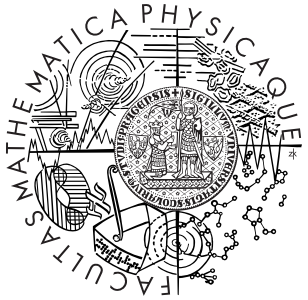
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Štefan Čudai

Multiagentní systém pro řízení dopravy

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Pavel Surynek
Studijní program: Informatika, Programování

2007

Rád by som poďakoval vedúcemu mojej bakalárskej práce RNDr. Pavlu Surynkovi za trpezlivosť, rady a za konštruktívne pripomienky pri písaní práce. Ďalej by som chcel poďakovať svojim rodičom, ktorí ma podporovali a študentke Marte Lungovej za jazykovú korektúru.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 10.8.2007 Štefan Čudai

Obsah

Název práce: Multiagentní systém pro řízení dopravy
Autor: Štefan Čudai
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: RNDr. Pavel Surynek
e-mail vedoucího: surynek@ktiml.mff.cuni.cz

Abstrakt: Predložená práca sa zaoberá multiagentným systémom pre riadenie cestnej premávky. Každý účastník cestnej premávky je racionálny agent, schopný pozorovať svoje okolie a komunikovať s ostatnými agentmi. Agent je navrhnutý, ako samostatná distribuovateľná entita. V práci boli navrhnuté dva rôzne algoritmy, ktoré agenti používajú pri plánovaní trás do cieľa. Systém je implementovaný v jazyku JAVA.

Klíčová slova: simulácia, agent, racionálny agent, distribuované systémy, multiagentné systémy

Title: Multiagent Traffic Control System
Author: Štefan Čudai
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: RNDr. Pavel Surynek
Supervisor‘ s e-mail address: surynek@ktiml.mff.cuni.cz

Abstract: The presented work is about a multiagent system for road traffic control. Every member of traffic is a rational agent able to watch his surroundings and communicate with other agents. The agent is designed as an independent distributable entity. Two different algorithms were suggested in this work for the agents to use when planning routes to the destination. The system is implemented in JAVA language.

Keywords: simulation, agent, rational agent, distributed systems, multiagent systems

Kapitola 1

Úvod

Témou tejto práce je navrhnutie multiagentného systému na riadenie cestnej premávky a overenie funkčnosti v simulácii.

V súčasnej dobe sa čoraz častejšie stávame obeťami dopravnej zápchy. Jedným z hlavných dôvodov je nedostatočná kapacita ciest, ktoré využívajú vodiči na dosiahnutie cieľa svojej cesty. Kapacita nestačí pretože väčšina vodičov volí trasu cesty tak, aby bola čo najkratšia, prípadne najrýchlejšia.

V mestách je sieť ciest a križovatiek väčšinou bohatá, môže existovať veľa rôznych trás do cieľa. Predstavme si, že by každý vodič bol včas informovaný o dopravnej situácii, potom by sa mohol rozhodnúť a vybrať si inú trasu. Zrejme by išlo o dlhšiu cestu, respektíve pomalšiu, ale tak by sa mohlo zamedziť zväčšovaniu už existujúcej dopravnej zápchy, vodič by sa mohol vyhnúť zápche a šetriť zdroje ako sú pohonné látky. Podieľal by sa teda na odstraňovaní dopravnej zápchy. Najefektívnejšie by bolo, keby každý vodič informoval všetkých o dopravnej situácii vo svojom okolí. Je samozrejmé, že tento predpoklad nemusí byť vždy splnený. Preto, aby bolo možné tento systém použiť v reálnom svete bude potrebné overiť jeho funkčnosť v simulácii a vyvodiť z pozorovaní nejaké závery.

Nie každá dopravná sieť je bohatá na počet rôznych trás. V takých prípadoch by mohol byť navrhovaný systém nefunkčný. Ide hlavne o miesta ako diaľnice alebo príjazdové cesty do miest. Riešenie pre zamedzenie dopravnej zápchy treba hľadať inde. Často jediným riešením je zväčšenie kapacít ciest stavbou cestných obchvatov a podobne. Toto riešenie je bohužiaľ finančne náročné.

1.1 Cieľ

Cieľom práce bude navrhnúť a otestovať multiagentný systém na plánovanie trasy cesty podľa kritérií zadaných od vodiča (dĺžka, rýchlosť, bezpečnosť). Systém musí byť schopný kooperovať s ostatnými agentmi. Významnou vlastnosťou systému bude schopnosť prispôsobovať sa zmenám v doprave, čo môže spôsobiť zmenu v už naplánovanej trase cesty. Teda agent sa bude vyhýbať miestam v dopravnej sieti, ktoré mu podľa zadaných kritérií nevyhovujú. Na

záver sa celý navrhnutý systém otestuje v simulácii dopravy, kde bude možné vizuálne overiť jeho funkčnosť.

V simulácii je každé auto reprezentované ako samostatný racionálny agent, ktorý nemusí kooperovať s ostatnými. Ale bude sa snažiť pohybovať po najkratšej ceste, teda bude reprezentovať tých vodičov, ktorí buď nedokážu informovať ostatných alebo nechcú poskytovať informácie o sebe. Testovať sa nebude len funkčnosť navrhnutého a implementovaného multiagentného systému, ale aj schopnosť efektívneho fungovania, pri malom počte kooperujúcich agentov. A to z dôvodu možného nasadenia do reálneho sveta, kde každý vodič bude vybavený počítačom, na ktorom bude existovať racionálny agent, prostredníctvom ktorého bude vodič posielať informácie o svojom stave a prijímať informácie o dopravných situáciách v sieti ciest. Agent tak môže poskytnúť radu vodičovi, kade má ísť, aby sa vyhol zápche. Celý systém je navrhovaný tak, aby bolo možné jednotlivé programové rozhrania použiť nielen v simulácii, ale po menšej úprave aj v praxi.

1.2 Prehľad kapitol

V druhej kapitole sa dozviete, čo je to multiagentný systém, agent, aké spôsoby komunikácie medzi agentmi existujú, aké je rozdelenie agentov. Aký typ agentov a komunikácie sme použili pri návrhu systému. Popíšeme algoritmy, ktoré používajú agenti pri rozhodovaní a pri plánovaní trás svojich ciest. A na záver kapitoly porovnáme výsledky testov rôznych algoritmov plánovania.

V tretej kapitole sa pozrieme na implementáciu tohto systému, na použité dátové štruktúry a objektový model niektorých častí systému.

Vo štvrtej kapitole sa naučíme používať priložené aplikácie, pomocou ktorých sme schopní zostrojiť mapu, na ktorej bude prebiehať simulácia a simulovať dopravu pre potreby testovania funkčnosti navrhnutých multigentných systémov.

Zhrnutie, prínos práce, pozorovania, predstavy o tom čo by sa dalo do budúcnosti zlepšiť, sú popísané v piatej kapitole.

Kapitola 2

Analýza a návrh

V úvode kapitoly si povieme niečo o multiagentných systémoch. Dozviete sa, aké vlastnosti a schopnosti majú agenti v navrhovanom multiagentnom systéme. Aké typy algoritmov používame, ktorý z nich je určený na aký účel. Na záver porovnáme niekoľko výsledkov zo simulácie dopravy.

2.1 Multiagentné systémy

Multiagentné systémy (MAS) spadajú pod Distribuovanú umelú inteligenciu (*Distributed Artificial Intelligence*, DAI) [2]. Obecne by sa dalo povedať, že DAI sa zaoberá súčinnosťou niekoľkých totožných alebo rôznorodých systémov pri riešení spoločného problému, pričom sa sústreďuje predovšetkým na problematiku reprezentácie, spracovania a využitia znalostí v týchto systémoch (*Daron, 1992*).

MAS by sa dal popísať ako množina voľne prepojených autonómnych systémov spolupracujúcich pri hľadaní spoločného cieľa. Vzájomné prepojenie systémov a možnosť komunikácie za účelom výmeny informácií pripomína tímovú prácu. Prepojené systémy dokážu riešiť úlohy paralelne, a tak skracujú dobu potrebnú pre nájdenie riešenia [1].

Pri návrhu multiagentných systémov sa do centra pozornosti dostávajú otázky kompetencie agentov, spôsob a formalizmus komunikácie medzi agentmi, metódy koordinácie a kooperácie. Agenti môžu vytvárať skupiny, začleňovať sa do nich a opúšťať ich. Kvôli tomu je potrebné do návrhu zahrnúť aj vnútornú organizáciu skupiny (komunity), zabezpečiť komunikáciu v rámci skupiny, ale aj mimo ňu. Multiagentné systémy kladú dôraz hlavne na autonómnosť agentov a ich spoluprácu pri hľadaní riešení [2].

2.1.1 Agent

Agentný systém (*Agent System*, AS) je daný prostredím, v ktorom pôsobí agent, vybavený určitou inteligenciou, používanou pri riešení zadanej úlohy. Agent je aktívny prvok, vytvorený a vložený do AS človekom. Svojou prítomnosťou agent v AS mení vlastnosti prostredia z aktuálneho stavu do cieľového [1].

Existuje niekoľko definícií agenta. Spomeňme jednu, ktorá najlepšie vystihuje taký typ agenta a jeho vlastnosti, ktorý používame v navrhovanom multiagentnom systéme.

Definícia: Agent je výpočtový proces s jedným centrom riadenia a s určitým (lokálnym) cieľom. Agenti koordinujú hlavne svoje znalosti, ciele, schopnosti a plány tak, aby ich mohli využívať spoločne pri riešení úloh. Agenti v multiagentnom systéme môžu pri svojej činnosti smerovať k jednému globálnemu cieľu alebo k jednotlivým oddeleným cieľom, ktoré spolu môžu, ale nemusia súvisieť. Agenti zdieľajú znalosti o úlohách a riešeniach (*Bond, Gasser, 1998*), [2].

Agenta možno charakterizovať podľa toho, či a na akej úrovni je schopný zvažovať rôzne varianty riešenia svojho cieľa. Podľa toho sa rozlišujú tri kategórie: *reaktívny, intencionálny a sociálny* (*Wooldridge a Jennings, 1994*), [2].

- **Reaktívny agent** reaguje na podnety z reálneho sveta. Má vopred danú množinu akcií, pričom výber akcie ako odpoveď na podnet je daný stavom okolitého prostredia. Tento typ agenta môže informovať ostatných (agentov) o svojej činnosti.
- **Intencionálny agent** je špecifický vytváraním plánu, ktorý vedie k cieľu. V skupine agentov dochádza ku koordinácii výmenou plánu.
- **Sociálny agent** pracuje s explicitnými modelmi chovania ostatných agentov. Agent musí byť schopný tieto modely upravovať a aktualizovať, pretože ich používa pri rozhodovaní a tvorbe plánu.

2.1.2 Komunikácia medzi agentmi

Už sme spomínali, že agenti v MAS, by mali byť schopní vytvárať spoločenstvá, v rámci ktorých riešia spoločne zadanú úlohu. Na to, aby sa im podarilo napredovať vpred, je potrebné, aby sa vedeli medzi sebou dorozumievať, Pomocou komunikácie by si vedeli rozdeliť úlohy alebo zdieľať medzi sebou výsledky. Preto potrebujú nejaký spoločný, vopred definovaný jazyk respektíve protokol. Spôsob, akým bude prebiehať komunikácia, výrazne ovplyvňuje vlastnosti distribuovaného systému.

Základné typy komunikácie sa líšia podľa toho, ako sú správy posielané. Podľa toho, či je možné poslať správu priamo konkrétnemu agentovi, hovoríme o *priamej komunikácii* alebo naopak, správy sa posielajú na vopred známe miesto, ktoré zodpovedá zdieľanej pamäti, vtedy hovoríme o *nepriamej komunikácii* [2].

Podľa [2] je možné metódy *priamej komunikácie* presnejšie charakterizovať podľa počtu príjemcov odosielanej správy.

Adresné posielanie správ znamená posielanie správy priamo na vopred známu adresu príjemcu. Za výhody sa pokladá bezpečnosť, pretože agent vie, komu posiela správu. Hlavným problémom tejto metódy je spôsob, akým sa dozvie agent adresu, kam má odoslať správu. V otvorených systémoch je tento

problém väčší, pretože počet agentov sa môže dynamicky meniť (agenti prichádzajú a odchádzajú).

Všesmerové vysielanie správ (*broadcasting*) je mechanizmus, ktorý sa čoraz častejšie používa v distribuovaných počítačových systémoch. Podstatou je, že sa správa neposiela konkrétne jednému príjemcovi, ale všetkým naraz. Za výhody sa považuje hlavne vlastnosť adaptivity, čo znamená, že ak nejaký agent z neznámych dôvodov opustí distribuovaný systém je možné ho nahradiť iným bez potreby poznať adresu nového agenta. Určite to môžeme pokladať za výhodné, ale aj tak nie je táto metóda veľmi používaná kvôli istým nevýhodám. Keďže má každý agent prístup ku všetkým správam, je problém so zabezpečením bezpečnosti. Okrem toho v rozsiahlych distribuovaných systémoch môže dôjsť k zahlteniu komunikačnej infraštruktúry pri posielaní veľkého množstva správ.

Selektívne vysielanie správ je kombináciou adresovej a všesmerovej komunikácie, pričom agenti sú rozdelení do skupín. Každý agent je členom aspoň jednej skupiny. Vysielaná správa sa môže dostať do jednej alebo viacerých skupín. Inak povedané, broadcasting prebieha na úrovni skupín a nie v celej komunikačnej infraštruktúre.

Pre potreby navrhovaného MAS postačí nepriamy spôsob komunikácie. Ten je založený na predpoklade existencie zdieľanej dátovej štruktúry (nazývanej *tabuľa*, [2] alebo *nástenka* , [1]). Agenti posielajú informácie, ktoré pokladajú za dôležité na *nástenku*. Ostatní agenti majú možnosť tieto informácie čítať a vyvodiť z nich vlastné pozorovanie. Nástenka sa tak stáva zdrojom informácií. Jej adresa je všetkým agentom dobre známa a v multiagentných systémoch, využívajúcich na komunikáciu nástenku, jedinou potrebnou informáciou na zabezpečenie komunikácie. Okrem tejto výhody je tu ešte ďalšia a to zníženie zaťaženia komunikačnej infraštruktúry.

2.1.3 Architektúra agenta

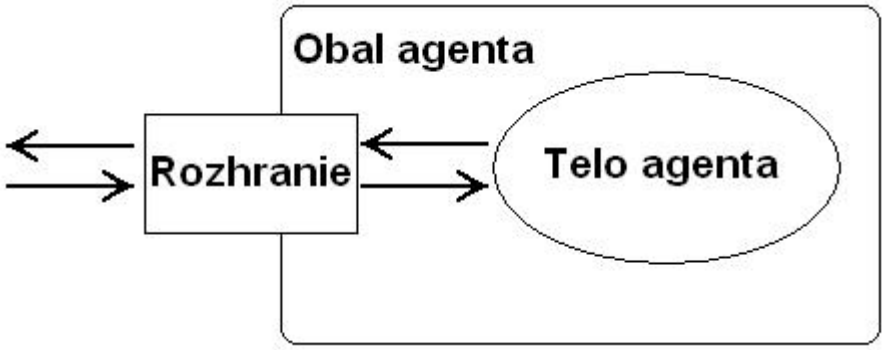
Podľa [1] je agent obvykle zložený z *obalu* a *vlastného tela*.

- **Obal** má na starosti plánovanie a realizáciu interakcií. Je zložený z *komunikačnej vrstvy* a z *modelu sociálneho chovania*.
- **Vlastné telo** nemá informácie o komunite.

Štruktúru agenta je možné vidieť na obrázku 2.1.

2.1.4 Čo sa používa v navrhovanom MAS

V predchádzajúcich kapitolách sme sa mohli len povrchne oboznámiť s problematikou multiagentných systémov, s rôznymi typmi agentov, spôsobom ko-



Obr. 2.1: Štruktúra agenta

munikácie a podobne. Otázkou ale stále zostáva, čo sme použili v našom navrhovanom multiagentnom systéme.

V navrhovanom multiagentnom systéme sú agenti špecifickí svojím racionálnym správaním. *Racionálny agent* je agent, ktorý je inteligentný, má schopnosť reagovať na podnety, ovplyvňovať prostredia (AS), má schopnosť vyvodzovať logické závery z pozorovaní okolia a má schopnosť racionálne plánovať postup svojich akcií [1]. Pozrime sa, čo to znamená v našom prípade.

- **Inteligencia** agenta je veľmi dôležitá. Keďže ide o agentov, ktorí reprezentujú vodičov dopravných prostriedkov, je nevyhnutné, aby každý agent poznal dopravné predpisy a podľa nich sa aj správal. Inak by nebolo možné ani len pomyslieť na možnosť aplikovania týchto agentov do reálnych situácii v doprave. Dopravné predpisy sú definované pravidlami, ktoré sú zapísané v *algoritme pohybu*. Tomuto algoritmu a pravidlám je venovaná celá podkapitola. Do inteligencie agentov radíme aj schopnosť rozhodnutia predbehnúť iného agenta.

- **Reakciou na podnet** pokladáme brzdenie pred križovatkou, pred autom, dávanie prednosti agenta inému, ak je na vedľajšej ceste. Podnet vyvolávajú nejaká iné objekty. Agent vie, ako sa má zachovať pri danom objekte a musí byť schopný reagovať dostatočne rýchlo a s predstihom, aby nedošlo ku kolízii. Od agenta sa teda očakáva, že bude schopný predvídať. Ako má reagovať na rôzne objekty (podnety) je popísané v podkapitole *Algoritmus pohybu*.

- **Ovplyvňovanie prostredia** je samozrejmé už len z existencie agenta v danom prostredí. Prostredím v našom MAS je dopravná sieť ciest a križovatiek. To, že agent existuje v tomto prostredí spôsobuje, že iní agenti ho vnímajú a podľa toho aj reagujú. Agent má schopnosť sa pohybovať po dopravnej sieti, čím mení svoju polohu, a teda aj stav prostredia. Podstatou je, že agent by sa mal pohybovať, tak aby nevznikla dopravná zápcha. Ak existuje v sieti iná cesta ako tá, kde je už veľa áut, tak si ju vyberie. Týmto rozhodnutím ovplyvní ďalších agentov, ktorí majú rovnaký smer cesty.

- **Vyvodzovanie záverov** je zahrnuté v inteligencii agenta. Ak získa informáciu od iných agentov, že nejaká cesta sa stáva nepriechodnou, agent si z toho dokáže vyvodiť, že tam môže vzniknúť dopravná zápcha a preto si zvolí inú cestu, ak existuje.

- **Racionálne plánovanie** v našom prípade znamená skutočné naplánovanie trasy do cieľa. Agent si vždy predtým, ako sa pohne vpred, pripraví plán cesty a podľa neho napreduje. Plán cesty je postupnosť križovatiek, cez ktoré má prejsť. Plánovanie cesty využíva hlavne schopnosť agenta vyvodzovať závery. Teda z informácií, ktoré získal od agentov si odvodí, ktoré cesty sú nebezpečné a ktoré vhodné. O tejto problematike sa viac dozviete v podkapitole *Algoritmus plánovania*.

Na komunikáciu medzi agentmi sa používa *nástenka*. Ide o nepriamu komunikáciu. Na nástenke si agenti nechávajú informácie (odkazy) o tom, kde sa nachádzajú a ako rýchlo sa pohybujú. Každý agent má prístup na túto nástenku. Môžeme povedať, že vytvárajú jedno spoločenstvo, v rámci ktorého sa všetky znalosti zdieľajú na nástenke. Druhú skupinu tvoria agenti, ktorí nekomunikujú s nikým a nemajú ani prístup na nástenku. Ide o agentov čiastočne racionálnych, pretože nedokážu vyvodzovať žiadne závery, keďže nemajú z čoho. Takýto agenti sú v reálnom nasadení úplne zbytoční. Využívame ich iba pre potreby testovania v simulácii. Je jasné, že pri reálnom nasadení nie je možné, aby bol každý vodič vybavený počítačom, na ktorom by existoval náš racionálny agent. V simulácii agenti druhej skupiny reprezentujú vodičov bez počítača, pričom sa sleduje efektivita vyhýbania sa dopravným zápcham agentov z prvej skupiny.

2.2 Návrh algoritmov

Na to, aby sme mohli skúmať navrhovaný systém v simulácii, potrebujeme nejakým spôsobom definovať správanie agentov na cestách. Keďže snahou celého projektu je navrhnúť, taký systém, ktorý by sa dal použiť aj v praxi, musíme agentov naučiť dodržiavať dopravné predpisy. Použitím agentov, ktorí poznajú a dodržiavajú dopravné predpisy, by proces simulácie napodobňoval skutočné situácie. Tým by sa výsledky simulácia stali hodnovernými a na základe nich by sme mohli vyvodiť rôzne závery. Výsledkom simulácie je venovaná celá podkapitola v kapitole *Záver*.

Pri navrhovaní systému som sa nechal inšpirovať, pozorovaniami z reálneho sveta. Jedným z nich je, že prácu vodiča za volantom môžeme rozdeliť na dve fázy. Rozhodovanie kade pôjde a fáza pohyb po trase, ktorú si vybral. Pri voľbe trasy sa berie väčšinou do úvahy aktuálny stav v okolí vodiča, prípadne ak má k dispozícii informácie o dopravných situáciách z iných miest dopravnej siete. Samotné vykonanie pohybu po danej trase, je už o niečo komplikovanejšie, pretože povinnosťou vodiča je dodržiavať dopravné predpisy. Do fázy vykonania pohybu radíme napríklad dodržiavanie dostatočného odstupu od auta pred nami, prednosť v jazdne autám na križovatke, ale aj rozhodovanie

či predbehnúť auto.

Snahou teraz bude previesť toto pozorovanie do návrhu MAS. Inteligencia agenta je daná algoritmom, ktorý používa. Algoritmus agenta je vlastne mozog agenta. Podľa neho vie, ako sa má správať v určitých situáciách. Každý algoritmus je navrhnutý podľa vzoru vodiča z reálneho sveta, ktorý sme popísali. Najprv si trasu naplánuje a potom sa po nej pohybuje. Úlohou agenta v simulácii teda bude určiť trasu, po ktorej pôjde a samotný pohyb po tejto trase, pričom bude dodržiavať dopravné predpisy.

Najdôležitejšou časťou algoritmu agenta, bude plánovanie trasy. Presne táto časť, by sa mala použiť aj v reálnych podmienkach. Predstavme si, že niektorí vodiči áut, by boli vybavený počítačom, na ktorom by bol agent, ktorý by nejakým spôsobom vedel poradiť vodičovi, akou cestou ísť, aby sa vyhol čo najviac dopravnej zápche. Ako by agent našiel najlepšiu trasu pre vodiča si povieme v podkapitole *Algoritmus plánovania*.

Návrh algoritmu agenta bol prispôsobený myšlienke, že sa v praxi použije iba tá časť algoritmu, ktorá rozhoduje o výbere trasy. Koly tomu je potrebné rozlíšiť plánovanie trasy a samotný pohyb. Algoritmus agenta je teda zložený z dvoch iných a to *Algoritmus plánovania* a *Algoritmus pohybu*. Toto rozdelenie prináša veľa výhod. Je možné použiť rôzne implementácie oboch algoritmov zároveň a tak vytvoriť úplne nové. Je daný teda priestor iným implementáciám a okrem toho to sprehľadňuje celý návrh. Môžeme teda navrhnúť iný *Algoritmus plánovania* a vďaka už existujúcemu *Algoritmus pohybu* otestovať jeho funkčnosť na simulácii.

Vieme už, ako bude vyzerať mozog agenta. Povedzme si ešte, čo bude potrebovať, aby mohol plnohodnotne fungovať.

V podstate môžeme povedať, že agenta nezaujíma mapa dopravnej siete, po ktorej sa pohybuje, teda jej grafická reprezentácia. Dôležitý pre neho je graf tejto mapy. Navrhovaný agent potrebuje dva typy grafov, jeden sa používa v algoritme pre pohyb, ten nie je vhodný na vyhľadávanie ciest. Druhý sa používa v algoritme plánovania, ktorý je stavaný na to, aby našiel najkratšiu cestu. Dokonca bol navrhnutý, tak aby podľa kritérií, ktoré zadá agent prispôsobil hodnoty váh na hranách. Zatiaľ bolo implementované iba jedno kritérium a to, aby pri vyberaní ciest sa vybrali také, po ktorých sa nepôjde pomalšie ako zadaná rýchlosť. Samozrejme, graf sám o sebe nenájde najkratšiu cestu, to má na starosti algoritmus *Dijkstra*, ktorý pri spustení hľadania cesty dostane ako parameter kritéria, ktoré určil agent. Ako sa generujú grafy sa dočítate v kapitole *Implementácia*.

Hovorili sme, že agenta nezaujíma mapa, ale pre optimalizovanie rýchlosti simulácie bolo potrebné, aby agent na základe svojich súradníc a informáciách o tvare vytvoril svoj obraz na mape. A na to potrebuje mapu dopravnej siete, na ktorej prebieha simulácia.

Vieme, už ako bude agent rozhodovať o svojom pohybe a vieme čo na to potrebuje. Ostáva nám vysvetliť, ako vyzerá pohyb agenta v simulácii. Agent sa pohybuje po dopravnej sieti ciest iba vtedy, keď ho o to požiada server simulácie. Čo je to a ako funguje server simulácie, sa dočítate v kapitol *Imple-*

mentácia. Pre túto chvíľu nám stačí vedieť, že server rozhoduje o tom, kedy sa má agent pohnúť. Agentovi vždy oznámi čas, ktorý má využiť na pohyb. Mohli by sme povedať, že ide o diskrétnu simuláciu. Agent môže pridelený čas využiť podľa svojej potreby. Samozrejme nemôže z ničoho nič zastaviť, ak sa pohybuje nejakou rýchlosťou. Keď chce zastaviť, začne brzdiť iba tak veľmi, ako najviac vie. Teda musí rešpektovať fyzikálne vlastnosti auta, ktoré ovláda.

Pri počítaní novej pozície musí rešpektovať stav okolia. Ak je v jeho okolí auto, musí svoj pohyb podľa toho korigovať. Preto musí poznať polohu agentov, tieto polohy sú vždy posledné spočítané pozície. Inak povedané, ak pri počítaní novej pozície potrebuje agent získať informácie o okolí, teda polohu iných agentov, tak získaná poloha bola spočítaná ešte v minulom cykle simulátora. Informácie o polohe iných agentov napríklad pred križovatkou alebo na ceste, na ktorej práve náš agent je, poskytuje server simulácie.

Výpočty pozícií sa vykonávajú na grafe danej mapy. Hodnoty výpočtu sú uložené vo *value* objekte *PoziciaAuta*, ktorý je posielaný serveru ako výsledok, okrem toho, ako sme už spomenuli, agent pošle aj svoj obraz na mape, ktorý získa z objektu *PoziciaAuta* a z objektu, ktorý reprezentuje mapu. Výpočet novej pozície má na starosti *algoritmus pohybu* o ktorom si povieme viacej neskôr.

Na obrázku 2.2, môžeme vidieť objektový model navrhnutého algoritmu agenta. Skladá sa z troch interfacov, ktoré v podstate rozdeľujú algoritmus agenta, na algoritmus pohybu a algoritmus plánovania. Trieda *AbstractBaseAlgoritmus* reprezentuje algoritmus pohybu. Dedí z triedy *BasePohybAlgoritmus*, ktorá implementuje metódu *PoziciaAuta pohybVpred()*. Práve ona spočítava novú polohu agenta. Ak by sa mala implementovať nová trieda, ktorá by definovala správanie agenta pri pohybe, musela by implementovať rozhranie *PohybAlgoritmusInterface*.

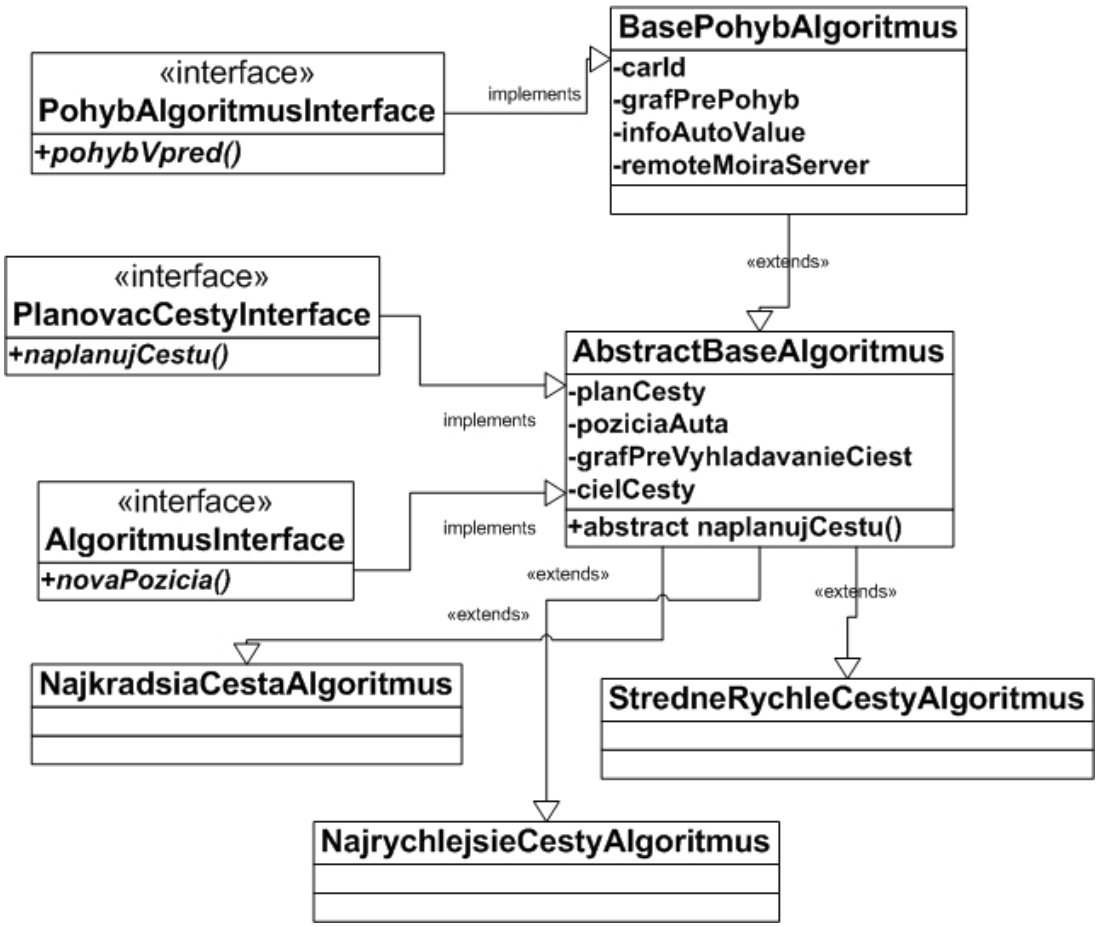
Každý nový algoritmus plánovania, by mal dediť z triedy *AbstracBaseAlgoritmus*, ktorá obsahuje abstraktnú metódu *PlanCesty naplanujCestu()*, ktorú treba implementovať. Napríklad triedy *NajkradšiaCestaAlgoritmus*, *StredneRychleCestyAlgoritmus* a *NajrychlejsieCestyAlgoritmus*, sú rôzne algoritmy plánovania, pretože definujú danú metódu. O týchto algoritmoch si povieme viac v ďalších kapitolách.

2.2.1 Algoritmus pohybu

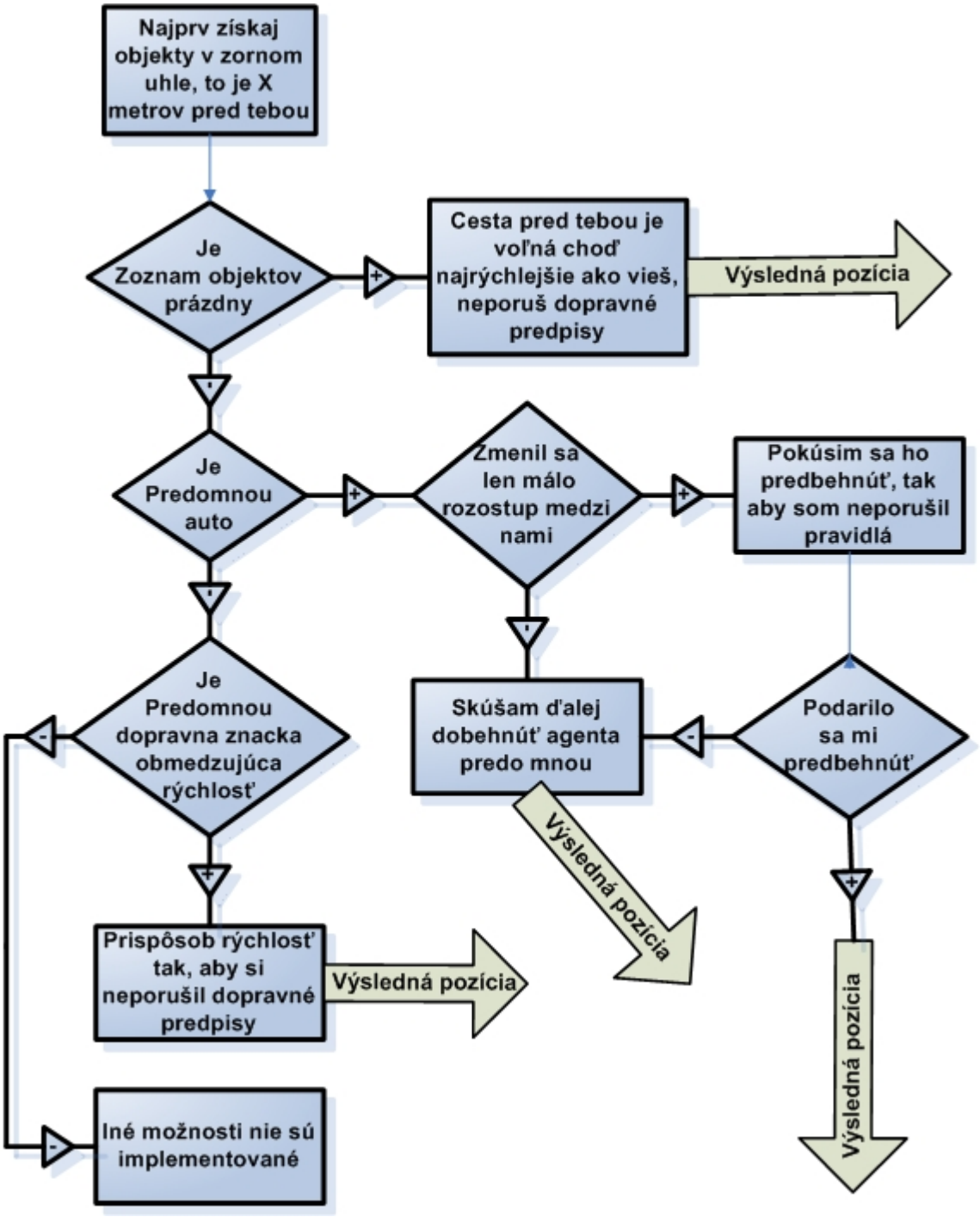
V tejto podkapitole si povieme, ako reaguje agent na rôzne stavy okolia. Ako sa rozhoduje, či dať prednosť vjazde alebo predbehnúť iné auto a podobne. Dozvieme sa aj to, aké pravidlá musí dodržiavať pri pohybe, aby neporušil dopravné prepisy.

Na obrázku 2.3, je vidieť podrobne algoritmus, ktorý používa agent pri počítaní novej polohy.

Agent sa po celú dobu pohybu snaží ísť čo najrýchlejšie, ako vie a zároveň, tak aby neprekročil maximálnu povolenú rýchlosť. Ďalej, ak existuje viacej jazdných pruhov v rovnakom smere, tak agent ide v tom, z ktorého sa dá ďalej pokračovať do križovatky podľa plánu. Agent môže predbiehať iného iba



Obr. 2.2: Objektový model algoritmu agenta



Obr. 2.3: Algoritmus, ktorý sa používa pri pohybe agenta v pred

s pravej strany a zároveň musí platiť, že aj zo susedného pruhu, do ktorého by sa preradil, sa dá pokračovať podľa plánu. Teda tiež vedie ďalej do križovatky, ktorá nasleduje v pláne. Ďalej, ak existuje viacej jazdných pruhov, ktoré by pokračovali do križovatky v pláne, tak sa snaží ísť v tom, ktorý je najviac vpravo. Agent je schopný dať prednosť v jazde inému na križovatke, ak je to nutné. Robí sa to tak, že X metrov pred križovatkou už agent nemôže zmeniť plán cesty, teda musí pokračovať ďalej. Naznačí smer, teda prechod cez križovatku, ktorý použije. Inak povedané vyhodí smerovku. Agent potom pri križovatke zistí či existuje auto, ktorému treba dať prednosť, ak áno tak začne spomaľovať tak, aby to stihol ubrzdiť na hranicu križovatky. Ak už nie je nikto, komu treba dať prednosť, tak pokračuje ďalej v ceste podľa plánu.

Agent pri prechode križovatkou, preskočí na jazdný pruh, ktorý definuje prechod na križovatke. Teda, pohyb cez križovatku nie je plynulý ale ide o priamy skok.

Pri pohybe vpred, si agent najprv zistí, či je nejaký objekt pred ním. Objektom sa rozumie, buď iný agent, dopravná značka alebo križovatka. Ak pred ním nič nie je, tak môže pokračovať ďalej maximálne rýchlo ako vie. Ak je pred ním auto, prispôsobí rýchlosť jazdy tak, aby sa dostal čo najbližšie za neho. Ak dlhšiu dobu udržiava rovnaký rozostup, tak sa ho pokúsi predbehnúť, ak to ide.

Ak je pred agentom dopravná značka obmedzujúca rýchlosť, tak prispôsobí rýchlosť, tak aby neporušil predpisy. Posledný objekt, ktorý može byť v zornom uhle agenta je križovatka. Ak je v dohľade križovatka, začne meniť svoju rýchlosť, tak aby pri prechode ňou, mal rýchlosť definovanú konštantou. Okrem toho kontroluje či sa dostal už na takú vzdialenosť pred križovatkou, že musí dať prednosť inému agentovi, ak je to nutné. Komu má dať prednosť zistí tak, že z mapy predností, ktorú má vygenerovanú každá križovatka, nájde smery prechodov cez križovatku, z ktorých treba dať prednosť. Podľa toho vie, na ktoré cesty sa má pozrieť či tam nie je dáky agent.

V prvom návrhu si agent zo serveru simulácie vypýtal zoznam všetkých áut v okolí križovatky a vyberal z nich tie, ktorým trebalo dať prednosť. Komu treba dať prednosť je definované mapou predností. V súčasnom systéme sa koly optimalizácii agent spýta serveru, či má dať niekomu prednosť a server simulácie mu to zistí sám. Dôvod prečo sme prešli na druhú alternatívu, bol koly veľkému množstvu prenášaných dát medzi serverom a agentmi, čo výrazne narušilo plynulosť simulácie.

Vo chvíli, keď agent dorazí do svojho cieľa, je zo serveru simulácie odregistrovaný. Teda jeho existencia v simulácii končí.

2.2.2 Algoritmus plánovania

Naplánovaná cesta sa ukladá do value objektu *PlanCesty*. Plán cesty sa uchováva v podobe zoznamu križovatiek, po ktorých treba prejsť, aby sme sa dostali do cieľa. Tento zoznam obsahuje na začiatku ID vrcholu, z ktorého sa začala cesta vyhľadávať a na konci ID vrcholu, kam sa chce agent dostať. Algoritmus pre plánovanie, teda musí vracať postupnosť vrcholov, po ktorých treba ísť.

Objekt *PlanCesty* obsahuje aj niekoľko pomocných metód. Tak napríklad metódu, ktorá vráti ďalšiu križovatku v poradí, cez ktorú má agent prejsť.

Plánovanie cesty prebieha vždy pred tým, ako sa agent pohne. U algoritmu *NajkradšiaCestaAlgoritmus* to tak nie je. Agent len raz spraví plán cesty a to hneď, ako dostane graf pre vyhľadávanie najkratšej cesty. Vytvorený plán si uloží a používa sa po celú dobu existencie agenta na mape. Tento typ plánovania reprezentuje tých vodičov z reálneho sveta, ktorí sa chcú dostať do svojho cieľa za každú cenu najkratšou cestou a nemienia zmeniť trasu. Pretože, buď neexistuje iná alebo sa obávajú, že by sa aj na inej trase mohli dostať do zápchy. Algoritmus najkratšej cesty bol navrhnutí, pre potreby simulácie, aby bolo možné overiť efektívnosť algoritmov iného typu plánovania.

Algoritmus *StredneRychlaCesta* a *NajrychlejsieCesty* funguje presne naopak, ako predošlí. Pri každom dotaze na agenta sa najprv naplánuje cesta a až potom sa pohne vpred podľa vytýčenej trasy. Podstatou algoritmov je, že všetci agenti, ktorí používajú jeden z nich, medzi sebou komunikujú prostredníctvom nástenky, na ktorej si nechávajú oznamy. Oznamy hovoria o tom, akou rýchlosťou sa daný agent práve pohybuje po danej ceste (hrane). Agent, ktorý práva plánuje svoju cestu, si najprv pozrie oznamy na nástenke. Tým sa rozumie, že získa informácie o rýchlosti všetkých agentov, ktorí poslali oznam. Z rýchlostí agentov, ktorí sú na rovnakej ceste sa spraví aritmetický priemer, čím dostaneme rýchlosť, ktorá prislúcha danej ceste, teda hrane. Je to jedna z hodnôt každej hrany. Hodnoty hrany sa potom použijú na výpočet váhy.

Ak existuje cesta, na ktorej nebol agent, ktorý by poslal oznam o rýchlosti ktorou sa po nej pohybuje, tak sa vezme vážený priemer maximálnych povolených rýchlostí na danej ceste. Napríklad ak máme cestu, po ktorej môžme ísť 80 metrov rýchlosťou 40 *km/h* a 200 metroch 100 *km/h* tak výsledná rýchlosť bude určená ako *(80*40+200*100)/280*.

Hranám, v grafe pre hľadanie najkratšej cesty, je teda možné nastaviť hodnoty, ako je vážený priemer maximálnych rýchlostí, priemerná rýchlosť agentov na danej hrane a dĺžka hrany. Z uvedených rýchlostí sa vyberie vždy tá najmenšia. Keďže poznáme dĺžku hrany a máme k dispozícii minimálnu rýchlosť zo všetkých uvedených, tak vieme spočítať čas, aký potrebujeme na prejdenie danej hrany, ak by sme išli najmenšou z uvedených rýchlostí. Táto hodnota potom určuje váhu hrany.

Celé je to navrhnuté tak, že agentovi vo chvíli, keď má spočítať nový plán cesty, stačí nastaviť priemerné rýchlosti na hranách získané z nástenky a spustiť algoritmus pre hľadanie najkratšej cesty. Ten totiž získava informáciu o váhe hrany pomocou metódy, ktorá sama zabezpečí jej spočítanie.

Na začiatku tejto kapitoly sme spomenuli, že je možné používať isté kritérium výberu hrán (ciest). Tým kritériom je rýchlosť, akou by sa chcel agent pohybovať. Tento spôsob zadávania rýchlostí sa používa aj u GPS zariadení, ktorým keď zadáme danú rýchlosť, tak nám nájdu podľa toho najkratšiu cestu. Napríklad chodec, by zadal 5 *km/h*. V takom prípade nemá veľký význam ísť po ceste, na ktorej je povolená maximálna rýchlosť 80 *km/h* a jej dĺžka je jeden kilometer, keď existuje cesta dĺžky 300 metrov na pešej zóne.

Práve spomínané kritérium zadanej rýchlosti, nám bude hľadať cesty tak,

aby boli najrýchlejšie vzhľadom k zadanému kritériu. Docielime to tým že, pri počítaní váhy danej hrany vezmeme vždy tú najmenšiu rýchlosť z týchto troch: rýchlosť z kritéria, priemerná rýchlosť z nástenky, vážený priemer maximálnych povolených rýchlostí na hrane. Keď sa na chvíľu zamyslíme, zistíme, že ak sa vytvorila niekde zápcha a boli tam agenti, ktorý poslali oznam na nástenku, tak ich rýchlosť bude veľmi malá, určite menšia ako maximálna povolená. Ak agent požaduje, aby sa plánovala cesta, tak aby bola pomerne rýchla, tak najmenšia rýchlosť zo všetkých na danej hrane bude práva tá z nástenky. Pri hľadaní najkratšej cesty, je veľmi pravdepodobné, že sa táto hrana nepoužije v pláne. A to vďaka tomu že agenti kooperujú pri pohybe v dopravne sieti. Vzniká tak dojem že sa informujú o tom, kde je a kde práva vzniká dopravná zápcha.

Pre optimalizovanie rýchlosti počítania nových pozícií agentov, si nástenka vždy, ako dostane nový oznam, spočíta priemernú rýchlosť na danej hrane. Potom, ak agent príde a bude chcieť vedieť, aká je priemerná rýchlosť agentov na konkrétnej hrane nemusí nič počítať a rovno dostane túto informáciu. Okrem toho je potrebné zaručiť, že každý agent bude mať na nástenke maximálne jeden oznam. Väčší počet nemá zmysel, naopak vznikali by problémy. Toto si tiež zabezpečí nástenka. Čo nedokáže nástenka je, zistiť či agent ešte existuje alebo nie. Preto vždy, keď sa odhlasuje zo simulácie, musí o tom informovať nástenku, ktorá zruší jeho posledný oznam.

Algoritmus *StredneRychlaCesta* má nastavnené kritérium rýchlosti na 60 *km/h* a *NajrychlejsieCesty* na 120 *km/h*. Teda algoritmus *NajrychlejsieCesty* sa bude snažiť nájsť iba také cesty, ktoré sú časovo rýchle, vzhľadom dopravným informáciám. Pričom druhý algoritmus, bude preferovať aj pomalšie úseky.

Na záver menšie upozornenie. Vo chvíli, keď sa agent pozerá na nástenku, nie je zaručené, že priemerné rýchlosti sú aktuálne, ale je isté, že nie sú staršie ako jeden výpočet dozadu.

Kapitola 3

Implementácia

V kapitole sa dočítate o objektových modeloch významných častí aplikácie. O štruktúrach, ktoré reprezentujú prvky mapy a to cesta, križovatka, dopravná značka a body zdroj/stok. Povieme si, aké typy grafov sú potrebné pre beh simulácie a ako sa generujú z vytvorenej mapy. Nezabudneme ani na popis životného cyklu agenta a princípu na akom funguje simulácia.

3.1 Knižnica lib_gui

Pre potreby bakalárskej práce, vznikla jednoduchá knižnica, na vytváranie grafického rozhrania pre užívateľa. Pomocou knižnice má programátor možnosť oddeliť grafickú časť, od logiky programu. Vďaka tomu, sa stáva projekt prehľadný. Nebudeme presne popisovať ako funguje, ale iba ukážeme, ako sa používa.

Na vytvorenie nového okna potrebujeme dve triedy a jeden interface. Jedna z tried je určená na definovanie grafického vzhľadu okna. Teda aké má rozmery, aké komponenty obsahuje a iné. Druhá slúži pre definovanie logiky daného okna. To znamená, čo sa má stať, keď užívateľ klikne na tlačítko. Čo sa stane, keď sa otvorí okno, zatvorí, v akom režime sa má otvoriť a podobne. Interface obsahuje iba statické konštanty, ktoré definujú názov jednotlivých komponent, ktoré užívateľ použije pre definovanie logiky. Interface teda slúži na premapovanie komponent z GUI do logiky.

Vytvorenie GUI
Pri vytváraní nového okna, by sme mali dodržiavať nasledovnú konvenciu. Trieda pre GUI (ďalej už len panel), by mala mať názov v tvare *NazovTriedy**Panel***. Musí dediť z *DefaultFrame* alebo *DefaultDialog*. Frame je klasické okno a Dialog je dialógové okno. Panel ďalej musí implementovať interface, ktorý by mal byť pomenovaný ako *NazovTriedy**Map***.

Po dokončení okna, je nutné *registrovať* komponenty, ktoré chceme použiť v logike, teda v druhej triede (ďalej už len controller). Odporúča sa umiestniť registrovanie do konštruktora panelu. Registruje sa pomocou metód *registrujNieco(nazov, komponenta)*, kde *Nieco* je typ komponenty. Napríklad Label,

TextComponent, AbstractButton, a iné.

Controller, by mal mať názov podľa vzoru *NazovTriedy**Controller***. Musí dediť z *DefaultFrameController* alebo *DefaultDialogController*, podľa toho, akého predka má panel, ktorý prislúcha k tomuto controlleru. Controller podobne, ako panel, implementuje interface. Aby sme jednoducho určili, ktorý controller, panel a mapa spolu tvoria okno, je vhodné ich pomenovať rovnako. Všetky controllery, ktoré tvoria GUI, by mali byť umiestnené v jednom balíčku. V našom projekte sa nachádzajú v balíčkoch s názvom *vva_controller*. Podobne všetky panely sú v balíčku *vva_panel* a mapy *vva_map*.

Podobne ako na panely, tak aj na controllery, je potrebné najprv zaregistrovať komponenty, než ich začneme používať. Registruje sa pomocou metód *registrujNieco(názov)*, ktoré vrátia komponentu typu *Nieco*, ktorá je registrovaná na panely, pod rovnakým menom. V tej chvíli máme k dispozícii danú komponentu a môžeme jej pridať nejaký listener alebo zmeniť je vlastnosti.

Udalosti na controlleri

Okrem komponent, ktoré získa controller ich registrovaním, má k dispozícii možnosť definovať, čo sa má stať, pri istých udalostiach na okne. Slúžia na to metódy

- *onCreateComplete()* – tesne pred zobrazením okna volá predok controllera túto metódu ako prvú
- *onNactiData()* – ak je režim okna nastavený na *detail* alebo *oprava*, tak tesne pred zobrazením okna, sa zavolá z predka
- *onNactiNovy()* - ak je režim okna nastavený na *nový*, tak tesne pred zobrazením okna, sa zavolá z predka
- *onRezimNastaven()* – v tejto metóde by sa mali vykonávať rôzne nastavenia okna, podľa toho, v akom režime bolo otvorené, volá sa tesne pred zobrazením
- *validateAndReport()* – po kliknutí na *OK* sa okno pokúsi volať metódu *onUlozData()*, ale ešte predtým, volá túto metódu, do ktorej by mali byť umiestnené validačné kontroly, ktoré zabránia zavrieť okno a teda aj uložiť nekorektné dáta
- *onUlozData()* – sa zavolá ak metóda *validateAndReport()* vráti **true**, po ukončení metódy sa okno zavrie
- *onClose()* – sa volá vždy po kliknutí na krížik alebo na tlačítko *Close*

V tejto chvíli už vieme, ako vytvoriť nové okno. Potrebujeme ešte vedieť, ako ho naštartujeme. Najprv vytvoríme triedu, ktorá bude dediť z *FrameFactory* a bude umiestená na takom mieste, aby inštancia tejto triedy, vznikla skôr, akoby sme sa pokúsili otvoriť nové okno. Najlepšie bude, keď ju pomenujeme *PanelFactory*. Jej konštruktor, by mal mať podobný tvar ako:

```
public PanelFactory() {
      super();
      add(MainMap.THIS_NAME, MainPanel.class);
      add(RoadDialogMap.THIS_NAME, RoadDialogPanel.class);
      add(JunctionDialogMap.THIS_NAME, JunctionDialogPanel.class);
}
```

Metóda *add(názovPanelu, triedaPanelu)* pridáva panely do zoznamu všetkých panelov, ktoré daná aplikácia používa. Ak budeme chcieť otvoriť okno, ktorého panel nebol pridaný do zoznamu, aplikácia spadne.

Spustenie nového okna sa robí podobne, ako je uvedené v príklade:

```
JunctionDefineDialogController contr =
      new JunctionDefineDialogController(krizovatka, cesty);
contr.setRezimOprava();
contr.startController();
```

Prípadne môžeme nastaviť režim na *setRezimNovy()* alebo *setRezimDetail()*.

3.2 Mapa

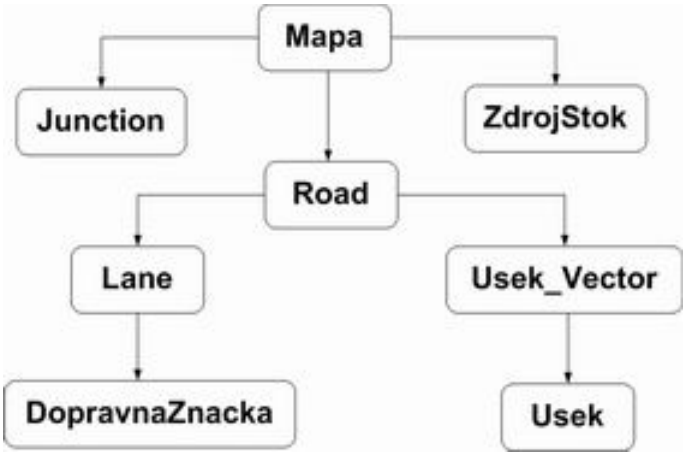
Aby sme mohli testovať a simulovať navrhnuté algoritmy, je potrebné mať k dispozícii mapu. Mapa sa skladá z ciest, križovatiek, z bodov zdroj/stok a dopravných značiek. Na obrázku 3.1 môžeme vidieť, aké objekty mapa, ako dátová štruktúra, obsahuje. Smer šípky naznačuje, že objekt X, obsahuje objekt Y.

Objekt mapy, sa používa iba pre potreby vizualizácie simulácie, aby mal užívateľ predstavu o tom, kde a ako sa agent pohybuje. Nepoužíva sa pri počítaní novej polohy agenta a ani pri rozhodovaní, ktorou trasou sa vidať. Na to slúžia špeciálne grafy, ktoré sa s mapy vygenerujú. Jediný výpočet, ktorý sa vykonáva na mape počas simulácie je zistenie tvaru agenta. Teda jeho natočenie a poloha na mape. Pozrime sa ďalej na jednotlivé objekty, z ktorých je mapa zostavená.

3.2.1 Cesta

Cesta má jeden začiatok a koniec. Spája práve dva objekty, kde jeden z nich môže byť križovatka alebo bod zdroj/stok.

Každá cesta je zložená s aspoň jedného úseku. Úsek je súvislá rovná časť cesty. Teda cestu s jednou zákrutou reprezentujeme, ako dva rôzne úseky. Každý úsek cesty, si nesie informáciu o svojej dĺžke, súradnici odkiaľ sa kreslí, smer akým sa má kresliť a pravý normálový vektor. *Úsek* je objekt, ktorý je uložený do iného objektu *Usek_Vektor*, ktorý pribaľuje ďalšie informácie, potrebné pre vykreslenie cesty. Sú to smerové vektory pre ľavú stranu úseku a pravú stranu. Inak povedané objekt *Usek*, je iba čiara, ktorá ma nejaký tvar a umiestnenie. Pričom *Usek_Vektor* nesie informácie koľko takýchto čiar treba



Obr. 3.1: Objektový model mapy

nakresliť zľava a sprava, teda tvary jazdných pruhov.

Okrem toho, potrebujeme ešte určiť vlastnosti jazdných pruhov. Na to slúži trieda *Lane*, v ktorej sa uchováva šírka a dopravné značky na danom jazdnom pruhu. Taktiež je možné pamätať si, odkiaľ, kam smeruje jazdný pruh. To sa používa pri generovaní grafov.

Objekt *Road*, ako celok okrem spomenutých tried, obsahuje aj informácie o tom, odkiaľ, kam vedie cesta a pomocné dáta, ktoré urýchľujú vykreslovanie.

Spomenieme ešte spôsob, ako sa určuje ľavá a pravá strana cesty. Pri kreslení cesty, sa určí počiatočný bod prvého úseku. Kreslením určujeme smerové vektory jednotlivých úsekov. Pravá strana cesty je tá, ktorá obsahuje pravý normálový vektor, na smer kreslenia cesty. Ľavá je prirodzene na opačnej strane. Potom atribúty *from, to* sú korektné pre pravé jazdné pruhy. Pre ľavé jazdné pruhy je potrebné zobrať ID objektu *to* ako *from* a *from* ako *to*.

3.2.2 Križovatka

Grafická reprezentácia križovatky, je jednoduchšia, ako cesty. Križovatka je množina bodov (vrcholov), ktoré ju tvoria. Keďže ide o polygón, sú X-ové súradnice bodov a Y-ové súradnice uložené v rozdielnich zoznamov. Tie potom pri vykreslovaní stačí priamo vložiť, ako parametre, do metódy knižnice Swing [4], na kreslenie polygónov.

Z pohľadu logiky križovatky, ide o miesto, kde sa stretávajú dve a viac ciest. Je povolené vytvárať iba križovatky, ktoré sú tvorené maximálne štyrmi cestami. Definujeme si jazdný pruh typu *from* ako pruh cesty, ktorý vchádza do križovatky a jazdný pruh typu *to* ako pruh, ktorý vychádza z križovatky.

Každá križovatka obsahuje mapu prechodov cez ňu. Prechod je usporiadaná dvojica *(from,[to, to, . . .])*, kde *from* znamená, z ktorého jazdného pruhu začína prechod a *[to, to, . . .]*, do ktorých jazdných pruhov vedie. Okrem tejto mapy, obsahuje aj opačnú, a to mapa dvojíc *(to,[from, from, . . .])*. Tá sa používa na rýchle vyhľadanie všetkých jazdných pruhov, z ktorých sa môžeme dostať do daného jazdného pruhu.

Ďalšia štruktúra, ktorá sa nachádza v tomto objekte, je mapa predností. Ide o mapu, ktorá obsahuje pod každým kľúčom množinu dvojíc *(from, to)*, ktoré určujú, z ktorých smerov je nutné dať prednosť. Smer je jednoznačne určený dvojicou *(from, to)*, kde *from* znamená, na ktorom jazdnom pruhu sa agent práve nachádza a *to*, do ktorého sa chce dostať. Mapa predností sa vytvorí pomocou triedy *PrednostVJazdeSimply* a jej metódy *getPrednosti(mapa_fromTo, mapa_toFrom)*. Na to, ale potrebujeme ešte poznať, ktorá cesta je hlavná a takzvané poradie ciest. Informácia o hlavnej ceste je uložená v premennej *mainRoad*. Ide o dvojicu ciest, ktoré určujú hlavnú cestu. Dôležité je ešte vedieť, že ak máme križovatku zloženú zo štyroch ciest, je možné určiť hlavnú cestu iba cestami, ktoré sú oproti sebe.

Informácia o poradí ciest sa získa od užívateľa, ktorý v smere hodinových ručičiek postupne vyberie cesty. Tak vieme určiť, ktorá cesta je vľavo alebo vpravo. Túto informáciu potrebujeme, aby sme mohli definovať pravidlo pravej ruky, pri dávaní prednosti v jazde.

Vodič dáva vždy prednosť druhému vodičovi na križovatke po pravej ruke, ak sa práve nenachádza na hlavnej ceste alebo odbočuje z hlavnej na vedľajšiu cestu.

Generovanie predností vjazde

Trieda *PrednostVJazdeSimply* implementuje rozhranie *PrednostVJazdeInterface*. Rozhranie deklaruje metódu, ktorá spočítanie, z ktorých smerov treba dať prednosť. Parametre metódy sú mapa prechodov *fromTo* a mapa prechodov *toFrom*. Vďaka rozhraniu bol daný priestor na ďalšie, iné implementácie, generovania predností vjazde.

Algoritmus generovania využíva sadu pravidiel, podľa ktorých, je možné určiť všetky smery, z ktorých treba dať prednosť. Generovanie obecne funguje iba pre križovatky, ktoré sú tvorené maximálne štyrmi cestami a hlavná cesta je definovaná dvoma cestami, ktoré sú oproti (nesmú byť susedia). Križovatka môže byť zložená aj z troch ciest. V takom prípade je jedno, ktorá dvojica ciest bude hlavná cesta.

Pre beh výpočtu, je ďalej nutné, mať definované poradie ciest v smere hodinových ručičiek. Slúži to na určenie, ktoré dve cesty sa krížia.

Príklad: Ak máme križovatku zo štyroch ciest, ID ciest je 2,4,5,6 a ich poradie určíme ako *[4,5,6,2]*, tak pri zisťovaní, ktorá cesta kríži, ktorú postupujeme nasledovne. Nech *From* je ID cesty odkiaľ ideme, *To* je ID cesty kam ideme. Potom

index cesty ***From*** *z poradia ciest - index cesty* ***To*** *z poradia ciest =* ***x****.*

Ak ***x***:

- *je väčšie ako 0*, potom pri prechode križovatkou, sa cesta *From* do *To* nekríži z iným prechodom
- *je menšie ako 0*, potom pri prechode križovatkou, sa cesta *From* do *To* kríži z iným prechodom

- *je rovné 2*, potom cesty *From* a *To* sú oproti sebe, teda krížia všetky prechody ostatných ciest
- *je väčšie ako 2*, potom pri prechode križovatkou, sa cesta *From* do *To* kríži z iným prechodom
- *je menšie ako 2*, potom pri prechode križovatkou, sa cesta *From* do *To* nekríži z iným prechodom

Napríklad: From=5, *To*=6, potom *2-3=-1*, čo je menšie ako 0, teda cesta s ID *From* kríži nejakú inú cestu. Akú, to ešte nevieme. To sa určí pomocou ďalších pravidiel.

Obecne rozlišujeme dva prípady prechodov križovatkou:

- *1. prípad:* auto je na hlavnej ceste a po prechode križovatkou pokračuje na ceste, ktorá je hlavná *(H → H)* alebo je na hlavnej ceste a po prechode križovatkou sa dostane na vedľajšiu *(H → V)*
- *2. prípad:* auto je na vedľajšej ceste a po prechode križovatkou sa dostane na vedľajšiu (V → *V*) alebo je na vedľajšej a po prechode križovatkou sa dostane na hlavnú (V → *H*)

Použitím informácie o tom, či by pri prechode križovatkou auto krížilo nejaký iný definovaný prechod, určením či ide o 1. alebo 2. prípad prechodu a použitím nasledujúcich pravidiel sme schopný spočítať všetky smery, z ktorých musí auto dávať prednosť.

- *(H → H)* alebo *(H → V)*:
  - ak kríži, potom
    - hľadáme také definované prechody, kde *From* je na hlavnej ceste a *To* je miesto, kam ide auto (pre ktoré zisťujeme odkiaľ dať prednosť)
    - všetky prechody, kde*From* je na hlavnej ceste a *To* smeruje do cesty, z ktorej vychádza naše auto (pre ktoré zisťujeme odkiaľ dať prednosť)
  - ak nekríži, potom auto nemusí dávať nikomu prednosť
- *(V → V)* alebo *(V → H)*:
  - ak kríži, potom
    - všetky prechody, ktoré idú z hlavnej cesty na hlavnú cestu
    - všetky prechody, kde *From* je na hlavnej ceste a *To* je cesta, z ktorej vychádza naše auto (pre ktoré zisťujeme odkiaľ dať prednosť) a zároveň táto hlavná cesta kríži, keď ide do cesty odkiaľ vychádza auto
    - hľadáme také definované prechody, kde *From* je na hlavnej ceste a *To* smeruje do cesty, kam ide auto (pre ktoré zisťujeme odkiaľ dať prednosť)

    - všetky prechody, kde *From* je na vedľajšej ceste a *To* je na ceste, kde ide naše auto a zároveň vedľajšia cesta sa nekríži, keď ide tam kam auto
  - ak nekríži, potom *From* je na hlavnej ceste a *To* smeruje do cesty, z ktorej vychádza naše auto (pre ktoré zisťujeme odkiaľ dať prednosť)

3.2.3 Body zdroj/stok

Bod zdroj/stok je miesto, z ktorého cesta začína, *zdroj*, alebo v ktorom končí, *stok*. Objekt *ZdrojStok* je teda bod, ktorý je umiestnený, buď na jednom alebo na druhom konci cesty. Tento objekt môže byť typu, buď zdroj, stok alebo zároveň aj zdroj aj stok. Bod typu zdroj je miesto, z ktorého môžu štartovať agenti. Bod typu stok je miesto, kam sa môžu dostať. Teda, ak sa chce agent dostať do dopravnej siete ciest (mapy), musí odštartovať zo zdroja, iná pozícia pre jeho štart nie je prípustná.

Podľa toho, o aký typ bodu ide, sa nastavuje lambda číslo. Toto číslo sa používa pri náhodnom generovaní agentov. Skôr, ako sa agent začne pohybovať po mape, musí vedieť, kde štartuje a kde je jeho cieľ. Štart a cieľ konkrétneho agenta sa náhodne vygenerujú pomocou lambda čísel. Vďaka týmto číslam vieme nastaviť, aby sa z niektorých zdrojov generovali agenti častejšie ako z iných a podobne, aby niektoré stoky boli vyberané, ako cieľ častejšie, ako iné.

Predtým ako sa pustíme do podrobného vysvetlenia náhodného generovania agentov, ešte popíšeme, aká je grafická reprezentácia bodu zdroj/stok. Bod zdroj/stok, je na mape znázornený, ako úsečka. Teda, je tvorený dvomi bodmi, ktoré sú uložené v objekte *ZdrojStok*. Jeden bod by mal byť nakreslený mimo cestu a druhý na ceste. Aplikácia pre editovanie máp, to ale nijak nekontroluje, je to na užívateľovi ako znázorní na mape bod zdroj/stok.

Náhodné generovanie agentov

Ako sme už spomenuli, každý bod daného typu má takzvané lambda číslo, ktoré sa používa pri náhodnom generovaní agentov. Ďalej vieme, že skôr, ako sa začne agent pohybovať po mape, musí poznať svoj štart a cieľ. Štartovný bod zdroj/stok generuje server simulácie. Cieľový bod dogeneruje *Agent manager*, v ktorom sa vytvárajú náhodný agenti.

Server simulácie si najprv vygeneruje, pre každý zdroj, náhodné časy z intervalu, ktorý mu zadal užívateľ. Časy sa generujú podľa exponenciálneho rozdelenia. Lambda číslo sa použije v nasledovnej funkcii, ktorá vráti čas, ktorý sa postupne pripočítava k počiatočnému. Čím menšie lambda číslo, tým väčšie číslo vráti. Inak povedané, čím väčšie lambda číslo bude nastavené na zdroji, tým častejšie budú agenti zo zdroj štartovať.

```
// vracia casy v milisekundach
private double randomExp(double lambda) {
  double x = Math.random();
  while (x == 0) {
    x = Math.random();
  }
  return Math.log(x)/-lambda;
}
```

Napríklad, nech bol zadaný časový intervalom *(I,J)*. Funkcia vracia časy *(x_1, x_2, ...)*, potom dostaneme zoznam nasledovných časov, ktoré hovoria, kedy má agent odštartovať *[I+x_1, I+x_1+x_2, ...,K]*, kde K je menšie, ako J a posledný krok bol, že sa vygeneroval čas, ktorý by po pripočítaní ku K prekročil J.

Každý zdroj má teda vlastný zoznam časov, kedy má z neho vyštartovať nejaký agent. Tieto zoznamy časov, sú zotriedené do jedného, rastúco. Pričom sa vie, že v danom čase, majú z ktorých zdrojov vyštartovať agenti. Server simulácie, má tak jeden zoznam, ktorý kontroluje v každom cykle (hlavného cyklu simulácie), či nie je čas odštartovať nových agentov. Ak zistí, že je treba vytvoriť nového agenta, pošle dotaz na Agent managera, ktorý je prihlásený k serveru. Teda skôr, ako je sa spustí generovanie náhodných časov, musí existovať Agent manager, ktorý je prihlísený k serveru simulácie.

Agent sa vytvorí, až vo chvíli, keď príde jeho čas na štart. Vytváranie má na starosti Agent manager, ktorý prostredníctvom metódy *addRandomCars()*, ktorú má sprístupnenú pre vzdialené volanie, dogeneruje ostatné náhodné parametre agenta. Metódu zavolá server simulácie, ktorý ako jediný pozná aktuálny čas v simulácii a teda vie, kedy treba vytvoriť nového agenta.

Metódy *addRandomCars()* náhodne vygeneruje stok, do ktorého sa má agent dostať. Stok sa určí podľa normálneho rozdelenia a to nasledovne. Vezmú sa všteky stoky a ich lambda čísla sa naukladajú vedľa seba. Vygeneruje sa náhodné číslo medzi 0 a 1. Toto číslo sa vynásobí súčtom lambda čísel všetkých stokov a podľa to ho, do ktorého intervalu spadne, ten stok sa vyberie ako cieľ.

Napríklad, nech sú na mape tri stoky s lambda číslami *(1,1,1)*. Naukladaním vedľa seba rozumieme, vytvorenie nasledovného zoznamu *[1,2,3]*. Nech sa náhodne vygeneruje číslo *0,5*, potom *0,5*3=1,5*. Toto číslo *1,5* patrí do intervalu medzi číslami 1 a 2. Teda, ako cieľ by sa vybral stok, ktorý je na druhej pozícii.

Okrem zdroja a stoku sa náhodne vygeneruje agentovi algoritmus podľa, ktorého sa bude pohybovať po mape. Ten sa generuje rovnakým spôsobom ako stok. Jedným z parametrov metódy *addRandomCars()* je aj zoznam algoritmov, ktoré sa majú použiť, pri generovaní agentov. Pričom každý algoritmus má zadanú pravdepodobnosť použitia, čo je to isté, ako lambda číslo u stoku. Ďalej sa pripočíta náhodne číslo, obmedzenej veľkosti, k maximálnej rýchlosti akou dokáže auto ísť a taktiež aj k maximálnemu zrýchleniu a spomaleniu.

3.2.4 Dopravná značka

Do mapy je možné umiestniť, na každý jazdný pruh, dopravnú značku. Na jeden jazdný pruhu je povolené pridať neobmedzene veľa dopravných značiek. V *Editore máp* sú implementované iba značky rýchlostného obmedzenia.

Každá dopravná značka implementuje rozhranie *DopravnaZnackaInterface*, ktoré deklaruje metódu *getPozicia()*, ktorá vráti objekt, ktorý popisuje pozíciu značky. Na presné určenie pozície značky na mape potrebujeme vedieť, na ktorej ceste sa nachádza, v ktorom jazdnom pruhu danej cesty a pozíciu od počiatku cesty. Tieto údaje sa používajú iba pri výpočtoch v simulácii. Pre grafické zobrazenie značky na mape, si objekt dopravnej značky musí pamätať presnú súradnicu, kde sa má na mape zobraziť.

Zoznam všetkých dostupných dopravných značiek sa nachádza v triede *DopravneZnackyDostupne*, ktorá je vytvorená podľa návrhového vzoru *singleton*. Po získaní inštancie tejto triedy, môžeme pomocou metódy *getDopravneZnacky()*, dostať všetky dopravné značky, ktoré sa používajú. Preto sa musia všetky nové implementované dopravné značky umiestniť do konštruktora triedy, aby boli sprístupnené na použitie v aplikácii. Okrem toho musí každá nová dopravná značka dediť z abstraktnej triedy *DopravnaZnacka*, ktorá obsahuje údaje, ktoré by mala mať každá značka.

3.3 Generovanie grafov

Graf je veľmi dôležitý prvok celého systémy. Všetky výpočty, ktoré sa vykonávajú v simulácii, sú robené na grafoch. Hľadanie najkratšej cesty medzi dvoma vrcholmi alebo samotný pohyb po cestách. Výsledky, ktoré sú z výpočtov získané sú potom pretransformované, tak aby zodpovedali skutočnej polohe agenta na mape. V skratke povedané, grafická časť simulácie a výpočtová časť sú od seba oddelené.

Graf je zložený z hrán a vrcholov. Každý jazdný pruh je reprezentovaný ako jedna hrana. Každá križovatka a bod zdroj/stok sú reprezentované ako vrchol, pričom sa rozlišuje typ vrcholu *zdroj, stok, zdroj_stok, križovatka*.

Generovanie grafov prebieha na serveri simulácie, vo chvíli potvrdenia výberu mapy, na ktorej bude prebiehať simulácia. V navrhnutých algoritmoch sa používajú dva typy grafov, oba orientované. Prvý typ sa používa pre pohyb po mape, pri počítaní novej pozície agenta. Tento typ grafu, nie je vhodné používať na hľadanie najkratšej cesty. Nájdená cesta by nemusela skutočne existovať, pretože tento graf neberie ohľad na definované prechody cez križovatku.

Druhý typ grafu sa používa na hľadanie najkratších ciest, medzi dvoma zadanými vrcholmi. Tento typ grafu je vygenerovaný tak, aby cesty, ktoré v ňom skutočne sú, existovali aj na mape. Teda pri generovaní sa berie ohľad na zadefinované prechody cez križovatku. Na nájdenie najkratšej cesty sa používa algoritmus *Dijkstra*, ktorý je implementovaný v triede *Dijkstra*.

Na generovanie oboch grafov sa používa trieda *GenerovanieGrafu*, ktorá

obsahuje dve metódy *generujGraf()* a *generujGrafPreVyhladavanie()*. Prvá generuje graf prvého typu, druhá graf pre vyhľadávanie najkratších ciest. Obe metódy dostávajú do parametru mapu dopravnej siete, z ktorej sa má vygenerovať graf.

Graf pre pohyb po mape

Pripomeňme že graf je orientovaný. Generuje sa nasledovne. Najprv sa vytvoria všetky vrcholy z križovatiek a z bodov zdroj/stok. Potom sa vytvoria hrany, zo všetkých jazdných pruhov ciest na mape. Po vytvorení hrán a vrcholov, sa ešte vytvorí niekoľko veľmi dôležitých pomocných štruktúr. Každá je vhodná na niečo iné. Napríklad ak potrebujeme zistiť, či existuje hrana medzi dvoma vrcholmi alebo potrebujem získať hrany medzi konkrétnymi dvoma vrcholmi, tak tieto informácie nájdeme v premennej *hranyMedziVrcholmi* v objekte *Graf*. Je to mapa, kde kľúčom je *(ID vrcholu z, ID vrcholu do)* a hodnota pod týmto kľúčom je zoznam hrán, ktoré skutočne existujú medzi vrcholmi.

Ďalšou dôležitou štruktúrou je *susediaVrcholu*, ktorá je opäť mapou. Udržiava v sebe zoznam vrcholov, do ktorých je možné dostať sa, z nejakého konkrétneho vrcholu. Ide v podstate o maticu susednosti. Poslednou štruktúrou je mapa *vrcholySuseda*. Pod kľúčom nájdeme zoznam vrcholov, ktoré sú susedmi, z ktorý je možné sa dostať do vrcholu, ktorý je kľučom v mape.

Graf pre vyhľadávanie najkratších ciest

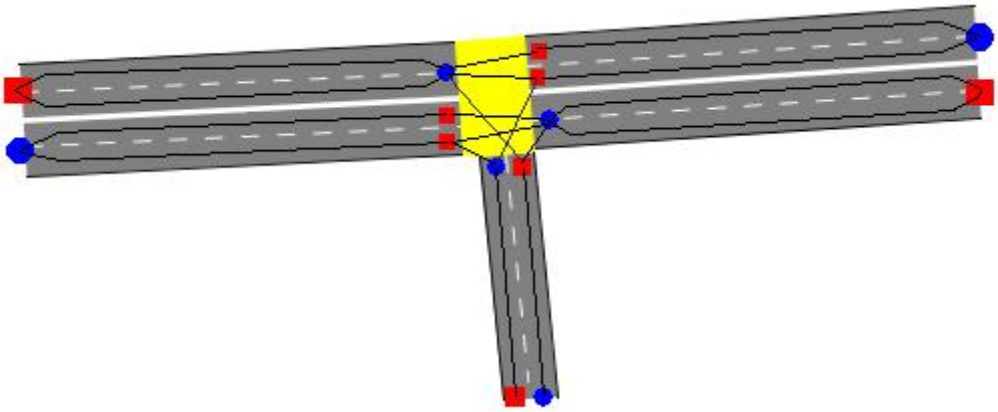
Keďže sa používa na vyhľadávanie ciest, musí sa vytvoriť tak, aby cesty v grafe zodpovedali skutočným parametrom mapy. Križovatku si musíme predstaviť, ako ďalší graf, obsahujúci vrcholy, ktoré sú navzájom poprepájané hranami, ktorých dĺžka je 0. Niektoré vrcholy sú *vstupné*, iné *výstupné*. *Vstupné* sú tie, ktoré sa napoja na jazdný pruh smerujúci do križovatky, *výstupné* sú tie, ktoré sa napoja na jazdný pruh smerujúci z križovatky. Ako sa majú vrcholy medzi sebou prepojiť je dané definíciou prechodov cez križovatku.

V predošlom grafe sme z jednej križovatky, vytvorili jeden vrchol. V tomto grafe dostaneme z jednej križovatky niekoľko vrcholov. Počet *výstupných* vrcholov je presne daný počtom ciest, ktoré tvoria križovatku. Ale počet *vstupných* vrcholov je daný počtom jazdných pruhov, ktoré vchádzajú do križovatky.

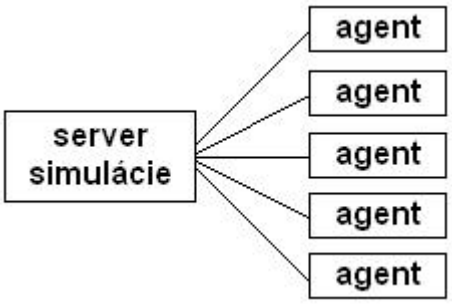
Na obrázku 3.2, je vidieť, ako by taký graf pre danú mapu mohol vyzerať. Modrý kruh naznačuje, že ide o výstupný vrchol z križovatky a červený štvorec, vstupný vrchol do križovatky. Hrany na cestách sú orientované z modrého kruhu do červeného štvorca, pričom hrany v križovatke naopak.

Podobne, ako križovatka, aj bod zdroj/stok sa rozdelí na niekoľko vrcholov. Ak ide o bod, ktorý je typu zdroj a stok zároveň, vytvoria sa dva vrcholy, jeden vstupný a druhý výstupný. Teda, pre každý typ bodu zdroj/stok sa vytvorí jeden vrchol.

Vo chvíli, keď máme pripravené vrcholy, dogenerujú sa hrany z ciest. A nakoniec, ako pri grafe prvého typu, aj tu sa vytvoria rovnaké pomocné štruktúry na urýchlenie pri vyhľadávaní hrán a vrcholov.



Obr. 3.2: Graf pre vyhľadávanie cesty



Obr. 3.3: Server simulácie a prihlásení agenti

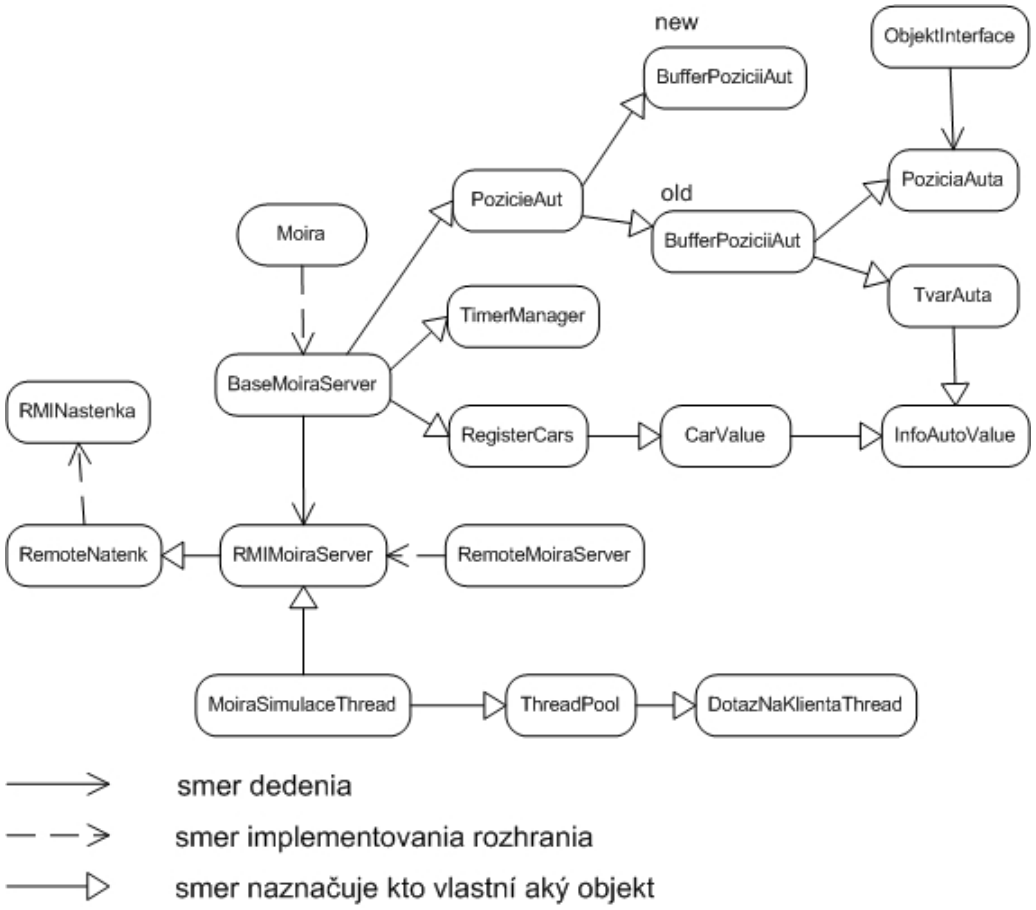
3.4 Simulácia

V tejto chvíli už vieme ako je reprezentovaná mapa, vieme ako z nej zostrojiť graf. Pozrime sa teda na to, ako sa používa v simulácii a ako vlastne prebieha simulácia.

3.4.1 Server simulácie

Keďže cieľom práce je implementovať navrhnutý systém ako distribuovanú simuláciu, vznikol za týmto účelom takzvaný *server simulácie*. Server má za úlohu riadiť celú simuláciu. Rozhodovať o tom, ktorý agent sa má pohybovať, kedy sa má vytvoriť nový náhodný agent a podobne. Server je akýmsi virtuálnym svetom, v ktorom žijú agenti, ktorý sa v tomto fiktívnom svete pohybujú. Aby agent mohol žiť v tomto svete, musí najprv požiadať server. Ten mu pošle potrebné údaje o svete a prihlási ho.

Cieľom práce je taktiež, aby bol každý agent samostatnou distribuovateľnou entitou. Teda agenti sa prihlasujú na server, na ktorom práve prebieha simulácia, viď obrázok 3.3. Server dáva pokyn agentom, aby sa pohli ďalej



Obr. 3.4: Objektový model serveru simulácie

a následne na to zobrazuje nové polohy agentov.

Vďaka tomuto návrhu je možné server a agentov umiestniť na ľubovoľné počítače, ktoré sú sieťovo prepojené. Predpokladá sa, že prepojené počítače dokážu medzi sebou komunikovať vďaka protokolu TCP/IP. Potom komunikáciu medzi serverom a agentmi je zabezpečená systémom RMI [5], ktorý jazyk JAVA podporuje. Ide o vzdialené volanie metód ***R**emote **M**ethod **I**nvocation*, z jedného virtuálneho stroja, na druhom.

Na obrázku 3.4 je vidieť objektový model serveru simulácie. Asi najvýznamnejšia je trieda *MoiraSimulaceThread*, ktorá je spustiteľná ako samotné vlákno. Implementuje hlavný cyklus simulácie, o ktorom si povieme neskôr. Trieda obsahuje objekty *ThreadPool* a *RMIMoiraServer*. *ThreadPool* je vlákno, ktoré má na starosti správu iných vlákien. Keďže sa predpokladá, že v simulácii bude existovať naraz niekoľko stoviek agentov a server vytvára pre každého agenta samotné vlákno, potrebujeme správcu, ktorý by riadil vytváranie a beh vlákien. Pretože tvorba a spustenie nového vlákna môže trvať relatívne dlho, nehovoriac o tvorbe veľkého množstva vlákien naraz, [3]. Trieda *ThreadPool* vlastne združuje vlákna. Ide o to, že keď nejaké vlákno dokončí pracú a v danom okamžiku pre neho nie je ďalšia práca, nechá si ho správca po nejakú dobu v zálohe. Ak sa do určitej doby nenájde pre vlákno žiadna práca a pritom počet

spustených vlákien vzrastie na hodnotu väčšiu ako požadované minimum, nechá správca vlákno zaniknúť. Okrem toho je definované maximálne množstvo vlákien, ktoré môžu bežať naraz. Pri vytváraní tejto triedy sa dajú nastaviť nasledovné parametre: minimálny počet vlákien ktoré musia byť vždy k dispozícii, maximálnu počet vlákien ktoré môžu pracovať v jednom okamžiku a čas, ktorý čaká vlákno ak nemá nič na práci.

Objekt *RMIMoiraServer* je samotný server simulácie. Implementuje rozhranie *RemoteMoiraServer*, ktoré sprístupňuje metódy, ktoré môže použiť agent pri vzdialenom volaní metód. Ide o metódy, ktoré vrátia graf mapy, na ktorej sa práve simuluje alebo sa pomocou jednej z nich môže odhlásiť zo simulácie. Trieda *RemoteMoiraServer* dedí od *BaseMoiraServer*, ktorá je akousi nadstavbou. Ak by niekto iný, chcel implementovať vlastný server, mal by dediť z tejto triedy. Ide totiž o základ serveru simulácie. Obsahuje objekt na prácu s časom *TimerManager*, objekt v ktorom sú uložené všetky registrované autá v simulácii *RegisterCars* a pomerne významný objekt *PozicieAut*, do ktorého sa ukladajú nové vypočítané polohy agentov.

Trieda *RemoteMoiraServer* obsahuje inštanciu triedy *RMINastenka*, ktorá implementuje rozhranie *RemoteNastenka*. Tento dôležitý objekt zabezpečuje komunikáciu medzi agentmi, ktorý majú schopnosť komunikovať s okolím. Rozhranie sprístupňuje metódy pomocou, ktorých agenti pridávajú nové oznamy na nástenku a mažú ich.

Vráťme sa späť k registru áut. Táto trieda si pamätá zoznam všetkých zaregistrovaných agentov na serveri. Informácie o agentoch si uchováva v objekte *CarValue*, v ktorom nájdeme informácie o aute agenta, vzdialenú referenciu na agenta, čas kedy chce odštartovať, kde začína, kam sa chce dostať a či už odštartoval. Vzdialená referencia je potrebná koly RMI.

Informácie o aute agenta, sú uložené v objekte *InfoAutoValue*. Štruktúra obsahuje údaje o rozmeroch auta, farbe auta, maximálnej rýchlosti, ktorú dosiahne, maximálne zrýchlenie a spomalenie ktoré dokáže vynaložiť.

Spomenuli sme už nejaký čas na odštartovanie agenta. Obecne platí, že keď je agent prihlásení k serveru, neznamená to, že aj existuje na mape. Agent začne existovať na mape, až ho server odštartuje, teda ak príde jeho čas, ktorý si agent určil sám. Tento proces je trochu komplikovaný. Predstavme si, že máme dvoch agentov, ktorý chcú v jednom okamžiku odštartovať z toho istého miesta. Keďže navzájom o sebe nevedia, dostali by sa do kolízie. Jedine server vie v danej chvíli, že z jedného zdroja štartujú naraz dvaja agenti. Tento problém sa rieši zamykaním zdrojov. Postup je nasledovný:

Predpokladajme, že agent ešte neodštartoval. Potom server zistí, ktoré jazdné pruhy sú voľné. Spýta sa agenta či chce použiť niektorý z týchto pruhov. Ak agentovi nevyhovuje ani jeden, tak musí počkať do ďalšieho kola. Ak si agent vybral nejaký jazdný pruh, do ktorého by chcel odštartovať, server pre neho zamkne tento pruh, čím zaistí, že už nikto iný naň nevstúpi. Vo chvíli, keď agent odštartuje, je mu nastavený príznak *wasStarted* na **true**. Server má taktiež na starosti odomknutie daného jazdného pruhu, ak sa agent, ktorý tu štartoval, dostal do bezpečnej vzdialenosti od počiatku.

Trieda *PozicieAut* sa používa na uchovávanie informácii o nových spočíta-

ných pozíciách agentov. Obsahuje dve triedy *BufferPoziciiAut*, v ktorých sú uchované, jak pozície, tak aj spočítané tvary áut, ktoré sa potom použijú na vykreslenie v mape. Jeden buffer sa používa vždy pre ukladanie nových hodnôt, pričom druhý sa používa pri výpočtoch nových pozícii. Pri počítaní novej pozície agenta, potrebuje poznať staré (posledné) pozície agentov. Je to koly tomu, aby agent vedel, či má dať niekomu prednosť v jazde alebo či má predbiehať iné auto a podobne.

Okrem novej pozícia auta, agent vráti aj takzvaný tvar auta v objekte *TvarAuta*, ktorý sa pri prekresľovaní mapy vezme a použije sa na vykreslenie auta daného agenta. Vypočítať správnu polohu auta a natočenie na mape je pomerne zložité, keby to mal robiť server, veľmi by ho to spomalilo. Preto sa počítanie tvaru auta prenieslo na agenta, ktorý nie je tak veľmi vyťažený. Tento objekt musí mať každý agent vytvorený, pretože nevieme nijak inak určiť, či sa má vykresliť alebo nie. Keďže výrez obrazovky je menší ako mapa, je treba orezať. Nebudeme predsa vykresľovať autá , ktoré nie je vidieť.

Trieda *PoziciaAuta* obsahuje informáciu o ID čísle agenta, ktoré mu pridalil server po prihlásení, pozíciu prednej a zadnej časti auta. Pozíciu hmotného bodu, ID cesty a ID jazdného pruhu (je rovnaké ako ID hrany v grafe), na ktorom sa nachádza. Obsahuje aj aktuálnu rýchlosť agenta a zvolený smer cesty. Smer cesty je usporiadaná dvojica ID čísel jazdných pruhov, kde prvé hovorí, na ktorom pruhu je teraz a druhé, na ktorý jazdný pruh skočí po prechode križovatkou. Táto dvojica sa dá chápať ako smerovka, ktorú auto vždy vyhodí keď prechádza cez križovatku. Slúži to na určenie áut, ktorým treba dať prednosť na križovatke.

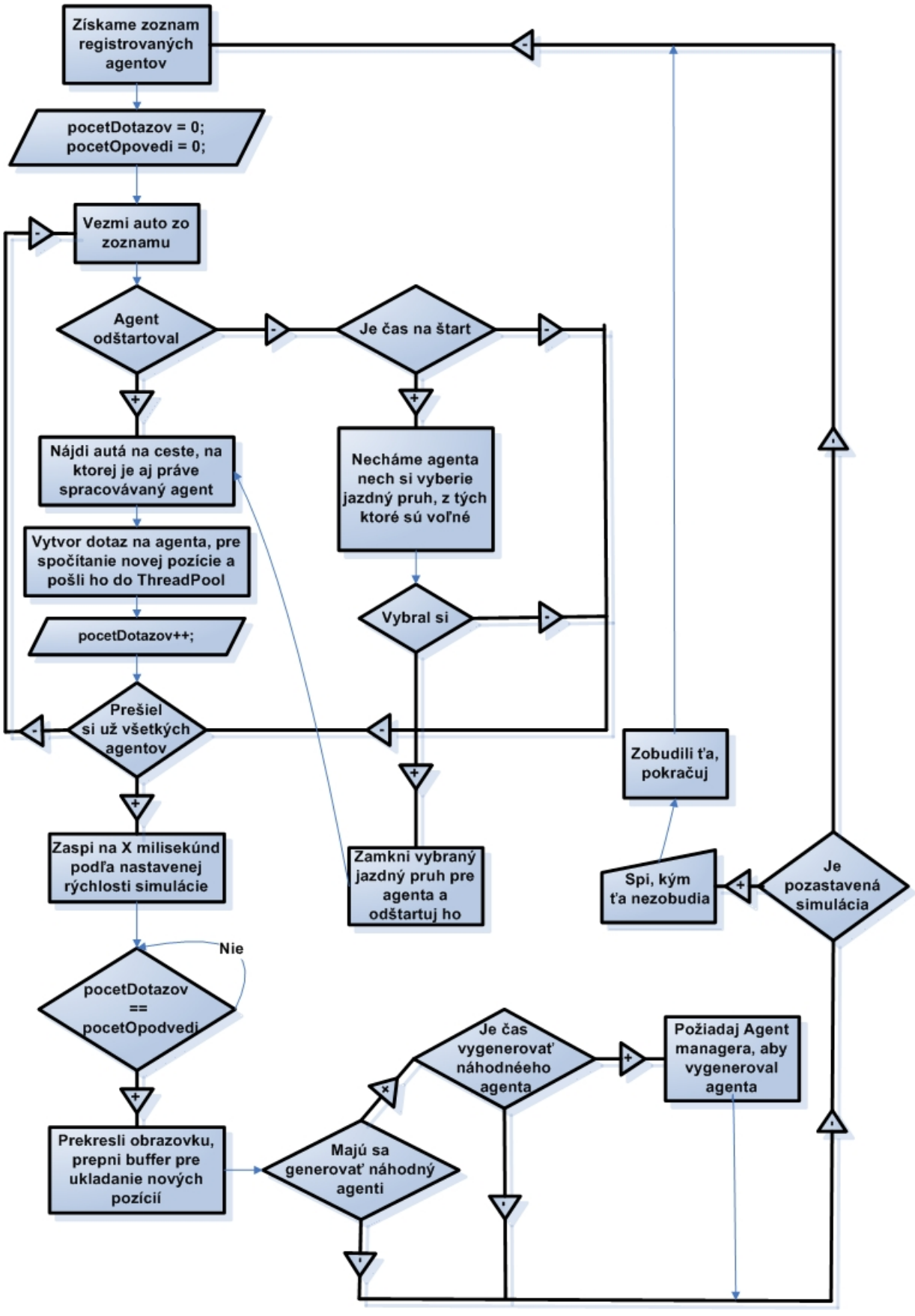
Na záver spomenieme ešte triedu *DotazNaKlientaThread*, ktorá je vlákno obsluhujúce konkrétneho agenta. Klientom sa rozumie práve agent. Keď sa server rozhodne požiadať agenta, aby si spočítal novú polohu, pošle naňho dotaz prostredníctvom tejto triedy. Tá čaká kým agent neodpovie. Vo chvíli keď agent vráti spočítané údaje, trieda informuje server simulácie. Životný cyklus tejto triedy má na starosti správca združovania vlákien *ThreadPool*.

3.4.2 Hlavný cyklus simulácie

Je čas povedať si, ako funguje hlavný cyklus a komunikácia medzi serverom simulácia a agentom. Hlavný cyklus je umiestnený v triede *MoiraSimulaceThread*, ktorá je vlastne vlákno. Okamžite po naštartovaní vlákna sa spúšťa hlavný cyklus, v ktorom sa dokola posielajú dotazy na agentov. Tento cyklus je možné hocikedy pozastaviť, zrýchliť alebo spomaliť pomocou GUI. Na obrázku 3.5 je vidieť, ako je hlavný cyklus definovany.

Hneď prvý pokyn, ktorý sa v cykle vykoná, je získanie agentov, ktorý sú registrovaný na serveri simulácie. Povedzme si teda, ako prebieha registrácia.

Na to, aby sa mohol agent prihlásiť, musí server bežať s už načítanou mapou a vytvorenými grafmi. Ďalej, musí byť spustený aspoň jeden Agent manager, ktorý je prihlásený k serveru. Agent sa môže prostredníctvom neho prihlásiť k serveru. Len menšie upozornenie, že agent je navrhnutý a implementovaný



Obr. 3.5: Popis algoritmu hlavného cyklu

tak, aby nepotreboval Agent managera na prihlásenie k simulácii. Tento postup registrovania bol zvolený z dôvodu, že po prihlásení obdrží každý agent nemalé množstvo dát (dva grafy a mapu). A teda pri veľkom počte agentov, ktorí sa prihlasovali naraz, dochádzalo k dočasnému zamrznutiu simulácie, keďže sa všetok čas venoval na registrovanie nových agentov. Dáta, ktoré agent potrebuje, teraz získava Agent manager, teda k prenosu dochádza iba raz. A agent, ktorý sa registruje cez daný Agent manager dostáva referenciu na tieto dáta.

Vo chvíli, ako agent získa dáta potrebné pre simuláciu, ho môžeme považovať za registrovaného. Zo získaných grafov vie, z ktorých vrcholov sa dá odštartovať. Pri vytváraní agentov ručne, sa ponúkne užívateľovi na výber štart a cieľ. Ak cesta medzi zadanými vrcholmi neexistuje, je užívateľ o tom informovaný. Ak cesta existuje, po nastavení aj ostatných parametrov, je auto pripravené na odštartovanie.

Agenta je možné vytvoriť aj automatiky náhodným vygenerovaním. O tom sa dočítate v paragrafe *Náhodné generovanie agentov.*

Agent pri počítaní novej pozície potrebuje vedieť, kto sa nachádza v jeho okolí. Preto sa pre každého agenta, ktorý sa práve spracováva v cykle, nájdu všetci agenti na rovnakej ceste. Toto vyhľadávanie nie je zdĺhavé, pretože pri ukladaní posledných spočítaných pozícií sa do pomocnej štruktúru poznamenávali agenti, podľa toho na akej ceste sú. Okrem toho, sa môže niekedy agent požiadať servera, aby mu vrátil zoznam všetkých agentov, ktorý sú v okolí nejakej križovatky a smerujú do nej. Táto informácia sa získa podobne ako predošlá.

Keď sú už rozposlané všetky dotazy, server zaspí na krátku dobu, podľa toho, ako je nastavená rýchlosť simulácie a ako dlho trvalo agentom odpovedať na dotaz. Po prebudení sa ešte čaká, ak niekto neodpovedal. Ak sa počet rozposlaných dotazov zhoduje s počtom získaných odpovedí, prepneme buffer, prekreslíme pracovnú plochu, zistíme či netreba vygenerovať náhodného agenta a všetko sa opakuje znovu.

Kapitola 4

Užívateľská dokumentácia

V tejto kapitole sa dozviete, ako vytvárať mapy a ako ich použiť v simulácii. Ako vytvárať vlastných agentov a ako ovládať server simulácie.

4.1 Editor máp

Užívateľ môže za pomoci editora máp vytvoriť mapu dopravnej siete, na ktorej bude skúmať správanie agentov používajúcich navrhnuté algoritmy plánovania ciest. Užívateľovi sú k dispozícii nasledovné prvky modelu: *cesta, križovatka, body zdroj a stok, dopravná značka*. Editor je jednoduchý na ovládanie, ale nie je schopný vyrobiť komplikovanejšie mapy. Na to by bolo potrebné implementovať ďalšie prvky modelu.

S editorom máp sa pracuje ako s jednoduchým grafickým editorom. Všetko sa kreslí na plátno, ktoré je možné pomocou kolieska myši priblížiť alebo oddialiť. Taktiež je možné plátno posúvať buď pomocou šípok, čo funguje aj vtedy, keď máme otvorené dialógové okno (v tej chvíli sa okno s plátnom stáva neaktívne) alebo pomocou myši, a to tak, že podržíme ľubovoľné tlačítko a posúvame myš. Po štarte programu sa nám zobrazí okno ako na obrázku 4.1.

Otvoriť existujúcu mapu môžeme kliknutím na tlačítko *Open* a výberom správneho súboru v diálógu, ktorý sa zobrazí.

Vytvorenú mapu si môžeme uložiť a to kliknutím na tlačítko *Save*. Zobrazí sa dialógové okno, v ktorom si buď vyberieme už existujúci súbor, alebo vytvoríme nový. Bolo by dobré dodržiavať konvenciu a pomenovávať mapy podľa vzoru *mapanieco.xml* .

Ak chceme celú rozpracovanú mapu zmazať a začať kresliť novú, stačí kliknúť na tlačítko *New*.

V nasledujúcich podkapitolách si ukážeme ako sa vytvára mapa.

4.1.1 Cesta

Každá cesta musí mať svoj začiatok a koniec. Môže vychádzať buď z križovatky alebo zo zdroja a končiť buď v križovatke alebo v stoku. Ak by sme chceli použiť mapu v simulácii, kde existuje cesta, ktorá nemá určené, odkiaľ

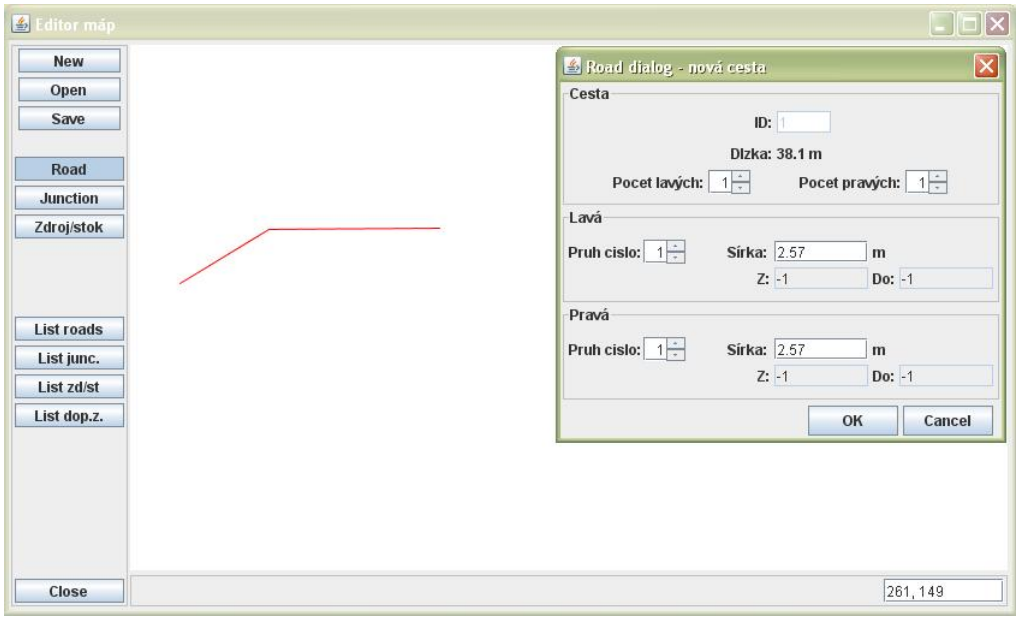
Obr. 4.1: Editor máp po štarte

alebo kam ide, tak nás na to upozorní simulátor. Z takejto mapy nejde vytvoriť graf, ktorý je najdôležitejší z celej mapy. O takejto mape povieme, že je *nevalidná*.

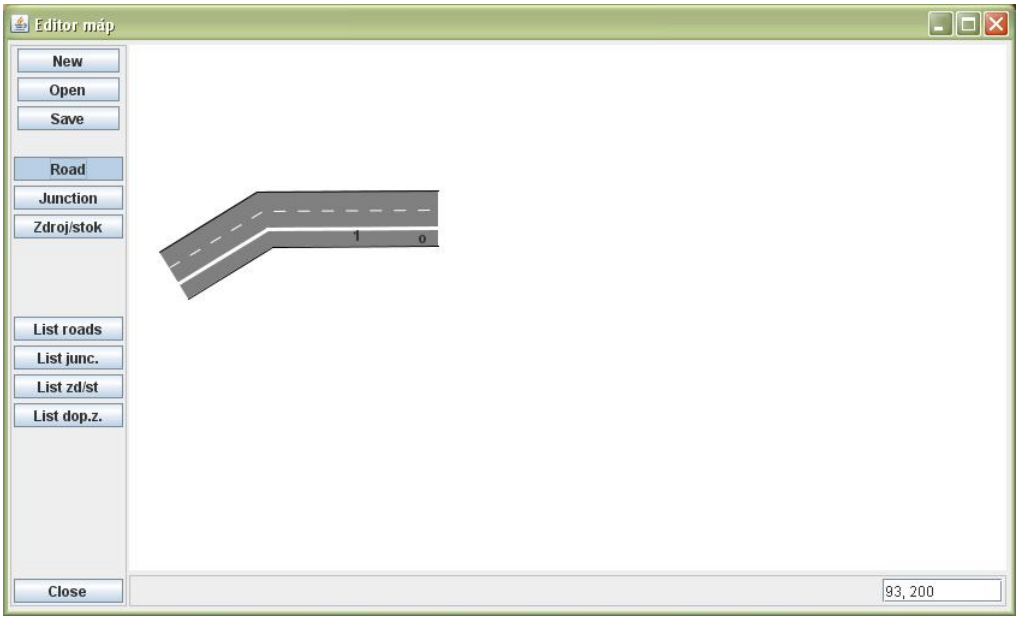
Cesta sa skladá z niekoľkých *úsekov*. Úsek je časť cesty, ktorý je rovný. Nakresliť cestu je veľmi jednoduché. Stačí kliknúť na tlačítko *Road*, posunúť kurzor myši nad miesto, kde bude cesta začínať a kliknúť ľavým tlačítkom. Vo chvíli, ako posunieme myš, budeme vidieť červenú čiaru, ktorá nasleduje kurzor. Kreslíme prvý úsek prvej cesty. Keď kurzor myši dorazí na pozíciu, kam sme chceli, klikneme druhýkrát ľavým tlačítkom. Nakreslil sa prvý úsek a kreslíme druhý, červená čiara sleduje kurzor, presne ako na obrázku 4.2. Ak chceme, aby sa cesta skladala len z dvoch úsekov, tak pri určení koncovej pozície, práve kresleného úseku klikneme na koliesko myši (prostredné tlačítko). V tej chvíli sa ukončí kreslenie celej cesty a objaví sa dialógové okno *Road dialog* ako môžme vidieť na obrázku 4.3. Predstavme si inú situáciu, chceme zmazať posledný nakreslený úsek. Na to stačí kliknúť na pravé tlačítko myši. Takýmto spôsobom môžeme zmazať celú, práve kreslenú cestu.

Road dialog je okno, v ktorom sa môžeme dozvedieť informácie o dĺžke cesty a o ID čísle, ktoré jej bolo pridelené. Okrem toho je to miesto, kde určujeme ako bude cesta v konečnom štádiu vyzerať. Ak chceme, aby mala cesta dva ľavé jazdné pruhy a jeden pravý, stačí zmeniť číslo 1 na 2 pri políčku *Počet ľavých*. V druhej polovici okna sa nastavuje šírka konkrétneho jazdného pruhu. Jazdné pruhy sú číslované od hlavnej stredovej čiary vzostupne. Teda jednotka je jazdný pruh, ktorý je v danom smere najviac vľavo. Ak ide o ľavý pruh, vyberieme poradové číslo a zmeníme mu šírku. Aby k zmene skutočne

Obr. 4.2: Kreslenie cesty



Obr. 4.3: Road dialog



Obr. 4.4: Nakreslená cesta

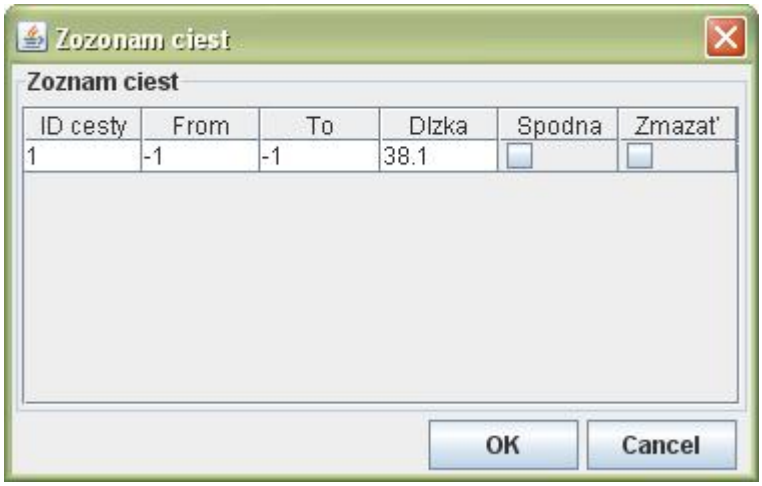
došlo, je nutné vždy po zadaní novej hodnoty stlačiť ENTER. Ak sme skončili prácu v tomto okne, klikneme na tlačítko *OK*, čím vytvoríme cestu. Ak sme si to medzi časom rozmysleli a túto cestu nechceme v mape, stačí kliknúť na tlačítko *Cancel*, čo spôsobí, že sa okno zavrie a posledná kreslená cesta sa zmaže.

Po stlačení tlačítka *OK* sa nám zobrazí ďalšie dialógové okno, ktoré slúži na pridávanie dopravných značiek. V tejto chvíli ho môžeme ignorovať a ukončiť ho. Ako sa pridávajú dopravné značky sa dočítate v podkapitole *Dopravné značky*.

Šírka jazdného pruhu nehrá v simulácii žiadnu rolu. Nebol implementovaný výber takých ciest, aby sa do nich auto zmestilo. Ide iba o vizuálny pocit a zároveň aj o predprípravu do budúcnosti, pre túto možnosť rozšírenia.

Určovanie pravej a ľavej strany cesty je pomerne triviálne. Pre jednoduchosť uvažujme o ceste s jedným úsekom. V smere, v ktorom sme kreslili cestu, určíme pravý vektor. Je to pravá kolmica na náš jeden úsek. Táto kolmica určuje, ktorá strana je pravá, opačná strana cesty je ľavá. Ďalej platí, že cesta má v sebe informáciu odkiaľ kam ide. Tento smer sa vždy udáva v smere v akom sme kreslili cestu. Podrobnejší popis štruktúry cesty nájdete v kapitole *Implementácia*.

Na obrázku 4.4 je vidieť, ako vyzerá už úplne hotová cesta. Cesta má svoj jednoznačný identifikátor(ID číslo) okrem toho si všimnime, že obsahuje aj malý kruh. Ten slúži na to, aby sa dalo určiť, odkiaľ, kam sa cesta kreslila. To je potrebné vedieť pri vytváraní križovatiek. Tam, kde je kruh nakreslený, je miesto, kam smeruje cesta.



Obr. 4.5: Zoznam ciest

Prehľad ciest a ich mazanie je posledné, o čom si povieme v tejto podkapitole. Dialógové okno *Zoznam ciest* slúži práve na tento účel. Zobrazí sa po kliknutí na tlačítko *List roads*. Na obrázku 4.5 si môžme všimnúť aj dve zaškrtávacie políčka. Ak je zaškrtnuté prvé, tak to znamená, že sa bude cesta spomedzi všetkých ciest kresliť medzi prvými. Najprv sa kreslia cesty, ktoré majú príznak, že sú spodné. Až potom ostatné cesty. V akom poradí sú kreslené cesty, ktoré sú spodné, nie je presne dané. Táto možnosť určenia spodnej cesty tu je, aby sa dali nakresliť dve cesty krížom cez seba, kde jedna v skutočnosti ide pod druhou. Vtedy je potrebné určiť, ktorá je tá spodná.

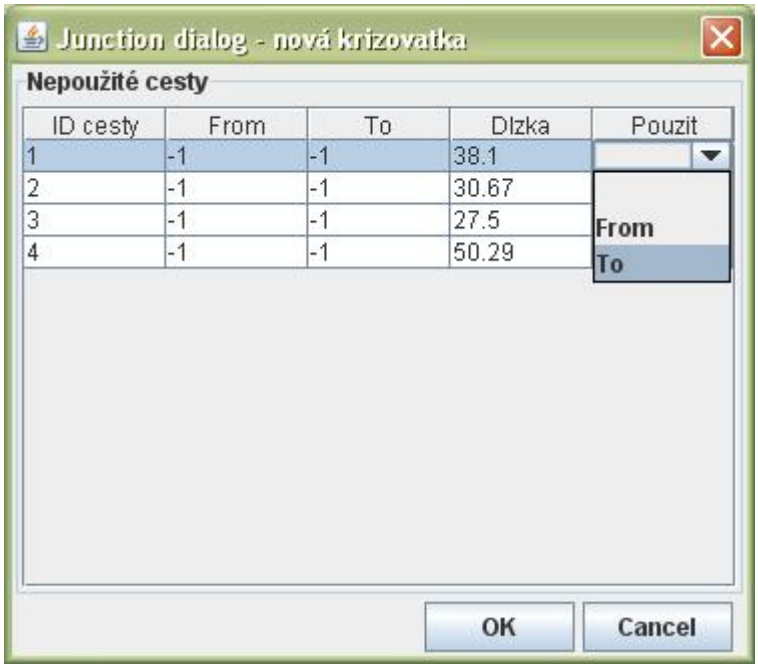
Druhé zaškrtávacie políčko sa používa na určenie ciest, ktoré sa majú zmazať. Ak zaškrtneme zopár ciest a klikneme na *OK*, dialógové okno sa zavrie a zmažú sa aj zaškrtnuté cesty. Kliknutím na *Cancel* sa zavrie okno a nič sa nestane, aj keď sme mali zaškrtnuté hocijaké políčko.

Každý riadok tabuľky obsahuje informácie o dĺžke cesty a odkiaľ kam vedie. Ak v stĺpcoch *From* alebo *To* je hodnota *-1* , znamená to, že daný smer ešte nebol nastavený.

4.1.2 Križovatka

Križovatkou rozumieme miesto, kde sa stretávajú dve, tri alebo štyri cesty. Neodporúča sa vytvárať križovatky, ktoré sú tvorené väčším počtom ciest ako štyri. Problémom je nedostatočne otestované generovanie smerov, v ktorých ma agent dať prednosť.

Vytvorenie križovatky je v editore veľmi jednoduché. Užívateľ musí najprv vytvoriť cesty, ktoré tvoria križovatku a až potom kresliť samotnú križovatku. Je skutočne dôležité dodržať toto poradie. Keď máme cesty pripravené klikneme na tlačítko *Junction*, prejdeme s kurzorom myši nad miesto, kde chceme začať kresliť križovatku, a klikneme na ľavé tlačítko myši. Podobne, ako pri kreslení cesty, sa nám zobrazí červená čiara, ktorá sleduje kurzor. Po-

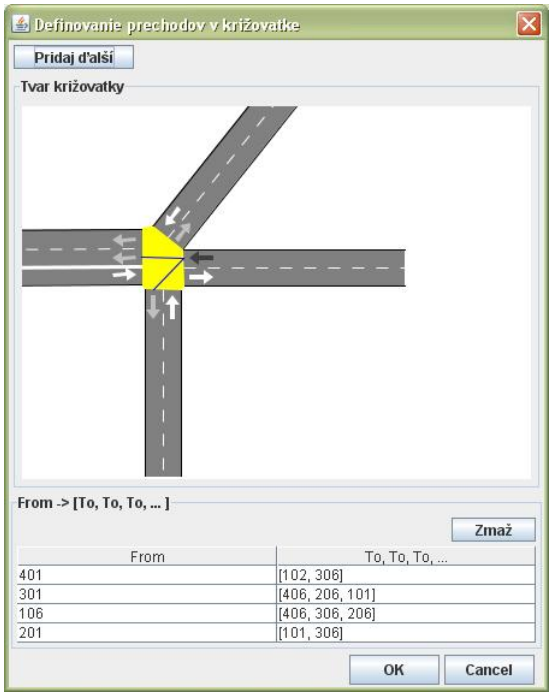


Obr. 4.6: Definovanie ciest, ktoré tvoria križovatku

stupne prechádzame po mape a klikáme ľavým tlačítkom na miesta, kde bude križovatka. Mali by sme dostať konvexný útvar, aj keď iné tvary nie sú vylúčené. Ide len o vizuálny dojem. Križovatka je hotová vo chvíli, keď sa kreslený útvar uzavrie. Lepšie povedané, keď sa dostane kurzor myši do okolia bodu, odkiaľ sme začali a čiara bude zelená. Keď je čiara zelená a kliknutím ľavého tlačítka ukončíme kreslenie, posledná čiara sa automaticky spojí s počiatočným bodom. V tejto chvíli je tvar križovatky hotový. Otvorí sa nám dialógové okno *Junction dialog*, v ktorom užívateľ definuje, ktoré cesty tvoria križovatky. Editor nedokáže sám zistiť, ktoré cesty tvoria križovatku.

Junction dialog 4.6, obsahuje jednu tabuľku, v ktorej je zoznam ciest. Cesta sa zobrazí v tabuľke, iba ak nemá zadaný počiatok, odkiaľ vychádza alebo koniec, kam vedie. Inak povedané, sú to cesty, ktoré môžu ešte tvoriť križovatku. Určiť, ktoré cesty tvoria križovatku, je veľmi jednoduché. Posledný stĺpec tabuľky obsahuje "combo box", v ktorom sú na výber dve položky *From* - znamená že cesta vychádza z križovatky a *To* – cesta vchádza do križovatky. Užívateľ si musí prečítať ID čísla ciest, ktoré tvoria križovatku. Potom sa pozrieť, na ktorej strane má cesta kruh. Ak je to na strane bližšej ku križovatke, tak takejto ceste nastaví, že je *To*, inak *From*.

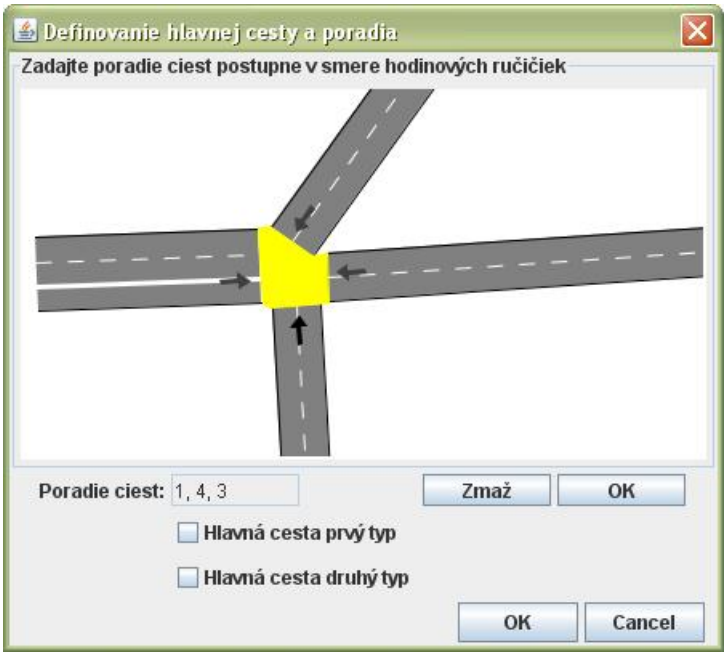
Ak užívateľ vybral už všetky cesty, kliknutím na tlačítko *OK* potvrdí výber a dostane sa do ďalšieho dialógového okna *Definovanie prechodov*, v ktorom určí, kade môže auto ísť cez križovatku. Kliknutím na tlačítko *Cancel* sa zmaže posledná kreslená križovatka a zavrú sa všetky dialógové okná.



Obr. 4.7: Definovanie prechodov cez križovatku

Prechody cez križovatku sú množiny bodov, konkrétne vrcholov, kam sa môže auto dostať z nejakého iného vrcholu. Pozrime sa na to trochu podrobnejšie. Už vieme, že každá cesta je zložená z úsekov, ale taktiež aj z jazdných pruhov. Predstavme si, že v mieste, kde vchádza jazdný pruh do križovatky je takzvaný *vstupný* vrchol a v mieste, kde jazdný pruh vychádza z križovatky je *výstupný* vrchol. V dialógovom okne 4.7 môžeme vidieť nakreslenú križovatku a cesty, ktoré ju tvoria. Okrem toho sú na cestách šípky, ktorých smer určuje či ide o vrchol *vstupný* alebo *výstupný*. Úlohou užívateľa je len správne poprepájať tieto šípky. Dve šípky môžeme prepojiť, iba ak prvá je vstupná a druhá výstupná. Prepojenie takýchto dvoch šípok znamená, že ak auto bude v tom jazdnom pruhu ako vstupná šípka, tak bude môcť ďalej pokračovať do jazdného pruhu, v ktorom je výstupná šípka. Auto môže vojsť iba do toho jazdného pruhu, do ktorého je definovaný prechod z pruhu, z ktorého vychádza.

Definovanie prechodov je veľmi jednoduché, editor robí polovicu práce za užívateľa. Ak zájdeme kurzorom nad jednu zo šípok zafarbí sa na červeno, ak ide o šípku výstupnú. Tu nemôžme začať s definíciou prechodu. Alebo sa zafarbí na zeleno, ak ide o šípku vo vstupnom smere. Z takýchto šípok môžeme začať definíciu, a to kliknutím na ľavé tlačítko myši. Po kliknutí sa počiatočná šípka zafarbí na čierno a všetky výstupné šípky na šedú farbu. Šedé šípky sú tie, do ktorých môžeme definovať prechod, do iných sa nedá. Prechod je úplne definovaný, ak klikneme na šedú šípku. Ak sme už vybrali prechody, ktoré potrebujeme, kliknutím na tlačítko *Pridaj ďalší* sa šípky odfarbia a môžme zadávať nové prechody. Počas pridávania prechodov sme si mohli všimnúť, že aj

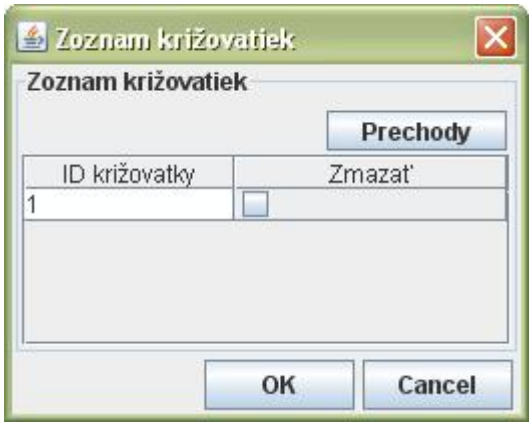


Obr. 4.8: Definovanie hlavnej cesty

v tabuľke postupne pribúdali prechody. Tabuľku je možné využiť na prehľad už zadaných prechodov, a to kliknutím na riadok v tabuľke, alebo môžeme zmazať všetky prechody z jednej vstupnej šípky kliknutím na tlačítko *Zmaž*. Samozrejme musí byť vybraný riadok v tabuľke. Ak sa rozhodneme časom dodefinovať ešte nejaký prechod, tak stačí kliknúť na zelenú šípku, zobrazia sa už definované prechody a potom pridáme ďalší.

Ak sme zadali už všetky prechody, klikneme na tlačítko *OK*, čím sa presunieme do ďalšieho okna, v ktorom sa definuje hlavná cesta. Ak užívateľ nedefinoval všetky prechody, vyskočí upozornenie. Stane sa to vtedy, ak existuje vstupný jazdný pruh, ktorý nedefinuje nikam prechod alebo ak existuje výstupný jazdný pruh, do ktorého nevedie žiadny prechod. Takéto jazdné pruhy bolo buď zbytočné vytvárať v ceste alebo užívateľ na ne len zabudol. Tlačítko *Cancel* ukončí vytváranie križovatky a zmaže celú poslednú vytváranú križovatku.

Hlavná cesta sa určuje za pomoci postupnosti ciest, ktoré tvoria križovatku. Postupnosť musí vytvoriť užívateľ a to nasledovne. Zvolí si ľubovoľnú cestu ako počiatočnú, klikne na šípku, ktorá sa nachádza na tejto ceste. A ďalej v smere hodinových ručičiek postupne označuje ďalšie šípky(cesty). Ak sú označené všetky klikne na tlačítko *OK*. Je vedľa tlačítka *Zmaž*, ktoré má za úlohu zmazať zadanú postupnosť. Po potvrdení zvoleného poradia ciest vyberie hlavnú cestu, zaškrtnutím jedného z políčok. Hlavnú cestu tvoria dve šípky zafarbené na zeleno. Pre križovatky tvorené štyrmi cestami platí, že hlavnou cestou môžu byť iba dve protiľahlé cesty. Pre križovatky z troch ciest sa určí niektorá z dvojíc ciest.



Obr. 4.9: Zoznam vytvorených križovatiek

Na obrázok 4.8 môžeme vidieť dialógové okno pre určenie hlavnej cesty a jedno z možných označení poradia.

Ak si užívateľ vybral hlavnú cestu, kliknutím na tlačítko *OK* potvrdí voľbu. V tej chvíli je vytváranie križovatky hotové. Editor sám vygeneruje smery, podľa ktorých má dať vodič prednosť inému, a to podľa zadefinovanej hlavnej cesty. Ak si to užívateľ rozmyslí, kliknutím na tlačítko *Cancel* sa posledná vytváraná križovatka zmaže a užívateľ sa vráti späť do hlavného okna editora.

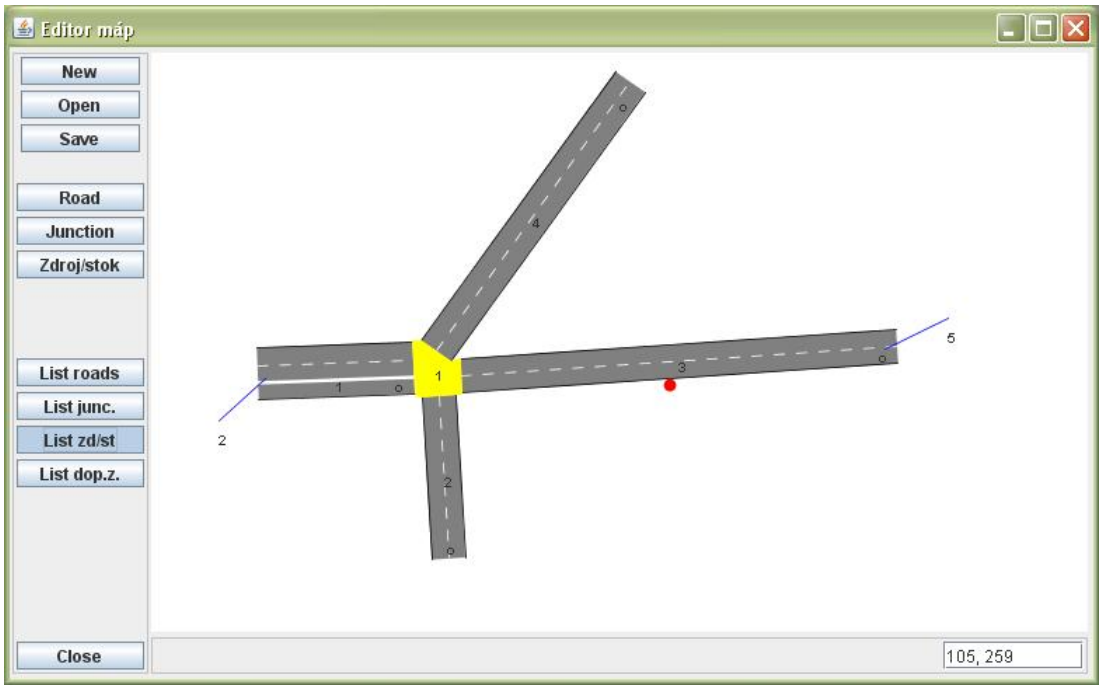
Prehľad vytvorených križovatiek funguje podobne ako prehľad ciest. Kliknutím na tlačítko *List junc.* v hlavnom okne editora, sa zobrazí okno 4.9, v ktorom je možné križovatku zmazať prípadne si overiť správnosť definovaných prechodov križovatkou. Na to je potrebné, aby užívateľ klikol na jeden riadok v tabuľke, a následne na to na tlačítko *Prechody*. Otvorí sa dialógové okno pre definovanie prechodov, ktoré obsahuje už dáta, ktoré sme zadali pri vytváraní križovatky. Práca s týmto oknom je popísaná v predchádzajúcich paragrafoch. Po kliknutí na *Cancel* sa nám okno zavrie bez toho, aby sa križovatka zmazala. Križovatku zmažeme označením zaškrtávacieho políčka v okne so zoznamom križovatiek a kliknutím na *OK*.

4.1.3 Zdroj, stok

Ďalší dôležitý prvok mapy je bod, z ktorého auto odštartuje *zdroj* a bod do, ktorého sa chce dostať *stok*. Ďalej ho budeme nazývať ako bod zdroj/stok. Môže nadobúdať tri rôzne typy:

- zdroj
- stok
- zdroj a stok zároveň

Pre bod typu *zdroj* sa uvádza číslo lambda, ktoré sa používa pri náhodnom generovaní áut podľa exponenciálneho rozdelenia. Čím je toto číslo väčšie, tým



Obr. 4.10: Nakreslený bod zdroj/stok

častejšie budú z daného zdroja vychádzať autá. Každé auto mieri do bodu, ktorý je typu *stok*. Ktorým smerom auto pôjde, sa tiež generuje náhodne podľa normálneho rozdelenia. Bod, ktorý je stokom má tiež číslo lambda, ktoré hovorí, s akou pravdepodobnosťou bude cieľom áut daný stok. Nemusí platiť, že súčet všetkých lámbd čísel stokov na mape sa rovná 1. Je na užívateľovi, aby si sám zabezpečil správne očíslovanie stokov, ktoré by zodpovedalo pravdepodobnosti výberu vrcholov, ako si zvolil. Aplikácia simulácie dokáže z tohto očíslovania vytvoriť náhodný výber podľa normálneho rozdelenia. Pozrime sa ako sa taký bod zdroj/stok vytvára.

Vytvorený bod zdroj/stok je vidieť na obrázku 4.10. Tvar tohto objektu je úsečka, ktorá má jeden bod mimo cestu a druhý na ceste, na ktorú sa bod zdroj/stok napája. Je jasné, že správnosť nastavenia typu bodu závisí od užívateľa. Napríklad, ak ide o jednosmernú cestu, ktorá vychádza z takého bodu, tak tomuto bodu nastavíme, že je typu zdroj.

Kreslenie bodu zahájime kliknutím na tlačítko *Zdroj/stok* v hlavnom okne editora. Presunieme sa kurzorom na miesto, kde chceme zadať prvý bod úsečky, ktorá bude reprezentovať bod zdroj/stok. Klikneme na ľavé tlačítko myši. Kurzor nám bude sledovať zelená čiara. Druhý bod úsečky by sme mali umiestniť na cestu, do ktorej má bod zdroj/stok viesť. Ak sme na mieste, klikneme opäť na ľavé tlačítko, čím dokončíme kreslenie. Zobrazí sa nám dialógové okno podobné oknu pri definovaní ciest, ktoré tvoria križovatku. V ňom nastavíme, na riadku s ID cesty, do ktorej bude viest bod zdroj/stok, v poslednom stĺpci, jednu z možností *From* alebo *To*. Ak je kruh na ceste bližšie k bodu zdroj/stok

Obr. 4.11: Dialógové okno, v ktorom nastavujeme parametre bodu zdroj/stok

tak nastavíme *To* inak *From*. Užívateľ môže vybrať iba jednu cestu.

Ak sme nastavili cestu, kliknutím na tlačítko *OK* potvrdíme voľbu a dostávame sa k nastaveniu parametrov bodu zdroj/stok v dialógovom okne *Zdroj/stok*. Kliknutím na *Cancel* sa zmaže posledný vytváraný bod zdroj/stok a dostaneme sa do hlavného okna editora.

Nastavenie parametrov bodu sa robí za pomoci okna *Zdroj/stok*, viď. na obrázku 4.11. Užívateľ má možnosť zvoliť typ bodu a podľa toho čo zvolí, sa sprístupnia políčka na zadanie lambda čísla. Ak si užívateľ vybral, kliknutím na *OK* potvrdí voľbu. V tej chvíli je bod zdroj/stok už hotový. Kliknutím na *Cancel* sa zmaže posledný vytváraný bod.

Prehľad a editovanie bodov zdroj/stok sa vykonáva za pomoci dialógového okna *Zoznam bodov zdroj/stok*, ktoré je vidieť na obrázku 4.12. Každý riadok tabuľky obsahuje informáciu o type bodu zdroj/stok, ID cesty, do ktorej smeruje a lambda čísle, podľa toho o aký typ bodu ide. Ak je niektoré lambda číslo rovné -1.0, tak ide o nezadanú hodnotu. V poslednom stĺpci je zaškrtávacie políčko, ktoré keď zaškrtneme, označíme bod zdroj/stok, ktorý chceme zmazať. Všetky označené body sa zmažú po kliknutí na *OK*.

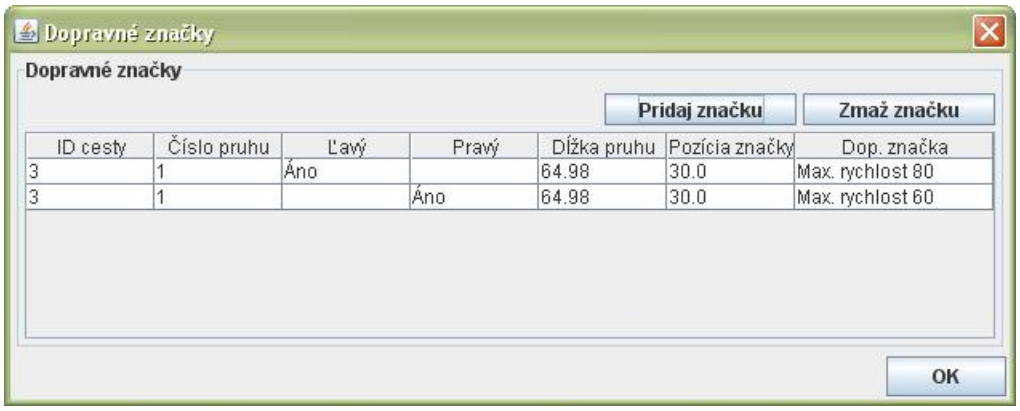
Pre zmenu typu bodu alebo lambda čísel, označíme riadok, na ktorom sa nachádza požadovaný bod a klikneme na tlačítko *Edituj*. Otvorí sa okno *Zdroj/stok*, ktorého funkciu sme už popisovali v predošlom odseku.

4.1.4 Dopravná značka

V tejto verzii podporuje editor máp iba dopravné značky obmedzujúce rýchlosť vozidla. Sú to

- Maximálna rýchlosť 40 kilometrov za hodinu
- Maximálna rýchlosť 60 kilometrov za hodinu
- Maximálna rýchlosť 80 kilometrov za hodinu
- Maximálna rýchlosť 100 kilometrov za hodinu

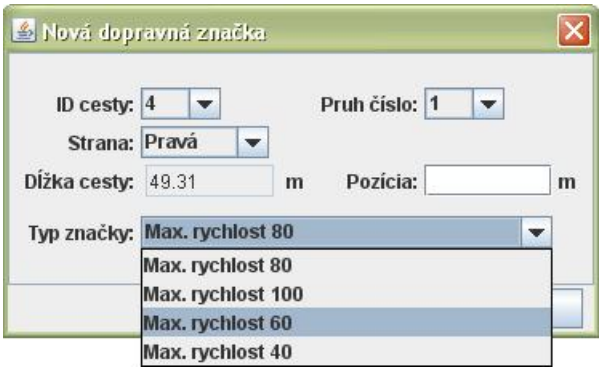
Obr. 4.12: Zoznam vytvorených bodov zdroj/stok



Obr. 4.13: Zoznam dopravných značiek na cestách

Na pridávanie alebo mazanie dopravných značiek slúži dialógové okno *Dopravné značky* viď. 4.13, ktoré sa otvorí po kliknutí na tlačítko *List dop.z.* v hlavnom okne editora. Toto okno sa nám otvorí aj vo chvíli, keď dokončíme vytváranie novej cesty.

Novú dopravnú značku pridáme kliknutím na tlačítko *Pridaj značku* v okne *Dopravné značky*, čím sa nám otvorí dialógové okno *Nová dopravná značka* viď. 4.14. Užívateľ by mal dodržať nasledovný postup zadávania údajov. Zvolí si cestu, na ktorú chce pridať značku. Hneď ako si vyberie cestu zmení sa hodnota v políčku *Dĺžka cesty*. Do susedného políčka *Pozícia* zapíše pozíciu novej značky. Hodnota by mala byť väčšia ako 0 a menšia ako hodnota v políčku *Dĺžka cesty*. Ďalej si vyberieme či ide o ľavý jazdný pruh alebo pravý. Následne na to zvolíme číslo jazdného pruhu, v ktorom sa zobrazí nová dopravná značka. Len pripomeňme že, jazdné pruhy sú číslované od hlavnej stredovej čiary rastúco. Teda jednotka je jazdný pruh, ktorý je v danom smere najviac vľavo.



Obr. 4.14: Pridanie novej dopravnej značky

Odstránenie dopravnej značky sa robí tiež v okne *Dopravné značky*. Užívateľ kliknutím na riadok v tabuľke vyberie dopravnú značku, ktorú chce odstrániť. Klikne na tlačítko *Zmaž značku* a potvrdí upozornenie, ktoré sa zobrazí.

4.2 MoiRa – server simulácie

Keďže cieľom projektu je navrhnúť systém, ktorý by bolo možné nasadiť aj do reálnych situácií je potrebné mať k dispozícii nástroj, pomocou ktorého otestujeme vlastnosti systému.

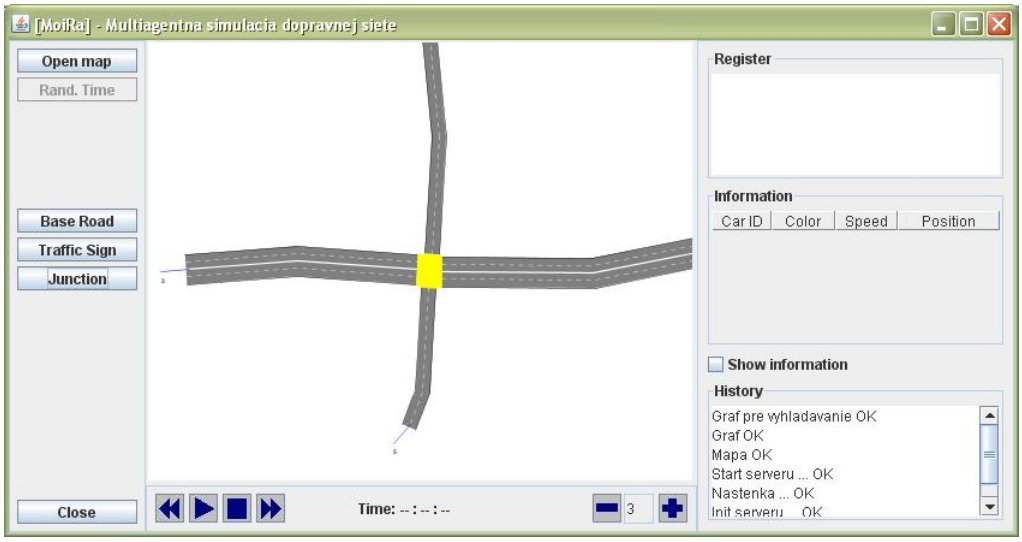
Server simulácie je aplikácia, na ktorej má užívateľ možnosť odskúšať funkčnosť navrhnutých algoritmov plánovania ciest alebo algoritmov pohybu agentov. Tento program by sme mali vnímať ako pomocný prvok, ktorý nahradzuje reálny svet. Aby server fungoval potrebujeme mapu dopravnej siete, ktorú sme si vopred pripravili v aplikácii *Editor máp*. Mapa musí byť v korektnom stave, aby bolo možné z nej vygenerovať grafy, ktoré sa používajú v simulácii. Kedy je mapa v korektnom stave sa dočítate v podkapitole *Cesta*.

Keď server beží, je možné prihlásiť niekoľko agentov do simulácie. Po prihlásení, sa budú na podnet servera, pohybovať po mape, pričom každý agent dodržiava dopravné predpisy. Úlohou server simulácie je umožniť agentom prihlásiť sa do práve prebiehajúcej simulácie, posielať im podnety nato, aby sa pohybovali a vykresľovať agentov na mape.

Meno serveru *MoiRa* pochádza z gréckej mytológie. Je pomenovaný podľa bohyne, ktorá dokázala ľudom predpovedať osud.

Ovládanie behu servera sa robí pomocou tlačidiel na spodnej lište. Je možné spustiť simuláciu, pozastaviť, zrýchliť, spomaliť alebo reštartovať. Počas reštartu, server informuje agentov, čo ukončí ich existenciu. Server sa pripraví na nový štart.

Pri pohľade na obrázok 4.15, si môžme všimnúť aj textové polia pre históriu behu servera a zoznam registrovaných áut(agentov). Musíme si ale uvedomiť, že keď je auto registrované(prihlásené) na server simulácie, neznamená to, že sa aj aktívne zúčastňuje simulácie. Podrobnejší popis vzťahov medzi agentom



Obr. 4.15: Simulačný server MoiRa

a serverom simulácie nájdete v kapitole *Implementácia*. Nespomenuli sme ešte možnosť nahliadnuť na informácie o agentovi a to konkrétne na jeho aktuálnu rýchlosť, pozíciu a farbu. Vzhľadom k tomu, že na overenie funkčnosti systému bude chcieť užívateľ, aby bolo na mape v jednom okamžiku čo najviac agentov, je možne informácie o agentoch nezobrazovať. Zobrazovanie týchto informácií môže v niektorých situáciách výrazne spomaliť beh simulácie. Kliknutím na zaškrtávacie tlačítko *Show information* povolíme alebo zakážeme aktualizovať informácie o agentoch.

Predtým ako sa pustíme k opisu ovládania serveru ešte pripomeniem, že podobne ako v *Editore máp*, je aj tu možné pohybovať, zmenšovať a zväčšovať mapu pomocou myšky.

Spustiť server je možné hneď potom, ako vyberieme mapu, na ktorej má prebiehať simulácia. Na bočnej lište má užívateľ k dispozícii tri tlačítka. *Base Road* nám otvorí mapu jednoduchej cesty zo zdroja do stoku, na ktorej je možné pozorovať správanie sa agentov na rovných úsekoch cesty. Ako sa predbiehajú, ako sa zaraďujú späť a ako sa vyhýbajú, aby do seba nevrazili je opísané v podkapitol *Návrh algoritmov*. Tlačítkom *Traffic Sign* sa otvorí mapa opäť jednoduchej cesty, ale tento krát obsahuje dopravné značky. Mapa slúži na testovanie správneho dodržiavanie dopravného značenia. Ako posledná pred pripravená mapa, je mapa jednoduchej križovatky, na testovanie, či agent dáva prednosť iným agentom, ak je to potrebné. Tú otvoríme kliknutím na *Junction*. Užívateľ má možnosť použiť aj inú mapu a to kliknutím na tlačítko *Open map*. Ak je mapa pripravená môžeme spustiť server simulácie kliknutím na tlačítko *Play* na spodnej lište. V textovom poli *History* sa nám okrem iného zobrazia aj informácie o pripravení dvoch grafov, ktoré používajú agenti. V tej chvíli je server simulácie pripravený. Užívateľ môže až v tejto chvíli vytvoriť agentov. Ako pridávať agentov sa dočítate ďalej.

Obr. 4.16: Nastavenie náhodného generovania agentov

Vytvoriť agenta je možné dvomi spôsobmi.

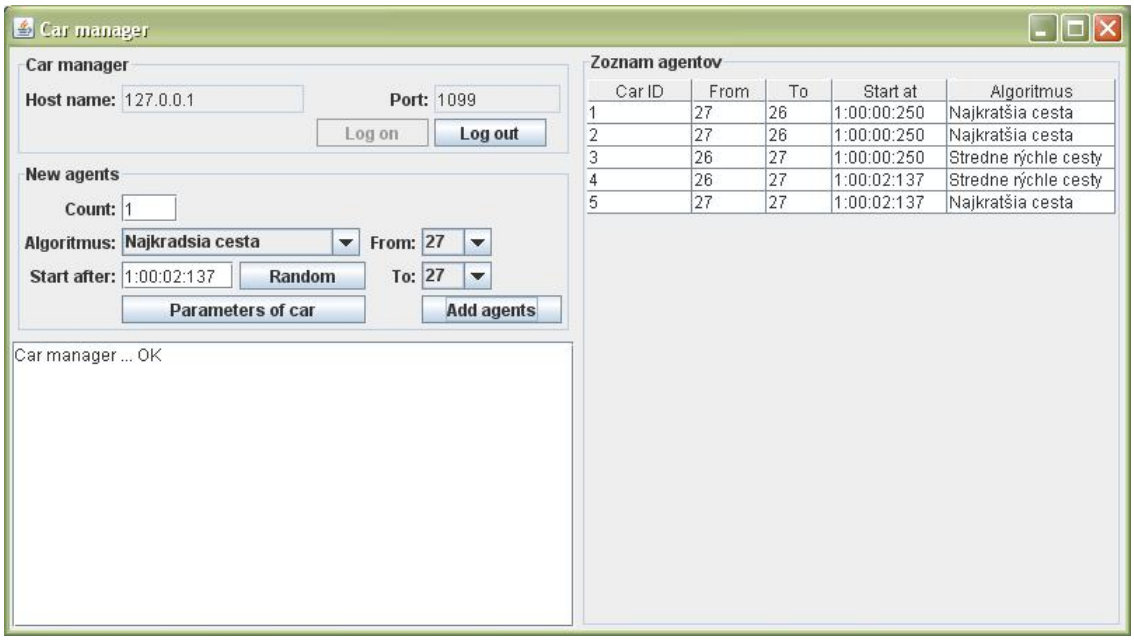
- Automaticky - pomocou serveru simulácie. Popíšeme si ho v tejto časti.
- Ručne – cez Agent manager. Ukážeme neskôr.

Oba spôsoby potrebujú aplikáciu *Agent manager*. Čo to je a ako funguje sa dozvieme v podkapitole venovanej tejto aplikácii. Pre jednoduchosť nám stačí predpokladať, že server simulácie úspešne beží a *Agent manager* je prihlásený k serveru. Potom kliknutím na tlačítko *Rand. Time* sa otvorí okno, ktoré slúži na nastavenie automatického generovania agentov v simulácii. Na obrázku 4.16 je možné toto okno vidieť. V dialógovom okne sa určí časový úsek, počas ktorého sa majú generovať náhodne agenti. Akým spôsobom sa generujú agenti sa podrobne dočítate v kapitole *Implementácia*.

Okrem nastavenia času, je dôležité ešte vybrať množinu algoritmov, z ktorých sa bude vyberať, pri generovaní agentov. Každý agent potrebuje algoritmus podľa, ktorého sa riadi a rozhoduje. Algoritmus z množiny má nastavené, s akou pravdepodobnosťou sa má vybrať pre práve vygenerovaného agenta. Aby sme si lepšie predstavili čo znamenajú tieto pravdepodobnosti uvediem príklad. Na obrázku 4.16 sme vybrali dva, kde jeden má nastavené 3 a druhý 1. Znamená to, že traja agenti zo štyroch vygenerovaných budú používať daný algoritmus.

Pre lepšie rozlíšenie agentov používajúcich konkrétny typ algoritmu, je možné nastaviť farbu ich auta. Preto sa doporučuje, aby každý typ algoritmu používal inú farbu.

Vybrané algoritmy sú umiestnené v tabuľke napravo. Užívateľ má možnosť zmazať konkrétny typ algoritmu označením riadku v tabuľke a kliknutím na tlačítko *Zmaž*. Po kliknutí na tlačítko *OK*, sa vezmú vždy všetky algoritmy v tabuľke a použijú sa pri generovaní. Pri opätovnom otvorení dialógu ostávajú v tabuľke posledné zadané typy algoritmov koly rýchlejšiemu ovládaniu.



Obr. 4.17: Agent manager

4.3 Agent manager

Agent manager je aplikácia, ktorá spravuje agentov prihlásených k simulácii. Slúži na vytváranie agentov a ich pridávanie do simulácie. V predošlých kapitolách sme sa zmienili, že existujú dva spôsoby ako pridať agenta do simulácie. Popísali sme spôsob automatického pridávania pomocou náhodného generovania. Pomocou tejto aplikácie je možné pridávať agentov aj ručne.

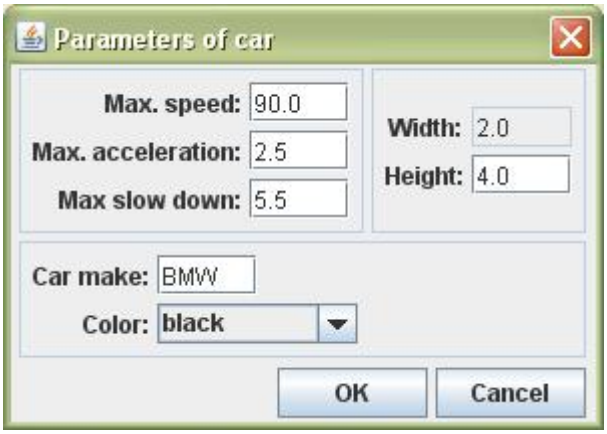
Po spustení aplikácie sa nám zobrazí okno ako na obrázku 4.17. Aby sme mohli vytvoriť a pridať agenta do simulácie je potrebné, aby server simulácie bežal a aby bola mapa načítaná.

Prihlásenie Agent managera k serveru simulácie, sa vykoná kliknutím na tlačítko *Log on*. Predtým je treba nastaviť správnu adresu servera a port, na ktorom služba (simulácia) beží. Adresa servera môže byť, buď **IP adresa** alebo **domain name**. Ak bude server aj manager na tom istom počítači stačí sa len prihlásiť. Ak by sme ale chceli simulovať na rozsiahlej mape, je vhodné umiestniť server simulácie na samostatný počítač a vytvoriť niekoľko Agent managerov. K serveru môže byť prihlásených naraz ľubovoľný počet a každý z nich môže byť na ľubovoľnom počítači.

Po prihlásení Agent managera, sa budú vytvoraní agenti zobrazovať v tabuľke vpravo, v ktorej je vidieť informácie o tom, aký typ algoritmu agent používa, odkiaľ kam ide a čas kedy sa má najskôr začať pohybovať.

Kliknutím na tlačítko *Log out*, odhlásime Agent managera zo servera simulácie a zároveň aj všetkých agentov, ktorý boli pomocou neho vytvorení.

Pridať agenta môžme pomocou tlačítka *Add agents*. Predtým je ale treba nastaviť parametre. V dialógovom okne 4.18, ktoré sa otvorí po kliknutí na



Obr. 4.18: Dialógové okno pre nastavenie parametrov auta

Parameters of car, nastaví užívateľ farbu auta, výrobcu, čo môže byť ľubovoľný text, maximálnu rýchlosť, ktorou dokáže ísť, maximálne možné zrýchlenie, maximálne možné spomalenie a rozmery. Keďže nie je implementované rozlišovanie, či sa auto zmestí na cestu, (môže byť veľmi široké), odporúča sa nechať rozmery podľa počiatočného nastavenia. Taktiež je veľmi dôležité zvážiť ako sa nastavia parametre zrýchlenia a spomalenia.

Rada pre užívateľa: maximálne spomalenie by malo byť aspoň 1,1 násobok maximálneho zrýchlenia.

Po potvrdení nastavenia musí užívateľ určiť, z ktorého vrcholu agent odštartuje *From* a do ktorého sa má dostať *To*. Ďalej sa nastaví čas, kedy sa má agent začať pohybovať. Tento čas neznamená, že pohyb agenta začne presne v daný čas, ale hovorí že nezačne skôr. Podrobnejší opis toho ako auto odštartuje nájdete v kapitole *Implementácia*.

Aby mohol agent v simulácii plnohodnotne existovať, potrebuje algoritmus pomocou, ktorého sa bude pohybovať a plánovať si svoju cestu. Ten musí užívateľ tiež vybrať. Aké typy algoritmov sú k dispozícii a ich popis nájdete v podkapitolách venované algoritmom.

Keďže pridávanie agentov po jednom je veľmi nepraktické, má užívateľ možnosť pridať naraz ľubovoľný počet agentov. Zadaním čísla do políčka *Count* sa zadá počet agentov, ktorý sa majú vytvoriť naraz. Všetci budú mať rovnaké parametre.

Agent ktorý dorazí do svojho cieľa sa sám odhlási a teda sa už nebude viac nachádzať v tabuľke agentov prihlásených k serveru simulácie.

Kapitola 5

Záver

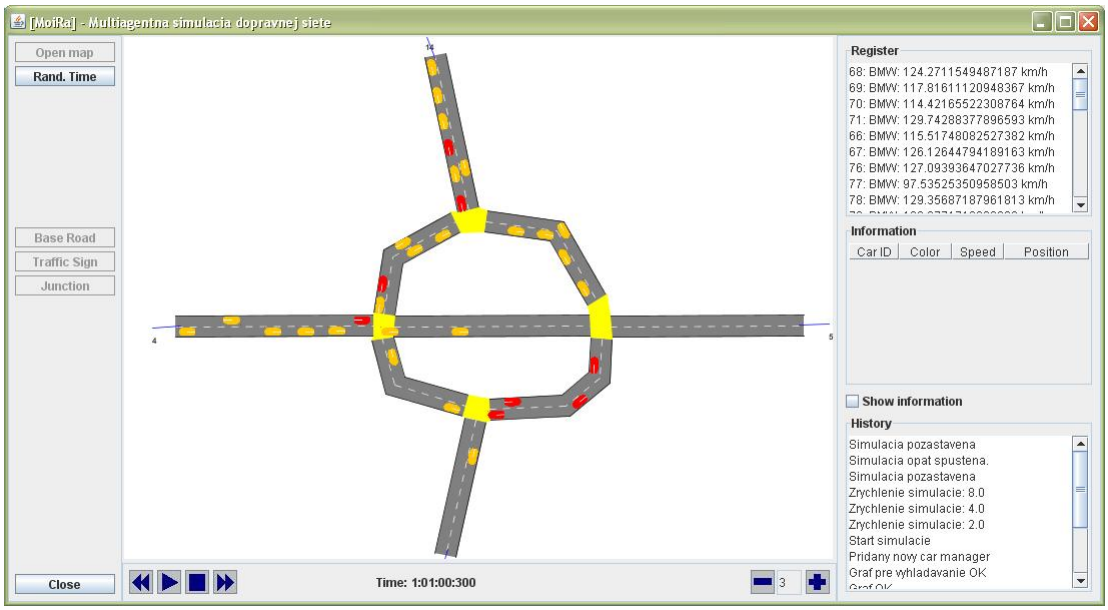
Systém bol od začiatku navrhovaný tak, aby sa dal ďalej vyvíjať, až do fázy nasadenie do skutočnej dopravy. Vzhľadom k tomu, že zaobstaranie hardwaru, potrebného pre nasadenie do praxe by bolo náročné, hlavne finančne, bol vytvorený program pre simulovanie cestnej dopravy, na ktorom by bolo možné vidieť, či bol systém navrhnutý správne. Okrem toho simulácia pomáha odhaliť rôzne typy problémov, napríklad chyby v navrhovanom plánovači trasy. Taktiež, veľkou výhodou existencie simulátora je, možnosť testovania rôznych typov plánovačov a ladenie prípadných chýb. Nájdenie chyby počas simulácie je pravdepodobnejšie, ako v reálnych podmienkach, keďže potrebných testovacích dát môžeme zo simulátora získať viac, ako z praxe. Taktiež, dokážeme jednoduchšie navodiť situácie, ktoré v reálnom svete môžu nastať iba zriedkavo.

Okrem spomenutého serveru simulácie, môžeme pokladať za prínos práce možnosť skúmania, ktoré časti dopravnej siete sú nevhodne navrhnuté. Vďaka editoru máp, sa môžeme pokúsiť tieto úseky siete vhodne modifikovať, napríklad zväčšiť počet jazdných pruhov. Následne nato otestovať zmenenú dopravnú sieť v serveri simulácie. Takže možnosti použitia simulátora sú aj v tomto smere.

5.1 Výsledky testovania

V tejto časti si povieme, ako dopadli výsledky rôznych testov navrhnutých algoritmov plánovania. Predpokladá sa, že čitateľ pochopil základné princípy práce so serverom simulácie a Agent managerom. Popis, ako ovládať tieto aplikácie je uvedený v kapitole *Užívateľská dokumentácia*. Pokusy bolo nutné popísať, vzhľadom k tomu, že na vytvorenie videí nebola dostupná dostatočná výpočtová kapacita. Na obrázku 5.1 je vidieť, ako to vyzerá počas simulácie. Červené autíčka reprezentujú agentov, ktorí medzi sebou komunikujú, oranžové autíčka sú agenti, ktorí nekomunikujú s nikým a riadia sa podľa plánu, ktorý bol zostavený ešte pred štartom.

Aby sme skutočne ukázali funkčnosť navrhovaného systému, ukážeme najprv, že agenti medzi sebou skutočne komunikujú a plánujú si svoje trasy podľa toho.



Obr. 5.1: Stav dopravnej siete počas behu simulácie

Najprv si spustíme server simulácie s priloženou mapou *test1.xml*. Taktiež, spustíme Agent managera a prihlásime ho k serveru. Potom na serveri nastavíme náhodné generovanie agentov, tak aby používali algoritmus najkratšej cesty. Ak toto všetko máme pripravené, môžeme odštartovať simuláciu. Po štarte sa nám po mape rozbehnú náhodný agenti. Môžeme chvíľku počkať, kým sa niekde nevytvorí zápcha, prípadne vhodne odštartovať nových agentov, ručne vygenerovaných, ktorí ju isto spôsobia. Vo chvíli ako spozorujeme miesto, ktoré sa stáva upchané, je vhodný čas pustiť do mapy agentov, ktorí medzi sebou komunikujú.

Vyberieme jeden z algoritmov *StredneRychleCesty* alebo *NajrychlejsieCesty*, nastavíme farbu áut na inú, ako sú momentálne farby na mape, koly tomu, aby sa nám lepšie rozoznával tento typ agentov, a odštartujeme ich. Je vhodné všetkých poslať z toho istého štartu, do toho istého cieľa. Aby sme videli, keď niektorý z nich pôjde inou trasou.

Prvý z týchto agentov pôjde určite najkratšou cestou, keďže ho ešte nemal kto informovať o cestách na trase, ktorá je najkratšia. Ale je možné že už druhý zvolí inú trasu ako jeho predchodca.

Ak sa pokus nepodaril a všetci išli rovnakým smerov zopakujeme to celé ešte raz, Je totiž možné, že neboli dostatočne nepriepustné cesty aby sa zmenila trasa.

Cieľom ďalšieho pokusu, je pozorovať ako často, prípadne ako rýchlo dôjde k zápche na mape *test1.xml*, ak agenti používajú rôzne typy algoritmov.

V prvom pokuse sme nechali pohybovať sa po mape iba agentov používajúcich algoritmus pre plánovanie trasy najkratšou cestou, *NajkratsiaCestaAlgoritmus*. Rovnakým postupom, ako sme popísali pred chvíľou, spustíme server

simulácie a Agent managera, pričom budeme generovať náhodne agentov s algoritmom typu *NajkratsiaCestaAlgoritmus*. V simulácii je pekne vidieť, že pri nárazových prívaloch agentov na križovatku začnú vznikať zápchy, ktoré sa len ťažko strácajú (zmenšujú). Je to spôsobené hlavne, tým že prichádzajú stále nový a nový agenti do miesta kde je zápcha.

Zápchy v tomto pokuse vznikajú už v začiatkoch, rádovo niekoľko desiatok sekúnd serverového času.

Pozrime sa, ako to dopadne v druhom pokuse, keď budeme generovať len agentov používajúci algoritmus *StredneRýchleCestyAlgoritmus*. Zápchy sa budú aj naďalej objavovať, s tým rozdielom, že ich trvanie je väčšinou krátkodobé. Agenti sa dokážu vysporiadať s faktom že, niekde je cesta upchaná. Volia iné trasy, tým sa zmenší počet agentov, ktorí smerujú do nej, čím sa urýchli proces odbúrania dopravnej zápchy. Ale časom sa môže stať, že sa cesty pomerne dosť upchajú, pretože ich kapacita je malá. Tento stav môže nastať až po niekoľkých minútach serverového času.

Najzaujímavejší výsledok ponúka tretí pokus, v ktorom sa pohybujú agenti používajúci len algoritmus *NajrychlejsieCestyAlgoritmus*. K dopravným zápcham dochádza zriedkavo, a ak niekde vznikne, k jej odbúraniu príde ešte skôr, ako keby sme použili iný typ algoritmu. Avšak podobne ako v predošlom pokuse, aj tu sa časom stane, že počet agentov na cestách presiahne kapacity ciest a dôjde k spomalenie premávky, len zriedkavo k úplnému zastaveniu ako to môžeme vidieť v predošlých pokusoch.

Okrem spomenutých pokusov, som preveril aj možnosti rôznych kombinácií používaných algoritmov. Výsledky z nich hovoria, ako by to mohlo dopadnúť keby určitá skupina vodičov bola informovaná o stave premávky a iná nie.

Na základe pokusov, ktoré sme tu popísali, by sme mohli povedať, že tretí typ algoritmu je pre navrhnutý systém najlepší. Bola by to pravda, ak by sme predpokladali, že každý vodič by používal tento algoritmus. Bohužiaľ, pri nasadení v praxi, by to tak nebolo. Sila systému je daná počtom kooperujúcich jedincov (agentov). Je zrejmé, že pri nasadení, by si nie každý mohol dovoliť zaobstarať potrebné vybavenie. Otázkou teda je, ako zabezpečiť, aby systém fungoval aj pri malom množstve kooperujúcich agentov. Respektíve, ako získavať informácie o dopravných situáciách, ak je počet agentov malý. Dočasným riešením by mohlo byť používanie informácií napríklad s dopravných správ. V dnešnej dobe už existujú systémy, ktoré prijímajú vysielané dopravné informácie z FM rádií.

Tvrdenia, ktoré som vyslovil v tejto podkapitole sú podložitelné výsledkami testov, ktoré si dokáže každý overiť sám. Stačí, ak budete postupovať podľa pokynov, ktoré som popísal.

5.2 Ďalší vývoj

V práci bolo spomenuté, že pre pohyb agenta po mape potrebujeme dva grafy. Keďže bol projekt vyvíjaný vyše roka a niektoré časti boli naprogramované skôr, ako iné, vznikol tento problém. Nejde o žiadnu veľkú chybu, ktorá by

spôsobila neefektívne počítanie nových pozícii agentov, práve naopak. Vďaka tomu je výpočet celkom jednoduchý a priamočiary. Problémom je to, že sú grafy dva. Pri ďalšom vývoji, by bolo vhodné zamyslieť sa nad tým, ako by sa dal použiť iba jeden z uvedených grafov. Prípadne navrhnúť nový, ktorý by vyhovoval pre dané potreby.

V úvode kapitoly sme spomenuli možnosť skúmania dopravných sietí a navrhovania ich zmien. Súčasný editor máp je schopný, ale nie efektívne, vytvárať mapy podľa predlohy skutočnej cestnej infraštruktúry. Editor má možnosť nastaviť si pozadie. Pozadím by mal byť napríklad obrázok dopravnej siete mesta. Vďaka takejto predlohe vie užívateľ pomerne presne napodobniť daný typ siete. Nevýhoda je, že editor dokáže kresliť len cesty a križovatky, vďaka ktorým sa zostroja aj komplikovanejšie prvky siete. Návrhom do ďalšieho vývoja by bolo, implementovať ďalšie prvky dopravnej siete.

V tejto chvíli prebieha komunikácia medzi agentmi pomocou implementovanej nástenky. V ďalšom vývoji, by bolo celkom zaujímavé sa zamyslieť aj nad inými možnosťami. Napríklad navrhnutím dispečera, ktorý by sa dal použiť aj pri reálnom nasadení. Dispečer by zabezpečil priamu komunikáciu medzi dvoma, prípadne viacerými agentmi, ktorí by si vymieňali ľubovoľné informácie. Nástenka tu zabezpečuje iba nepriamu komunikáciu.

Odkazy na CD

[i] Preložené a spustitelné zdrojové súbory aplikácií
`CD:\Projekt`

[ii] Všetky zdrojové súbory
`CD:\Zdojaky`

[iii] Inštalačný súbor pre JAVA 6.0 JDK a nástroj pre logovanie textov
`CD:\Tools`

[iv] PDF súbor bakalárskej práce
`CD:\Text`

[v] Programátorská dokumentáciu vygenerovaná pomocou javadoc
`CD:\javadoc`

Literatúra

[1] Ing. Arnoštka Netrvalová: *Úvod do problematiky multiagentních systémů*

[2] V. Mařík, O. Štepánková, J. Lažanský a kol.: *Umělá inteligence 2*, pp. 143-173, 1997

[3] Brett Spell: *Java Programujeme profesionálne*, 2002

[4] Sun Microsystems, Inc.: *The Swing Tutorial*, 1995-2007
`http://java.sun.com/docs/books/tutorial/uiswing/`

[5] Sun Microsystems, Inc.: *Trail: RMI*, 1995-2007
`http://java.sun.com/docs/books/tutorial/rmi/index.html`