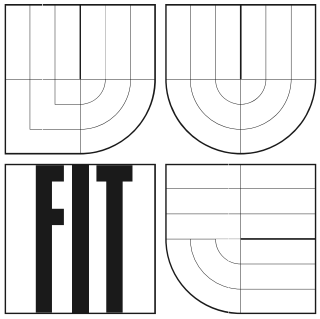
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Modelování dopravních systémů

Bakalářská práce

2007 Jiří Šmíd

Modelování dopravních systémů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Doc. Ing. Zdeny Rábové, CSc. a Dr. Ing. Petra Peringera.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Šmíd
17. 1. 2007

Abstrakt

Tato práce pojednává o modelování transportních a dopravních systémů. V první části popisuje rozšíření existující knihovny Simlib/C++ o podporu pro vytváření transportních systémů. Druhá část je věnována analýze celulárních automatů. Důraz je kladen zejména na to, aby modely dopravních systémů vytvořené pomocí této knihovny se co nejvíce přibližovaly svým reálným protějškům. Snadno modifikovatelný systém pro sběr a zpracování statistických dat umožňuje uživateli analyzovat nejen fundamentální statistická data, nýbrž i samotný pohyb vozidel v jednotlivých krocích. Vozidla různých rozměrů jsou schopna sama vyhledat cestu k cíli. Obě knihovny poskytují uživateli pro tvorbu modelů dopravních systémů maximální možnou volnost.

Klíčová slova

Simulace transportních a dopravních systémů, celulární automaty.

Abstract

The presented work describes both transportation and traffic flow simulation. For that purpose two different libraries were made. The first one is based on an existing library, Simlib/C++. For the traffic flow simulation a brand new library was created, using cellular automaton rules. Above all, it is designed to facilitate analysis of fundamental attributes of traffic network. Vehicles varying in size are able to find their own route plans from point where they were generated. Both libraries offer users a large amount of flexibility in specifying transport and traffic network.

Keywords

Transportation simulation, traffic flow simulation, cellular automata.

Obsah

Část A

Transportní systémy

1 Úvod

Transportní systém, jako specifický případ dopravního systému, je uplatňován zejména v případech kdy je naší prioritou efektivita a rychlost přepravy, než její negativní účinky na okolí. Typickým příkladem jsou výrobní linky a sklady podniků.

Snaha o zvýšení efektivity vede skrze úsporu místa, tedy minimalizaci nevyužitého prostoru. Tímto prostorem jsou myšleny především dopravní trasy, kde se dbá o jejich maximální vytíženost při postačující kapacitě přepravy.

Racionalizaci přepravních tras, zvláště uvažujeme-li o případném dalším rozšiřování výroby, není snadný úkol. Hlavním problémem při výpočtu jsou náhodné jevy, které mohou způsobit přechodné ucpání částí systému.

Řešením je vhodný návrh budoucího systému a jeho simulace pomocí prostředků výpočetní techniky. V případě, kdy pouze rozšiřujeme stávající systém, nebo vycházíme ze spolehlivých empirických poznatků již existujícího obdobného systému, dostává se nám do rukou možnost odhadnout velmi přesně nejen celkové náklady na výstavbu, ale můžeme již v době návrhu objevit skrytá úskalí námi zvoleného řešení.

Cílem je rozšířit knihovnu Simlib/C++ tak, aby s výsledným produktem bylo možné snadno vytvořit model transportního systému. Nabyté zkušenosti by měly posloužit jako vodítko při návrhu vlastní knihovny v druhé části této práce.

2 Návrh

Rozšíření objektově navržené knihovny spočívá v obohacení potomků existujících tříd o nové vlastnosti a funkcionalitu. To zda vůbec bude možné takové rozšíření provést závisí zejména na kvalitě návrhu rozšiřované knihovny.

2.1 Diskrétní simulace

Podstatou tohoto přístupu je rozdělení simulačního času na stejně dlouhé úseky. Každý element může za daný časový úsek provést jednu nebo určitý počet operací. Tyto operace, neboli události, jako jediné mohou měnit stav systému. Diskrétní systémy lze rozdělit na *deterministické* a *stochastické*. Přičemž pod označením diskrétní simulace, *discrete event simulation*, se myslí diskrétní stochastická simulace. Ta se od diskrétní liší v tom, že provádí výpočty s určitým pravděpodobnostním rozložením, získaným empirickým výzkumem. Typickým představitelem diskrétních stochastických systémů jsou *systémy hromadné obsluhy*.

2.2 Systémy hromadné obsluhy

Systémy hromadné obsluhy (SHO) jsou zařízení, poskytující služby příchozím procesům. Procesy, které nemohou být obslouženy mohou obsadit místo ve frontě, kde čekají až na ně přijde řada. V našem případě je SHO sklad, který poskytuje dopravníkům náklad, nebo kam může být náklad složen. Procesy jsou jednotlivé dopravníky, které vznášejí své požadavky vůči skladům.

Fronta vzniká v okamžiku, kdy o službu žádá více procesů, než je zařízení schopno obsloužit. Každou frontu lze charakterizovat *maximální délkou* a *frontovým řádem*, způsobem jakým jsou elementy do fronty vkládány a poté vybírány. Nejběžnějšími jsou *řádný frontový řád* (FIFO) a *inverzní frontový řád* (LIFO). Jsou-li požadavky řazeny do fronty tak jak přicházejí, jedná se o FIFO, fronta typu LIFO funguje opačně. K dalším zajímavým typům fronty patří *fronty s časovačem*, kdy požadavek nějakou dobu čeká a potom vykoná předdefinovanou činnost a *fronty s prioritou*, kdy jsou jednotlivé požadavky obsluhovány v pořadí podle své priority.

Mluvíme-li o procesech, máme většinou na mysli *Markovovy procesy*, někdy též nazývané náhodné procesy bez následných účinků. Markovovův proces s diskrétním časem a stavy bývá označován jako *Markovovův řetězec*.

2.3 Simlib/C++

Simulační knihovna byla vyvinuta na ústavu Informatiky a výpočetní techniky FEI VUT Brno v roce 1990. Její další rozšíření probíhají dodnes. Umožňuje efektivní popis simulačních modelů přímo v jazyce C++ a to pod operačními systémy Windows a Linux.

S využitím Simlib/C++ je možné provádět diskrétní i spojitou simulaci. Pro účely této práce se zmíním blíže o diskrétní části, kterou jsem přímo využil. Pro bližší informace odkazuji na kompletní dokumentaci [4].

2.3.1 Události a procesy

Každá diskrétní simulace může být v zásadě popsána pomocí událostí a procesů. Událost lze v podstatě reprezentovat pomocí obyčejné metody s modelovou dobou trvání rovnou nule, tedy proběhne v tom samém okamžiku, kdy byly vyvolána. Proces, přesněji *transakce*, probíhá v modelovém čase po delší dobu a je tedy nutné zajistit jistou formu *kvaziparalelismu* pro více takovýchto procesů, spuštěných v jednom okamžiku. Pro další výklad je nutné rozlišovat mezi *reálným časem* a *časem modelovým*, jehož jeden úsek může trvat v čase reálném libovolně dlouho. V Simlib/C++ a obdobných systémech je kvaziparalelismus řízen pomocí *kalendáře*, struktury podobné seznamu. Každá nová událost je registrována v kalendáři. Ten je seřazen podle modelového času kdy mají být registrované události provedeny a řídí simulaci tím, že je v pravý okamžik spouští.

2.3.2 Systémy hromadné obsluhy (SHO)

Jedná se o systémy, poskytující obsluhu transakcím. Je-li to třeba, čekají transakce ve frontách. V případě stochastického diskrétního modelu jsou příchozí transakce generovány s určitým typem pravděpodobnostního rozložení. To je dáno povahou modelu, ale zpravidla se pro střední dobu mezi přechody používá *exponenciální rozložení* a pro počet příchodů za jednotku času *rozložení Poissonovo*.

2.4 Požadavky na dopravníky

Pro vytvoření jakékoli simulace využívající dopravníky budeme potřebovat několik základních komponent. Logicky se nabízejí samotné dopravníky, tedy prostředky pro přepravu nákladu z jednoho místa na druhé. Ty mohou nabývat různých podob, v zásadě je však můžeme roztřídit do dvou typů, dopravníky *volné* a *pásové*.

2.4.1 Dopravníky volné a pásové

Volné dopravníky reprezentují nejrůznější vozíky s vlastním nebo jiným pohonem, mohou být řízeny obsluhou nebo se pohybovat v předem nastavených drahách a být pouze kontrolovány řídící jednotkou. Právě podle volnosti pohybu lze tyto dopravníky dále rozdělit na ty, které nejsou umístěny ve vodící dráze, *volné dopravníky schopné předjíždění* (dopravníky VSP), a na *dopravníky neschopné předjíždění* (dopravníky VNP).

Oba typy disponují několika společnými vlastnostmi. Pro přepravu nákladu je nutné znát maximální možné naložení dopravníku a velikost jeho aktuálního nákladu v *přepravních jednotkách* [PJ]. Přepravní jednotkou je myšlena atomická veličina, kterou si stanoví uživatel až při tvorbě simulace. Může jí být například jeden balík, jeden kilogram, jeden výrobek atd. Důležité je, aby pro celý sledovaný systém, nebo jeho uzavřenou část existovala pouze jedna definice přepravní jednotky.

Další význačnou vlastností je maximální rychlost dopravníku, vyjádřená v jednotkách délky na jednotku času, opět v závislosti na konkrétní simulaci a plně pod kontrolou uživatele. Spolu s rostoucím naložením může klesat maximální rychlost dopravníku, což by v návrhu simulačního modelu mělo být rovněž zohledněno. Poslední důležitou vlastností dopravníku je čas na naložení jedné přepravní jednotky.

Díky tomu, že se jedná o otevřenou knihovnu, je uživateli umožněno pomocí dědičnosti snadno přidávat nové vlastnosti. Příkladem může být minimální náklad, bez kterého dopravník neopustí nákladovou rampu. Tato poměrně užitečná vlastnost byla nakonec přidána již do základního návrhu.

VNP musejí být obohaceny nástrojem, který zabrání dvěma nebo více dopravníkům, pohybujícím se po jedné dráze, vzájemné předjíždění. Tuto funkci by mohl splňovat kontroler, který

by udržoval informaci o jednotlivých drahách, po kterých se tento typ dopravníků v daném okamžiku pohybuje. Určující informací by byl čas dopravníku, který tuto dráhu opustí jako poslední. Žádný nově příchozí by potom nebyl schopen tuto dráhu opustit dříve.

Pásové dopravníky představují automatické přepravní mechanismy opatřené zařízeními pro manipulaci s přepravovanými produkty. V praxi se s nimi setkáváme například v podobě obřích rypadel v povrchových dolech, automatických pásových přepravníků ve výrobních podnicích, plnicích přístrojích v potravinářském průmyslu, balicích linkách atp. Tento typ přepravy zpravidla neposkytuje prostor pro náhodnost a velice přesných výsledků lze dosáhnou jednoduchými matematickými výpočty. Z tohoto důvodu jsem se rozhodl tento typ dopravníků do mnou vytvářené knihovny nezahrnovat.

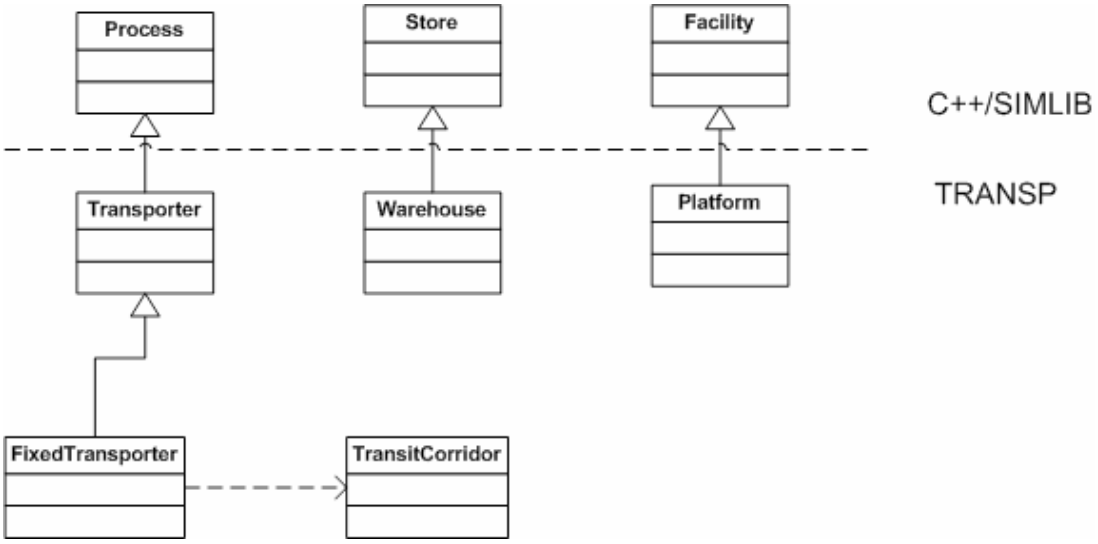
2.4.2 Nákladové a vykládkové prostory

Místa, kde je přepravovaný materiál nakládán a vykládán, tvoří uzlové body, mezi kterými se dopravníky cyklicky pohybují. Klíčovým údajem, který je třeba uchovávat a průběžně aktualizovat, je množství přepravovaných jednotek nákladu.

Obslužná jednotka musí být schopna řadit příchozí dopravníky do fronty, v případech kdy nebude schopna je okamžitě obsloužit. Současně by měla umět obsluhovat několik požadavků najednou, přičemž uživatel bude mít možnost nastavit maximální počet současně obsluhovaných dopravníků.

3 Implementace

Realizace návrhu pomocí knihovny Simlib/C++ byla provedena rozšířením stávajících tříd `Process`, `Facility` a `Store`. Pro aktivní prvky s vlastním chování existuje v Simlib/C++ třída `Process`. Pomocí její virtuální metody `Behavior()` lze snadno jednoduchými příkazy popsat chování aktivního prvku. Aktivními prvky jsou v našem případě dopravníky všech typů. Třídy `Facility` a `Store` představují obslužná zařízení pro procesy. Zařízení typu `Facility` pro obsluhu procesů používá tzv. *výlučný přístup*, tedy že v jednom okamžiku může být obsluhován pouze jeden proces. Tím se liší od zařízení typu `Store`, které může v jednom okamžiku obsluhovat více procesů, v závislosti na své kapacitě a požadavcích na kapacitu ze strany procesů. Pokud proces požaduje větší kapacitu než jaká je momentálně volná, musí počkat dokud nedojde k jejímu uvolnění. Mimoto není zařízení typu `Store` schopné, narozdíl od `Facility`, přerušovat obsluhu procesů. Z třídy `Facility` vychází třída `Platform`, zatímco třída `Store` je předkem třídy `Warehouse`.



Obrázek 2.4.1 - Diagram tříd knihovny Transp

3.1 Třída Transporter

Reprezentuje dopravníky VSP. Vychází z třídy `Process`, kterou obohacuje o vlastnosti dopravníku, tak jak byly definovány v návrhu. Jedná se o kapacitu, aktuální naložení, maximální rychlost, úbytek rychlosti na jednotku nákladu a čas na naložení jednotky nákladu.

Dále bylo třeba implementovat výpočet času, který transportér potřebuje na překonání určité vzdálenosti, v závislosti na jeho nákladu. Ten je nepřímo úměrný maximální rychlosti dopravníku a může růst spolu se zvyšujícím se naložením.

Metody pro naložení a vyložení nákladu obsadí dané zařízení po určitou dobu. Ta je dána množstvím nakládaných nebo vykládaných přepravních jednotek. K zajištění náhodnosti jsou vypočtené hodnoty uvnitř všech tří metod použity jako základ pro generátor pseudonáhodných čísel exponenciálního rozložení.

V případě dopravníků neschopných předjíždění bylo nutné pozměnit způsob výpočtu doby potřebné pro absolvování trasy. Třída `FixedTrasporter` je tedy potomkem třídy `Trasporter`. Dopravník neschopný předjíždění se smí pohybovat pouze po určitých trasách, přičemž pokaždé musí být jasné, po které trase se právě pohybuje, aby bylo možné ověřit, zda na této trase není blokován jiným dopravníkem. Po trasách, po nichž se pohybují dopravníky VNP se samozřejmě mohou pohybovat i jejich protějšky. Obecně se předpokládá, že pokud je dopravník schopný nějakou překážku objet, je také schopný se jí vyhnout, nebo jí uvolnit průjezd, a proto není nutné takovéto dopravníky explicitně do výpočtu doby průjezdu zahrnovat. Může se ovšem stát, že dopravník z nějakého důvodu není schopen svému rychlejšímu kolegovi dráhu uvolnit. Potom lze tento

dopravník vytvořit, jako instanci třídy `FixedTransporter`, přičemž na úsecích, kde je předjíždění povoleno, lze volat metodu pro výpočet doby pohybu z třídy předka, `Trasporter`.

Přehled o tom, kdy která trasa bude uvolněna spravuje třída `TransportCorridor`. Pokaždé, když dopravník vjede na dráhu označenou identifikátorem, což je kladné celé číslo, dojde k aktualizaci časového údaje. Žádný dopravník, který není schopen předjíždění nemůže takovou dráhu opustit dříve než udává časový údaj nastavený předchozím vozidlem.

3.2 Třídy Platform a Warehouse

Účelem těchto tříd je obsluha dopravníků. Vzhledem k tomu, že samotný proces nakládání a vykládání ošetřují dopravníky, zde stačí pouze aktualizovat počet přepravních jednotek, které jsou právě „na skladě“.

Pro dopravníky, které nemohou být právě obslouženy, zdědily obě třídy od svých předků fronty, v kterých dopravníky mohou čekat. Tato problematika je podrobně rozebrána v předchozích kapitolách a zde již není třeba se k ní dále vracet. Taktéž rozdíly mezi těmito dvěma třídami již byly objasněny.

4 Testování

Jednotlivé testy tohoto bloku jsou rozděleny do dvou částí. Testy, které prověřují základní funkčnost objektů jednotlivých tříd a testy, které mají předvést schopnosti vytvořené knihovny, případně ukázat možnosti dalších rozšíření.

4.1 Ověření funkčnosti

Pro ověření základní funkčnosti jsem vytvořil několik jednoduchých modelů. Každý z nich je zaměřen na jednu konkrétní schopnost. Ve všech příkladech se dopravníky pohybují mezi třemi sklady, dva jsou typu `Platform`, jeden typu `Warehouse.` Sklad typu `Warehouse` je schopen najednou obsluhovat tři dopravníky. Dopravníky převážející náklad mají nastavenou úroveň minimálního naložení rovnou jejich kapacitě, polovině kapacity a poslední typ nemá minimální naložení nastaveno vůbec. Celkový součet přepravních jednotek se rovná osmdesáti procentům maximálního naložení všech dopravníků.

K ověření činnosti je zaznamenáván počet kusů přepraveného nákladu. Pro sběr těchto dat bylo možno využít některou ze statistických tříd Simlib/C++, ale nakonec jsem se rozhodl vytvořit vlastní kolektor počtu přepraveného nákladu. Ten nejenom shromažďuje potřebná data, ale patřičně zajišťuje i jejich výstup. Chtěl jsem tím poukázat zejména na fakt, kdy se uživatel nemusí spoléhat při vytváření modelu pouze na knihovní třídy a jejich rozšiřování, nýbrž může pro svůj model svobodně

vytvořit třídy vlastní. Ty mohou plnit různé speciální úkoly, na které při tvorbě knihovny nemohlo být pamatováno. Tato třída `TSum` je implementována jako singleton, tedy třída z níž lze vytvořit pouze jedinou instanci.

Série testů se skládá ze tří částí. V první jsou testovány VSP dopravníky, v druhé pouze dopravníky VNP a konečně v třetí části oba typy dopravníků, přičemž na jedné z drah je zakázáno předjíždění. Dalším krokem je vytvoření potomků třídy `Transporter` s definicí metody `Behavior()`. Ta popisuje chování objektu této třídy, typicky odkud a kam se bude pohybovat, kde bude nakládat a vykládat zboží a podobně.

V druhém příkladu je třída `Transporter` pochopitelně nahrazena třídou `FixedTransporter`. Třetí test je zajímavý tím, že řeší možnost, kdy by uživatel požadoval aby se VSP dopravníky v určitém úseku chovali jako VNP dopravníky. V tomto případě je toto chování způsobeno tím, že v dané dráze se pohybují dopravníky VNP i VSP, přičemž VSP dopravníky nemají možnost své protějšky předjet.

Řešením tohoto problému je vytvořit veškeré dopravníky jako VNP, přičemž v úsecích, kde je předjíždění povoleno je zavolána metoda pro výpočet času nutného k překonání dané vzdálenosti z třídy `Transporter`, nikoli `FixedTransporter`. To je možné díky tomu, že třída `FixedTransporter` je potomkem třídy `Transporter`, tudíž dědí všechny metody svého rodiče, včetně této.

Následuje výkonná část programu, kdy ve funkci `main()` je nejprve inicializována simulace potom a v tomto případě ještě sčítačka přepraveného zboží. Potom jsou zavolány konstruktory dopravníků a metodou `Run()` je celá simulace spuštěna. Na konci metoda sčítačky `fDump()` vypíše výsledek testu.

Pro účely ladění jsou třídy `Transporter` i `FixedTransporter` opatřeny kontrolními výpisy o tom, jak dlouho se daný dopravník zdržel ve skladu a jak rychle překonal úseky mezi sklady. Ladící výpisy lze zapnout definicí makra `__DEBUG__`.

4.2 Ukázkový příklad využití knihovny

Cílem tohoto testu je zjistit závislost kapacity dopravníku na efektivitě přepravy. Efektivita je dána podílem celkového přepraveného nákladu (v PJ) a celkové vzdálenosti, kterou dopravníky urazily. Dopravníky se pohybují uvnitř uzavřeného systému s neměnným počtem přepravních jednotek. Proto nás také zajímá, jakou průměrnou dobu budou dopravníky čekat na naložení.

Jedná se opět o tři sklady, dva typu `Platform`, jeden typu `Warehouse`. Sklad typu `Warehouse` je schopen najednou obsluhovat tři dopravníky. Každý ze skladů na počátku obsahuje stejné množství přepravních jednotek. Celkový součet přepravních jednotek se rovná pěti stům kusů. Na trase mezi dvěma sklady není možné předjíždění.

Majitel skladu se rozhoduje mezi koupí několika typů dopravníků, lišících se svoji maximální kapacitou, přičemž požaduje aby minimální naložení bylo vždy rovno sedmi desetinám maximální kapacity. Zároveň požaduje aby za půl hodiny bylo mezi sklady přepraveno nejméně 500 přepravních jednotek nákladu při co nejnižší vzdálenosti, kterou by dopravníky musely ujet. Jednotlivé dopravníky nesmí čekat na naložení v průměru déle než pět minut.

Test je třeba provést v několika krocích, v každém kroku je nutné zaznamenat počet přepravených jednotek a celkovou vzdálenost, kterou dopravníky během simulace urazily a čas který strávily čekáním na minimální naložení. Maximální kapacity dopravníků jsou deset, patnáct, dvacet a dvacetpět přepravních jednotek.

Rozšířením třídy pro sběr statistických dat, z předchozích příkladů, byla získána data zobrazená v tabulce 4.2.1. Nově bylo využito zjištění simulačního času a jeho zpracování pro získání průměrné doby, kterou dopravníky čekají na naložení.

Na základě výsledků měření, požadavkům nejvíce vyhovuje dopravník s kapacitou dvacetipěti přepravních jednotek nákladu. Tento dopravník dokázal v daném termínu přepravit o patnáct procent více nákladu než bylo požadováno, přitom z porovnání se zbylými dopravníky vyšel jako nejefektivnější. Průměrná doba čekání jednoho dopravníku, tohoto typu, na naložení je rovněž v normě.

Tento test je víceméně cvičný. Jeho úkolem bylo hlavně naznačit možnosti knihovny Transp, respektive Simlib/C++, která díky kvalitnímu návrhu umožňuje uživateli, s využitím standardních postupů objektově orientovaného návrhu, postupovat naprosto svobodně při tvorbě simulačních modelů.

		Maximální kapacita dopravníků [PJ]			
		10	**15**	**20**	**25**
přepraveno	[PJ]	550	525	580	575
	%	110	105	116	115
vzdálenost	[m]	29160	18600	15360	12240
	%	190	121	100	80
doba čekání	[s]	270	238	344	278
	%	90	80	115	93

Tabulka 4.2.1 - množství přepraveného nákladu a vzdálenost

5 Závěr

Využití existující knihovny s sebou nese nutnost strávit určitý čas, někdy poměrně významný, jejím studiem. Teprve poté je možné přistoupit k samotnému řešení problému. Část věnovaná diskrétní simulaci v prostředí Simlib/C++ je snadno pochopitelná a je jí tudíž možno zvládnout poměrně rychle.

Při návrhu tříd pro simulaci dopravníků, jsem se snažil postupovat v podobném duchu. Výsledkem je několik tříd plně splňujících výchozí požadavky. Navíc mohou v budoucnu bez jakýchkoli úprav posloužit jako základ pro další možná rozšíření.

Část B

Dopravní systémy

1 Úvod

Modelovat dopravní systém není v žádném ohledu snadné. Na jedné straně chtějí mít lidé k dopravě snadný přístup, na straně druhé jsou obtěžováni především její hlučností i dalšími škodlivými projevy. Tento rozpor je bez výpočetní techniky často neřešitelný, a to zejména v nejhustěji zalidněných aglomeracích.

Výpočetní systémy, určené k řešení takto komplexních problémů, mají svůj všeobecně přejatý anglický název, *multi-agent simulations* (MAS), který vyjadřuje fakt, že každá entita simulace je spravována samostatně a má svá vlastní pravidla, na základě kterých se rozhoduje.

Má-li simulace plnit svůj účel je bezpodmínečně nutné se co možná nejvíce přiblížit realitě. To především znamená zahrnout do ní empirické poznatky získané dlouhodobým výzkumem chování účastníků dopravního provozu i teoretické poznatky, o kterých bude řeč později.

Hlavním úkolem simulace dopravních systémů je poskytnout uživateli přehled o hustotě dopravy v jednotlivých částech systému, průměrné rychlosti vozidel a plynulosti dopravy v případě havarijních stavů. Z takto získaných dat lze poté snadno odvodit lokální překročení úrovně hluku, množství exhalací, a dalších škodlivých vlivů. Tímto způsobem je možné řešit problémy dříve než vzniknou a ušetřit tak významné finanční prostředky.

2 Návrh

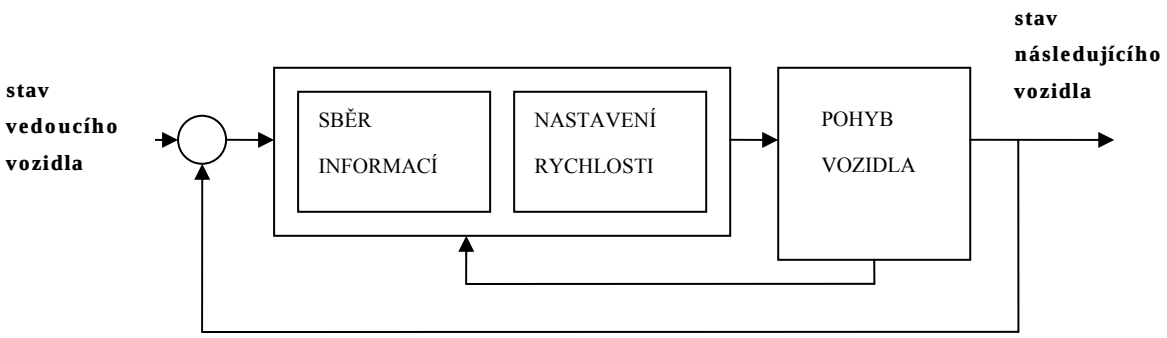
Pro úspěšnou realizaci systému simulace dopravy je nezbytné vyjít z již prověřených poznatků, shrnutých do několika fundamentálních teorií. Podrobněji se zmíním pouze o těch, které byly využity při tvorbě této práce.

2.1 Car-following theory

Tato teorie je naprosto nezbytná pro tvorbu jakéhokoli dopravního simulačního systému. Je založena na dvou jednoduchých předpokladech. Každý z účastníků dopravního provozu, bývá obvykle nazýván *agent*, se snaží udržet své vozidlo v chodu, směrem určeným jeho cestovním plánem a zároveň zabránit kolizi s ostatními agenty.

Představíme-li si simulační model jako jedinou dopravní komunikaci o jednom pruhu, kde se jednotlivá vozidla mohou pohybovat pouze směrem kupředu, je výchozím agentem ten, který je nejbližší konci této komunikace. Takovému agentu se říká *vůdce*. Každý další agent je jeho následníkem a musí tudíž do svého rozhodování zahrnout stav, v jakém se nachází vůdce. Jsou-li jeho

vstupní informace vyhodnoceny tak, že je bezpečné se přesunout vpřed, provede pohyb. Totéž je provedeno pro každého ze zbývajících agentů.



Obrázek 2.1.1: Blokový diagram CFT

2.2 Úroveň detailu

Na simulaci můžeme pohlížet dvěma, respektive třemi, způsoby. Jedná o *makroskopický* pohled, který je velmi obecný. Nezabýváme se jednotlivými agenty, nýbrž vytváříme pravidla simulující model jako celek. Výpočetní čas takovéto simulace je potom velmi rychlý, avšak pravidla, která by celý systém přibližovala realitě, může být velmi obtížné stanovit.

Mikroskopický model přistupuje k simulaci dopravního systému skrze jednotlivé agenty. Je snadno implementovatelný a blízký chování skutečných dopravních systémů. Jeho nevýhodou je dlouhá výpočetní doba a z toho plynoucí nemožnost dostatečně detailně modelovat rozsáhlé systémy na běžně dostupných počítačích.

Na půl cesty mezi mikroskopickým a makroskopickým pohledem na simulaci je *mesoskopický* model. Podobně jako mikroskopický model je založen na interakci agentů, ovšem tito postrádají individuální charakteristiky a nejsou schopni samostatného rozhodování.

2.3 Objektově orientovaný návrh

Nejjednodušším způsobem tvorby návrhu, je vyjít ze skutečných entit reálného systému, definovat jejich vlastnosti, způsob chování a jejich vzájemné vztahy. Objektově orientované paradigma spočívá v definici objektů, korespondujících se svými skutečnými předlohami, pomocí datových typů a struktur popsat jejich vlastnosti a nakonec vytvořit metody, které představují chování těchto objektů samotných i mezi sebou navzájem, pomocí zasílání a přijímání zpráv. Přenos zpráv bývá realizován pomocí funkčních volání. V krátkosti se zmíním o nejdůležitějších pojmech OOP.

Třída, je uživatelem vytvořený datový typ, definující svá vlastní data a funkcionalitu. Třída by měla být uzavřená, tj. obsahuje všechny podstatné atributy vzorové entity. Abstrahování základních vlastností, nutných k řešení problému se nazývá *abstrakce*.

Jak jsem zmínil výše, třída je datovým typem, tudíž musí existovat možnost vytvořit proměnné tohoto typu. Taková možnost samozřejmě existuje a proměnné se označují jako *instance třídy*, Například uvažujeme-li třídu pes, potom instancí je konkrétní zvíře, třeba sousedův Alík.

Ne vždy je žádoucí, aby měl uživatel přístup ke všem metodám natož pak k atributům. Důvodem jsou nejčastěji neautorizované změny, které mohou vyvolat neočekávané chování aplikace s takto poškozenou instancí. Mimoto viditelné oddělení soukromých metod od veřejných usnadní uživateli orientaci při využívání nebo rozšiřování našeho produktu. Transparentní přístup uživatele k existující třídě skrze rozhraní je podstatou *zapouzdření*.

Dědičnost, další klíčový pojem OOP, umožňuje rozšiřovat rodičovské třídy o potomky, přičemž tito potomci dědí všechny metody a atributy svého rodiče. Existují dva druhy dědičnosti veřejná a soukromá. Vztah mezi potomkem a rodičem při veřejné dědičnosti definovat slovesem „je“, foxteriér je druh (rasa) psa. Veřejnou dědičností tedy dochází ke *specializaci* rodičovské třídy. Naproti tomu soukromou dědičnost lze vyjádřit slovem „má“, chovatel má jednoho nebo několik psů. Tento vztah se označuje jako *kompozice*. Pomocí dědičnosti lze jeden problém řešit s různou mírou jemnosti pohledu, s různou mírou abstrakce o níž jsem se zmínil na začátku kapitoly.

S dědičností je úzce svázán *polymorfismus*. V podstatě se jedná o různé chování stejné metody, v závislosti na třídě, z níž je volána. Za polymorfismus se také považuje přetěžování funkcí, kdy se chování přetížené funkce odvíjí podle kombinace vstupních parametrů.

2.4 Návrh systému

Pohledem na reálný dopravní systém můžeme jednoduše odvodit hned několik komponent budoucího modelu. Tento postup je pro objektově orientovaný návrh naprosto typický. Nejprve uvedu základní stavební bloky dopravní simulace a kapitolu zakončím návrhem řídící komponenty.

2.4.1 Celulární automaty

Silniční síť je především složena z jednotlivých dopravních komunikací, které budeme nazývat *hrany*. Jelikož by vytvářená knihovna měla poskytovat podklady pro vytvoření diskrétní simulace, je nutné každou hranu rozdělit na menší a stejně velké části, *buňky*. Každý agent se potom za jednu časovou jednotku smí přesunout o určitý počet těchto buněk. Současně platí, že každá buňka smí být obsazena nejvýše jedním agentem.

Takové systémy označujeme *celulární automaty*. Délka každé buňky by měla být větší než běžná délka agenta. Literaturou bývají doporučovány délky buněk pět nebo sedm a půl metru.

Definování časové jednotky jedna sekunda a délky buňky pět metrů, odpovídá rychlosti 18km/h. Rychlost tří buněk za jednu časovou jednotku je blízká maximální povolené rychlosti v obci a rychlost pěti buněk za časovou jednotku je rovna nejvyšší povolené rychlosti mimo obec.

2.4.2 Základní charakteristiky celulárního automatu

Pro popis dopravního systému, ať už reálného nebo modelu, jsou nezbytné tři základní veličiny. Tok dopravy, hustota dopravy a rychlost. Jejich přesná definice a pochopení vzájemných vztahů jsou nutné pro ověření, nakolik je vytvořený model blízký reálnému systému. K tomuto porovnání slouží tzv. *fundamentální diagramy*, o kterých bude řeč později.

- **průměrná rychlost**

  nebo také cestovní rychlost, lze ji snadno získat podílem součtu rychlostí všech vozidel, která jsou v daný okamžik přítomna na hraně s jejich počtem.

$$\upsilon_L = \frac{1}{N_L}\sum_{i=1}^{N_L}\upsilon_i \qquad \textbf{(2.4.1)}$$

- **okamžitá rychlost**

  na rozdíl od průměrné rychlosti je měřena pouze v určitém úseku. Jsou sčítány okamžité rychlosti každého projíždějícího vozidla a na konci měření je výsledek podělen počtem vozidel, která daným úsekem po dobu běhu simulace projela.

$$\upsilon_T = \frac{1}{N_T}\sum\upsilon_i \qquad \textbf{(2.4.2)}$$

- **okamžitý tok dopravy**

  je jednoduše součet vozidel, která projela měřeným úsekem. Tok dopravy bývá někdy též nazýván propustnost. Výsledek bývá podělen počtem pruhů, na kterých měření probíhalo. Obvyklou jednotkou toku dopravy je počet vozidel za hodinu a pruh.

$$q_T = \frac{N_T}{TN_{lanes}} \qquad \textbf{(2.4.3)}$$

- **průměrný tok dopravy**

  není možné jej měřit přímo, ale lze jej odvodit ze vztahu $q = \rho\upsilon$, potom dostáváme

$$q_L = \rho_L\upsilon_L = \frac{1}{LN_{lanes}}\sum_{i=1}^{N_{veh}}\upsilon_i \qquad \textbf{(2.4.4)}$$

- **průměrná hustota dopravy**

  je rovna podílu součtu všech vozidel na hraně a její délky. Výsledek se, podobně jako u toku dopravy, ještě dělí počtem pruhů.

$$\rho_L = \frac{N_{veh}}{LN_{lanes}} \tag{2.4.5}$$

2.4.3 Agenti

Nezbytnou součástí i nejjednoduššího modelu jsou *agenti*. Za tímto poněkud tajemným označením se skrývají dynamické prvky simulace, typicky vozidla, termín vozidla budu nadále používat. Jednotlivá vozidla se mohou lišit rozměry, rychlostí i dalšími charakteristikami, ale jedno mají společné, jsou to *pravidla*, za kterých mohou provádět svůj pohyb, viz. dále.

Po nastavení rychlosti vozidel následuje samotný pohyb všech agentů. Pro zajištění paralelního zpracování je nutné přesně dodržet postup, který bude popsán v následující kapitole. V okamžiku, kdy je přesunut poslední agent a to pouze jednou, je ukončeno jedno kolo simulace. To může reprezentovat jednu nebo několik sekund reálného času.Již v průběhu návrhu modelu dopravního systému musíme určit, v kterých místech budou agenti do simulace vstupovat. Přesně k tomuto účelu slouží speciální typy hran, tzv. *generátory*. Ty pracují naprosto stejně jako hrany, ale navíc obsahují struktury a mechanismy, které jim umožňují generovat agenty, uchovávat je a v pravý okamžik vpouštět do systému.

Jednotliví agenti by měli být schopni, jen na základě znalosti cíle své cesty, samostatně vyhledat trasu z místa svého vstupu do systému. Kritériem pro volbu z několika nalezených tras bývá standardně rychlost. Na automatické vyhledávání tras navazuje systém samostatného rozhodování [3].

2.4.3.1 Reakční doba

V simulačních modelech dopravních systémů se zpravidla počítá, že jednotliví agenti se budou snažit zabránit nehodě. Což koneckonců odpovídá reálnému systému. Doba, kterou potřebuje řidič k tomu, aby včas zareagoval na změněné podmínky bývá nazývána *reakční doba*. Z praxe odvozená průměrná reakční doba řidiče je rovna jedné vteřině.

Uvažujeme-li o dvou vozidlech jedoucích za sebou, přizpůsobuje zadní řidič rychlost svého vozidla tak, aby v příští vteřině byla přední část jeho vozu v místě, kde se v tuto chvíli nachází zadní část prvního vozidla.

2.4.3.2 Deterministický model celulárního automatu

Jak bylo zmíněno výše, celulární automat sestává z pole buněk o délce ℓ, každá buňka může být buďto prázdná nebo obsazená jedním vozidlem. Rychlost vozidel se pohybuje v rozmezí od nuly do maximální rychlosti.

Pravidla pro změnu rychlosti vozidel uvnitř deterministického celulárního automatu byla definována v roce 1993 v práci, kterou publikovali Nagel a Hermann. Položíme-li *t* jako současný stav a *t + 1* jako stav následující, potom nastavíme tzv. bezpečnou rychlost, tedy rychlost, při které nedojde ke kolizi s vedoucím vozidlem jako

$$\upsilon_{safe} = \upsilon_{t+1} = \min\left[g, \upsilon_t + 1, \upsilon_{\max}\right] \tag{2.4.6}$$

kde *g* značí počet neobsazených buněk ve směru pohybu vozidla. Uvažujeme-li o rychlosti υ jako o počtu buněk, které vozidlo urazí za jednu vteřinu, potom lze vzdálenost, o kterou se vozidlo přesune odvodit jako

$$x_{t+1} = x_t + \upsilon_{t+1} \tag{2.4.7}$$

Dopravní situace může nabývat dvou stavů:

- **plynulá doprava**

  Všechna vozidla mají ve směru svého pohybu dostatek prostoru, aby se mohla pohybovat svojí maximální rychlostí. Z čehož lze odvodit průměrný tok dopravy jako

$$q_L = \rho_L \upsilon_{\max} \tag{2.4.8}$$

- **dopravní zácpa**

  V tomto případě jsou rozestupy mezi vozidly menší nebo rovny maximální rychlosti. Vozidla mají rychlost vždy rovnu vzdálenosti k nejbližšímu vozidlu ve směru pohybu. Průměrný tok dopravy je v tomto případě

$$q_L = 1 - \rho_L \tag{2.4.9}$$

  Na základě tohoto vztahu lze vypočítat maximální průměrný tok na hraně, jako

$$q_{\max} = \frac{\upsilon_{\max}}{\upsilon_{\max} + 1} \tag{2.4.10}$$

2.4.3.3 Stochastický model celulárního automatu

Rychlost jakou se bude vozidlo pohybovat je dále ovlivněna vnějšími vlivy. V reálném systému, je plynulý pohyb vozidla z velké části závislý na schopnostech řidiče. Existuje tedy jistá pravděpodobnost, že řidič neakceleruje dostatečně na to, aby mezi svým a vedoucím vozidlem udržel konstantní vzdálenost, nebo naopak přibrzdí až příliš, což opět zvětší rozestup mezi ním a vedoucím

vozidlem. Na základě empirických poznatků byla pravděpodobnost tohoto jevu stanovena na $p_{noise} = 0{,}2$, tedy s pravděpodobností p_{noise}

$$\upsilon_{t+1} = \max\left[\upsilon_{t+1} - 1\right] \tag{2.4.11}$$

2.4.3.4 Pravidlo zpomaleného startu pro stochastický model celulárního automatu

Pozorováním reálného dopravního systému, bylo zjištěno, že při pohybu vozidel dochází za určitých podmínek k silné hysterezi, tedy závislosti aktuálního stavu vozidel na stavu předchozím. Dopravní situace přechází z plynulé do dopravní zácpy, při průměrné hustotě dopravy kolem třiceti procent. Ovšem při snížení průměrné hustoty dopravy pod tuto hranici, nedochází k přechodu do stavu plynulé dopravy, nýbrž teprve při mnohem nižší průměrné hustotě.

Uvedená pravidla lze aplikovat následovně :

- jestliže ($\upsilon_t = 0$ AND $g_t \leq 1$) potom υ_{t+1} přiřaď 0
- jinak υ_{t+1} přiřaď rychlost běžným způsobem

2.4.3.5 Časově orientovaný celulární automat

Liší se od stochastického modelu, delší vzdáleností, na které vozidla zrychlují a zpomalují. Časově orientovaný celulární automat byl poprvé publikován v roce 1998 (Brilon a Wu), s úmyslem zvýšit realističnost klasického stochastického modelu.

Podstatou tohoto modelu je uplatnění pravděpodobnosti pro zrychlení nebo zpomalení vozidla na základě porovnání volného prostoru ve směru jízdy a aktuální rychlosti vozidla

- jestliže ($g > \upsilon \cdot \tau_H$) potom s pravděpodobností p_{AC}

$$\upsilon = \min\left[\upsilon + 1, \upsilon_{max}, g\right] \tag{2.4.12}$$

- jestliže ($g < \upsilon \cdot \tau_H$) potom s pravděpodobností p_{DC}

$$\upsilon = \max\left[\upsilon - 1, 0\right] \tag{2.4.13}$$

2.4.4 Křižovatky

Pomocí hrany a agentů lze vytvořit dopravní systém pouze o jedné hraně. Pro vytváření rozsáhlejších systémů budeme potřebovat něco, co by jednotlivé hrany spojilo a umožnilo agentům plynulý přechod z jedné na druhou. Tyto tzv. *uzly* korespondují s křižovatkami reálného systému, proto často obsahují řídící systém. V případě, že uzlem chce projet více agentů, je jim na základě nějakého klíče nebo mechanismu, přidělena priorita, načež jsou obslouženy ty s vyšší prioritou, zatímco zbylí musejí počkat do příštího kola.

Je vhodné, aby tento způsob přidělování priorit jednotlivým vozidlům vycházel z priority hrany, z které agent do uzlu vjíždí, a aby prioritu hrany mohl uživatel nastavit. Tím bude moci rozlišit komunikaci hlavní od vedlejší.

2.4.5 Statistická data a jejich sběr

K nejdůležitějším statistickým datům patří vztah tok-hustota. Tento vztah bývá často vyjádřen graficky a nelze bez něj provést porovnávání simulačního a reálného systému. Každé měření se provádí po dobu několika simulačních kroků, tedy více než jednoho, v oblasti jejíž délka je rovna maximální rychlosti agentů. Většinou se jedná o maximální povolenou rychlost na dané hraně, vyjádřenou v počtu buněk za kolo simulace, pokud ovšem není systém nastaven tak, že tato rychlost může být překračována.

To že budeme provádět měření pouze v prostoru omezeném maximální rychlostí výrazně zrychlí běh aplikace, při nepatrném zkreslení výstupních údajů. Což se týká zejména měření hustoty, která se nebude měnit tak jemně, jako by tomu bylo při sběru údajů z celé délky hrany. Navíc se projeví určité zpoždění dané místem na hraně, v kterém bude probíhat sběr dat.

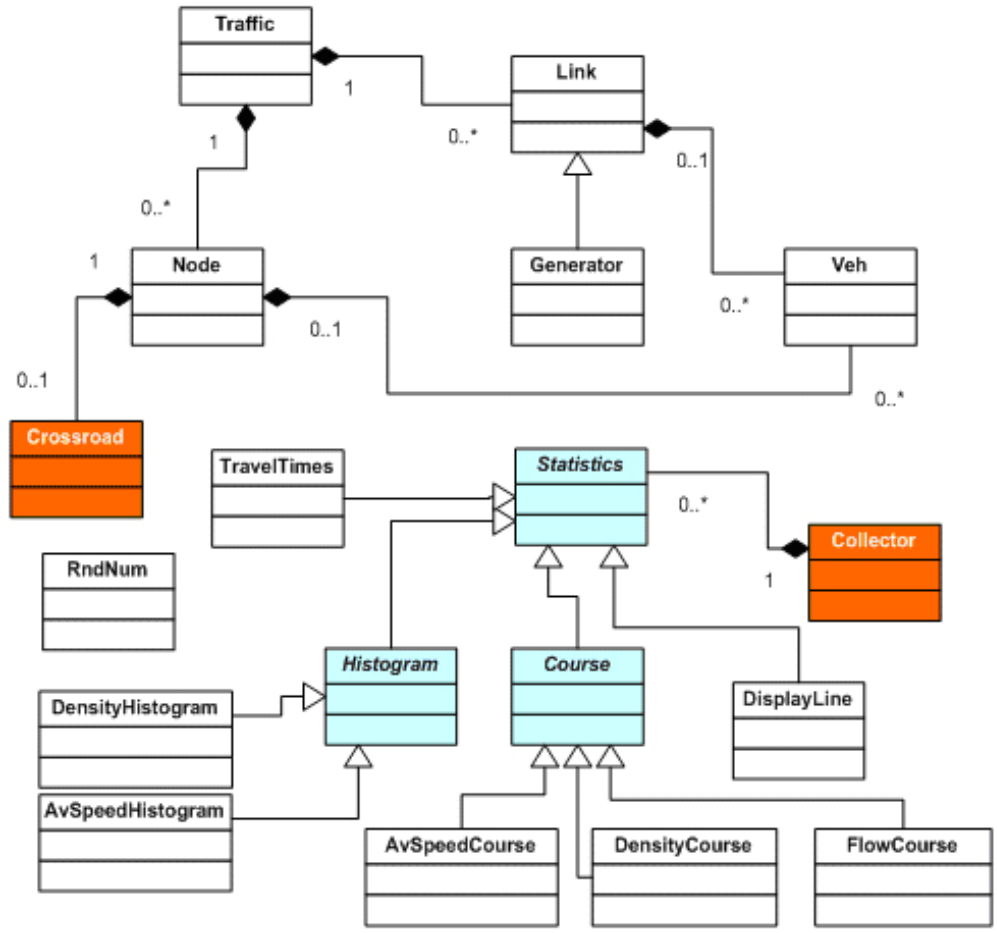
Nyní k výpočtu jednotlivých veličin. Tok lze snadno určit jako součet vozidel, která projela danou lokací během doby měření. Hustotu lze snadno určit jako podíl délky hrany a počtu agentů na této hraně. Ovšem pro naše účely bude vhodnější určit hustotu poněkud složitějším způsobem. Sčítejme po dobu měření množství vozidel v dané oblasti, resp. počet obsazených polí. Podílem tohoto údaje se součinem doby měření a délky oblasti získáme hledanou hustotu dopravy.

Další, z pohledu uživatele, atraktivní veličinou je průměrná rychlost vozidel na dané hraně. Tento údaj úzce souvisí s hustotou a opět bývá často využíván pro společný grafický výstup. Tuto veličinu lze opět určit velice snadno jako součet rychlostí všech vozidel ku jejich počtu.

Pro sběr a uchovávání statistických dat je nutné vytvořit *kolektor*. V něm může uživatel zaregistrovat své požadavky a po ukončení měření, mu kolektor zprostředkuje odpovídající výstup.

3 Implementace

V této části bych se rád zmínil o praktickém postupu jaký jsem zvolil při realizaci návrhu jednotlivých tříd. O věcech, které se mi podařilo implementovat a v závěrečném shrnutí i o věcech, které se mi implementovat nepodařilo, ale které by mohly být implementovány v dalším rozšíření.



Obrázek 2.4.1 - Diagram tříd knihovny traffic

3.1 Hrany a generátory

Vycházeje z návrhu, byla mým cílem třída, obsahující pole ukazatelů na třídu agentů. Místo pole jsem využil kontejner typu vector. To má oproti dynamickému poli řadu výhod, zejména ve snadné správě paměti a řadě vestavěných funkcí usnadňující práci a přístup k jednotlivým prvkům. Tuto třídu jsem pojmenoval `Link`.

Jelikož musí být každá hrana připojena ke dvěma uzlům, přičemž z jednoho vychází a do druhého směřuje, obsahuje třída `Link` dvě datové položky ukazatelů na typ `Node`. Je přitom možné mít více hran se stejným vstupním a výstupním uzlem. Počet buněk na každé hraně je ovlivněný její délkou a definovanou délkou jedné buňky, ta je neměnná a uživatel má možnost pouze nastavit délku hrany, přičemž samotný výpočet a vytvoření jednotlivých buněk je provedeno při inicializaci.

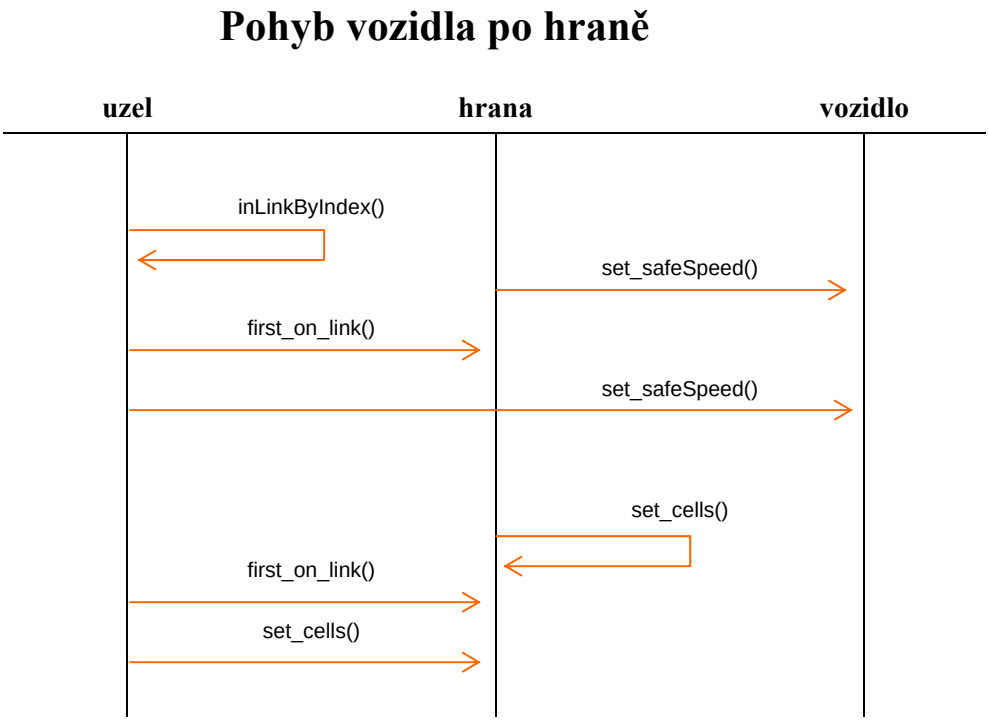
Pro testové účely se často uvažuje jediná hrana, na níž lze sledovat mechanismus pohybu vozidel v modelu, a následně je pomocí statistických ukazatelů porovnávat s reálným systémem. Jak je znázorněno na časovém diagramu 3.1.2, uzel prochází všechny své vstupní hrany. Každá z nich zajistí nastavení bezpečné rychlosti všem vozidlům kromě vedoucího, kterému ji nastavuje uzel.

V tomto případě uzel pouze supluje činnost, kterou by mohla provádět hrana přímo, výhody tohoto postupu se projeví teprve v okamžiku, kdy bude nutné provést vedoucí vozidla křižovatkou.

Nastavení bezpečné rychlostí se liší podle druhu použitého celulárního automatu. Pomocí nastavených maker při překladu, lze výpočet rychlosti vozidel provádět jako *deterministický CA* (DETERMINISTIC_CA), *stochastický CA* (STOCHASTIC_CA), *stochastický CA se zpomaleným startem* (SLOW_START_CA) a nebo jako *časově orientovaný CA* (TIME_ORIENTED_CA).

Konečným krokem je fyzické přesunutí vozidel o odpovídající vzdálenost danou bezpečnou rychlostí. Opět o vedoucí vozidlo se stará uzel a o zbylá příslušné hrany.

Při přesunutí vozidel delších než jedna buňka, dochází k nepatrnému zkreslení v okamžiku, kdy takové vozidlo vstupuje do křižovatky. Není řešena situace, kdy část takového vozidla je již v křižovatce a část ještě stále na hraně. Každé vozidlo vjíždí do křižovatky celé.



Obrázek 3.1.1 – sled operací pro pohyb vozidel po hraně

Vzhledem k faktu, že generátory, tedy hrany generující vozidla, ze své podstaty vycházejí z návrhu hrany, využil jsem s výhodou dědičnosti a třída Generátor tak obsahuje pouze pozměněnou virtuální metodu pro nastavení rychlosti a pohyb agentů a vlastní specifické datové položky.

Jedná se zejména o seznam tříd jednotlivých agentů a k nim příslušných vah, z nichž generátor náhodným způsobem vybere vždy jednoho. Uživatel může vkládat libovolný počet těchto elementů, respektive je omezen maximálním počtem prvků v kontejneru typu vektor. Při každém vložení je k váze zadané v parametru připočtena váha předka, případně nula, pokud vkládáme první prvek.

Výběr potom probíhá na základě vygenerování náhodné hodnoty rovnoměrného rozložení z intervalu nula až váha posledního vozidla.

Zda bude dané kolo vygenerováno vozidlo nebo ne, je pravděpodobnostní funkce s náhodnou veličinou rovnou nebo větší jedné Poissonova rozdělení. Vstupním parametrem lambda je průměrný počet agentů, kteří vstoupí na tuto hranu za jednu časovou jednotku, tedy tzv. *vstupní hustota dopravy*. Uživatel má možnost nastavit více těchto hodnot, přičemž každé z nich je přiřazen stejně dlouhý úsek běhu simulace. Hustotu na vstupních hranách lze tedy dynamicky měnit podobně, jako k tomu dochází v reálných systémech.

Po vygenerování musejí být vozidla umístěna do úložiště. To se běžně označuje jako *parkoviště* a přestože mívá podobu fronty, k jeho implementaci jsem použil asociativní pole ve formě kontejneru typu multimapa. V něm jsou jednotliví agenti řazeni podle času kdy budou vpuštěni do simulace.

Někdy se ovšem může stát, že agent do simulace sice vstoupit smí, ale vstupní buňka je blokována jiným agentem. Takový agent již nemůže dále zůstat na parkovišti a je přesunut do alternativní struktury, kde čeká na uvolnění vstupní buňky. Tato bezprioritní fronta FIFO je implementována pomocí kontejneru typu fronta. Nově příchozí agenti jsou řazeni na konec fronty a opouštějí ji podle pořadí v jakém do ní vstoupili.

Pro generování pseudonáhodných hodnot jsem vytvořil kongruentní generátor náhodných čísel, s využitím postupů popsaných v [4]. Třída Rand obsahuje metody pro generování hodnot uniformního rozložení, exonenciálního, Poissonova a Gaussova. Postup a konstanty použité v rozptylovací funkci byly převzaty ze zdrojů volně dostupných na internetu.

3.2 Dopravní prostředky

Třída `Veh` zapouzdřuje charakteristiky jednotlivých agentů. Ti se vzájemně liší zejména délkou, tedy kolik buněk v daném okamžiku obsadí, a maximální konstrukční rychlostí. Mimoto lze každému agentu přiřadit vlastní cestovní plán. Ten sestává z identifikátorů hran, kterými agent musí projet, aby se dostal k cíli. Seznam je implementován jako kontejner typu vektor.

Každý agent se smí pohybovat nejvýše takovou rychlostí, aby neohrozil agenta před sebou. V praxi je rychlost pohybu zpravidla nižší než tato hodnota. Při běhu simulace je nejprve nastavena datová položka uchovávající informaci o takzvané bezpečné rychlosti a teprve poté, co je aktualizována u všech agentů, je proveden vlastní pohyb.

3.2.1 Cestovní plán

Agenti procházejí systémem podle svého cestovního plánu. Ten sestává z identifikátorů hran, kterými musí agent projet, aby dosáhl svého cíle. Cíl každého agenta určí uživatel. S každým průjezdem křižovatkou je z cestovního plánu odebrán identifikátor hrany, kterou vozidlo právě

projelo. Agent by měl dosáhnout cíle ve stejném okamžiku, kdy je z jeho cestovního plánu odebrána poslední položka.

Cestovní plán je vytvářen automaticky, pro každé vozidlo, které uživatel vloží do generátoru. Jednotlivé hrany jsou ohodnoceny minimální dobou, kterou vozidlo potřebuje k jejímu projetí. Tím nám z vytvořeného dopravního systému vzniká ohodnocený graf, na jehož procházení lze použít některou ze standardních metod. V tomto případě jsem se pro nalezení nejrychlejší cesty rozhodl využít *Dijkstrův algoritmus*. Ten jsem mírně upravil tak aby vyhledal všechny cesty.

3.2.2 Systém automatického vyhledávání tras

Při prohledávání ohodnoceného orientovaného grafu pomocí Dijkstrova algoritmu, postupujeme z výchozího uzlu. Ohodnotíme všechny výstupní hrany a jejich výstupní uzly vložíme do zásobníku s informací o tom, kolik času potřebujeme k tomu, abychom jich z počátečního uzlu dosáhli, a kterými hranami přitom musíme projet.

V případě, že uzel již byl do zásobníku vložen, provedeme jejich vzájemné porovnání a data aktualizujeme podle uzlu, kterého jsme schopni dosáhnout rychleji. Pokud se dostaneme do koncového uzlu, našli jsme jednu z možných cest. Jakmile se dostaneme na konec zásobníku, prohledali jsme všechny možné cesty a zbývá vybrat z nich tu nejrychlejší.

Vzhledem ke konstrukci křižovatek, je výše popsaný algoritmus mírně poupraven tak, aby vytvářel cestovní plán jako seznam identifikátorů hran. Pro očekávané rozšíření knihovny o systém rozhodování jsou uchovávány informace o všech nalezených cestách.

3.3 Uzly a křižovatky

Jediným účelem uzlu, implementovaném třídou `Node`, je nést informaci o vstupních a výstupních hranách. Tyto struktury, kontejnery typu vektor, obsahují ukazatele na třídu `Link` a jsou automaticky aktualizovány v okamžiku, kdy je některé z hran přiřazen vstupní nebo výstupní uzel. Z tohoto důvodu je vhodné, avšak nikoli nutné, dodržet postup návrhu simulačního modelu, tak jak jsem jej nastínil v testovém modelu dopravního systému města Uničova, který je přiložen k této práci.

Kritickým místem simulace dopravního systému jsou právě křižovatky. Při vytváření třídy `Crossroad`, jsem se snažil v maximální možné míře kopírovat reálné zákonitosti, tak aby na druhou stranu nebyl pro uživatele jejich návrh příliš složitý nebo zdlouhavý.

Nakonec jsem zvolil postup, kdy je konstruktoru třídy předáno předem inicializované pole hodnot, z nichž si až samotná třída vytvoří struktury potřebné ke svému běhu. Tak jako v návrhu hran, je i v tomto případě ponechána volba na uživateli, který pomocí řady metod může každou křižovatku inicializovat ručně.

Poté, co je instance třídy vytvořena, je nutné ji přiřadit konkrétnímu uzlu. Každý uzel smí obsahovat nejvýše jednu instanci třídy `Crossroad`. I v případě, že ji uzel neobsahuje, je zajištěn pohyb agentů na vstupních i výstupních hranách, ale nikoli skrze uzel, tj. všechna vozidla na vstupních hranách, která dosáhnou jejího konce jsou ze simulace odstraněna podobně, jakoby dokončila svůj cestovní plán.

Průjezd vozidel křižovatkou, tak jak je znázorněn v časovém diagramu 3.3.1, již plně využívá uzlů, které rozhodují o tom, kdy a která vozidla do křižovatky vpustí. Postupuje vstupními hranami podle jejich priority, zvolená hrana nastaví všem svým vozidlům, mimo vedoucí, bezpečnou rychlost, potom uzel najde na aktuální hraně vedoucí vozidlo a nastaví mu bezpečnou rychlost bez ohledu na to, zda bude moci později křižovatkou projet.

V druhé fázi každý uzel opět prochází své vstupní hrany podle priority. Nejprve přesune vozidla uvnitř hrany, načež vyhledá vedoucí vozidlo a ověří zda může vjet do křižovatky. V současné verzi simulační knihovny spočívá ověření v prozkoumání všech hran, které do křižovatky vstupují zprava, vzhledem k hraně, která je ověřována. Pokud je na některé z těchto hran vedoucí vozidlo křižovatce blíže než je vzdálenost, kterou je schopno v tomto kole urazit, je žádost o vjezd do křižovatky zamítnuta.

Teprve v okamžiku, kdy je jasné, zda vozidlo bude do křižovatky vstupovat, dojde k jeho přesunutí, případně je upravena jeho rychlost a vozidlo zastaví na okraji křižovatky. Pohyb skrze křižovatku zajišťuje výhradně instance třídy `Crossroad`. Ta provádí dopravní prostředky z jedné hrany na druhou a uchovává ta, která v daném kole nestihla křižovatkou projet. K těmto účelům disponuje kontejnerem typu multimap.

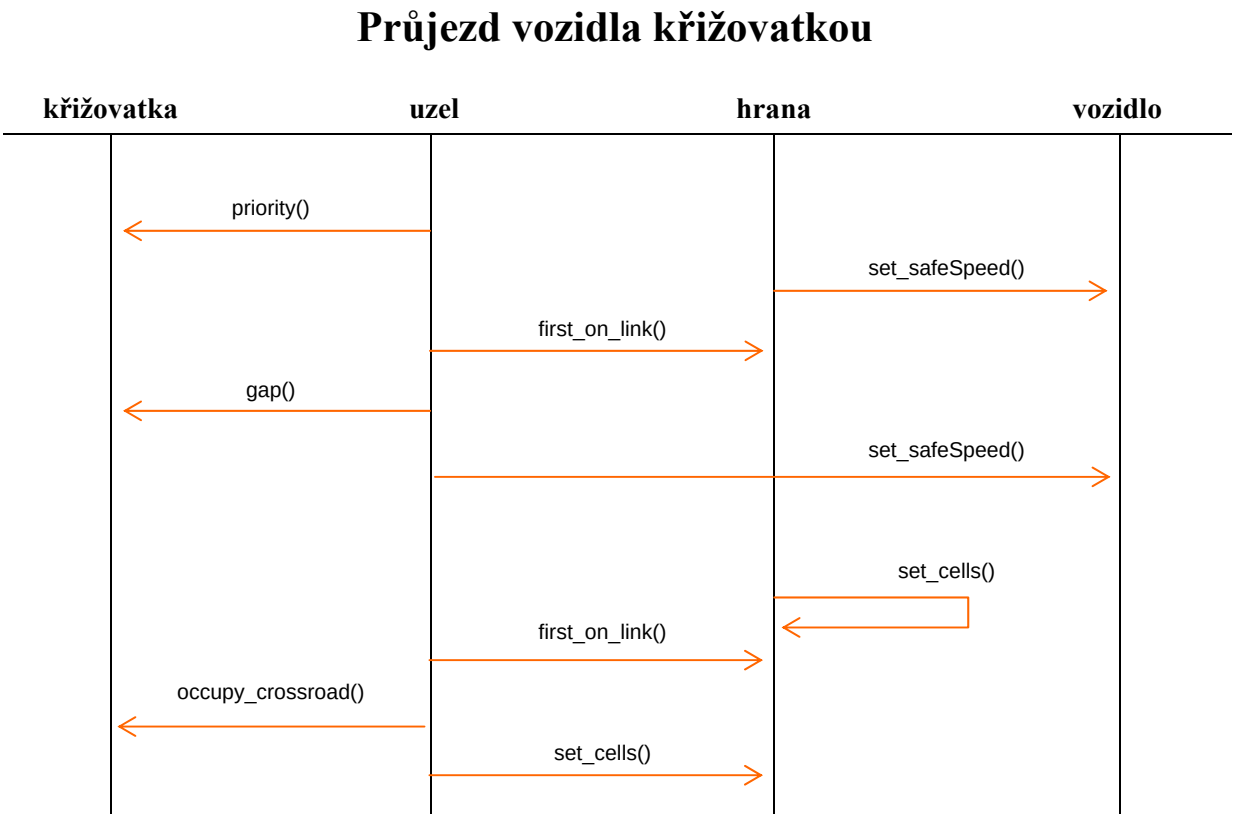
Chceme-li se v nejvyšší možné míře přiblížit reálnému systému, je nutné při průchodu agentů křižovatkou uvažovat vzájemné vzdálenosti vstupních a výstupních hran a jejich pozici. Rozmístění hran a jejich priorita jsou nutné pro reprezentaci pravidel silničního provozu.

Jak již bylo zmíněno výše, každá hrana nese informaci o své prioritě. Platí, že čím nižší hodnota, tím vyšší priorita. Nejvyšší priorita má nulovou hodnotu. V případě, kdy by více hran mělo shodnou prioritu, docházelo by, při vkládání do kontejneru typu multimap, k preferování první vložené hrany. K vyřešení tohoto problému bylo nutné zvolit jiný postup.

Indexem vkládaných hran není číslo celé (původní priorita), nýbrž desetinné. Toho je dosaženo součtem priority hrany a náhodného čísla v rozsahu 0 až 0,999 9 periodických. Tímto způsobem se při opakovaném spouštění simulace, obsahující hrany se shodnou prioritou, dosáhne vždy jiného pořadí těchto hran.

Při běhu systém nejprve ověří, zda v křižovatce nezůstalo vozidlo z minulého kola, v případě že tomu tak není, postupuje od první hrany v asociativním poli priorit a ověřuje, zda vedoucí vozidlo může vstoupit do křižovatky. V případě, že tomu tak je, uloží původní prioritu této hrany jako práh. Od tohoto okamžiku smí do křižovatky vstoupit pouze vozidla z hran jejichž původní priorita je shodná s prahem. Vzhledem k rozložení hran a platným dopravních předpisům se rozhodne zda

takové vozidlo smí do křižovatky vstoupit. V současné verzi systém uplatňuje pouze pravidlo pravé ruky. Vozidla, která nemohou do křižovatky vstoupit, zůstávají stát na hranici křižovatky, tedy poslední buňce hrany. Vozidla, jímž rychlost nedovolila v daném kole křižovatkou projet, jsou vložena do kontejneru typu vektor, kde vyčkají do dalšího kola.



Obrázek 3.3.1 – sled operací pro průjezd vozidla křižovatkou

3.4 Kolektor

Kolektor slouží ke sběru statistických dat. Jelikož předem není známo o jaká data bude mí uživatel zájem, měl by návrh poskytovat dostatečnou podporu, pro maximální kreativitu. Toho lze dosáhnout využitím bázových abstraktních tříd.

Nejprve je třeba rozložit celou operaci sběru a zpracování statistických dat na jednotlivé úkony a pokusit se najít společné ukazatele, které budou platit pro všechny potomky této bázové třídy. Za tímto účelem jsem vytvořil bázovou abstraktní třídu `Statistics`. Tato třída obsahuje kromě datových položek tři významné čistě virtuální metody, tedy metody, které budou definovány teprve v potomcích této třídy a bez jejich definice nebude možné vytvořit instanci příslušné třídy.

- `getData()` – slouží pro sběr dat z uživatelem definované hrany,
- `collect_data()` – zpracovává sebraná data,
- `output()` – vypisuje zpracovaná data na standardní výstup.

Hrana z níž budou data sbírána je rovněž součástí třídy `Statistics`, neboť data týkající se dopravy budou v převážné většině získávána právě odtud. Nic ovšem nebrání uživateli v některém z potomků definovat jiný zdroj dat, například uzly.

Třída `Statistics` sama o sobě zatím neví nic o způsobu jakým má s daty pracovat a proto je třeba od ní odvodit třídy, u kterých již budeme vědět jak některé z výše zmíněných metod budou pracovat. Informace o tom, zda bude uživatel chtít na výstupu histogram, nebo bude-li chtít zobrazit vývoj dat v čase, případně pouze zobrazit pohyb vozidel na hraně, nám stačí k tomu, abychom byli schopni definovat metodu pro zpracování sebraných dat a jejich grafickou reprezentaci.

Přesně k tomuto účelu slouží třídy Histogram a `Course`. Třída `DisplayLine` je poněkud specifická v tom, že jsme v ní schopni definovat i metodu pro sběr dat. Jak název napovídá, třída `Histogram` slouží ke zpracování a zobrazení histogramu, zatímco třída `Course` vypíše údaje o vývoji dat během simulace.

Vzhledem k tomu, že stále nevíme jaká data budou třídy `Histogram` a `Course` zpracovávat, byly rovněž vytvořeny jako abstraktní. Uživateli tudíž stačí ve svých odvozených třídách jednoduše definovat metodu pro sběr dat, o která má zájem a o zbytek je postaráno. Součástí knihovny jsou třídy několika nejpoužívanějších statistických údajů, o nichž se nyní krátce zmíním.

Jak bylo zmíněno v návrhu, mezi nejdůležitější statistické ukazatele patří histogram hustoty dopravy a průběh toku dopravy. K tomuto účelu jsem sestrojil dva potomky třídy `Statistics`. Třída `DensityHistogram` obsahuje metody pro výpočet hustoty a zpracování histogramu hustoty dopravy a třída `FlowCourse` poskytuje informace o průběhu toku dopravy na dané hraně. Totéž co třída `FlowCourse` provádí i třída `DensityCourse`, pouze vstupním údajem je hustota dopravy. Průměrnou rychlost vozidel na hraně zjišťují, zpracovávají a prezentují třídy `AvSpeedHistogram` a `AvSpeedCourse`.

Významnou je i další potomek třídy `Statistics`, třída `TravelTimes`. Jejím účelem je sbírat a zpracovávat informace o tom, jak průměrně dlouho trvala jednotli-vým vozidlům cesta z místa kde vstoupily do systému, do místa, kde systém opustila, a na kterém probíhá měření. Tento statistický údaj je nezbytný pro pozdější rozhodování vozidla, zda má stále pokračovat k vytyčenému cíli, nebo má změnit směr k cíli následujícímu.

Na rozdíl od hran a uzlů má statistika vlastní kolektor, tedy sběrné zařízení, které registruje všechny vytvořené instance a zajišťuje, že budou zpracovány všechny požadavky uživatele. Tuto funkcionalitu zajišťuje třída `Collector`, která instance třídy `Statistics` registruje v datové

struktuře typu vektor pomocí metody `add2collector()`, která tudíž musí být zavolána ihned po vytvoření instance dané třídy, a to nejlépe v konstruktoru této třídy.

3.5 Řídící jednotka

V předcházejících kapitolách byl popsán návrh a funkčnost jednotlivých komponent simulace. Nyní zbývá všechny tyto části spojit dohromady. Tento úkol plní řídící jednotka, která v dynamických datových strukturách uchovává hrany a uzly, aby bylo možné provést všechny předepsané úkony v daném pořadí. Důraz je kladen zejména na zajištění paralelnosti pohybu všech vozidel.

Z tohoto důvodu řídící jednotka nejprve všem vozidlům nastaví tzv. *bezpečnou rychlost*, tedy rychlost, kterou vozidlo v tomto kole dorazí k nejbližší překážce, nebo aktuální rychlost zvýšenou o akceleraci. Ještě před tím, než je nastavena bezpečná rychlost vedoucím vozidlům, jsou na generátory přemístěna vozidla z čekací fronty.

V druhé fázi nejprve na hranách přesune všechna vozidla za vedoucím vozidlem. Pokud se jedná o vstupní hranu, jsou vpuštěna všechna vozidla, se startovacím časem rovným nebo nižším než aktuální čas simulace. Následně se pokusí vyprázdnit křižovatku a nakonec přesune vedoucí vozidla, přičemž zpravidla dochází ke korekci rychlosti, v závislosti na tom, zda vozidlo bude moci křižovatkou projet, nebo nikoli.

3.5.1 Postup při vytváření modelu

Je rozumné nejprve vytvořit globální objekty uzlů a teprve potom hran. Zachováním tohoto pořadí můžeme pro vytvoření hrany zavolat konstruktor s ukazateli na počáteční a koncový uzel, který současně nastaví v příslušných uzlech vytvářenou hranu jako vstupní nebo výstupní a ušetří nám tak práci s ručním nastavováním.

Uzly, které tvoří křižovatku, tj. mají alespoň jednu vstupní a výstupní hranu, obsahují nejvýše jednu instanci třídy `Crossroad`. Konstruktor této třídy se volá s parametrem, kterým je dvourozměrné pole celočíselných kladných čísel. Ty vyjadřují možné přechody mezi vstupními a výstupními hranami uzlu a jejich vzájemné vzdálenosti. Aby tyto údaje mohly být správně načteny a zpracovány je bohužel nutné zadat vždy na konci celé sekvence, tj. na pozici posledního řádku speciální číselnou sekvenci. Takto vytvořené instance křižovatek stačí připojit k příslušným uzlům pomocí jejich členské metody.

V tuto chvíli je definována kostra celé simulace. Každý z uzlů zná své vstupní a výstupní hrany a ví které z nich jsou generátory a je tedy možné vzniklou strukturou plynule procházet. Jako další krok je tedy třeba vytvořit vozidla, která poslouží jako vzor při generování, poté co budou přiřazena k příslušným generátorům.

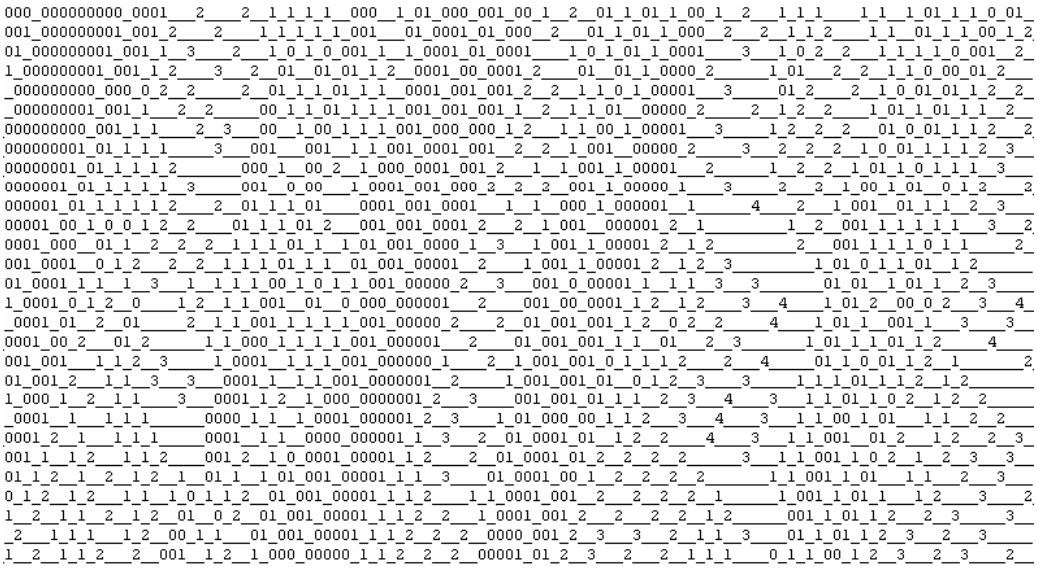
Pomocí globální metody třídy `Traffic`, `create()`, dojde k inicializaci řídící jednotky. Do ní zbývá zaregistrovat vytvořené uzly, hrany, generátory a případné statistiky. Metoda `simulate()`, jejímž parametrem je počet kol, nastartuje běh celé simulace a na závěr zobrazí požadované statistiky.

4 Testování

Jednotlivé testy postupují po krocích od nejzákladnější funkcionality, jakou je pohyb vozidla po jediné hraně, přes okruh až k průchodu křižovatkou. Posledním testem je model dopravního systému, vytvořený na základě skutečných reálií. Ten by měl především poskytnout vodítko, jak postupovat při vytváření obdobných projektů a zároveň otestovat hranice, zejména co se týče paměťové náročnosti.

4.1 Test č. 1A

Cílem tohoto testu je ověřit základní funkčnost systému na jednoduché hraně. Kontrolním výstupem je zobrazení pohybu vozidel na hraně. Jednotlivá vozidla by se měla pohybovat podle pravidel definovaných ve specifikaci návrhu. Po dosažení konce hrany budou ze systému uvolněna.



Obrázek 4.1.1 - Znázornění pohybu vozidel podle pravidel TO celulárního automatu

Jedna hrana, propojuje dva uzly. Na hraně jsou generována vozidla, která jsou po dosažení konce hrany odstraněna. V průběhu simulace dochází k postupném snižování pravděpodobnosti příjezdu vozidla. Jsou generovány dva typy vozidel, osobní a nákladní, které je třikrát delší než osobní.vozidlo.

Na obrázku 4.1.1 představují jednotlivé řádky časovou posloupnost simulace v krocích. Číslice vyjadřuje počet polí, o které se vozidlo přesune v příštím kroku. Vozidla se pohybují podle pravidel časově orientovaného celulárního automatu.

Z výstupu testu je jasně patrné, že pohyb vozidel po hraně probíhá přesně podle specifikace. Současně lze vypozorovat, že i četnost nově vygenerovaných vozidel se podle očekávání v čase taktéž snižuje. V několika případech je akcelerace ovlivněna náhodností a vozidlo se tak nerozjíždí lineárně, což dále zvyšuje reálnost simulačního modelu. Výsledky tohoto testu lze tedy označit za zcela vyhovující.

4.2 Test č. 1B

Tento test, by měl prověřit vývoj statistických ukazatelů v okamžiku, kdy je hrana uzavřena tak, že vytváří kruh, tedy vozidla, která dosáhnou konce hrany pokračují v pohybu od jejího začátku. Současně na hranu vjíždějí nová vozidla, čímž dochází ke zvyšování hustoty dopravy a vzniku dopravní zácpy. Výsledek testu by měl poskytnout důležité statistické ukazatele o závislosti mezi hustotou provozu a tokem dopravy.

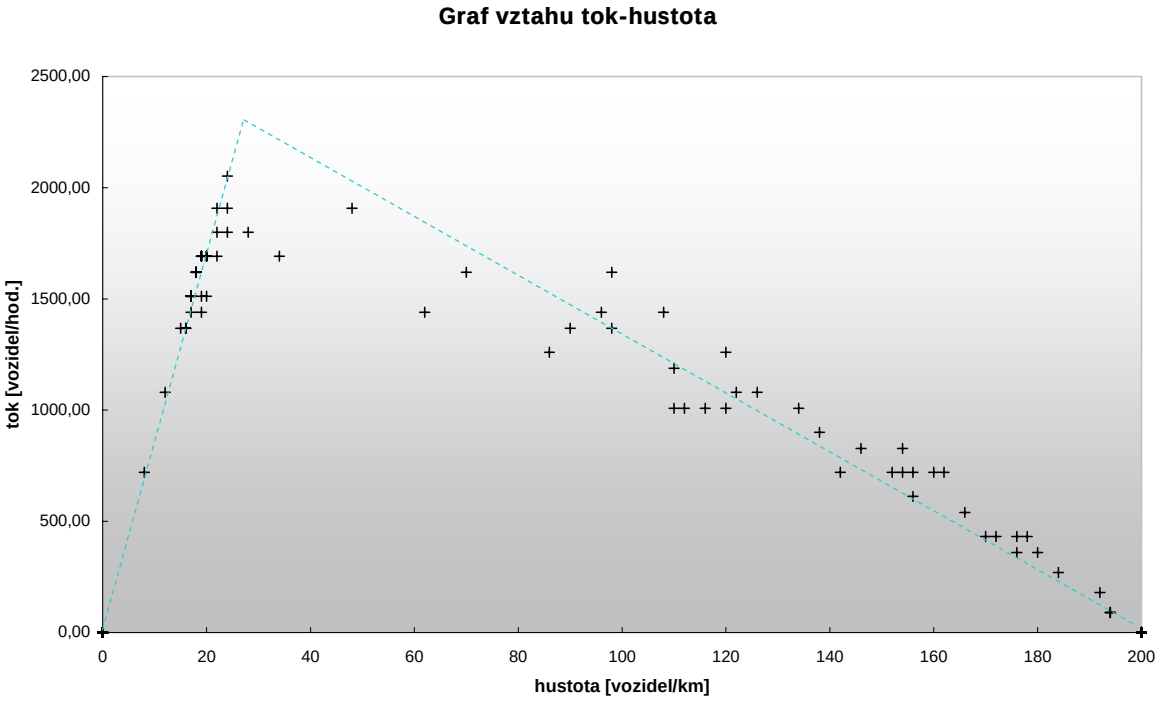
Jeden uzel má jako vstupní i výstupní tutéž hranu. Na hraně budou generována vozidla, po dobu rovnou délce hrany, vyjádřené v počtu polí. Pravděpodobnost příjezdu vozidla je po celou dobu simulace konstantní.

S rostoucím počtem vozidel uvnitř kruhové dráhy, dochází k růstu hustoty provozu a vzniku front. Tak jak se uvolňuje prostor ve směru pohybu vozidel, dává se i fronta zvolna do pohybu. Statistická data byla sesbírána z místa uprostřed hrany a odpovídají očekávaným datům, popsaným v dostupné literatuře [1].

Jak bylo zmíněno v návrhu vztah toku k hustotě dopravy patří k fundamentálním statistickým ukazatelům dopravního systému. Na ose x je zobrazen průměrný počet vozidel na jednom kilometru dopravní komunikace, v daném časovém úseku. Na ose y potom jemu odpovídající tok dopravy, vyjádřený v počtu vozidel za jednu hodinu, která projela měřeným úsekem. I tento údaj je zprůměrňován z několika sekund, v tomto případě z dvaceti.

Výsledkem je graf na obr. 4.2.1, Graf vztahu tok-hustota. Je z něj patrné, že při nulové hustotě provozu je tok rovněž nulový, neboť měřenou oblastí nemohla projet žádná vozidla. Při úplném zaplnění hrany je tok opět nulový, protože žádná nová vozidla nejsou schopna do měřené oblasti vjet. Při hustotě provozu mezi desíti až patnácti procenty dochází k prudkému nárůstu toku dopravy. Vozidla mají dostatek volného prostoru, aby se mohla pohybovat maximální rychlostí a je jich dostatečný počet, aby v krátkých časových intervalech projížděla měřeným úsekem. Maximálního toku dosáhne systém někde kolem patnácti procent hustoty dopravního provozu. Potom začíná docházet k zpomalování dopravy vlivem vznikajících front a tok dopravy zvolna klesá.

Systém využívá pro pohyb vozidel časově orientovaný model celulárního automatu. Což se projevuje v hodnotě maximálního toku dopravy, která je mírně nižší než v případě stochastického CA a více než o jednu šestinu nižší než u deterministického CA, viz. kapitola 2.5.3.



Obrázek 4.2.1 - Graf vztahu tok-hustota

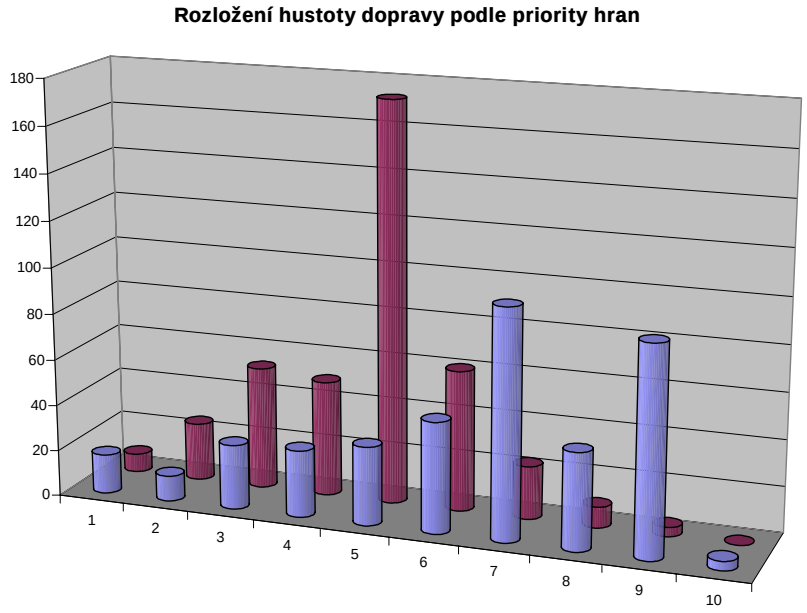
4.3 Test č. 2

Ověřuje, podobně jako test č. 1A, funkcionalitu knihovny. Tentokrát se ale zaměřuje na plynulý průchod vozidel křižovatkou. Je nutné ověřit, zda nedochází k zablokování křižovatky v průběhu simulace a zda jsou do křižovatky vpouštěna i vozidla z hran s nižší prioritou.

Simulační model sestává z osmi hran a pěti uzlů. Na čtyřech hranách probíhá generování vozidel a společně směřují do jediného uzlu. Zbylé hrany z toho samého uzlu směřují každá k jednomu ze zbývajících. Středový uzel obsahuje křižovatku. Čtyři hrany, z toho dvě vstupní a dvě výstupní, mají nižší prioritu. Všechny generující hrany mají totožné hodnoty pravděpodobnosti pro vygenerování vozidla, i co se týče typů vozidel.

Z výsledků testu vyplývá že na hranách s nižší prioritou, na obr. 4.3.1 modrá část, je hustota dopravy častěji vyšší než na hranách s vyšší prioritou. Výsledek odpovídá, logickému předpokladu, kdy vozidla přijíždějící ke křižovatce z hrany s vyšší prioritou mají přednost před těmi, které přijíždějí z hran, jejichž priorita je nižší.

Hodnoty na ose y v grafu Rozložení hustoty dopravy podle priority hran vznikly součtem četností na všech hranách stejné priority. Modrá část značí hrany s nižší prioritou, část červená hrany s vyšší prioritou. Se vzdáleností od levého okraje grafu roste naměřená hustota na daných hranách.



Obrázek 4.3.1 - Porovnání histogramů hustot dopravy na hranách o různých prioritách

4.4 Simulační model města Uničova

Pro náležité prověření funkčnosti celé knihovny jsem navrhl simulační model reálného dopravního systému města Uničova, viz. obr. 4.4.1. Celý systém sestává z dvaadvaceti uzlů, šedesáti hran, šestnácti křižovatek. Simulace probíhá po dobu dvou celých dní, tedy celkem 172800 vteřin.

Dopravní systém města lze podle směru rozdělit na dvě části. Místa uvnitř systému, odkud vozidla směřují směrem ze systému nebo do jiné jeho části a místa na okraji systému, odkud vozidla směřují dovnitř systému. Tato místa budeme označovat jako generátory vozidel.

Zatímco rozmístění vnějších generátorů, je zcela intuitivní, pro správné umístění generátorů vnitřních je třeba provést analýzu městské zástavby v jednotlivých částech města. Vodítkem mohou být sídliště v jednotlivých městských částech a nemělo by chybět centrum města, které bývá zpravidla obydleno velmi hustě.

Každý vlastník automobilu je jedno potenciální vozidlo uvnitř dopravního systému. Je proto nutné zjistit jaká je struktura obyvatel v okolí generátorů. Zejména kam a jak často cestují. Na základě těchto dat, lze určit hodnotu pravděpodobnosti, s jakou budou generována vozidla a hodnotu pro náhodný výběr vozidla směřujícího do konkrétní lokace.

Simulační model sestává z dopravních tepen města. Vynechány byly nevýznamné komunikace, které byly zčásti nahrazeny generátory uvnitř systému. Ty jsou rozmístěny v blízkosti centra a dvou hlavních sídlišť. Veškeré komunikace mají jediný pruh, což odpovídá realitě, ovšem s tím, že není možné předjíždění skrze protisměrný pruh. Tato funkcionalita nebyla v době odevzdání hotova.

Při vytváření simulačního modelu bylo nutné, vedle samotné konstrukce dopravní sítě rozhodnout, jakým způsobem se bude v průběhu dne měnit pravděpodobnost s jakou budou generována vozidla. Vyšel jsem z jednoduchého předpokladu, kdy z rezidenčních částí směrem k průmyslovým a komerčním zónám míří lidé za prací, nákupem apod., a po určité době se vracejí zpět.

Z tohoto důvodu tato pravděpodobnost narůstá mezi pátou a osmou ráno, kdy lidé směřují do práce a do škol, dále kolem jedné hodiny odpoledne a mezi devátou a desátou večerní, kdy převážně v průmyslových podnicích nastupuje odpolední a noční směna. Mimoto, z průmyslových zón míří zejména na výpadovky nákladní vozidla s výrobky nebo materiálem potřebným pro výrobu.

Naopak v průmyslových částech roste doprava kolem sedmé hodiny ranní, třetí hodiny odpolední a kolem půlnoci, tak jak končí jednotlivé pracovní směny. Z a do centra, které je jak rezidenční tak i komerční zónou proudí doprava téměř nepřetržitě. S mírnými výkyvy zhruba kopíruje průmyslové části, s tím že rozdíly mezi dopravní špičkou a dobou útlumu nejsou tak velké jako v průmyslových částech.

Na základě této analýzy lze nastavit generátorům nastavit potřebné parametry, tak aby model co možná nejvěrněji simuloval reálný systém. Nejčastěji generovaná vozidla budou ta, která směřují do průmyslových zón a do centra. Naopak v průmyslových zónách budou nejčastěji generována vozidla směřující do nehustěji obydlených částí města. Mimoto zde budou výrazněji zastoupeny i kamiony, směřující na výpadovky.

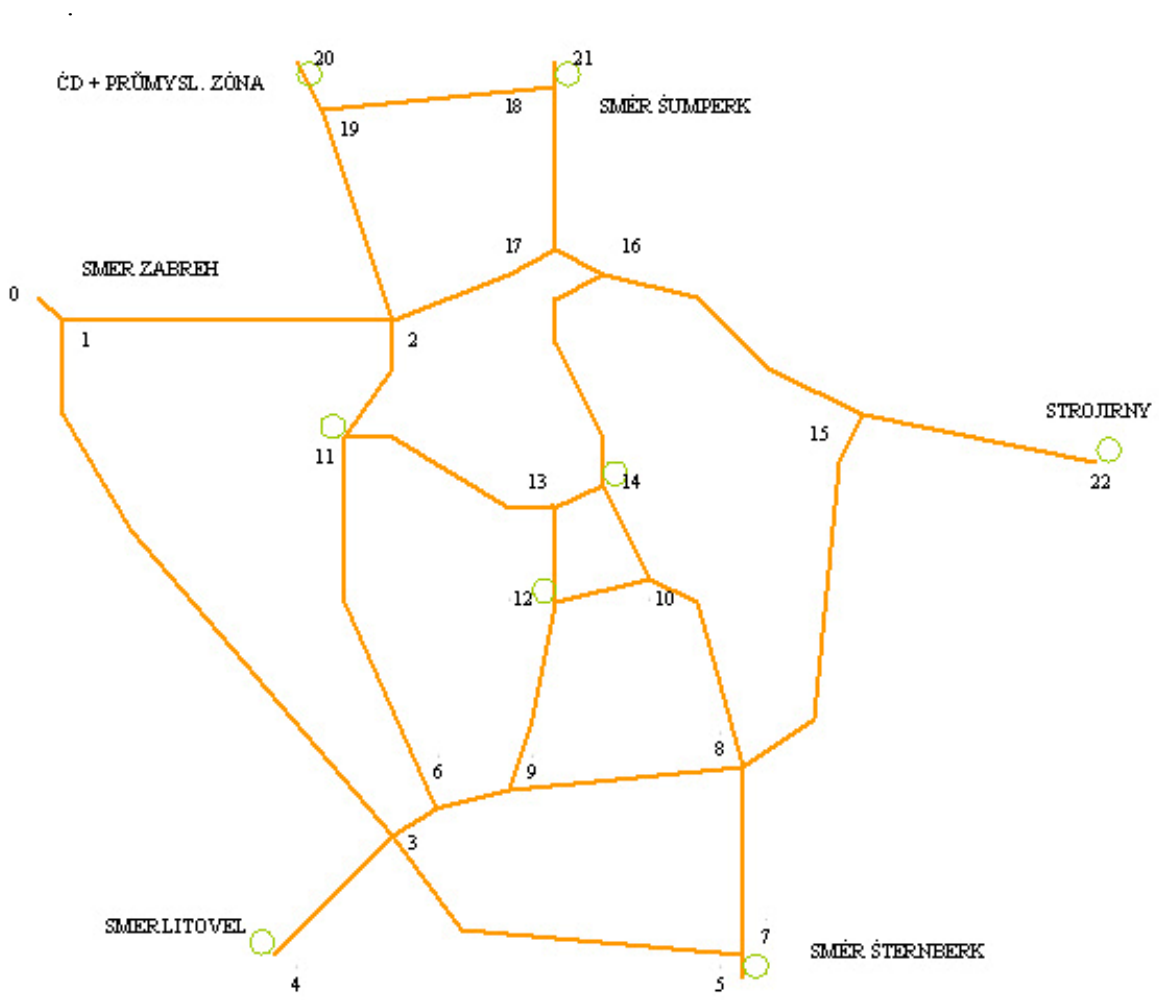
Poměr časů skutečného a očekávaného dosažení cíle		
měření	**průmyslová zóna**	**strojírny**
1	1,959	1,781
2	2,074	1,666
3	3,210	2,056
4	2,865	1,907
5	2,483	1,811
průměr	2,518	1,844

Tabulka 4.4.1 - Průměrné zpoždění vozidel

Statistickým údajem, který nás bude zajímat je průměrná doba, porovnaná s ideálním časem, za který vozidla dosáhla svého cíle. Konkrétně jedné ze dvou hran v průmyslových zónách. Jinými slovy, kolikrát vyšší je čas, který potřebují řidiči k tomu, aby se v ranní špičce dostali do práce, oproti

ideálnímu stavu, kdy by se pohybovali maximální možnou rychlostí. Samozřejmě s ohledem na rychlostní limity.

Prováděním tohoto testu na školním serveru byl odhalen problém s nedostatkem volné paměti. V první verzi byla všechna vozidla generována na začátku simulace, tak že pro každé vozidlo byla již v této fázi alokována paměť. To se ukázalo nejen jako zbytečné, ale navíc to na průběh simulace kladlo extrémní paměťové nároky.



Obrázek 4.4.1 - Vyznačení hlavních komunikací města Uničova

Proto bylo nutné změnit způsob inicializace tak, aby byla vozidla generována, až za běhu simulace. Tedy součástí každého kola se stalo generování vozidel. Je-li generující hrana volná, vstupují na ni okamžitě po vytvoření, jinak čekají ve frontě, dokud se vstupní pole neuvolní. Rozdíl je v tom, že vozidla, která mezitím dosáhnou cíle své cesty, jsou fyzicky odstraněna, a uvolněná paměť tak může posloužit nově vygenerovanému vozidlu. Tato verze již dokázala provádět simulaci za stejných podmínek s více než dvojnásobným počtem kroků, než zhavarovala pro nedostatek paměti. Úskalí tohoto řešení se skrývalo v situaci, kdy hrana byla ucpaná, tedy nemohly na ni vjíždět žádná

vozidla, která zůstávala v tzv. *čekací frontě*. Přesto pro ně již byla alokována paměť, i když to v daném okamžiku vůbec nebylo potřeba.

V zatím poslední verzi jsem se vrátil k vygenerování všech vozidel v průběhu inicializace. Rozdíl oproti minulým verzím je v tom, že do kontejneru uchovávajícím informaci o tom, kdy které vozidlo smí vstoupit do simulace, je vložen pouze čas kdy se tak stane. Jakmile tento čas nastane, je vybráno vozidlo, které v okamžiku uvolnění hrany bude vloženo. Zatím stále není alokována žádná nová paměť, do čekací fronty je pouze vložen ukazatel na vybrané vozidlo. Teprve potom, co se vstupní oblast hrany uvolní, tak dojde k alokaci paměti a nastavení specifických datových položek objektu. Díky tomuto přístupu, lze vyčíslit maximální paměťovou náročnost simulace, jako součet polí všech hran a počtu křižovatek, nezávisle na délce běhu simulace. Velikost jednoho vozidla je proměnlivá v závislosti na složitosti cesty, kterou nalezlo k dosažení svého cíle.

S poslední úpravou bylo možné provést simulaci v požadovaném rozsahu. Řidiči směřující do průmyslové zóny musejí počítat s tím, že jim tato cesta v ranní špičce v průměru zabere o více než sto padesát procent více času, zatímco strojíren vozidla dosáhnou s asi osmdesátiprocentním zpožděním

5 Závěr

Funkcionalita současné verze knihovny pro podporu modelování dopravních systémů splňuje veškeré požadavky, které byly vytyčeny v návrhu. Díky objektově orientovanému přístupu je možné ji v budoucnu snadno rozšiřovat. To se týká zejména víceproudých hran a schopnosti vlastního rozhodování agentů.

Během práce jsem několikrát modifikoval specifikaci, s tím jak jsem poznával nové postupy a možnosti při práci s objekty a zejména dědičností. V poslední fázi se to týkalo zejména optimalizace práce s pamětí. Podobně bouřlivým vývojem prošla téměř každá část, nejvíce však soubor tříd pro sběr, zpracování a výstup statistických dat. Kdy z několika samostatných tříd, vznikla robustní struktura, která s využitím abstraktních bázových tříd a dědičnosti plně pokryla veškeré potřeby v této oblasti nejen nyní, ale i v budoucnosti.

I přes veškerou snahu není knihovna odolná proti hrubým zásahům uživatele, které tak mohou způsobit pád běhu simulace. Ty jsou samozřejmě ošetřeny zpracováním výjimek, přesto mnou vytvořené prostředí není zdaleka tak uživatelsky přívětivé jako například prostředí Simlib/C++, použité v první části této práce.

Literatura

[1] Kai, N. *Multi-agent transportation simulation*. Computer Based Learning Unit, University of Leeds, 2004.

[2] Xue, W., Hudson, Ch. *System dynamics based traffic flow simulation*. Buckinghamshire Chilterns University College, 2003.

[3] Zwicker, Ch., *Parallel within-day re-planning in traffic simulation*. Diploma Thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2004

[4] Rábová, Z. a kol., *Modelování a simulace*. Brno, 2005.

[5] Prata, S., *Mistrovství v C++*. Brno, 2004.