**SIMULAREA ȘI OPTIMIZAREA TRAFICULUI URBAN.**

***Simularea macroscopică și optimizarea utilizând algoritmul genetic***

LUCRARE DE LICENŢĂ

|  |  |  |
| --- | --- | --- |
|  | Absolvent: | **Edwin-Wilhelm HOBOR** |
|  |  |  |
|  | Coordonator ştiinţific: | **Prof.dr.ing. Gheorghe SEBESTYEN** |

**2015**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | |  | |
| DECAN, |  | | DIRECTOR DEPARTAMENT, | |
| **Prof. dr. ing. Liviu MICLEA** |  | **Prof. dr. ing. Rodica POTOLEA** | |

Absolvent: **Edwin-Wilhelm HOBOR**

**SIMULAREA ȘI OPTIMIZAREA TRAFICULUI URBAN**

**Simularea macroscopică și optimizarea utilizând algoritmul genetic**

1. **Enunţul temei:** *Simularea la nivel macroscopic a traficului urban și optimizarea utilizând algoritmul genetic*
2. **Conţinutul lucrării:** *Introducere - Contextul proiectului, obiectivele proiectului, studiu bibliografic, analiză și fundamentare teoretică, proiectare de detaliu și implementare, testare și validare, manual de instalare și utilizare, concluzii, bibliografie, anexa 1.*
3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare și Google Scholar
4. **Consultanţi**:
5. **Data emiterii temei:** 1 Octombrie 2014
6. **Data predării:** 18 Iunie 2015

|  |  |
| --- | --- |
| Absolvent: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |
| Coordonator ştiinţific: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**Declaraţie pe proprie răspundere privind**

**autenticitatea lucrării de licenţă**

Subsemnatul(a)**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**, legitimat(ă) cu \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ seria \_\_\_\_\_\_\_ nr. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
CNP \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, autorul lucrării \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ a anului universitar \_\_\_\_\_\_\_\_\_\_, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

|  |  |  |
| --- | --- | --- |
| Data  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Nume, Prenume  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
|  |  | Semnătura |

Cuprins

[Capitolul 1. Introducere – Contextul proiectului 1](#_Toc422152968)

[1.1. Contextul proiectului 1](#_Toc422152969)

[1.2. Domeniul temei alese 2](#_Toc422152970)

[1.2.1. Ingineria Traficului 2](#_Toc422152971)

[1.2.2. Planificarea Sistemului de Transport 3](#_Toc422152972)

[Capitolul 2. Obiectivele Proiectului 4](#_Toc422152973)

[Capitolul 3. Studiu Bibliografic 6](#_Toc422152974)

[Capitolul 4. Analiză şi Fundamentare Teoretică 11](#_Toc422152975)

[4.1. Definirea modelului 11](#_Toc422152976)

[4.1.1. Analiza unei intersecții 11](#_Toc422152977)

[4.2. Elementele mediului de simulare 15](#_Toc422152978)

[4.2.1. Elementul intersecție, puncte de intrare/ieșire și semafor 15](#_Toc422152979)

[4.2.2. Generatorul de fluxuri de mașini 16](#_Toc422152980)

[4.2.3. Generatorul de configurații pentru semafoare 16](#_Toc422152981)

[4.3. Optimizarea traficului urban 17](#_Toc422152982)

[4.3.1. Algoritmul genetic 17](#_Toc422152983)

[4.3.2. Inițializarea populației 18](#_Toc422152984)

[4.3.3. Procesul de evaluare a populației (Funcția de fitness) 19](#_Toc422152985)

[4.3.4. Procesul de selecție 19](#_Toc422152986)

[4.3.5. Crossover 21](#_Toc422152987)

[4.3.6. Procesul de mutație a genelor 22](#_Toc422152988)

[4.3.7. Oprirea procesului de optimizare 22](#_Toc422152989)

[Capitolul 5. Proiectare de Detaliu si Implementare 23](#_Toc422152990)

[5.1. Funcționalitățile sistemului 23](#_Toc422152991)

[5.2. Elementele simulatorului 24](#_Toc422152992)

[5.2.1. Clasa Intersecție 25](#_Toc422152993)

[5.2.2. Clasa Punct de intare/ieșire 26](#_Toc422152994)

[5.2.3. Clasa Segment (Tronson) 27](#_Toc422152995)

[5.2.4. Clasa Autovehicul 27](#_Toc422152996)

[5.3. Procesul de implementare 28](#_Toc422152997)

[5.3.1. Model-View-Controller 29](#_Toc422152998)

[5.3.2. Framework-ul JGAP 29](#_Toc422152999)

[5.4. Modulele aplicației 30](#_Toc422153000)

[5.4.1. Modulul principal de simulare 31](#_Toc422153001)

[5.4.2. Modulul de configurare a semafoarelor 33](#_Toc422153002)

[5.4.3. Modulul de generare a fluxurilor de mașini 35](#_Toc422153003)

[5.4.4. Modulul de configurare a algoritmului genetic 36](#_Toc422153004)

[5.5. Fazele procesului de simulare și optimizare 39](#_Toc422153005)

[5.5.1. Faza de construcție a mediului de simulare 39](#_Toc422153006)

[5.5.2. Faza de simulare 40](#_Toc422153007)

[5.5.3. Faza de optimizare 42](#_Toc422153008)

[Capitolul 6. Testare şi Validare 44](#_Toc422153009)

[Capitolul 7. Manual de Instalare si Utilizare 48](#_Toc422153010)

[Capitolul 8. Concluzii 56](#_Toc422153011)

[Bibliografie 58](#_Toc422153012)

[Anexa 1 59](#_Toc422153013)

# Introducere – Contextul proiectului

## Contextul proiectului

Simularea traficului reprezintă modelarea matematică a sistemului de transport pentru a putea planifica, proiecta și opera mai ușor traficul. Ea a început să fie studiată în urmă cu cel puțin 40 de ani și a devenit un domeniu de studiu important în rândul “*Ingineriei Traficului*” și “*Planificarea sistemului de transport*”.

Simularea este importantă deoarece poate studia modele care sunt mult prea complexe să fie studiate în mod analitic și poate fi folosită pentru diverse experimente executate pe scenarii diferite. Poate cel mai complex scenariu care merită să fie studiat și simulat este cel al traficului urban.

*Traficul congestionat* reprezintă cea mai mare problemă in zonele urbane caracterizate de o densitate mare a populației. Intersecțiile devin pline cu cozi de mașini, crescând astfel timpul de așteptare pentru fiecare autovehicul în parte. Datorită acestor cozi de autovehicule și timp crescut de așteptare se produce așa numitul fenomen de furie din trafic. Acest fenomen este caracterizat de un comportament agresiv din partea participanților la trafic care își exprimă practic nemulțumirea cu privire la situația rutieră. Chiar și mediul înconjurător este afectat din cauza blocajelor, fiind crescute emisiile gazelor toxice și consumul de combustibil de la autovehicule.

Există o relație matematică prin care se poate calcula fluxul de mașini de pe un tronson. Presupunem că *Q* reprezintă fluxul de mașini rezultat de pe tronsonul curent (numărul de autovehicule pe oră), *V* reprezintă viteza maximă de pe tronson (kilometri pe oră) și *K* reprezintă densitatea autovehiculelor de pe tronson (numărul de autovehicule pe fiecare kilometru). Astfel, ecuația dintre cele trei elemente va deveni:

*Q = K \* V*

*Semafoarele*, denumite și *semnale de control al traficului* reprezintă dispozitivele folosite pentru dirijarea fluxurilor de mașini din intersecții și alte segmente de drum, precum și a pietonilor care vor să traverseze un segment de drum. Semafoarele folosesc trei culori standard pentru gestionare: *roșu*, *galben* și *verde*. Culoarea roșie împiedică orice fel de trafic să se deplaseze pe direcția curentă de mers. Culoarea verde permite participanților la trafic să se deplaseze pe direcția de mers în cazul în care ea este sigură. Culoarea galbenă semnifică tranziția semaforului de la culoarea verde la culoarea roșie sau invers, în anumit cazuri. Rolul acestei culori este de a atenționa participanții din trafic că urmează o tranziție.

Identificarea timpilor optimi pentru fiecare fază a unui semafor reprezintă o problemă importantă studiată de către inginerii de trafic. Există câte un controller care se ocupă de gestionarea semafoarelor dintr-o intersecție. Fiecare controller are propriul mecansim prin care furnizeză semafoarelor diferite configurații pe baza stării curente a intersecției. Câteva din metodele folosite sunt: detectarea autovehiculelor care trec pe un anumit segment de drum prin dispozitive inductive îngropate sub acele segmente, detectarea folosind camere video și o metodă de pixelare. Însă, majoritatea semafoarelor au configurațiile deja făcute și nu folosesc nici o metodă de detecție a traficului.

Există trei tipuri majore de control al traficului:

* *Caotic*
* *Pre-temporizat*
* *Sistem de trafic adaptiv*

Primul tip de control al traficului presupune să avem timpii pentru fazele semafoarelor fixe pe durata întregii zile. Modul caotic reprezintă cel mai prmitiv mod de gestionare al traficului.

Al doilea mod de control al traficului este puțin mai avansat față de cel precedent și presupune dezvoltarea unui plan de timp. Prin aceasta se vor asigna timpi diferiți pentru fiecare fază a semafoarelor în funcție de momentul zilei. Timpurile care se vor asigna vor rezulta dintr-o simulare a scenariului propriu-zis cu diferiți timpi căutând astfel configurația mai eficientă a fazelor. Un exemplu de scenariu pe care am putea să îl simulăm ar putea fi o oră de vârf din trafic când de obicei șoferii se indreaptă către lucru sau către casă. Aceaste ore de vârf apar în majoritatea cazurilor în intervalele de timp 07:00 – 09:00 și 17:00 – 19:00.

Ultimul mod de control al traficului este dependent de stare. Acesta poate configura instatant timpii pentru fiecare fază a semafoarelor în funcție de fluxul care tocmai străbate intersecția. Datele despre fluxuri sunt colectate de către senzori care la rândul lor transmit mai departe informația către un controller care translatează datele într-un set de timpi corespunzători pentru fazele semaforului. Acest mod de control al traficului este puțin mai costisitoare din cauza sistemului de captare a datelor și modelare/translatare a informației. Senzorii care vor prelua datele din trafic trebuie amplasați în fiecare intersecție lângă fiecare semafor. Controllerul care va prelua informațiile de la senzori va putea fi amplasat la nivelul unei intersecții sau o mulțime de intersecții.

Alegerea modului de control al traficului este o problemă majoră in domeniul ingineriei traficului. În lucrarea de față modul de control al traficului ales este cel *pre-temporizat* prin care se simulează diferite configurații ai parametrilor de timpi de la semafoare în contextul a mai multor scenarii.

## Domeniul temei alese

Tema aleasă se încadrează în două mari domenii care se ocupă de simularea traficului. Acestea sunt: *Ingineria Traficului* (Traffic Engineering) și *Planificarea sistemului de transport* (Transportation planning).

### Ingineria Traficului

*Ingineria traficului* (Traffic Engineering) este o ramură a ingineriei civile și se folosește de așa numitele tehnici inginerești pentru a avea ca și rezultat un flux de trafic cât mai sigur și eficient. Aceste tehnici sunt folosite pentru controlul semafoarelor din intersecții, adăugarea unor indicatoare suplimentare sau a unor marcaje pe tronsoane. Gestionarea traficului din intersecții poate fi aplicat pe termen scurt, în cazul unor construcții sau rute ocolitoare aplicate pe diferite sectoare de drum sau pe termen lung, atunci când se cunosc zonele cu traficul cel mai intens.

În prezent, există diverse elemente dinamice care sunt introduse în trafic pentru monitorizarea ei. Aceste elemente pot fi, de exemplu senzori pentru măsurarea fluxurilor de mașini de pe diferite segmente de drum sau indicatoare automatizate care dirijează traficul într-o anumită direcție. Toate aceste sisteme fac parte dintr-un *sistem inteligent de transport* (Intelligent transportation system). Acest tip de sistem are ca și obiectiv principal furnizarea de servicii despre diferite moduri de transport și de gestionare a traficului.

Acest domeniu se ocupă și de *ingineria siguranței de pe autostrăzi*, care reprezintă la rândul său o ramură a ingineriei traficului. Acestă ramură se ocupă de reducerea frecvenței cu care se produc accidentele de pe autostrăzi. Folosește fizica și dinamica autovehiculelor, totodată ia în considerare și factorul psihologic a conducătorilor auto pentru a putea reduce accidentele. Există chiar și un domeniu care se ocupă de factorii psihologici care stau la baza conducătorilor auto care se numește *ingineria factorilor umani*.

Porcesul care stă la baza ingineriei siguranței de pe autostrăzi va fi descris pe scurt în continuare. Primul pas din acest proces ține de identificarea și prioritizarea zonelor de investigat. Se vor lua în calcul zonele care prezintă cea mai mare rată de accidente. În pasul al doilea se vor colecta date care cuprind informații despre semafoare, starea drumurilor, rapoarte de accidente ale poliției și indicatoare. Pasul al treilea constă din analiza propriu-zisă a datelor culese și mai exact identificarea coliziunilor sau a condițiilor de drum care ar putea contribui la accidente. În pasul al patrulea se încearcă găsirea unor posibile măsuri de evitare a acestor evenimente neplăcute din trafic. Pasul al cincelea reprezintă implementarea soluțiilor găsite. În ultimul pas se evaluează rezultatele soluției implementate și în cazul în care frecvența numărului de accidente nu a fost redusă se reia pasul doi din acest proces.

### Planificarea Sistemului de Transport

*Planificarea sistemului de transport* (Transportation planning) se ocupă de evaluarea, proiectarea și amplasarea mijloacelor de transport (acestea includ străzi, autostrăzi și linii pentru transportul în comun). Înainte de planificarea propriu-zisă trebuie definite goluri și obiective, trebuie identificare problemele, eventualele alternative și într-un final elaborarea unui plan. Planificarea trebuie să țină cont de mai multe aspecte provenite din diverse domenii, cum ar fi cel al protecției mediului. În prezent, multe planuri sunt elaborate având la bază protecția mediului înconjurător și folosirea transportului în comun, în favoarea autovehiculelor proprii.

# Obiectivele Proiectului

**Obiectivul proiectului este realizarea unei aplicații de modelare, simulare și optimizare a traficului într-un context urban.**

În continuare se vor enumera obiectivele principale ale proiectului:

* *Modelarea traficului urban (fluxuri de mașini și elemente de control al traficului)*
* *Simularea traficului urban (pașii de simulare, definirea unui timp discret, procesul de generare a autovehiculelor, schimbarea fazelor unui semafor)*
* *Vizualizarea grafică a procesului de simulare*
* *Căutarea unei soluții suboptimale pentru optimizarea traficului (algoritmul genetic)*
* *Culegerea și afișarea rezultatelor*
* *Vizualizarea rezultatelor în formă grafică și numerică*
* *Definirea și calcularea unor metrici de evaluare a calității traficului*
* *Interfețe pentru definirea configurațiilor (configurare fluxuri de mașini, configurare faze semafoare, configurare parametri algoritm genetic)*

Modelarea traficului urban reprezintă pasul prin care se încearcă maparea traficului urban într-un format obiectual cât mai apropiat de realitate. În vederea modelării traficului trebuie luate în considerare două elemente ce joacă un rol important în trafic, mai exact *fluxul de mașini* și *elementele de control al traficului*.

Un flux de mașini reprezintă studiul interacțiunilor dintre autovehicule, șoferi și infrastructură și dezvoltarea unei metode optimale de deplasare în trafic, precum și minimizarea cozilor de autovehicule.

Elementele de control al traficului denumite și semafoare reprezintă dispozitivile prin care întreg traficul poate să fie dirijat.

Simularea traficului urban presupune la rândul său definirea unor aspecte importante: *pașii de simulare*, *definirea unui timp discret*, *procesul de generare a autovehiculelor*, *schimbarea fazelor unui semafor*.

La fiecare pas de simulare se vor executa mai multe verificări pentru fiecare autovehicul din mediul de simulare. Aceste verificări pot include: verificarea existenței unui obstacol în cale, accelerație sau frânare, verificarea stării semaforului pe direcția curentă de mers, verificarea rutei.

Pentru simularea în mai mulți pași s-a definit un timp discret de simulare. Prin definirea acestui timp s-a încercat o metodă de modelare cât mai apropiată de realitate. Timpul discret ales este de o secundă pentru fiecare pas din simulare.

Generarea autovehiculelor reprezintă procesul prin care autovehiculele sunt adăugate în simulare. Această generare are loc la timpi definiți încă din procesul de configurare. În fiecare pas de simulare se verifică dacă s-a ajuns la un timp în care se poate adăuga o nouă mașină în cadrul simulării.

Schimbarea fazelor semafoarelor din intersecții are loc la timpi bine definiți încă din procesul de configurare a simulării. La fiecare pas de simulare se verifică dacă faza curentă a semaforului a ajuns la sfârșit și în caz afirmativ se execută trecerea la următoarea fază din ciclu.

Vizualizarea grafică a procesului de simualare ne permite vizualizarea în timp real a întregului proces de simulare. Datorită acestei părți se vor putea observa mișcările autovehiculelor de pe tronsoane pe direcțiile lor de mers, schimbarea fazelor semafoarelor din intersecții, generarea mașinilor, pătrunderea lor în mediul de simulare și părăsirea simulării după atingerea destinației. Fiecare intersecție va putea avea câte două tronsoane pe toate cele patru direcții de mers. Fiecare tronson va permite autovehiculelor deplasarea pe o singură direcție.

Căutarea unei metode suboptimale de eficientizare a traficului urban reprezintă pasul prin care se înceracă găsirea unei soluții prin care timpul mediu de așteptare a autovehiculelor din intersecții să fie cât mai mic. Nu există o modalitate prin care soluția optimă va fi identificată. Prin folosirea algoritmului genetic se va putea identifica o soluție apropiată de cea optimă, fiind importantă alegerea parametrilor algoritmului.

Pe tot parcursul simulării se vor colecta date despre diferiți metrici ale autovehiculelor, intersecțiilor și fluxurilor de mașini. Toate informațiile colectate pe parcursul simulării vor putea fi vizualizate într-un mod grafic și numeric. Cu ajutorul acestor statistici se vor putea identifica scenariile de simulare și configurațiile prin care anumiți metrici au fost îmbunătățiți.

Definirea configurațiilor va fi posibilă prin interfețe grafice. Elementele ce vor putea fi configurate vor fi: *fluxurile de autovehicule, semafoarele din intersecții* și *parametri metodei de optimizare*.

Interfața de configurare a fluxurilor de autovehicule va conține două metode de definire a unui flux. Prima metodă de definire a unui flux ne va permite definirea unor fluxuri de mașini care au un punct de pornire, după care fiecare mașină are o destinație aleatoare. Prin această metodă se va putea simula scenariul din viața reală în care mai multe autovehicule au un punct comun de intrare în oraș după care fiecare din aceștia își urmează propria sa rută spre casă. A doua metodă de definire unui flux ne permite specificarea unei rute statice pentru un anumit flux. Prin această metodă se va putea simula scenariul din viața reală prin care mai multe autovehicule au un punct comun de intare în oraș și continuă să se deplaseze pe aceeași rută în continuare, simulând astfel comportamentul unui trafic pe o centură.

Interfața de definire a semafoarelor din intersecții ne va permite configurarea fazelor semafoarelor din fiecare intersecție. Fazele tuturor semafoarelor vor fi descrise în acest modul de configurare. Se vor putea vizualiza ordinea fazelor, precum și configurația stărilor semafoarelor în fiecare fază. Valorile ce vor putea fi configurate vor fi timpii pentru faze exprimați în secunde.

Interfața de configurare a paramtetrilor metodei de optimizare va conține totalitatea valorilor care pot fi modificate în ceea ce privește algoritmul genetic.

# Studiu Bibliografic

În articolul **[1]** autorii prezintă o modalitate de optimizare a traficului urban utilizând un algoritm numit ”*reinforcement learning algorithm*”. Datorită acestui algoritm se rețin timpii de așteptare ai autovehiculelor pentru culorile roșu și verde din fiecare intersecție, după care se setează toată configurația timpilor bazându-se pe aceste valori. Controller-ul folosit va înregistra locațiile autovehiculelor din jurul intersecțiilor înainte ca acestea să ajungă acolo.

Algoritmul folosit în acest articol pentru optimizare funcționează astfel: Agentul primește ca și intrări starea mediului înconjurător și se folosește de funcțiile sale definite intern pentru a lua decizii. Există o funcție de evaluare care va analiza deciziile luate de agent și va returna o valoare în funcție de aceasta. Golul agentului va deveni astfel găsirea unei decizii cu privință la mediul actual care să satisfacă cel mai bine această funcție de evaluare.

Mediul înconjurător conține mai multe stări *s ∈ {S1,...,Sn}*, iar în cazul în care considerăm *m* fiind numărul total de mașini, precum *n* numărul total al stărilor din mediul înconjurător, vor rezulta *nm* configurații posibile pentru trafic. Funția care va analiza dacă decizia luată de către agent este cea mai eficintă va fi mai degrabă o funcție de cost, prin care vom încerca să minimizăm timpii de așteptare a autovehiculelor din intersecții. Astfel, funcția va returna un cost 1 dacă agentul trebuie să aștepte: *R(s, s) = 1*, iar în cazul în care agentul se poate deplasa va returna un cost 0: *R(s, s) = 0*.

Pentru optimizare se va folosi funcția *Q(s, l)* care reprezintă un autovehicul fiind într-o stare, unde *s = [p, d]*, *p* fiind poziția curentă și *d* destinația autovehiculului, iar *l* reprezentând cele două culori posibile ale semaforului: *red* și *green*:

*Q(s, red) = R(s, s) + γV (s)=1+ γV (s)*

și

*Q(s, green) =*

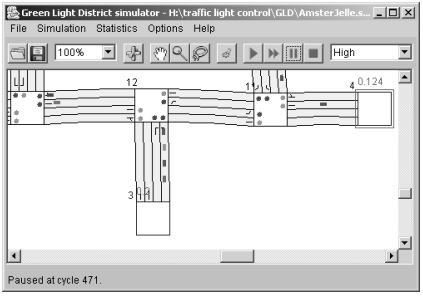
*P(s, s’)* reprezintă probabilitatea ca un autovehicul să treacă la o stare următoare *s’*. Valoarea lui *γ* va fi: *γ < 1* și va transforma funcția *Q* într-o funcție finită. În continuare avem funcția *V(s)*, care repreintă timpul mediu de așteptare a unui autovehicul până la destinație:

*V (s) = P(green|s) Q(s, green) + P(red|s) Q(s, red)*

unde *P(l|s)*, cu *l ∈ {red, green}* reprezintă probabilitatea ca semaforul să fie pe verde sau pe roșu.

Dacă luăm timpii tutror mașinilor care stau la coadă din fiecare intersecție vom obține o decizie locală optimă *A* care va seta semafoarele *i* pe verde.

Datorită faptului că fiecare agent are acces la timpii de așteptare din intersecții, aceștia îșî pot selecta ruta pe care doresc să circule până ajung la destinația prestabilită. Fiecare autovehicul va consulta funcția *V* pentru a selecta tronsonul următor.

*Figura 3.1* reprezintă interfața grafică a simulatorului. Scenariul reprezentat în simulator constă din două noduri terminale, trei intersecții cu semafoare și drumurile dintre ele cu autovehicule în mișcare.

*Figura 3.1 Interfață grafică simulator (articolul [1])*

Simulatorul implementat de către autori are posibilitatea să analizeze diferite scenarii din trafic, având posibilitatea desenării scenariilor. Acest simulator dipune de două tipuri de agent: autovehicul și semafor. Fiecare intersecție este controlată de către un controller de semafoare (*TLC*) care ia decizii asupra configurațiilor celor mai optime.

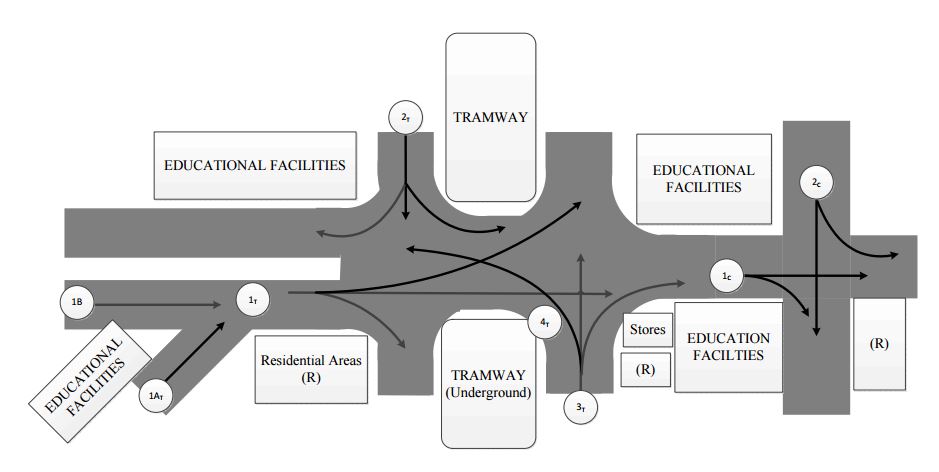
Avantajele acestei metode reies din faptul că, controllerul va calcula timpii de așteptare din intersecții înainte ca autovehiculele să ajungă acolo. Astfel, agenții vor putea lua decizii din timp fiind astfel eficientizată mișcarea lor.

Un dezavantaj al metodei ar putea fi faptul că se folosesc multe ecuații matematice, care pot deveni tot mai complexe pe măsură ce numărul intersecțiilor și agenților cresc. Totodată, punerea în practică a metodei ar însemna instalarea controller-elor între intersecțiile din oraș care doresc să fie optimizate. Această operațiune ar putea fi costisitoare.

Prin acestă metodă de optimizare autorii au reușit să reducă timpul mediu de așteptare cu mai mult de 25% în cazul unui trafic aglomerat.

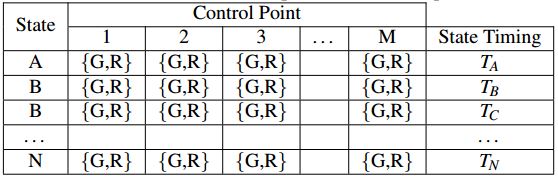
În articolul **[2]** autorii prezintă un mediu de simualare pentru traficul urban și o posibilă metodă de optimizare prin simularea diferiților timpi ai semafoarelor din intersecții.

Sistemul studiat de către autori a fost o rețea de drumuri care conține două intersecții mari semnalizate din Alexandria, Egipt. Una din intersecții are și o linie de tramvai. Autorii au ales acest scenariu deoarece reprezintă un punct important în care traficul devine congestionat. În *figura 3.2* putem observa acest sistem.



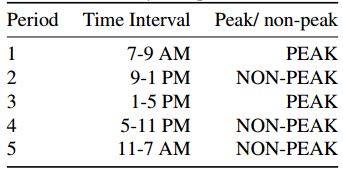
*Figura 3.2 Scenariul studiat (Alexandria, Egipt) [2]*

În continuare au fost definite stările semafoarelor din intersecții si au fost determintate secvențele acestor stări. *Tabela 3.1* reprezintă stările semafoarelor, secvențele dintre stări și, totodată timpii stărilor. După cum se vede din tabelă, există două stări posibile pentru un punct de control: *green (G)* și *red (R)*. Fiecare stare este păstrată pentru un anumit timp *T*.



*Tabel 3.1 Forma generală a tranzițiilor stărilor [2]*

Pentru a avea o simulare cât mai eficientă, autorii au împărțit ziua în intervale, în funcție de trafic. Astfel, intervalele orare 7-9 AM, 1-5 PM au fost considerate ore de vârf, în timp ce restul intervalelor orare au fost considerate ore cu trafic normal. În *tabela 3.2* putem observa aceste valori.



*Tabel 3.2 Tipul traficului pe intervale orare [2]*

Optimizarea semafoarelor a fost executată folosind mecanismul de optimizare evoluționistă a sistemului *ExtendSim*. Variabilele de decizie folosite în procesul de optimizare sunt timpii stărilor () pentru fiecare punct de control *i* din fiecare intersecție *k* și timpii ciclurilor (*C*) din fiecare intersecție *k*.

unde *k* reprezintă identificatorul intersecției din rețea, *i* reprezintă indexul stării intersecției, *j* reprezintă indexul punctului de control, *N* reprezintă numărul stărilor din intersecție și *M* reprezintă numărul maxim de puncte de control.

Avantajul metodei descrise de către autori constă în faptul că timpii fiecărei intersecții se pot configura împreună cu fazele care vor exista.

Unul din dezavantajele metodei este faptul că se folosește un tool extern (*ExtendSim*), care are propriul mecanism de optimizare implementat nefiind accesibil de către utilizatori. Totodată, simularea a fost executată pe un singur scenariu, cel din *Alexandria, Egipt*, fiind destul de greu să se tragă concluzii concrete de pe urma rezultatelor.

În articolul **[3]** autorul prezintă un model de trafic optim pentru simulare. Modelul ales de către acesta este cel de transmisie celulară (*cell-transmission model*) *CTM* folosit în simulările macroscopice.

Simularea traficului la nivel macroscopic tratează autovehiculele la nivel de densitate (număr de autovehicule pe kilometru), flux (număr de autovehicule care trec pe tronson pe secundă) și nu la nivel individual. Avantajul simulării macroscopice față de cea microscopică este viteza și numărul de scenarii care se pot analiza.

Folosind modelul transmisiei celulare, rețeaua de transport devine împărțită în mai multe celule, fiecare din acestea reprezentând câte o secțiune din drum. O astfel de celulă are asociată diverse atribute fizice, cum ar fi: densitatea *k* (autovehicule/m), densitatea maximă , viteza fluxului *v* (m/s), fluxul autovehiculelor *q* (autovehicule/s), fluxul maxim de autovehicule . Relația dintre viteză *v*, densitate *k* și flux *q* este în mod natural *q = v \* k*.

Există mai multe politici de control al fazelor semafoarelor din intersecții. Fazele semafoarelor trebuie să asigure un flux sigur al autovehiculelor fără posibile accidente. Politicile definite de către autor sunt:

* *Politică aleatoare de control* – după cum sugerează și numele fazele semafoarelor vor fi interschimbate într-un mod aleator având totodată timpi aleatori pe fiecare fază;
* *Politică cu timpi fixi și offset* – este politica cea mai utilizată în practică prin care se asignează o serie de timpi fixi pentru o intersecție principală A și un offset pentru o intersecție secundară B. Offset-ul respectiv va reprezenta timpul în care intersecția B va trebuie să treacă pe verde asigurând astfel o deplasare continuă a fluxului de autovehicule;
* *Politică locală adaptivă* – folosește senzori pentru detectarea tronsoanelor cu traficul cel mai intens și un controller pentru furnizarea configurației optime pentru fazele semafoarelor.

Avantajul acestei metode este reprezentat de către modelul ales de către autor, *cell-transmission model*, prin care se poate face o modelare macroscopică apropiată de realitate a fluxurilor de autovehicule din mediul urban.

Dezavanatjul metodei alese constă în faptul că nu a fost aleasă o metodă de optimizare a traficului, fiind tratată doar partea de modelare.

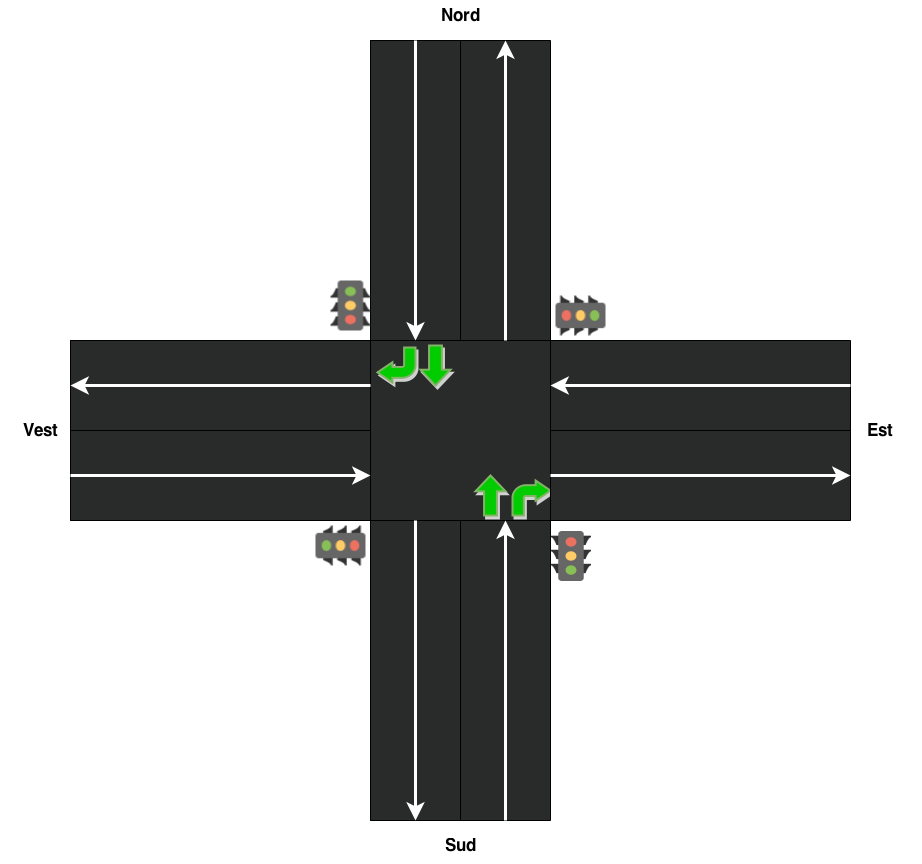
# Analiză şi Fundamentare Teoretică

## Definirea modelului

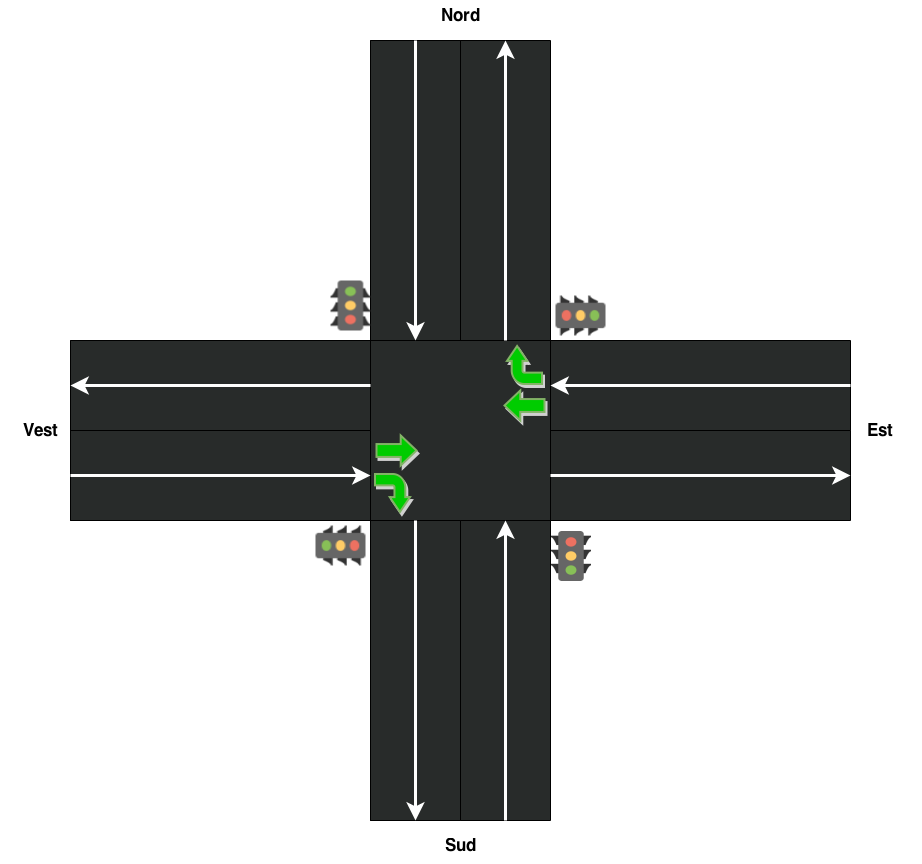
Primul pas în dezvoltarea oricărui sistem îl reprezintă definirea unui model cât mai corect. Acest model trebuie să prezinte un comportament similar sau cel puțin apropiat cu cel din viața reală. Doar în urma definirii acestui model putem începe o simulare apropiată de realitate.

În cazul mediului de simulare trebuie să ne definim un model pentru un scenariu de gestionare a diferiților parametri a semafoarelor din intersecții. Semafoarele trebuie să permită participanților la trafic să se deplaseze pe toate direcțiile de mers (*stânga*, *înainte* și *dreapta*).

### Analiza unei intersecții

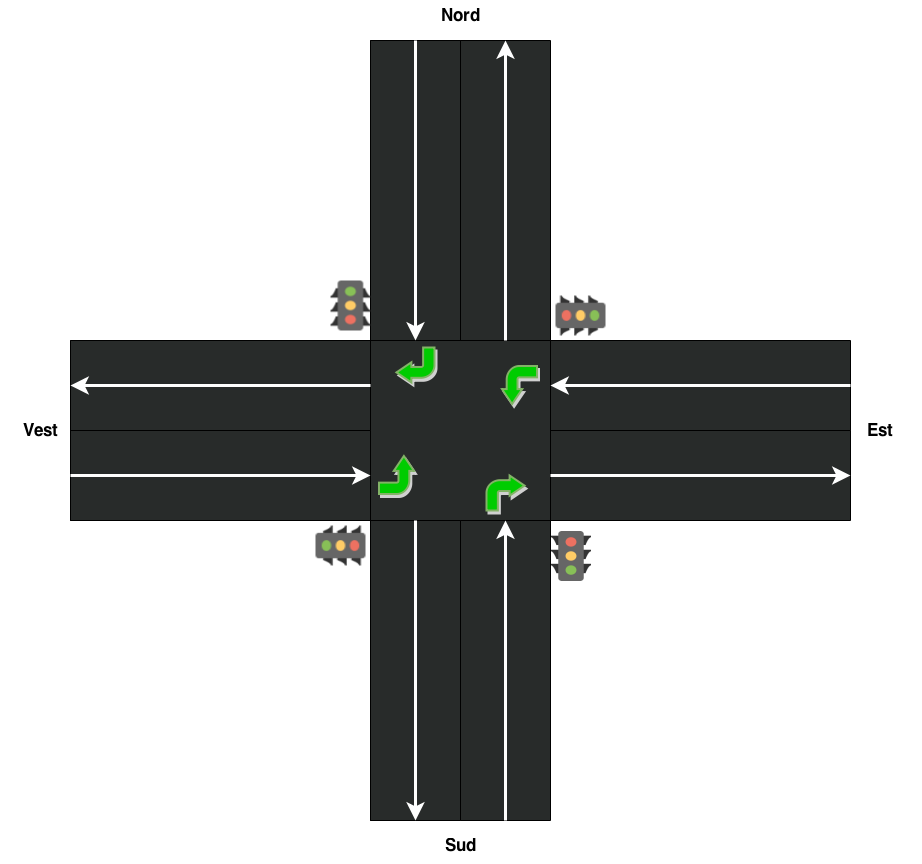
Pentru definirea modelului a fost mai întâi studiată o intersecție din viața reală prin care traficul devine mai intens la diferite intervale orare. Intersecția aleasă conține patru tronsoane cu direcții de mers pe ambele sensuri. Numărul de semafoare din intersecție este tot de patru, fiecare având toate cele trei direcții de mers. În primul rând au fost studiate numărul fazelor semafoarelor și timpii pe fiecare fază.

*Figura 4.1 Prima fază*

În *figura 4.1* este prezentată prima fază a celor patru semafoare din intersecție. După cum se vede și în figură, direcțiile de mers pe care semaforul este verde sunt: *Sud → Nord*, *Sud → Est*, *Nord → Sud* și *Nord → Vest*. Pe restul direcțiilor de mers semaforul va fi roșu.

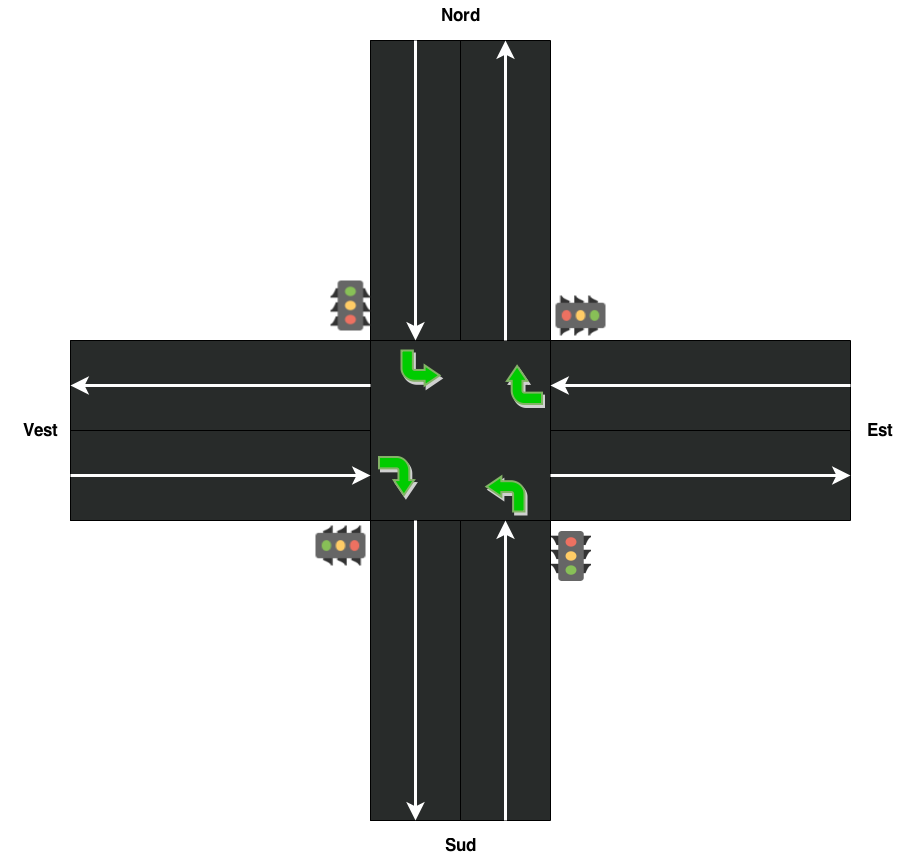
*Figura 4.2 A doua fază*

În *figura 4.2* este prezentată a doua fază a celor patru semafoare din intersecție. După cum se vede și în figură, direcțiile de mers pe care semaforul este verde sunt: *Est → Nord*, *Est → Vest*, *Vest→ Sud* și *Vest → Est*. Pe restul direcțiilor de mers semaforul va fi roșu.



*Figura 4.3 A treia fază*

În *figura 4.3* este prezentată a treia fază a celor patru semafoare din intersecție. După cum se vede și în figură, direcțiile de mers pe care semaforul este verde sunt: *Sud → Est*, *Est → Sud*, *Nord→ Vest* și *Vest → Nord*. Pe restul direcțiilor de mers semaforul va fi roșu.



*Figura 4.4 A patra fază*

În *figura 4.4* este prezentată ultima fază, cea de-a patra a celor patru semafoare din intersecție. După cum se vede și în figură, direcțiile de mers pe care semaforul este verde sunt: *Sud → Vest*, *Est → Nord*, *Nord→ Est* și *Vest → Sud*. Pe restul direcțiilor de mers semaforul va fi roșu.

|  |  |
| --- | --- |
| Numărul fazei | Durată (secunde) |
| 1 | 30 |
| 2 | 30 |
| 3 | 30 |
| 4 | 30 |

*Tabel 4.1 Timpii fazelor*

În *tabela 4.1* sunt specificați timpii petru fiecare fază din intersecția analizată. Conform tabelei durata totală a fazelor este de 120 de secunde, adică 2 minute. Acești timpi răman neschimbați indifererent de starea traficului, fie aceea una normală sau în care se produce o aglomerare a autovehiculelor.

|  |  |
| --- | --- |
| Interval orar | Tip trafic |
| 07:00 – 09:00 | Oră de vârf |
| 09:00 – 12:00 | Trafic normal |
| 12:00 – 14:00 | Oră de vârf |
| 14:00 – 16:00 | Trafic normal |
| 16:00 – 19:00 | Oră de vârf |
| 19:00 – 07:00 | Trafic normal |

*Tabel 4.2 Tipul traficului pe intervale orare*

În *tabela 4.2* avem o împărțire pe intervale de timp a unei zile obișnuite de lucru, precum și tipul de trafic prezent în acel moment. Astfel, intervalele de timp în care se vor produce cele mai multe cozi de așteptare în cadrul intersecțiilor vor fi dimineața între orele 07:00 și 09:00, atunci când majoritatea se deplasează către locurile de muncă sau școli, pauza de masă dintre orele 12:00 și 14:00 și nu în utlimul rând seara în intervalul 16:00 – 19:00, atunci când majoritatea se întorc de la locurile de muncă.

## Elementele mediului de simulare

În urma definirii unui model corect pentru mediul de simulare vor rezulta câteva elemente abstracte care vor fi incluse în simulator. Aceste elemente vor fi după cum urmează:

* *Intersecție*
* *Segment (Tronson)*
* *Semafor*
* *Punct de intrare în simulare*
* *Punct de ieșire din simulare*
* *Autovehicul*
* *Generator flux de mașini*
* *Generator configurații semafoare*
* *Configurator optimizator*

### Elementul intersecție, puncte de intrare/ieșire și semafor

Fiecare intersecție va conține patru semafoare cu trei direcții de mers: *înainte*, *stânga* și *dreapta*. Intersecțiile care conțin cel puțin trei tronsoane vor avea patru faze.

Semafoarele vor funcționa pe două faze: *roșu* (autovehiculele așteaptă) și *verde* (autovehiculele se deplasează pe direcția de mers).

Fazele semafoarelor din intersecții au fost definite după cum urmează:

* Faza 1: *Sud -> Nord, Sud -> Est*, *Nord -> Sud*, *Nord -> Vest*
* Faza 2: *Est -> Vest*, *Est -> Nord*, *Vest -> Est*, *Vest -> Sud*
* Faza 3: *Sud -> Est*, *Est -> Sud*, *Nord -> Vest*, *Vest -> Nord*
* Faza 4: *Sud -> Vest*, *Vest -> Sud*, *Nord -> Est*, *Est -> Nord*

Punctele de intrare în simulare vor fi locațiile din care autovehiculele vor pătrunde în mediul de simulare. Acestea vor fi punctele de pornire din ruta fiecărui autovehicul.

Punctele de ieșire din simulare vor fi locațiile prin care autovehiculele vor părăsi mediul de simulare. Aceste puncte vor reprezenta destinațiile participanților la trafic.

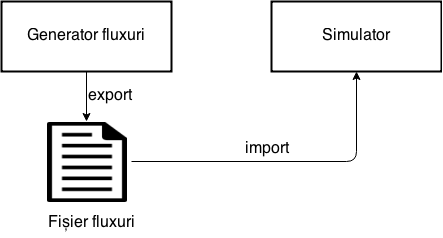
### Generatorul de fluxuri de mașini

Generatorul de fluxuri de mașini va reprezenta modulul prin care utilizatorul își va putea defini diferite fluxuri de autovehicule care vor pătrunde în mediul de simulare prin punctele de intrare. Există două tipuri de flux de mașini: *flux de mașini cu rută fixă* și *flux de mașini cu punct de pornire*.

Un flux de mașini cu rută fixă are specificată întreaga sa rută, de la un punct de pornire până la o destinație fixă.

Fluxul de mașini cu un punct de pornire are specificată doar punctul din care fiecare mașină din flux va porni, urmând ca fiecare autovehicul să își identifice o destinație aleatoare.

Acest generator de fluxuri va furniza un fișier către simulator care va conține toate detaliile legate de fluxurile definite de către utilizator.



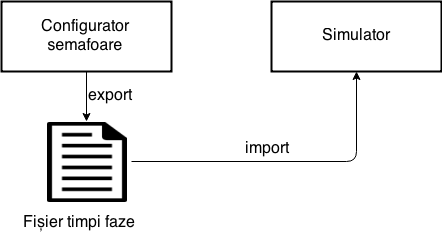
*Figura 4.5 Generator de fluxuri de mașini*

*Figura 4.5* reprezintă schema generatorului de fluxuri de mașini. Se pot observa metodele de *import* și *export* executate și fișierul rezultat în urma exportului din generator.

### Generatorul de configurații pentru semafoare

Generatorul de configurații pentru semafoare reprezintă modulul prin care utilizatorul va putea să își definească timpii pentru fiecare fază din intersecții. Totodată, prin acest modul vor putea fi vizualizate toate configurațiile de semafoare pentru fiecare fază în parte.

Acest generator va furniza un fișier către simulator care va conține toate detaliile legate de configurarea semafoarelor definite de către utilizator.



*Figura 4.6 Generator de configurații pentru semafoare*

*Figura 4.6* reprezintă schema generatorului de configurații pentu semafoare. Se pot observa metodele de *import* și *export* executate și fișierul rezultat în urma exportului din generator.

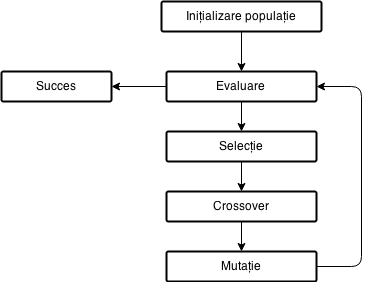
## Optimizarea traficului urban

Procesul de optimizare a traficului urban reprezintă o problemă complexă datorită numărului mare de parametri care apar în trafic. Datorită acestui aspect s-a încercat găsirea unei metode prin care soluția optimizării să fie una apropiată de cea optimă. Metoda de optimizare aleasă pentru acest caz este integrarea *algoritmului genetic*.

### Algoritmul genetic

Orice organism are un set de reguli prin care este descris modul în care acel organism este conturat. Aceste reguli sunt codificate în *gene* care la rândul lor, grupate formează șiruri lungi numite *cromozomi*. Fiecare geană reprezintă o însușire specifică fiecărui organism (de exmplu: culoarea ochilor, culoarea părului) și are numeroase variații. Ca și exemplu, culoarea părului poate fi blond, șaten, roșcat. Genele împreună cu variația lor sunt numite *genotipul* organismului. Atunci când două organisme se unesc ele au aceleași gene. Rezultatul unirii a două organisme s-ar putea sa aibă o jumătate din genele unui părinte, iar cealaltă jumătate de la genele celuilalt părinte. Acest proces poartă denumirea de recombinare.

Pe baza acestor principii genetice și ale selecției naturale enunțate de Darwin (”*supraviețuiește cel care e cel mai bine adaptat*”) funcționeză și algoritmul genetic.

Algoritmul genetic reprezintă un algoritm complex de căutare euristică care funcționează printr-un proces de selecție naturală. Acești algoritmi reprezintă o aplicație a inteligenței artificiale și cel mai adesea sunt folosiți în probleme de optimizare, planificare ori căutare. Soluția unei optimizări cu algoritmi genetici va fi una apropiată de cea optimă și nu cea optimă.

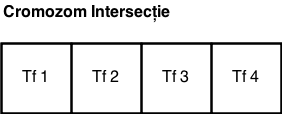
*Figura 4.7 Pașii algoritmului genetic*

În *figura 4.7* sunt reprezentați pașii efectuați de algoritmul genetic. În continuare se vor prezenta pașii în detaliu.

### Inițializarea populației

În primul pas din acest proces se va inițializa populația de cromozomi. Fiecare din acești cromozomi va avea o codificare a genelor proprie generată într-un mod aleator. Mărimea populației depinde de mărimea spațiului de căutare și timpul efectiv de evaluare a fiecărui individ. În general mărimea populației va fi undeva între *50* până la *500* de cromozomi. Cu cât numărul de indivizi dintr-o populație este mai mare, cu atât îtreg procesul de optimizare durează mai mult.

În cazul mediului de simulare, o intersecție va reprezenta un cromozom. Intersecția este compusă din patru semafoare având în total patru faze.



*Figura 4.8 Cromozom Intersecție*

În *figura 4.8* avem reprezentarea unui cromozom de intersecție. Cromozomul este compus din patru gene. Fiecare geană este o valoare întreagă și reprezintă timpul pentru o fază din cele patru existente, în cadrul intersecției. Astfel, *Tf1* reprezintă timpul primei faze din intersecție, *Tf2* reprezintă timpul fazei doi din intersecție, *Tf3* reprezintă timpul fazei trei din intersecție și *Tf4* reprezintă timpul fazei patru din intersecție.

C:\Users\edWin\Desktop\LicentaGIT\trunk\popCROMO.pngGenele pot avea valori întregi cuprinse în intervalul *[5, 30]*. Valoarea minimă aleasă reprezintă timpul minim necesar unui autovehicul să traverseze o intersecție, pe când timpul maxim ales reprezintă timpul maxim pentru care o fază poată să fie activă în cadrul unei intersecții.

*Figura 4.9 Exemplu de populație*

*Figura 4.9* reprezintă un exemplu simplu de o populație având 4 indivizi.

### Procesul de evaluare a populației (Funcția de fitness)

Procesul de evaluare denumit și funcție de fitness va determina cât de bun este un cromozom comparat cu restul indivizilor din cadrul populației. Acest pas reprezintă cea mai importantă parte a algoritmului genetic. În cazul în care funcția de evaluare nu este corectă, algoritmul va produce soluții eronate.

Funcția de evaluare a indivizilor nu va putea returna o valoare booleană (*True* sau *False*) deoarece acestă valoare trebuie să fie una comparabilă. De cele mai multe ori se folosesc valori întregi, care pot reprezenta probabilități sau statistici ale soluției curente.

Pentru problema optimizării traficului s-a ales ca și valoare comparativă între două configurații de semafoare media timpilor de așteptare a tuturor autovehiculelor care au străbătut cu succes întreaga lor rută de la punctul de pornire până la destinație. Deoarece funcția de selectare, a algoritmului genetic va alege indivizii cu valorea de *fitness* cea mai mare, această funcție va returna diferența dintre un număr mai mare și această medie de așteptare. Prin această metodă vom putea compara cu succes două configurații de timpi de fază, având la sfârșit o configurație apropiată de cea optimă.

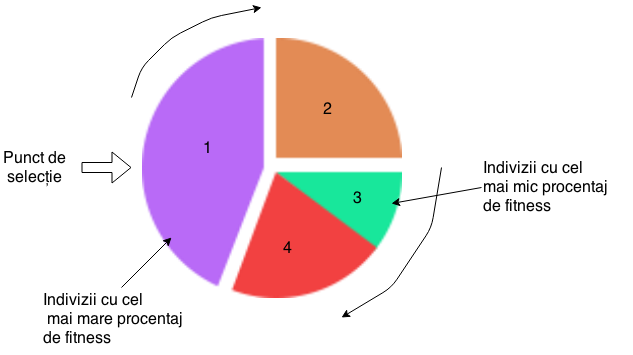
*Ecuația (1)* reprezintă media timpilor medii de așteptare a mașinilor care au terminat simularea. Astfel, *n* reprezintă numărul mașinilor care au terminat simularea, iar *Ta* reprezintă timpul de așteptare a mașinilor în intersecții.

### Procesul de selecție

Procesul de selecție a părinților candidați poate fi executat în mai multe moduri. Câteva din aceste moduri sunt: *Selecție tip ruletă (Roulette Wheel Selection)* și *Selecție turneu (Tournament Selection)*.

Primul mod este bazat pe algoritmul de selecție tip ruletă (*roulette wheel*). Acest algoritm va selecta într-un mod aleator un cromozom bazat pe funcția sa de *fitness*. Aceasta metodă poate produce un rezultat într-un timp destul de îndelungat.

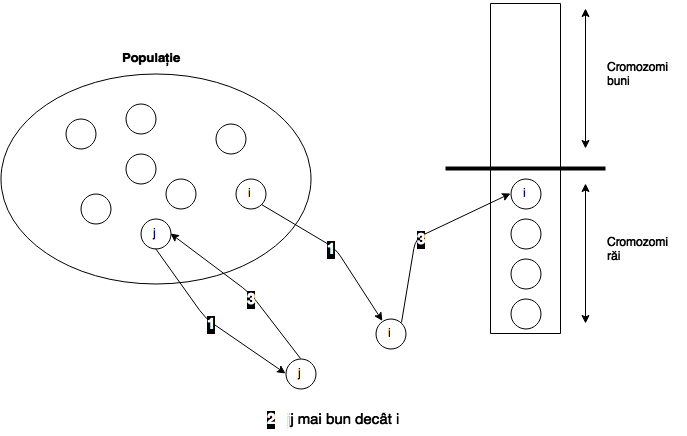
*Figura 4.10 Selecție tip ruletă*



*Figura 4.10 Selecția tip ruletă*

În *figura 4.10* putem observa reprezentarea modului de selecție folosind algoritmul de selecție tip ruletă. Putem observa faptul că indivizii cu valoare de firness mai mare sunt dominanți, ceea ce duce la o mai mare probabilitate de selcție în rândul lor.

Al doilea mod de selecție, cel bazat pe *turneu* va realiza mai multe turnee între diferiți indivizi aleși în mod aleator din populație. Câștigătorul fiecărui turneu, cel cu valoarea de fitness cea mai mare, va fi selectat mai departe pentru etapa de crossover. Mărimea turneului va putea fi modificată, iar în cazul în care avem un turneu cu indivizi mulți, șansele ca un individ slab să fie selectat devin mai mici. Se va putea specifica chiar și probabilitatea de selectare a indivizilor din turnee.

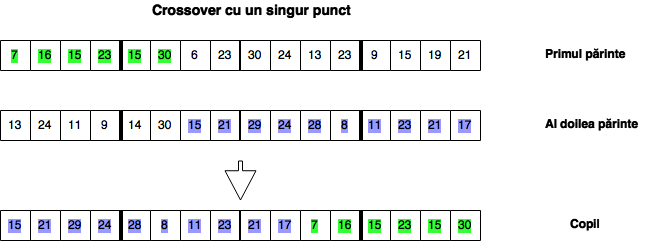


*Figura 4.11 Selecția tip turneu (inspirat din [7])*

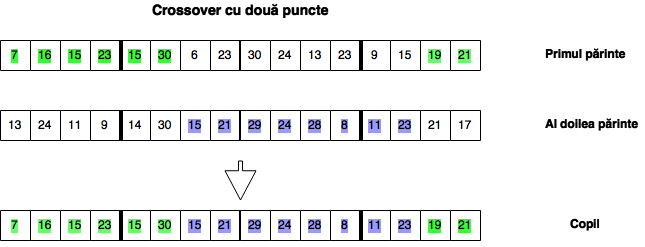
În *figura 4.11* observăm selecția tip turneu. În primul pas *(1)* se aleg doi indivizi *i* și *j*, după care în pasul *(2)* se compară funcțiile lor fitness. Presupunem ca *j* este mai bun decât *i*, adtfel urmează pasul *(3)* în care individul mai *slab* este poziționat în coada noii populații.

Metoda de selecție alesă pentru procesul de optimizare a traficului urban este cea de tip turneu (*Tournament Selection*) prin care se specifică mărimea turneului și probabilitatea de selecție a unui individ. Cu cât mărimea turneului este mai mare cu atât șansele ca un individ mai slab să fie selectat scad. Această metodă oferă mai multe beneficii, printre care eficiența codului scris, funcționeză pe arhitecturi paralele și oferă posibilitatea specificării parametrilor de mărime și probabilității de selecție.

### Crossover

După alegerea părinților va trebui să producem o populație nouă de copii prin combinarea genelor părinților. Acest proces se numește *crossover*. Există mai multe moduri prin care se poate executa acest proces: *Crossover cu un singur punct*, *Crossover cu două puncte* și *Crossover aleator*.

*Figura 4.12 Crossover cu un singur punct*

În *figura 4.12* este prezentată metoda de *crossover cu un singur punct*. Prin această metodă se alege o poziție aleatoare din primul și al doilea părinte după care se execută crossover-ul și vor rezulta doi copii, fiecare din aceștia având câte o parte din genele părințiilor lor.

*Figura 4.13 Crossover cu două puncte*

În *figura 4.13* este prezentată metoda de *crossover cu două puncte*. Prin această metodă se selecteză două poziții aleatoare, după care se va face crossover-ul și va rezulta un copil având câte o parte de gene din părinții săi.

Prin metoda de crossover aleator se selectează mai multe poziții aleatoare din cei doi părinți, după care se va face crossover-ul și va rezulta un copil având bucăți de gene din părinții săi.

Metoda de *crossover* aleasă în cazul optimizării traficului urban este ***Crossover cu un singur punct***. Prin această metodă se selectează aleator un punct din fiecare cromozom părinte după care vor rezulta doi urmași care vor conține câte o parte din genele părinților. Numărul de operații crossover var fi dat ca și procentaj din numărul total al populației de cromozomi.

### Procesul de mutație a genelor

Procesul de mutație adaugă diversitate algoritmului. Prin acest proces ne asigurăm faptul că indivizii rezultați vor avea diferențe între ele, asigurând faptul că algoritmul continuă să găsească și alte combinații de gene. În cazul în care scoatem acest proces de mutație avem șanse mari să ajungem într-un caz în care toți indivizii rezultați seamănă între ei și nu mai pot produce urmași diferiți față de ei.

C:\Users\edWin\Desktop\LicentaGIT\trunk\mutationn.pngFiecare geană din cromozomi are, de obicei o probabilitate de *2-5%* să fie modificată aleator. Această probabilitate trebuie să fie una destul de mică, astfel încât indivizii noi să nu fie slabi.

*Figure 4.14 Mutația genelor*

În *figura 4.14* observăm un exemplu simplu de mutație produs în cadrul unui cromozom cu patru gene. Pe ultima geană, cea cu timpul de 10 secunde, s-a produs o mutație și a rezultat o nouă valore de 22 de secunde. Această mutație a avut loc prin generarea unui nou număr aleator.

### Oprirea procesului de optimizare

După procesul de evaluare putem alege să ne oprim, în cazul în care am găsit o soluție suboptimală, sau putem continua tot acest proces până o soluție apropiată de cea optimă va fi identificată. În funcție de metoda de oprire aleasă, durata acestui proces va varia.

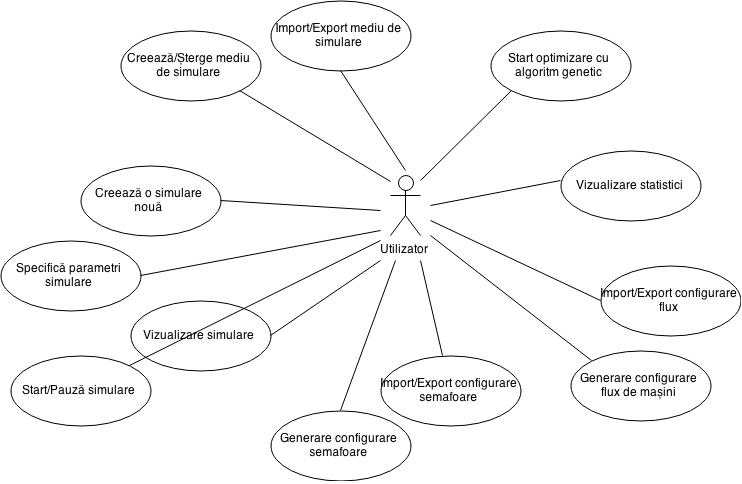
În cazul optimizării traficului urban, acest proces continuă până când s-a atins numărul de evoluții introduse de către utilizator în faza de configurare a optimizării. O evoluție reprezintă un pas din întreg procesul executat de către algoritmul genetic. Pentru a produce rezultate bune, de obicei acest număr se alege undeva între *50* și *500*, ca și în cazul dimensiunii unei populații. Cu cât acest număr este unul mai mare, cu atât întreg procesul va dura mai mult.

# Proiectare de Detaliu si Implementare

În urma definirii obiectivelor și a cerințelor proiectului următoarea etapă din procesul de dezvoltare software îl reprezintă procesul de proiectare. Proiectarea software reprezintă activitatea de stabilire, pe baza analizei unei probleme, a modului general de rezolvare, a împărțirii problemei in subprobleme, precizând în acest fel structura bloc a programului și interfețele între modulele de program ce vor corespunde subproblemelor, de stabilire a funcțiilor modulelor, de testare la nivel de ansamblu a validității proiectului.

## Funcționalitățile sistemului

Atunci când modelăm un sistem trebuie să avem în vedere capturarea comportamentului său dinamic. Diagrama *Use Case* reprezintă cea mai simplă reprezentare de interacțiune a utilizatorului cu aplicația. Prin această diagramă putem identifica diferiții tipi de utilizatori care există în aplicație și acțiunile pe care aceștia le pot executa.

În cazul aplicației de simulare avem doar un singur tip de utilizator, acesta având acces la toate resursele și acțiunile aplicației.

*Figura 5.1 Diagrama Use case a sistemului*

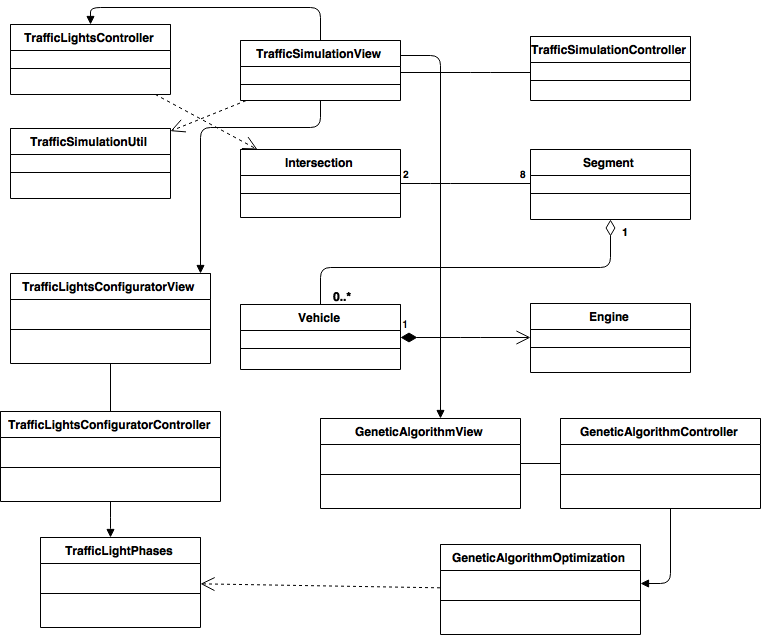
*Figura 5.1* reprezintă diagrama Use Case a aplicației de simulare. Putem identifica acțiunile cele mai importante pe care un utilizator le poate executa. Printre acestea se numără:

* *Creearea/Ștergerea unui mediu de simulare* prin adăugarea de componente, precum *intersecții*, *segmente* și *puncte de intrare/ieșire*
* *Import/Export la întreg mediul de simulare*
* *Creearea unei simulări noi*
* *Specificarea parametrilor unei simulări* (timpul de simulare, pasul simulării)
* *Start/Pauză la procesul de simulare*
* *Vizualizarea în timp real a simulării create*
* *Generarea unei configurații pentru semafoarele din intersecții*
* *Import/Export la configurația de semafoare*
* *Generarea unei configurații de fluxuri de mașini*
* *Import/Export la configurația de fluxuri*
* *Vizualizarea statisticilor simulării* sub formă grafică și numerică
* *Start proces de optimizare* utilizând algoritmul genetic

## Elementele simulatorului

În cadrul proiectării software un pas important îl reprezintă definirea elementelor principale care vor sta la baza aplicației de simulare.

Elementele trebuie să fie o reprezentare cât mai apropiată de obiectele din viața reală. Acestea trebuie să înglobeze *date*, *structuri de date* și *atribute* care să ajute la o funcționare cât mai corectă a aplicației de simulare.

În urma analizelor au rezultat următoarele elemente abstracte: *intersecție*, *punct de intrare/ieșire*, *segment* și *autovehicul*.

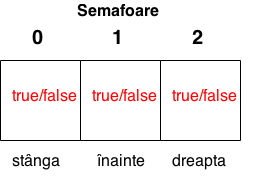
*Figura 5.2 Diagrama de clase a simulatorului*

În *figura 5.2* observăm diagrama de clase a aplicației. Prin această diagramă putem vedea relațiile claselor participante. Astfel, avem o relație de *asociere* de 2 la 8 între *Intersection* (Intersecție*)* și *Segment*, o intersecție având maxim 8 segmente, pe când un segement conecteză neaparat două intersecții. Relația dintre clasele *Segment* și *Vehicle* (Autovehicul*)* este una de *agregare* de 1 la 0..\*, un segment având o listă de 0 sau mai multe autovehicule. Un autovehicul are o relație de *compoziție* cu un *Engine* (Motor). Restul claselor participante reprezintă componentele de generare a configurațiilor pentru semafoare, modulul de optimizare cu algoritmul genetic și clasa de gestionare a semafoarelor din intersecții.

### Clasa Intersecție

Clasa intersecție înglobează toate datele și atributele care țin de o intersecție din mediul urban. Pentru procesul de modelare s-a ales o intersecție având cel mult patru direcții de mers.

Acestă clasă are înglobată și entitatea de *semafor*. Astfel, pe fiecare direcție de mers din intersecție avem câte un semafor care gestioneză traficul. Semaforul are trei direcții posibile de mers: *stânga*, *înainte* și *dreapta*.

Pentru modelarea semafoarelor s-a ales reprezentarea acestora prin liste având a câte trei valori booleene (*true* și *false*).

*Figura 5.3 Reprezentarea semafoarelor*

În *figura 5.3* observăm reprezentarea logică a semafoarelor din cadrul unei intersecții. Cele trei poziții din lista cu valori booleene reprezintă cele trei sensuri de mers: *stânga*, *înainte* și *dreapta*. Valoarea booleană *true* și *false* din cadrul fiecărei poziții semnifică cele două culori ale semaforului: *verde* pentru *true* și *roșu* pentru *false*.

Fazele semafoarelor din intersecții au fost definite după cum urmează:

* **Faza 1**: *Sud -> Nord, Sud -> Est*, *Nord -> Sud*, *Nord -> Vest*
* **Faza 2**: *Est -> Vest*, *Est -> Nord*, *Vest -> Est*, *Vest -> Sud*
* **Faza 3**: *Sud -> Est*, *Est -> Sud*, *Nord -> Vest*, *Vest -> Nord*
* **Faza 4**: *Sud -> Vest*, *Vest -> Sud*, *Nord -> Est*, *Est -> Nord*

Principalele atribute ale unei intersecții sunt:

* *id* – valoare întreagă care reprezintă identificatorul unic al unei intersecții
* *trafficLightsNorth* (semafoare Nord) – listă cu valori booleene care reprezintă semafoarele din nordul intersecției pe toate cele trei direcții de mers
* *trafficLightsSouth* (semafoare Sud) – listă cu valori booleene care reprezintă semafoarele din sudul intersecției pe toate cele trei direcții de mers
* *trafficLightsEast* (semafoare Est) – listă cu valori booleene care reprezintă semafoarele din estul intersecției pe toate cele trei direcții de mers
* *trafficLightsVest* (semafoare Vest) – listă cu valori booleene care reprezintă semafoarele din vestul intersecției pe toate cele trei direcții de mers
* *numberOfPhases* (numărul fazelor) – valoare întregă ce reprezintă numărul de faze din intersecție
* *currentPhase* (faza curentă) – valoare întreagă ce reprezintă faza curentă
* *phaseCounter* (contorizare fază) – valoare întreagă pentru procesul de contorizare al fazelor
* *phaseTimes* (timpii fazelor) – listă cu valori întregi pentru timpii pe fiecare fază
* *segmentNorthIn* – segment care intră din Nord
* *segmentNorthOut* – segment care iese în Nord
* *segmentSouthIn* – segment care intră din Sud
* *segmentSouthOut* – segment care iese în Sud
* *segmentEastIn* – segment care intră din Est
* *segmentEastOut* – segment care iese în Est
* *segmentVestIn* – segment care intră din Vest
* *segmentVestOut* – segment care iese în Vest

C:\Users\edWin\Desktop\LicentaGIT\trunk\inters.JPG

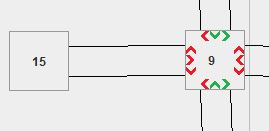
*Figura 5.4 Intersecție*

În *figura 5.4* avem o reprezentare grafică din mediul de simulare a unei intersecții. Se pot observa cele trei semafoare pe toate cele patru direcții de mers. Semaforul este reprezentat de către o săgeată orientă spre direcția de mers. În cazul în care culoarea săgeții este *verde*, participanții la trafic au dreptul să traverseze intersecția pe direcția afișată. În cazul în care culoarea săgeții este *roșu*, autovehiculele se vor opri înaintea intersecției și vor aștepta un anumit timp până când culoarea se va face verde din nou. Totodată, fiecare intersecție are inclus identificatorul său unic pentru a fi mai ușoară găsirea lor în mediul de simulare.

### Clasa Punct de intare/ieșire

Punctele de intrare și de ieșire din simulare reprezintă defapt niște intersecții care au cel mult două segmente conectate pe o parte. Ele vor reprezenta punctele prin care autovehiculele își vor face apariția în simulare și punctele prin care acestea vor părăsi simulatorul.

Atributele punctelor de intrare și de ieșire vor fi identice cu cele ale unei intersecții, având însă unele valori prestabilite sau ignorate. Astfel, vom avea:

* *trafficLightsNorth* (semafoare Nord) – toate valorile booleene din listă vor fi true
* *trafficLightsSouth* (semafoare Sud) – toate valorile booleene din listă vor fi true
* *trafficLightsEast* (semafoare Est) – toate valorile booleene din listă vor fi true
* *trafficLightsVest* (semafoare Vest) – toate valorile booleene din listă vor fi true
* *numberOfPhases* (numărul fazelor) – valoare ignorată
* *currentPhase* (faza curentă) – valoare ignorată
* *phaseCounter* (contorizare fază) – valoare ignorată
* *phaseTimes* (timpii fazelor) – valori ignorate

*Figura 5.5 Punct de intrare/ieșire*

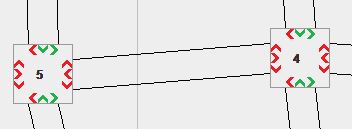
În *figura 5.5* avem o reprezentare grafică a unui punct de intrare/ieșire (identificator *15*) care are legatură directă la o intersecție (identificator *9*). Putem observa asemănarea cu o intersecție, singura diferență fiind inexistența semafoarelor.

### Clasa Segment (Tronson)

Segmentul sau tronsonul reprezintă legătura pe o anumită direcție a două intersecții sau un punct de intrare/ieșire și o intersecție. Fiecare segement modelează deplasarea traficului pe o anumită direcție. În cazul de față avem posibilitatea să definim câte un segment pe fiecare direcție de mers.

Fiecare segment va simula deplasarea autovehiculelor aflate pe segment.

Principalele atribute ale unui segment sunt următoarele:

* *id –* valoare întreagă care reprezintă identificatorul unic al unui segment
* *vehicles* – lista de autovehicule care se află pe segment
* *intersectionFrom* – intersecția de la care începe segmentul
* *intersectionTo* – intersecția în care se termină segmentul

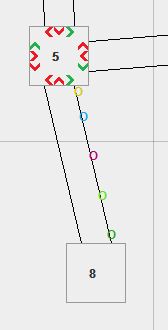
*Figura 5.6 Segmente între intersecții*

În *figura 5.6* putem observa o reprezentare a mai multor segmente în cadrul mediului de simulare. Două dintre aceste segmente leagă cele două intersecții, *4* respectiv *5*.

### Clasa Autovehicul

Autovehiculul reprezintă elementul dinamic (agentul) din simulare și încearcă să înglobeze toate atributele unui autovehicul real din trafic.

Atributele principale ale unui autovehicul sunt:

* *id* – valoare întreagă care reprezintă identificatorul unic al unui autovehicul
* *size* – valoare întreagă ce reprezintă lungimea unei mașini
* *routeList* – listă care conține segmentele din ruta autovehiculului
* *destination –* segmentul care reprezintă destinația din ruta autovehiculului
* *currentSegment* – segmentul curent pe care se găsește autovehiculul
* *currentDistance* – valoare reală ce reprezintă distanța curentă parcursă pe segment
* *distanceToObstacle* – valoare reală ce reprezintă distanța până la următorul obstacol
* *speed* – valoarea reală ce reprezintă vteza curentă a autovehiculului
* *engine* – tipul de motor al autovehiculului

*Figura 5.7 Deplasarea unui flux de mașini*

În *figura 5.7* avem reprezentarea unui pas de simulare a deplasării unui flux de mașini. Autovehiculele din flux vor avea culori aleatorii pentru o identificare mai ușoară, iar forma lor va fi un cerc.

## Procesul de implementare

Procesul de implementare presupune în primul rând alegerea unui limbaj de programare adecvat. Deoarece avem nevoie de un limbaj de programare obiectual și cu o librărie grafică bogată am restrâns căutările la două limbaje: *Java* și *C#*.

Din cauza nivelului mai mare de familiaritate cu limbajul *Java*, acesta a fost ales ca și limbajul de programare utilizat în procesul de implementare. Există însă și alte motive care au stat la baza acestei decizii. Aceste motive includ faptul că un program scris în Java poate să ruleze pe orice platformă (*Write once, run anywhere*), este un limbaj dinamic și extensibil, oferă performanță și este un limbaj *multithreaded*.

Ca și mediu de dezvoltare s-a folosit IDE-ul *IntelliJ 14.0.3* [11] din cadrul companiei *JetBrains*.

Pentru procesul de versionare a codului sursă s-a folosit ca și host online *GitHub*. Accesul a fost asigurat de către clientul *Subversion* integrat în *IntelliJ*. Clientul oferă funcționalitățile principale ale unui astfel de sistem: *commit*, *update* și *compare*. Procesul de *commit* va încerca să salveze modificările locale ale codului sursă pe sistemul centralizat (GitHub), pe când procesul de *update* va actualiza codul sursă local cu cel din sistemul centralizat (GitHub). Procesul de *compare* ne permite să vizualizăm diferențele dintre codul nostru sursă local cu cel de pe sistemul centralizat (GitHub). Prin folosirea acestui sistem de versionare s-a putut urmări îndeaproape întreg procesul de implementare și s-a putut reveni foarte ușor la variante funcționale mai vechi ale codului.

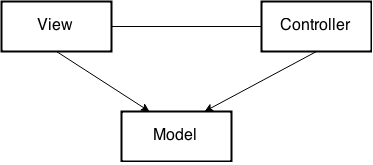
### Model-View-Controller

MVC reprezintă un pattern arhitectural folosit pentru implementarea interfețelor utilizator. Prin acest pattern un sistem software se poate împărți în trei componente interconectate rezultând astfel componente ce reprezintă modelul aplicației (*model*), partea de interfață a aplicației (*view*) și componente ce se ocupă de cererile utilizatorilor (*controller*). Datorită acestei separări nu vom putea amesteca cod într-o singură clasă.

Pe întreg procesul de implementare a sistemului de simulare s-a încercat urmărirea acestui pattern și separarea componentelor în *model*, *view* și *controller*.

Câteva din avantajele oferite de acest pattern:

* separarea responsabilităților
* flexibilitate în ceea ce privește adăugarea sau schimbarea unei componente



*Figura 5.8 Model-View-Controller*

În *figura 5.8* avem reprezentate legăturile dintre cele trei componente ale pattern-uui arhitectural MVC. Astfel, *controller*-ul va putea gestiona starea unui model și va putea actualiza partea de view, *modelul* va conține date care vor fi reprezentate în view, iar partea de *view* se va ocupa de prezentarea datelor din model.

### Framework-ul JGAP

JGAP (*Java Genetic Algorithm Package*) [10] reprezintă un framework Java care include implementarea algoritmului genetic. Conține mecanisme genetice de bază care pot fi aplicate ușor pe orice fel de probleme.

Pentru partea de optimizare a simulatorului s-a considerat o soluție bună integrarea acestui framework. Integrarea framework-ului JGAP în aplicație presupune urmărirea următorilor pași:

* *definirea unui cromozom*
* *implementarea unei funcții de fitness*
* *definirea unui obiect de configurare*
* *crearea unei populații de soluții posibile*
* *evoluția populației*

Pentru a avea un cromozom trebuie să ne definim genele care vor compune acest cromozom. În cazul simulatorului genele vor fi reprezentate prin valori întregi care au semnificația celor pentru timpi ai fazelor. Astfel s-au definit câte patru *IntegerGene*-uri în intervalul ***[5, 30]*** care reprezintă timpul minim respectiv maxim pentru o fază. Aceste gene au fost incluse într-un *CompositeGene*, care reprezintă o intersecție. După definirea tuturor genelor, acestea vor fi incluse într-un *Chromosome* care va reprezenta cromozomul nostru. Valorile generate de către gene în cadrul cromozomilor vor fi aleatoare datorită generatorului *StockRandomGenerator* specificat.

Pentru definirea unei funcții de fitness trebuie să extindem clasa *FitnessFucntion* și să suprascriem metoda de evaluare a valorii fitness pentru cromozomul curent *evaluate*. În cadrul acestei metode vom încerca să executăm o simulare folosind timpii cromozomului curent. Timpul mediu de simulare rezultat în urma simulării va reprezenta valorea care va trebui să fie comparată. Întrucât cromozomii cu valorea fitness mai mare sunt considerați mai buni, iar timpul mediu de așteptare trebuie să fie cât mai mic s-a ales ca valoarea returnată de către funcția de fitness să fie diferența dintre un număr relativ mare și timpul mediu de așteptare returnat de către simulator. Astfel, se va putea face diferența dintre doi cromozomi luând în considerare acești timpi. Clasa rezultată poartă denumirea de *MinimizingWaitingTimeFitnessFunction*.

Obiectul de configurare *Configuration* va conține parametrii specifici algoritmului gentic. Pentru procesul de selecție s-a ales *Selecția prin turneu*, având astfel un selector de tip *TournamentSelector*. Valoarea parametrilor algoritmului genetic va fi specificată de către utilizator.

Crearea unei populații inițiale sau *genotip* are loc prin asignarea de valori aleatorii tuturor genelor din cadrul cromozomilor pe baza configurației specificate de către utilizator.

Procesul de evoluție va aplica procedeul *Crossover într-un singur punct* având ca și rezultat populații cu indivizi noi.

În urma procesului de evoluție se va selecta cel mai *fit* cromozom din populație și se va salva configurația rezultată într-un fișier. Configurația se va obține prin accesarea genelor de tip *CompositeGene* din cadrul cromozomului. Fiecare geană va conține configurația timpilor fazelor pentru câte o intersecție din mediul de simulare.

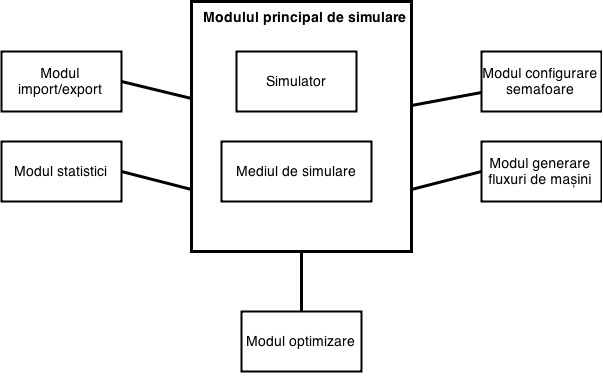
## Modulele aplicației

Sistemul de simulare este alcătuit din mai multe module care furnizeză diverse funcționalități. Fiecare din aceste module poate fi considerată o aplicație de sine stătătoare putând fi dezvoltată separat față de restul modulelor.

Modulele aplicației sunt după cum urmeză:

* *modulul principal de simulare*
* *modulul de configurare a semafoarelor*
* *modulul de generare a fluxurilor de mașini*
* *modulul de configurare a algoritmului genetic*

Pe lângă toate aceste module mai există și un modul folosit pentru funcțiile de import și export a diferitelor fișiere de configurare, precum și un modul folosit pentru înregistrarea statisticilor procesului de simulare.



*Figura 5.9 Modulele sistemului de simulare*

*Figura 5.9* reprezintă schema generală a tuturor modulelor sistemului de simulare. Această reprezentare este denumită și diagramă bloc a sistemului, având reprezentarea fiecărui modul sub formă de *black box*.

### Modulul principal de simulare

Modulul de simulare reprezintă componenta principală a simulatorului, înglobând toate funcționalitățile oferite de către sistem, precum și restul modulelor. Două dintre funcțiile principale oferite de către acest modul sunt procesul de simulare a unor fluxuri de autovehicule în mediul de simulare creat, precum și posibilitatea construirii unui astfel de mediu.

În cadrul procesului de proiectare a mediului de simulare s-a ales împărțirea cadrului în trei zone principale: *zona de vizualizare a simulării*, *zona conținând logul simulării* și *zona de vizualizare a statisticilor simulării*.

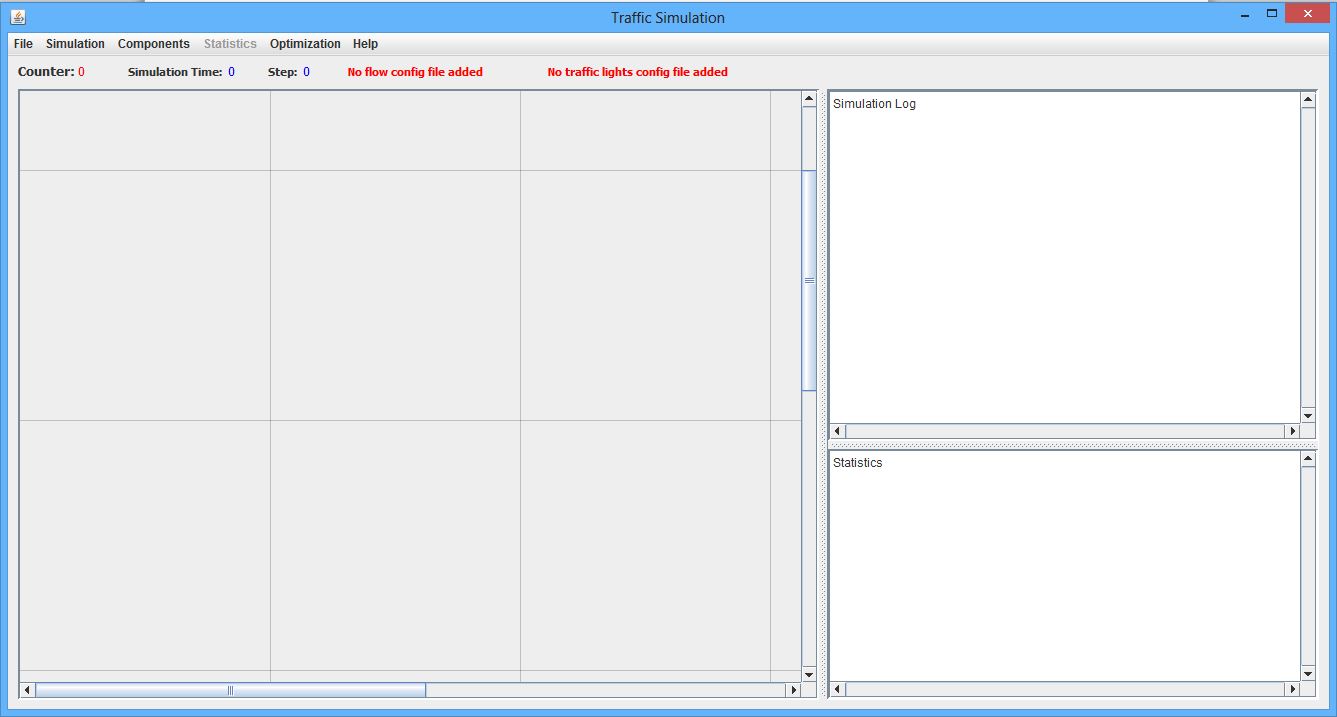
Zona de vizualizare a simulării reprezintă cel mai important cadru al sistemului de simulare. Întreg procesul de simulare al unui scenariu din traficul urban poate fi urmărit prin această zonă. Totodată, utilizatorul va putea să își definească un anumit scenariu de simulare datorită diferitelor componente care pot fi adăugate în acest cadru.

Zona conținând logul simulării reprezintă cadrul prin care utilizatorul va fi informat prin mesaje de tip log asupra evenimentelor din cadrul procesului de simulare. Aceste evenimente pot fi de exemplu legate de procesul de generare a autovehiculeleor, timpii în care aceste generări au avut loc și rutele care au fost asignate pentru fiecare mașină.

Zona de vizualizare a statisticilor simulării reprezintă cadrul prin care utilizatorului îi sunt furnizate statistici cu privire la simularea care tocmai a luat sfârșit. Câteva din statisticile care apar în această zonă sunt timpul mediu de așteptare al tutror mașinilor din simulare, respectiv timpul mediu de existență a mașinilor în cadrul simulării.

Simulatorul mai este compus dintr-o zonă care conține informații cu privire la procesul curent de simulare. Elementele acestei zone sunt:

* *un contorizator de timp* folosit pentru afișarea timpului curent al simulării
* *timpul de simulare* ales de către utilizator
* *pasul de simulare* ales de către utilizator
* existența unui *fișier cu fluxuri de mașini* în cadrul simulării
* exitența unui *fișier cu configurații pentru semafoarele* din simulare



*Figura 5.10 Reprezentarea grafică a modulului de simulare*

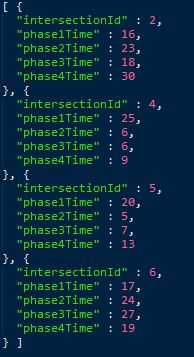
În *figura 5.10* putem observa reprezentarea grafică a modulului de simulare. Meniul principal al acestui modul este compus din următoarele elemente:

* ***File –> Export environment to JSON***– folosit pentru a exporta întreg mediul de simulare într-un fișier de tip JSON
* ***File –> Import environment from JSON*** – folosit pentru a importa un mediu de simulare deja existent dintr-un fișier de tip JSON
* ***File –> Exit*** – folosit pentru închiderea aplicației
* ***Simulation –> New*** – folosit pentru crearea unei simulări noi. Se va șterge întreg mediul de simulare și se vor reseta toate valorile
* ***Simulation –> Generate flow config file*** – folosit pentru deschiderea modulului de generare a unui fișier de configurare a fluxurilor de mașini
* ***Simulation –> Add flow config file*** – folosit pentru încărcarea unui fișier deja existent de tip JSON conținând fluxuri de mașini
* ***Simulation –> Generate traffic lights config file*** – folosit pentru deschiderea modulului de generare a unui fișier de configurare a semafoarelor intersecțiilor
* ***Simulation –> Add traffic lights config file*** – folosit pentru încărcarea unui fișier deja existent de tip JSON conținând configurări pentru semafoare
* ***Simulation –> Start*** – folosit pentru pornirea procesului de simulare. Se vor preciza timpul de simulare, precum și pasul de simulare
* ***Simulation –> Pause*** – folosit pentru a opri simularea pentru câteva clipe
* ***Simulation –> Resume*** – folosit pentru continuarea procesului de simulare
* ***Components* *–> Clear all*** – folosit pentru ștergerea tuturor componentelor care apar în mediul de simulare
* ***Components* *–> Segment*** – folosit pentru a crea un segment între două intersecții deja existente
* ***Components* *–> Intersection*** – folosit pentru a crea o intersecție sau un punct de pornire/oprire, în cazul în care numărul de segmente conectate va fi mai mic sau egal decât 4. Cât timp componenta *Intersection* va rămâne selectată utilizatorul va putea continua să adauge intersecții în mediul de simulare
* ***Statistics* –** conține diferite statistici ale simulării care pot fi vizualizate sub formă grafică
* ***Optimization –> Genetic algorithm*** – folosit pentru deschiderea modulului de optimizare cu algoritm genetic
* ***Help* *–> About*** – informații cu privire la sistemul de simulare

### Modulul de configurare a semafoarelor

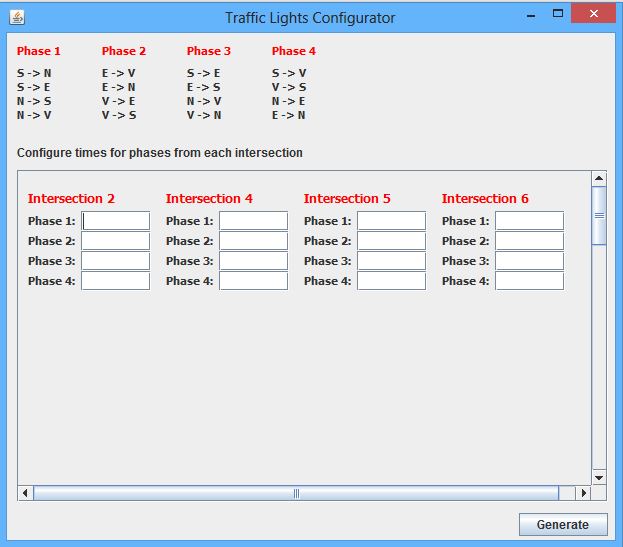
Modulul de configurare a semafoarelor conține parametri semafoarelor care pot fi confihurați de către utilizator. În procesul de modelare a acestui modul s-au ales timpii fazelor din fiecare intersecție ca fiind configurabili, fiind considerații elementele care pot avea un impact major asupra procesului de optimizare a traficului urban.

Acest modul informează utilizatorul asupra configurației de faze și ordinea lor din fiecare intersecție. Astfel, utilizatorul va putea alege timpi potriviți pentru fiecare fază.

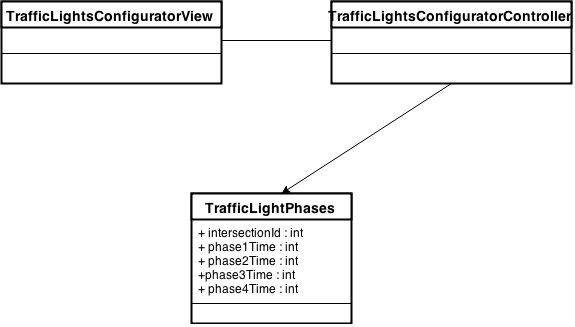
În momentul în care toți timpii au fost definiți, se generează un fișier de tip JSON care va conține informații cu privire la identificatorii intersecțiilor și timpii pentru fiecare fază.

*Figura 5.11 Exemplu de fișier cu configurații ale fazelor*

În *figura 5.11* avem un exemplu de fișier JSON care conține configurațiile tutoror timpilor fazelor din patru intersecții având identificatorii 2, 4, 5 și 6.

Fișierul care rezultă în urma procesului de generare se va putea importa în mediul de simulare pentru a putea testa eficiența timpilor aleși.

*Figura 5.12 Modulul de configurare a semafoarelor*

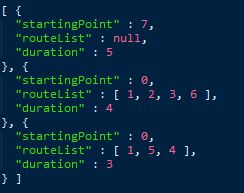
În *figura 5.12* observăm reprezentarea grafică a modulului de generare a unei configurații pentru semafoarele din mediul de simulare. Prima parte a cadrului conține informații cu privire la fazele semafoarelor din intersecții, precum și ordinea lor de execuție. În cea de-a doua parte a cadrului avem listate toate intersecțiile existente din simulare, având posibilitatea configurării timpilor pentru fiecare dintre cele patru faze ale semafoarelor. După specificarea tuturor timpilor, utilizatorul va trebui să apese butonul *Generate* și va trebui să specifice calea în care fișierul JSON va fi salvat.

*Figura 5.13 Diagrama de clase a modulului de generare a unei configurații pentru semafoare*

În *figura 5.13* avem reprezentată diagrama de clase a modulului de generare a unei configurații pentru semafoarele din intersecții. După cum se observă, pattern-ul athitectural folosit este *Model-View-Controller*. Pe partea de model avem clasa *TrafficLightPhases* care înglobează atributele necesare pentru configurarea fazelor unei intersecții. Această clasă va fi folosită în cadrul procesului de import/export a fișierului cu configurații. Clasa *TrafficLightsConfiguratorView* se ocupă cu partea de prezentare a modulului de configurare, iar clasa *TrafficLightsConfiguratorController* reprezintă partea de controller a modulului care procesează întreaga configurație dată de către utilizator și o salvează într-un fișier de tip JSON.

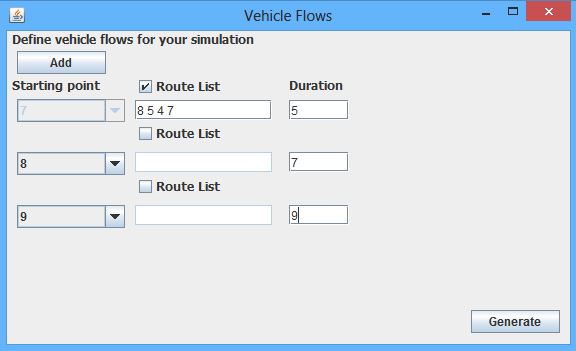
### Modulul de generare a fluxurilor de mașini

Modulul de generare a fluxurilor de mașini oferă posibilitatea creării a diferite fluxuri care vor fi pe urmă simulate în cadrul mediului de simulare. Există posibilitatea definirii a două tipuri de fluxuri: *Flux de mașini cu un punct de pornire* și *Flux de mașini cu o rută fixă*. Pentru definirea unuii flux de mașini cu un punct de pornire, utilizatorul va trebui să specifice doar identificatorul punctului de intrare în simulare și timpul de generare al fluxului. În cazul fluxului de mașini cu rută fixă, utilizatorul va trebui să specifice identificatorii intersecțiilor prin care fluxul va trebui să treacă. Primul și ultimul identificator vor trebui să aparțină unui punct de intrare și ieșire din simulare.

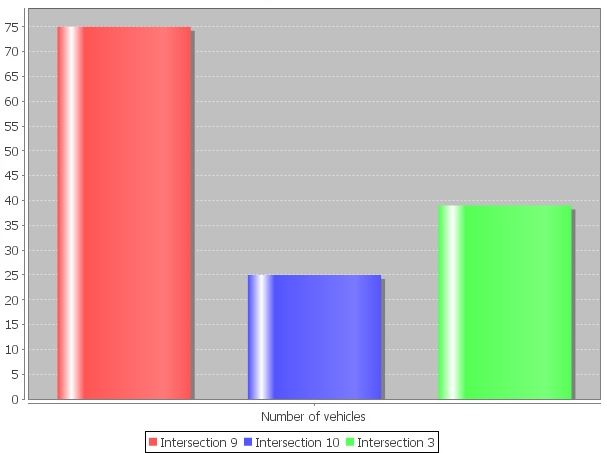
În urma specificării fluxurilor, modulul va genera un fișier de tip JSON conținând definițiile tuturor fluxurilor configurate.

*Figura 5.14 Exemplu de fișier cu fluxuri de mașini*

În *figura 5.14* avem un exemplu de fișier JSON care conține configurațiile tutoror fluxurilor de mașini definite de către utilizator. Fișierul este constituit de punctele de pornire a fluxurilor (*startingPoint*), lista cu ruta fluxulu în cazul unu flux static (*routeList*) și nu în ultimul rând durata unui flux (*duration*) exprimată în secunde.



*Figura 5.15 Modulul de generare al fluxurilor de mașini*

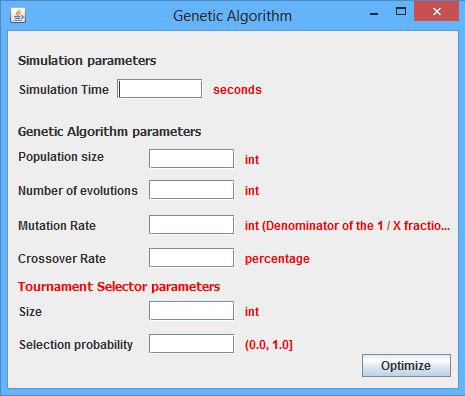
În *figura 5.15* observăm reprezentarea grafică a modulului de generare al fluxurilor de mașini. Utilizatorul va putea defini cele două tipuri de fluxuri prin alegerea unui identificator de punct de pornire din *drop box* sau definirea întregi rute într-un *text field*. După specificarea tuturor fluxurilor, utilizatorul va trebui să apese butonul *Generate* și va trebui să specifice calea în care fișierul JSON va fi salvat.

*Figura 5.16 Histograma intersecții*

*Figua 5.16* reprezintă o histogramă a numărului de mașini care au trecut prin intersecțiile cu identificatorii *3*, *9* și *10*.

### Modulul de configurare a algoritmului genetic

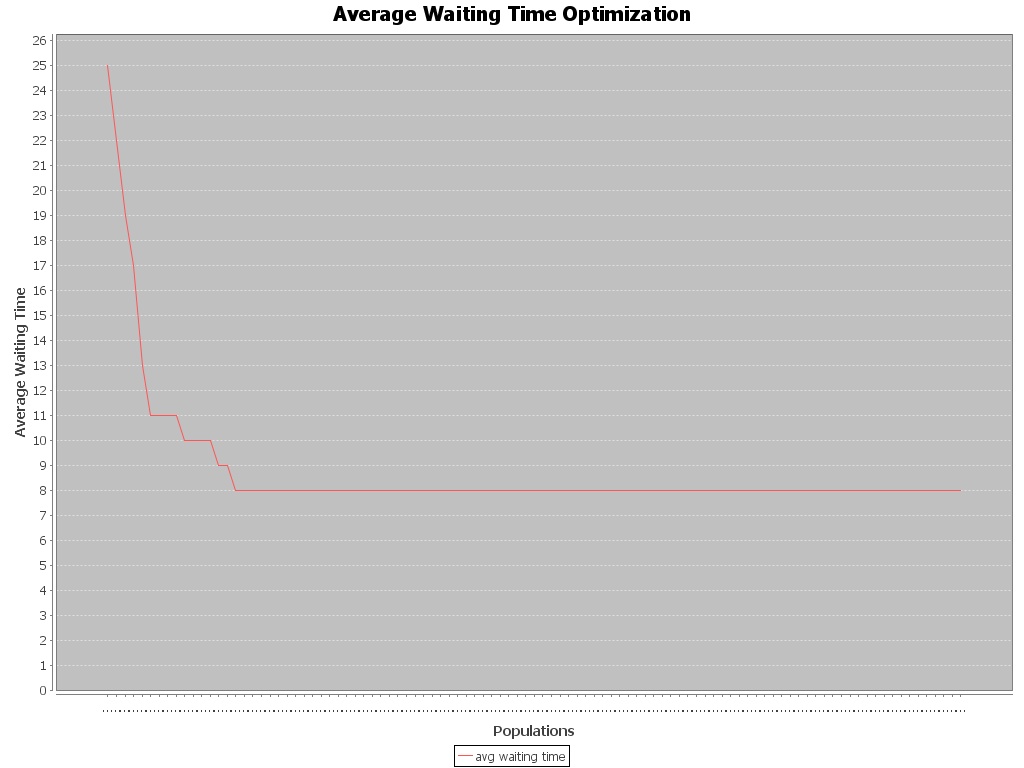
Modulul de configurare a algoritmului genetic reprezintă componenta prin care utlizatorul va putea să își optimizeze scenariul de simularea prin utillizare algoritmului genetic. Algoritmul genetic va încerca să găsească o configurație suboptimală de timpi ale fazelor intersecțiilor. Pentru aceasta, există câțiva parametrii care trebuie specificați înaintea procesului de optimizare:

* ***timpul de simulare*** – valoare întreagă exprimată în secunde
* ***mărimea populației de indivizi*** *–* valoare întreagă
* **numărul de evoluții** – valoare întreagă
* ***rata mutației*** - valoare întreagă care reprezintă numitorul fracției ***1/X***
* ***rata de crossover*** – valoare reală
* ***mărimea turneului*** – valoare întreagă
* ***probabilitatea de selecție din cadrul turneului*** – valoare reală din intervalul ***(0.0, 1.0]***

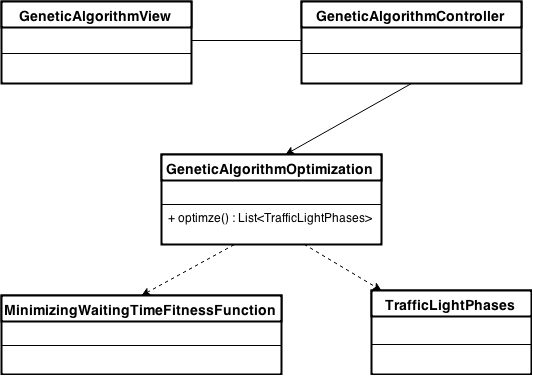
*Figura 5.17 Modulul de configurare a algoritmului genetic*

În *figura 5.17* observăm reprezentarea grafică a modulului de configurare a algoritumului genetic. Fiecare parametru configurabil conține informații cu privire la tipul valorii acceptate de către sistem.

În urma specificării tuturor parametrilor, utilizatorul va apăsa butonul *Optimize* care va executa procesul de optimizare utilizân algoritmul genetic. În timpul în care are loc optimizarea, se va afișa o bară de progres pentru a simula trecerea timpului. În momentul în care s-a găsit o configurație suboptimală pentru timpii fazelor, modulul va genera o reprezentare grafică a întreg procesului de optimizare prin care ar trebui să observăm scăderea timpului mediu de așteptare a mașinilor din simulator. Totodată, modulul ne va permite să salvăm fișierul care conține configurația suboptimală a timpilor fazelor într-o locație aleasă de noi. Acest fișier va putea fi folosit în cadrul procesului de simulare pentru a observa deplasarea mai eficientă a fluxurilor de mașini datorită configurației suboptimale.



*Figura 5.18 Reprezentarea grafică a procesului de optimizare*

În *figura 5.18* observăm reprezentarea grafică a procesului de optimizare. După cum observăm timpul mediu de așteptare a scăzut drastic de la 25 de secunde la 8 secunde. Datorită flexibilității procesului de optimizare aceste valori se pot modifica prin specificarea unei configurații diferite.

*Figura 5.19 Diagrama de clase a modulului de configurare a procesului de optimizare*

În *figura 5.19* avem diagrama de clasă pentru modulul de optimizare folosind algoritmul genetic. După cum se poate observa pattern-ul arhitectural folosit este *Model-View-Controller*. Pe partea de model avem două clase: *GeneticAlgorithmOptimization* și *MinimizingWaitingTimeFitnessFunction*. Prima clasă încapsuleză logica de optimizare folosită de către algoritmul genetic, iar cea de-a doua clasă conține logica de definire a funcției de fitness. Pe partea de model s-a mai utilizat și clasa *TrafficLightPhases,* din modulul de generare a unei configurații pentru semafoare, pentru a putea furniza utilizatorului un fișier de tipul JSON cu configurația optimă a timpilor. Clasa *GeneticAlgorithmView* reprezintă partea de prezentare a modulului prin care utilizator introduce parametrii de configurare a procesului de optimizare. Pe partea de controller avem clasa *GeneticAlgorithmController* care se ocupă de inițierea procesului de optimizare precum și de afișarea graficului rezultat de pe urma acestui proces. Totodată, această clasă se ocupă și de salvarea configurației suboptimale într-un fișier de tipul JSON.

Codul sursă al cestui modul poate fi consultat în ***Anexa 1***.

## Fazele procesului de simulare și optimizare

Sistemul de simulare implementat este constituit din câteva faze importante. Fiecare din acestea are definite pre-condiții care trebuie respectate înaintea inițierii lor.

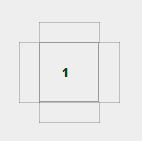
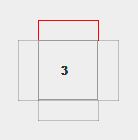
În continuare se vor descrie câteva dintre fazele procesului de simulare și optimizare.

### Faza de construcție a mediului de simulare

Faza de construcție a mediului de simulare reprezintă primul pas în procesul mare ce reprezintă simularea traficului urban. Pentru aceasta sistemul de simulare pune la dispoziția utilizatorului câteva instrumente necesare pentru realizarea procesului. Instrumentele disponibile utilizatorului sunt: *mediul se simulare*, *intersecții* și *segmente*.

Primul pas al acestui proces îl reprezintă inițializarea mediului de simulare. În cazul în care utilizatorul dorește să folosească un mediu de simulare mai vechi, el poate alege să importe mediul respectiv prin funcționalitatea de *Import* oferită de simualtor. Dacă utilizatorul dorește să creeze un mediu nou de simulare, el va trebui să se asigure că lucrează pe un mediu curat. Pentru aceasta se va apela din meniul *Components* opțiunea *Clear all* care va șterge toate componentele mediului de simulare.

Următorul pas al acestui proces îl reprezintă construcția propriu-zisă a mediului de simulare. Pentru aceasta se vor alege componentele necesare mediului din meniul *Components* care conține cele două modele fundamentale ale aplicației, și mai exact *Intersection* și *Segment*. În cazul în care se alege includerea unei intersecții în aplicație, această opțiune va rămâne activă până când utilizatorul o dezactivează. Segmentele vor putea fi incluse doar după ce vor exista cel puțin două intersecții care se pot interconecta. În momentul în care se adaugă o intersecție, aceasta va deveni din start un punct de intrare/ieșire, întrucât nu are segmente conectate. În momentul în care punctul de intrare/ieșire respectiv va avea cel puțin 6 segmente, ea va deveni o intersecție.



*(a)*

*(b)*

*Figura 5.20 Intersecții după selectarea elementului segment*

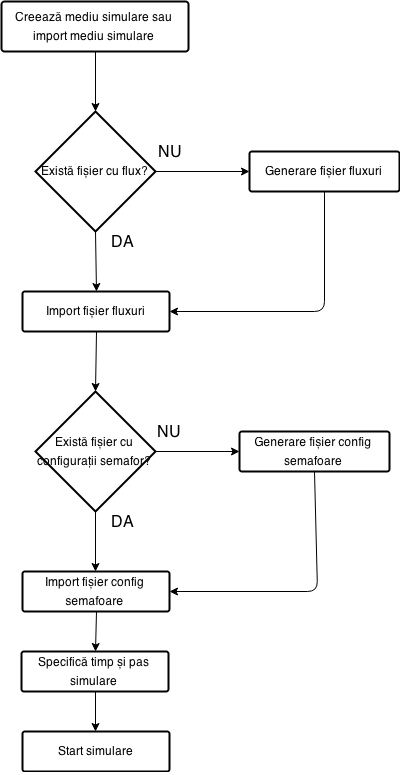
În cazul în care mediul de simulare conține cel puțin două intersecții și se dorește adăugarea unui segment, în jurul intersecțiilor vor apărea patru zone selectabile în partea nordică, sudică, estică și vestică (vezi *figura 5.20-a*). Aceste zone vor reprezenta locurile în care se vor putea conecta segmentele de către utilizator. *Lungimea* acestei zone este de *60 pixeli*, respectiv *lățimea* de *20 pixeli*. În cazul în care se va selecta una dintre zone, aceasta va fi încadrată cu culoarea roșie pentru a putea urmări cu ușurință activitatea utlizatorului (vezi *figura 5.20-b*).

Înainte de finalizarea procesului de construcție utilizatorul trebuie să se asigure că mediul construit de către el este unul corect și că există cel puțin un punct de intrare și un punct de ieșire din simulare. Fără existența acestor puncte procesul de simulare nu va putea avea loc.

În cazul în care utilizatorul dorește să își salveze întreg mediul de simulare, el va trebui să aleagă din meniul *File* opțiunea *Export environment to JSON*. În continuare se va specifica calea și numele fișierului care va rezulta în urma procesului de export. Acest fișier va putea fi importat mai târziu pentru a ințializa mediul de simulare.

### Faza de simulare

Faza de simulare reprezintă una dintre cele mai importante procese ale sistemului. Prin invocarea acestui proces utilizatorul poate să simuleze diferite scenarii apărute în traficul urban.

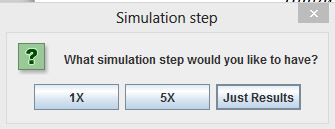


*Figura 5.21 Diagrama flow chart a procesului de simulare*

În *figura 5.21* avem diagrama *flow chart* a procesului de simulare. În continuare se vor descrie pașii acestui proces.

Ca și prim pas al acestui proces îl reprezintă existența unui mediu de simulare fără de care simularea nu poate avea loc. Acest mediu poate să fie importat dintr-un fișier JSON existent sau poate să fie construit pas cu pas.

În continuare se verifică existența unei configurații de fluxuri de mașini în cadrul simulatorului. Aceasta se poate importa dintr-un fișier JSON existent sau poate fi generat din modulul de generare a fluxurilor de mașini.

Pasul următor verifică dacă avem o configurație a timpilor fazelor pentru semafoarele din intersecții. Configurația respectivă se poate importa dintr-un fișier JSON existent sau se poate genera cu ajutorul modulului de generare a unei configurații pentru semafoare.

*Figura 5.22 Pasul de simulare*

Următorul pas presupune existența celor două fișiere în cadrul simulatorului și reprezinte specificarea parametrilor simulării. Parametrii care vor trebuie specificați de către utilizator sunt timpul de simulare exprimat în secunde și pasul de simulare (vezi *figura 5.22*). Pentru pasul de simulare se vor putea alege trei variante: *1X*, *5X* și *Just Results*. Prima variantă reprezintă simularea pas cu pas, cea apropiată de realitate; a doua vasriantă reprezintă o simulare având un pas de 5 secunde, iar ultima variantă reprezintă o simulare prin care se vor afișa doar rezultatele finale, pasul de simulare fiind egal cu timpul de simulare.

În cazul în care toți pașii precedenți s-au executat cu succes, simularea va porni și se vor putea observa fluxurile de mașini făcându-și apariția în mediul de simulare. Utilizatorul va putea opri simularea pentru câteva clipe accesând meniul *Simulation* și acțonând butonul *Pause*. Simularea va putea continua doar la acționarea butonului *Resume* din meniul *Simulation*.

În momentul în care simularea se va sfârși se vor putea vizualiza diferite statistici grafice și numerice. Statisticile de tip grafic vor putea fi accesate din meniul *Statistics*, pe când cele numerice vor putea fi vizualizate în zona din simulator destinată statisticilor.

### Faza de optimizare

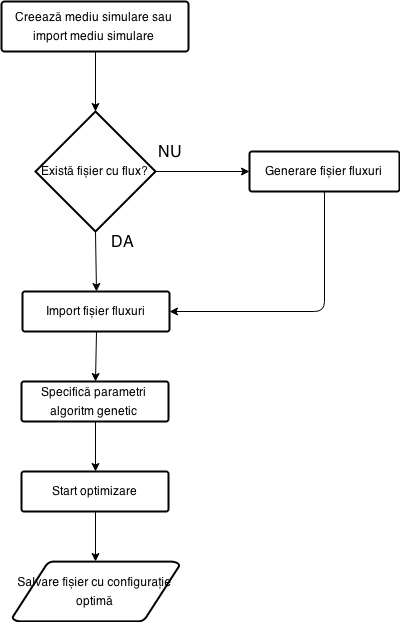
Faza de optimizare reprezintă un alt proces important din cadrul simulatorului. Urmând acest proces utilizatorul poate să găsească o configurație de timpi ai fazelor suboptimală prin care să minimizeze timpul mediu de așteptare a mașinilor din intersecții.

Primul pas al acestui proces, fiind chiar o pre-condiție, îl reprezintă existența unui mediu de simulare. Mediul poate să fie importat dintr-un fișier JSON existent sau poate să fie construit pas cu pas.

În continuare se verifică existența unei configurații de fluxuri de mașini în cadrul simulatorului. Aceasta se poate importa dintr-un fișier JSON existent sau poate fi generat din modulul de generare a fluxurilor de mașini. Procesul de optimizare are nevoie de această configurație pentru a ști care vor fi fluxurile de mașini care necesită optimizarea timpilor de așteptare din intersecții.

Pasul următor presupune invocarea modulului de optimizare cu algoritm genetic din meniul *Optimization* opțiunea *Genetic algorithm*. Acest proces necesită specificarea unor parametri specifici algoritmului. Aceștia includ timpul de simulare, numărul indivizilor dintr-o populație, numărul evoluțiilor, rata de mutație, rata de crossover, mărimea turneului și probabilitatea de selecție din turneu. După definirea tuturor parametrilor se va porni procesul de optimizare.

La finalul procesului de optimizare vom putea vizualiza un grafic ce va reprezenta evoluția timpului mediu de așteptare pentru autovehicule. Totodată, vom putea salva într-un fișier configurația suboptimala de timpi ai fazelor rezultat în urma procesului de optimizare.



*Figura 5.23 Diagrama flow chart a procesului de optimizare*

*Figura 5.23* reprezintă diagrama *flow chart* a procesului de optimizare din cadrul simulării. Pașii acestui proces au fost descriși mai sus.

Procesul de optimizare este executat cu ajutorul framework-ului *JGAP* care conține implementarea algoritmului genetic.

# Testare şi Validare

În cadrul procesului de dezvoltare a unui produs software etapa de testare și validare reprezintă procesul prin care se verifică dacă sistemul are implementate toate specificațiile și funcționează corect.

Procesul de testare și validare trebuie să fie efectuat în paralel cu procesul de implementare a unui produs. Astfel, în urma definirii și implementării unei funcționalități noi dezvoltatorul sistemului poate să se asigure de corectitudinea acesteia prin diferite tehnici de testare. Există două categorii importante de tehnici de testare: *testare manuală* și *testare automatizată*. Prin testare manuală, utilizatorul care efectueză testarea trebuie să joace rolul utilizatorului final al sistemului și să execute toate procesele prinicipale ale sistemului. Prin testare automatizată se folosesc tool-uri specializate care verifică în mod automat funcționarea unor componente ale sistemului.

În cadrul procesului de dezvoltare a sistemului de simulare a traficului urban tehnica de testare și validare a produsului aleasă a fost cea manuală. Astfel, după implementarea unui proces sau a unei funcționalități s-a efectuat o testare manuală din perspectiva unui utilizator final al aplicației.

Funcționalitățile oferite de către sistemul de simulare sunt următoarele:

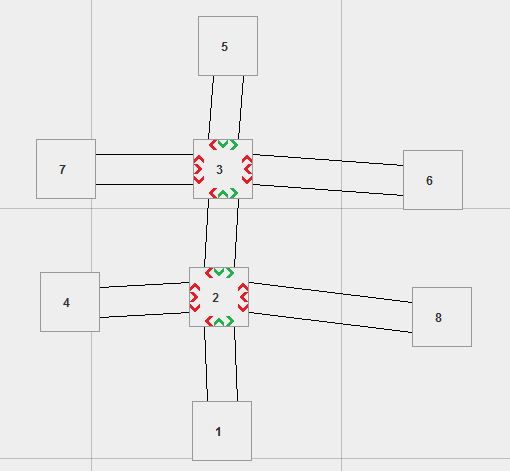
* *crearea unui mediu pentru simulare folosind instrumentele oferite de sistem*
* *salvarea mediului de simulare sub formă de fișier JSON*
* *încărcarea unui mediu de simulare dintr-un fișier JSON deja existent*
* *generarea și salvarea unei configurații pentru fluxuri de autovehicule*
* *generarea și salvarea unei configurații pentru semafoare*
* *definirea și vizualizarea unui scenariu de simulare din traficul urban*
* *optimizarea scenariului de simulare definit utilizând algoritmul genetic*
* *vizualizarea statisticilor simulării sub formă grafică și numerică*

Primul proces mai complex care a fost testat a fost cel prin care utilizatorul poate să își definească un mediu de simulare cu ajutorul componentelor oferite de sistem. În primul rând toată testarea s-a bazat pe partea grafică a procesului. Astfel, s-a încercat definirea unor medii de simulare greșite, având intersecții care ies din panoul de simulare sau încercând conectarea a două intersecții prin mai mult de două segmente. După validarea procesului grafic de crearea s-a testat corespondența elementelor grafice la nivelul logic de entități. Pentru aceasta s-a folosit *debugger*-ul oferit de către mediul de dezvoltare. Astfel, în urma definirii din punct de vedere grafic a mediului s-a verificat, prin introducerea unor *breakpoint*-uri în cadrul codului sursă corespondența cu modelele logice de intersecție și segment.

Pentru procesul de salvare și încărcare a mediului de simulare s-a creat în primul rând un mediu valid, după care s-a încercat salvarea acestuia sub formă de fișier JSON. Fișierul rezultat a fost validat din punct de vedere sintactic și a fost verificată corespondența sa cu mediul construit. După aceasta s-a încercat încărcarea sa într-un mediu de simulare inițial gol. Pe baza acestui fișier sistemul a construit un mediu de simulare nou.

După implementarea și integrarea modulului de generare a unei configurații pentru semafoare în cadrul simulatorului, acesta a fost testat folosind tehnica de testare manuală. În primul rând s-a test interfața grafică a acestui model. S-a verificat comportamentul sistemului în cazul în care se introduc valori nepermise, ca de exemplu orice alte caractere față de numere întregi, care reprezintă numărul de secunde pentru care faza respectivă este menținută. Totodată, s-a testat și cazul în care câmpurile pentru timpii respectivi rămân goi. În fiecare dintre aceste cazuri modulul a răspuns cu un mesaj de eroare informativ pentru utilizator. Un alt aspect important ce a fost testat în cazul acestui modul a fost generarea unui fișier de tip JSON corect. Pentru a putea testa aceasta s-a generat în primul rând un fișier de configurare arbitrar, după care s-a inspectat conținutul acestui fișier identificând astfel eventualele erori de sintaxă sau logice.

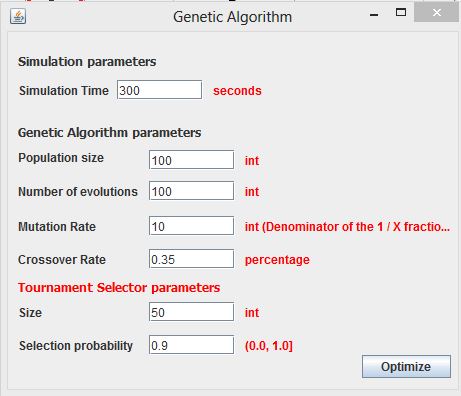
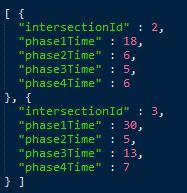
Pentru testarea modulului de optimizare folosind algoritmul genetic s-a testat în primă fază librăria JGAP. S-a creat o mini aplicație de tipul *Proof of Concept* (*PoC*) prin care s-a încercat validarea librăriei folosite. Această aplicație a fost asemănătoare cu cea prezentată ca și exemplu în documentația librăriei. După efectuarea testelor și având certitudinea că algoritmul implementat în cadrul librăriei este unul corect a început procesul de integrare a acesti librării în cadrul modulului de optimizare din simulator. Pentru testarea acestui modul la nivel de interfață grafică s-au aplicat aceleași metode de testare ca și în cazul modului de generare a unei configurații pentru semafoare. Astfel, s-a încercat introducerea unor valori greșite care pe urmă au fost semnalizate de către modul. Fișierul care rezultă în urma procesului de optimizare a fost de asemenea verificat din punct de vedere sintactic și logic. Acest fișier trebuie să fie unul asemănător cu cel generat de către modulul de generare a unei configurații de semafoare.

În continuare s-a testat întreg procesul de simulare prin definirea unui scenariu din trafic și încercarea optimizării acesteia. Se vor prezenta pașii acestui caz de testare.

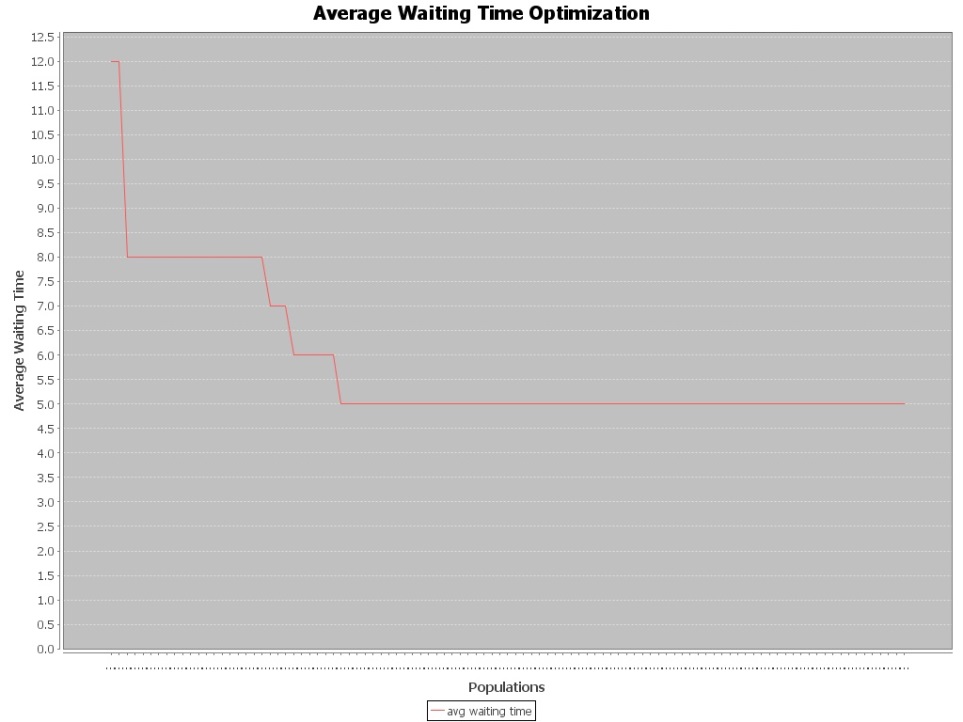
*Figura 6.1 Mediul de simulare testat*

În *figura 6.1* avem reprezentarea mediului de simulare care va fi supus testării. După cum observăm vom avea două intersecții principale (*intersecția 2* și *intersecția 3*), iar în rest șase puncte de intrare, respectiv ieșire din simulare. Aceste puncte sunt conectate la cele două intersecții prin intermediul segmentelor.

În continuare s-a generat un flux de mașini prin intermediul modulului de generare a fluxurilor, care va avea ca rută fixă următoarea listă cu identificatori: *1 -> 2 -> 3 -> 6*. Timpul de generare a mașinilor a fost setat la 7 secunde. Fișierul JSON rezultat în urma procesului de generare a fost încărcat în simulator.



*Figura 6.2 Valori test pentru optimizator și fișierul rezultat*

Pasul următor reprezintă pornirea procesului de optimizare prin algoritmul genetic. În *figura 6.2* putem observa valorile parametrilor algoritmului genetic introduși pentru testare și fișierul rezultat. Astfel, ca și timp de simulare s-a ales *300* de secunde, mărimea populației de cromozomi *100*, numărul de evoluții *100*, rata în care se execută mutațiile *1/10*, rata de crossover *0.35*, mărimea unui turneu *50* și probabilitatea de selecție din cadrul turneului *0.9*. Fișierul de configurare rezultat conține timpii suboptimali al fazelor.

*Figura 6.3 Grafic proces optimizare*

În *figura 6.3* observăm reprezentarea grafică a procesului de optimizare. După cum vedem timpul mediu de așteptare a fost redus de la *12 secunde* la *5 secunde*. Acest grafic ne este prezentat la finalizarea întregului proces de simulare.

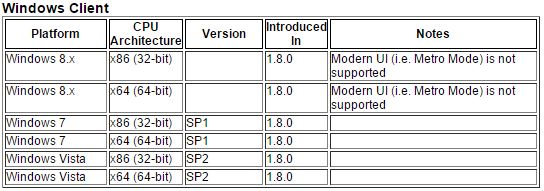
Pentru a valida configurația găsită vom încărca fișierul JSON rezultat de pe urma optimizării în simulator și vom porni procesul de simulare. Se poate observa prioritizarea traficului de pe direcția *Sud –> Nord* din *intersecția 2* și traficul de pe direcția *Sud –> Est* din *intersecția 3*. Aceste prioritizări au rezultat din timpii fazelor 1 configurați pe fiecare din cele două intersecții.

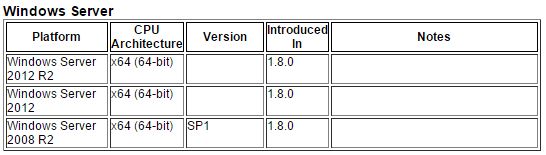
În urma acestui proces amplu de testare din punct de vedere al unui utilizator final putem afirma faptul că produdul este unul funcțional și respectă lista de funcțioalități impusă la începutul proiectării.

# Manual de Instalare si Utilizare

Sistemul de simulare constă dintr-un fișier de tip *jar* executabil. Acest fișier împachetează toate resursele și clasele simulatorului. Ceea ce face acest pachet executabil este invocarea unei metode tip *main* din rutina principală a programului. Pentru a putea rula acest *jar* e nevoie de *JDK* (Java Development Kit) cu o versiune de cel puțin *7*.

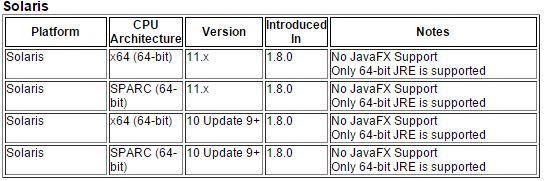
Resursele hardware și software necesare pentru rularea și instalarea aplicației sunt strâns legate de resursele necesare pentru rularea Java. Nu este descrisă nici o procedură de instalare a aplicației întrucât întegul proiect constă dintr-un singur fișier executabil de tip jar. În continuare se vor enumera câteva din platformele pe care se poate rula Java, respectiv simulatorul.



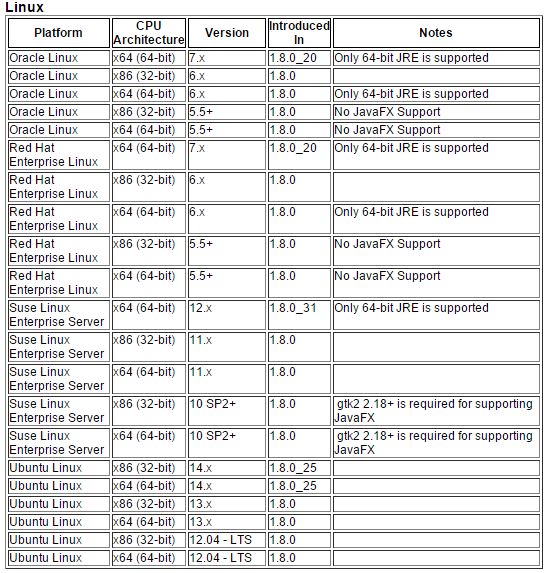


*Tabel 7.1 Cerințe pentru Windows*

*Tabel 7.2 Cerințe pentru Windwos Server*



*Tabel 7.3 Cerințe pentru Solaris*

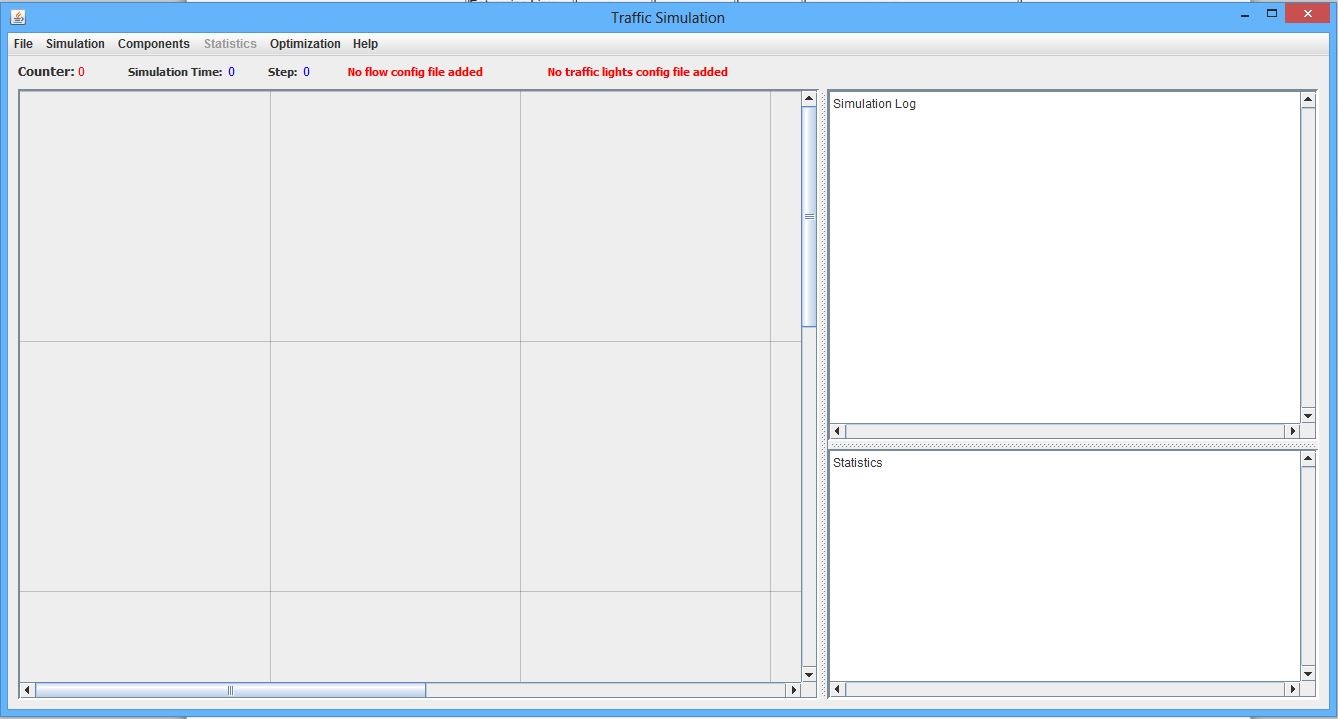


*Tabela 7.4 Cerințe pentru Linux*

În *tabelele* *7.1*, *7.2*, *7.3* și *7.4* sunt enumerate câteva din sistemele de operare pe care Java poate să ruleze, precum și anumite cerințe specifice platformelor respective. Câteva din cerințele hardware standard pentru *Java 7* sunt după cum urmează:

* *RAM*: 128 MB
* *Spațiu disponibil*: 124 MB pentru JRE
* *Procesor*: minim Pentium 2 cu 266 MHz

În continuare se va descrie folosirea aplicației din punct de vedere a utilizatorului final. În momentul pornirii simulatorului utilizatorul va putea observa apariția panoului principal de simulare. Acest panou conține un meniu, o secțiune de informații cu privire la simulare, un panou folosit pentru reprezentarea vizuală a simulării, un panou folosit pentru afișarea de mesaje tip log și nu în ultimul rând un panou pentru afișarea statisticilor cu privire la simulare.

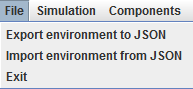
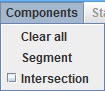


*Figura 7.1 Panoul principal al simulatorului*

*Figura 7.1* reprezintă panoul principal al simulatorului.

Meniul panoului de simulare conține următoarele elemente: *File*, *Simulation*, *Components*, *Statistics*, *Otimization* și *About*.

*(d)*

C:\Users\edWin\Desktop\LicentaGIT\trunk\utilOptimz.jpg

*(b)*

*(c)*

*(a)*

*Figura 7.2 Elementele principale ale meniului mediului de simulare*

*Figura 7.2* conține elementele meniului din mediul de simulare. Astfel, avem următoarele elemente:

* ***File –> Export environment to JSON***– folosit pentru a salva întreg mediul de simulare într-un fișier
* ***File –> Import environment from JSON*** – folosit pentru a încărca un mediu de simulare deja existent dintr-un fișier
* ***File –> Exit*** – folosit pentru închiderea aplicației
* ***Simulation –> New*** – folosit pentru crearea unei simulări noi
* ***Simulation –> Generate flow config file*** – folosit pentru deschiderea modulului de generare a unor fluxuri de mașini
* ***Simulation –> Add flow config file*** – folosit pentru încărcarea unui fișier deja existent conținând fluxuri de mașini
* ***Simulation –> Generate traffic lights config file*** – folosit pentru deschiderea modulului de configurare a semafoarelor din intersecții
* ***Simulation –> Add traffic lights config file*** – folosit pentru încărcarea unui fișier deja existent conținând configurări pentru semafoare
* ***Simulation –> Start*** – folosit pentru pornirea procesului de simulare
* ***Simulation –> Pause*** – folosit pentru a opri simularea pentru câteva clipe
* ***Simulation –> Resume*** – folosit pentru continuarea procesului de simulare
* ***Components* *–> Clear all*** – folosit pentru ștergerea tuturor componentelor care apar în mediul de simulare
* ***Components* *–> Segment*** – folosit pentru a crea un segment între două intersecții deja existente
* ***Components* *–> Intersection*** – folosit pentru a crea o intersecție sau un punct de pornire/oprire. Cât timp componenta *Intersection* va rămâne selectată utilizatorul va putea continua să adauge intersecții în mediul de simulare
* ***Optimization –> Genetic algorithm*** – folosit pentru deschiderea modulului de optimizare cu algoritm genetic
* ***Help* *–> About*** – informații cu privire la sistemul de simulareC:\Users\edWin\Desktop\LicentaGIT\trunk\utilINFO1.JPG

*(a)*

C:\Users\edWin\Desktop\LicentaGIT\trunk\utilINFO2.JPG

*(b)*

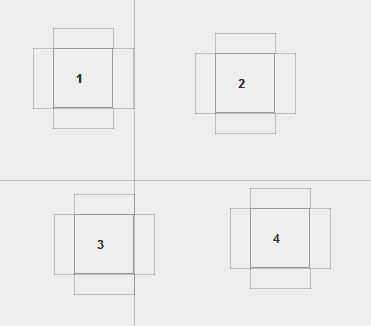
*Figura 7.3 Informații cu privire la simulare*

În *figura 7.3* putem observa cele două stări ale panoului cu informații. Prima variantă *(a)* apare în cazul în care fișierele de configurare a fluxurilor și semafoarelor nu au fost încărcate, iar cea de-a doua variantă în caz contrar. Totodată, se mai poate observa timpul total de simulare, pasul de simulare și un contorizator de timp al simulării.

Pentru a crea un scenariu nou de simulare utilizatorul are la dispoziție două variante:

* încărcarea din fișier a unui mediu de simulare deja existent, din meniul *File –> Import environment from JSON*
* construirea unui mediu de simulare nou cu ajutorul instrumentelor găsite în meniul *Components*

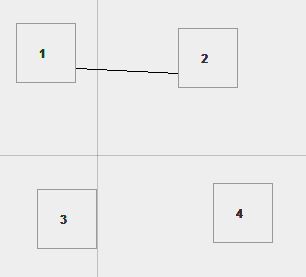
Primul pas din construcția unui mediu nou de simulare este amplasarea intersecțiilor. Pentru aceasta se va alege din meniul *Components* elementul ***Intersection***. Se observă faptul că acest element va rămâne activ până când utilizatorul va dezactiva elementul respectiv. În urma amplasării intersecțiilor utilizatorul va putea să interconeteze acestea folosind segmente. Acest element poate fi activat din mediul *Components* -> *Segment*. Se va observa apariția unor zone dreptunghiulare în jurul tuturor intersecțiilor. Aceste zone vor reprezenta spațiul în care utilizatorul va putea să aplice acest element. Sunt permise câte două segmente pe fiecare parte a intersecțiilor.



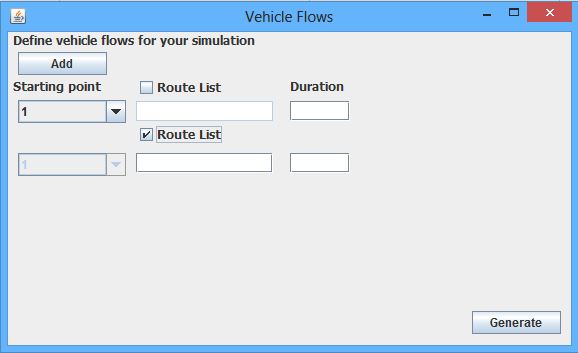
*Figura 7.4 Intersecțiile după selectarea elementului segment*

*Figura 7.4* reprezintă intersecțiile după alegerea elementului *segment* din meniul *Components*. Spațiile apărute vor reprezenta zonele din care vor putea fi trasate segmentele.

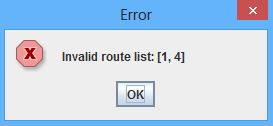
În momentul în care se va selecta o zonă din cele apărute, aceasta va fi colorată cu culoarea roșie. La interconectarea a două intersecții, segmentul va fi așezat pe direcția selectării acestora. De exemplu dacă se va selecta zona estică a intersecției cu identificatorul 1, după care se va selecta zona vestică a intersecției cu identificatorul 2, segmentul rezultat va avea sensul de mers dinspre intersecția 1 spre intersecția 2 (vezi *figura 7.5*). În cazul în care există deja un astfel de segment, această operație va fi ignorată.



*Figura 7.5 Segmentul rezulat*

Pentru generarea unor fluxuri de mașini se va deschide modulul din meniul *Simulation -> Generate flow config file*. Utilizatorul poate să își definească două tipuri de fluxuri de mașini: cu un punct de pornire reprezentat printr-un identificator și cu rută fixă reprezentată de o serie de identificatori. Durata unui flux va fi specificată printr-un număr întreg care va reprezenta timpul în secunde.

*Figura 7.6 Generatorul de fluxuri*

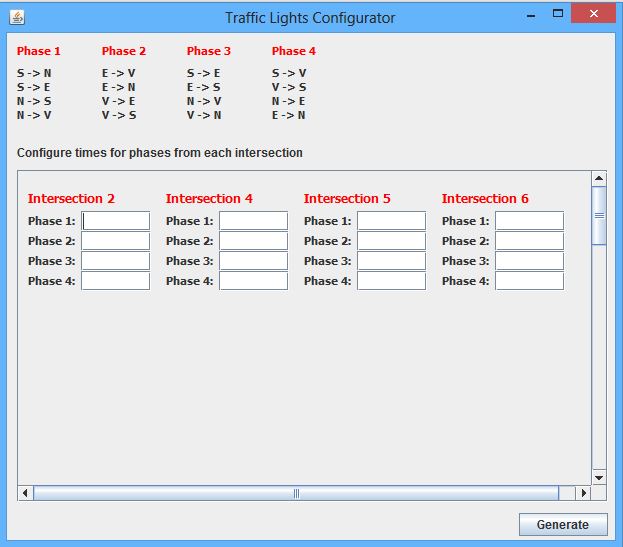
 *Figura 7.6* reprezintă modulul de generare a fluxurilor de mașini. Punctul de pornire a fluxului se specifică prin alegerea unui identificator dintr-un *drop box*, iar ruta fixă se specifică într-un *text box*. Ruta va trebuie să conțină în primul rând identificatorii unui punct de pornire, respectiv punct de ieșire și identificatorii intersecțiilor doriți. Formatul în care se vor introduce identificatorii va fi: ***idPunctPornire id1 id2 ... idn idPunctOprire***. Lista de identificatori specificată de utilizator va fi validată de către sistem.

*Figura 7.7 Eroare la generare*

*Figura 7.7* reprezintă un exemplu de mesaj de eroare apărut în cazul în care s-a specificat o rută invalidă de către utilizator. Mesajul va conține ruta cu probleme.

Pentru generarea de configurații pentru semafoare se va deschide modulul din

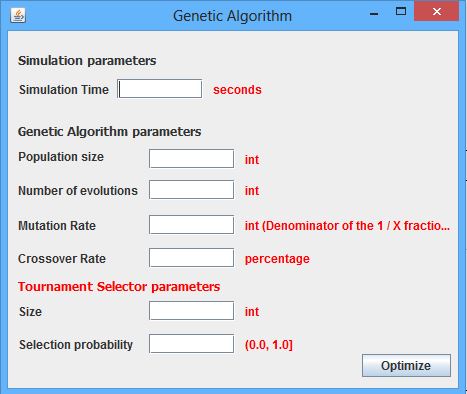
*Simulation –> Generate traffic lights config file*. Utilizatorul va putea să definească configurații pentru toate semafoarele din intersecțiile vizibile în mediul de simulare. Acesta va putea specifica timpii pentru fiecare din cele patru faze ale intersecțiilor. Timpii vor fi specificați prinr-o valoare întreagă care vor reprezenta secundele pentru care faza respectivă va fi în funcțiune.



*Figura 7.8 Generatorul de configurații pentru semafoare*

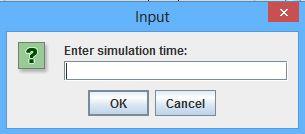
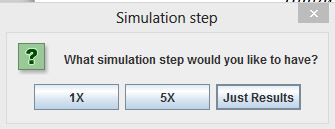
*Figura 7.8* reprezintă modulul de generare a configurațiilor pentru semfoarele din intersecții. Prima secțiune a modulului conține definirea celor patru faze prezente în cadrul intersecțiilor din mediul de simulare.

În urma definirii celor două fișiere, acestea vor trebui încărcate în mediul de simulare din meniul *File –> Import environment from JSON*. Dacă aceste fișiere vor fi adăugate cu succes va trebui să observăm un mesaj informativ cu numele fișierelor adăugate ca și în *figura 7.3-b*.

Pentru procesul de optimizare se va deschide acest modul prin meniul *Optimization –> Genetic algorithm*. Aici se vor putea specifica diverși parametri pentru algoritmul genetic.

*Figura 7.9 Modulul de optimizare cu algoritm genetic*

*Figura 7.9* reprezintă modulul de optimizare cu algoritm genetic. Fiecare paramateru configurabil din cadrul algoritmului genetic are informațiile necesare pentru introducerea unei valori valide. În timpul în care se produce optimizarea pe ecran se va afișa o bară de progres. La sfârșitul procesului modulul va produce un fișier cu o configurație suboptimală a timpilor fazelor, precum și un grafic cu întreg procesul.

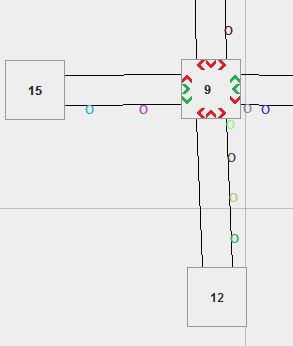
Pentru pornirea procesului de simulare se va alege din meniul *Simulation* opțiunea *Start*. Utilizatorul va specifica timpul de simulare în secunde, precum și pasul de simulare.

*(b)*

*(a)*

*Figura 7.10 Timpul și pasul de simulare*

*Figura 7.10* reprezintă timpul și pasul de simulare care vor trebui specificați de către utilizator. Pasul de simulare va putea fi de *1X*, simulare în timp real, *5X* sau doar afișarea rezultatelor/statisticilor finale.

 În momentul pornirii simulării vor putea fi observate mașinile care își urmăresc rutele lor interne. Simularea poate fi oprită pentru câteva clipe de către utilizator prin meniul *Simulation –> Pause*, respectiv reluată prin meniul *Simulation –> Resume***.** În caz că utilizatorul dorește să creeze o simulare nouă, se va apela meniul *Simulation –> New****.***

*Figure 7.11 Deplasarea autovehiculelor*

În *figura 7.11* observăm deplasarea autovehiculelor din punctele de intare *12* și *15* în intersecția cu identificatorul *9*.

# Concluzii

În urma procesului de modelare și dezvoltare a acestui sistem de simulare a traficului urban s-a ajuns la o reprezentare apropiată de realitate în ceea ce privește comportamentul autovehiculelor din traficul urban. Prin procesul de simulare pas-cu-pas s-a reușit reprezentarea în timp real a efectului configurațiilor de semafoare avut asupra fluxurilor de autovehicule din trafic. Datorită flexibilității oferite de fiecare semafor în ceea ce privesc timpii fazelor s-au putut observa diferențe considerabile între timpii de așteptare a autovehiculelor din intersecții.

Algoritmul genetic din cadrul modulului de optimizare reprezintă componenta centrală în ceea ce privește optmizarea. Datorită flexibilității algoritmului în privința parametrilor care pot fi configurați se pot obține rezultate diferite la fiecare proces de optimizare. Rezultatele obținute se vor apropia de fiecare dată de cel optim, neavând însă certitudinea că soluția curentă este cea optimă. Algoritmul genetic nu garanteză prin definiție găsirea soluției optime.

Produsele software nu pot fi considerate niciodată finalizate existând întotdeauna părți care pot fi îmbunătățite sau funcționalități noi care pot fi adăugate. Chiar și în cazul sistemului nostru de simulare există posibile dezvoltări și îmbunătăţiri ulterioare. Astfel, putem avea:

* posibilitatea efectuării procesului de import și export folosind alte tipuri de fișiere, de exemplu XML-uri
* includerea conceptului de zoom în mediul de simulare
* vizualizarea autovehiculelor care traversează o intersecție
* eficientizarea procesului de construcție a mediului de simulare
* adăugarea opțiunii de Undo și Redo în cadtrul procesului de construcție a mediului de simulare
* posibilitatea ștergerii unor anumite componente din mediul de simulare
* posibilitatea definirii unor scenarii de simulare care cuprind pietoni, fiind elemente care pot avea un impact important asupra procesului de simulare
* pentru a avea pietoni care traverseză un anumit tronson trebuie definit conceptul de trecere de pietoni
* definirea unui modul de generare a pietonilor
* includerea unei componente care este reprezentarea unui sens giratoriu din traficul urban
* modelarea autovehiculelor după stilul de conducere a șoferului (agresiv, defensiv)
* introducerea mai multor tipuri de autovehicule care au viteză de deplasare diferită
* modelarea conceptului de depășire între două autovehicule
* definirea mai multor benzi pe o direcție de mers
* modelarea intersecțiilor cu mai mult de patru faze
* extinderea modulului folosit pentru înregistrarea statisticilor simulării
* definirea altor metode de optimizare a traficului urban și posibilitatea comparării acestora

Sistemul de simulare implementat nu reflectă în totalitate comportamentul unui trafic urban mai complex. Există mai multe elemente din trafic care au fost omise în procesul de dezvoltare a aplicației. Traficul urban și infrastructura mediului în care a avut loc simularea este unul simplu, având la dispoziție câte o bandă de mers pe fiecare sens și un singur tip de autovehicul, respectiv șofer. Datorită acestui fapt rezultatele obținute pot fi considerate valide doar în cazul unei infrastructuri cu o singură bandă pe fiecare sens și în care șoferii au același comportament, respectând toate legile rutiere existente. În realitate însă știm foarte bine că lucrurile nu stau chiar așa, având parte de șoferi agresivi sau mai defensivi care pot impacta rezultatele simulării.

# Bibliografie

[1] Marco Wiering, Jilles Vreeken, Jelle van Veenen, and Arne Koopman, "Simulation and Optimization of Traffic in a City" in *IEEE Intelligent Vehicles Symposium*, 2004, pp. 453 – 458.

[2] Ahmed A. Ezzat, Hala A. Farouk, Khaled S. El-Kilany, Ahmed F. Abdel Moneim, "Optimization Using Simulation of Traffic Light Signal Timings" in *International Conference on Industrial Engineering and Operations Management*, Bali, Indonesia, 2014.

[3] Tan Nguyen, "A Flexible Model For Traffic Simulation And Traffic Signal Control Optimization" in *COMP3740*, Australia, 2014.

[4] Fitsum Teklu, Agachai Sumalee and David Watling, "A Genetic Algorithm Approach for Optimizing Traffic Control Signals Considering Routing" in *Computer-Aided Civil and Infrastructure Engineering*, Volume 22. Issue 1, pp. 31 – 43, 2007.

[5] P. Cortés, J. Larrañeta, L. Onieva, "Genetic algorithm for controllers in elevator groups: analysis and simulation during lunchpeak traffic" in *Applied Soft Computing*, Volume 4, Issue 2, pp. 159 – 174, 2004.

[6] Karima BENHAMZA, Salah ELLAGOUNE, Hamid SERIDI and Herman AKDAG, "Agent-based modeling for traffic simulation" at *University of Biskra*, 2013.

[7] Tiffany Hunt and David Nikaido, "Multi-agent Behavior in a GIS Traffic Simulation" at *Trinity Summer REU* 2010.

[8] Charles M. Macal, Michael J. North, "AGENT-BASED MODELING AND SIMULATION: ABMS EXAMPLES" at *Argonne National Laboratory*

*Argonne*, IL 60439, USA.

[9] Larry E. Owen , Yunlong Zhang, Lei Rao, Gene McHale, "Street and traffic simulation: traffic flow simulation using CORSIM" *at Society for Computer Simulation International San Diego*, CA, USA, 2000.

[10] Java Genetic Algorithms Package (JGAP), <http://jgap.sourceforge.net/>.

[11] IntelliJ, <https://www.jetbrains.com/idea/>.

[12] Genetic Algorithm, <http://www.otletinstitute.org/wikics/Genetic_Algorithms.html>

# Anexa 1

public class GeneticAlgorithmOptimization {

public static int MINIMUM\_PHASE\_TIME = 5;

public static int MAXIMUM\_PHASE\_TIME = 30;

private static List<Double> fitnessValues;

/\*\*

\* Start genetic algorithm.

\*

\* @param view TrafficSimulationView instance.

\* @param populationSize the size of the population.

\* @param evolutions the number of evolutions.

\* @param mutationRate

\* @param crossoverPercentage

\* @param tournamentSize

\* @param selectionProbability

\* @return

\* @throws InvalidConfigurationException

\*/

public static List<TrafficLightPhases> *optimize*(TrafficSimulationView view, int populationSize, int evolutions, int mutationRate, double crossoverPercentage, int tournamentSize, double selectionProbability) throws InvalidConfigurationException {

fitnessValues = new ArrayList<>();

// set default configuration for GA

Configuration conf = new Configuration("MyConfiguration");

conf.setBreeder(new GABreeder());

conf.setRandomGenerator(new StockRandomGenerator());

conf.setEventManager(new EventManager());

TournamentSelector e = new TournamentSelector(conf, tournamentSize, selectionProbability);

conf.addNaturalSelector(e, false);

conf.setMinimumPopSizePercent(0);

conf.setSelectFromPrevGen(1.0D);

conf.setKeepPopulationSizeConstant(true);

conf.setFitnessEvaluator(new DefaultFitnessEvaluator());

conf.setChromosomePool(new ChromosomePool());

conf.addGeneticOperator(new CrossoverOperator(conf, crossoverPercentage));

conf.addGeneticOperator(new MutationOperator(conf, mutationRate));

FitnessFunction myFitnessFunction = new MinimizingWaitingTimeFitnessFunction(view);

conf.setFitnessFunction(myFitnessFunction);

Genotype.setStaticConfiguration(conf);

List<Integer> ids = new ArrayList<>();

for (Intersection intersection : view.getIntersections()) {

if (intersection.isFourPhased()) {

ids.add(intersection.getId());

}

}

Gene[] intersectionGenes = new Gene[ids.size()];

// create genes

for (int i = 0; i < ids.size(); i++) {

CompositeGene intersectionGene = new CompositeGene();

Gene p1 = new IntegerGene(conf, MINIMUM\_PHASE\_TIME, MAXIMUM\_PHASE\_TIME);

Gene p2 = new IntegerGene(conf, MINIMUM\_PHASE\_TIME, MAXIMUM\_PHASE\_TIME);

Gene p3 = new IntegerGene(conf, MINIMUM\_PHASE\_TIME, MAXIMUM\_PHASE\_TIME);

Gene p4 = new IntegerGene(conf, MINIMUM\_PHASE\_TIME, MAXIMUM\_PHASE\_TIME);

intersectionGene.addGene(p1);

intersectionGene.addGene(p2);

intersectionGene.addGene(p3);

intersectionGene.addGene(p4);

intersectionGenes[i] = intersectionGene;

}

Chromosome chromosome = new Chromosome(conf, intersectionGenes);

conf.setSampleChromosome(chromosome);

conf.setPopulationSize(populationSize);

Genotype population = Genotype.randomInitialGenotype(conf);

for (int i = 0; i < evolutions; i++) {

fitnessValues.add(population.getFittestChromosome().getFitnessValue());

System.out.println(i);

population.evolve();

}

IChromosome bestSolution = population.getFittestChromosome();

fitnessValues.add(bestSolution.getFitnessValue());

// generate and show line chart

generateLineChart();

List<TrafficLightPhases> trafficLightPhaseses = new ArrayList<>();

for (int i = 0; i < bestSolution.getGenes().length; i++) {

CompositeGene intersectionGene = (CompositeGene) bestSolution.getGenes()[i];

trafficLightPhaseses.add(new TrafficLightPhases(ids.get(i),

(int) intersectionGene.getGenes().get(0).getAllele(),

(int) intersectionGene.getGenes().get(1).getAllele(),

(int) intersectionGene.getGenes().get(2).getAllele(),

(int) intersectionGene.getGenes().get(3).getAllele()));

}

return trafficLightPhaseses;

}

/\*\*

\* Generate and show line chart of waiting times.

\*/

private static void generateLineChart() {

DefaultCategoryDataset line\_chart\_dataset = new DefaultCategoryDataset();

for (int i = 0; i < fitnessValues.size(); i++) {

line\_chart\_dataset.addValue(MinimizingWaitingTimeFitnessFunction.BIG\_NUMBER - fitnessValues.get(i),

"avg waiting time", String.valueOf(i + 1));

}

JFreeChart lineChartObject = ChartFactory.createLineChart(

"Average Waiting Time Optimization",

"Populations",

"Average Waiting Time",

line\_chart\_dataset, PlotOrientation.VERTICAL,

true, true, false);

int width = 1024; /\* Width of the image \*/

int height = 768; /\* Height of the image \*/

File lineChart = new File("GALineChart.jpeg");

try {

ChartUtilities.saveChartAsJPEG(lineChart, lineChartObject, width, height);

JLabel label = new JLabel(new ImageIcon("GALineChart.jpeg"));

JFrame f = new JFrame();

f.setTitle("Genetic Algorithm Results");

f.getContentPane().add(label);

f.pack();

f.setLocation(100, 100);

f.setVisible(true);

} catch (IOException e1) {

e1.printStackTrace();

}

}

}