Пловдивски Университет

Факултет "Математика и информатика"

Дипломна работа

Оптимизиране на транспортната мрежа в град Пловдив

Дипломант:
Венелин Вълков
фак. № 0901261093

Научен ръководител:
гл. ас. д-р Ангел Голев
кат. „Компютърни технологии“

03. 07. 2012 г.

Съдържание

1 Увод **3**

1.1 Цел . . . 3
1.2 Изисквания . . . 3
1.3 Постигане на целта . . . 4
1.4 Целеви групи . . . 4
1.5 Желан резултат . . . 4
1.6 Задачи . . . 4
1.7 Структура . . . 5

2 Изложение **6**

2.1 Проучване . . . 6
2.1.1 Какво е моделиране? . . . 6
2.1.2 Какво е симулация? . . . 7
2.1.3 Визуална симулация . . . 8
2.1.4 Преглед на системи за създаване на симулация . . . 8
2.2 Създаване на симулационна система . . . 15
2.2.1 Модел . . . 15
2.2.2 Компоненти на симулацията . . . 18
2.2.3 Избор на технология и програмен език . . . 19
2.2.4 Програмиране на системата . . . 26

3 Резултати **28**

4 Заключение **29**

A Използвани съкращения **32**

Списък на фигурите

2.1 Модел на симулационната система 17
2.2 Резултат изпълнението на първата програма на *Moai* 25
2.3 Клас диаграма на симулационната система 27

Глава 1

Увод

Искали ли сте да прекарате повече време с детето си на закуска, да се приберете по-рано от работа и да прекарате време с любимият човек или да излезнете с приятели? Може и да поспите повече, разбира се. Колко хубаво би било това да е правило, вместо изключение. Всеки пловдивчанин, използващ пътната мрежа на града, губи средно около 60 мин. на ден в трафика.

1.1 Цел

Целта на дипломната работа е да предложи начин за намаляне на времето прекарано в пътната мрежа на град Пловдив с 10%.

1.2 Изисквания

- цената за използването на градската мрежа трябва да остане същата или да е по-ниска
- повишаване на комфорта и спокойствието при използването на градската мрежа
- увеличаване печалбите на превозващите компании
- намаляне на вредните емисии във въздуха
- намаляне на броя катастрофи

1.3 Постигане на целта

За постигане на целите на дипломната работа, спрямо поставените изисквания, се разглеждат -

- модерни технологични решения в подобни ситуации
- създаване на високо паралелизирана симулация за намиране на критичните точки от транспортната мрежа и тяхното оптимизиране
- за получените резултати се анализират за да се получи частично или пълно решение на проблема

1.4 Целеви групи

Хора, които използват транспортната мрежа, през най-натоварените часове на денонощието. Важно за всеки от участниците е бързото достигане на съответна точка от града, без това да пречи на личното им здраве и комфорт.

1.5 Желан резултат

Работата може да се сметне за успешна, ако се постигне намаляне на прекараното време в транспортната мрежа с 10%.

1.6 Задачи

- Проучване върху методите за изграждане на симулации
- Избор между съществуваща и специализирана за целта система
- Избор на програмен език
- Програмиране на самата система
- Провеждане на симулации

1.7 Структура

Настоящата дипломна работа се състои от:

Увод обосновка на проблема, поставяне на конкретна цел, целеви групи и желан резултат

Изисквания

Изложение разделено на три основни части:

Проучване основни концепции при създаване на компютърна визуална симулация

Създаване на симулация система, специфично създадена за нуждите на градската мрежа в град Пловдив. Избор на технологии. Програмиране.

Провеждане на симулации използване на реални данни като вход за създадената система

Резултати обявяване на получените резултати

Заключение Наблюдения върху получените резултати и дискусия. Постигнати ли са поставените цели и къде е имало проблеми. Какво може да бъде развито в бъдеще.

Глава 2

Изложение

2.1 Проучване

2.1.1 Какво е моделиране?

Моделирането е процесът по създаване на модел. Моделът е представяне на съществуващ обект от анализираната система. Той е близък, но по-прост от обекта в реалната система. Когато създава модел, анализаторът използва модела за да предскаже промените в системата. От една страна, моделът трябва да е близък до реалния обект и притежава неговите свойства. От друга страна, трябва да е лесен за разбиране и променяне. Добрият модел е добър компромис между леснота и реализъм [Anu]

Препоръчва се, усложняването на модела да става итеративно (iteratively). Важно е, моделът да продължава да бъде верен през този процес. Някои техники за това включват:

- Симулиране, включваики модела, с познати входни и изходни данни
- Пресечено валидиране (cross-validation) [Mahoney]

В зависимост от средствата, използвани за построяването им, моделите биват: физически (обекти, процеси и явления, евентуално различни по физическата си природа от оригинала, но с аналогични свойства), математически

(теории, методи и обекти – функции, уравнения, редове и други), информационни (информационни методи, обекти и процеси), компютърни (програми, данни и други) и така нататък [Totkov]

Компютърното моделиране на информацията и автоматизирането на информационните дейности предполагат въвеждането, изучаването и използването на различни модели на информацията за обектите и за дейностите, в които те участват. Информационните обекти и информационните процеси са абстрактни модели на информацията и информационните дейности, наречени абстрактни (концептуални) информационни модели. В компютърната информатика абстрактните информационни модели се проектират и създават под формата на алгоритми и структури от данни. [Totkov]

Симулационните компютърни модели представляват интерес за научните и стопанските дейности. Компютърният модел дава възможност за разкриване на структурата, механизма и закономерностите, на които се подчиняват различни природни феномени или проблеми от реалния свят. По този начин е възможно прогнозиране, управление или изкуствено възпроизвеждане на тези модели, което намалява средствата влагани в експериментални изследвания. [Iliev]

2.1.2 Какво е симулация?

Симулация е процес при който се извършват различни действия/експерименти върху построения модел, вместо върху реална система. В най-общото си значение, симулация е инструмент за наблюдаване на това как една система работи. Симулацията може да се тества под различни условия и за различни времеви диапазони.

Симулация може да се използва когато искаме да:

- Намаляне на риска за недостигане на поставен срок
- Намиране на непредвидени пречки и проблеми
- Оптимизация на поведението на системата

Видове:

Непрекъсната симулация(Continuous simulation) (НС) променя състоянието си в неопределени моменти във времето, използват диференциални уравнения

Дискретно-събитийна симулация(Discrete-event simulation) (ДСС) променя състоянието си в определени моменти във времето, използват се събития

2.1.3 Визуална симулация

Интересно за поставената цел е, че е необходима направата на визуален компонент за системата. По този начин, лесно ще може да бъдат наблюдавани различни движения на обектите. Това може да доведе до по-успешна и лесна оптимизация на цялостната пътна мрежа в града.

Такъв тип симулация налага използването на знания от компютърната графика. Още повече, доближава ни (и дори довежда) до използването на системи за съставяне на компютърни игри. Изненадващо, игрите не са разглеждани задълбочено, или поне тяхната структура и начин на работа, в академичните среди [Holzkorn].

Визуалната част на системата е критична за това дали системата ще бъде използвана. Лесното и удобно използване са важни за всеки софтуерен продукт. Потребителят е този, за който системата ни трябва да се грижи добре. [Microsoft]

Следва разглеждане на някои от основните и най-добри симулационни системи[1]. Това се прави с цел, правене на възможно най-добър избор на такава, която е в съответствие с поставените ни задачи и изисквания.

2.1.4 Преглед на системи за създаване на симулация

Simplex3

Simplex3 е система за симулации, която може да работи под Windows и Unix операционни системи. Тя е универсално приложима когато имаме дискретни модели, процеси лесно моделируеми с помощта на опашки или транспортни модели.

[1] `http://en.wikipedia.org/wiki/List_of_computer_simulation_software`

Как се създава симулация

Позволява лесно и бързо научаване на системата, като това не изисква научаването на нов програмен език от високо ниво.

Технологии

Притежава собствен език за създаване на модел, наречен Simplex-MDL. Той позволява описанието на почти всякакъв вид модели. Въпреки това, модели използващи частни диференциални уравнения не могат да се представят лесно.

Приложение

Благодарение на универсалността си, Simplex3 може да се използва за академични цели. Лесно може да се приспособи за случаи, в които няма разработен специализиран симулационен софтуер.

Оценка

От сайта на системата, не става ясно до колко тя е поддържана. Липсва добра документация.[Simplex3]

AnyLogic

AnyLogic е платена система за общ вид моделиране и симулационен инструмент за дискретни, непрекъснати и хибридни системи.

Как се създава симулация

Системата предоставя Графичен потребителски интерфейс(Graphical User Interface) (ГПИ) за създаване на моделите, които допълнително могат да бъдат разширявани с помощта на програмният език Java. AnyLogic предоставя моделиране чрез UML-базирано обектно-ориентирано моделиране, блок-схеми, диаграми на автомати, диференциални и алгебрични уравнения и други.

Технологии

Предоставя възможност за създаване на Java аплети, които позволяват лесно споделяне на симулациите, както и поставянето им в интернет.

Програмни парадигми

Моделиращият език на системата е разширение на UML-RT - голяма колекция от най-добри практики доказали се при моделирането на големи и сложни симулации.

Приложение

Тя е полезна когато искаме да разглеждаме симулации относно:

- Контролни системи
- Производство
- Телекомуникации
- Обучение
- Логистика(Logistics)
- Компютърни системи

Оценка

AnyLogic е доста обширна и понякога тежка система. Предоставя голям набор от полезни инструменти. Тя е *много скъпа*[2], което е в разрез с изискванията ни. [AnyLogic]

[2]Цената на най-евтиното издание на продукта е 4800 €

Tortuga

Tortuga е симулационна система с отворен код, разработена от MITRE Corporation [3] в периода от 2004 до 2006 година. Тя е ориентирана изцяло към дискретно-събитийни симулации.

Как се създава симулация

Симулация в системата може да се представи като взаимодействия между процеси или планирани (scheduled) събития. Системата изисква множество от написани (от потребителя) класове на Java, инсталиран Java средства за разработка(Java Development Kit) (JDK), Ant - инструмент за автоматично билдване (build) и самата симулационна система. Предоставя се и ГПИ за управление и наблюдение на самата симулация.

Технологии

Tortuga е изцяло написана на Java. Това прави възможно използването й под Windows и Unix базирани системи. Интересното тук е, че за постигане на своите цели Tortuga използва AOP (Aspect Oriented Programming) [AOP], като не се изисква от потребителя да има познания в тази насока, стига да компилира програмата си по специфичен начин.

Програмни парадигми

Системата използва програмна парадигма която многократно намаля сложността при създаване на симулация. Тя третира всяка единица от симулацията като отделна нишка. Java виртуалната машина ограничава броя на нишките [4], което води до ограничение броя на симулационните единици.

Приложение

[3] `http://www.mitre.org/`

[4] Броят варира спрямо имплементацията, настройки при пускане и хардуера

Tortuga може да бъде използвана самостоятелно или като част от по-голям проект. Тя предоставя споделяне на симулации в Java аплети, също както и AnyLogic.

Оценка

Лицензът който използва е Lesser General Public License (LGPL)[5], което прави системата добър кандидат за по-нататъшно разглеждане. Възможността й за използване под много платформи (благодарение на Java виртуалната машина) е още един голям плюс. Не трябва да забравяме използването и на *AOP*, което я изкарва от "стандартна"Java програма. Според сайта на системата, от 2008 година, поддръжката за нея е спряна. [Tortuga]

SimPy

SimPy е система с отворен код, създадена през 2002 г. от Klaus G. Muller и Tony Vignaux. След това към тях се присъединяват много други разработчици. Тя предоставя възможност за създаване на дискретно-събитийни симулации.

Как се създава симулация

Симулациите в системата могат да се представят като взаимодействия между процеси. Те се изграждат чрез програмен код. За създаването на проста симулация се изискват около 10 реда код, които включват познания по основни класове и функции от системата. Има GUI(използващо библиотеката Tk[6]) за предоставяне на входни данни и наблюдаване на симулациите.

Технологии

SimPy е написана на Python[7] и поддържа версии от 2.3 до 3.2 включително. Благодарение на Python виртуалната машина (при използване на стандартната

[5] `http://www.gnu.org/licenses/lgpl.html`
[6] `http://www.tcl.tk/`
[7] `http://www.python.org/`

CPython имплементация[8], SimPy може да се изпълнява под Windows и Unix операционни системи.

Програмни парадигми

Системата използва обектно-ориентиран подход към създаване на симулации. Благодарение на езика на който е написана, с нея бързо може да се направи дори и по-сложен модел.

Приложение

SimPy може да бъде използвана самостоятелно, както и навякъде където има Python интерпретатор. Предоставя се набор от пакети за изобразяване на графики и управление на структури от данни, които могат да бъдат използвани извън рамките на библиотеката. Готови симулации лесно могат да бъдат пакетирани като изпълними файлове и предоставени на други потребители.

Оценка

Лицензирана е под LGPL лиценз. Системата изглежда добре поддържана. Последната версия е 2.3, излезнала през Декември 2011 г. Използва лесен и достъпен език, но без много допълнителни инструменти за по-бързо и лесно създаване на симулация. [SimPy]

GarlicSim

GarlicSim е безплатен продукт с отворен код[9], който се опитва да предостави нова концепция, за това как всъщност може да се използват симулациите. Автор на системата е Ram Rachum[10].

Как се създава симулация

[8] `http://wiki.python.org/moin/CPython`
[9] `https://github.com/cool-RR/GarlicSim`
[10] `http://ram.rachum.com/`

Проектът обещава да напише повтарящата се част от кода, необходим за една симулация, вместо нас и ни остави да работим върху по-важната част от проекта - самата симулация. Самата тя, се създава чрез писане на програмен код. Може да постигнем направата на проста симулация с 5 реда код!

Технологии

GarlicSim е изцяло написана на Python и официално все още не поддържа Python 3.x сериите. Системата предлага изчистен и лесен GUI за управление и наблюдение на симулациите.

Програмни парадигми

За създаване на системата е използван изцяло обектно-ориентиран подход, който допринася за лесното научаване на основните стъпки при правене на симулация. Основна дейност при извършване на симулация е т.нар. симулационен пакет (simulation package), който съдържа функция която определя стъпката за дадената симулация.

Приложение

Според автора, системата е достатъчно обща за да позволи симуларинето на каквато и да е симулация. Дадени възможности са:

- Физични
- Теория на игрите
- Разпространение на епидемии
- Квантова механика
- Електрически

Оценка

Лицензирана е под LGPL лиценз. GarlicSim е един чудесен продукт за начални опити за създаване на симулации. Основен проблем е, че е в алфа версия. Няма сведения за успешни употреби в някаква насока. Има добра, но не напълно достатъчна документация. За създаване на собствена симулация е необходимо написването на симулационен пакет, което до някъде обезсмисля обещанието за това да пишем по-малко код. [GarlicSim]

2.2 Създаване на симулационна система

След анализът върху съществуващите симулационни системи изглежда, че система която да реши поставените проблеми, не съществува. Повечето от тях, имат примитивни ГПИ които не предоставят необходимата гъвкавост на точно определена създадена за целта такава.

Под голям въпрос е и качеството на документацията, както и състоянието на наличните примери и качество на кода. Това оставя възможността да бъде разгледан един нетрадиционен вариант за решаване на такъв проблем.

2.2.1 Модел

За създаването на нашият модел и цялостната програма ще се използва итеративен подход [ArtOfAgile]. Това означава, че ще изградим основните части от системата - първо[11]. Допълнителна функционалност ще добавяме само, когато сме доволни от качеството на текущата програма.

Основната част от програмата ще контролира заложените закони и взаимодеиствия между обектите. При всяка нова стъпка от симулацията, обектите участващи в нея ще предоставят информация за състоянието в което се намират. То може да се променя, като това е изразено в промяна на техните свойства.

Необходимостта за изобразяване на самата симулация, по време на изпълнение и когато тя вече е приключила, води до някои допълнителни ограничения. Системата трябва да работи достатъчно бързо[12], за да предоставя гладка и лека

[11] Посоченият подход е лична интерпретация

[12] Между 30 и 60 кадъра в секунда

работа за потребителя.

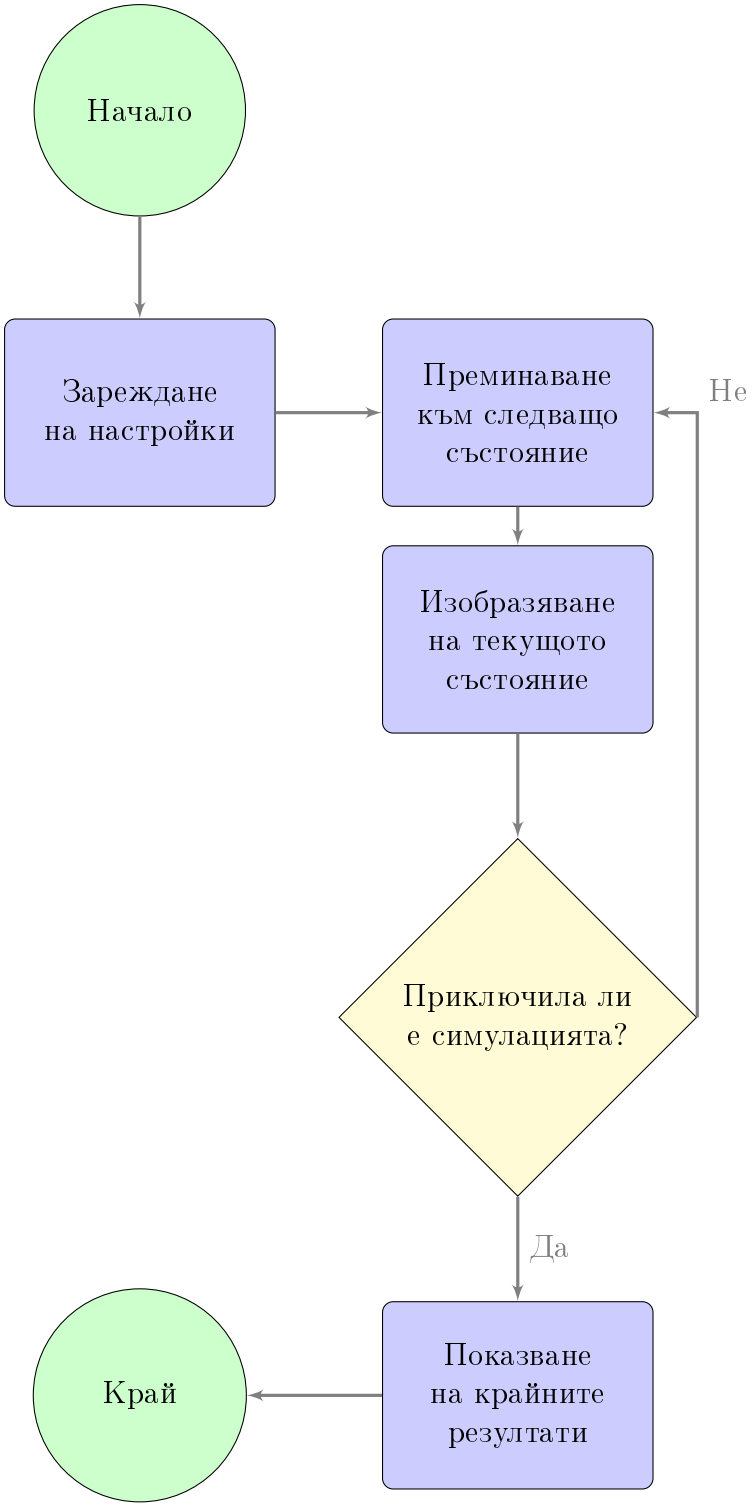
С така поставените ограничения и подходи, използването на ДСС предоставя:

- Гъвкавкост
- Лесна имплементация
- Лесна промяна
- Лесно скриване на детайлите от самия потребител

Още повече, за симулация на трафик е необходима симулация от тип опашка, която в повече от случаите се имплементира с помощта на ДСС [Barlas].

При стартиране на системата, тя ще изисква от потребителя да въведе необходимите данни за стартиране на симулация. Когато тя започне, на потребителя се дава възможност визуално да наблюдава симулацията на всяка своя стъпка. След приключването ѝ се предоставят данни за това как е протекла симулацията. На фиг. 2.1 е изобразена диаграма, която показва работата на така зададения модел.

Фигура 2.1: Модел на симулационната система



2.2.2 Компоненти на симулацията

Една ДСС се представя като поредица от случващи се събития. Всяко събитие в определен момент и поражда промяна на състоянието на системата.

Основни изграждащи единици

Всяка ДСС се нуждае от логика, която да отговаря на това какво се прави, когато дадено събитие се случи. Трябва и да се определят точните времена в които те ще се случат. Някои други компоненти, обаче, също са задължителни:

Часовник грижи се за запазването и променянето на текущото време за симулацията. Единиците в които времето се измерва се избират спрямо самата симулация. Благодарение на незабавното случвания на събитията, времето подскача (hops).

Участници в симулацията са конкретни обекти, които взаимодействат със други такива и влият пряко на резултатите на изпълнение. Определянето на това, какви точно трябва да бъдат те, зависи изцяло от модела. Тяхното количество в точно определен момент на симулацията, зависи от входните данни, както и текущата стъпка.

Списък на събитията представлява подредено множество от събития, които още не били изпълнени. Много от тях не са предварително дефинирани, а са създадени в следствие на изпълнение на предшестващи ги събития. Събитие е описано чрез тип и време на случване.

Понякога е необходимо да изпълняваме различни задачи, които се простират извън границите на един момент от времето. Обикновено това се моделира като поредица от събития. Те биват създвани като се използва диапазон от време.

Системата може да поддържа еднонишково(singnle-threaded) и многонишково(multi-threaded) изпълнение на събития в даден момент. При вторият вид може да се породят проблеми със синхронизацията между различните участници в симулацията. От друга страна, той допринася за по-бързото й представяне на потребителя.

За имплементация на списъка със събития се използва приоритетна опашка. Приоритетът е времето за случване на събитието. Няколко алгоритъма за създаване на приоритетни опашки са се доказали като подходящи при създаването на ДСС[Jones], най-вече "разширено дърво"(splay tree).

Случайни числа се използват за различни нужди по време на изпълнение на симулацията. Тя се нуждае от случайно генерирани променливи, в зависимост от модела. Използват се псевдо-случайни числа, вместо случайни числа, за да се използват еднакви данни когато искаме да проведем симулация с еднакво поведение.

Статистики се събират по време на изпълнение на симулацията. Тяхното естество, отново, е зависимо от модела на симулацията. Те представляват данните, който потребителя вижда след края на всяка симулация.

Условие за край е това, което трябва да е изпълнено за да приключи симулацията. При отсъствие на такова, теоретично, една симулация може да продължи безкрайно. Създателят на симулацията трябва да прецени какво точно да бъде то, за да може тя да бъде прекратена.

2.2.3 Избор на технология и програмен език

Светът на Lua

Lua е малък скриптов език, който кара да се усмихнеш, когато го използваш. Според авторите му той е: [13]

- Бърз
- Лек
- Лесен за вграждане (embeddable)
- Доказан

[13] `http://www.lua.org/about.html`

Той е напълно безплатен за употреба. Разпространява се под *MIT*[14] лиценз. Доказал се е като добър избор за скриптов език за множество комерсиални игри[15].

Автори на езика са Roberto Ierusalimschy, Waldemar Celes и Luiz Henrique de Figueiredo. Те разработват езика в университета PUC-Rio[16], като част от нуждите за тяхната група от технологии за компютърна графика.

Името на езика идва от португалската дума Lua, която означава "Луна".

Lua е смесица между обектно-ориентиран, функционален и програмиране свързано с данни (data-driven development) подход към създаването на програми. Има лесен за научаване синтаксис и сравнително малък брой концепции.

Имплементиран е като библиотека за програмния език *C*[17]. Поради тази причина, той няма концепция за"основна програма" (main program). Нуждае се от приемник (host), който извиква различни части от код, написан на *Lua*.

С помощта на *C* функции, *Lua* може да бъде пригоден за работа в различни, непредвидени от авторите му, области и споделя вече съществуващия синтаксис.

Концепции

Lua е динамично типизиран език (dynamically typed), което означава, че само стойностите имат тип.

Променливи

```
1 myIq = 1
2 print(myIq)
```

Изход

```
1 1
```

Има концепция за прихващане на грешките, която много се доближава до тази на изключенията от езици като C++ и Java.

[14] http://www.opensource.org/licenses/mit-license.php
[15] http://en.wikipedia.org/wiki/Category:Lua-scripted_video_games
[16] http://www.puc-rio.br/index.html
[17] http://en.wikipedia.org/wiki/C_(programming_language)

Грешки

```
function foo ()
  error("Hmm, something is wrong")
end

local okStatus, errorMessage = pcall(foo)
if okStatus then
  print("Works fine")
else
  -- error is raised
  print(errorMessage)
end
```

Изход

```
errors.lua:3: Hmm, something is wrong
```

Езикът предоставя и съвместни програми (coroutines). Съвместна програма в Lua представлява изпълнение на независима нишка.

Съвместни програми

```
function foo (a)
  print("foo", a)
  return coroutine.yield(2*a)
end

myCoroutine = coroutine.create(function (a,b)
  print("myCoroutine-body", a, b)
  local r = foo(a+1)
  print("myCoroutine-body", r)
  local r, s = coroutine.yield(a+b, a-b)
  print("myCoroutine-body", r, s)
  return b, "end"
end)

print("main", coroutine.resume(myCoroutine, 1, 10))
print("main", coroutine.resume(myCoroutine, "r"))
print("main", coroutine.resume(myCoroutine, "x", "y"))
print("main", coroutine.resume(myCoroutine, "x", "y"))
```

Изход

```
myCoroutine-body        1       10
foo             2
main            true    4
myCoroutine-body        r
main            true    11      -9
myCoroutine-body        x       y
main            true    10      end
main            false   cannot resume dead coroutine
```

Използването на класове в *Lua* е малко по-интересно и странно, в сравнение с езици като *C* и *Java*. То по-скоро напомня на прототипното наследяване в *JavaScript*[18]. Това позволява на всеки разработчик да създаде свой, собствен стил на писане на класове. Това може да е добро и лошо нещо.

Класове

```
Dog = {}
Dog.__index = Dog

function Dog.create(balance)
    local dog = {}           -- our new doggy
    setmetatable(dog,Dog)  -- make Dog handle lookup
    return dog
end

function Dog:speak()
  return "Bay Bay"
end

-- create and use an Dog
dog = Dog.create("Barky")
print("While playing with the dog, it says: ", dog:speak())
```

Изход

```
While playing with the dog, it says:    Bay Bay
```

[18] http://www.crockford.com/javascript/inheritance.html

Нужда от по-висока абстракция

Lua е прекрасен малък език, но сам по себе си, той не доставя необходимите инструменти за лесно създаване на симулационна система. Намиране на библиотека предоставяща по-висока абстракция е задължително.

Една такава библиотека е *Moai*[19] (произнася се Мое-Еуе). Тя е по-скоро двигател за игри (game engine). Разработката ѝ е започнала през 2010г. и е активно разработвана и до днес. Тя е с *отворен код*[20] и използва лицензът *CPAL*[21], който позволява да използваме софтуера за нашите цели. Разработена е от *Zipline Games*[22].

Moai е библиотека за разработване на игри върху множество от платформи, вариращи от десктоп до мобилни операционни системи. За основна графична библиотека се използва стандартът *OpenGL*[23], който позволява тази разнообразност. Поради добрата си свързаност със *C*, Moai може да постигне производителност, по-висока от тази на приложения написани на *Objective C* [24] за *iPhone*.

Обеткният модел на *Moai* е изцяло написан на *C++* и използва префикс "MOAI". Тези обекти се изчистват от Събирач на боклук(Garbage Collector) (СБ), също както и тези в *Lua*. За това спомагат и т.нар. контейнери в *Moai*, които се грижат да не останат неизчистени обекти от паметта.

По време на създаването на *Moai*, авторите са искали да направят библиотека, която да помага много за лесното създаване на симулации[25]. Например, по подразбиране началото на координатната система е в центъра на екрана. Това позволява по-лесното позициониране на обекти като цяло, тъй като обектите които са в центъра на екрана са по-важни за наблюдателя.

[19] http://getmoai.com/
[20] http://github.com/moai/moai-dev
[21] http://www.opensource.org/licenses/cpal_1.0
[22] http://www.ziplinegames.com/
[23] http://www.opengl.org/
[24] http://en.wikipedia.org/wiki/Objective-C
[25] http://getmoai.com/moai-basics-series/moai-basics-part-1.html

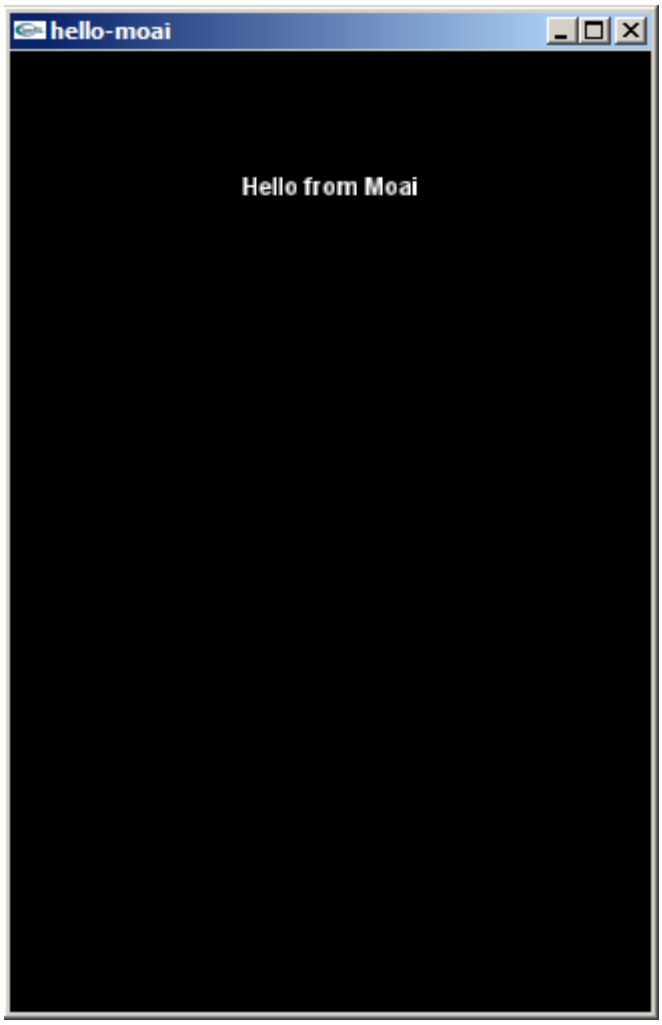
Програма, изписваща на екрана “Hello from Moai“, излежда така:

Здравей Moai

```
-- Author: Venelin Valkov
-- You will need Moai SDK to run this
-- based on hello-moai sample from the Moai SDK
MOAISim.openWindow ( "hello-moai", 320, 480 )

viewport = MOAIViewport.new ()
viewport:setSize ( 320, 480 )
viewport:setScale ( 320, 480 )

layer = MOAILayer2D.new ()
layer:setViewport ( viewport )
MOAISim.pushRenderPass ( layer )

font = MOAIFont.new ()
font:loadFromTTF ( "assets/fonts/arialbd.ttf", "
    abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789,.?!"
    , 12, 163 )

textbox = MOAITextBox.new ()
textbox:setFont ( font )
textbox:setTextSize ( 12 )
textbox:setRect ( -160, -80, 160, 80 )
textbox:setLoc ( 0, 100 )
textbox:setYFlip ( true )
textbox:setAlignment ( MOAITextBox.CENTER_JUSTIFY )
layer:insertProp ( textbox )

textbox:setString ( "Hello from Moai" )
textbox:spool ()
```

Когато програмата се изпълни, на екрана се визуализира фиг. 2.2

Фигура 2.2: Резултат изпълнението на първата програма на *Moai*



2.2.4 Програмиране на системата

За да се спази итеративния подход към създаването на системата, се имплементират някои от изискванията към системата. Също така, тези изисквания може допълнително да бъдат разбити на по-малки такива.

Първата версия има следната функционалност:

- Симулацията представлява пътно кръстовище с 4 посоки на движение.
- Всяка посока има само 1 пътно платно.
- Превозните средства могат да се движат само направо.
- Скоростта за движение е постоянна.
- Всяко кръстовище има 4 светофара.

Посочените опростявания ще ни позволят да намерим грешките, които сме допуснали при създаването на предварителния модел на системата. Още повече, те ще ускорят създаването на първоначална версия на системата.

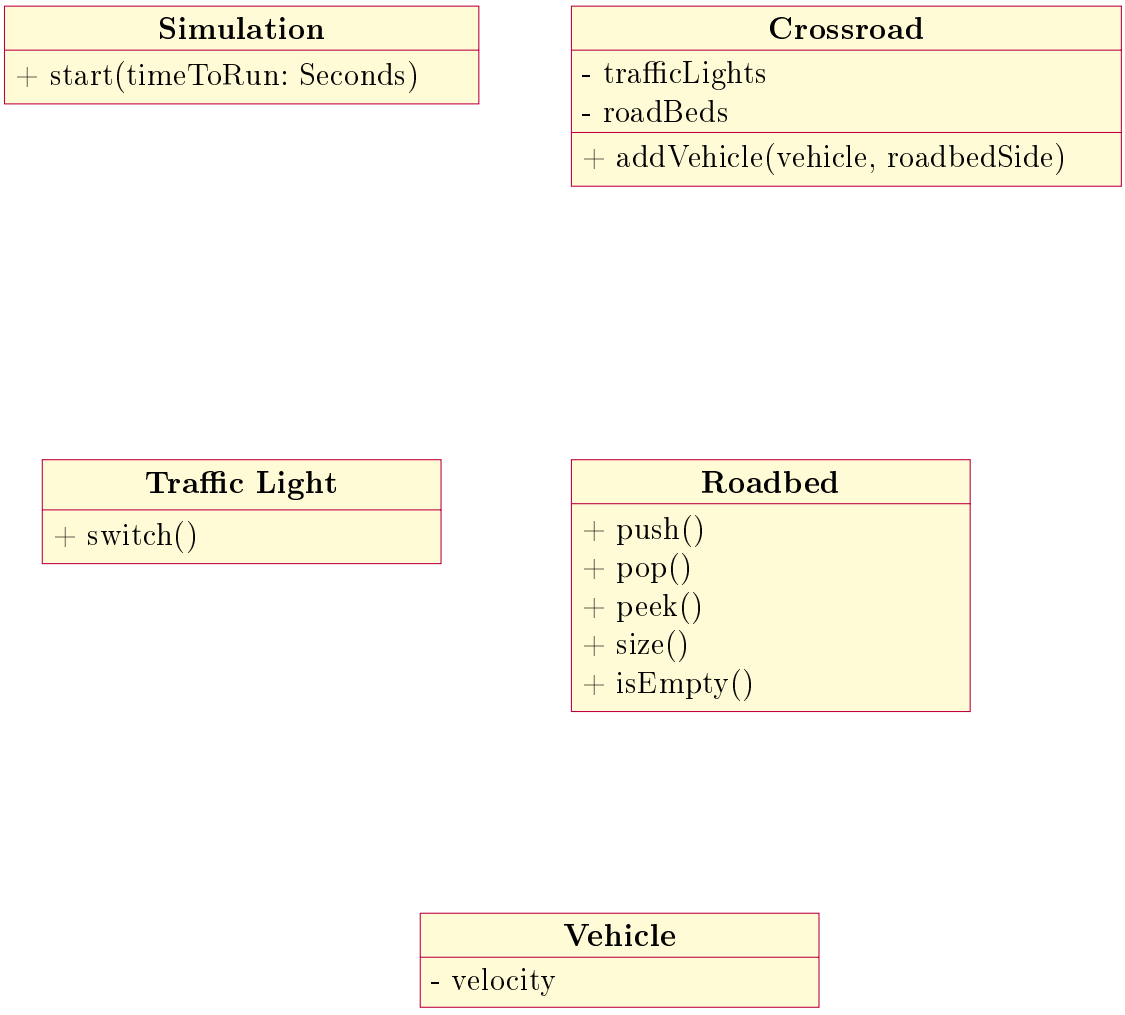
Итерация I

За описаният по-горе модел може да се използва проста архитектура, която да ни позволи по-лесно добавяне на нови възможности. Примерна такава е дадена на фиг. 2.3

Основният клас, който се грижи за правилното провеждане на симулацията е *Simulation*. Той съдържа условието за край, както и това да казва на всички участници в симулацията кога е минала една секунда.

Crossroad описва едно крастовище. То съдържа 4 пътни платна на които се добавят самите превозни средства. Важно е да се отбележи, че превозните средства не знаят за кръстовищата, а само за платната на които са и тези на които искат да отидат.

Фигура 2.3: Клас диаграма на симулационната система



Глава 3

Резултати

Глава 4

Заключение

Библиография

[AOP] Yuliyan Kiryakov and John Galletly. Aspect-oriented programming – case study experiences. 2003.

[Anu] Maria Anu. Introduction to modeling and simulation.

[AnyLogic] `http://www.xjtek.com/anylogic/why_anylogic/`.

[ArtOfAgile] James Shore. *The Art of Agile Development*. 2007.

[Barlas] Onur Ozgun and Yaman Barlas. Discrete vs. continuous simulation: When does it matter? 2009.

[GarlicSim] `http://garlicsim.org/`.

[Holzkorn] Peter Holzkorn. Physics simulation in games. 2008.

[Iliev] Антон Илиев и Георги Христозов и Тодорка Терзиева. Софтуерна среда за представяне на детерминирани динамични модели с възможност за статистика. 2006.

[Jones] Douglas W. Jones. An empirical comparison of priority-queue and event-set implementations. 1986.

[Mahoney] Kevin Mahoney. Model validation techniques. 2010.

[Microsoft] Microsoft Corporation. Usability in software design. 2000.

[SimPy] `http://simpy.sourceforge.net/`.

[Simplex3] `http://www.simplex3.net/Body/Introduction/English/indexAbstract.html`.

[Tortuga] `http://code.google.com/p/tortugades/`, 2004.

[Totkov] Георги Тотков. Концептуално и компютърно моделиране на езикови структури и процеси (с приложения за българския език). 2004.

Приложение А

Използвани съкращения

JDK Java средства за разработка(Java Development Kit)

НС Непрекъсната симулация(Continuous simulation)

ДСС Дискретно-събитийна симулация(Discrete-event simulation)

ГПИ Графичен потребителски интерфейс(Graphical User Interface)

LGPL Lesser General Public License

СБ Събирач на боклук(Garbage Collector)