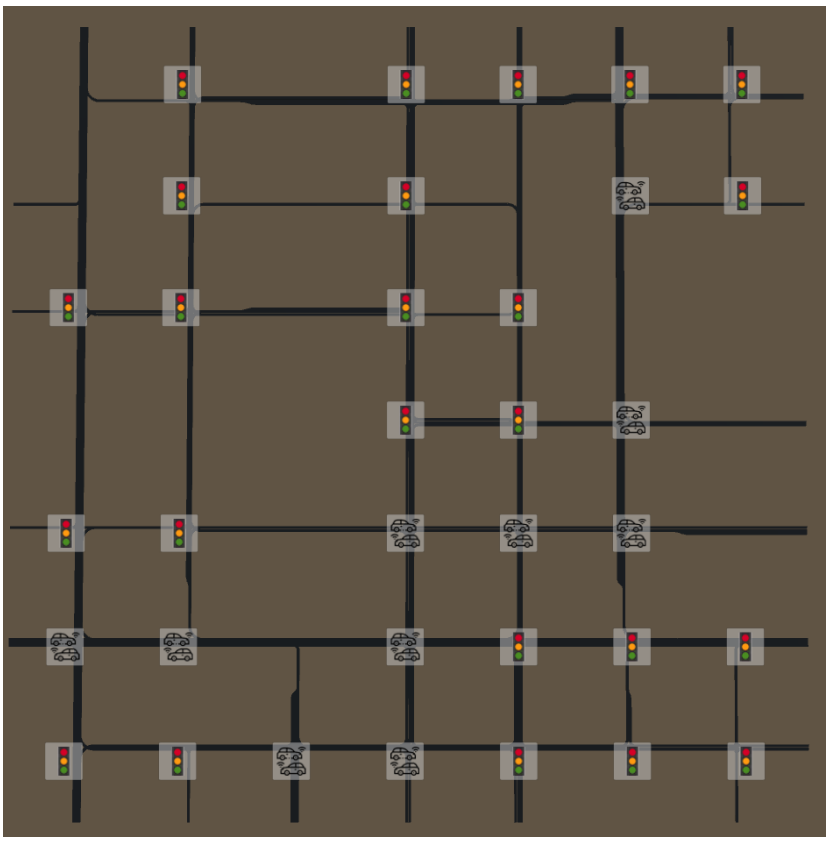
Ukazuje se, že velice náhodná města, jako je bez pochyby to jednosměrkové, jsou nepříznivá extenzivnímu testování zapojení autonomních křižovatek. Problém je jednak v tom, že některé pevné hodnoty v simulaci nejsou na nové křižovatky připravené, ale hlavně to, že náhodně vygenerované silnice jsou často velmi nesmyslné a nerealistické. Pro získání stabilních výsledků bychom se na tyto možnosti museli dopodrobna zaměřit.



Obrázek 3.6: Náhodné město s 10 nejvytíženějšími křižovatkami nastavenými na AIM

Poznámka		**Vzdálenost**	**Zpoždění**	**ZVNK**	**ZZNK**
Semafory	0	8699265	147,83	—	—
Křížení hlavních	3	9737274	125,12	346003	7,57
Všechny hlavní	19	10622050	106,65	101199	2,167
Všechny vedlejší	8	9327064	133,3	78475	1,816
Prvních 5 AIM	5	11283260	97,73	516799	10,021
Prvních 10 AIM	10	17688570	36,60	898930	11,123
Prvních 20 AIM	20	19611770	26,02	545625	6,091
Plošný AIM	27	20738770	20,58	445908	4,713

Tabulka 3.16: Náhodné město s 500 vozidly



Obrázek 3.7: Jednosměrkové město s 10 nejvytíženějšími křižovatkami nastavenými na AIM

Poznámka		**Vzdálenost**	**Zpoždění**	**ZVNK**	**ZZNK**
Semafory	0	8183719	167,47	—	—
Křížení hlavních	2	8091706	169,49	−46006	−1,011
Všechny hlavní	14	8718120	152,3	38172	1,083
Všechny vedlejší	20	12374380	88,67	209533	3,94
Prvních 5 AIM	5	12888430	84,83	940942	16,528
Prvních 10 AIM	10	3602994	457,67	−458072	−29,021
Prvních 20 AIM	20	5972884	252,43	−110542	−4,248
Plošný AIM	34	6510016	227,43	−49227	−1,764

Tabulka 3.17: Město s jednosměrkami pro 500 vozidel

Závěr

V kapitole 2 (2) jsme popsali systém, který se ukázal být velice efektivní pro testování. Cílem experimentů nebylo zjistit, jak rychle simulace běží, ale uvedeme zde alespoň orientační hodnoty, jelikož nám přijdou pozitivní. Na stolním počítači (6-jádrový procesor s maximální frekvencí 4.9 GHz a 32 GB RAM) simulace s 49 autonomními křižovatkami a 1000 vozidly běží plynule v reálném čase. Při pětinásobném zrychlení se dá pohodlně sledovat a při desetinásobném trhaně. Samostatný test (49 autonomních křižovatek a 1000 vozidel) s hodinovým simulačním časem, pokud má k dispozici veškeré komputační zdroje počítače, zabere asi 11 minut (při spuštění s co nejvyšším zrychlením času).

V simulovaném prostředí jsme vyzkoušeli nový typ algoritmu autonomních křižovatek, který plánuje změnu rychlosti na více místech. Několik částí jsme si sice zjednodušili, ale výsledek působí věrně a funguje efektivně. V realističtějším prostředí by náš přístup byl nejspíš naimplementován tak, že křižovatka by vytvořila rámcový plán a sama vozidla by musela řešit změnu rychlostí pro dodržení tohoto plánu.

Naměřili jsme výsledky a došli k určitým závěrům, které představujeme níže. Hotový systém nabízí možnosti pro další experimentování či rozšiřování.

Je potřeba uvědomit si, že v této práci jsme přistoupili k problematice určitým způsobem a bylo potřeba na této cestě činit mnohá rozhodnutí, která často skutečný svět zjednodušila. Stojíme si za představovanými výsledky, ale rozhodně podporujeme další výzkum v této oblasti, který se zaměří na problematiku, kterou jsme rozsahem práce nemohli do hloubky prozkoumat.

3.11 Výsledky

Ve 3. kapitole (3) jsme nejdříve měřili účinnost algoritmů na samostatné křižovatce. Došli jsme k tomu, že stopky fungují lépe než semafory na velmi malých zátěžích. Při typické využívanosti (od nejmenší po velkou) však nejlépe dopadl náš chytrý algoritmus. S vyšší hustotou vozidel efektivita lehce klesla, ale stále výsledky předčil ostatní přístupy.

Dále probíhalo testování, které bylo hlavním cílem práce – porovnání více algoritmů vedle sebe a hledání nejlepší konfigurace. Ukázalo se, že nasadit AIM plošně má velice dobré výsledky – vždy se jednalo o konfiguraci s největší celkovou propustností a nejmenším průměrným zpožděním. To bylo však poměrně očekávatelné. Plošné autonomní křižovatky si vedly poměrně dobře i co se týče hodnot na průměrnou přidanou křižovatku a bylo těžké je porazit.

Ukázalo se, že aplikace AIMu na náhodné uzly produkuje velice špatné výsledky (ale stále lepší než čistě semaforové řešení). O dost lépe na tom byla zapojení, kde jsme chytré křižovatky nasazovali na uzly hlavních silnic, či na nejvytíženější místa. Tyto dva přístupy u prvních experimentů šly dost ruku v ruce. Později jsme tyto vlastnosti rozdělili, aby hlavní silnice nebyly tak využívané, a odhalili, že zátěž křižovatky je tím nejdůležitějším faktorem.

Zkoušeli jsme i různá další nastavení. Ukázalo se, že střídat vedle sebe semafory a autonomní křižovatky nenabízí žádnou výhodu. Lépe funguje výběr dle vytíženosti. To samé se ukázalo i na náhodně generovaných městech, kde tato

konfigruace opět ukazovala nejlepší výsledky.

Co se týče *počtů* nejvytíženějších křižovatek a jaký dává nejlepší hodnoty na průměrnou vylepšenou křižovatku, byly výsledky různorodější. Nedokážeme říct jasné číslo a je potřeba znát širší kontext. Pokud víme, že hlavní silnice jsou skutečně silně využívané, pak nám tato informace o designu města může pomoct a můžeme se při určování počtu zastavit například na počtu hlavních křížení, či na počtu křížení, která zahrnují hlavní silnici.

Na druhou stranu jsme objevili, že nasazovat chytrý algoritmus na skutečně málo používané křižovatky (tam, kde stačí stopka) téměř nedává smysl. Pokud byla auta v naší simulaci na řadě (a nestála zde předtím), tak před stopkou nezastavovala. Stopky díky tomu byly efektivnější než v realitě. I tak platí, že pokud není křižovatka moc využívaná, umisťovat na ni speciální (potenciálně drahá) zařízení přinese minimum celkového zlepšení.

Je možné, že v praxi budou designéři města omezení rozpočtem, a tudíž bude maximální počet nových křižovatek daný předem. Pak bychom doporučili co nejefektivněji určit nejpoužívanější křižovatky a na těch nové algoritmy nasadit.

3.12 Budoucí práce

Jedná se o velice komplexní problematiku, takže existuje skutečně mnoho vylepšení, která bychom mohli učinit. Zde budeme rozebírat ta, která nám přijdou nejdůležitější vzhledem k účelu této práce.

Bylo by dobré do simulace přidat další autonomní algoritmy jako alternativy k tomu současnému nebo vytvořit nějakou platformu, kde se dají řešení porovnávat ve stejném prostředí. Podobný výzkum zatím zoufale schází. Minimálně by bylo zajímavé porovnat centralizovaný přístup s distribuovaným řízením křižovatky – například kdybychom věděli, že je nějak omezené, nebo pokud by nebylo tak účinné, a měli bychom za úkol určit, kde se který přístup použije.

Simulace by šla vylepšit po fyzikální stránce. Použili jsme dost jednoduchý model vozidel a zjednodušili trajektorie pohybu. Zrychlování a zpomalováníní byly implementovány také na poměrně umělé úrovni a bylo by zajímavé prozkoumat realističtější situace, kdy jejich funkce nejsou lineární a vysoká změna rychlosti má negativní vliv na komfort (což ale nemusí být nutná podmínka, protože vozidla teoreticky budou moct jezdit bez lidského osazení).

Práce by určitě zasloužila plnohodnotné řešení přejezdů mezi pruhy.

Dále by se dal rozšířit generátor měst. Ten funguje poměrně dobře pro standardní konfigurace, ale při náhodném nastavení může produkovat trochu podivná města. Chtělo by to se do hloubky zaměřit na jednotlivé případy. Respektive by šlo přidat funkcionalitu, kdy se algoritmy více přizpůsobují vzhledu křižovatky (my jsme na několika místech použili pevné hodnoty). Pak by také generování mohlo probíhat na základě skutečných topologií měst.

Obecně by šlo rozšířit práci o množství nových scénářů: nové typy úseků silnic (zatáčky, dálnice, kruhové objezdy, parkovací místa, ...), nové typy a velikosti vozidel (nákladní auta, kamiony, autobusy, motocykly, kola, ...), zapojení chodců a další. Jedná se už o poměrně malé detaily, které by nejspíš neměly tak velký vliv na výsledky, spíš pomohly představit si, jak systém bude vypadat v praxi.

Úplně novou kapitolou by bylo začít problém řešit po stránce vyšší spolupráce než jen na mikroúrovni komunikace blízkých vozidel a křižovatkových algoritmů.

Výsledek by mohl vypadat o dost jinak, kdyby v něm bylo přítomné nějaké celkové řízení. Pokud bychom měli řadu nových systémů křižovatek, kromě komunikace s vozidly by si i mohly předávat informace navzájem, nebo dokonce nějaké vyšší řídící entitě. V praxi dnes něco podobného umožňují navigační aplikace, které měří vytíženost úseků a nabízejí řidičům alternativní cesty.

V budoucnu bychom obecně uvítali více textů, které se nesnaží objevit nové řešení a nový algoritmus pro křižovatku, ale využívají rozsáhlou základnu existujícího výzkumu a dále na něm staví. Často si práce (včetně té naší) kladou různá omezení, čímž si problém zjednodušují. Bylo by dobré se takovýchto rozhodnutí zbavit a připravit co nejkomplexnější systém, který počítá s co nejvíce faktory reálného světa.

Seznam použité literatury

Au, T.-C., Zhang, S. a Stone, P. (2015). Autonomous intersection management for semi-autonomous vehicles. In *The Routledge handbook of transportation*, pages 88–104. Routledge.

Bichiou, Y. a Rakha, H. A. (2018). Developing an optimal intersection control system for automated connected vehicles. *IEEE Transactions on Intelligent Transportation Systems*, **20**(5), 1908–1916.

Braess, D. (1968). Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung*, **12**, 258–268.

Carlino, D., Boyles, S. D. a Stone, P. (2013). Auction-based autonomous intersection management. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 529–534. IEEE.

Dresner, K. a Stone, P. (2008). A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, **31**, 591–656.

Emami, P., Pourmehrab, M., Martin-Gasulla, M., Ranka, S. a Elefteriadou, L. (2018). A comparison of intelligent signalized intersection controllers under mixed traffic. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 341–348. IEEE.

Fayazi, S. A., Vahidi, A. a Luckow, A. (2019). A vehicle-in-the-loop (vil) verification of an all-autonomous intersection control scheme. *Transportation Research Part C: Emerging Technologies*, **107**, 193–210.

Fellendorf, M. a Vortisch, P. (2010). Microscopic traffic flow simulator vissim. *Fundamentals of traffic simulation*, pages 63–93.

Freeman, A. a Freeman, A. (2015). The object pool pattern. *Pro Design Patterns in Swift*, pages 137–156.

Freepik Company S.L. (2023). Flaticon. URL `https://www.flaticon.com/`.

Galletebeitia, B. (2011). *Comparative analysis between the diverging diamond interchange and partial cloverleaf interchange using microsimulation modeling*. Florida Atlantic University.

Gholamhosseinian, A. a Seitz, J. (2022). A comprehensive survey on cooperative intersection management for heterogeneous connected vehicles. *IEEE Access*, **10**, 7937–7972.

Hart, P. E., Nilsson, N. J. a Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, **4**(2), 100–107.

Hassan, A. A. a Rakha, H. A. (2014). A fully-distributed heuristic algorithm for control of autonomous vehicle movements at isolated intersections. *International Journal of Transportation Science and Technology*, **3**(4), 297–309.

Hausknecht, M., Au, T.-C. a Stone, P. (2011). Autonomous intersection management: Multi-intersection optimization. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4581–4586. IEEE.

Hillier, J. A. a Rothery, R. (1967). The synchronization of traffic signals for minimum delay. *Transportation Science*, **1**(2), 81–94.

Huang, H., Chin, H. C. a Haque, M. M. (2008). Severity of driver injury and vehicle damage in traffic crashes at intersections: a bayesian hierarchical analysis. *Accident Analysis & Prevention*, **40**(1), 45–54.

Jin, Q., Wu, G., Boriboonsomsin, K. a Barth, M. (2012). Multi-agent intersection management for connected vehicles using an optimal scheduling approach. In *2012 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 185–190. IEEE.

Jin, Q., Wu, G., Boriboonsomsin, K. a Barth, M. (2013). Platoon-based multi-agent intersection management for connected vehicle. In *16th international ieee conference on intelligent transportation systems (itsc 2013)*, pages 1462–1467. IEEE.

Khayatian, M., Mehrabian, M. a Shrivastava, A. (2018). Rim: Robust intersection management for connected autonomous vehicles. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 35–44. IEEE.

Khayatian, M., Lou, Y., Mehrabian, M. a Shirvastava, A. (2019). Crossroads+: A time-aware approach for intersection management of connected autonomous vehicles. *ACM Trans. Cyber-Phys. Syst.*, **4**(2). ISSN 2378-962X. doi: 10.1145/3364182. URL `https://doi.org/10.1145/3364182`.

Khayatian, M., Mehrabian, M., Andert, E., Dedinsky, R., Choudhary, S., Lou, Y. a Shirvastava, A. (2020). A survey on intersection management of connected autonomous vehicles. *ACM Transactions on Cyber-Physical Systems*, **4**(4), 1–27.

Koenig, N. a Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE.

Levin, M. W. a Rey, D. (2017). Conflict-point formulation of intersection control for autonomous vehicles. *Transportation Research Part C: Emerging Technologies*, **85**, 528–547.

Li, B., Zhang, Y., Zhang, Y., Jia, N. a Ge, Y. (2018). Near-optimal online motion planning of connected and automated vehicles at a signal-free and lane-free intersection. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1432–1437. IEEE.

Li, L. a Wang, F.-Y. (2006). Cooperative driving at blind crossings using intervehicle communication. *IEEE Transactions on Vehicular technology*, **55** (6), 1712–1724.

Liu, R. a Moini, N. (2015). Benchmarking transportation safety performance via shift-share approaches. *Journal of Transportation Safety & Security*, **7**(2), 124–137.

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P. a Wiessner, E. (2018). Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE. URL `https://elib.dlr.de/124092/`.

Lu, Q. a Kim, K.-D. (2016). Intelligent intersection management of autonomous traffic using discrete-time occupancies trajectory. *Journal of Traffic and Logistics Engineering Vol*, **4**(1), 1–6.

Microsoft (2023). `C#` documentation. URL `https://learn.microsoft.com/en-us/dotnet/csharp/`.

Nystrom, R. (2014). *Game programming patterns*. Genever Benning.

Shi, W., Shen, S. a Liu, Y. (2009). Automatic generation of road network map from massive gps, vehicle trajectories. In *2009 12th international IEEE conference on intelligent transportation systems*, pages 1–6. IEEE.

Simopt, s.r.o. (2023). Ucpávání křižovatek při jízdě. URL `https://www.bezpecnecesty.cz/cz/bezpecna-jizda-v-aute/ucpavani-krizovatek`.

Skog, I. a Handel, P. (2009). In-car positioning and navigation technologies—a survey. *IEEE Transactions on Intelligent Transportation Systems*, **10** (1), 4–21.

Škopková, V., Barták, R. a Švancara, J. (2020). What does multi-agent path-finding tell us about intelligent intersections. In *12th International Conference on Agents and Artificial Intelligence (ICAART)*.

Tachet, R., Santi, P., Sobolevsky, S., Reyes-Castro, L. I., Frazzoli, E., Helbing, D. a Ratti, C. (2016). Revisiting street intersections using slot-based systems. *PloS one*, **11**(3), e0149607.

Unity Technologies (2023a). Unity Real-Time Development Platform. URL `https://unity.com/`.

Unity Technologies (2023b). Unity Asset Store. URL `https://assetstore.unity.com/`.

Unity Technologies (2023c). Unity Documentation. URL `https://docs.unity3d.com/`.

Wuthishuwong, C. a Traechtler, A. (2013). Coordination of multiple autonomous intersections by using local neighborhood information. In *2013 international conference on connected vehicles and expo (ICCVE)*, pages 48–53. IEEE.

Zhong, Z., Nejad, M. a Lee, E. E. (2020). Autonomous and semiautonomous intersection management: A survey. *IEEE Intelligent Transportation Systems Magazine*, **13**(2), 53–70.

Seznam obrázků

Seznam tabulek

A. Přílohy

A.1 Technická dokumentace

V této sekci popíšeme fungování projektu po technické stránce. Vysvětlíme si základní principy použitých nastrojů, uděláme přehled dodávaného kódu na vysoké úrovni a představíme použité zdroje.

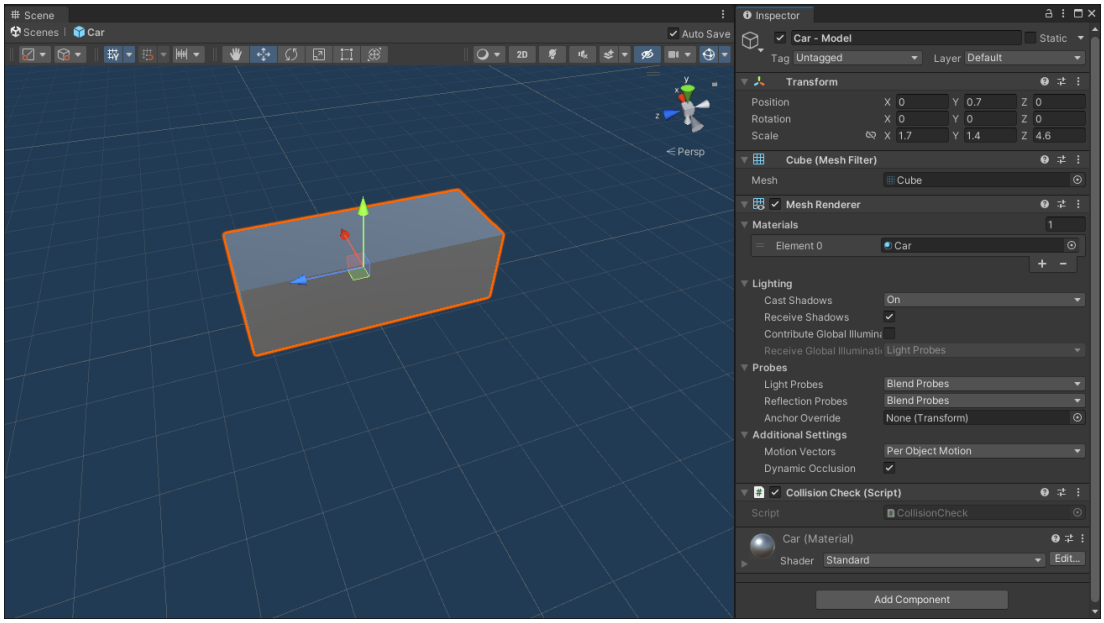
A.1.1 Použité nástroje

Engine Unity

Unity je framework, který byl použit jako základ programu. Je možné jej zdarma stáhnout na oficiálních stránkách: Unity Technologies (2023a).

Pro hlubší pochopení systému doporučujeme oficiální dokumentaci: Unity Technologies (2023c). V následujících řádcích zmíníme pouze několik základních konceptů, které byly v našem projektu použity:

- `Scene` neboli scéna je základní prostředí, ve kterém se přidávají všechny objekty. Pokud chceme v Unity cokoli vytvořit (3D objekty, uživatelské rozhraní, skripty), musíme to umisťovat do nějaké scény. Scéna v základu obsahuje 3D svět s kamerou a směrovým světlem.
- Obsah scény je tvořen `GameObjecty`, což můžou být právě různé postavy, objekty, kamery, světla, uživatelské prvky atd. Každý `GameObject` obsahuje komponentu `Transform` (viz níže). Objekty jsou ve scéně uspořádané v hierarchické struktuře.
- Jak jeden konkrétní objekt vypadá a chová se, poznáme dle jeho `Component`. Komponenta připnutá k objektu může být všelijaká, ale nejčastěji se setkáme s následujícími:
  - `Transform` zahrnuje informace týkající se 3D světa - pozici, rotaci a velikost („scale“). `Transform` se vždy chová lokálně. To znamená, že změna dat v transformu se přenese i na všechny potomky daného `GameObject`u a naopak neovlivní rodiče.
  - `MeshRenderer` umožňuje vykreslování modelu objektu.
  - `Collider` je součástí fyzikálního systému Unity. Umožňuje detekci kolizí mezi objekty.
  - `Rigidbody` umisťuje objekt do fyzikálního systému a dovoluje fyzikálním silám s ním pohybovat.
  - `Script` je `C#` třída, která musí dědit od vestavěné třídy `MonoBehaviour` a může obsahovat libovolný kód.
- `Material` určuje vzhled renderovaných modelů. Typicky se přidává do `MeshRenderer`u.



Obrázek A.1: `GameObject` v editoru – zde ukázka modelu auta

- `Prefab`em nazýváme možnost šablonovat objekty v Unity. `GameObject` typicky existuje jen ve scéně, ale právě pomocí `Prefab`u si ho můžeme uložit i se všemi hodnotami a dále ho využívat (například kopírovat a měnit jen nějaké hodnoty).

Objekty a jejich komponenty se dají před během nebo v průběhu zapínat a vypínat (povolovat a zakazovat). Jsou-li vypnuté, nezasahují nijak do simulace (jakoby neexistovaly). Na povolování a zakazování je navázána část kódu (a programátor může napojit další).

Uživatelské rozhraní (UI - User Interface) je také součástí scény a hierarchie objektů, ale UI objekty se lehce liší od klasických `GameObject`ů. Mají jiný transform a potřebují objekt s komponentou `Canvas`, pod který jsou umístěny. V práci jsme pro UI využili balíček `TextMeshPro`, který je vyvíjen přímo společností Unity.

Engine pro běh simulace (či hry v jiných případech) používá návrhový vzor *Game loop* (Nystrom, 2014). Prostředí se aktualizuje po časových úsecích, kdy se kontroluje input, pouští skripty, renderuje scéna atd. Fyzikální změny se odehrávají ve speciálních updatech, které jsou oddělené fixním časem.

Jazyk `C#` a platforma .NET

Aplikace byla naprogramována v jazyku `C#`, který je integrovaný do Unity. Jedná se o jazyk, který se kompiluje do CIL kódu virtuálního stroje .NET. V Unity s ním člověk pracuje především v podobě skriptů. Uživatelské skripty připnuté k objektům musí dědit od třídy `MonoBehaviour`, ale není problém je napojit na další vytvořený kód. `MonoBehaviour` poskytuje mnoho zajímavých funkcí, zde je několik příkladů:

- `Start` je metoda, která se spouští, když je skript poprvé povolen na objektu (často už na začátku programu/při vytvoření scény). Typicky se do

ní umisťuje inicializační kód, protože programátor nemá přístup přímo ke konstruktoru (resp. konstruktor se pouští skrz Unity a programátor nad tím nemá kontrolu).

- `Update` je metoda, která se pouští při každém (vykreslovaném) úseku simulace. Typicky se zde provádí logika programu. Doba mezi jednotlivými updaty není předem daná. Simulace se snaží updaty vykonávat co nejčastěji, ale než se pustí další, musí se vykonat všechny části toho předchozího a dohnat proběhlé fyzikální updaty.
- `FixedUpdate` je metoda, která se pouští pro fyzikální stránku simulace. Narozdíl od „obyčejného“ updatu se vykonává po pevné době. Pokud například vykreslování trvá dlouho, spustí se následně několik `FixedUpdatů` za sebou.
- `transform` je property, která poskytuje přístup k `Transformu` objektu.
- `Instantiate` je statická metoda, pomocí které se vytváří nové objekty do scény.

Unity také umožňuje kromě Updatů pouštět paralelně další procedury zvané `Coroutines`. Pomocí nich jde snadno naprogramovat čekání před nějakou akcí.

Odkazy na objekty přítomné ve scéně jde skriptům vkládat předem přímo v editoru – proto bychom při inicializaci neměli spoléhat na konstruktor, ale až na metodu `Start`, kdy máme jistotu přítomnosti těchto odkazů.

A.1.2 Otevření projektu

Projekt byl vyvíjen ve verzi Unity `2021` a přiložené soubory jsou specificky ve verzi `2021.3.24f1`, ale velmi pravděpodobně bude po převedení fungovat bez problémů i v novější verzi.

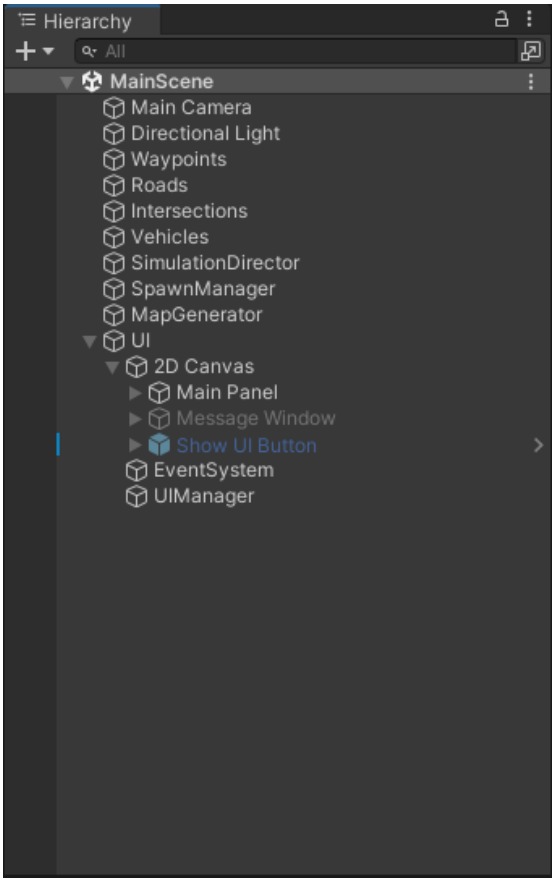
Po otevření projektu se dost možná v Unity objeví nějaká chybová hlášení související s importem. Je potřeba aplikaci restartovat a projekt otevřít ještě jednou. Potom by mělo vše fungovat bez problémů. Následně stačí otevřít scénu `MainScene` ze složky `Scenes` (není-li otevřená) a projekt lze začít prohlížet a upravovat v prostředí Unity. Také je možné si simulaci spustit rovnou ve vývojovém prostředí pomocí šipečky nahoře uprostřed obrazovky.

Vygenerování spustitelné aplikace se vykonává pomocí menu `Build Settings` (dá se nalézt v horním liště po kliknutí na záložku `File`).

A.1.3 Využití Unity v simulaci

Výše jsme popsali engine Unity na obecné rovině. Nyní si povíme, jak příslušné prvky využívá naše aplikace. Pro veškeré chování máme pouze jednu scénu - `MainScene`. Na obrázku (A.2) si můžeme prohlédnout její základní hierarchii.

`Main Camera` a `Directional Light` jsou základní objekty Unity, které používáme standardním způsobem. `Waypoints`, `Roads`, `Intersections` a `Vehicles` slouží víceméně jako složky, neboť jim jako děti ukládáme za běhu jednotlivé prvky simulace: (postupně) waypointy pro navigaci, modely silnic, objekty křižovatek a samotná vozidla.



Obrázek A.2: Hierarchie scény našeho projektu

`SimulationDirector`, `SpawnManager` a `MapGenerator` jsou pouze nosiči skriptů se stejnými jmény, které budeme více popisovat v sekci A.1.4. Asi nejsložitější část tvoří `UI`, proto jednotlivým prvkům v rámci `2D Canvas`u věnujeme samostatnou podkapitolku (A.1.3). Zde zmíníme pouze to, že objekt `UIManager` slouží také jako nositel skriptu a `EventSystem` obsahuje stejně pojmenovanou komponentu, která je potřeba pro správné fungování UI.

`Prefaby` v projektu využíváme především pro UI, ale jsou zde dva navíc. `Car` je základ objektu pro vozidla, který obsahuje skript pro řízení a fyzický model v dítěti. `Car` se do simulace dostává automaticky pomocí kódu. `Intersection marker` slouží k zobrazení typu algoritmu křižovatky, proto obsahuje vlastní `Canvas` a na něm obrázek a tlačítko.

Uživatelské rozhraní

Uživatelské rozhraní obsahuje mnoho prvků, které lze vidět v menu po spuštění (nebo v náhledu v editoru). `Message Window` slouží k zobrazení chybových hlášek. `Show UI Button` je tlačítko, které slouží k zobrazení ostatních položek, pokud jsou skryté. Většina prvků se nachází pod objektem `Main Panel`, kde jsou hierarchicky uspořádané tak, aby se vůči sobě správně zarovnávaly.

Abychom nemuseli dopodrobna popisovat všechny položky, popíšeme si jednotlivé `Prefaby`, z kterých jsou prvky UI vytvořené:

- `Distribution Input Field` se generuje automaticky, když program vy-

tváří možnosti pro distribuci algoritmů. Objekt zahrnuje číselné pole a popisek.

- `General Button` představuje obecné tlačítko s textem.
- `General Label` představuje obecný popisek.
- `Heading` představuje nadpis – uprostřed a větším písmem.
- `Labeled Dropdown` představuje výběr z možností s popiskem.
- `Labeled Number Input Field` představuje číselný vstup s popiskem.
- `Labeled Toggle` představuje checkbox s popiskem.
- `Major Heading` představuje velký centrovaný nadpis.
- `Number Input Field` představuje číselný vstup (bez popisku).
- `Seed Generator` představuje víc objektů, které se používají pro nastavení seedů při náhodném generování. Zahrnuje v sobě číselný vstup s popiskem a checkbox s popiskem.

Pro jednotnost uživatelského rozhraní by se v příslušných případech měly používat tyto prvky. Výhodou `Prefab`ů je, že se dají upravovat hromadně.

A.1.4 Skriptové pokrytí

Bude následovat popis jednotlivých částí dodávaného kódu na středně vysoké úrovni. Rozebereme si, jak jednotlivé části fungují, neboť většina funkcionality se nachází právě v kódu, ne v prvcích Unity editoru. Detailnější popis tříd a jejich veřejných metod lze nalézt přímo v kódu, takže zde se zaměříme hlavně na to, jak třídy spolupracují.

Skripty jsou uložené ve složce *Scripts*. Skripty jsou uložené ve složce `Scripts`. Většina se nachází ve vlastní podsložce. Třída `SimulationDirector` slouží pro propojení dalších částí (UI, spawnování, generátor, ...). V ní najdeme řízení simulace. Samotný `SimulationDirector` zodpovídá jen za celkový běh. Stará se především o volání funkcí jiných částí a komunikuje s nimi přes příslušná rozhraní.

Navigace

Třídy týkající se navigace po městě můžeme najít ve složce `Navigation`. `Waypoint`y představují body, skrz které auta na své trase projíždí a jsou to `GameObject`y skutečně umisťované do scény. `Edge` propojuje vždy dva `Waypoint`y a existuje pouze v kódu. Při její konstrukci se dle pozice a směrů `Transform`ů `Waypoint`ů počítá přesná trasa (zda zatáčí/jede rovně/...). `Edge` je rozdělena na jednu či více `EdgePart`s. Jednotlivé `Edge` se v průběhu používají pro pamatování vozidel na nich jedoucích a také tyto informace předávají autonomním řidičům, aby si mezi sebou mohli udržovat rozestupy.

Při běhu simulace se k nalezení cesty z jednoho `Waypoint`u do druhého používá `IPathFinder`. Toto rozhraní je implementováno třídou `PathFinder`, která si uvnitř pomocí algoritmu A* spočítá nejkratší vzdálenosti mezi počátečními

a koncovými body. Implementace tohoto algoritmu pro ukládání nalezených uzlů používá haldu (`PriorityQueue`).

Generátor

Složka `Generator` obsahuje třídy, které se týkají části generování mapy. V základu se s touto částí komunikuje přes Interface `IMapGenerator`. Toto rozhraní implementuje hlavní třída `MapGenerator`.

Generování by šlo rozdělit do dvou částí – nejprve se vše plánuje a zajišťuje se správnost vybraného počtu silnic apod. K plánování se používá třída `GeneratorCity`, která si vnitřne vytvoří mřížku křižovatek a výjezdů z města. Tato třída si počítá počty jednotlivých silnic na místech, kterými povedou.

Druhou část představuje samotné generování, když už víme, jak má město vypadat. Generátor vytváří jednotlivá místa, která dědí od abstraktní třídy `GeneratorPlace`. Místa při volání konstruktoru nagenerují potřebné `Waypointy` a `Edge` pro následnou navigaci, která bude používaná při simulaci. Program obsahuje dva typy míst:

- `GeneratorStandardIntersection` představuje křižovatku s až 4 vstupními i výstupními směry (které jsou na sebe kolmé a tvoří pravidelný křížek).
- `GeneratorSimplePass` znamená jen několik `Waypointů` vedle sebe, ale i toto jednoduché místo se využívá při následném napojování.

Když už jsou místa vytvořená, generátor jim nakonfiguruje algoritmy dle daného nastavení. a nakonec se dle navigačních struktur vytvoří *meshe*, tedy polygonové modely, které budou zobrazeny v simulaci.

Vozidla

Kód týkající se vozidel nalezneme ve složce `Vehicles`. Rozhraní navenek pro vytváření aut tvoří `ISpawnManager`, který je implementován třídou `SpawnManager`. Ten po inicializaci začne běžet a sbírat data o hustotě (běží-li simulační čas) a vkládá nová auta do města. Používá k tomu data, která získává přes rozhraní UI části (viz níže). `SpawnManager` také opět auta odebírá, jakmile dokončí svou trasu. Vnitřně však objekty nemizí trvale a nevytváří se nové, ale stále se používá ta samá sada dle principu návrhového vzoru *Object pooling* (Freeman a Freeman, 2015).

Každý `GameObject` vozidla má jako komponentu `VehicleController`, který zařizuje ovládání, řízení a nastavování hodnot jako je rychlost, zrychlení, otočení kol, a tudíž se stará i o aktualizaci pozice v systému. Toto se děje v metodě `FixedUpdate`, neboť aktualizovat informace o pozici je potřeba pořád a nejde čekat na hlavní `Update`. Samotný `VehicleController` však nemá na starosti žádné rozhodování. O hodnoty si žádá svého „řidiče“ v podobě třídy `VehicleDriver`, který má za úkol dodat hodnoty zrychlení (plynu), brždění a otočení kol (natočení volantu).

`VehicleDriver` samotný je abstraktní třída, která má dva možné potomky:

- `HumanDriver` je pouze třída z testování, která se nyní nepoužívá. Bylo možné řídit auto pomocí klávesnice. Nyní by se však ovládání překrývalo

s ovládáním kamery, takže pro znovupoužití by bylo potřeba provést více zásahů do kódu.

- Naopak `ArtificialDriver` je jednou z klíčových tříd projektu. Jedná se o autonomního řidiče, který používá konstrukty z navigace (`Waypoint` a `Edge`), aby se pohyboval po systému. Žádné speciální senzory (nebo nějaké Unity ekvivalenty) neobsahuje. Umělí řidiči jsou také schopni komunikovat s dodávanými algoritmy křižovatek. Umělý řidič si sám pomocí naprogramovaných funkcí řeší následující části jízdy: zpracování pokynů křižovatky, kontrola maximální rychlosti úseku silnice, kontrola správného úhlu v zatáčce, změna rychlosti na požadovanou hodnotu, udržování bezpečné vzdálenosti od předcházejícího vozidla.

Třída `CollisionCheck` se připojuje k modelům aut (které tím získají `Rigidbody` a `Collider`). Za běhu sleduje, zda do sebe vozidla nenarazila, když neměla.

Křižovatky

Složka `Intersections` obsahuje třídy týkající se algoritmů křižovatek. Konkrétní implementace dědí od třídy `Intersection`, která sama je potomkem `MonoBehaviour`, takže umožňuje použití výše zmíněných funkcí této třídy. `Intersection` také zahrnuje statickou část, která pomocí *Reflection* drží přehled o použitých algoritmech. Pokud programátor chce přidat algoritmus křižovatky do systému, musí vytvořit třídu dědící od `Intersection` a označit ji atributem `ShowAlgorithm`. `Intersection` také zařizuje sběr statistických dat.

Zde jsou existující implementace:

- `StopSignIntersection` představuje stopkovou křižovatku. Uvnitř je použitá společná fronta pro přijíždějící vozidla. Tento algoritmus nastavuje řidičům jako `StopPoint` místo vjezdu do křižovatky.

- `TrafficLightIntersection` představuje semaforovou křižovatku. Seznamy vozidel na jednotlivých pruzích si uchovává pomocí struktury `Deque`, která umožňuje kromě přidávání a odebírání z konce i začátku v konstantním čase také indexovat jednotlivé položky. `TrafficLightIntersection` také nastavuje řidičům `StopPoint` na vjezd u křižovatky. Pro střídání světelných fází byly použity `Coroutines` se speciálním čekáním, které čeká po fixních časových úsecích.

- `AutonomousIntersection` zprostředkovává chování AIM křižovatek. Využívá k tomu další 2 třídy z této sekce: `ConflictPoint` pro konfliktní body na cestě a speciální případ `FirstConflictPoint`. Fungování algoritmu je popsáno v 2.5.2. Autonomní křižovatka řidičům posílá přímo informace o rychlostech a čekání pomocí jejich `TargetSpeeds` property.

- `ChangeLaneIntersection` je v současnosti nepoužívána třída, která představovala pokus o vyřešení přejezdů mezi pruhy. Na začátku spočítá, které pruhy se navzájem blokují (kříží se či slučují) a následně funguje jako stopka, akorát umožňuje průjezd více vozidlům naráz, pokud se vzájemně neblokují. Pro účely změny pruhů se toto řešení ukázalo jako příliš pomalé (více se lze dočíst v 2.6).

Měření dat

Ve složce `Statistics` se nachází třídy sloužící k uchovávání naměřených dat a jejich následnému výpisu do souboru. `IStatisticsCollector` slouží k získání dat. Tento soubor také obsahuje readonly třídy, které přímo slouží k uložení dat pro jednotlivé části. `StatisticsCollector` implementuje toto rozhraní, uvnitř si ukládá kolekce naměřených tříd s daty a umožňuje také ukládat hustotu v simulaci za čas.

`IOutputSaver` se používá pro zápis do souboru. Rozhraní je implementováno třídou `OutputSaver`, která při použití vezme nasbíraná data a zapíše je pomocí standardních IO konstruktů do `.csv` souborů.

Uživatelské rozhraní

Kód stojící za uživatelským rozhraním najdeme ve složce `UI`. Hlavní třídou je `UIManager`, která v sobě obsahuje odkazy na všechny prvky v menu a zajišťuje jejich správné fungování. `UIManager` se tedy stará o správnost dat ve vstupech, aktualizaci UI, předávání dat do dalších částí programu a pouštění příslušných funkcí navázaných na tlačítka.

`UISettings` je datová třída, která slouží pro ukládání a načítání položek UI nastavení. Soubor je rozdělen do několika podtříd. Jako vnější formát souborů byl použit `JSON`. `UIManager` umožňuje pomocí této třídy ukládat a načítat položky menu. Pro dialogová okna při ukládání jsme použili externí balíček `NewtonsoftJson`.

`IGeneratorInput`, `ITimeInput` a `IVehicleInput` jsou rozhraní, která jsou všechna implementovaná pomocí `UIManager`u. Slouží k předávání důležitých informací jiným částem kódu.

`Intersection marker` je pomocná třída k dříve zmiňovanému ukazateli typu křižovatky.

`UIHotkeySelect` je externě získaný kód, který zajišťuje překlikávání pomocí klávesnice mezi jednotlivými prvky UI.

Pomocné skripty

Některé samostatné třídy jsme neumisťovali do žádné z podsložek, takže tady je výčet těch, které jsme ještě nezmínili:

- `CameraController` slouží k ovládání objektu kamery pomocí klávesnice a myši. Jedná se o převzatý kód.
- `Config` umožňuje načítání základních programových konstant a důležitých čísel z `JSON` souboru. Detailní popis jednotlivých položek se nachází v uživatelské dokumentaci (A.2.7).
- `Utility` je statická třída, která poskytuje řadu pomocných funkcí pro práci s geometrií, matematikou či prvky Unity.

A.1.5 Použité externí zdroje

Během práce na programu jsme použili několik externích volně dostupných balíčků. Pokud se jednalo pouze o drobné kusy kódu, uvádíme odkaz přímo na

příslušných místech ve zdrojových souborech. Zde ještě představíme výčet větších celků, které jsme využili:

- `NewtonsoftJson` zajišťuje serializaci a deserializaci `JSON` objektů.
- `Nito.Collections.Deque` poskytuje datovou strukturu `Deque`, která umožňuje přístup k prvkům, přidávání i odebírání na začátek i konec, a to vše v konstantním čase.
- Kód `PathCreatoru` byl převzatý pro tvorbu 3D modelů silnic.
- `Priority Queue` poskytuje datovou strukturu `PriorityQueue`, tedy haldu, která se hodí v algoritmu A* při hledání cest.
- `StandaloneFileBrowser` umožňuje funkcionalitu dialogových oken pro prohlížení souborů v úložišti.
- `TextMeshPro` poskytuje prvky pro uživatelské rozhraní (text, tlačítka, ...).

Dále samozřejmě používáme mnoho standardních knihoven .NETu, `C#` a Unity, ale ty zde nebudeme vyjmenovávat.

Jako ikony jednotlivých křižovatek jsme využili následující obrázky ze stránky Flaticon (Freepik Company S.L., 2023):

Obrázek A.3: Ikona pro autonomní křižovatku

Obrázek A.4: Ikona pro semaforovou křižovatku

Obrázek A.5: Ikona pro stopkovou křižovatku

Obrázek A.6: Ikona pro neznámou křižovatku

A.2 Uživatelská dokumentace

Zde je popsáno jak používat přiloženou aplikaci pro provádění vlastních experimentů a testování.

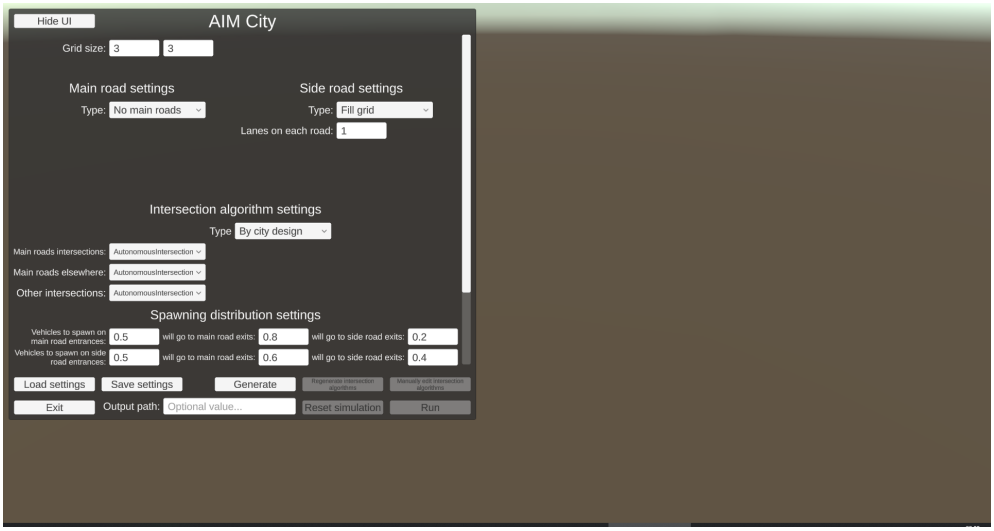
Uživatel po otevření programu má možnost nastavit si mnoho možností, které zde popíšeme, a následně spustit simulaci. Po úspěšném doběhnutí se uloží výsledná data, která si uživatel může sám dále zpracovat.

A.2.1 Spuštění

Ke spuštění je potřeba mít všechny potřebné vygenerované soubory, které jsou dodávány v příloze této práce.

Projekt se dá spustit v systému Windows pomocí souboru `AIMCity.exe`.

Po zapnutí se objeví prostředí simulace (prázdné) a uživatelské menu, kde je potřeba učinit výběr různých nastavení (lze vidět na obrázku A.7).



Obrázek A.7: Aplikace po spuštění

Aplikaci jde kdykoli ukončit tlačítkem *Exit*.

A.2.2 Nastavení

V menu jsou ovládací prvky sloužící k vygenerování města a následnému spuštění simulace.

Do všech číselných polí v menu jde zadávat pouze hodnoty v určitém rozsahu. Pokud uživatel nechá nějaké pole prázdné, automaticky se do něj doplní přednastavená hodnota.

Bude následovat popis jednotlivých možností a obsažených prvků.

Náhodné generátory

Pokud se uživatel rozhodne nějakou část simulace generovat náhodně, zobrazí se mu pro to nastavení náhodného generátoru. Náhodné generování probíhá na principu *seedů* - pro daný seed (číslo) se vždy vygenerují ty samé výsledky. Seed

je možné nechat generovat automaticky (*Regenerate seed on generation*), pak se pokaždé nastaví nový sám, nebo se dá tato možnost vypnout a číslo ručně zadat.

Základní nastavení

Mezi základní nastavení patří výběr velikosti města. Město se generuje jako mřížka křižovatek a do polí *Grid size* se vkládá šířka a výška města (maximální počet silnic/křižovatek v jednom směru).

Hlavní silnice

V sekci *Main road settings* si uživatel může zvolit nastavení pro *hlavní silnice*.

Hlavní silnice jsou jeden ze dvou typů silnic, které město tvoří. Typicky vytváří hlavní infrastrukturu a jsou větší (to se dá nastavit). Toto rozlišení se objevuje v dalších nastaveních.

Základem této sekce je typ generování hlavních silnic:

- **No main roads** znamená, že ve městě žádné hlavní silnice generovány nebudou.
- **Standard** vytvoří daný počet horizontálních (*Horizontal roads*) a vertikálních (*Vertical roads*) silnic z jedné strany na druhou. Silnice ve stejném směru mezi sebou budou na síti co nejpravidelněji rozmístěné. Dále se dá zvolit, zda mají být pouze jednosměrné – *One way roads* a kolik má být na každé silnici v jednom směru pruhů – *Lanes on each road*.
- **Random** vygeneruje daný počet (*Roads count*) silnic náhodně po mřížce. Stejně jako v předchozím bodě se dá vybrat počet pruhů a jednosměrnost. Náhodné generování funguje tak, že se pro každou silnici vyberou dva nespojené body – na začátku pouze kraje mřížky, následně i dříve vygenerované silnice, a ty se propojí novou silnicí. Pokud jsou vybrány jednosměrné silnice, pak se oba směry řeší zvlášť.

Vedlejší silnice

Vedlejší silnice tvoří druhý typ silnic oproti *hlavním*. Také se dá vybrat typ jejich generování:

- **Fill grid** doplní mřížku silnicemi všude, kde nejsou hlavní silnice. Počet pruhů silnic se vybere pomocí *Lanes on each road*. Pokud se někde vygenerovala hlavní silnice pouze v jednom směru, vedlejší v tom opačném generována nebude.
- **Random** funguje stejně jako generování náhodných hlavních cest (akorát tyto budou vedlejší s vybranými nastaveními).

Algoritmy křižovatek

Tato sekce slouží pro výběr *algoritmů* jednotlivých křižovatek. Ty jsou v dodávaném základu následující:

- *AutonomousIntersection* – autonomní křižovatka s plánováním tras.
- *StopSignIntersection* – stopkou řízená křižovatka.
- *TrafficLightIntersection* – semafory řízená křižovatka.

Algoritmy jde generovat několika způsoby:

- **By city design** se řídí rozvržením typů silnic po městě. Lze vybrat, jaké algoritmy se mají objevovat v těchto místech: *Main roads intersections* – vzájemné křížení hlavních silnic, *Main roads elsewhere* – křížení hlavních silnic s vedlejšími, a *Other intersections* – zbylé křižovatky (pouze vedlejší silnice).
- **By traffic** pro vygenerovanou mřížku dle distribuce (A.2.2) spočítá, kudy budou vozidla jezdit a jaká bude jejich hustota. Následně lze vybrat algoritmy pro: *First _ most busy intersections* – daný počet nejvytíženějších křižovatek, *Last _ least busy intersections* – daný počet nejméně vytížených křižovatek a *Other intersections* – ostatní křižovatky.
- **Random** – tato možnost generuje náhodně dle zadaného rozvržení – *<Jméno algoritmu> probability* pro každý algoritmus. Zadání 0 do tohoto pole znamená, že tento algoritmus se neobjeví. Jinak lze zadat libovolná nezáporná čísla a výsledné rozdělení bude v těchto poměrech.

Distribuce objevování

Sekce *Spawning distribution settings* ovlivňuje, kde se vozidla budou při běhu objevovat. Nejprve je třeba v nejlevějších polích (*Vehicles to spawn on main road entrances* a *Vehicles to spawn on side road entrances*) napsat rozdělení pro vstupy hlavních a vedlejších silnic. Použije se poměr těchto čísel. Pokud je některé z nich 0, další možnosti v řádku se zablokují (protože se žádná vozidla na daných místech objevovat nebudou). Jinak lze nastavit, zda daná vozidla budou směřovat na místa, kde mřížku opouští hlavní silnice (*will go to main road exits*), nebo vedlejší silnice (*will go to side road exits*). Opět se použije poměr těchto dvou čísel.

Nastavení provozu

Předchozí nastavení se týkala především generátoru města. Sekce *Traffic settings* nastavuje hodnoty, které jsou důležité po spuštění simulace:

- *Vehicles to simulate* – počet vozidel, kterého se simulace pokusí dosáhnout postupným přidáváním.
- *Time scale* – jakou rychlostí má simulace běžet. 1 znamená v reálném čase, nižší hodnoty způsobí zpomalení (například 0.1 bude odpovídat zpomalení na desetinu) a vyšší hodnoty naopak zrychlení běhu času (například 10 bude odpovídat zrychlení na desetinásobek). Při příliš vysokých hodnotách však není zaručeno, že systém bude schopen rychlost udržet a je možné, že dojde k umělému zpomalení. Toto číslo však nemá na výsledná data žádný vliv.

- *Total time to run* – jak dlouho má simulace běžet (kolik vteřin). Reálná doba běhu bude ovlivněna hodnotou *Time scale*.
- *Traffic seed* – náhodný generátor pro objevování vozidel dle A.2.2.

A.2.3 Ukládání a načítání

Po kliknutí na tlačítko *Save settings* lze všechny zadané hodnoty v nastavení uložit do externího souboru – zobrazí se systémový dialog pro ukládání souboru.

Tlačítkem *Load settings* lze (pomocí systémového dialogu) dříve uložené nastavení opět načíst do programu.

A.2.4 Tlačítka pro generátor

Pro potvrzení zadaných dat ke generování je potřeba stisknout tlačítko *Generate*. Následně se vygeneruje mřížka silnic dle nastavení. Tento proces může v závislosti na rozsahu nějakou dobu trvat. Poté se objeví město s křižovatkami. Algoritmy křižovatek jsou znázorněny ikonou nad nimi.

Tlačítko *Regenerate intersection algorithms* nechá vygenerovanou mapu stejnou, ale spustí znovu část generátoru pro přiřazení algoritmů.

Pomocí tlačítka *Manually edit intersection algorithms* se dají ručně měnit algoritmy jednotlivých křižovatek. Po stisku ikony nad křižovatkami zesvětlají - stají se z nich tlačítka, pomocí kterých se dá proklikávat skrz možné algoritmy. Jakmile je úprava hotová, je potřeba stisknout tlačítko *Stop editing*.

A.2.5 Běh simulace

Simulace se spustí kliknutím na tlačítko *Run*. Tím se zablokují některá jiná tlačítka.

Při spuštění simulace se také uloží hodnoty nastavené v menu, které se později zapíšou do výstupního souboru. Pokud uživatel za běhu něco v menu upraví (například v rámci přípravy na další běh), s uloženými daty se nic nestane.

Program se přepne zpět do režimu nastavování po doběhnutí simulace. Případně jde simulaci úplně přerušit a zrušit záznam dat pomocí tlačítka *Reset simulation*.

Menu s nastaveními se dá kdykoli skrýt pomocí tlačítka *Hide UI*.

Pohyb simulací

Kamera, kterou uživatel vidí simulaci, se dá kdykoli volně ovládat následujícím způsobem:

- Klávesy *WASD* slouží k pohybu dopředu a do stran.
- Klávesa *mezerník* slouží k pohybu nahoru a klávesa *Ctrl* k pohybu dolů.
- Klávesa *Shift* pohyb zrychlí.
- Při držení *pravého tlačítka myši* se kamerou dá otáčet.

A.2.6 Výstup

Po doběhnutí simulace se výsledky uloží do výstupních souborů, které budou umístěny ve společné složce. Název složky (či dokonce relativní nebo absolutní cesta) lze zadat do vstupního pole *Output path*. Zůstane-li pole prázdné (nebo je v něm neplatná hodnota), bude složka pojmenována „AIMCity“ ve spojení se současným datem a časem.

Soubory s daty se ukládají ve formátu `csv`, jako oddělovač se používá středník. Zde je seznam jednotlivých souborů a jejich obsah:

- `GeneralData.csv` obsahuje obecné výsledky simulace. Většina přítomných dat by šla dopočítat z jiných dat výstupu, takže se jedná spíš pro usnadňující soubor při zpracování. Také se zde nachází souhrnný přehled přítomných algoritmů křižovatek.
- `IntersectionsData.csv` obsahuje naměřená data každé křižovatky - jaký byl použit algoritmus, kolik přes ně procestovalo vozidel atd.
- `TrafficDensityOverTime.csv` obsahuje přehled měřeného provozu za jednotky času.
- `VehiclePassages.csv` je největší ze souborů, protože obsahuje informace o všech průjezdech skrze město. Ke každému jde dohledat ujetou vzdálenost, strávený čas, počet projetých křižovatek atd.
- `UISettings.json` obsahuje uložené hodnoty uživatelského rozhraní, jako kdybychom použili ukládání popsané v části A.2.3.

A.2.7 Speciální konfigurace

Systém obsahuje dvě různé konfigurace v podobě `JSON` souborů. Jednu pevnou, jejíž hodnoty by se neměly moc měnit. a jednu pro nastavení, která se ukládá po každém běhu a dá se uložit či načíst pomocí tlačítek (popsáno v sekci A.2.3). Složky simulace obsahují i základní nastavení v souboru `DefaultAIMCitySettings.json`, které se načte při spuštění simulace. Dále tuto konfiguraci nebudeme popisovat, neboť jsou zde uložené pouze hodnoty, které se mapují na data v uživatelském rozhraní.

Zajímavější je první konfigurace, ze které se načítají některé základní hodnoty fungování programu. Najdeme ji v souboru `AIMCityConfig.json`. Při základní práci není potřeba hodnoty měnit, ale kdyby uživatel měl zájem zasahovat i do vnitřního fungování simulace, zde si může přečíst jejich popis (všechny uvedené hodnoty se zadávají reálnými čísly – pokud se nejedná o číslo, které logicky může být jen celé (počet pruhů, počet vozidel, ...)):

- `RoadSettings` obsahuje nastavení ke generování silnic.
  - `RoadWidth` představuje šířku silnice v metrech.
  - `DistanceBeforeIntersection` představuje vzdálenost v metrech mezi křižovatkou a zónou pro změnu pruhů.

  - `MapEntranceLength` představuje vzdálenost v metrech mezi okrajem mapy a zónou pro změnu pruhů.

- `IntersectionGenerationSettings` obsahuje nastavení týkající se vzdáleností waypointů v rámci křižovatky.

  - `BasicWaypointDistance` představuje základní vzdálenost waypointů od středu křižovatky.
  - `NextWaypointDistance` představuje vzdálenost, která se přidává k té základní dle počtu pruhů.

- `AutonomousIntersectionSettings` obsahuje nastavení týkající se autonomních křižovatek.

  - `WaitTimeBeforeNextRequest` představuje dobu čekání v sekundách, než se křižovatka pokusí znovu najít rezervaci vozidlu, které ji dříve nedostalo.
  - `SmallTimeBuffer` představuje minimální časový úsek v sekundách, který se vkládá mezi dvě okna rezervací na jednom konfliktním bodu. Mělo by to být dostatečně malé číslo.
  - `MinConflictPointSpeed` představuje minimální rychlost v metrech za sekundu, kterou vozidla mohou cestovat skrze konfliktní body autonomní křižovatky.
  - `EntryWaypointBufferSize` představuje velikost přidaného bufferu (poměrově vůči velikosti vozidla) u waypointů, které jsou na vstupu do křižovatky (kde se pruh rozděluje do cest skrz křižovatku).
  - `MinInnerWaypointBufferSize` představuje minimální velikost přidaného bufferu pro body uvnitř křižovatky (minimum pokud dané dvě cesty jsou kolmé).
  - `MaxInnerWaypointBufferSize` představuje maximální velikost přidaného bufferu pro body uvnitř křižovatky (maximum pokud dané dvě cesty jdou co nejvíc proti sobě).
  - `LeaveWaypointBufferSize` představuje velikost přidaného bufferu u waypointů, které jsou na výstupu z křižovatky (kde se cesty skrz křižovatku slučují do pruhu).
  - `FarWaypointBufferSize` představuje velikost přidaného bufferu pro vzdálený (virtuální) bod za křižovatkou.

- `TrafficLightIntersectionSettings` obsahuje nastavení týkající se semaforových křižovatek.

  - `GreenLightTime` představuje dobu v sekundách, po kterou je v jednom směru zelená.
  - `YellowLightTime` představuje dobu v sekundách, kdy nejsou nová vozidla vpouštěná do křižovatky a čeká se na uvolnění křižovatky před vystřídáním pruhů.

  - `OffsetSeed` představuje seed pro náhodný generátor, který semaforovým křižovatkám nastavuje zpoždění, aby nebyly synchronizované, a vybírá jim první pruh.

- `StopSignIntersectionSettings` obsahuje nastavení pro stopkové křižovatky.

  - `WaitTimeBeforeNextAttempt` představuje dobu čekání ve vteřinách, jak dlouho se čeká, než se křižovatka pokusí pustit další auto ve frontě, pokud je výjezdní pruh plný.

- `GridSettings` obsahuje nastavení pro generátor mřížky města.

  - `MaxRowsColumns` udává maximální počet řádek či sloupců v mřížce.
  - `MaxLanes` představuje maximální počet pruhů na silnicích.

- `VehicleSettings` obsahuje nastavení týkající se jednotlivých vozidel.

  - `VehicleLength` udává délku vozidla v metrech.
  - `MaxRotationAngle` udává do jakého maximálního úhlu ve stupních může vozidlo natočit virtuální kola.
  - `TurningSpeedCoefficient` představuje koeficient, podle kterého se omezuje maximální rychlost v zatáčkách. Čím vyšší bude tato hodnota, tím vyšší bude povolená rychlost.
  - `MaxSpeed` udává maximální rychlost vozidla v metrech za sekundu.
  - `Acceleration` udává maximální hodnotu zrychlení vozidla v metrech za sekundu na druhou.
  - `BrakeDeceleration` udává maximální hodnotu zpomalení vozidla při brždění v metrech za sekundu na druhou.
  - `SafetyGap` představuje vzdálenost v metrech, která je dodržována mezi dvěma vozidly jedoucími za sebou.

- `VehicleUISettings` obsahuje nastavení pro sekci uživatelského rozhraní, které se týká vozidel a běhu simulace.

  - `MaxVehicles` představuje maximální hodnotu pro nastavitelný počet vozidel v simulaci.
  - `MinTimeScale` představuje minimální hodnotu, jakou jde násobit čas, a tudíž zpomalit simulaci.
  - `MaxTimeScale` představuje maximální hodnotu, jakou jde násobit čas, a tudíž zrychlit simulaci.

- `StatisticsSettings` obsahuje nastavení pro měření statistických dat.

  - `TrafficDensityCollectingInterval` představuje interval v sekundách mezi okamžiky, kdy simulace zachytí současný počet vozidel a počet dokončených průjezdů od posledního měření.