České vysoké učení technické
v Praze

Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Autor: Ondřej Milenovský

Katedra kybernetiky

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 12. 5. 2011 .....

.....
podpis

Poděkování

Děkuji bývalým kolegům, kteří se mnou pracovali na projektu Vojtěchovi Eliášovi za spolupráci v začátcích projektu, Liboru Wagnerovi hlavně za import map a Radkovi Holému za psaní testů. Dále bych chtěl poděkovat lidem z ATG Antonínovi Komendovi za implementační poradenství a Ondřejovi Vaňkovi a Ondřejovi Hrstkovi za spolupráci na Google Earth vizualizaci. Nakonec bych chtěl poděkovat vedoucímu celého projektu Michalovi Jakobovi za řízení projektu a pomoc při psaní této práce.

Abstrakt

Tato práce popisuje framework simulující pohyb osob a vozidel ve městě. Popisuje navržený agentní model, celé prostředí, různé objekty a entity včetně použitých dat a interakce. Jeden z hlavních přínosů je model chování agenta implementovaný pomocí konceptu aktivit. Popsán je obecný model aktivit a dále konkrétní agenti v simulaci. Druhý hlavní přínos je v multimodálním plánovači, který plánuje nejrychlejší cestu po městě pěšky a s využitím prostředků MHD podle jízdních řádů. Popsány jsou různé heuristiky pro plánovač a nakonec jsou porovnány v experimentech. Dále je popsána Google Earth vizualizace, která vykresluje objekty simulace přímo do Google Earth.

Abstract

This paper describes a framework for simulating movement of people and vehicles in city. It describes proposed agent model, whole environment, objects and entities including used data and interaction. One of the main benefits is behavior model implemented using concept of agent activities. Described is a generic model of activities and then specific agents in the simulation. The second major contribution is in the multimodal planner which plans the fastest path through city by foot and using public transportation according to timetables. Described are various heuristics for the planner and finally compared in the experiments. Also is described Google Earth visualization that draws simulation objects directly to Google Earth.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Ondřej Milenovský**

Studijní program: Otevřená informatika (magisterský)

Obor: Umělá inteligence

Název tématu: Agentní simulace městské mobility

Pokyny pro vypracování:

1. Seznamte se s problematikou agentních simulací (městské) mobility.
2. Seznamte se se simulačními platformami A-lite a AgentPolis.
3. Navrhněte model městské mobility vycházející z konceptu agenta-občana.
4. Navrhněte uvažovací mechanismy agenta-občana související se simulací mobility (denní cyklus potřeb, plánování pohybu atd.)
5. Implementujte navržený model a uvažovací algoritmy pomocí platforem A-lite a AgentPolis Empiricky analyzujte klíčové vlastnosti implementované simulace.

Seznam odborné literatury:

Bonabeau, E.: Agent-based modeling: methods and techniques for simulating human systems. Proceeding of the National Academy of Sciences 99(3): 7280-7287, 2002.

Balmer, M.; Cetin, N.; Nagel, K. & Raney, B.: Towards truly agent-based traffic and mobility simulations. Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 1: 60-67, 2004.

Balmer, M.; Meister, K.; Rieser, M.; Nagel, K. & Axhausen, K.: Agent-based simulation of travel demand: Structure and computational performance of MATSim-T. 2nd TRB Conference on Innovations in Travel Modeling, 2008.

Bernhardt, K.: Agent-Based Modeling in Transportation. Artificial Intelligence in Transportation, E-C113:72-80, 2007.

Krajzewicz, D.; Hertkorn, G.; Rössel, C. & Wagner, P.: SUMO (Simulation of Urban MObility)-an open-source traffic simulation, Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM20002), 183-187, 2002.

Vedoucí diplomové práce: Ing. Michal Jakob, Ph.D.

Platnost zadání: do konce letního semestru 2011/2012

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 25. 3. 2011

Obsah

Seznam obrázků **x**

Seznam tabulek **xii**

1 Úvod **1**

2 Agentní simulace **3**
2.1 Agentní model 3
2.2 Simulace dopravy 4
2.2.1 Použití agentního modelu na simulaci dopravy 5
2.2.2 Existující platformy na simulaci dopravy 5

3 Navržený model mobility **7**
3.1 Simulační model 7
3.1.1 Prostředí 7
3.1.2 Zdroje 8
3.1.3 Agenti 8
3.1.4 Akce agenta 9
3.1.5 Senzory agenta 10
3.1.6 Fronty na hranách 13
3.1.7 Parkování 13
3.2 Data 13
3.2.1 Mapa 14
3.2.2 Jízdní řády 14
3.2.3 Generování objektů a značek v mapě 17
3.3 Uživatelské rozhraní 17
3.3.1 A-lite vis 17
3.3.2 Google Earth 17
3.3.3 Statistiky 19
3.4 Výstup frameworku 25

4 Model chování agenta **26**
4.1 Struktura agenta 26
4.2 Navržený koncept aktivit 26
4.3 Konkrétní agenti a jejich aktivity 28
4.3.1 Dospělý občan 28
4.3.2 Věčný pracovník - MHD řidič 29
4.3.3 Aktivity pohybu 30
4.3.4 Ostatní aktivity 32

5 Navigace **33**
5.1 Multimodální plánovač 33
5.1.1 Motivace 33
5.1.2 Definice problému 33
5.2 Existující multimodální plánovače 34
5.3 Implementovaný plánovač 35
5.3.1 Preprocesing 35
5.3.2 Plánovač 35
5.4 Testované heuristiky 37
5.4.1 Přímé heuristiky 37
5.4.2 Oříznutí elipsou 38
5.4.3 Násobení stanic a ostatních uzlů 38
5.4.4 Přepínač heuristik 38

6 Implementace **39**
6.1 Základ frameworku 40
6.2 Vnitřní reprezentace jednotlivých objektů 40
6.2.1 Storage 40
6.2.2 Storage na fronty 41
6.2.3 Storage na semafory 42
6.2.4 Storage na statistiky 42
6.2.5 Akce a senzory 43
6.2.6 Grafy 44
6.2.7 Entity 46
6.3 Plánovač A* 47
6.4 Google Earth 48
6.4.1 Stručně o KML 49
6.4.2 Proud dat ze simulace do Google Earth 49
6.4.3 Implementace AgentPolis pro Google Earth 50

7 Evaluace **54**
7.1 Evaluace celého frameworku 54
7.1.1 Konfigurace testů 54
7.1.2 Výsledky 54
7.1.3 Diskuze 55
7.2 Evaluace vizualizace 56
7.2.1 Konfigurace testů 56
7.2.2 Výsledky 57
7.2.3 Diskuze 57
7.3 Evaluace plánovače 58
7.3.1 Nastavení testů a metriky 58
7.3.2 Naměření metrik všech heuristik 60
7.3.3 Základní porovnání heuristik 63
7.3.4 Závislost optimality na výpočetním času 63
7.3.5 Souhrnné porovnání heuristik 66
7.3.6 Diskuze 66

8 Závěr **68**

Literatura **72**

A Obsah přiloženého DVD **I**

Seznam obrázků

2.1 Interakce agenta s prostředím 4

3.1 Ukázka alite vis, černě silnice, šedivě koleje, červeně občané, zeleně MHD vozidla, modře osobní vozidla, dále semafory . . . 18
3.2 Ukázka Google Earth - hustoty: červeně občané, modře osobní vozidla, zeleně MHD vozidla 19
3.3 Ukázka Google Earth detail - zobrazené jednotlivé entity, grafy silnic a firmy 20
3.4 Ukázka Google Earth popis entity - zobrazen aktuální stav rozhodovacího mechanismu 21
3.5 Ukázka statistik - seznam záznamů reportování jednoho agenta 22
3.6 Ukázka statistik - graf poměr stavů občanů 23
3.7 Ukázka statistik - popis entit stejně jako v GE 24

6.1 Proces vytváření událostí 40
6.2 Proces provádění akcí 44
6.3 Provolávání sensingů po změně ve storagi 45
6.4 Proces volání sensingů 45
6.5 Tok dat v Google Earth 50
6.6 Ukázka zobrazených objektů v GE (vozidla, agenti, firmy, linky metra, silnice, grafy, info) 53

7.1 Závislost výpočetního času a počtu událostí za sekundu na počtu agentů (logaritmická osa x) 56
7.2 Počet snímků za vteřinu jednotlivých vizualizátorů v závislosti na počtu agentů 58
7.3 Struktura testované heuristiky, *dist* je přímá vzdálenost aktuálního uzlu od cíle 61
7.4 Porovnání heuristik Hopt, Hel, Hst a Hth 63
7.5 Porovnání heuristik Hw5, Hw10 a Hw15 64
7.6 Porovnání optimality a poměru expandovaných uzlů heuristik H1 až H8 65

7.7 Porovnání čtyř nejvýznamnějších heuristik 66

Seznam tabulek

3.1 Seznam akcí agenta (akce obsahující nastupování zahrnuje i vystupování) 9
3.2 Seznam senzorů agenta 11
3.3 Tabulka počtu importovaných dat 14
3.4 Vygenerovaný jízdní řád - všechny tramvaje a jízdní řád z idosu - tramvaj 16, všední den 16

4.1 Seznam aktivit agenta 28
4.2 Tabulka počátečních položek v paměti občana 29

6.1 Obyčejné storage 41
6.2 Událostní storage 42
6.3 Typů záznamů 43

7.1 Konfigurace simulace 55
7.2 Výkonnostní výsledky frameworku za jeden simulační den . . . 55
7.3 Výsledky vizualizátorů - počet snímků/s 57
7.4 Parametry struktury heuristik 61
7.5 Poměr uzlů nalezené cesty expandovaným uzlům Hopt 62
7.6 Metriky základních heuristik 62
7.7 Metriky jedné heuristiky s různými parametry 62

Kapitola 1

Úvod

Tato práce popisuje implementovaný framework AgentPolis (původně UrbanSim) simulující pohyb lidí a vozidel ve městě. Simulace je agentní tedy na mikro úrovni. Simuluje každodenní potřeby lidí ve městě včetně používání osobních vozidel nebo městské hromadné dopravy. Modelování dopravy je důležitý problém, jehož vyřešení umožní testovat různé možnosti zlepšování dopravy ve městech. Framework je stále ve vývoji, dost částí není dokončených, jiné by se daly udělat lépe. Značná část práce je zaměřená na multimodální plánovač, který plánuje trasu pěšky po cestách s využitím spojů MHD. Plánování trasy je v simulaci nejnáročnější proces, tedy musí být co nejrychlejší.

V 2. kapitole agentní simulace se stručně popisuje, co to je agentní přístup, jaké má výhody a nevýhody oproti ostatním přístupům. Dále popisuje, jakým způsobem je možno pomocí agentní simulace modelovat dopravu a popisuje tři existující platformy. Obsahuje i zdůvodnění, proč jsme žádnou existující platformu nepoužili jako základ, ale vytvořili vlastní.

3. kapitola popisuje navržený simulační model. Popisuje, z čeho se skládá prostředí, jaké jsou v něm objekty a entity a jakým způsobem spolu interagují. Dále se zaměřuje na vstupní data simulace jako například mapa, jízdní řády nebo rozmístění populace po městě. Nakonec popisuje tři implementovaná uživatelská rozhraní.

Ve 4. kapitole je detailně popsán model agenta, jak funguje jeho rozhodovací mechanismus a reprezentace paměti. Popsán je jednak obecně koncept aktivit, tak i konkrétní implementované aktivity, z kterých vychází chování agentů v simulaci. Není zde popsáno plánování cesty po mapě.

Celá 5. kapitola se zaměřuje pouze na plánování agentů po mapě. Popisuje plánování ve vozidle, pěšky i s použitím MHD. Popisuje formálně celý problém, dále existující plánovače řešící tento problém a nakonec implementovaný plánovač. Přínos plánovače spočívá v různých heuristikách, z kterých

je možno stavět stromy.

Kapitola 6 popisuje detaily implementace, nejdříve popisuje jádro celého systému, dále se pak zaměřuje na specifické detaily. Popsána je struktura prostředí a objektů v něm, dále fungování senzorů a akcí. Další část kapitoly je zaměřená na implementaci plánovače, s jakými strukturami pracuje a v jakém formátu vrací nalezenou cestu. Poslední část kapitoly je zaměřená na implementaci Google Earth vizualizace, jakým způsobem se předávají data k zobrazení a jaké informace se zobrazují.

Poslední 7. kapitola vyhodnocuje tři klíčové komponenty frameworku: výkon simulace, rychlost vizualizátorů a evaluaci plánovače. Ukazuje závislost výpočetního času na počtu agentů, dále počet snímků za sekundu jednotlivých vizualizátorů pro různé počty agentů a nakonec plánovač. U plánovače jsou měřeny různé metriky pro různé scénáře a různé heuristiky. Porovnává mezi sebou různé heuristiky a zjišťuje jejich výhody a nevýhody.

Kapitola 2

Agentní simulace

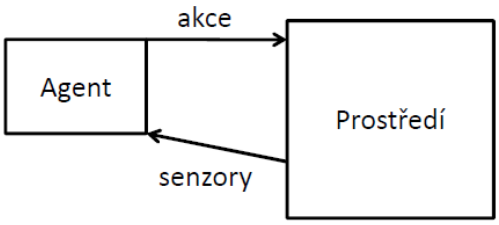
Tato kapitola popisuje obecně agentní simulace, jaké jsou další přístupy simulace, jejich výhody a nevýhody. Dále se zaměřuje na simulaci dopravy a popisuje některé konkrétní platformy. Je zde i zdůvodnění, proč jsme nepoužili žádnou existující platformu, ale vytvořili vlastní.

2.1 Agentní model

Agentní model (anglicky agent-based model, zkratka ABM) je specifický způsob modelování určitého scénáře. Narozdíl od systémové dynamiky v agentním modelu je jedna entita v prostředí jeden agent. Model už není tvořen soustavou diferenciálních rovnic ale množinou agentů a prostředím, ve kterém se pohybují. Agent s prostředím interaguje pomocí akcí a senzorů (obr. 2.1). Pomocí senzorů zjišťuje stav okolního světa, například svojí pozici, kde jdou dveře nebo ostatní agenty. Prostředí ale i sebe ovlivňuje pomocí akcí. Akce vyvolají nějakou změnu v celém prostředí, můžou ovlivnit okolní svět, okolní agenty, sebe samého, ale můžou i selhat. Akce může být například krok dopředu, otevření dveří nebo strčení do jiného agenta. Akce může selhat například, když se pokusí otevřít zamčené dveře nebo udělat krok proti zdi. Prostředí pouze vykonává akce a odpovídá na dotazy senzorů, samotný agent rozhoduje, které akce kdy vykoná na základě senzorických informací a svého vnitřního stavu.

Agenti většinou mají paměť, ale můžou být i čistě reaktivní. Do paměti si ukládají informace, které později budou potřebovat, například se můžou učit, pamatovat si cesty nebo své cíle. Můžou kooperovat s ostatními agenty, tedy mít globální cíl anebo jednají pouze ve vlastním zájmu. V agentech se dobře využívá herní teorie.

Výhody agentní simulace jsou: možno modelovat na nízké úrovni, tedy



Obrázek 2.1: Interakce agenta s prostředím

velmi detailně, je možno monitorovat jedince, bere se ohled na pozici agenta a vzdálenost od ostatních, možno modelovat složitější interakce mezi agenty, agenti se můžou učit, pomocí jednoduchých agentů je možno modelovat složitý systém, tedy implementace daného problému je mnohem jednodušší. **Nevýhody** agentního modelu jsou: výrazně pomalejší výpočet než pomocí diferenciálních rovnicí, je potřeba více dat, hůře vyhodnotitelné výsledky, agenti nedělají globální rozhodnutí, rozhodují se pouze lokálně (KRISTEN L. SANFORD BERNHARDT, 2007). Z toho plyne, že agentní přístup se používá všude tam, kde je potřeba simulovat interakce mezi agenty, záleží na jejich pozici, je třeba sledovat jednotlivce a je potřeba tak detailní simulace, že nejde simulovat pomocí diferenciálních rovnic. Pokud jde něco simulovat pomocí diferenciálních rovnic, tak je agentní model zcela nevhodný, protože by byl výpočet příliš pomalý. V praxi byl agentní model nasazen například při simulaci řízení letového provozu, synchronizace semaforů, simulace dopravy, simulace evakuace nebo pohybu lidí v obchodním domě nebo zábavním parku.

2.2 Simulace dopravy

Doprava se dá simulovat více způsoby podle toho, jak detailní simulaci potřebujeme a jak rychlá by měla být. Makro simulace je vyšší způsob simulace, simuluje celé prostředí bez větších detailů, je to spíše soubor diferenciálních rovnicí. Simuluje dlouhodobý průběh a velké měřítko. Mikro simulace simuluje na nižší úrovni, simuluje větší detaily. Je určená pro kratší rozmezí simulace, ale zase rozlišuje mnohem menší časové okamžiky (Dirk Helbing; Ansgar Hennecke; Vladimir Shvetsov; Martin Treiber, 2002).

2.2.1 Použití agentního modelu na simulaci dopravy

Simulace dopravy pomocí agentního modelu je mikro přístup, většinou je jeden agent jedno vozidlo nebo řidič s vozidlem, agenti ale můžou být i semafory nebo letoví dispečeři. Simuluje se na úrovni jednotlivých vozidel, jezdí po silnicích, reagují na značky a semafory, vyhýbají se ostatním vozidlům a interagují s nimi (KRISTEN L. SANFORD BERNHARDT, 2007), (Shinya KIKUCHI; Jongho RHEE; Dušan TEODOROVIĆ, 2002), (Michael Balmer; Nurhan Cetin; Kai Nagel; Bryan Raney, 2004).

2.2.2 Existující platformy na simulaci dopravy

Jedna existující platforma pro simulaci dopravy se jmenuje Multi Agent Transport Simulation (**MATSim**) [MATSIM]. Jedná se o agentní mikro simulaci, která simuluje průjezd aut určitým grafem silnic. Vstupem je graf silnic, vozidla začínající na uzlech v grafu a cíle vozidel. Výstup jsou dojezdové časy jednotlivých vozidel a jejich trasy. Každé vozidlo se snaží dostat do svého cíle co nejrychleji, hledá nejrychlejší cestu v závislosti na ucpanosti silnic. Celá simulace běží v iteracích, pokaždé se spouští stejný scénář, ale řidiči si pamatují informace z minulých běhů. Postupně tok aut dokonverguje k nějakému stavu a tam se ustálí, tento stav se nazývá nashovo equilibrium. Využití je například otestovat propustnost dané sítě nebo odhadnout dojezdový čas.

Transportation Analysis and Simulation System (**TRANSIMS**) je další simulační platforma, která modeluje pohyb aut v síti silnic [TRANSIMS]. Modeluje i interakce mezi vozidly. Zaměřuje se více na reálné potřeby lidí, všichni chodí do práce, jenom někteří chodí nakupovat a nakoupit jdou například i v závislosti na tom, jak velké jsou zácpy. Generuje populaci ve městě tak, aby co nejvíce odpovídala reálnému rozložení občanů. Potřebuje mnoho vstupních dat, aby simulace co nejvíce odpovídala realitě, výstup je jako u předchozího frameworku nashovo equilibrium, kudy se lidem nejvíce vyplatí jezdit, také měří dojezdové časy.

Další platforma je Simulation of Urban Mobility (**SUMO**), která také simuluje dopravu na úrovni jednotlivých vozidel [SUMO]. Platforma je více otevřena, je navržena tak, aby byly možné implementovat vlastní algoritmy simulace. Umožňuje různé typy vozidel, změnu jízdního pruhu, interakci s ostatními vozidly a infrastrukturou. Nemá pevné použití, je na uživateli, na co platformu použije, může například hledat nashovo equilibrium jako v předchozích dvou platformách nebo simulovat jen několik propojených křižovatek a sledovat pohyb vozidel.

Všechny tyto platformy modelují pohyb vozidel v síti silnic na úrovni

mikrosimulace, ale vždy je jeden agent jedno vozidlo. My se zaměříme spíše na model chování občana, tedy agent bude člověk - občan a vozidla budou jen prostředky používané agenty, potom bude možné detailně simulovat i městskou hromadnou dopravu. V žádné z této platforem by nebylo tento model možno vytvořit anebo velice složitě, proto jsme vyvinuli vlastní platformu AgentPolis.

Kapitola 3

Navržený model mobility

Tato kapitola obsahuje obecný popis našeho přínosu. Popisuje vytvořený framework spíše z uživatelského pohledu. Zaměřuje se spíše na prostředí s objekty, data a uživatelská rozhraní, nepopisuje vnitřní logiku. Chování agentů bude popsáno v kapitole 4 a dále navigace agentů v prostředí v kapitole 5. V kapitole 6 budou popsány detaily implementace.

3.1 Simulační model

Naším přínosem je simulační framework, který modeluje pohyb lidí a aut po ulicích města. Lidé chodí do práce, jíst nebo nakoupit, jezdí svými auty nebo MHD prostředky, tedy cílem simulace je simulovat denní potřeby občanů města. Simulace obsahuje prostředí a entity, entity se ještě dělí na agenty a zdroje.

3.1.1 Prostředí

Prostředí se skládá z grafů, po kterých se pohybují entity a ze značek. Jsou tyto typy grafů:

- Cesty - pro pohyb agentů pěšky
- Silnice - pro silniční vozidla (auta, kola, autobusy)
- Koleje - pro vlaky (metra a tramvaje)

Grafy nejsou disjunktní, více grafů může mít stejné uzly i hrany. Zatím nerozlišujeme silnice a cestičky, tedy grafy silnic a cest jsou totožné.

Základní graf je tvořen pouze uzly a hranami. Graf je orientovaný, ale jednosměrky nemáme, tedy všude, kde je hrana jedním směrem, je i opačným.

Graf silnic a kolejí je pokryt grafem front. Pokud jsou na silnici i koleje, fronta je sjednocená, tedy se za sebe řadí auta i tramvaje. Dále je graf silnic pokryt parkovacími místy, parkuje se na hranách a každá hrana má omezenou kapacitu.

Na některých křižovatkách jsou semafory, vždy svítí zelená na protilehlých silnicích křižovatky a kolmo na ně červená. Pokud má vozidlo zelenou, může vyjet z křižovatky jakoukoliv hranou, neřešíme směrové semafory. Semafory se přepínají automaticky všechny se stejným intervalem ale jiným offsetem, přepínají se barvy: zelená, oranžová, červená a červená s oranžovou. Dále jsou na grafech maximální povolené rychlosti.

3.1.2 Zdroje

Zdroje jsou různé objekty v prostředí, které využívají agenti. Jsou základní dva typy zdrojů: vozidla a zařízení (firmy, školy, obchody). Vozidla jsou pohyblivé entity, které jsou využívány agenty, ale samy o sobě nic nedělají. Ve městě jsou tyto typy vozidel: osobní auta, kola, metra, tramvaje a autobusy. Každé vozidlo má jedno místo pro řidiče a určitou kapacitu pro pasažéry. Každý občan má doma auto a kolo. Těmito prostředky jezdí jako řidič, může do nich nastupovat a zaparkovat je. Metra, autobusy a tramvaje jsou vždy řízeny řidiči MHD, občané nimi můžou jezdit pouze jako pasažéři. Agenti se zatím nevozí autem, například že by jeden agent vzal druhého autem do práce.

Zařízení jsou nehybné entity, lidé v nich pracují nebo využívají jejich služby jako hosti. V prostředí je několik typů zařízení: firmy (pracovní), dále zařízení s hosty: školy, obchody, restaurace a zábavy. Zařízení s hosty jsou rozšířené pracovní firmy, tedy občané můžou pracovat ve všech firmách. Kapacita hostů zatím není omezená.

3.1.3 Agenti

Agenti jako jediní mají mozek a můžou se rozhodovat, jaké akce vykonají. Agenti můžou buď samostatně chodit po cestách nebo využívat různé dopravní prostředky, buď jako řidiči nebo jako pasažéři. Dále můžou pracovat ve firmách nebo vstupovat do firem jako hosti, agenti spolu nijak neinteragují. Máme dva typy agentů. První typ je občan, který žije svůj normální život ve městě, chodí do práce, nakupovat, jíst nebo za zábavou. Druhý typ je řidič MHD, který se narodí ve svém dopravním prostředku a celý běh simulace jenom vozí pasažéry.

Agenti na základě senzorických informací rozhodují, jaké akce vykonají. Některé akce mají nulovou dobu trvání a provedou se hned, jiné nějakou dobu

trvají a provedou se až po určité době. Některé akce můžou selhat, pokud akce selže, tak to agentovi oznámí a on se rozhodne, co bude dělat dál.

3.1.4 Akce agenta

Agent pomocí akcí ovlivňuje sebe nebo okolí. Akce trvají nějakou dobu, tedy akce se neprovede okamžitě, jakmile jí agent zavolá a mezitím se může prostředí změnit. Seznam akcí agenta je v tabulce 3.1.

Tabulka 3.1: Seznam akcí agenta (akce obsahující nastupování zahrnuje i vystupování)

Akce s firmami	nastupování do firmy jako pracovník/host
Akce řidiče s vozidlem	nastupování do vozidla jako řidič
Akce pasažéra s vozidlem	nastupování do vozidla jako pasažér
Akce pro řidiče MHD	ohlášení/odhlášení ze stanice
Pohyb po hraně	přesunutí se z jednoho uzlu hrany na druhý
Pohyb po hraně frontou	zařazení do fronty a dojetí na konec
Reportování do statistik	zapsání záznamu do statistik
Zápis informace pro vis	uložení informace do storage

- **Vcházení/vycházení z firmy, buď jako zaměstnanec nebo jako host:** při vcházení má jako parametr jméno firmy, při vycházení žádné parametry. Agent musí být na uzlu s určenou firmou a nesmí být v jiné firmě. Selhat může pouze vcházení do firmy jako host, pokud je už firma plná.
- **Nastupování/vystupování z vozidla jako řidič:** nastupování do vozidla má jako parametr jméno vozidla, vystupování nemá žádný parametr, pokud jde o parkování na hranách, obě funkce mají ještě jako parametr ID hrany. Při nastupování musí být agent na uzlu s autem, případně na jednom uzlu hrany, auto musí být zaparkováno na příslušném místě. Vystoupení z auta znamená i zaparkování, před zavoláním musí auto stát. Pokud parkuje na uzlu, tak se akce vždy povede, pokud parkuje na hraně, akce selže, pokud je hrana v době zaparkování plná.
- **Nastupování/vystupování z vozidla jako pasažér:** nastupování má jako parametr jméno vozidla, vystupování žádný parametr. Agent musí být před zavoláním na stejném uzlu jako vozidlo a vozidlo musí stát. Akce nasoupení selže, pokud se mezitím vozidlo rozjelo a nebo zaplnilo. Vystoupení selže, pokud se vozidlo během vystupování rozjelo.

- **Akce pro řidiče MHD - ohlášení příjezdu/odjezdu ze stanice:** akce nemá žádné parametry a nemá prodlevu, tedy se provede okamžitě. Ohlásí/odhlásí vozidlo řízené agentem na stanici, na které musí právě stát, nemůže selhat.
- **Pohyb po hraně:** jako parametr má ID cílového uzlu, může se zadat i rychlost. Cílový uzel musí být přímo dostupný z aktuálního uzlu, nemůže selhat.
- **Pohyb po hraně s použitím front:** jako parametr má ID cílového uzlu, může se zadat i rychlost. Cílový uzel musí být přímo dostupný z aktuálního uzlu a musí zde být fronta, kam se vozidlo vejde. Akce se dokončí, jakmile vozidlo dojede na konec fronty, což je ovlivněno ostatními vozidly. Akce nemůže selhat.
- **Reportování informací do statistik (akce pro rozhraní):** vždy má jako parametr text a typ zprávy, dále má parametry specifické pro druh zprávy. Může se reportovat pouze text, číslo, pole čísel nebo vlastní stav. Akce uloží záznam okamžitě do storage a sama doplní aktuální uzel a čas, nemůže selhat.
- **Zápis informace, která se použije čistě pro vizualizaci (akce pro rozhraní):** jako parametr má klíč a objekt, okamžitě zapíše do storage objekt pod daný klíč, nemůže selhat.

3.1.5 Senzory agenta

Senzory vracejí agentovi informace o prostředí, jsou dvojího typu, první typ na dotaz okamžitě vrátí informaci (jaká barva svítí na semaforu). Druhý typ senzoru je callbackový, oznámí agentovi, pokud se něco v prostředí změnilo (naskočila zelená). Pro poslouchání senzorů je použit sensing, je to callback ze senzoru do agenta. Agent si musí registrovat sensing na callbackový senzor, který chce poslouchat. Senzor oznámí změnu, některé senzory odstraní sensing okamžitě po zavolání, některé ho nechají a agent ho musí odstranit sám. Seznam senzorů pro agenta je v tabulce 3.2.

Tabulka 3.2: Seznam senzorů agenta

Obyčejné:	
Informace o firmách	hledání firem, zjišťování pozic
Senzor na parkování	hledání zaparkovaného vozidla
Senzor na silnice	náhodné uzly, odchozí hrany z uzlu
Informace pro vis	vrátí objekt ze storage
Maximální rychlosti	povolené rychlosti na hranách
Plány MHD řidiče	plán stanic určité linky
Informace o vozidlech	parametry a aktuální stav vozidla
Senzor pro řidiče	parametry vozidla pro řidiče
Pouze callbackové:	
Dokončování akcí	oznámí (ne)povedení akce
Smíšené:	
Senzor na svojí pozici	vrací svou pozici, oznamuje změnu
Senzor na čas	vrací aktuální čas
Informace o MHD	MHD vozidla na stanici, čekání
Fronty	jestli se vejde do fronty, čekání
Semafory	stav semaforu, čekání

- Obyčejné
  - **Informace o firmách:** různé použití s potřebnými parametry: náhodná firma {}, náhodná firma určitého typu {typ}, nejbližší firma {}, nejbližší firma daného typu {typ}, pozice firmy {jméno firmy}, firma určitého typu na aktuálním uzlu {typ} a typ firmy {jméno firmy}.
  - **Senzor na parkování:** má více použití: najít vozidlo na hraně okolo aktuálního uzlu {jméno auta}, zda je auto zaparkované na aktuálním uzlu {jméno auta} a najít volnou hranu okolo uzlu {ID uzlu}.
  - **Senzor na silnice:** má tyto metody: všechny dostupné sousední uzly z uzlu {ID uzlu}, náhodná odchozí hrana {ID uzlu}, náhodný uzel z celého grafu {}, náhodný uzel ve středu mapy {}, náhodný uzel na okraji mapy {}.
  - **Informace pouze pro vizualizaci (senzor pro rozhraní):** na zadaný klíč vrátí uložený objekt.
  - **Maximální rychlost na hraně:** zadá se cílový uzel, který musí být přímo dostupný z aktuálního uzlu, vrátí maximální povolenou rychlost na hraně mezi aktuálním a zadaným uzlem.

  - **Plán cesty pro řidiče MHD:** agent musí řídit MHD vozidlo, potom na určité ID linky vrátí pořadí stanic, které má projet. Dále je možnost vrátit opačnou než aktuální linku, to řidič využije na konečné stanici.
  - **Informace o vozidlech:** má více použití: zda je vozidlo na stejném uzlu {jméno vozidla}, zda má vozidlo ještě volné místo pro pasažéra {jméno vozidla}, zda vozidlo stojí {jméno vozidla}, v kterém vozidle právě agent je {}, typ vozidla {jméno vozidla}, délka vozidla {jméno vozidla} a linka MHD vozidla {jméno vozidla}.
  - **Senzor pro řidiče:** má tři metody bez parametrů: zda se řízené vozidlo musí řadit do front, jeho maximální rychlost a uzel, z kterého zrovna přijel na aktuální uzel.

- Pouze callbackové
  - **Dokončování akcí:** oznamuje, jestli se akce povedla nebo ne. Agentovi přijde typ akce a zda byla úspěšná.

- Smíšené
  - **Senzor na svojí pozici:** při dotazu bez parametrů vrátí ID aktuálního uzlu, pokud agent registruje sensing, tak pokaždé, když se přesune na jiný uzel (ať už akcí pohybu nebo popojede ve vozidle), tak mu přijde ID aktuálního uzlu.
  - **Senzor na čas:** vrací aktuální čas, dále má možnost nastavit budík na určený čas nebo za určitou dobu, který po uplynutí času zpátky zavolá sensing s aktuálním časem.
  - **Informace o MHD:** má více použití: zbývající stanice vozidla než dojede na konečnou {jméno vozidla}, jména všech čekajících MHD vozidel na aktuálním uzlu {}, zda vozidlo čeká na aktuálním uzlu {jméno vozidla}, dále má možnost registrovat sensing, který agentovi oznámí jméno každého MHD vozidla, které zastaví na aktuálním uzlu. Předpokládá se, že agent stojí na uzlu s MHD stanicí.
  - **Fronty:** předpokládá se, že agent právě řídí vozidlo. Pokaždé se jako parametr zadává uzel, na který chce jet z aktuálního uzlu, potom má dvě využití: zeptá se, zda se vejde do fronty nebo registruje sensing na frontu, potom mu sensing oznámí, až se pro něj fronta uvolní.

– **Semafory:** má dvě využití, buď se podívá na stav semaforu a nebo může začít čekat na semafor a ten mu oznámí každou změnu světla. Vždy se bere semafor z posledního uzlu vozidla do aktuálního uzlu.

Agenti pomocí senzorů dostávají informace o značkách, ale můžou je ignorovat. Semafory řidiče nijak fyzicky neomezují, vozidlo může projet na jakoukoliv barvu, je jenom na agentovi, jestli bude čekat nebo ne. Také může překročit povolenou rychlost.

3.1.6 Fronty na hranách

Agenti a kola ignorují fronty a jakoukoliv kapacitu, tedy může jich být neomezeně na jednom uzlu. Ostatní vozidla ale musí čekat ve frontách. Řidič se nejdříve podívá, jestli se do dané fronty s vozidlem vejde. Pokud se nevejde, tak musí čekat, dokud se pro něj neuvolní místo. Jakmile vjede do fronty, tak už nemůže dělat nic, dokud nevyjede. Ve frontách se nedá předjíždět, tedy musí počkat, až všechny vozidla před ním odjedou a také svojí rychlostí dojede na konec. Na několika místech je úsek silnice kratší než některá delší vozidla, aby se tudy dalo projet, tak do prázdné fronty se vejde jakkoliv dlouhé vozidlo.

3.1.7 Parkování

Parkuje se na hranách, každá silnice má v každém směru danou kapacitu v metrech. Pokud není pro dané vozidlo místo, tak nemůže zaparkovat. Akce parkování nějakou dobu trvá, může se tedy stát, že parkovací místo zabere někdo při parkování, v takovém případě akce selže. Kola se neparkují na hranách ale na uzlech, tedy bez omezené kapacity.

3.2 Data

Aby byla simulace co nejvěrohodnější, používáme co nejvíce reálných dat. Zatím importujeme z reálných dat graf silnic a cest, semafory, MHD linky, obchody a restaurace. Později může být importována obydlenost jednotlivých čtvrtí města, informace o lidech, kteří zde žijí nebo jízdní řády. Některé části prostředí jsou importované z reálných dat, některé jsou vygenerované tak, aby se co nejvíce přiblížily realitě.

Tabulka 3.3: Tabulka počtu importovaných dat

Objekt	počet
Silniční uzly	8271
Silniční hrany	10728
Uzly kolejí	444
Hrany kolejí	503
Průměrný počet sousedů uzlu silnice	2.6
Maximální počet sousedů uzli silnice	6
MHD linky	34
MHD stanice	267
Semafory	195
Restaurace	684
Obchody	1

3.2.1 Mapa

Mapa je importovaná z open street maps OSM [OSM]. Graf silnic a kolejí je reálný, cesty jsou zatím pouze kopie grafu silnic, ale máme možnost importovat i cesty. Dále jsou z OSM importovány stanice metra, tramvají a některých autobusů včetně jednotlivých linek bez jízdních řádů. Na jednom uzlu může být víc stanic pro víc typů MHD, stanice metra jsou vždy na samostatném uzlu a uměle připojeny ke grafu cest. Graf linek metra byl původně zcela oddělený od grafu silnic, spojili jsme tedy každou stanici metra s nejbližším uzlem silnice. Importovány jsou i restaurace, jeden obchod a několik semaforů, přesné počty importovaných dat jsou v tabulce 3.3.

Vzali jsme mapu celé Prahy, ale protože načítání bylo příliš pomalé a mapa zabírala hodně paměti, tak jsme ořízli okraj a nechali pouze čtverec v centru.

3.2.2 Jízdní řády

Vozidla MHD zatím nejezdí podle jízdních řádu, ale z obou konečných stanic každé linky v pravidelných rozestupech během dne vyjede určitý počet vozidel a dále jezdí až do konce simulace. Na každé zastávce čeká každý typ MHD určitou dobu, doby čekání byly určeny:

- $wait_{metro} = 30s$
- $wait_{tram} = 45s$
- $wait_{bus} = 45s$

Máme také generátor jízdních řádů, který je zatím použit pouze pro testování plánovače, vozidla se jízdními řády neřídí. Generátor má na vstupu první výjezd $first$, poslední výjezd $last$, minimální $min\_interval$ a maximální interval $max\_interval$ a dále n gaussiánů $mean_1, \ldots, mean_n$, $sigma_1, \ldots, sigma_n$. Nejdříve vygeneruje jízdní řády pro počáteční stanici linky algoritmem 1. Kde funkce $gauss(x, mean, sigma)$ vrátí hodnotu gaussiánu za-

Algorithm 1 Create time-table

generateTimeTable($first$, $last$, $min\_interval$, $max\_interval$, $mean[1..n]$, $sigma[1..n]$):
$timeTable \leftarrow \{first\}$
$actual \leftarrow first$
while $actual < last$ **do**
  $value \leftarrow$ maxGauss($actual$, $mean[1..n]$, $sigma[1..n]$)
  $interval \leftarrow min\_interval * value + max\_interval * (1 - value)$
  $actual \leftarrow actual + interval$
  $timeTable \leftarrow timeTable \bigcup actual$
end while
$timeTable \leftarrow timeTable \bigcup last$
return $timeTable$

maxGauss(x, $mean[1..n]$, $sigma[1..n]$):
return max(gauss(x, $mean[i]$, $sigma[i]$) / gauss($mean[i]$, $mean[i]$, $sigma[i]$))

daného pomocí $mean$ a $sigma$ pro vstup x a funkce max maximalizuje přes všechny $i \in < 1, n >$. Pro každou další zastávku linky se vytvoří nové jízdní řády, kde se ke všem výjezdům z minulé stanice přičte $length_{s,s+1}/$ $average\_speed_{type}$, kde $length_{s,s+1}$ je ujetá vzdálenost mezi stanicemi a $average\_speed_{type}$ je průměrná rychlost daného typu prostředku. Průměrné rychlosti byly určeny takto:

- $average\_speed_{metro} = 55km/h$
- $average\_speed_{tram} = 25km/h$
- $average\_speed_{bus} = 45km/h$

Rychlosti jsou vypočítány na podle doby jízdy z první do konečné stanice ze serveru idos.cz.

Jeden typ MHD má stejně vygenerované jízdní řády, navíc jsou symetrické, tedy z obou konečných zastávek vyjíždějí vozidla ve stejnou dobu. Pro konkrétní typy MHD byly jízdní řády vygenerovány takto:

Tabulka 3.4: Vygenerovaný jízdní řád - všechny tramvaje a jízdní řád z idosu - tramvaj 16, všední den

Hodina	minuty (vygenerovaný)
5	0 15 29 43 56
6	9 22 33 45 56
7	6 16 26 36 45 54
8	3 11 20 28 36 44 52
9	0 8 16 24 33 41 49 57
10	6 15 24 33 43 52
11	3 13 24 35 47 59
12	10 22 33 44 54
13	5 15 25 35 45 55
14	4 13 23 32 41 50 58
15	7 16 24 32 41 49 57
16	5 14 22 30 38 46 54
17	2 10 18 26 34 42 50 58
18	6 15 23 31 39 48 56
19	5 14 23 31 41 50 59
20	8 18 28 37 47 58
21	8 18 29 40 51
22	3 14 26 38 50
23	3 16 29 42 56
24	0 10

Hodina	minuty (z idosu)
5	22 42 57
6	09 19 29 39 47 55
7	03 11 19 27 35 43 51 59
8	07 15 23 31 39 47 55
9	03 13 23 33 34 43 53 55
10	03 13 23 33 43 53
11	03 13 23 33 43 53
12	03 13 23 33 43 53
13	03 13 23 33 43 51 59
14	07 15 23 31 39 47 55
15	03 11 19 27 35 43 51 59
16	07 15 23 31 39 47 55
17	03 11 19 27 35 43 51 59
18	07 15 23 33 35 43 53
19	03 13 14 23 35 40 50
20	05 06 20 35 50 55
21	05 10 22 25 42 59
22	02 22 42
23	02 22 42
24	

- $time\_tables_{metro} = generateTimeTable(3h, 24h, 3min, 10min, [7h, 17h], [3h, 3h])$
- $time\_tables_{tram} = generateTimeTable(5h, 23h, 7min, 25min, [8h, 18h], [4h, 5h])$
- $time\_tables_{bus} = generateTimeTable(6h, 22h, 12min, 45min, [12h], [5h])$

Hodnoty byly zvoleny tak, aby se výsledné jízdní řády co nejvíce podobaly reálným denním linkám. Tabulka 3.4 ukazuje vygenerovaný jízdní řád a jízdní řád ze serveru idos.cz, jízdní rády nejsou úplně stejné, ale pokud máme zatím pro všechny linky jednoho typu MHD stejné jízdní řády, tak takto vygenerovaný jízdní řád stačí.

3.2.3 Generování objektů a značek v mapě

Pozice pracovních firem, škol a zábav jsou generovány gaussiánem se středem v centru mapy. Domovy občanů jsou generovány jako doplněk gaussiánu se středem v centru mapy. Tedy lidé bydlí spíše na okraji a pracují v centru. Maximální rychlost na hraně je generovaná v závislosti na délce hrany v intervalu 20km/h až 60km/h, na delších hranách je větši povolená rychlost. Maximální rychlost pro hranu e se vypočítá $max\_speed_e = min(20 + max(0, d - 10)/120 * 40, 60)$, kde d je délka hrany e. Parkovací místa jsou generována také v závislosti na délce hrany, na 1m hrany v jednom směru připadá 0.2m parkovacího místa. Na semaforu svítí zelená i červená vždy 30s a oranžová 2s. Každý občan má doma auto i kolo a pracuje v právě jedné náhodné firmě.

3.3 Uživatelské rozhraní

Máme více typů uživatelských rozhraní, každý je dobrý na něco jiného. A-lite vis slouží jako jednoduchý grafický výstup a ovládání simulace, Google Earth je rozhraní pro koncového uživatele a statistiky zobrazují seznam důležitých událostí během simulace.

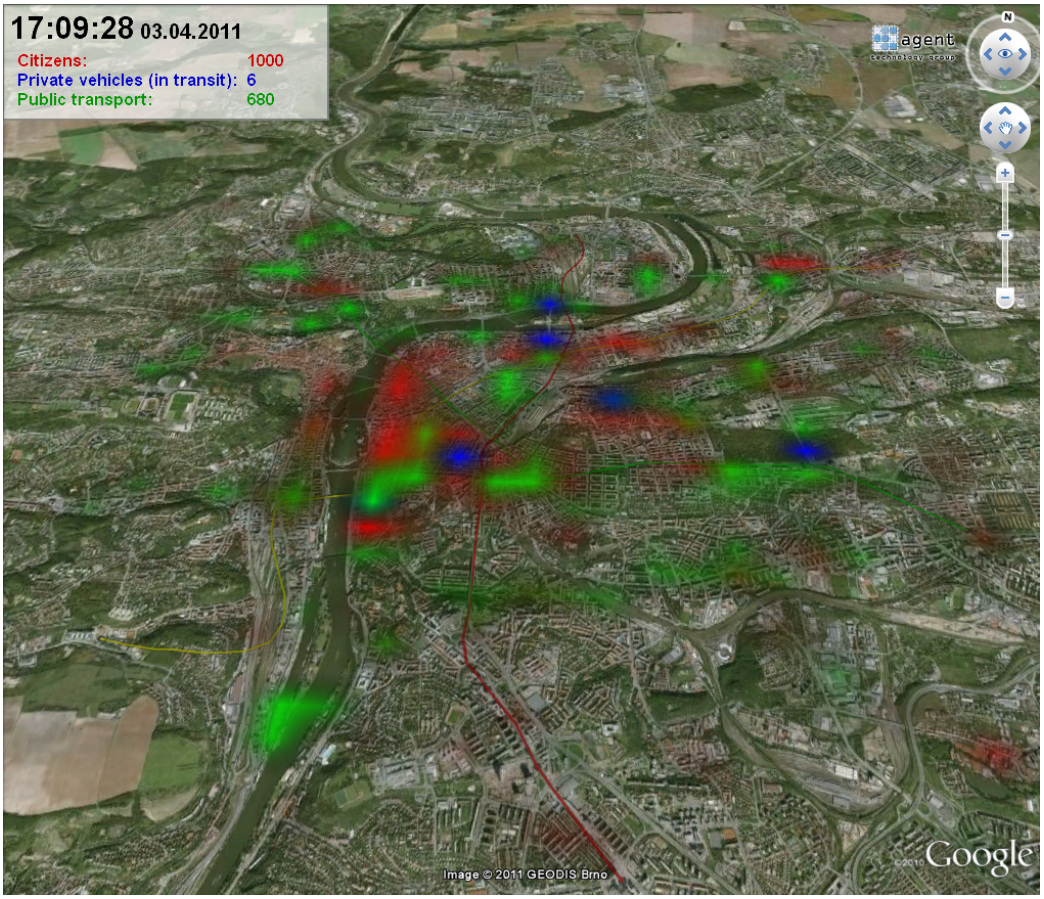
3.3.1 A-lite vis

Základní uživatelské rozhraní je A-lite vis (obr. 3.1). Jedná se o 2D vizualizátor z frameworku A-lite, který slouží zároveň k ovládání simulace. Zobrazuje pouze některé objekty prostředí a zobrazuje je velmi jednoduše. Například se nedají nijak rozlišit jednotlivé entity jednoho typu, nezobrazuje firmy a entity uprostřed hrany nejsou interpolované. Umožňuje zrychlit, zpomalit nebo zastavit simulaci.

3.3.2 Google Earth

Druhý vizualizátor je Google Earth, zobrazuje všechny objekty v prostředí. Pohybující se objekty interpoluje na hraně, takže jedoucí auto neskočí z jednoho uzlu na druhý, jako u A-lite vis, ale jede plynule po silnici. Při pohledu z výšky zobrazuje pouze hustotu jednotlivých typů entit (obr. 3.2). Při bližším pohledu postupně nahradí hustotu jednotlivými objekty (obr. 3.3), u objektů zobrazuje informace (obr. 3.4). Občané mají dvě úrovně detailu: hustotu a jednotliví lidé. Vozidla mají tři úrovně: hustota, relativní hustota silnic, kterou určuje tloušťka silnice a nakonec jednotlivá vozidla. Zobrazuje i jednotlivé

Obrázek 3.1: Ukázka alite vis, černě silnice, šedivě koleje, červeně občané, zeleně MHD vozidla, modře osobní vozidla, dále semafory



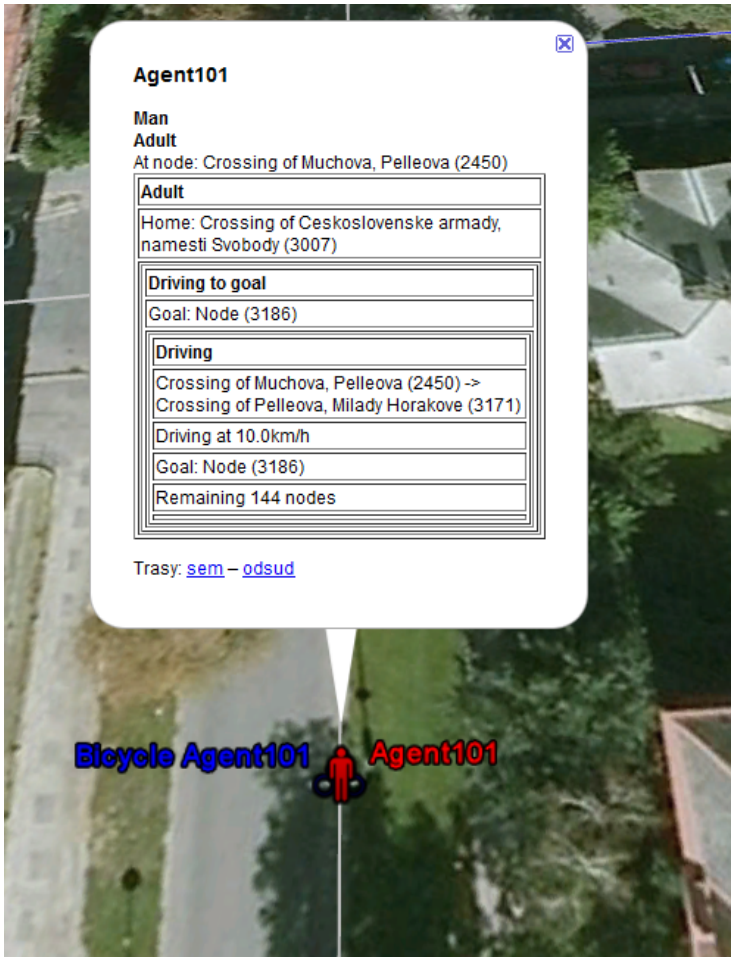
Obrázek 3.2: Ukázka Google Earth - hustoty: červeně občané, modře osobní vozidla, zeleně MHD vozidla

firmy, ale u nich jsou jen statické informace, není u nich například aktuální počet hostů.

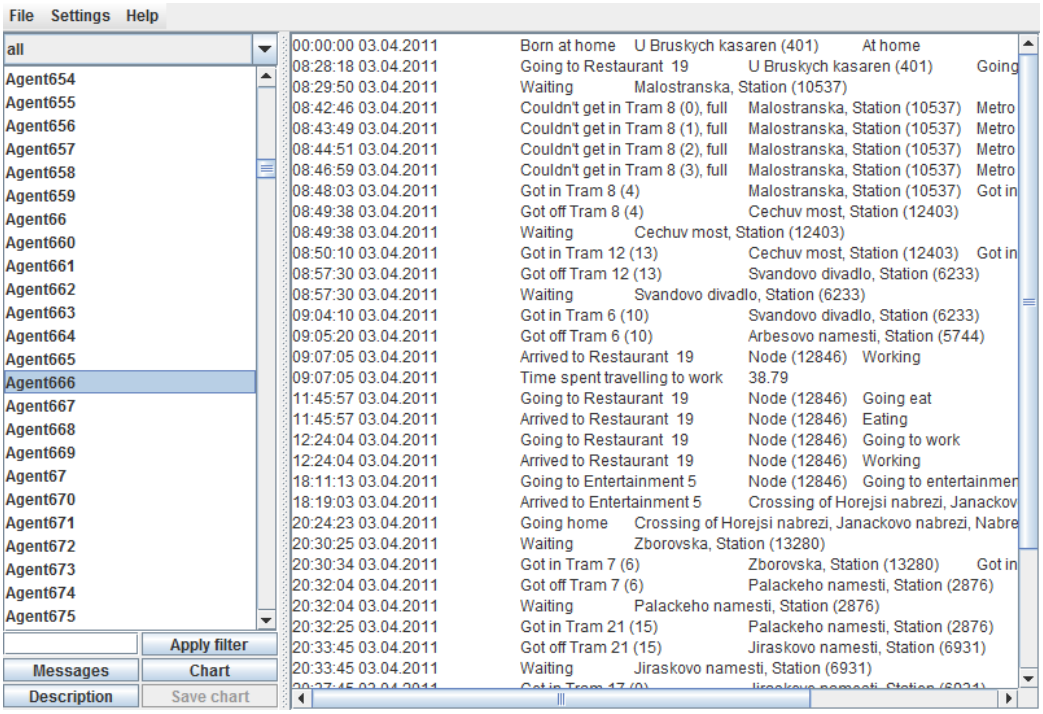
3.3.3 Statistiky

Poslední typ rozhraní jsou statistiky. Zobrazují jednak jednotlivé reporty entit (obr. 3.5), ale i agregované statistiky jako například grafy (obr. 3.6). Dále můžou ke každé entitě otevřít okno popisu stejně jako v Google Earth (obr. 3.7). Obsahuje jednoduché filtry, dají se filtrovat zobrazené entity v seznamu, ale i záznamy. Například se dají zobrazit pouze záznamy od všech občanů ohledně využívání MHD.

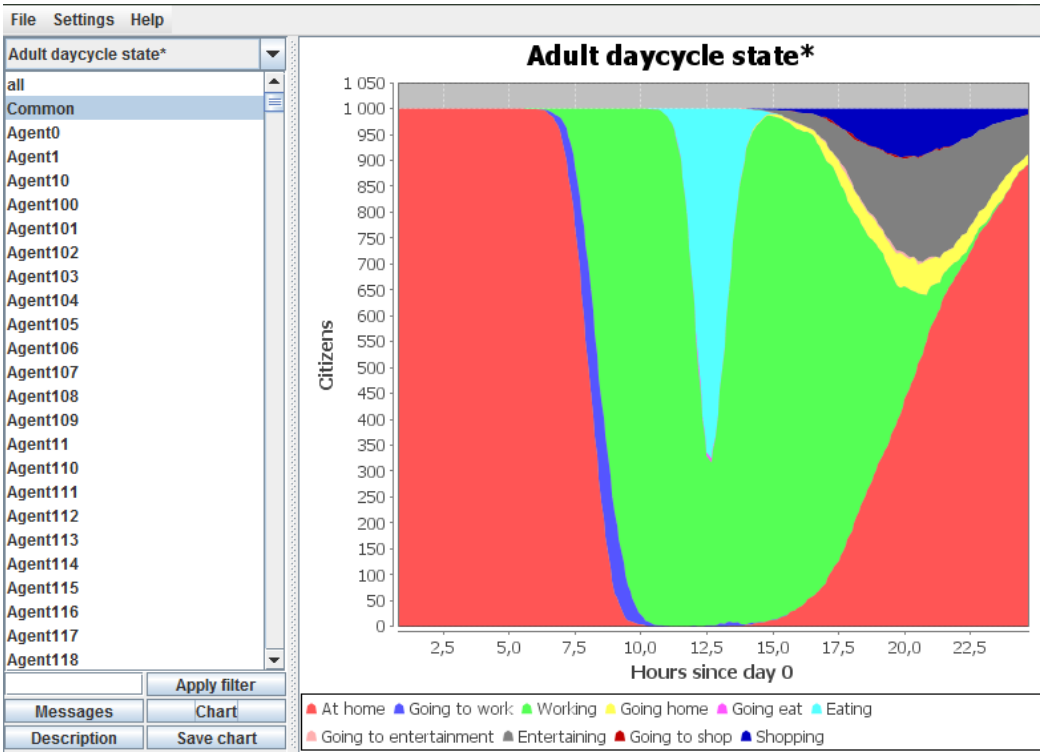
Obrázek 3.3: Ukázka Google Earth detail - zobrazené jednotlivé entity, grafy silnic a firmy



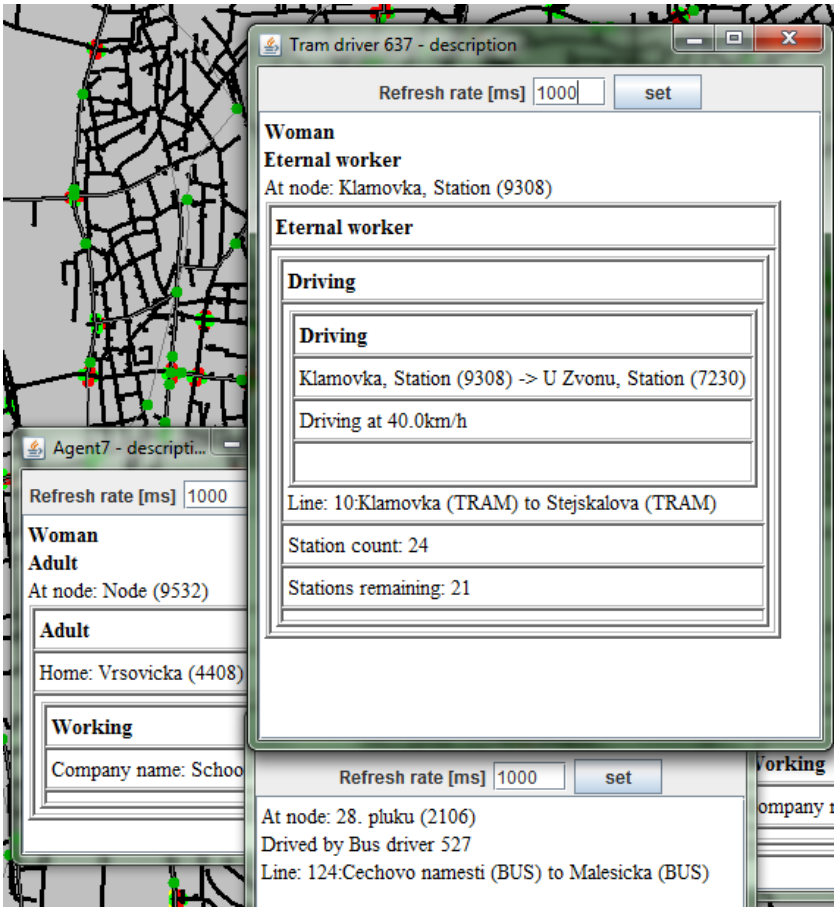
Obrázek 3.4: Ukázka Google Earth popis entity - zobrazen aktuální stav rozhodovacího mechanismu



Obrázek 3.5: Ukázka statistik - seznam záznamů reportování jednoho agenta



Obrázek 3.6: Ukázka statistik - graf poměr stavů občanů



Obrázek 3.7: Ukázka statistik - popis entit stejně jako v GE

3.4 Výstup frameworku

Framework vyhodnocuje různé údaje o dopravě, zaměřuje se hlavně na dojezdové časy. Zatím vyhodnocuje tyto údaje: aktuální poměr stavů občanů (obr. 3.6), dojezdová doba do práce a využití MHD. Dojezdová doba se reportuje každých 10 minut. Každý report je průměr všech dojezdových dob agentů, kteří v tomto intervalu dorazili do práce. Reporty se zobrazují v grafu. Dále se zobrazuje graf průměrů přes celé dny, každý den v 18:00 se zprůměrují všechny dojezdové časy za aktuální den a uloží se jako jedna hodnota. Dále se zobrazuje vytíženost MHD. Reportování je také po deseti minutách, reportuje se počet úspěšných a neúspěšných nastoupení do MHD vozidla. Vytíženost se zobrazuje jako graf sumy úspěšných nástupů a neúspěšných nástupů. Neúspěšný nástup je, pokud se občan pokusí nastoupit do vozidla, ale ve vozidle už není místo. Toto je dočasné řešení, počítáme s tím, že reportování rozlišíme minimálně na typ prostředku.

Na těchto statistikách se dá pozorovat, jaký vliv mají nějaké změny v prostředí. Například pokud přestane jezdit jedna linka metra, o kolik se prodlouží dojezdová doba. Nebo pokud v centru omezíme maximální povolenou rychlost na 20km/h, za jak dlouho se dostanou do práce řidiči. Později se nebudou agenti rozhodovat náhodně, zda do práce pojedou autem nebo MHD, ale budou se rozhodovat na základě zkušeností nebo aktuálního stavu ucpanosti silnic a potom budeme moci sledovat další údaje.

Kapitola 4

Model chování agenta

V této kapitole je detailně popsán rozhodovací mechanismus agentů. Popsáno je, jak se rozhodují, jak reagují na prostředí a jejich vnitřní stav. Součástí agentů je i plánování v prostředí, ale to je popsáno v kapitole 5, tato kapitola je zaměřena na chování agenta.

4.1 Struktura agenta

Agenti se jako jediní v simulaci můžou rozhodovat a využívat zdroje. Aby toho byli schopní, musí mít nějaký rozhodovací mechanismus. Rozhodovací mechanismus našeho agenta se skládá z paměti, paměti na sdílené paměti a vlastního chování.

Hlavní paměť je jenom seznam, kde si pod určitým klíčem uloží nějakou informaci, například domov → adresa, zaparkované auto → adresa, kdy jsem naposledy jedl → čas. Záznamy v paměti může kdykoliv vytvářet, přepisovat, mazat a číst je. Paměť na sdílené paměti je také seznam, klíčem se dotazuje na určitou paměť. Sdílená paměť zatím není využitá, ale bude se používat například jako zjednodušená náhražka komunikace taxikářů nebo policistů. Například pod klíčem paměť taxikářů bude seznam lidí čekajících na taxíky. Tento koncept není čistě agentový přístup, protože nikdy žádná sdílená paměť neexistuje, ale pro naši simulaci se to hodí, protože komunikace by byla příliš zatěžující a nepotřebujeme simulovat na tak nízké úrovni.

4.2 Navržený koncept aktivit

Pro reprezentaci chování agenta jsme zavedli koncept aktivit. Je to hierarchická struktura jednotlivých aktivit. Jedna aktivita je nějaká jedna schopnost složitější než akce, například chození z bodu A do bodu B, řízení auta,

ježdění MHD včetně navigace. Struktura právě spuštěných aktivit se dá zobrazit jako popis agenta (obr. 3.7).

Aktivita je modul, který je spuštěn rodičovskou aktivitou, nějakou dobu vykonává svou činnost a pak se ukončí. Jedna aktivita může spustit i více podaktivit najednou, například dojít na nějaké místo a přitom telefonovat. Rodičovská aktivita může požádat o ukončení, například pokud jede z práce domů, ale jede déle než předpokládal a už přišel čas na večeři, tak jízdu domů ukončí a p;jde se najíst. Ukončení většinou není okamžité, podaktivita musí dokončit rozdělanou činnost a potom oznámit rodičovské aktivitě, že skončila. Například když agent jede v autě, najednou chce jít pěšky na jídlo, tak musí počkat, až se jízda v autě ukončí, tedy vystoupí z auta a až teprve potom může jít pěšky. Když aktivita skončí a oznámí to, tak rodičovská aktivita může zjistit stav ukončené aktivity, aby zjistila, zda se aktivita povedla dokončit nebo selhala.

Rozlišujeme tyto základní typy aktivit: celý život (life cycle), jeden den (day cycle), pracování (working), pohyb (going to) a ostatní aktivity. Agent má jednu life cycle aktivitu, která určuje jeho celý život. Tato aktivita je spuštěna samotným agentem při vytvoření a nikdy neskončí. Nedělá nic jiného, než že postupně spouští jednotlivé dny. Může to být pokaždé stejná day cycle aktivita, nebo můžou být různé na všední dny a na víkendy, nebo může být výběr dne ještě složitější.

V jeden den agent dělá postupně určité činnosti, může mít buď daný rozvrh, nebo se může rozhodnout, kdy co začne dělat nebo jestli udělá ještě něco navíc. Například může jet ráno do práce, ale pokud zrovna nejezdí metro, tak ví, že mu to potrvá déle a vyjde dříve. Nebo může být nemocný a do práce vůbec nepůjde, ale zase mu není tak špatně, aby večer nešel do hospody. Může se rozhodnout, jestli po práci půjde domů, nakoupit nebo za zábavou.

Aktivity pracování vykonávají činnost, která je ve světě považována za práci. Může to být například řízení MHD prostředků nebo práce v kanceláři. Pracovní aktivity se spouští všechny stejným způsobem, tedy je jednoduché změnit práci.

Pohybové aktivity slouží k přemístění agenta z jedné pozice na mapě na druhou. Obsahují plánování, které bude podrobněji popsáno v kapitole 5, dále samotné vykonávání cesty. Některé aktivity pohybu zahrnují například nastupování a vystupování z vozidel, hledání parkovacího místa nebo čekání na zastávkách. Většina pohybových aktivit má stejné použití, tedy jenom se řekne, kam se má agent dostat a aktivita už agenta dopraví na určené místo. Některé mají ale i dodatečné parametry, například jestli má jet na kole nebo autem. Některé zase nemůžou být použity samostatně, například schopnost řízení auta nezahrnuje nastupování a parkování, proto je obalena

další aktivitou.

4.3 Konkrétní agenti a jejich aktivity

V této podkapitole je popsáno chování jednotlivých aktivit. Typ agenta je určen life cycle aktivitou, tedy tato aktivita zároveň popisuje celého agenta. Seznam všech aktivit je v tabulce 4.1

Tabulka 4.1: Seznam aktivit agenta

Dospělý občan:	
AdultLifeCycle	celý život občana
AdultDayCycle	jeden den občana
MHD řidič:	
EternalWorkerLifeCycle	celý život věčného pracovníka
WorkingAsMHDDriver	ježdění MHD vozidlem po lince
Pohyb:	
WalkingTo	chůze po mapě
GoingToUsingMHD	chůze a využití MHD
TravellingByMHDTo	pouze využití MHD
DrivingTo	celkové používání vozidla včetně chůze
DrivingVehicle	řízení vozidla, parkování
Ostatní:	
WorkingInOffice	pracování v kanceláři
SpendingTimeAt	využívání služeb zařízení jako host
Null	čekání určitou dobu

4.3.1 Dospělý občan

Dospělý občan je definován počátečními položkami v paměti a svým life cyclem. Při inicializaci mu jsou do paměti vloženy položky, časové hodnoty jsou reprezentovány náhodnými veličinami normálního rozdělení, jak zobrazuje tabulka 4.2. Konkrétní čas vždy vygeneruje, když začíná předchozí aktivitu, tedy například pokud jde spát, tak si hned vygeneruje přesný čas, kdy vstane. Místa jsou vygenerovány, jak je popsáno v kapitole 5. Dále ví, že má doma auto i kolo.

Chování dospělého občana určuje hlavní aktivita **AdultLifeCycle** a aktivita dne **AdultDayCycle**. Občan má zatím jenom jeden možný scénář dne, tedy life cycle aktivita spouští každý den ten samý day cycle.

Den dospělého agenta vypadá takto:

Tabulka 4.2: Tabulka počátečních položek v paměti občana

Položka	střední hodnota [h]	rozptyl [min]
Místo domova		
Firma kde pracuje		
V kolik vstane	8	40
V kolik jde na oběd	12	30
Jak dlouho pracuje	9	90
Jak dlouho jí	1	20
Jak dlouho se baví	4	120
Jak dlouho nakupuje	3	30

1. Vzbudí se a jede do práce
2. Začne pracovat
3. Jde na oběd
4. Jde zpátky do práce
5. Jde nakoupit, za zábavou nebo domů, vybere jednu variantu
6. Pokud šel za zábavou nebo nakoupit, tak jede domů

Hned na začátku dne si vybere, jestli bude pořád jezdit autem, na kole nebo využívat MHD. Autem bude jezdit s pravděpodobností 0.4. Pokud nebude jezdit autem, tak s pravděpodobností 1/6 bude jezdit na kole, jinak bude využívat MHD. Během dne už nikdy nezmění způsob pohybu po městě.

Jakmile začne pracovat, tak si vygeneruje, jak dlouho bude pracovat. Když přeruší práci obědem a pak se zase vrátí, tak dopracuje zbytek, aby pracoval přesně stanovenou dobu. Po práci s pravděpodobností 0.15 půjde nakupovat, pokud nejde nakupovat, tak s pravděpodobností 0.4 půjde za zábavou, jinak domů. Často se stává, že se baví, když už by měl končit den, v tom případě day cycle trvá déle než 24 hodin, dokud se agent nevrátí domů. Poměr aktuálních stavů agentů zobrazují statistiky (obr. 3.6).

4.3.2 Věčný pracovník - MHD řidič

Řidiči MHD nemají v simulaci jinou funkci než řídit svůj prostředek, narodí se ve svém vozidle a zůstanou tam až do konce simulace. Jejich hlavní aktivita je **EternalWorkerLifeCycle**. Tato aktivita předpokládá, že agent už má v paměti jednu working aktivitu a dobu, za jak dlouho začne. Po narození tedy čeká určitý čas a začne práci, která nikdy neskončí.

Řidiči MHD celý život vykonávají **WorkingAsMHDDriver** aktitivu. Jejím úkolem je jezdit dopravním prostředkem po lince tam a zpátky podle plánu. Zastavuje na zastávkách a vždy čeká určený čas, než lidi vystoupí a nastoupí. Řidič ze svého aktuálního vozidla pomocí senzoru získá ID linky a z něj plán stanic. ID linky vždy obsahuje typ dopravního prostředku, číslo linky a směr, například "4:Klamovka (TRAM) to Cechovo namesti (TRAM)" nebo "A:Dejvicka, Metro A (METRO) to Strasnicka, Metro A (METRO)". Když má plán zastávek a ví, na které zastávce se nachází, tak už mu jenom chybí plán, kudy se dostane na další zastávku. Plány už jsou předpočítané, ale zatím není vytvořen senzor, takže musí pokaždé naplánovat cestu. Když přijede na stanici, tak ohlásí, že přijel na určitou stanici, aby čekající občané věděli, že něco přijelo. Potom čeká určitou dobu, specifickou pro každý typ prostředku a po uplynutí doby se odhlásí ze stanice a jede dál. Na konečné zastávce po vystoupení cestujících otočí směr, znova se registruje do stanice a pokračuje dál v opačném směru.

4.3.3 Aktivity pohybu

Aktivity pohybu dopraví agenta z aktuální pozice na určené místo. Některé mají jediný vstupní parametr a to cíl cesty, některé mají další parametry jako například vozidlo, kterým mají jet.

- Nejzákladnější je **WalkingTo**, reprezentuje chození agenta po silnicích. Nejdříve naplánuje cestu k cíli na grafu silnic, potom zavolá akci pohybu na další uzel grafu. Senzorem na svojí pozici mu přijde sensing, že došel na uzel a zase přejde na další. Tímto způsobem se dostane až k cíli a pak aktivita skončí.

- Aktivita **GoingToUsingMHD** dopraví agenta na místo určení pomocí chození a využívání MHD. Nejdříve naplánuje celou cestu, potom dojde na první zastávku pomocí WalkingTo, dále se pomocí TravellingByMHDTo aktivity dostane na koncovou zastávku a dál pokračuje pěšky případně na další stanici. Tato aktivita jenom spojuje tyto dvě aktivity do jedné, aby poskytla jednoduchý způsob volání.

- **TravellingByMHDTo** aktivita dostane agenta z jedné stanice na druhou na souvislém grafu linek. Dokáže přestupovat na stanicích na jiné prostředky, ale nedokáže například dojít z metra na tramvaj. Aktivita dostane plán a agent musí být na první zastávce z plánu. Zatím dostane jen plán po sobě jdoucích stanic a ne linky, to se změní, až bude nový plánovač popsaný v následující kapitole nasazen v praxi. Nejdříve se podívá, jestli na zastávce nestojí vozidlo požadované linky, pokud ne,

tak začne čekat. Pokaždé, když na zastávku přijede nějaký prostředek, tak se podívá, kam nejdále s ním může dojet. Pokud nejede vůbec v jeho směru, tak čeká dál, pokud s ním může jet alespoň jednu zastávku, tak nastoupí, přestane čekat (odregistruje sensing). Během transportu se dívá pokaždé, když zastaví na nějaké stanici a až dojede na určenou stanici, tak vystoupí. Pokud ještě nedojel na poslední stanici v plánu, tak zase začne čekat. Jakmile vystoupí, na poslední stanici, tak aktivita skončí.

- K ježdění autem se používá **DrivingTo** aktivita. Zahrnuje chůzi k vozidlu, jízdy včetně zaparkování a chůzi k cíli. Nejdříve agent dojde pěšky k místu, kde je zaparkované určené vozidlo, potom nastoupí, dojede do cíle a zaparkuje. Po zaparkování ještě dojde pěšky na místo určení a aktivita skončí. Má více způsobů použití: používání svého primárního auta, používání nějakého svého vozidla a používání jiného vozidla. Pokud chce jet svým primárním autem, tak se aktivita zavolá pouze s parametrem cíl cesty. V paměti má agent uloženo jméno auta i kde je zaparkováno, po zaparkování se místo zaparkování přepíše. Pokud chce použít jiné své vozidlo, musí se jako parametry zadat klíč do paměti k jménu vozidla a klíč k místu zaparkování, o zapsání místa zaparkování se postará aktivita sama. Pokud ale chce použít jiné vozidlo, tak se jako parametry zadají jméno vozidla a uzel, na kterém je zaparkované. Po skončení aktivity se musí zpětně získat místo zaparkování, protože nemusí být zaparkované u cíle cesty. Je už na rodičovské aktivitě, kam místo uloží.

- Samotné řízení vozidla poskytuje aktivita **DrivingVehicle**, zahrnuje nastupování do vozidla, dojetí do cíle, zaparkování a vystoupení. Předpokládá se, že agent je na uzlu s vozidlem nebo sedí ve vozidle, pokud v něm nesedí, tak nastoupí. Dále naplánuje cestu na grafu silnic a dojede do cíle, kde může nebo nemusí zaparkovat. Pokud nelze zaparkovat na žádné hraně sousedící s cílovým uzlem, tak začne jezdit zcela náhodně, ale nevrací se na poslední navštívený uzel. Jakmile najde uzel, s kterým sousedí volná hrana, tak na hraně zaparkuje. Pokud se mu nepodaří zaparkovat, tak hledá místo dál. Samotné ježdění funguje na stejném principu jako chůze jen s několika rozšířeními. Pokud na semaforu svítí červená, tak zastaví a začne čekat na zelenou, až pak se zase rozjede. Pokud vozidlo musí jezdit frontami, tak se před každou frontou nejdříve podívá, jestli se do ní vejde, pokud ne, tak začne čekat na volné místo.

4.3.4 Ostatní aktivity

- Občané zatím pracují pouze jedním způsobem a to v kanceláři. Aktivita **WorkingInOffice** zahrnuje pouze nastoupení do firmy jako pracovník, pak čeká zadanou dobu a nakonec odejde z firmy.
- Podobně je udělaná **SpendingTimeAt** aktivita, ta představuje nakupování, bavení se v zábavě a jezení v restauraci. Jediný rozdíl oproti WorkingInOffice je ten, že do firmy nenastupuje jako pracovník, ale jako host. Tedy je to jakékoliv využívání služeb firmy jako host.
- Nakonec máme pomocnou **Null** aktivitu, která jenom čeká určený čas a nic nedělá. Tato aktivita je využívána například jako spánek.

Kapitola 5

Navigace

V předcházející kapitole bylo popsáno chování agenta v prostředí, ale nebylo popsáno, jak plánuje. Tato kapitola je zaměřená na plánování cesty v mapě, popisuje hlavně multimodální plánovač, který najde cestu ze startu do cíle s využitím prostředků MHD. Hlavním přínosem této kapitoly jsou heuristiky pro plánovač. Část kapitoly je věnovaná existujícím plánovačům, ale z nich implementovaný plánovač vychází.

5.1 Multimodální plánovač

5.1.1 Motivace

Aktivity agentů reprezentují schopnost chodit po silnicích a jezdit MHD, ale aby věděli kam jít a čím jet, musí nejprve naplánovat cestu. Pokud chtějí jet autem, tak naplánují algoritmem A* (P. E. Hart; N. Nilsson; B. Raphael, 1968) s heuristikou, která jako hodnotu dává přímou vzdálenost k cíli. Pokud chtějí jenom chodit, použijí stejný plánovač, ale pokud chtějí naplánovat cestu pomocí chůze a využití MHD prostředků, tak musí plánovat jinak. Vozidla vyjíždí a přijíždí do stanic podle jízdních řádů, tedy plánovač musí mít nějakou informaci o čase. Jako dosavadní cena cesty už nebude procestovaná vzdálenost, ale aktuální čas, protože se do cíle chce dostat co nejdříve, v našem případě je to jeho jediné kritérium.

5.1.2 Definice problému

Mějme graf $G = (V, E, tt)$, kde V je množina vrcholů, E je množina orientovaných hran a funkce $tt(e)$ (time-table) přiřadí každé hraně $e \in E$ množinu přejezdů vozidla. Jeden přejezd (transfer) $tr = (tl, ta) \in tt(e)$ se skládá z odjezdového tl (time leave) a příjezdového ta (time arrive) času. Jeden dotaz se

skládá ze startovního uzlu *start*, cílového uzlu *goal* a startovního času t_0. Validní cesta *path* na grafu G je posloupnost n přejezdů $tr_1, \ldots, tr_n$ po hranách $e_1, \ldots, e_n$, $tr_i \in tt(e_i)$ za těchto podmínek: $tr_1 = (tl_1, ta_1)$, kde $t_0 \leq tl_1$, $e_1 = (start, v_1)$. Dále $\forall i \in < 1, n >$: $tr_i = (tl_i, ta_i)$, $tl_{i+1} = (tl_{i+1}, ta_{i+1})$, $ta_i \leq tl_{i+1}$ a $e_i = (u_i, v_i)$, $e_{i+1} = (u_{i+1}, v_{i+1})$, potom $v_i = u_{i+1}$. Poslední přejezd $tr_n = (tl_n, ta_n)$, $e_n = (u, goal)$. Označíme P množinu všech validních cest, potom optimální cesta $path* = argmin_{path \in P}(tl_{last} \in path)$. To znamená, že se snažíme najít nejmenší dobu trvající cestu po hranách a vždy jedeme nějakým prostředkem, který musíme časově stihnout.

5.2 Existující multimodální plánovače

Téměř všechny existující plánovače hledají optimální řešení, jako základ používají dijkstrův algoritmus (E. W. Dijkstra, 1959) na hledání nejkratších cest do všech uzlů ze startu. Jejich urychlení spočívá v preprocesingu případně v obousměrném prohledávání, pokud je to možné. V (Thomas Pajor, 2009) jízdní řády zjednodušili na intervaly, potom preprocesovali celý graf a udělali z něj časově nezávislý, tedy mohli použít obecné algoritmy na hledání nejkratší cesty. My ale máme přesné jízdní řády, proto se budeme zabývat hlavně algoritmy pracujícími s přesnými jízdními řády. Existuje několik způsobů předzpracování, které urychlí dijkstrův algoritmus.

Preprocesing zvaný Landmarks (Daniel Delling; Dorothea Wagner, 2009, 5-7) vybere množinu $L \in V$ takzvaných landmarků a pro každý landmark $l \in L$ se předpočítají vzdálenosti do všech uzlů $v \in V$. Vzdálenosti se potom použijí jako heuristika, tedy dijkstrův algoritmus se změní na A*. Efektivita urychlení algoritmu spočívá na výběru množiny L, nejlepší by bylo předpočítat všechny vzdálenosti, tedy $L = V$, to by ale zabralo příliš paměti. Tedy je dobré vybrat jenom určité uzly.

Přístup Contraction (Daniel Delling; Dorothea Wagner, 2009, 9-10) z grafu odstraní zbytečné uzly, tedy se vytvoří zkratky. Iterativní proces odstraňuje určité uzly, například uzly s ≤ 2 sousedy a okolní uzly spojí novými hranami s přepočítanými délkami. Předzpracování Access-Nodes zvolí množinu $N \in V$ a potom se v každém $v \in V$ předpočítá vzdálenost do všech $n \in N$. Předpočítaná vzdálenost se potom použije v algoritmu A* jako heuristika. Kvalita opět závisí na zvolených uzlech, dobré je volit frekventované uzly jako například přestupní stanice, v (Thomas Pajor, 2009) volili hlavně letiště.

Další metody předzpracování jsou například Arc-Flags (Daniel Delling; Dorothea Wagner, 2009, 7-9), který rozdělí graf na buňky a přiřadí hranám flagy. Dále SHARC (R. Bauer; D. Delling, 2008), který je kom-

binace Arc-Flags a Contradiction, sestaví hierarchii cest.

5.3 Implementovaný plánovač

5.3.1 Preprocesing

Ještě před samotným plánováním se preprocesují data z prostředí simulace. Plánovač pracuje s množinou grafů a jízdních řádů. Jeden graf je graf cest pro chůzi, na tomto grafu nejsou žádné jízdní řády, vyjít po hraně je možno v kterémkoliv čase, výsledný čas po přechodu pro hraně e se přepočítá $t = t + length(e)/walk\_speed$, kde $length(e)$ je délka hrany e a $walk\_speed = 4.7km/h$ je rychlost chůze. Dále pro každý typ MHD je další graf, tedy v našem případě máme celkem čtyři grafy: cesty a linky tramvají, autobusů a metra. Uzly grafů linek MHD jsou pouze zastávky, nejsou zde žádné další uzly, hrana mezi uzly je jedině tam, kde je přímý spoj mezi stanicemi. Jízdní řády jsou zatím zjednodušeny, nejsou pro hrany, ale pro uzly a linky, není žádná specifikovaná spojitost mezi uzly jedné hrany jedné linky. Na každém uzlu je pro každou linku, která tam jezdí, v každém směru jízdní řád odjezdů. Následující stanice linky má také pouze jízdní řád odjezdů, ale jízdní řády byly generovány podle linek a rychlostí prostředků, takže časy odpovídají. Mějme hranu $e = (u, v)$, linku x, která jede z uzlu u do uzlu v a jízdní řády uzlů pro linku x tt'_{ux} a tt'_{vx}, pak pro jeden určitý odjezd $tr'_{ux} = (tl_{ux})$ najdeme čas příjezdu $tl_{vx}* = argmin_{tr'_{vx}=(tl_{vx})\in tt'_{vx}}(|tl_{vx} - wait_x - estimated\_arrive\_time|)$, kde $wait_x$ je čas daného vozidla, který čeká na zastávce a $estimated\_arrive\_time = length(e)/average\_speed_x$, kde $average\_speed_x$ je průměrná rychlost daného prostředku na lince. To znamená, že spočítáme, za jak dlouho pravděpodobně vozidlo dojede na druhou stanici a najdeme nejbližší odjezdový čas bez čekací doby.

5.3.2 Plánovač

Jeden dotaz do plánovače se skládá ze startovního uzlu *start*, cílového uzlu *goal* a startovního času t_0. Pokud existuje cesta ze startu do cíle, tak vrátí cestu jako posloupnost přesunů. Tento přesun už není pouze odjezdový a příjezdový čas, ale $transfer_i = (tl_i, ta_i, line\_id_i, v_i)$, kde tl_i a ta_i jsou odjezdový a příjezdový čas, $line\_id_i$ je název linky včetně směru ($null$ znamená chůzi) a v_i je cílový uzel. Výsledná cesta potom je $path = (transfer_0, transfer_1, \ldots, transfer_n)$, kde $transfer_0 = (t_0, t_0, null, start)$ a ostatní přesuny udávají nalezenou cestu.

Agenti v simulaci nepotřebují nejkratší cesty do všech uzlů ale pouze

do cíle, myšlenka tedy je navštívit co nejméně uzlů. I pokud bychom měli plánovač se složitostí $O(n)$, který projde všechny uzly, tak ve výsledku bude velice pomalý, pokud budeme mít velkou mapu a agent se bude chtít dostat pouze o pár bloků dál. Potom se plánovač s vyšší složitostí, například $O(n^3)$, ale vzhledem k nalezené cestě, se bude oproti lineární složitosti jevit jako konstantní. Například pokud bychom měli 1.000 uzlů, cesta by měla 10 uzlů, tak lineární algoritmus by udělal 1.000 iterací zatím co složitější pouze 100. Chceme nezávislost výpočetního času na velikosti mapy, pouze na délce cesty. Zvolili jsme plánovač A*, tedy drží si seznam vyexpandovaných (open) a uzavřených uzlů (closed), vždy expanduje uzel s nejmenším součtem dosavadní ceny a odhadnuté heuristiky, jak ukazuje popis 2. Dosavadní cena uzlu je příjezdový čas na aktuální uzel, různé heuristiky budou popsány v následující podkapitole. Když expanduje uzel, tak nejdříve expanduje po cestách, tedy do seznamu open přidá uzly, kam se lze dostat pěšky. Potom expanduje všechny stanice na aktuálním uzlu, vezme všechny odchozí linky, najde nejbližší spoj a do seznamu open přidá uzel, kam se lze danou linkou dostat jedním přejezdem.

Algorithm 2 Multi-modal planner A*

open $\leftarrow \{start\}$
closed $\leftarrow \{\}$
while *open* $\neq \{\}$ **do**
  node $\leftarrow best\_from(open)$
  closed $\leftarrow closed \bigcup \{node\}$
  if *node* = *goal* **then**
    return reconstruct_path(node)
  end if
  for all $expanded\_node_i \leftarrow expand(node)$ **do**
    if $expanded\_node_i \notin closed$ **and** (($expanded\_node_i \in open$ **and** $expanded\_node_i$ is better than the node in *open*) **or** $expanded\_node_i \notin open$) **then**
      $open \leftarrow open \bigcup \{expanded\_node_i\}$
    end if
  end for
end while
return no_path

5.4 Testované heuristiky

Pro A* jsem vytvořil několik heuristik, dělí se na dva typy:

- Koncové - mají na vstupu aktuální stav (aktuální uzel a čas), cíl a graf, spočítají odhadovaný čas k cíli
  - Přímo metrem - za jak dlouho se dostane k cíli, pokud by jel přímo metrem
  - Přímo pěšky - za jak dlouho se dostane k cíli, pokud by šel přímo pěšky
- Upravující - navíc mají na vstupu minimálně jednu podřazenou heuristiku, přepočítají hodnotu podřazené heuristiky
  - Oříznout graf elipsou - uzly mimo elipsu vynásobí v závislosti na vzdálenosti od elipsy
  - Vynásobit stanice a ostatní uzly - uzly se stanicemi vynásobí konstantou, uzly bez stanic jinou konstantou
  - Přepínač heuristik - podle vzdálenosti k cíli přepíná dvě heuristiky

Z těchto heuristik se dají stavět stromy. Jediná přípustná heuristika je přímo metrem, všechny ostatní ve většině případu nejsou přípustné, ale pokud předpokládáme rozumně navržené město, tak nebudou dávát o mnoho horší řešení než optimum a mohou značně urychlit výpočet. Například pokud bude graf hustý a nebudou zde příliš velké okliky, tak oříznutí grafu dostatečně velkou elipsou velice pravděpodobně neodstraní uzly na optimální cestě. Nebo pokud agent většinu cesty pojede MHD prostředky, tak umělé podhodnocení uzlů bez MHD také pravděpodobně příliš nezhorší nalezenou cestu. Poměry zrychlení výpočtu a zhoršení nalezené cesty budou ukázány v kapitole 7.

V plánování cesty po městě optimalita nehraje zase takovou roli, pokud například pojede 46 minut místo 45, tak to nebude nikomu vadit, natož v naší simulaci. Žádná heuristika nepenaltuje přestupy, reálný člověk by nechtěl moc často přestupovat, ale to zatím v simulaci neřešíme. Pokud budeme později uvažovat, že občané neradí přestupují, tak bude jednoduché vytvořit další upravující heuristiku.

5.4.1 Přímé heuristiky

Tyto heuristiky spočítají přímou vzdálenost k cíli a dále spočítají čas potřebný k dosažení cíle. Označím d vzdálenost vzdušnou čarou od aktuálního uzlu

k cíli. Potom přímo metrem vrací $d/average\_speed_{metro}$ a přímo pěšky $d/walk\_speed$. Neboli za jak dlouho je nejrychleji se možno dostat přímo daným způsobem do cíle. Přímo metrem je přípustná heuristika, protože vždy dává optimističtější odhad než je realita. Nikdy není možno se do cíle dostat rychleji, než kdyby jel z aktuálního uzlu přímo metrem, metro je nejrychlejší prostředek v simulaci.

5.4.2 Oříznutí elipsou

Oříznutí grafu elipsou má jako parametry relativní velikost $size$ a sklon $slope$. Označíme hodnotu podřazené heuristiky $h0$, vzdálenost cíle od startu $border$ a součet vzdáleností aktuálního uzlu od startu a od cíle $dist$. Potom oříznutí elipsou vrátí $h0$ pokud $dist \leq border * size$, jinak $h0 + (dist - border * size) * slope$. Neboli vše, co je uvnitř elipsy nezmení a vše co je vně, inkrementuje o vzdálenost od elipsy. Touto heuristikou se zamezí tomu, aby se zbytečně neexpandovaly uzly mimo oblast mezi startem a cílem, protože předpokládáme, že ve městě cesta povede spíše mezi startem a cílem.

5.4.3 Násobení stanic a ostatních uzlů

Vynásobení stanic a ostatních uzlů má jako parametry $mul\_stations$ a $mul\_streets$. Nejdříve zjistí, zda se na aktuálním uzlu nachází nějaká stanice MHD. Pokud je uzel stanice, tak vrátí $h0 * mul\_stations$, jinak vrátí $h0 * mul\_streets$. Jednoduše řečeno heuristiku stanic násobí nějakou konstantou a heuristiku ostatních uzlů jinou konstantou. Pokud uzly bez stanic vynásobíme větší konstantou, tak se nebude expandovat tolik pěšky, ale bude se expandovat hlavně na linkách.

5.4.4 Přepínač heuristik

Přepínač heuristik dostane dvě heuristiky $h0$ a $h1$ a hodnotu prahu $thresh$. Označíme $dist$ přímou vzdálenost aktuálního uzlu od cíle, potom pokud $dist > thresh$, tak vrátí $h0$, jinak $h1$. V závislosti na vzdálenosti k cíli vybere jednu z heuristik. To se hodí například pokud je už blízko k cíli, tak už se nebude snažit hledat MHD, ale půjde pěšky přímo do cíle.

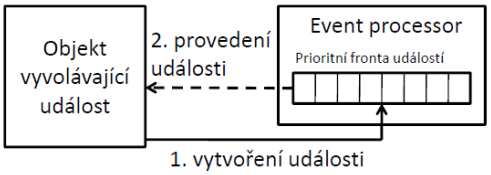
Kapitola 6

Implementace

Tato kapitola popisuje specifickou implementaci frameworku. V kapitolách 3 až 5 byl framework popsán abstraktněji, tato kapitola se zaměřuje na podrobnější detaily. Popsána je vnitřní reprezentace všech objektů, fungování akcí a senzorů, reprezentace grafů, detaily ohledně implementace plánovačů a nakonec fungování Google Earth.

Framework je implementovaný v Javě a využívá framework A-lite vyvíjený skupinou ATG. Dále používá několik externích knihoven:

- Alite
  - vecmath - 2D a 3D geometrie
  - JAK - generování KML a KMZ pro Google Earth
  - jaxb - práce s xml
  - jetty - HTTP server
  - servlet - HTTP server
- Agentpolis
  - libosm - import open street map
  - jcommon - potřebné pro zobrazování grafů
  - joda-time - práce s časem
  - jopt-simple - zpracování argumentů z příkazové řádky
  - jfrechart - zobrazování grafů ve statistikách
  - log4j - výpisy do konzole



Obrázek 6.1: Proces vytváření událostí

6.1 Základ frameworku

Simulace je událostní, tedy čas neběží po konstantních skocích, ale čas se posune vždy s další událostí. Například když se člověk rozhodne jít po hraně, neposunuje se průběžně. Od doby, kdy se rozhodl vyjít až do doby, než dorazí na místo určení je fyzicky pořád na výchozím uzlu, teprve až po uplynutí času přechodu se posune na místo určení. Díky tomu je simulace výrazně rychlejší, než kdyby byla kroková. Pokud se delší dobu nic neděje, tak tato doba bude moci být simulována za konstantní čas procesoru. Při krokové simulaci by doba výpočtu byla lineárně závislá na době simulace.

Simulace je postavená na frameworku A-lite. Jádro simulace je prioritní fronta, která drží všechny odstartované události seřazené podle času dokončení. Pokud nějaká entita zavolá akci, která má nějaký čas trvání, tak se akce pouze uloží jako událost do prioritní fronty. Vlákno simulace postupně bere události z vrcholu fronty a provádí je (obr. 6.1).

Přes A-lite vis se dá regulovat rychlost simulace. Je možno simulaci pozastavit, měnit rychlost, například 10x realtime a nebo je možnost maximální rychlosti, kdy je prodleva mezi událostmi nulová. V tomto módu rychlost simulace závisí pouze na množství událostí. Pokud se v noci ve městě moc věcí neděje, tak simulace jede velice rychle a celá noc trvá například vteřinu. Jakmile začnou občané chodit do práce a plánovat trasy, tak se rychlost simulace zpomalí. V celé simulaci se jako jednotka času používá milisekunda a jako jednotka vzdálenosti metr.

6.2 Vnitřní reprezentace jednotlivých objektů

6.2.1 Storage

Stav světa je držen ve storagích, jsou to datové struktury, v kterých jsou uloženy fyzické informace o světě. Jeden storage drží jeden typ dat, například pozice agentů, graf silnic, stavy semaforů nebo registrování MHD vozidel na

Tabulka 6.1: Obyčejné storage

Název storage	Držené informace
ActionCompletionStorage	storage na dokončování akcí
CompanyStrorage	pozice a informace o firmách
EntityPositionStorage	pozice entit (agentů a vozidel)
EntityStorage	seznam entit, seskupování podle třídy a typu
MaxSpeedStorage	maximální rychlosti pro hrany
MHDStorage	registrování MHD vozidel na stanicích
NodeNameStorage	reálné názvy uzlů (např. Karlovo Náměstí)
PlannerStorage	předzpracovaná data pro plánovač
PositionLinkInfoStorage	spojení mezi entitami (občan je v metru)
PublicTransportStorage	seznam zastávek a linek MHD
RailwayGraphStorage	graf kolejí
StreetStorage	graf silnic
TransportPlanStorage	grafy z MHD linek a plány mezi stanicemi
VehicleParkingStorage	zaparkovaná vozidla
VisStorage	položky použité čistě pro vizualizaci

zastávkách. Do storagů se zapisuje pouze pomocí akcí a čte se z nich pouze pomocí senzorů. Není možné, aby jeden storage zapisoval do druhého nebo přímý přístup od agentů. Aby storage neměly přímé odkazy mezi sebou, všude se dotazuje přes textové ID do hash map. Každá entita, uzel, hrana a firma má své unikátní ID. Jsou dva typy storagů:

- Obyčejné - pouze databáze
- Událostní - obyčejné rozšířené o možnost vytvářet události

Obyčejné storage pouze drží informace, odpovídají na dotazy a provolávají sensingy, událostní storage navíc vytváří události. Například storage na statistiky každých 10 minut zavolá další událost, aby spočítal agregované statistiky za daný interval, nebo storage na semafory přepíná barvy. Seznam všech obyčejných storagů je v tabulce 6.1 a seznam událostních storagů v tabulce 6.2. Obyčejné storage jsou pouze databáze, událostní storage jsou o něco složitější a budou popsány podrobněji.

6.2.2 Storage na fronty

QueueStorage drží pro každou hranu jednu frontu. Jakmile vozidlo vstoupí do fronty, tak storage vytvoří událost, která se vykoná v čase, kdy vozidlo teoreticky dojede na konec hrany. Jakmile se událost provede, tak pokud je

Tabulka 6.2: Událostní storage

Název storage	Držené informace
QueueStorage	drží vozidla čekající ve frontě včetně jejich pořadí
StatisticsStorage	storage pro statistiky (jednotlivé záznamy)
TimeStorage	drží aktuální reálný čas (nevytváří události)
TrafficLightStorage	kontrolery a semafory

vozidlo první ve frontě, tedy není před ním žádné jiné vozidlo, tak vystoupí z fronty a akce pohybu po frontách se dokončí. Pokud vozidlo není první a jsou před ním nějaká vozidla, tak se sečte celková délka vozidel před ním a spočítá se, za jak dlouho vozidlo ujede tuto vzdálenost, vytvoří se další událost a takto pořád dokola, dokud vozidlo nevyjede.

6.2.3 Storage na semafory

TrafficLightStorage drží seznam kontrolerů na křižovatkách, dále každý kontroler drží seznam jednotlivých semaforů ve všech směrech. Kontroler vytváří události a přepíná jednotlivá světla všech semaforů. Jeden kontroler nemusí být nutně pouze na jednom uzlu, pokud je složitější křižovatka, která pokrývá více uzlů, v kontroleru jsou jednotlivé semafory na více uzlech a samotný kontroler je přiřazen více uzlům. Tento případ zatím v prostředí nemáme, ale je možné ho udělat.

6.2.4 Storage na statistiky

StatisticsStorage je sama o sobě také pouze databáze, ale má možnost přidávat postprocessory. Drží seznam záznamů seřazených podle času přidání, základní záznam má tyto položky:

- Čas zapsání
- ID vlastníka (entity, která zapsala záznam)
- Textovou zprávu
- Typ
- Uzel, na kterém entita byla v době zapsání

Ne všechny položky musí být nutně vyplněny, například záznamy z postprocessorů mají ID “Common” a žádný uzel. Dále jsou rozšiřující záznamy, které obsahují kromě základních položek ještě dodatečné:

Tabulka 6.3: Typů záznamů

Typ záznamu	Význam
BODY_STATE	agent reportuje svůj stav
BODY_GET_CAR	nastupování/vystupování z vozidla
BODY_IN_CAR	záznamy během řízení vozidla
BODY_MHD	jízda v MHD prostředku
WALKER_ADULT_STATES	poměr stavů občanů
BODY_INFO	textové informace
WALKER_TIME_TRAVELLING	graf dojezdových časů za den
WALKER_TIME_TRAVELLING_10	graf dojezdových časů za 10 minut
MHD_OVERFLOW_10	graf vytíženosti MHD

- RecordEnum - stav agenta
- RecordNum - jedno číslo (např. dojezdová doba)
- RecordNums - pole čísel (např. poměry stavů agentů)

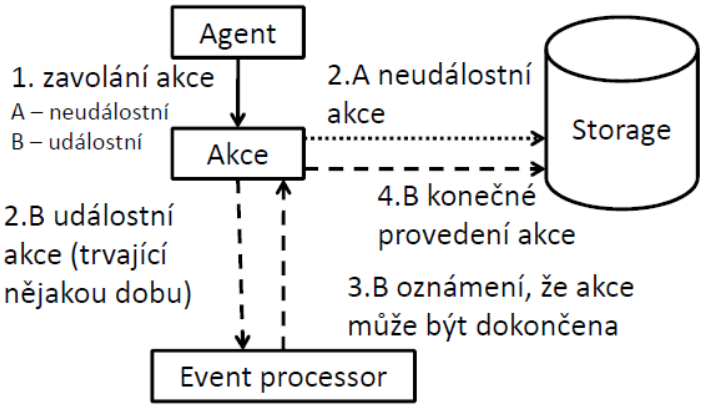
Podle ID vlastníka a typu záznamu se ve vizualizátoru statistik filtrují zobrazené záznamy, typy záznamů jsou v tabulce 6.3.

Postprocessor odchytává jednotlivé příchozí záznamy a určitým způsobem je zpracuje. Buď může po každém záznamu přidat další, nebo po n záznamech přidat další anebo může mít interval, po kterém zapíše výsledek. Jsou používány tyto postprocessory:

- StatsCommutingTimePostprocessor - zapisuje průměrný dojezdová čas každých 10 minut a každý den v 18:00
- StatsMetroOverflowPostprocessor - zapisuje zaplněnost metra každých 10 minut(úspěšné a neúspěšné nástupy)

6.2.5 Akce a senzory

Akce provádějí změny ve storagích, jsou dvojího typu: obyčejné a událostní. Obyčejné se provedou okamžitě, například registrace MHD vozidla na zastávce. Událostní trvají nějakou dobu, například pohyb po hraně. V okamžiku zavolání akce se pouze vytvoří událost, ale nic dalšího se neděje. Akce se provede celá najednou, až když se provede událost (obr. 6.2). Tedy pokud chce agent jít do uzlu a cesta bude trvat 10 vteřin, tak celých 10 vteřin bude stát na původním uzlu a až po uplynutí doby se okamžitě přesune na další uzel.

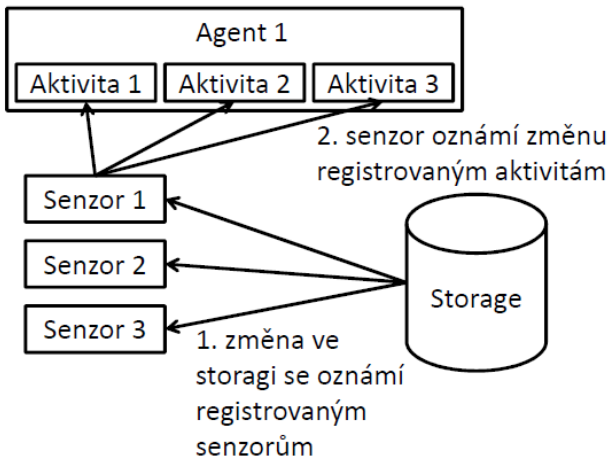


Obrázek 6.2: Proces provádění akcí

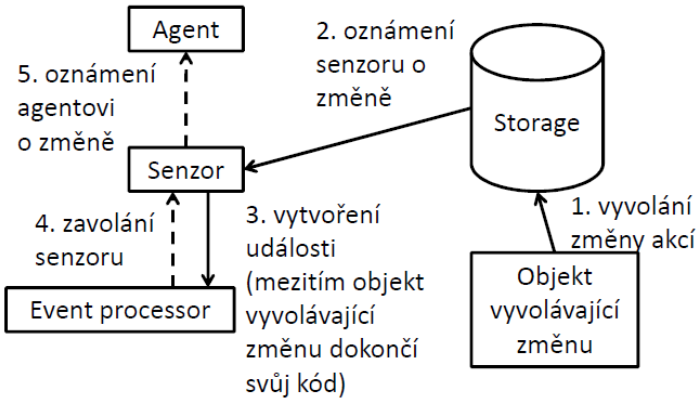
Senzory mají dvojí použití: dotaz a oznamování o změnách. Dotaz pouze přes senzor získá informaci ze storage a vrátí jí. Oznamování o změnách funguje tak, že si nejdříve entita registruje sensing do senzoru a ten se registruje do storage. Pokud někdo vyvolá akcí změnu ve storagi, tak storage oznámí změnu všem registrovaným senzorům a každý senzor všem registrovaným aktivitám daného agenta (obr. 6.3). Senzor neoznamuje aktivitám změnu okamžitě, ale se zpožděním 1ms simulačního času, aby akce vyvolávající změnu ve storagi stihla dokončit svůj kód, než aktivity začnou reagovat na změnu (obr. 6.4). Například když si aktivita agenta registruje senzor na zastávce, tak se konkrétní sensing nejdříve registruje do senzoru (kvůli možnosti více sensingů jednoho agenta z více aktivit) a dále se senzor registruje do storage. Jakmile přijede vozidlo, akcí se registruje do storage a zároveň akce zavolá všechny příslušné senzory a ty zavolají své sensingy se zpožděním, až potom aktivita může reagovat.

6.2.6 Grafy

Jeden graf obsahuje seznam uzlů a hran spojující uzly. Každý uzel na mapě má své ID, v jednom grafu může být pouze jeden uzel s určitým ID. Pokud má více uzlů z různých grafů stejnou pozici, potom mají i stejné ID, například uzel silnice a stejný uzel kolejí jako stanice metra nebo tramvaje. V jednom grafu může být také pouze jedna hrana s určitým ID. ID hrany je tvořeno pouze ID spojovaných uzlů s ohledem na orientaci, neorientované hrany jsou zdvojené a převedené na orientované. Z toho plyne, že také více grafů může obsahovat hrany se stejnými ID, například pokud vedou ko-



Obrázek 6.3: Provolávání sensingů po změně ve storagi



Obrázek 6.4: Proces volání sensingů

leje tramvaje po silnici, hrana kolejí i silnice má stejné ID. Uzly mají mapu odchozích a příchozích hran, kde klíč je druhý uzel hrany. To slouží pro rychlou expanzi při plánování nebo pro rychlé získání hrany do určitého souseda. Kvůli lepší vizualizaci hrana obsahuje seznam bodů, kterými je tvořena, například pokud je zatáčka, bylo by neefektivní vložit do grafu každý uzel zatáčky a spojit hranami. Celá zatáčka se degraduje do jedné hrany, které se spočítá reálná vzdálenost, tedy délka hrany není vždy vzdálenost spojovaných uzlů.

Pomocí těchto ID se potom dotazuje do storagů, například pokud chceme získat stanici na uzlu, dotážeme se ID uzlu, nebo pokud chceme maximální rychlost na hraně, dotážeme se ID hrany. Stejně to platí pro parkovací místa, firmy na uzlu, skutečná jména uzlů, plány mezi stanicemi nebo fronty. Z toho plyne, že nemůže být například různá maximální rychlost pro auta a tramvaje na stejné hraně. Sice by se storage dal rozšířit i o ID grafu, ale pro naše účely je dosavadní řešení dostačující.

Linky MHD nejsou primárně tvořeny grafem (třídou Graph jako silnice nebo koleje), graf je výsledek postprocesingu PublicTransportStorage. Na určitých uzlech jsou MHD zastávky, pro každý typ MHD je jedna zastávka, tedy pokud na jednom uzlu je zastávka pro autobusy i tramvaje, každá zastávka je jiný objekt. Stanice metra jsou vždy na samostatném uzlu, graf kolejí metra nemá společné uzly s grafem silnic. Každá stanice obsahuje seznam linek, linka je tvořena seznamem stanic. Pro každý směr je vždy jedna linka.

6.2.7 Entity

Jak už bylo řečeno, entity mají také své unikátní ID, v našem případě je ID reálné jméno, například Agent128, Josef Novák 2, Car Agent 3 nebo Metro A (6). Generátor reálných jmen máme, ale zatím není používaný, později se použije při generování rodin, kdy všichni členové rodiny budou mít stejné příjmení. Entity se dělí na občany a vozidla. Firmy jsou sice také entity, ale jsou statické a není potřeba jim věnovat větší pozornost. Agenti i vozidla mají dále typ, jak je popsáno v kapitole 3. Entity mají tyto parametry:

- Agent
  - Typ
  - Pohlaví (zatím použito pouze ve vizualizaci)
  - Mozek (LifeCycleActivity)
  - Vlastní paměť

  - Sdílená paměť

- Vozidlo
  - Typ
  - Maximální rychlost
  - Délka
  - Kapacita
  - ID linky (použito pouze u MHD vozidel)

V samotných entitách není žádná logika, logika je pouze v mozku agenta. Všechny entity mají metodu na vygenerování popisu, popis vystihuje statické parametry a aktuální stav dané entity. U agenta popis vypisuje i celou hierarchii právě aktivních aktivit.

6.3 Plánovač A*

Používají se dva plánovače, jeden pro plánování na jednom grafu a druhý pro plánování cesty s využitím MHD. Oba jsou A* plánovače (P. E. Hart; N. Nilsson; B. Raphael, 1968) a mají stejnou strukturu, jediný rozdíl je v reprezentaci aktuálního stavu a v expandování uzlu. Algoritmus byl už popsán v předchozí kapitole pseudokódem 2. Closed seznam je reprezentovaný hash množinou, složitost vložení, odstranění i dotazu je $O(1)$. Do seznamu closed se neukládá celý uzel plánovače ale pouze stav uzlu.

Open seznam je reprezentovaný pomocí dvou struktur, jedna je prioritní fronta implementovaná prioritní haldou. Přidání prvku má složitost $O(log(n))$, odebrání prvního prvku $O(log(n))$ a odebrání libovolného prvku $O(n)$. Druhá struktura je hash mapa, přidání, odebrání i navrácení hodnoty má složitost $O(1)$. Prioritní fronta drží uzly plánovače seřazené podle $value = cost + H$, hash mapa drží pro stavy jejich hodnoty. Při přidání do open se do fronty přidá celý uzel, do hash mapy jen stav z uzlu jako klíč a *value* jako hodnota. Při odebrání prvního prvku se vezme uzel plánovače z fronty, dále se odstraní stav uzlu z hash mapy. Při expandování se u expandovaného stavu nejdříve kontroluje, zda již není v closed, pokud není, kontroluje se, zda není v open. Pokud je v open hash mapě, tak se zkontroluje, zda nový stav má menší hodnotu. Pokud nemá, tak se nový stav zahodí, pokud má lepší hodnotu, tak se odstraní z prioritní fronty. Dále pokud nový stav nebyl dosud zahozen, tak se přidá do prioritní fronty a vloží/přepíše se hodnota v hash mapě. Protože hash mapa má konstantní složitost na všechny používané operace, výslednou složitost určuje prioritní fronta.

Protože plánujeme ve městě, předpokládáme, že uzly mají malý počet sousedů, v závislosti na celkovém počtu hran můžeme říci, že každý uzel má konstantní počet odchozích hran. Potom složitost plánovače v závislosti na všech dotčených uzlech je $O(n^2)$.

Rozdíl mezi plánovači je pouze v reprezentaci aktuálního stavu a expandování. Uzly plánovače obsahují:

- Předchozí uzel (null pro start)
- Dosavadní cena
- Hodnota heuristiky

Jednotka ceny a heuristiky závisí na plánovači, na jednom grafu to je vzdálenost v metrech, u multimodálního plánovače čas v milisekundách. Stavy uzlů plánovačů se skládají z:

- Na jednom grafu
  - Uzel na grafu
- Multimodální
  - Uzel na grafu
  - Čas odjezdu z minulého uzlu
  - Čas příjezdu na aktuální uzel (= dosavadní cena)
  - ID linky (null pro chůzi)

Expandování uzlů se také liší v závislosti na plánovači, plánovač na jednom grafu vezme pouze přímé sousedy aktuálního uzlu a k dosavadní ceně přičte délku hran. Multimodální plánovač expanduje jednak sousedy uzlu na ulici, čas přepočítá jako dobu chůze, dále pokud je na uzlu stanice, tak vezme sousední stanice. Najde nejbližší spoj do každé přímo dosažitelné stanice a jako dosavadní cenu zapíše příjezdový čas. Testování cíle pouze porovná, zda se aktuální uzel rovná cíli. Výsledná cesta se rekonstruuje odzadu z konečného uzlu pomocí předchůdců v uzlech plánovače.

6.4 Google Earth

Jeden z grafických výstupu frameworku je Google Earth, který zobrazí objekty simulace na přesné místo na zemi, v našem případě zobrazí graf a entity

na Prahu. Google Earth je samostatný program, kterému je potřeba poskytovat data k zobrazení. Zobrazení simulace v Google Earth z uživatelského pohledu je velice jednoduché, stačí otevřít vygenerovaný soubor urbansim.kml a vše se zobrazí. Pokud se simulace zastaví nebo ukončí, tak se přeruší obnovování objektů. Pokud simulaci znovu spustíme, je třeba jednou ručně obnovit dynamické objekty, aby se začaly obnovovat automaticky.

6.4.1 Stručně o KML

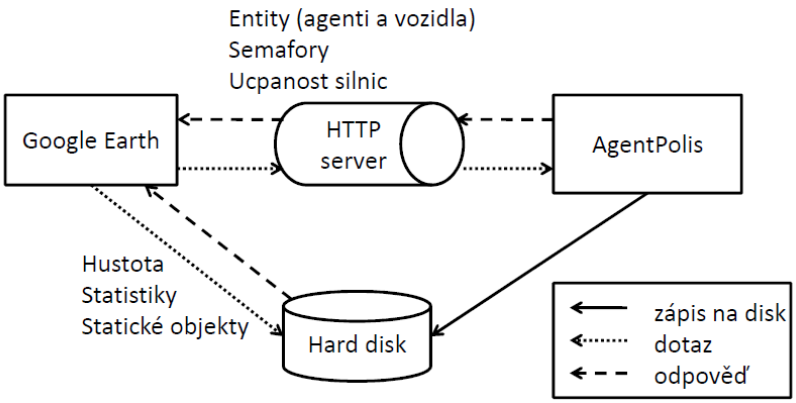
Formát přijímaný Google Earth je KML nebo KMZ, což je zazipované KML [KML]. KML je speciální XML, ve kterém jsou jednotlivé objekty reprezentovány placemarky s různými parametry a styly. Parametry jsou většinou jen pozice, jméno a popis, většina zobrazovaných informací je ve stylech. Pokud chceme vytvořit ikonku, úsečku o určité šířce nebo polygon o určité barvě, tak vždy musíme vytvořit patřičný styl. Každý styl i placemark by měly mít unikátní ID. Ke každému placemarku se dá přiřadit popis v podobě HTML, který se zobrazí, pokud na zobrazený objekt klikneme. Celá struktura placemarků je ve složkách, v GE potom můžeme vypínat a zapínat jednotlivé složky a máme přehledný seznam všech placemarků. Složkám se dají přiřadit meze vzdálenosti kamery, kdy se složka zobrazí, tedy můžeme například při bližším pohledu zobrazovat jednotlivé agenty, při oddálení kamery pouze hustotu.

6.4.2 Proud dat ze simulace do Google Earth

Simulace poskytuje data k zobrazení dvěma způsoby: uložením na disk a poskytováním z HTML serveru (obr. 6.5). V případě uložení na disk framework vygeneruje patřičné KML a Google Earth si ho sám načte. Může ho načíst jednou v případě statických objektů nebo ho může načítat v pravidelných intervalech, případě při posunu kamerou.

Získávání dat z HTML serveru vypadá ze strany Google Earth stejně, jen se místo adresy lokálního souboru vyplní adresa na serveru. Framework při dotazu na určitý soubor vygeneruje nové KML a vrátí ho, Google Earth ho zobrazí. Při dotazu přes HTML server může i Google Earth posílat informace jako například výška a pozice kamery nebo obdélník, který kamera zabírá, takže se generované objekty dají přizpůsobit kameře.

Přijímání dat v Google Earth funguje dvěma způsoby: přepíše stávající soubor nebo updatuje stávající soubor. Pokud přepisuje, tak při načtení nového KML smaže starý a přepíše ho novým. Tedy všechny objekty se vymažou a vytvoří znova, i pokud se jejich stav nezměnil. To způsobuje



Obrázek 6.5: Tok dat v Google Earth

blikání a v případě, že klikneme na nějaký placemark, tak při obnovení se okno s popisem zavře.

Při updatování souboru se nejdříve pošle prvotní KML, v kterém je počáteční stav, například předgenerované styly, dále se přijímá jiné KML, které nese pouze změny oproti poslednímu snímku. Tedy nemusí se posílat celý stav světa, ale jen změny, například změny pozic, změny popisů nebo vytváření a mazání objektů. Oproti prvnímu způsobu už placemarky neblikají a pokud otevřeme popis, tak se při změně pozice nezavře a dokonce se průběžně mění. V souboru změn přijme Google Earth tři složky:

- Create - obsahuje objekty, které v posledním snímku nebyly a v aktuálním jsou
- Change - obsahuje změny již existujících objektů
- Delete - obsahuje objekty, které mají být smazány

Zde je nutné, aby placemarky měly unikátní ID, podle něj se upravuje patřičný objekt.

6.4.3 Implementace AgentPolis pro Google Earth

Z AgentPolis jsou Google Earth poskytována data více způsoby, protože každý způsob má své výhody. Základní struktura urbansim.kml je hlavní složka, která obsahuje odkazy na jednotlivé KML soubory:

- Statistics - obrázky na obrazovce

- Agents - agenti a vozidla
- Static - všechny statické objekty (grafy, firmy)
- Street density - hustota jednotlivých hran
- Traffic lights - semafory
- Density - hustota při pohledu z dálky
- Update links - KML soubory změn (Agents, Street density a Traffic lights)

Statistics zobrazují všeobecné informace o simulaci: počet jednotlivých typů entit, aktuální čas, dále různé grafy a ATG logo. Framework generuje obrázky a ukládá na disk v intervalu 1s. Google Earth obnovuje data také v intervalu 1s ale asynchronně.

Agents zobrazuje entity jako ikonky se jménem a popisem, má podsložky:

- Citizens - občané a MHD řidiči
- Private vehicles - osobní vozidla (auta a kola)
- Public transport - vozidla MHD (metra, tramvaje a autobusy)

KML změn je poskytován přes HTML server. Při dotazu o soubor GE pošle i souřadnice maximálního obdélníku, který zabírá kamera, podle něj jsou vybrány pouze entity v obraze. Posílají se pouze změny oproti poslednímu snímku.

Static zobrazuje graf silnic, cest a meter, dále všechny firmy. Grafy cest a silnic zobrazuje jako úsečky, graf meter jako polygony s příslušnou barvou linky a pětiúhelníky jako stanice, firmy zobrazuje jako ikonky se jménem a popisem. Linky metra jsou zobrazovány jako polygony ve tvaru lomených čar proto, že obyčejné úsečky se zobrazují se stejnou šířkou bez závislosti na vzdálenosti kamery, polygony simulují úsečku, která je z dálky tenčí. Má podsložky:

- Companies - firmy
  - Non guest - pouze pracovní
  - With guests - restaurace, školy, obchody a zábavy
- Graphs - grafy
  - Paths - cesty pro agenty

- Streets - silnice
- Metro - linky metra

Při spuštění frameworku je vygenerovaný jeden KML soubor, který GE jednou otevře.

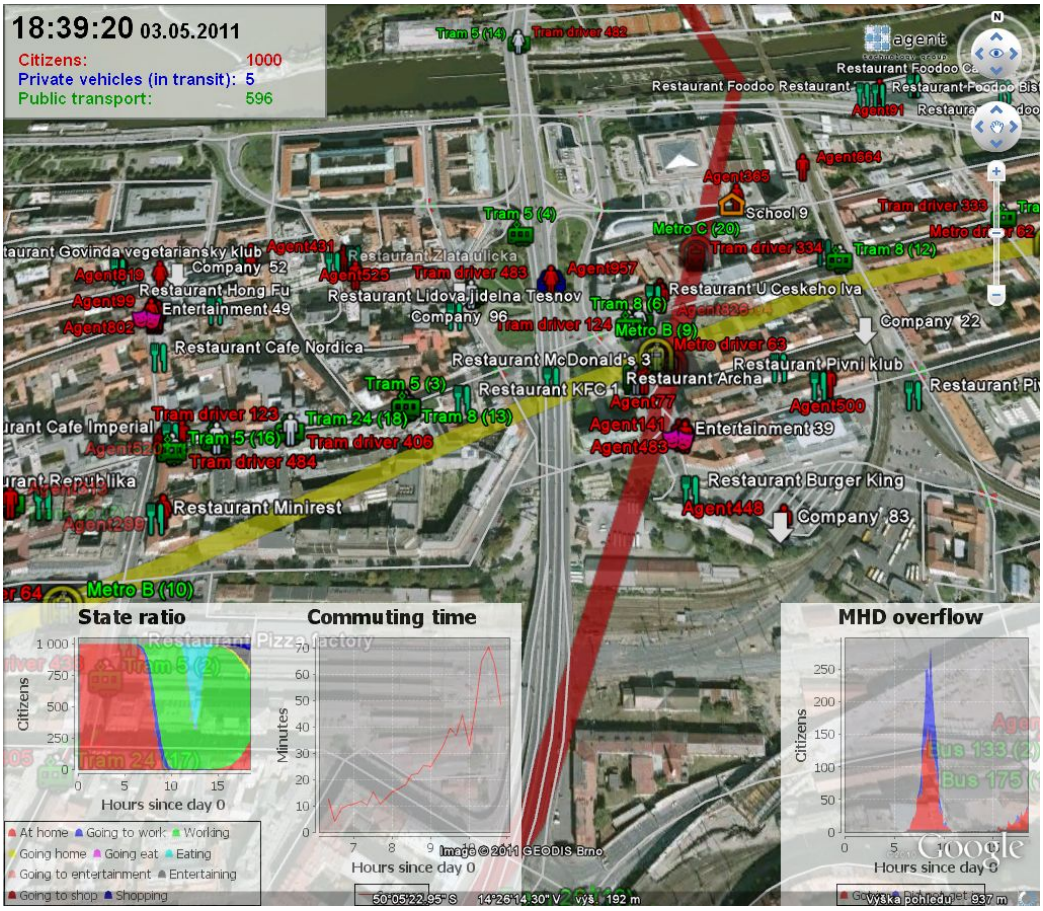
Street density zobrazuje tloušťku hran podle toho, kolik je v příslušné frontě vozidel. Posílání dat funguje stejně jako u agentů, tedy přes HTML server vrací změnu oproti poslednímu snímku, také zobrazuje pouze viditelné hrany. Šířka zobrazené hrany e se spočítá podle vzorce:
$width_e = max(1, min(30, vehicle\_count(queue_e)/length_e * 100))$,
kde $vehicle\_count(queue_e)$ je počet aut ve frontě na hraně e a $length_e$ je délka hrany e.

Traffic lights zobrazuje semafory jako polygony tvaru trojúhelníku. Na křižovatce zobrazí každé jednotlivé světlo jako trojúhelník příslušné barvy směřující do středu křižovatky. Poskytování dat funguje stejně jako u agentů a ucpanosti silnic.

Density zobrazuje hustotu entit jako překryvný obrázek, zobrazuje zvlášť hustotu pro občany, osobní vozidla a vozidla MHD. Nejdříve pro každý typ entit vytvoří histogram do mřížky. Naše mřížka má 300x300 políček a pokrývá celou Prahu včetně nejbližších vesnic. Z histogramu se intenzita barvy (0 až 1) v pixelu $[x, y]$ výsledného obrázku img vypočítá: $img[x, y] = min(1, 1 - exp(-hist[x, y]/n * intensity)$, kde $hist[x, y]$ je počet entit v daném políčku, n je celkový počet entit a $intensity = 100$. Zobrazuje hustotu v logaritmickém měřítku, aby se dala jak zachytit jediná entita, tak rozpoznat rozdíl, když je v mapě velké množství entit. Funguje stejně jako statistiky, tedy v intervalu 1s ukládá obrázky na disk a ve stejném intervalu asynchronně je GE obnovuje.

Update links drží pouze KML soubory pro přijímání změn agentů, ucpanosti silnic a semaforů.

Na obr. 6.6 je vidět většina zobrazitelných objektů. Zobrazené entity jsou: občané červeně, řidiči MHD bíle (s červeným jménem), auta modře, metra barvou linky, ostatní MHD zeleně. Zobrazené firmy jsou: pracovní firmy bíle, zábavy růžově, restaurace zelenomodře, školy oranžově. Dále jsou vidět koleje metra podle příslušné barvy linky a silnice bílou barvou. Obrazovku překrývají průhledné grafy a základní informace o simulaci, tyto grafy je možno zobrazit také z vizualizátoru statistik.



Obrázek 6.6: Ukázka zobrazených objektů v GE (vozidla, agenti, firmy, linky metra, silnice, grafy, info)

Kapitola 7

Evaluace

Tato kapitola se zaměřuje na vyhodnocení všech implementovaných částí. Ukazuje výkonnostní možnosti frameworku a vizualizace, paměťovou náročnost a největší část je zaměřena na výsledky plánovače. Ukazuje optimalitu a výpočetní náročnost jednotlivých heuristik, jejich závislost na parametrech a porovnání.

U frameworku je důležité vědět, jakou má výpočetní a paměťovou náročnost. Čím více je v simulaci agentů, tím je program více paměťově náročnější. Také čím více agentů, tím více událostí je vytvářeno, takže simulace běží pomaleji. V simulaci nejvíce výpočetního času zabere samotné vykonávání událostí, ale řazení událostí do prioritní fronty je jen o trochu rychlejší. Naším cílem je simulovat celou Prahu, tedy přibližně milión agentů.

7.1 Evaluace celého frameworku

7.1.1 Konfigurace testů

Zaměříme se na paměťovou a výpočetní náročnost frameworku. Testy byly prováděny na počítači s dvoujádrovým procesorem AMD Athlon 3GHz 64bit a 4GB RAM DDR3 1066MHz. Dostupná paměť pro Java VM byla nastavena 4GB. Nastavení parametrů simulace je v tabulce 7.1, každé MHD vozidlo má i svého řidiče, celkem je v simulaci 596 MHD vozidel s řidiči. Pro každého občana bylo vytvořeno jedno auto a žádné kolo. Při testech byla vypnutá veškerá vizualizace.

7.1.2 Výsledky

Testoval jsem simulaci pro různé počty občanů. Pro 600.000 agentů došla paměť při vytváření zaparkovaných aut pro každého agenta, při 500.000 agen-

Tabulka 7.1: Konfigurace simulace

Objekt	počet
Metra pro každou linku	24
Tramvaje pro každou linku	20
Autobusy pro každou linku	4
Pracovní firmy	100
Školy	30
Zábavní podniky	50

Tabulka 7.2: Výkonnostní výsledky frameworku za jeden simulační den

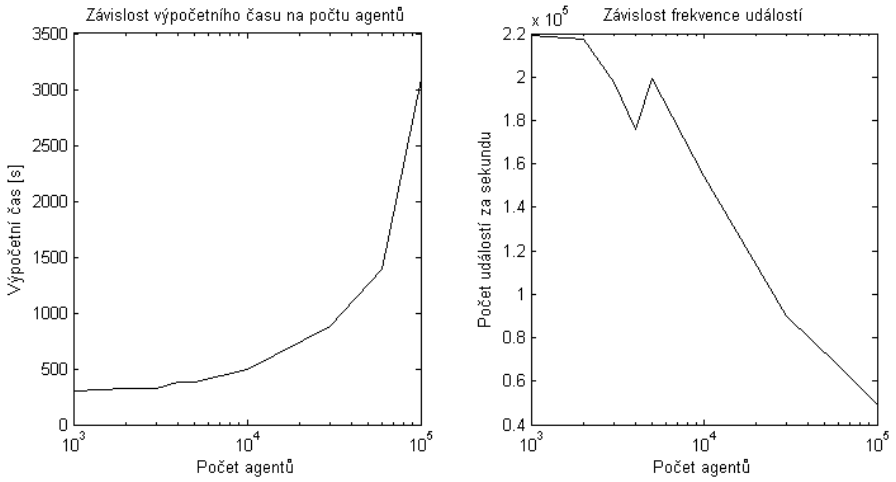
Počet občanů	počet událostí	CPU čas [s]	událostí / s	× realtime
1000	65.923.614	301	219.015	287
2000	70.399.507	324	217.282	267
3000	64.257.782	325	197.716	266
4000	66.568.219	378	176.106	229
5000	76.138.665	382	199.316	226
10000	75.779.966	491	154.338	176
30000	79.198.032	880	89.998	98
60000	93.267.810	1390	67.099	62
100000	152.012.077	3091	49.179	28

tech simulace vždy běžela chvíli přibližně 200x rychleji než realtime, ale vždy po několika sekundách se simulace zasekávala na pořád delší a delší dobu. Simulace se přibližně za 10 minut nedostala do okamžiku, kdy by agenti začali chodit do práce. Dále jsem pro různé počty nechal simulaci běžet vždy jeden den, výsledky jsou v tabulce 7.2 a v grafu obr. 7.1.

7.1.3 Diskuze

Z tabulky 7.2 je vidět, že při menším počtů agentů (jednotky tisíců) rychlost simulace občané téměř neovlivňují. To je způsobeno vozidly MHD, která vytváří nejvíce událostí, protože jezdí pořád od rána do večera, zatím co jeden občan se přesunuje 2x až 5x za den. Také při tak malém počtu velice záleží, jak daleko cestují, je vidět, že při 2.000 agentech se v simulaci stalo o 6 milionů událostí více, než při 3.000 agentech.

Řádově vyšší počet agentů se již začíná projevovat na počtu událostí a také na výpočetním času. Počet událostí už roste přibližně lineárně, ale výpočetní čas roste rychleji než lineárně, to je způsobeno implementací prioritní fronty. Na prioritní frontu byla použita PriorityQueue, která je součástí



Obrázek 7.1: Závislost výpočetního času a počtu událostí za sekundu na počtu agentů (logaritmická osa x)

Javy a je implementovaná pomocí haldy se složitostí přidání prvku/odebráni prvního prvku $O(log(n))$. Jedna událost je vykonána vždy za stejný čas procesoru, ale vložení události do fronty závisí na velikosti fronty, tedy pokud je více agentů, je více událostí, fronta je větší a vložení události trvá déle. Nejlépe je to vidět na počtu událostí za vteřinu v závislosti na počtu agentů (obr. 7.1).

7.2 Evaluace vizualizace

7.2.1 Konfigurace testů

Parametry simulace jsou stejné jako v předchozí podkapitole (tabulka 7.1). Měřit budu výkon A-lite vis a Google Earth, zaměřím se na počet snímků za vteřinu (FPS). Vizualizace je synchronizovaná se simulací, jeden snímek má timeout 1s, pokud nestihne vše vykreslit do daného intervalu, tak se přestane s vykreslováním, ale již vykreslené objekty nezmizí. Další snímek může být generovaný nejdříve 30ms po posledním snímku. U A-lite vis je nastaveno maximálně 25 FPS, u GE je 3.33 FPS.

Tabulka 7.3: Výsledky vizualizátorů - počet snímků/s

Počet občanů	A-lite vis celá mapa	A-lite vis centrum	GE centrum
1000	11	24	2.3
10000	8	24	2.0
30000	5	22	1.6
60000	3	17	1.4
100000	2	9	0.5

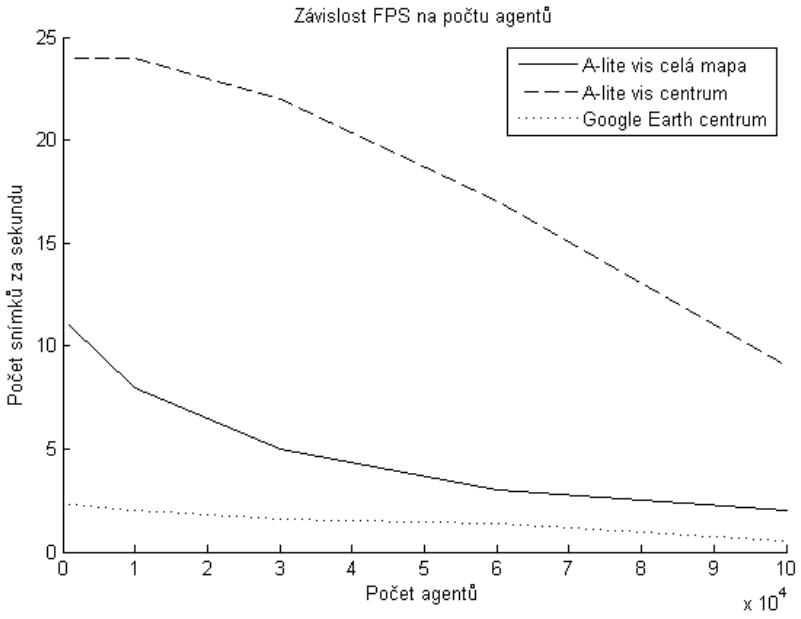
7.2.2 Výsledky

Měřil jsem FPS v závislosti na počtu agentů. V Google Earth jsem posunul kameru do centra a přiblížil tak, aby se zobrazovaly všechny entity, ucpanost silnic byla vypnuta. V záběru kamery nebyly všechny entity. V A-lite vis jsem posunul a oddálil kameru tak, aby byla vidět celá mapa a dále také pouze přiblížené centrum. Pokud se v GE oddálí kamera, tak se entity vůbec nezobrazují a FPS se nedá měřit. Výsledky jsou v tabulce 7.3 a v grafu obr. 7.2. Pokud bylo v záběru kamery GE příliš mnoho entit (přibližně 30.000), tak už generování snímku překročilo timeout 1s.

7.2.3 Diskuze

V tabulce 7.3 jsou vidět hodnoty FPS pro oba vizualizátory. A-lite vis při pohledu na celou mapu není příliš výkonný, ale při pohledu jen na určitou část mapy není FPS příliš ovlivněno počtem agentů. Pro hrubý přehled o entitách je tento vizualizátor vhodnější.

Google Earth je spíše vizualizátor pro koncové uživatele. Zobrazuje simulaci detailně graficky, ale zase je mnohonásobně pomalejší. Nejpomalejší záležitost je vygenerování a následné parsování KML souboru. Plynulé obnovování je téměř nemožné a při vyšším počtu agentů tento vizualizátor selhává, i když nejsou všechny entity v záběru kamery. Dále se občas stane, že se update KML vygeneruje, ale GE ho nezobrazí. AgentPolis si tedy myslí, že GE dostal nový snímek a další update KML bude rozdílem aktuálního snímku a snímku, který GE nedostalo. To způsobuje, že se některé dynamické objekty vůbec nezobrazí nebo se entitám vytvoří mrtvé klony. Předpokládáme, že chyba bude v implementaci Google Earth, protože nebylo stavěné na tak rychlé obnovování tolika objektů. Dokud se nepoužívalo updatování objektů, ale posílalo se vždy KML celého stavu, tak se tyto problémy neobjevovaly.



Obrázek 7.2: Počet snímků za vteřinu jednotlivých vizualizátorů v závislosti na počtu agentů

7.3 Evaluace plánovače

7.3.1 Nastavení testů a metriky

Multimodální plánovač byl testován na třech scénářích a byly měřeny čtyři metriky. Testovány byly různé heuristiky s různými parametry. Jediná přípustná heuristika je přímo metrem, proti ní byly počítány metriky. Předem bylo vygenerováno $n = 100$ testovacích cest, jedna testovací cesta se skládá ze startu, cíle a startovního času. Jako start a cíl byly pro každou testovací cestu vybrané dva náhodné uzly z grafu silnic. Startovní čas je vždy 8:00, tedy ve špičce všech typů MHD. Testované scénáře jsou:

- **Základní intervaly:** běžné vygenerované jízdní řády, popsáno v kapitole 3.2.2, například $time\_tables_{metro} = generateTimeTable(3h, 24h, 3min, 10min, [7h, 17h], [3h, 3h])$, představuje intenzitu spojů ve špičce
- **Intervaly * 3:** základní jízdní řády s třikrát delšími intervaly, například $time\_tables_{metro} = generateTimeTable(3h, 24h, 9min, 30min, [7h, 17h], [3h, 3h])$, představuje intenzitu spojů mimo špičku

- **Intervaly * 8:** základní jízdní řády s osmkrát delšími intervaly, například $time\_tables_{metro} = generateTimeTable(3h, 24h, 24min, 80min, [7h, 17h], [3h, 3h])$, představuje intenzitu spojů v noci

Měřené metriky jsou:

- **Poměr uzlů nalezené cesty expandovaným uzlům (M1)** $= 1/n * \sum_{i=1}^{n} path\_nodes_{Hi}/expanded\_nodes_{Hi}$
- **Poměr expandovaných uzlů oproti expandovaným přípustnou heuristikou (M2)** $= 1/n * \sum_{i=1}^{n} expanded\_nodes_{Hi}/expanded\_nodes_{Oi}$
- **Poměr ceny oproti ceny optimální cestě (M3)** $= 1/n * \sum_{i=1}^{n} cost_{Hi}/cost_{Oi}$
- **Poměr výpočetního času oproti výpočetnímu času přípustné heuristiky (M4)** $= 1/n * \sum_{i=1}^{n} cpu\_time_{Hi}/cpu\_time_{Oi}$

kde *path_nodes* je počet uzlů na nalezené cestě, *expanded_nodes* je počet uzlů, z kterých bylo expandováno, *cost* je celkový (dojezdový) čas cesty a *cpu_time* je výpočetní čas. Dále H značí právě testovanou heuristiku a O přípustnou heuristiku. Vždy je počítán průměr poměru přes všechny testované cesty. Simulační čas je reprezentovaný v milisekundách, výpočetní čas v nanosekundách.

Metrika poměr uzlů nalezené cesty expandovaným uzlům udává, jak byla heuristika informovaná. Hodnota 1 znamená, že A* expandoval jen uzly na nalezené cestě, tedy heuristika byla úplně informovaná. Vyšší hodnota znamená, že heuristika nebyla příliš informovaná, například hodnota 100 může znamenat, že nalezená cesta měla 10 uzlů, ale algoritmus expandoval 1000 uzlů. Tato metrika ale neříká nic o optimalitě, tedy pokud by algoritmus nikdy nebacktrackoval a vždy šel do prvního expandovaného uzlu, až by nakonec našel cestu, tak by hodnota byla 1, ale cesta by byla nepoužitelná.

Poměr expandovaných uzlů oproti expandovaným přípustnou heuristikou udává, kolikrát více byla testovaná heuristika informovaná oproti přípustné. Tato metrika nedává příliš přesný obraz o kvalitě heuristiky, je zde spíše pro ukázku.

Poměr ceny oproti ceně optimální cesty udává kvalitu nalezeného plánu. Pokud je hodnota 1, znamená to, že heuristika našla optimální cestu, vyšší hodnota znamená delší cestu. Pokud je hodnota 1.5, může to například znamenat, že optimální cesta trvala hodinu a nalezená cesta trvala hodinu a půl.

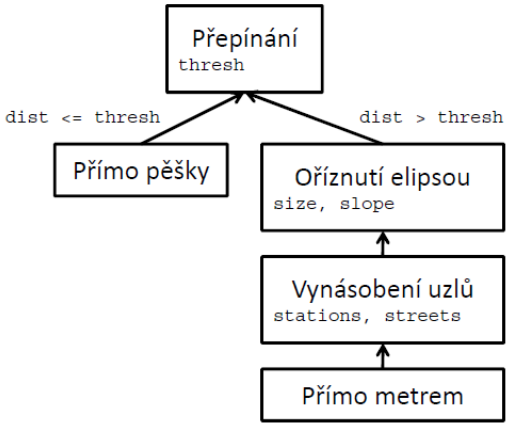
Poměr výpočetního času oproti výpočetnímu času přípustné heuristiky udává, kolikrát rychlejší nebo pomalejší byl výpočet oproti optimální cestě. Hodnota < 1 znamená rychlejší než s přípustnou heuristikou,

> 1 znamená pomalejší. Na tuto metriku nebude kladen příliš velký důraz, protože příliš závisí na implementaci plánovače.

7.3.2 Naměření metrik všech heuristik

Zvolil jsem několik testovaných heuristik, navrhnul několik struktur heuristik a zvolil různé parametry. Heuristiky a parametry jsem volil tak, abych mohl nejdříve porovnat každou upravující heuristiku, dále strukturu heuristik, která se zdála jako nejlepší a pro tuto strukturu různé parametry. Metriky jednotlivých heuristik jsou zobrazeny v tabulkách 7.6 a 7.7, z těchto hodnot budou dále vycházet všechny další experimenty. Přípustná heuristika není uvedena v tabulce, všechny metriky kromě M1 má rovny 1, poměr expandovaných uzlů vůči nalezené cestě je v tabulce 7.5. Testované jsou jednak jednoduché heuristiky a dále struktura zobrazená na obr. 7.3. Pojmenování konkrétních heuristik včetně parametrů je v tabulce 7.4 a dále:

- Hopt - přímo metrem, přípustná
- Hel - elipsa ($size = 0.812$, $slope = 932$)
- Hst - násobení uzlů ($stations * 6.962$, $streets * 7.099$)
- Hth - přepínání přímo metrem a přímo pěšky ($thresh = 372$)
- Hw5 - struktura ($size = 0.8$, $slope = 500$, $stations = 1$, $streets = 4$, $thresh = 358.33$), práh je spočítaný jako vzdálenost, kam dojde pěšky za 5 minut
- Hw10 - stejná heuristika jako Hw5, ale $thresh = 716.66$, neboli kam dojde za 10 minut
- Hw15 - stejná heuristika jako Hw5, ale $thresh = 1075.00$, neboli kam dojde za 15 minut



Obrázek 7.3: Struktura testované heuristiky, *dist* je přímá vzdálenost aktuálního uzlu od cíle

Tabulka 7.4: Parametry struktury heuristik

Heuristika	size	slope	stations	streets	thresh
H1	80.575	663.718	20.546	60.171	2382
H2	-1.114	1088.544	23.747	27.001	1645
H3	14.700	482.000	-14.000	87.000	1705
H4	11.102	419.825	2.730	100.340	2950
H5	0.794	310.071	14.159	16.189	3057
H6	1.072	613.977	9.099	9.195	1289
H7	0.300	256.654	5.258	6.004	2182
H8	0.608	261.267	4.993	4.730	-165

Parametry heuristik Hw5, Hw10 a Hw15 byly nastaveny ručně, parametry ostatních heuristik byly nalezeny optimalizací. Použil jsem CMA evoluční strategii (Hansen N.; Ostermeier A., 2001), minimalizoval jsem jedno kritérium $M3 * c + M1$, kde konstanta c dává váhu optimalitě, metriky byly průměrované přes všechny scénáře. U heuristik Hel, Hst a Hth jsem nastavil $c = 100$, u heuristik H1 až H8 jsem hodnotu měnil přibližně v rozmezí 10 až 200, abych dostal více výsledků a mohl porovnat optimalitu vůči expandovaným uzlům. Heuristiky už jsou seřazené podle optimality, H1 dává nejméně optimální výsledky ale nejrychleji, H8 dává výsledky velmi blízké optimalitě ale nejpomaleji.

Tabulka 7.5: Poměr uzlů nalezené cesty expandovaným uzlům Hopt

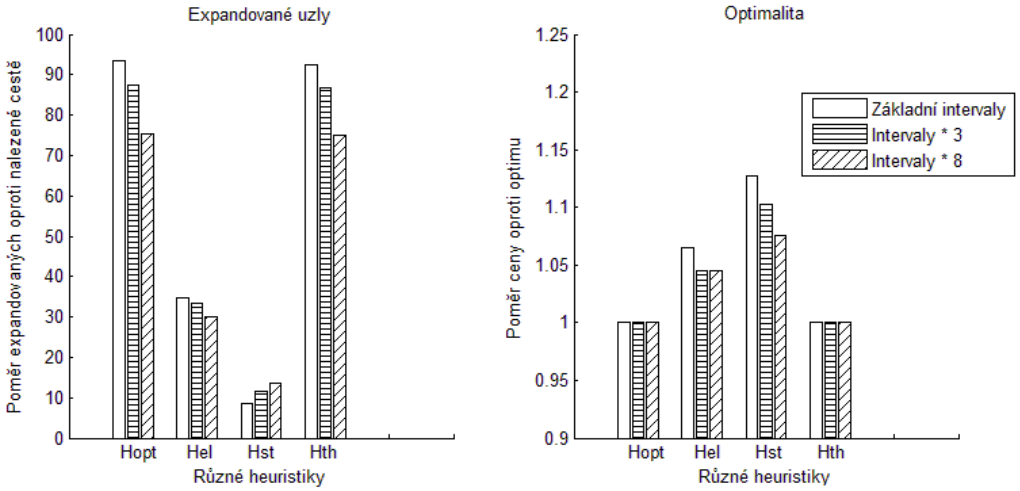
Scénář	Hodnota metriky M1
Základní intervaly	93.368
Intervaly * 3	87.416
Intervaly * 8	75.400

Tabulka 7.6: Metriky základních heuristik

Scénář	Metrika	Hel	Hst	Hth	Hw5	Hw10	Hw15
Základní	M1	34,714	8,486	92,588	12,106	10,124	10,003
	M2	0,433	0,128	0,982	0,151	0.135	0.136
	M3	1.065	1.128	1.000	1.114	1.127	1.124
	M4	0.778	0.133	1.278	0.409	0.363	0.334
* 3	M1	33.409	11.710	86.627	15.410	13.545	12.993
	M2	0.438	0.177	0.985	0.191	0.176	0.177
	M3	1.045	1.102	1.000	1.087	1.087	1.086
	M4	0.824	0.192	1.306	0.492	0.447	0.408
* 8	M1	30.001	13.434	74.915	17.177	15.534	14.958
	M2	0.468	0.226	0.986	0.237	0.220	0.222
	M3	1.045	1.076	1.000	1.070	1.069	1.074
	M4	0.885	0.225	1.317	0.561	0.500	0.486

Tabulka 7.7: Metriky jedné heuristiky s různými parametry

Scénář	Metrika	H1	H2	H3	H4	H5	H6	H7	H8
Základní	M1	4,094	3,626	7,029	5,609	4,938	5,869	5,698	17,197
	M2	0,097	0,078	0,142	0,105	0,096	0,099	0,096	0,228
	M3	1,492	1,486	1,464	1,368	1,349	1,222	1,194	1,102
	M4	0,146	0,132	0,610	0,228	0,137	0,185	0,157	0,512
* 3	M1	4,905	3,744	6,947	5,981	6,053	6,864	8,018	19,452
	M2	0,095	0,077	0,124	0,106	0,108	0,118	0,120	0,271
	M3	1,462	1,487	1,453	1,338	1,308	1,182	1,129	1,072
	M4	0,145	0,133	0,523	0,215	0,164	0,242	0,214	0,640
* 8	M1	5,031	3,817	6,235	6,556	5,908	7,843	9,257	20,172
	M2	0,111	0,091	0,118	0,130	0,128	0,152	0.163	0.333
	M3	1.491	1.469	1.541	1.360	1.247	1.163	1.113	1.070
	M4	0.161	0.150	0.387	0.224	0.173	0.314	0.283	0.751



Obrázek 7.4: Porovnání heuristik Hopt, Hel, Hst a Hth

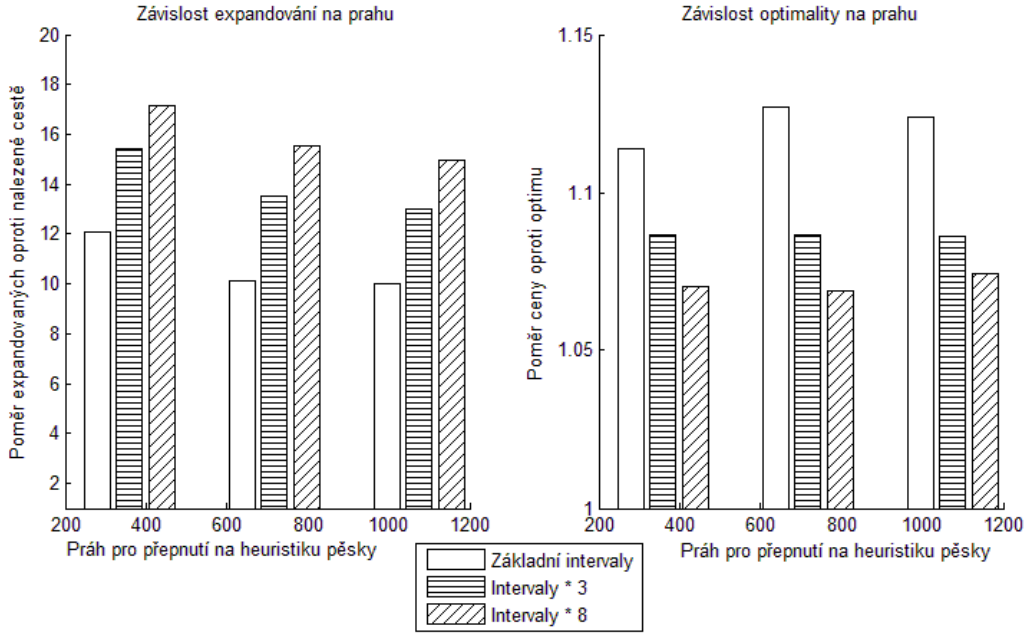
7.3.3 Základní porovnání heuristik

Nyní budu pracovat pouze s výsledky z minulé podkapitoly, budu porovnávat metriky M1 a M3 všech heuristik kromě H1 až H8. Graf obr. 7.4 ukazuje porovnání optimality a poměru expandovaných uzlů čtyř nejjednodušších heuristik Hopt, Hel, Hst a Hth. Ukazuje také závislost na scénáři.

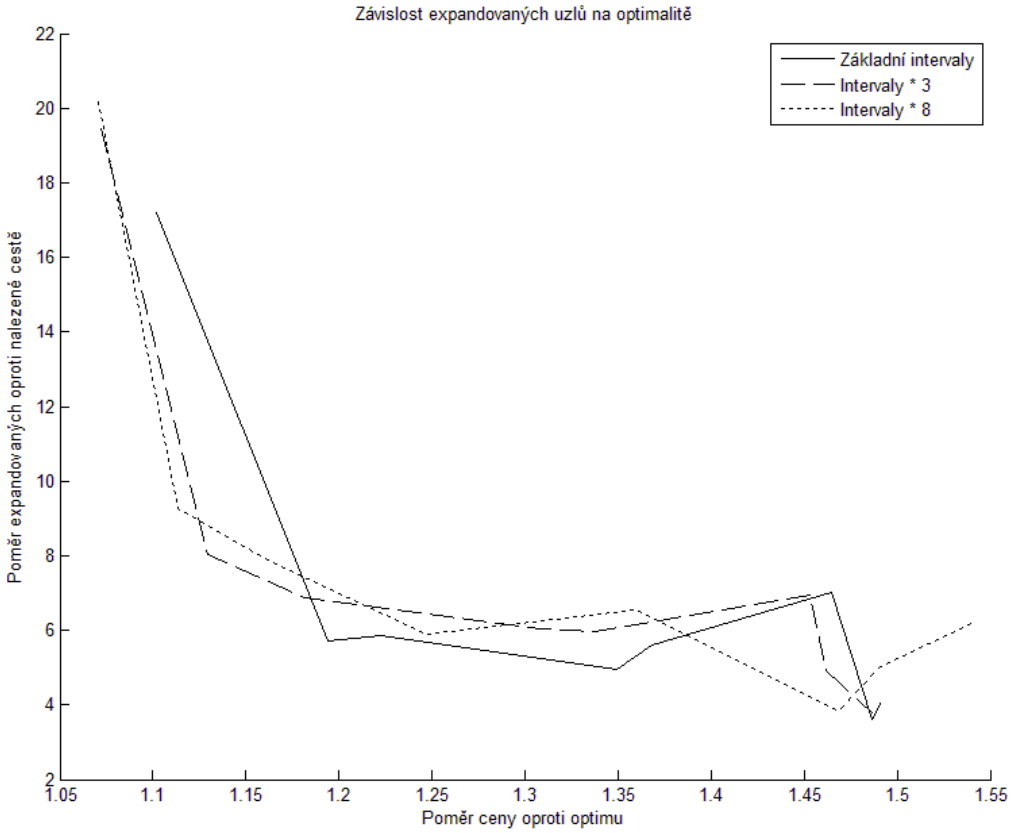
Dále se budu zabývat heuristikami se strukturou na obr. 7.3. Parametry heuristik Hw5, Hw10 a Hw15 jsem zvolil ručně, podívám se, jaký má vliv práh pro přepnutí heuristik. Graf obr. 7.5 ukazuje jednak závislost metrik M1 a M3 na prahu, tak i závislost na scénáři.

7.3.4 Závislost optimality na výpočetním času

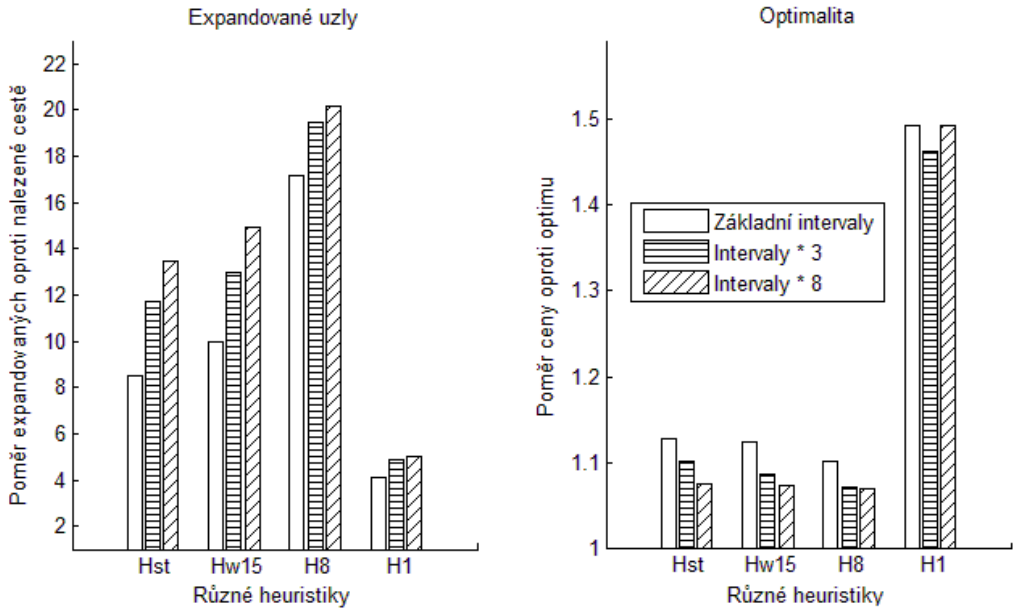
Nyní porovnám heuristiky H1 až H8, porovnám optimalitu vůči poměru expandovaných uzlů. Parametry byly nalezeny ruční změnou konstanci c při optimalizaci, ze všech experimentů už jsou vyfiltrované jen použitelné heuristiky. Některé heuristiky jsou téměř dominované, ale v některých scénářích se chovají jinak. Graf obr. 7.6 ukazuje závislost metrik M1 a M3 pro všechny scénáře. Zde se nebral ohled na závislost na scénáři, výsledky pro každý scénář jsou seřazeny zvlášť podle optimality.



Obrázek 7.5: Porovnání heuristik Hw5, Hw10 a Hw15



Obrázek 7.6: Porovnání optimality a poměru expandovaných uzlů heuristik H1 až H8



Obrázek 7.7: Porovnání čtyř nejvýznamnějších heuristik

7.3.5 Souhrnné porovnání heuristik

Nakonec porovnám nejvýznamnější heuristiky, zvolil jsem Hst, Hw15, H8 a H1. První tři jsem zvolil proto, že dávají řešení blízko optimu, H8 protože je výpočetně nejrychlejší. Porovnání je vidět na grafu obr. 7.7.

7.3.6 Diskuze

Vyzkoušel jsem několik heuristik, u některých jsem parametry odhadnul, u některých našel optimalizací. Nejdříve jsem porovnal čtyři nejjednodušší heuristiky, na grafu obr. 7.4 je vidět, že Hopt a Hth dávají téměř stejné výsledky, ve všech scénářích expandovaly nejvíce uzlů a dávají téměř optimální řešení. Hel expandovala o poznání méně uzlů, ale zase dává o trochu horší řešení. Hst expandovala nejméně uzlů, ale dává nejhorší řešení. Sledoval jsem i vliv prahu pro přepnutí na heuristiku chůze. Na grafu obr. 7.5 je vidět, že práh měl jen malý vliv na poměr expandovaných uzlů a ještě menší vliv na optimalitu, Hw15 dává ve dvou ze tří scénářů lepší řešení než Hw10.

Dále jsem porovnal osm heuristik se stejnou strukturou ale jinými parametry. Graf obr. 7.6 ukazuje závislost expandovaných uzlů na optimalitě. Pro

hodnoty blíže optimálnímu řešení poměr expandovaných uzlů prudce stoupá, pro méně optimální výsledky M1 klesá, ale průběh grafu není čistě klesající.

Nakonec jsem porovnal heuristiky, které se zdály jako nejpoužitelnější (Hst, Hw15, H8 a H1), porovnání je na grafu obr. 7.7. První tři heuristiky dávají větší váhu optimalitě, H1 dává váhu poměru expandovaných uzlů. První tři jsou téměř stejně blízko optimu, jen H8 je o trochu blíže, ale poměr expandovaných uzlů se liší výrazněji. Nakonec Hst, která pouze vynásobí hodnotu heuristiky Hopt konstantou přibližně 7 (přesně $stations * 6.962$, $streets * 7.099$), dopadla nejlépe. Ani není příliš rozdíl v násobení stanic a ostatních uzlů. Navržená struktura heuristik dopadla o něco hůře, ale zase v případě H1 dává řešení velice rychle oproti ostatním heuristikám.

Z grafů obr. 7.4, obr. 7.5 a obr. 7.7 je vidět i závislost metrik M1 a M3 na scénáři. Všechny zobrazené heuristiky dávají kvalitnější řešení pro větší intervaly spojů až na výjimku H1. Dále heuristiky Hopt, Hel a Hth expandují při větších intervalech méně uzlů, všechny ostatní heuristiky mají M1 přímo závislou na délce intervalů. Všechny heuristiky, které násobí stanice a ostatní uzly při delších intervalech expandují více uzlů, nejspíše to je způsobeno tím, že vždy násobí silnice větší konstantou než stanice, tedy algoritmus expanduje dříve do stanic. Protože je intenzita spojů nižší, expanduje převážně do stanic, které pravděpodobně stejně nepoužije, protože zde nebude vhodný spoj včas.

Kapitola 8

Závěr

Vytvořili jsme framework simulující pohyb osob a vozidel ve městě. Framework simuluje dny občanů, kdy občan jde do práce, na oběd, pak se vrátí do práce a po práci jde nakoupit, za zábavou nebo domů. Po městě buď jezdí svým autem, na kole nebo využívají linky MHD.

Jedním z přínosů této práce byl koncept aktivit agenta. Popisuje obecný přístup hierarchického modelování schopností jednoho agenta a jak mezi sebou aktivity komunikují. Tento koncept modularizuje jednotlivé části agentů tak, že mohou mít hlavní chování zcela jiné, například občan vs. MHD řidič, ale zároveň používají stejné menší části, např. řízení vozidla. Modularizace je možná i opačným směrem, tedy hlavní chování bude stejné, ale menší moduly budou jiné. Tuto možnost představuje věčný pracovník, jeho život se skládá z čekání a následného startu práce, práce může být jakákoliv, ale musí se předem vložit do paměti agenta. Stejným způsobem by bylo možné vložit do paměti různé způsoby řízení vozidla.

MHD vozidla zatím nejezdí podle jízdních řádů ale v pravidelných intervalech, agenti zatím využívají původní jednoduchý plánovač, který nebere v úvahu jízdní řády a jede maximálně jedním typem MHD. Značná část práce byla zaměřená na nový A* plánovač, který bere v úvahu jízdní řády. Plánovač byl otestován na několika scénářích a byly nalezeny nejpoužitelnější heuristiky. Vytvořeno bylo několik složitějších heuristik, u kterých jsem předpokládal, že budou více informované a zároveň najdou kvalitní řešení. Nakonec se v experimentech ukázalo, že nejlepší heuristika je přímo metrem vynásobená konstantou 7.

Plánovač zatím nebyl nasazen v běhu simulace, zde bude prostor pro další výzkum. Je možné, že v simulaci bude lepší jiná heuristika, například pokud budeme požadovat rychlejší výpočet a nebude kladen velký důraz na optimalitu nebo může způsobit problémy nepravidelnost MHD vozidel, například pokud se budou o trochu zpožďovat.

Framework zatím nebyl nijak optimalizovaný na výkon ani paměťovou náročnost a dokázal se spustit s 500.000 agenty a 100.000 agentů dokázal simulovat pořád velice efektivně. Později budeme framework optimalizovat za účelem odsimulování co nejvíce agentů co nejrychleji. Velké zrychlení by mohla přinést vhodnější prioritní fronta v event processoru, výpočetní čas plánovače by také zrychlila vhodnější prioritní fronta v open seznamu plánovačů.

A-lite vizualizace byla již částečně optimalizovaná, takže zvládla velice mnoho objektů relativně efektivně. Google Earth zatím nebyl nijak optimalizovaný a také se potýká jednak s výkonnostními problémy, tak i se závažnými chybami, kdy se ztratí rozdíl dvou snímků. První verze GE nepracovala s update soubory, ale přijímala celý seznam objektů, zde nebyly závažné chyby, ale zase bylo obnovování značně pomalejší a pro otevření popisu entity se musela zastavit simulace. Na Google Earth vizualizaci se bude dále pracovat v použitém frameworku A-lite.

Na frameworku se bude dále pokračovat, bude se používat více reálných dat, aby byla simulace co nejvěrohodnější. Model agenta bude později složitější a agent se bude moci i v průběhu simulace učit, dále bude v simulaci více typů agentů, například státní příslušníci nebo výtržníci. V tuto chvíli simulace modeluje jednoduché město s jednoduchými agenty, ale později se rozvine v mnohem složitější simulaci.

Literatura

Daniel Delling; Dorothea Wagner (2009), *Time-Dependent Route Planning*, Robust and Online Large-Scale Optimization, 207-230, Springer.

E. W. Dijkstra (1959), *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik 1, 269-271, Springer.

R. Bauer; D. Delling (2008), *SHARC: Fast and Robust Unidirectional Routing*, Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX'08), 13-26.

P. E. Hart; N. Nilsson andB. Raphael (1968), *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, Systems Science and Cybernetics, IEEE Transactions on 4, number 2, 100-107.

KRISTEN L. SANFORD BERNHARDT;(2007), *Agent-Based Modeling in Transportation*, Transportation Research Circular number E-C113 - Artificial intelligence in transportation, 72, Citeseer.

Hansen N.; Ostermeier A. (2001), *Completely derandomized self-adaptation in evolution strategies*, Comput. 9, 159-195.

Shinya KIKUCHI; Jongho RHEE andDušan TEODOROVIĆ (2002), *Applicability of an agent-based modeling concept to modeling of transportation phenomena*, Yugoslav Journal of Operations Research 12 (2002), Number 2, 141-156.

Michael Balmer; Nurhan Cetin; Kai Nagel; Bryan Raney (2004), *Towards truly agent-based traffic and mobility simulations*, IEEE Computer Society.

Dirk Helbing; Ansgar Hennecke; Vladimir Shvetsov; Martin Treiber (2002), *Micro- and Macrosimulation of Freeway Traffic*, Mathematical and Computer Modelling 35, number 5-6, 517-547, Elsevier.

Thomas Pajor(2009), *Multi-Modal Route Planning*, diplomová práce na univerzitě Karlsruhe v Německu.

OSM (2010), *Open Street Map*,
http://www.openstreetmap.org.

KML (2011), *Google Earth - KML*,
http://code.google.com/intl/cs/apis/kml/documentation/kmlreference.html.

TRANSIMS (2011), *Transportation Analysis and Simulation System*,
http://transims-opensource.net.

MATSIM (2010), *Multi Agent Transport Simulation*,
http://www.matsim.org.

SUMO (2011), *Simulation of Urban Mobility*,
http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page.

Literatura

Příloha A

Obsah přiloženého DVD

K této práci je přiloženo DVD, na kterém jsou uloženy zdrojové kódy. Adresáře v rootu jsou projekty pro Eclipse IDE.

- alite - A-lite vis a jádro simulace
- kml_visio - generování KML souborů a HTML server pro Google Earth
- LatexDip - tato práce v LaTexu
  - img - všechny obrázky použité v práci
  - matlab - skripty v MatLabu pro vykreslení grafů
  - ostatni - diagramy v Power Pointu
  - styles - stažené styly pro LaTex
- urbansim - AgentPolis

AgentPolis se spouští přes cs.agents.alite.Main s parametrem cz.agents.urbansim.creator.OsmScenarioCreator [další parametry]. Pokud se jako další parametr zadá "-h", tak se do konzole vypíše seznam všech dostupných parametrů.