

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA: Tehnologia informației

Traffic Simulator

LUCRARE DE DIPLOMĂ

Coordonator științific
S.l.dr.ing. Adrian Alexandrescu

Absolvent
Răzvan-Andrei Zavalichi

Iași, 2019

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII LUCRĂRII DE DIPLOMĂ

Subsemnatul(a) Zavalichi Răzvan-Andrei,
legitimat(ă) cu C.I. seria NT nr. 758951, CNP 1960328271696
autorul lucrării TRAFFIC SIMULATOR

elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea IULIE a anului universitar 2019, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data

01.07.2019

Semnătura

Cuprins

Introducere.....	1
Capitolul 1. Fundamente teoretice.....	2
1.1. Traficul rutier.....	2
1.2. Mediul urban.....	2
1.3. Structura sistemelor de trafic.....	3
1.4. Soluția propusă.....	4
1.5. Aplicații similare.....	5
1.5.1. CORSIM.....	5
1.5.2. PARAMICS.....	5
1.5.3. VISSIM.....	6
Capitolul 2. Proiectarea Aplicației.....	8
2.1. Platforma de dezvoltare.....	8
2.1.1. Opțiuni disponibile.....	8
2.1.1.1. Unity.....	8
2.1.1.2. Blender.....	9
2.2. Tehnologii folosite.....	10
2.2.1. Limbajul C#.....	10
2.2.2. Platforma .NET.....	10
2.3. Obiective.....	11
2.3.1.1. Crearea mediului de simulare.....	11
2.3.1.2. Crearea participanților la trafic.....	12
2.3.1.3. Crearea semafoarelor.....	15
Capitolul 3. Implementarea aplicației.....	17
3.1. Mediul pentru simulare.....	17
3.2. Căile rutiere.....	20
3.2.1. Nodurile.....	20
3.2.1.1. Meniul pentru noduri.....	21
3.3. Autovehiculele.....	22
3.3.1. Mașina reală.....	23
3.3.2. Mașina simplă.....	26
3.3.3. Mașina controlată.....	28
3.4. Ruta optimă.....	29
3.5. Indicatoarele rutiere.....	29
3.5.1. Intersecția.....	30
3.5.2. Curbă periculoasă.....	31
3.5.3. Cedează trecerea.....	31
3.5.4. Stop.....	33
3.5.5. Semafor.....	34
3.5.6. Funcționalitatea semaforului.....	34
3.5.7. Comunicarea cu autovehiculele.....	35
3.5.8. Intersecție inteligentă.....	36
3.5.9. Salvarea configurațiilor.....	36
3.6. Camera.....	37

3.7. Probleme întâmpinate.....	37
3.8. Funcționarea aplicației.....	38
3.9. Interfața cu utilizatorul.....	39
3.9.1. Setările aplicației.....	39
3.9.2. Meniul „Tools”.....	39
Capitolul 4. Testarea aplicației.....	44
4.1. Testarea componentelor.....	44
4.2. Limitări.....	44
Concluzii.....	45
Bibliografie.....	46
Anexe.....	47
Anexa 1. Diagrama UML de clase.....	47
Anexa 2. Diagrama UML de secvență a participanților la trafic.....	48
Anexa 3. Diagrama UML de utilizare.....	49
Anexa 4. Exemplu conținut fișier OSM.....	50
Anexa 5. Algoritmul Dijkstra.....	51
Anexa 6. Algoritmul A*.....	53

Traffic Simulator

Răzvan-Andrei Zavalichi

Rezumat

La ora actuală, traficul rutier reprezintă una din marile probleme cu care se confruntă majoritatea orașelor. Acest lucru este cauzat de semafoarele cu timpii de aşteptare prea mari sau prea mici, din cauza infrastructurii defectuoase dar și a numărului autovehicule care, în prezent, se află într-o continuă creștere. Noile mașini vin în dotare cu din ce în ce mai mulți senzori, utilizați pentru a ajuta conducătorul auto să ia decizii cât mai bune într-un moment cât mai scurt. Cu toate acestea, accidentele nu întarzie să apară. Cauza principală a accidentelor este neatenția, dar și factorii precum: viteza excesivă, oboseala, consumul de alcool la care se adaugă și condițiile meteorologice nefavorabile sau configurațiile greșite ale intersecțiilor sau ale semafoarelor.

Pentru a rezolva această problemă sunt folosite modelele de simulare a traficului. Acestea joacă un rol foarte important existând astfel posibilitatea de a studia evenimente de trafic ce nu pot fi analizate prin alte mijloace directe. Aceste modelele oferă posibilitatea de a evalua strategii de optimizare și planificare a traficului pe anumite rute și la anumite ore fără a folosi resurse costisitoare și consumatoare de timp necesare pentru implementarea strategiilor alternative din domeniu. Pentru acest motiv, modelele de simulare permit analize cu cost redus și rapide a situațiilor din trafic și oferă diverse soluții pentru a reduce blocajele din trafic și riscul de a face accident. Modelele de simulare, în același timp, ajută și la îmbunătățirea procesului de luare a deciziilor de către specialiștii și planificatorii din transporturi.

Programele software de optimizare a semafoarelor evoluează luând în considerare următoarele aspecte: ușurința utilizării, informațiile necesare și eficiența rezultatelor.

Simulările realizate de către programele software se studiază în trei perioade ale zilei (ora de vârf de dimineață, după-amiază și la prânz) precum și în anumite perioade ale anului când se aşteaptă diferite evenimente (Crăciunul, Paștele, concerte etc) și se evaluatează optimizarea programelor de coordonare și semnalizare cu ajutorul programelor de simulare a traficului.

Soluția propusă pentru rezolvarea problemei traficului rutier este un program software, care permite vizualizarea 3D a acestuia, unde se pot face diverse simulări în anumite zone dintr-un oraș. Pentru a optimiza programele de coordonare și semnalizare, simulările se pot realiza cu un număr de autovehicule stabilit de utilizator. Această aplicație software prezintă un meniu ușor de utilizat prin intermediul căruia utilizatorul poate configura după bunul plac semafoarele din intersecții precum și intervenția acestuia în simulare prin crearea de accidente, blocaje și conducerea unui autovehicul. Pentru a putea observa rezultatele obținute în urma optimizării aplicațiilor de semaforizare, aplicația software permite generarea rutei optime pentru a observa noua rută actualizată.

Introducere

Studiile de trafic prezentate de literatura de specialitate consideră că principalele cauze ale congestionării traficului rutier sunt reprezentate de: condițiile meteorologice nefavorabile, evenimentele neprevăzute din trafic . efectuarea operațiilor de reabilitare a infrastructurii rutiere. timpi de semaforizare ce nu sunt adaptați condițiilor reale din trafic, în funcție de zi și oră sau de condițiile atmosferice. Formele de dirijare a autovehiculelor și pietonilor într-o anumită zonă rutieră reprezintă de fapt conducerea traficului rutier, ce are ca obiectiv principal optimizarea în vederea fluidității traficului.

Metodele de conducere a traficului rutier implică pe lângă soluțiile adoptate în vederea fluidizării traficului și metodele de gestionare a evenimentelor special generate de ambulanță, poliție, pompieri sau a blocajelor pe anumite artere rezultate în urmă accidentelor rutiere sau a lucrărilor de menenanță.

Metodele de dirijare și conducere a intersecțiilor semaforizate sunt clasificate în patru categorii:

- *Conducere statică* – se bazează pe timpi prestabiliți pentru fiecare fază a ciclului de semaforizare;
- *Conducere dinamică* – determinate de timpi de semaforizare calculați în funcție de condițiile reale de trafic;
- *Conducere prin coordonare* – se realizează prin timpi de semaforizare stabiliți de către un anumit punct de control al traficului pe baza informațiilor generate de trafic;
- *Conducere la cerere* – activată prin apăsarea unui buton de către pietoni;

În fiecare zi ne întâlnim cu problema traficului care de multe ori duce la diferite tipuri de accidente. Această problemă este cauzată de neatenția participanților la trafic sau de metodele de conducere și dirijare a intersecțiilor semaforizate. Accidentele de circulație rutieră reprezintă o cauză principală de mortalitate în întreaga lume și o problemă și o prioritate majoră în domeniul sănătății publice. Deși au fost luate multe măsuri pentru a rezolva această problemă, s-ar putea face mult mai mult.

Având în vedere importanța globală a traficului, o atenție deosebită ar trebui acordată siguranței rutiere și efortului de a furniza standarde care au o mare importanță pentru eficiență și siguranța navetăștilor (ONU, 2011). Fără existența și aplicarea normelor de siguranță, accidentele de trafic ar putea deveni principala cauză a decesului în lume, cu peste două milioane pe an.

În ciuda faptului că gestionarea siguranței traficului este posibilă, există încă diferențe semnificative în numărul de accidente de trafic între țări. Pentru a îmbunătăți siguranța traficului la nivel global, Adunarea Generală a Națiunilor Unite a adoptat o serie de rezoluții. O rezoluție care a fost subliniată în mod special este îmbunătățirea siguranței traficului rutier (A / RES / 64/255). Conform acestei rezoluții, perioada dintre 2011 și 2020 a fost anunțată ca deceniu pentru luarea de măsuri pentru îmbunătățirea siguranței traficului (ONU, 2011).

Capitolul 1. Fundamente teoretice

1.1. Traficul rutier

Traficul rutier a devenit din ce în ce mai important pentru economia statelor, mai ales în ultimele decenii. La nivel mondial se înregistrează un interes din ce în ce mai mare pentru acest mod de transport. Preocupările se îndreaptă în domeniul sistemelor inteligente de transport atât asupra infrastructurii rutiere, pentru crearea de condiții de siguranță pe toate categoriile de drumuri, punându-se un accent deosebit pe dezvoltarea serviciilor în sprijinul utilizatorilor sistemelor de transport, dar și asupra vehiculelor, prin introducerea pe scară largă a „inteligentei” la nivel de autovehicul. De altfel, comunicarea între vehicul și infrastructura de la sol a devenit o necesitate, pentru creșterea vitezei de deplasare, cunoașterea apriorică a condițiilor de desfășurare a traficului și extinderea siguranței deplasării oriunde s-ar desfășura aceasta. Se poate vorbi despre o tendință de mondializare a comunicațiilor cu aplicații în domeniul rutier, în special după apariția sistemelor de comunicații telefonice, cu dezvoltarea serviciilor oferite de acestea și odată cu apropiata lansare a sistemelor 5G. Toate aceste beneficii ale sistemelor de transport intelligent nu se pot obține fară un proces de analiză a caracteristicilor traficului, a modului în care acesta se desfășoară de-a lungul timpului, cum se comportă în cazul apariției unor evenimente, precum și a măsurilor de fluentizare cu efect real în trafic. Orice sistem de management al traficului, și mai ales cele destinate să funcționeze în mediul urban, se realizează pe baza unei analize minuțioase a zonei în care se va aplică, a caracteristicilor arterelor rutiere, a intersecțiilor și a sistemelor de semnalizare.

1.2. Mediul urban

Mediul urban în care populația locuiește, activează, se instruiește, își întreține, dezvoltă sau reface capacitatea de acțiune, este limitat la dimensiunile spațiului urban care trebuie distribuit echitabil între toate formele de existență menționate, rezervându-se, totodată, și o cotă necesară satisfacerii cerințelor de comunicare și deplasare între diferitele arii ce se constituie în spațiul rutier urban. Creșterea spectaculoasă a traficului rutier care însوșește dezvoltarea activităților urbane nu poate fi satisfăcută de o creștere corespunzătoare a spațiului rutier. Pentru aceasta, în toate mediile economice dezvoltate s-au încercat soluții de descongestionare, orientate pe două direcții:

- Ameliorarea amenajării spațiului rutier pentru creșterea gradului de utilizare și de îmbunătățire a caracteristicilor și parametrilor ce favorizează creșterea traficului;
- Îmbunătățirea indicilor de utilizare a spațiului concomitent cu îmbunătățirea parametrilor de desfășurare a traficului prin control și monitorizare.

Eficiența eforturilor de îmbunătățire în acest domeniu este condiționată de abordarea sistematică a elementelor ce compun un sistem de trafic rutier. Acestea sunt:

a. Spațiul rutier care cuprinde în configurația sa, căi rutiere, noduri rutiere (intersecții), lucrări speciale (porturi, tunele, refugii, serpentine, pante etc), care pot facilita sau restricționa traficul rutier. La acestea se mai adaugă lucrările speciale de semnalizare și protecție, de iluminat etc.

b. Participanții la trafic, care sunt:

- Vehicule, autovehicule (din care o categorie aparte o constituie transportul public);

- Pietoni și bicicliști;

Participanții la trafic au diferite caracteristici (viteze de parcurs, grade de ocupare a căilor rutiere precum și grade de securitate sau pericolozitate). De asemenea, participanții la trafic prezintă diferite priorități: grupurile de copii, bătrâni și invalizi, în cadrul categoriei de pietoni, sau autovehicule oficiale sau de intervenție (salvare, pompieri, poliție), în cadrul categoriei de vehicule.

c. Condițiile naturale și de mediu, care influențează desfășurarea traficului rutier și care acționează asupra spațiului rutier sau asupra participanților la trafic (vânt, nebulozitate, precipitații etc.);

Controlul traficului din sistemele de trafic rutier are ca obiectiv creșterea capacitatii de trafic a rețelelor rutiere în următoarele condiții:

- Creșterea eficienței pentru participanții la trafic (economie de timp și de carburanți, creșterea gradului de confort prin servicii de informații și de asistență service);
- Creșterea gradului de siguranță pentru participanții la trafic;
- Reducerea poluării mediului (poluare sonoră, a aerului și a apei etc.).

1.3. Structura sistemelor de trafic

Sistemele de trafic se compun din:

a. *Trasee, tronsoane cu unul sau dublu sens*, fiecare sens beneficiind de una sau mai multe fire sau culoare de circulație, din care un fir sau două pot fi rezervate transportului public;

b. *Noduri rutiere sau intersecții* care pot fi cu trei sau mai multe căi sau ramuri de acces. Configurația sistemelor de trafic se completează cu traversări sau refugii pietonale, cu locuri de parcare sau de întoarcere etc. și poate cuprinde mai multe tronsoane și noduri rutiere.

Nodurile rutiere cu un singur nivel sunt elementele cele mai sensibile pentru traficul rutier, deoarece prilejuiesc cele mai frecvente situații conflictuale între participanții la trafic, care se deplasează pe direcții diferite sau care-și schimbă direcția din momentul intrării în intersecție și până la părăsirea ei.

În abordarea reglementării traficului rutier în nodurile rutiere intervin următoarele grupe de parametri:

- *Parametri generali*, provenind din politica de transport într-o anumită regiune și prioritățile rezultate din acestea, tipologia și amplasarea intersecțiilor și aspectele juridice instituționale și sociale, respectiv statutul administrativ și finanțier, jurisdicția polițienească și de întreținere;
- *Parametri fizici*, rezultând din dimensiunile și alinierea intersecțiilor;
- *Parametri de trafic și de securitate*, rezultând din repartizarea fluxului și din variația sa orară, zilnică sau sezonieră, precum și aspecte legate de circulația pietonilor sau a vehiculelor de transport public, aspecte legate de securitate și de semnalizare;
- *Parametri de mediu* care au în vedere aspectele legate de: urbanism și peisaj, activitățile riverane și staționări, precum și nivelul de zgomot.

Separarea temporară a circulației conflictuale se poate realiza prin: reglementări prin reguli de prioritate și semnalizări fixe, reglementări prin agent și reglementări prin semnalizare luminoasă. Dacă prima este cea mai rapidă, iar cea de a doua este foarte flexibilă (ambele aplicabile în noduri secundare), reglementarea prin semnalizare luminoasă are o mare răspândire și o eficiență în creștere ca urmare a evoluției conceptuale și tehnice a soluțiilor adoptate și a gradului de automatizare la care se pretează.

Un sistem de reglementare a traficului pentru o intersecție întrunește:

- *Echipamente de achiziție a datelor privitoare la trafic.* În această categorie intră senzorii de tip buclă inductivă având diferite forme și dimensiuni și care se amplasează sub calea de acces în intersecție. În cazuri mai speciale, se utilizează detectoare radar sau cu raze infraroșii care prezintă dezavantajul că pot fi perturbate în cazul precipitațiilor abundente. Toate aceste sisteme dispun și de un detector la care se cuplează senzorul și care trimite semnalul necesar către tabloul de comandă.
- *Echipamente de semnalizare luminoasă* care sunt montate sub formă de baterii luminoase colorate ce se adresează diferitelor categorii de participanți la trafic rutier. Pe lângă bateriile principale mai există baterii repetitoare de semnal pentru repetarea în profunzime a semnalizării pentru a fi perceput de întregul grup de participanți la trafic.
- *Tabloul de comandă sau controlerul de trafic*, care este amplasat în proximitatea intersecțiilor și care recepționează semnalele de la detectoarele de trafic, aplică planul de semnalizare și comandă bateriile de semnalizare potrivit programelor implementate. Planul de semnalizare este un ansamblu de programe care asigură elaborarea soluțiilor pentru comanda optimă a semafoarelor, în raport cu fluctuațiile de trafic.

În cazul unei intersecții independente, se pot aplica programe de reglare la intervale fixe (sistemele ce se bazează pe o automatizare clasică), sau programe de reglare adaptivă care țin seama de fluctuațiile momentane ale traficului rezultate din măsurători. Reglarea adaptivă aplică algoritmi de programare aciclică (programe de reglare calculate instantaneu) sau ciclică (selectarea unui program de reglare dintr-o listă), testați și verificați ca având rezultate optime pentru anumiți parametri de trafic orar sau zilnic.

Intersecțiile unui tronson (parte dintr-un obiect) rutier, care funcționează în regim coordonat, beneficiază de un nivel de micro reglare care ține seama de parametrii de programare ai tronsonului rutier, iar în cazul unei rețele, coordonarea în ansamblu a traficului este încredințată unui nivel de macro reglare. Sistemele moderne adaptive includ echipamente de calcul capabile să susțină sistemele de programare prezentate, sisteme care sunt rezultatul unor importante etape pregătitoare de simulare și testare, aplicate fiecărei intersecții sau fiecărui sistem de trafic rutier având în vedere parametrii specifici ai acestora.

1.4. Soluția propusă

Pentru optimizarea în vederea fluidității traficului a propusă realizarea unei aplicații software de modelare a traficului rutier, putându-se observa comportamentul participanților la trafic pentru diferite configurații ale semafoarelor. Cu ajutorul acestei aplicații utilizatorul va putea urmări grafic comportamentul participanților la trafic în zona de simulare precum și analiza operării intersecțiilor prin supraîncărcarea uneia sau a mai multor rute sau prin crearea intenționată de accidente. De asemenea, aceasta oferă o soluție pentru optimizarea și modelarea rețelelor de trafic urban și încearcă să conțină toate cerințele unui program de analiză a traficului, principalul scop fiind crearea unui set de fișiere de configurații ale semafoarelor pentru anumite situații.

Aplicația software este una 3D astfel încât să ofere utilizatorilor informații cât mai fidele realității, aceasta făcând parte dintr-un proiect mai mare care este planificat pe viitor.

1.5. Aplicații similare

1.5.1. CORSIM

CORSIM (Corridor Simulation Manual and Resources) reprezintă un instrument de simulare a traficului, aplicabil pentru străzi, autostrăzi și rețele integrate cu o selecție completă a dispozitivelor de control (semne de circulație, semafoare și sisteme de acces pe autostradă). CORSIM simulează traficul și sistemele de control pentru trafic pe baza unor modele universal valabile privind comportamentul conducătorilor auto și al vehiculelor. CORSIM combină două din cele mai larg utilizate modele de simulare a traficului și anume NETSIM pentru străzi de suprafață și FREESIM pentru autostrăzi. Acest program permite, printre altele, modelarea comportamentului și efectului vehiculelor de mari dimensiuni, a incidentelor de trafic ce pot apărea pe benzile ocupate de aceste vehicule, accesul pe autostrăzi etc.

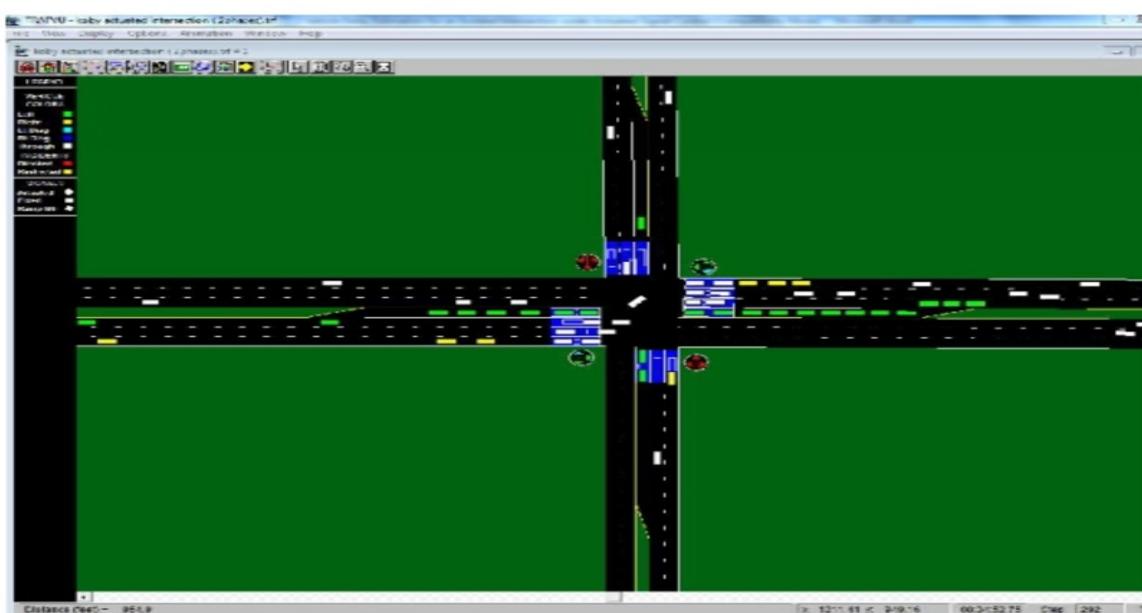


Figura 1.1: Aplicația CORSIM

1.5.2. PARAMICS

PARAMICS oferă o platformă puternică de simulare pentru realizarea de simulări a unei multitudini de situații din domeniul traficului și transportului rutier. El este bazat pe intercorelarea între mai multe module scalabile și proiectat pentru a rezolva scenarii ce se extind de la modelarea unei intersecții singulare până la autostrăzi congestionate de trafic sau la modelarea unui întreg sistem de transport pentru un oraș. La ora actuală, programul PARAMICS reprezintă o suită multifuncțională care a ajuns la versiunea 5.1 BETA. Ea cuprinde următoarele module: Modelator, Estimator, Procesor, Analizor, Programator, Monitor, Designer și Viewer.

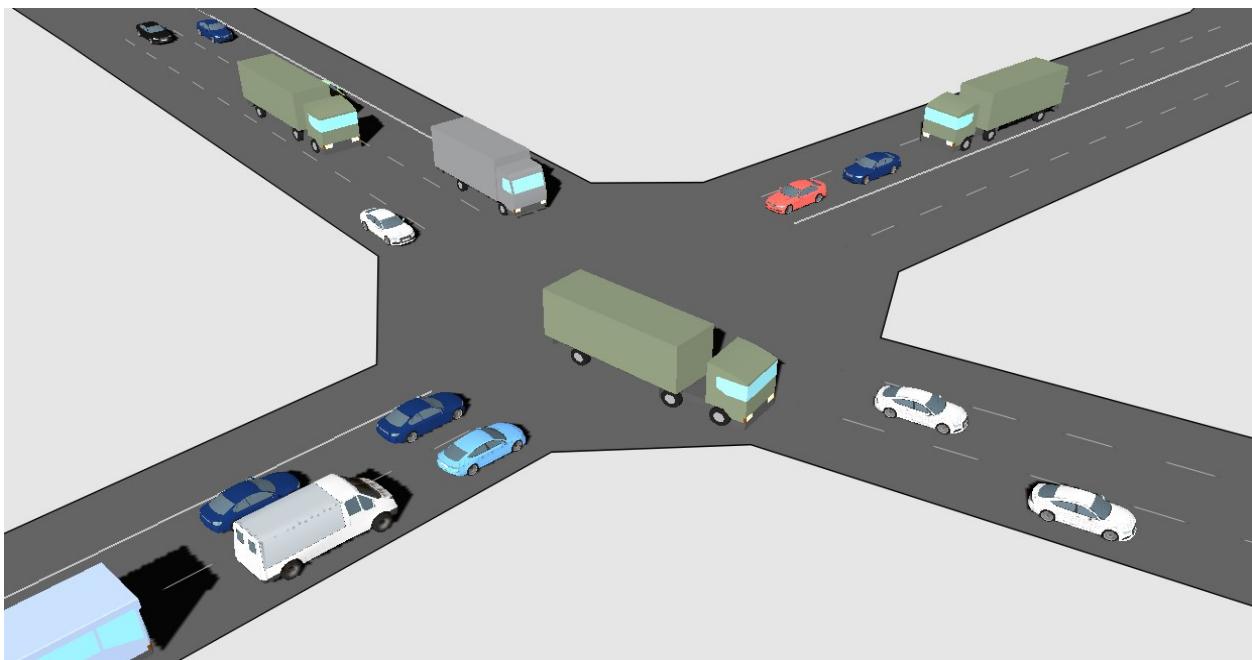


Figura 1.2: Aplicația PARAMICS

1.5.3. VISSIM

VISSIM – reprezintă un instrument de microsimulare, bazat pe comportament și având rol multifuncțional în analiza și simularea traficului. El oferă o serie de aplicații pentru analiza traficului urban sau pe autostrăzi și reprezintă un instrument de optimizare a sistemelor tehnice complexe. VISSIM permite integrarea transportului public sau a celui privat în simulații. Combinarea cu animația 3D și cu instrumentele de analiză oferă condiții de analiză deosebite.

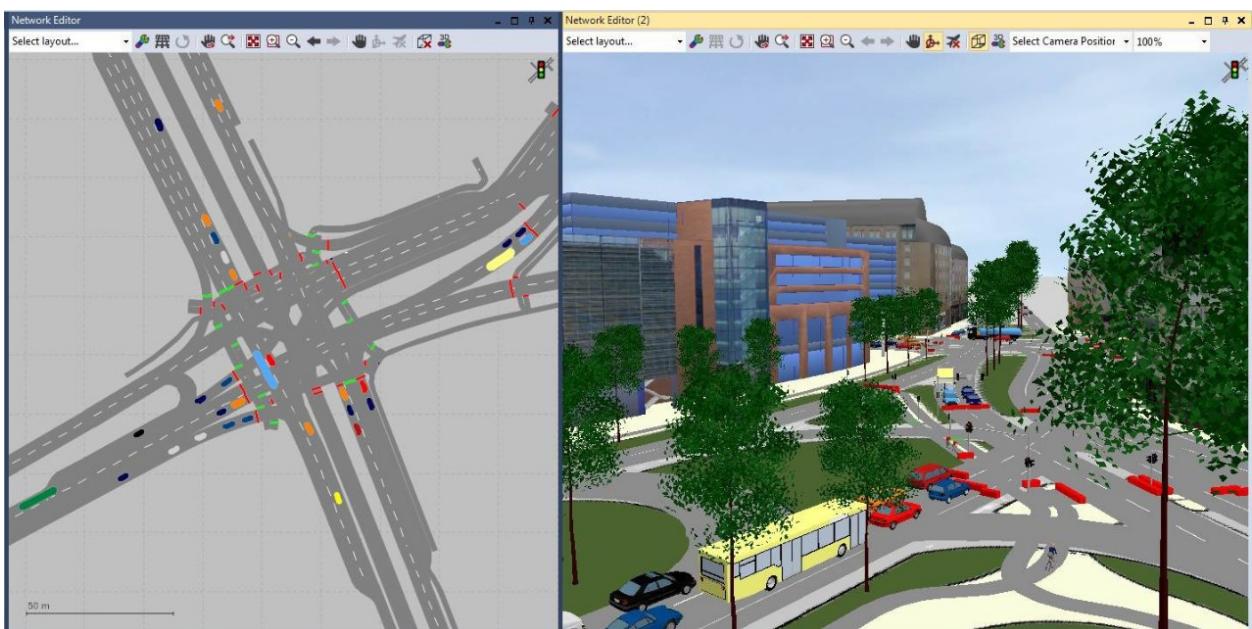


Figura 1.3: Aplicația VISSIM

De asemenea, mai sunt și alte aplicații precum *Cities: Skylines* fiind mai mult un joc cu un singur jucător care îi permite să construiască orașe și să stabilească propria strategie pentru simularea traficului și *Wbots: The autonomous vehicle simulator* care folosește vehicule autonome care se bazează pe inteligență artificială și algoritmi de prelucrare a imaginilor.

Traffic Simulator face parte dintr-un proiect mai mare care se dorește a fi cât mai complex, implementând toate funcționalitățile din aplicațiile prezentate anterior.

Capitolul 2. Proiectarea Aplicației

2.1. Platforma de dezvoltare

Pentru realizarea aplicației s-a folosit o platformă de dezvoltare a jocurilor și anume Unity. În momentul de față, piața platformelor de dezvoltare a jocurilor se bucură de o creștere uriașă, oferind o multitudine de opțiuni. Aceste opțiuni au un suport destul de bun încât oricine are atât o idee cât și suficientă motivație poate să dezvolte o aplicație sau un joc, folosind calculatorul personal. Platformele populare precum Unity, Unreal Engine, Godot Engine, RPG Maker VX Ace, Construct 3 etc, au ajuns într-un stadiu destul de matur încât, pentru crearea unui joc de dificultate mică nu sunt necesare cunoștințe mari legate de programare. De asemenea, trebuie menționat faptul că fiecare platformă, pe lângă documentația de bază, conține câte o comunitate bine dezvoltată, formată din oameni cu diferite experiențe. Astfel, în cazul în care un dezvoltator, indiferent de nivelul său de cunoștințe, se găsește în situația în care se confruntă cu diferite probleme pe care nu le poate rezolva de unul singur, are șanse mari de a găsi aceste probleme deja întâmpinate de către alții.

2.1.1. Opțiuni disponibile

În funcție de scopul final al produsului există posibilitatea alegerii dintre mai multe platforme de dezvoltare, printre care și cele enumerate mai sus. În cazul acestui proiect, s-a dorit de la început dezvoltarea unei aplicații software de modelare a traficului rutier care se va defășura într-un spațiu tridimensional, astfel aria de platforme disponibile s-a restrâns și s-au luat în calcul doar platformele gratuite. În urma analizei platformelor disponibile s-a ajuns la două: Unity și Unreal Engine 4. Unreal Engine este o opțiune foarte bună datorită calității produsului final și a posibilității de a scrie codul în limbajul de programare C++. Întrucât Unreal Engine a fost lansat pe piață mult mai recent de cât Unity, care are deja un avantaj în creșterea comunității și în suportul oferit, pentru dezvoltarea aplicației software s-a ales Unity.

2.1.1.1. Unity

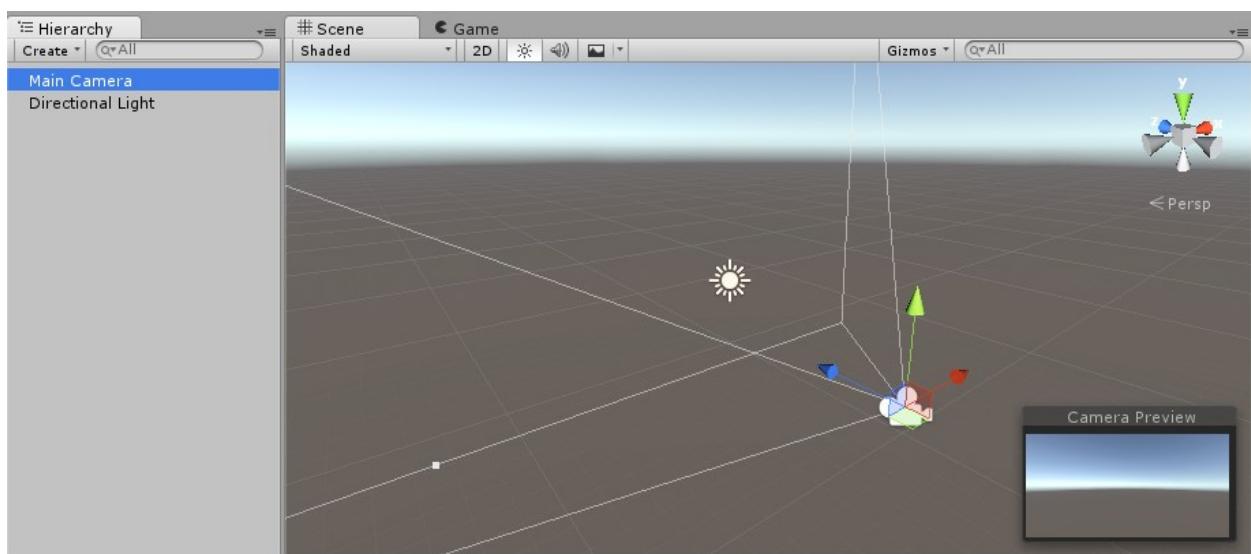


Figura 2.1: Unity – Modul Scene

Unity este un game-engine (un mediu de dezvoltare software conceput pentru dezvoltarea jocurilor video) scris în C și C++, prima versiune (Unity 1.0) fiind lansată în 2005, acum fiind la versiunea 2019.1.7 realizată pe data de 16-Iunie-2019. Unity oferă o gamă largă de beneficii, cum ar fi: posibilitatea de dezvoltare a aplicațiilor atât 2D cât și 3D ce pot fi rulate pe un număr mare de platforme (Windows, Linux, Mac, iOS, Android, Xbox, Play Station etc.). Un avantaj important este acela că portabilitatea nu se limitează doar la mediile în care aplicațiile Unity pot fi rulate, astfel, dezvoltatorii nu sunt forțați să aleagă o anumită platformă în vederea dezvoltării, aceștia având posibilitatea de a utiliza atât Windows cât și Linux sau MacOS.

Unul dintre aspectele utile în procesul de dezvoltare a aplicației de modelare a traficului este posibilitatea de a scrie script-uri pentru orice funcționalitate a acesteia, limbajele cele mai utilizate în Unity fiind Javascript și C#. Pentru realizarea aplicației s-a folosit C# datorită posibilității de a aplica o găndire POO (programare orientată pe obiect). Un alt avantaj important este faptul că majoritatea comunităților dedicate pentru Unity folosesc cele două limbi precizate.

Un alt aspect important care contribuie la dezvoltarea aplicațiilor în Unity este prezența Assets Store-ului. Asset Store este o platformă de descărcare a unor asset-uri create de-a lungul timpului de către membrii comunității. Un asset poate fi un grup de obiecte, texturi, sunete sau scripturi care poate fi importat cu ușurință în aplicația din Unity.



Figura 2.2: Unity Asset Store – Standar Assets

În aplicație a fost folosit Standard Assets pentru adăugarea de copaci și a efectului de lac. Standard Assets reprezintă un grup de obiecte și scripturi care este pus la dispoziție de către Unity tuturor dezvoltatorilor de aplicații care folosesc acest motor de dezvoltare. Obiectele din aplicație au fost create în Blender, apoi importate în Unity.

2.1.1.2. Blender

Blender este un program software gratis de grafică 3D folosit la crearea obiectelor 3D, texturare, creare de animații etc. Blender este disponibil pe mai multe sisteme de operare precum: Windows, GNU/Linux, Mac OS X. Precum și Unity, acest program oferă posibilitatea

de a scrie cod pentru a obține diferite funcționalități ale modelelor create. Limbajul de programare cel mai des folosit este python.

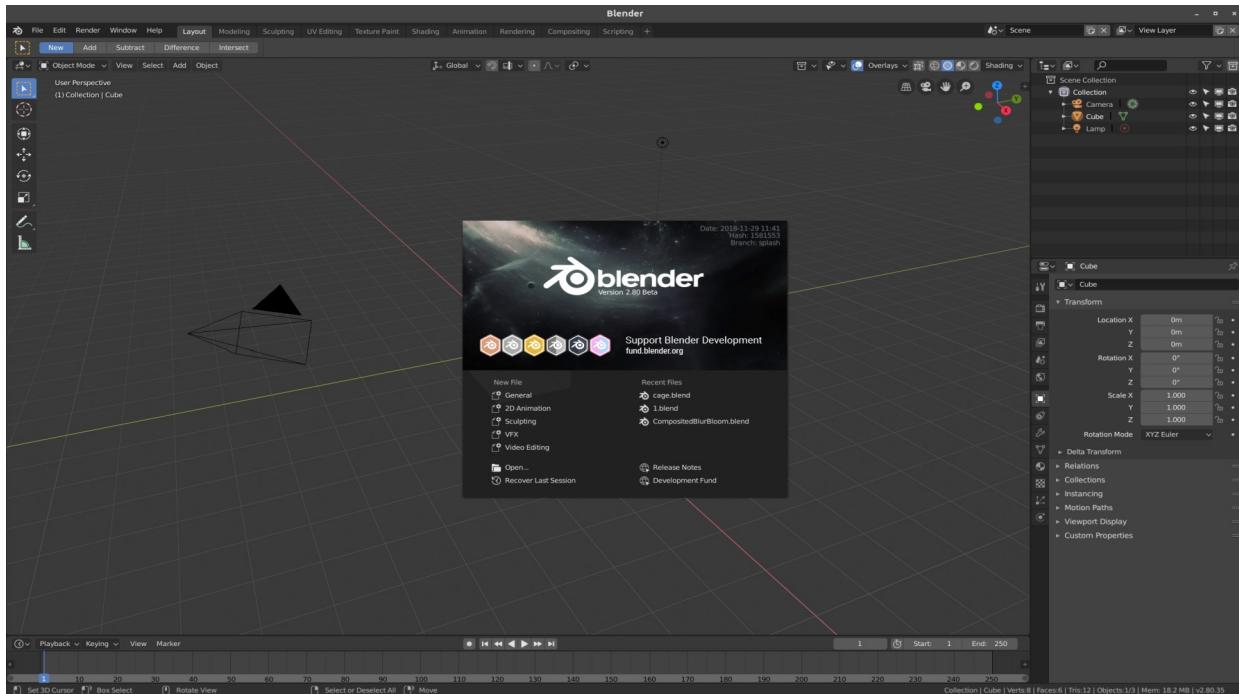


Figura 2.3: Blender

2.2. Tehnologii folosite

Pentru scrierea codului ca limbaj de programare s-a folosit C# și ca mediu de dezvoltare Microsoft Visual Studio.

2.2.1. Limbajul C#

C# este un limbaj de programare dezvoltat de compania Microsoft, inițial făcând parte din proiectul .NET, care ulterior a fost acceptat ca standart de către Ecma (ECMA-334) și ISO (ISO/IEC 23270 2006).

C# este un limbaj de programare orientat-obiect, fiind conceput ca și concurent pentru limbajul Java. Ca și acesta, C# este un derivat al limbajului de programare C. Fiind un limbaj de scop general, implicit oferă mai multe variante de rezolvare pentru o paletă largă de probleme. C# permite construirea de aplicații desktop prin WPF (Windows Presentation Foundation), aplicații mobile pe Windows Phone sau aplicații web prin intermediul ASP .NET.

2.2.2. Platforma .NET

.NET este o platformă dezvoltată de compania Microsoft, reprezentând un mod de interfață cu utilizatorul, care rulează programe independente de sistemul hardware. Cu alte cuvinte, utilizatorul are posibilitatea de a rula același program pe un alt echipament hardware, care rulează un sistem de operare (sau platformă) .NET. Un program destinat platformei dot net rulează atât pe sistemul de operare inițial cât și pe altul, fără recomplilare. Pe lângă C# mai sunt incluse și alte limbi de programare în .NET precum: F#, Visual Basic, C++ etc.

2.3. Obiective

Pentru realizarea scopului principal s-au stabilit următoarele obiective importante:

1. Crearea mediului de simulare
2. Crearea participanților la trafic
3. Crearea semafoarelor
4. Crearea interfeței cu utilizatorul

2.3.1.1. Crearea mediului de simulare

Pentru început se urmărește crearea spațiului rutier care cuprinde căile rutiere (drumurile cu una sau mai multe benzi de mers), nodurile rutiere (intersecțiile și sensurile giratorii) și lucrările speciale (poduri, zone pietonale și trecerile de pietoni), care pot facilita sau restricționa traficul rutier. Acestea sunt create într-un mediu de modelare grafică a obiectelor 3D, ulterior fiind adăugate în mediul cu simularea.

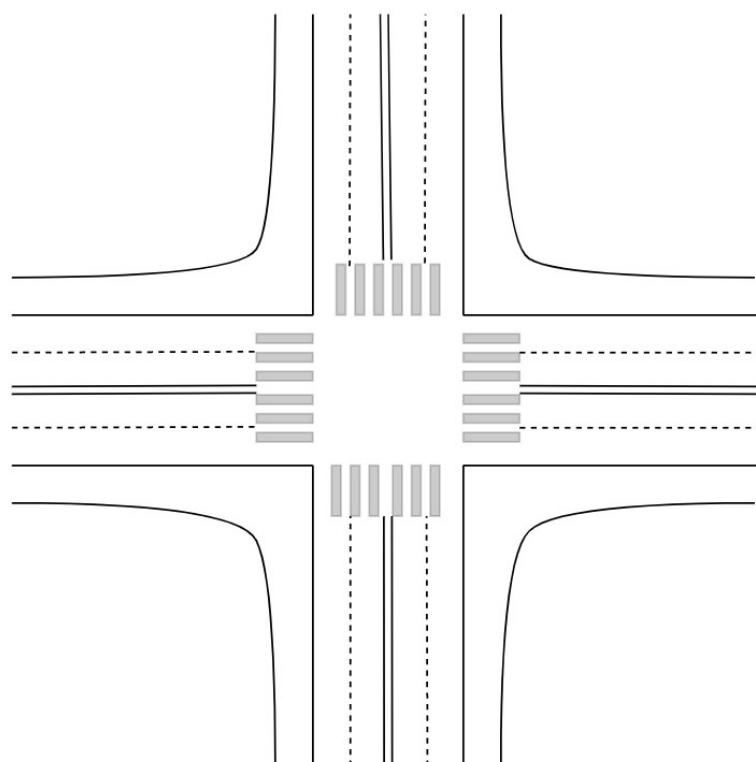


Figura 2.4: Schiță intersecție cu două benzi pe sensul de mers și trecere de pietoni

Pentru a putea stabili căile rutiere pentru fiecare participant la trafic, se folosesc puncte care alcătuiesc rețeaua de căi rutiere. Fiecare punct reprezintă coordonată spațială de tip x,y,z și este conectat cu alte puncte care permit participanților să aleagă o anumită direcție de mers.

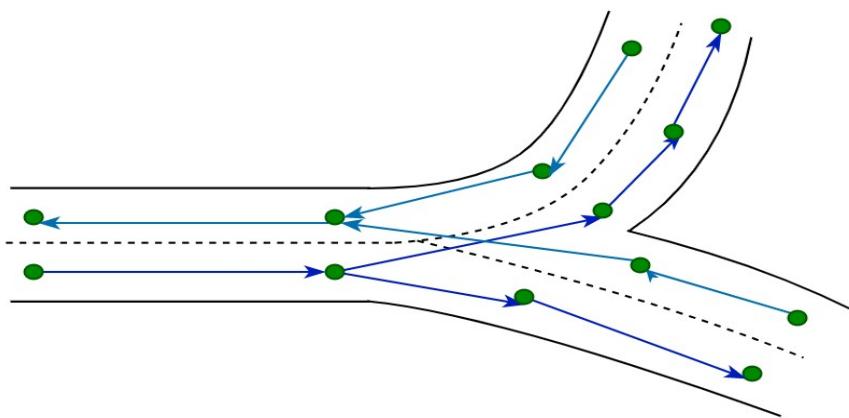


Figura 2.5: Schiță cu punctele care alcătuiesc căile de rețea

2.3.1.2. Crearea participanților la trafic

Participanții la trafic sunt de trei tipuri: mașini reale (Realistic car), mașini simple (Simple car) și mașina controlată (Car Controller). Autovehiculele sunt create într-un mediu de modelare grafică a obiectelor 3D. Utilizatorul poate influența traficul rutier prin schimbarea configurațiilor mașinilor și anume prin: oprirea acestora, schimbarea vitezei, schimbarea priorităților cât și prin conducerea unui autovehicul pentru a observa cât mai bine rezultatele simulării.

Simple car: este un tip de mașină care nu ține cont de coliziuni și de regulile fizice. Mișcarea acesteia se realizează prin mutarea obiectului după o direcție, mai precis, pe fiecare cadru mașina este mutată dintr-un punct în altul cu o anumită viteză, actualizându-se de fiecare dată rotația acesteia.

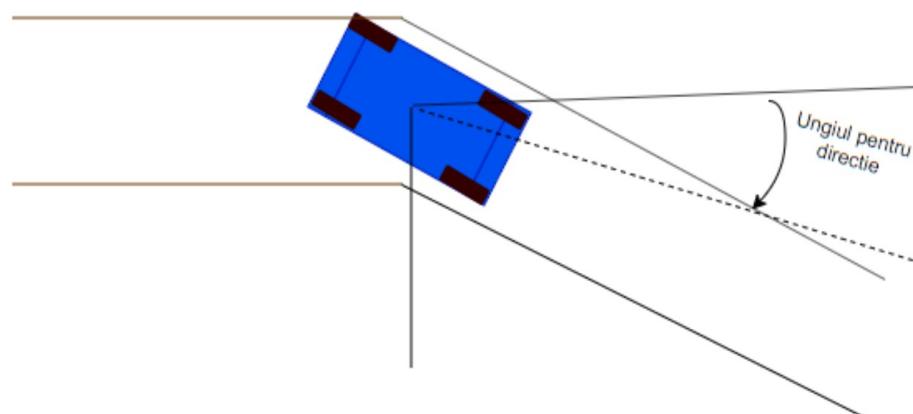


Figura 2.6: Simple Car – Unghiul de rotație

Fiecare mașină este prevăzută cu o logică simplă de circulație, aceasta având senzori activi, poziționați în fața mașinii (stânga, centru, dreapta). Dacă unul dintre senzori detectează un obiect mașina se oprește. În intersecții mașinile care merg înainte au prioritate.

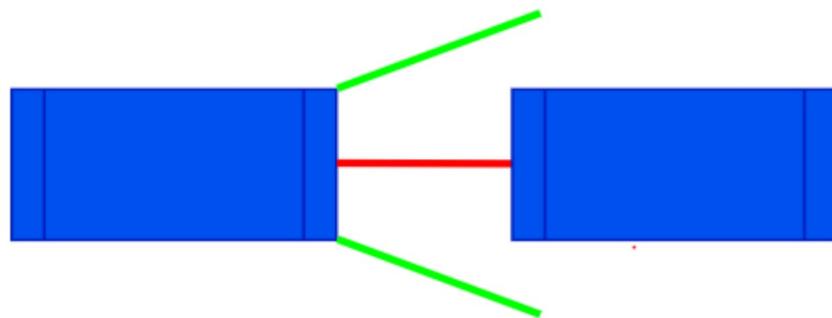


Figura 2.7: Simple Car – Senzori

Realistic car: acest tip de mașină va avea o masă aplicată unui corp rigid, precum și roți. Mișcarea se va realiza prin aplicarea unei forțe de rotații asupra roților. Pentru direcția mașinii se va calcula unghiul dintre roțile din față și punctul în care trebuie să ajungă mașina.

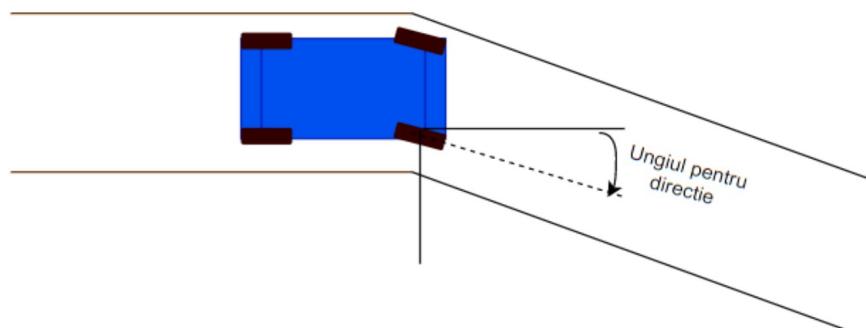


Figura 2.8: Realistic Car – Unghiul pentru direcție

Pentru participarea la trafic, acest tip de mașină este prevăzut cu un număr mai mare de senzori utilizați pentru o precizie mai bună de determinare a poziției acesteia față de ceilalți participanți la trafic.

Spre deosebire de mașinile de tip Simple Car acești senzori au scopul principal de a evita coliziunile cu ceilalți participanți la trafic urmând logica de acordare a priorității. Senzorii sunt poziționați astfel:

- Trei senzori în față, poziționați în continuarea direcției mașinii, paraleli între ei (stânga, centru, dreapta);
- Doi senzori în față, având un unghi de 15° , respectiv -15° față de senzorii anteriori (stânga, dreapta);
- Doi senzori în față pe lateral, la un unghi de 90° , respectiv -90° față de axul mașinii (stânga, dreapta);
- Doi senzori în spate, cu un unghi de 15° , respectiv -15° (stânga, dreapta);

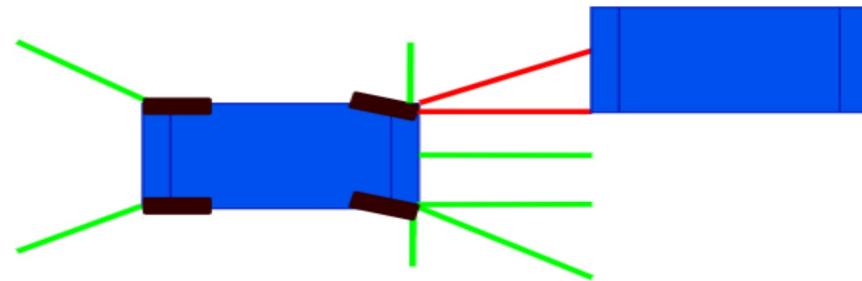


Figura 2.9: Realistic Car – Senzori

Car controller: acest tip de mașină poate fi controlat de către utilizator pentru a putea observa cât mai bine rezultatele simulării. De asemenea, prin controlul unui autovechicul se poate influența traficul după bunul plac. Mașina de acest tip are posibilitatea de a determina ruta minimă în funcție de traficul actual. Ruta este actualizată de fiecare dată când se ajunge într-un nod cu două sau mai multe căi.

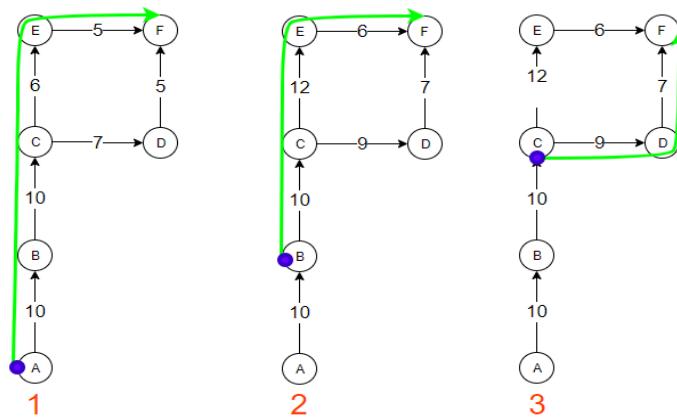


Figura 2.10: Car Controller – Actualizarea rutei optime

Întrucât totalitatea drumurilor reprezintă un graf complet orientat ponderat, pentru a rezolva problema rutei optime se folosesc doi algoritmi. De fiecare dată utilizatorul are posibilitatea de a compara timpii de răspuns ai acestora pentru a stabili care este mai rapid pentru mediul de simulare selectat. Cei doi algoritmi sunt Dijkstra Anexa 5. și A* Anexa 6. .

- *Algoritmul Dijkstra:* este un algoritm pe care îl putem folosi pentru a găsi distanțele cele mai scurte sau costurile minime în funcție de ceea ce este reprezentat într-un grafic. Acest algoritm poate găsi calea cea mai scurtă de la un nod într-un grafic la

fiecare alt nod din aceeași structură de date grafice, cu condiția ca nodurile să fie accesibile din nodul de pornire. Algoritmul va continua să ruleze până când vor fi vizitate toate vârfurile din grafic.

- *Algoritmul A**: este printre cei mai utilizați algoritmi folosiți pentru a rezolva problema rutei optime. Pornind de la un nod specific algoritmul creează un arbore ale cărui ramuri provin de la nodul de început și extinde ramurile într-o singură direcție până când se ajunge la nodul final. Algoritmul evaluează nodurile combinând distanța deja parcursă până la nodul curent cu distanța estimată până la nodul final.

În cazul soluției propuse valoarea numerică a muchiilor este dată de timpul în care o mașină a trecut dintr-un nod în altul. Inițial toate muchiile reprezintă distanța dintre vârfurile corespondente, iar în momentul în care o mașină ajunge într-un nod, valoarea numerică va reprezenta timpul de parcurgere a distanței dintre nodul respectiv și cel precedent, astfel se va genera un drum de cost minim care corespunde cu traficul din acel moment.

2.3.1.3. Crearea semafoarelor

Semafoarele sunt dispozitive de semnalizare, care indică diferite semnale (roșu, galben și verde) referitoare la circulația rutieră, maritimă, feroviară etc. Ca și celelalte obiecte, semafoarele sunt create într-un mediu de modelare grafică a obiectelor 3D. Acestea au următoarele stări: roșu, galben și verde. Trecerea de la o stare la alta se face ca în figura următoare: Fig. 8 Semafor – stări

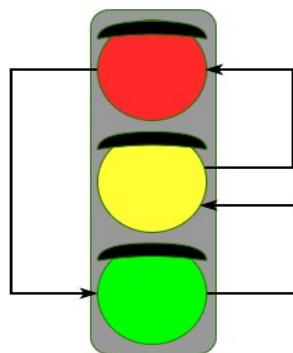


Figura 2.11: Semafor – Stări posibile

Fiecare semnal luminos este activ o anumită perioadă de timp după care se trece la următoarea stare. De asemenea utilizatorul poate modifica timpii de staționare pentru fiecare semnal luminos în parte, precum și stabilirea căruia semnal să fie activ.

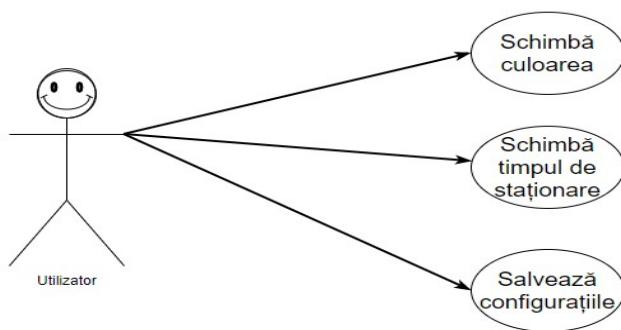


Figura 2.12: Semafor – Interacțiunea cu utilizatorul

Configurațiile semafoarelor sunt salvate într-un fișier sub forma următoare:

Nume Intersecție

- Semafor
 - Semnal Roșu
 - timpul de staționare
 - semnal activ (adevărat/fals)
 - Semnal Galben
 - timpul de staționare
 - semnal activ (adevărat/fals)
 - Semnal Verde
 - timpul de staționare
 - semnal activ (adevărat/fals)

Capitolul 3. Implementarea aplicației

Aplicația este formată din 6 module principale și alte module mai mici precum: generatorul de mașini, modulul cu setări, generatorul de blocuri etc.

Modulele principale:

1. Mediul pentru simulare
2. Căile rutiere
3. Autovehicule
4. Drumul minim
5. Indicatoarele rutiere
6. Camera

3.1. Mediul pentru simulare

Înțelesul său este că se poate simula și modela traficul rutier pentru orice zonă din lume. Pentru crearea acestui mediu s-a ajuns la folosirea unui API (un set de standarde de programare și instrucțiuni) oferit de OpenStreetMap (prescurtat OSM). OSM este un proiect colectiv, gratis, ce are ca scop construirea unei baze de date geografice globale, folosind atât date introduse manual cât și datele colectate de dispozitive GPS.

În România, OpenStreetMap acoperă toată rețeaua de drumuri europene și o parte a drumurilor naționale, ajungând la o acoperire de aproximativ 90% din aceste drumuri. De asemenea OpenStreetMap oferă o serie largă de servicii precum: planificarea de rute, afișarea punctelor de interes, afișarea unor imagini din satelit, hărți ale rețelelor de transport în comun, precum și posibilitatea de a raporta erori la nivelul hărții.

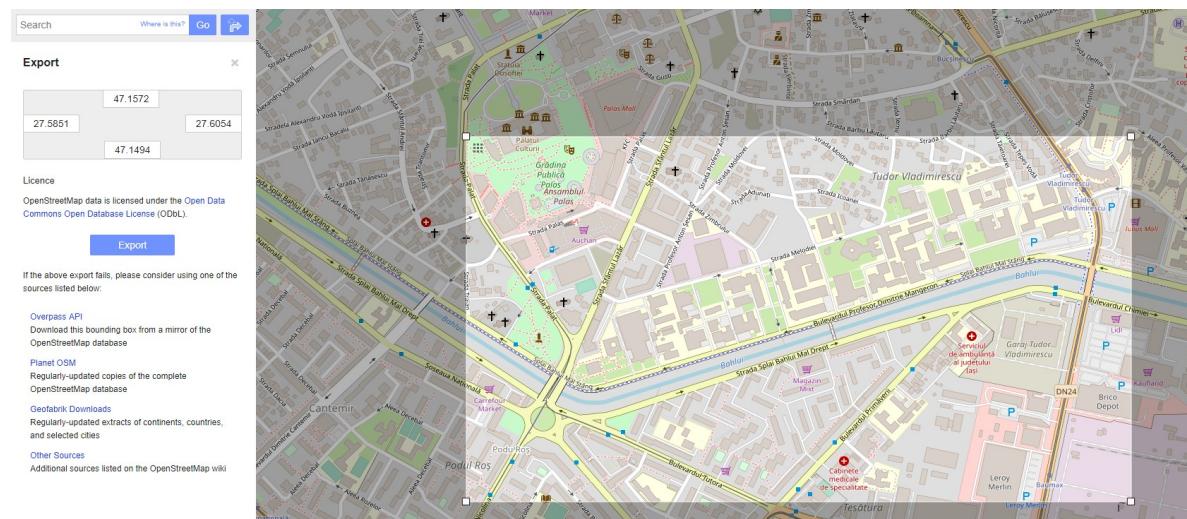


Figura 3.1: OpenStreetMap – Extragere informații

Pentru extragerea informațiilor dintr-o anumită zonă sau un oraș, de pe site-ul openstreetmap.org se selectează zona dorită și se dă export.

După ce se realizează descarcarea datelor, acestea vor fi salvate într-un fișier cu extensia .osm, acesta având structura unui XML. Fiecare fișier conține detaliile despre zona selecționată precum: nodurile care formează un drum, dimensiunea drumului, numărul de benzi, tipul drumului, numele drumului dar și alte informații precum: viteza maximă admisă, indicator

de semnale auto etc.

Un exemplu de fișier osm care conține o intersecție cu patru drumuri este prezentat în Anexa 4.

Întrucât aplicația software de modelare a traficului este una tridimensională, informațiile preluate de la OpenStreetMap nu erau suficiente pentru modelarea unei intersecții care să corespundă realității. Spre exemplu, cu ajutorul fișierului osm se pot prelucra datele astfel încât să se genereze o intersecție, dar aceasta nu este fidelă realității deoarece nu se cunosc dimensiunile acesteia, nu este specificat în ce locuri sunt amplasate semafoarele precum și adăugarea sau eliminarea unei benzi nu este specificată.

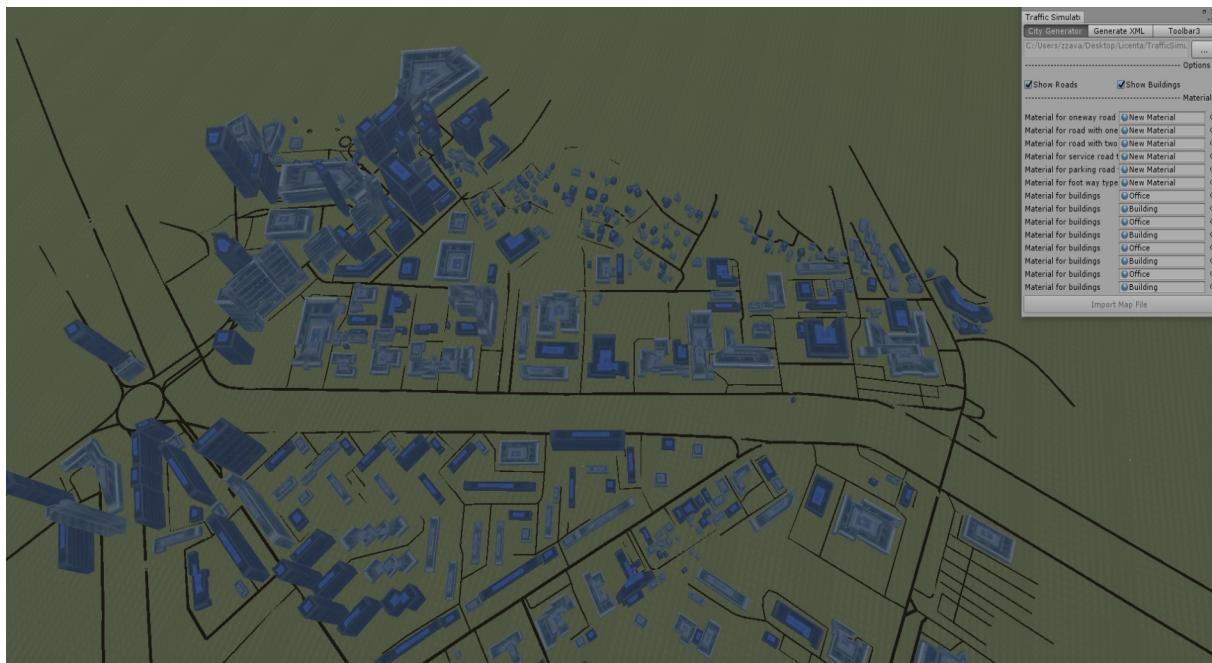


Figura 3.2: Unity – Hartă generată pe baza fișierului osm

O altă problemă importantă care a dus la alegerea altei metode de a crea mediul pentru simulare este dată de faptul că pentru procesarea informațiilor oferite de OpenStreetMap necesita un nivel avansat de cunoștințe în Unity pentru crearea dinamică a obiectelor , aşa că s-a trecut la ideea de a modela mediul dorit folosind Blender 3D.

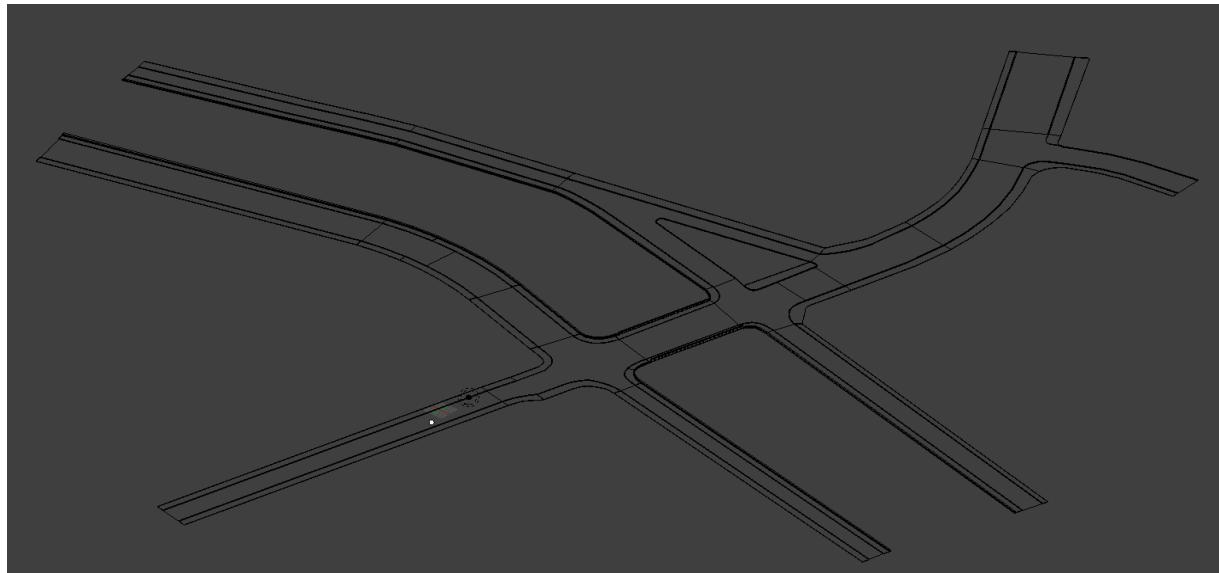


Figura 3.3: Blender – Schiță intersecție

După realizarea schiței în Blender, pentru a da un aspect cât mai real obiectului dorit se adaugă, la final, o textură care a fost creată în Photoshop.

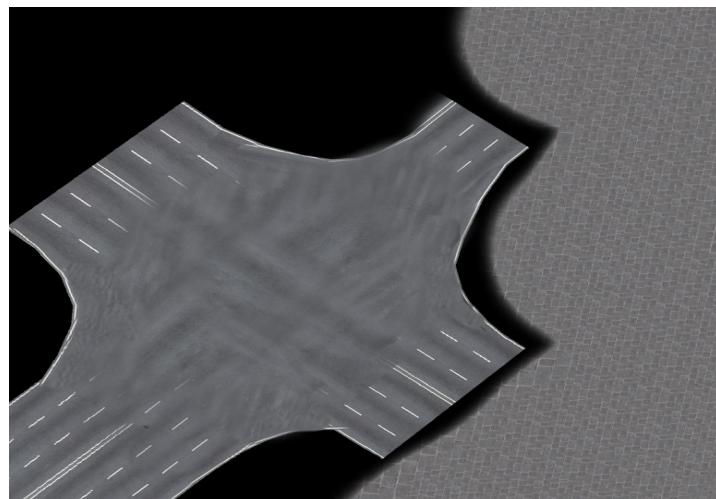


Figura 3.4: Exemplu de textură



Figura 3.5: Unity – Intersecția după adăugarea texturii

3.2. Căile rutiere

Pentru deplasarea autovehiculelor pe o anumită porțiune de drum este necesar ca acestea să cunoască zona în care se află și direcțiile în care se pot deplasa din punctul respectiv. Există două soluții care pot duce la rezolvarea acestei probleme: fie autovehiculul primește un set de imagini și acesta ia decizii în urma prelucrării acestora, fie sunt plasați senzori în diferite puncte ale drumului și trimit informațiile necesare către autovehicul. S-a ales soluția cu plasarea senzorilor deoarece consumă mai puține resurse și implementarea este mai ușoară decât cealaltă soluție.

Un senzor reprezintă, de fapt, un obiect plasat în anumite puncte de pe drum. Spre exemplu, un drum drept care are două benzi va avea 4 senzori: doi senzori pentru prima bandă, unul fiind plasat la început și unul la final și doi senzori pe cealaltă bandă de mers, plasați în același mod ca la primii doi senzori. În aplicația dezvoltată, senzorii poartă denumirea de noduri. În continuare vom vorbi despre nodurile plasate pe drumuri.

3.2.1. Nodurile

Fiecare nod reprezintă un obiect tridimensional de coordonate x,y,z. Pentru a putea să o anumită direcție de mers nodurile sunt conectate între ele, totalitatea acestora formând un graf.

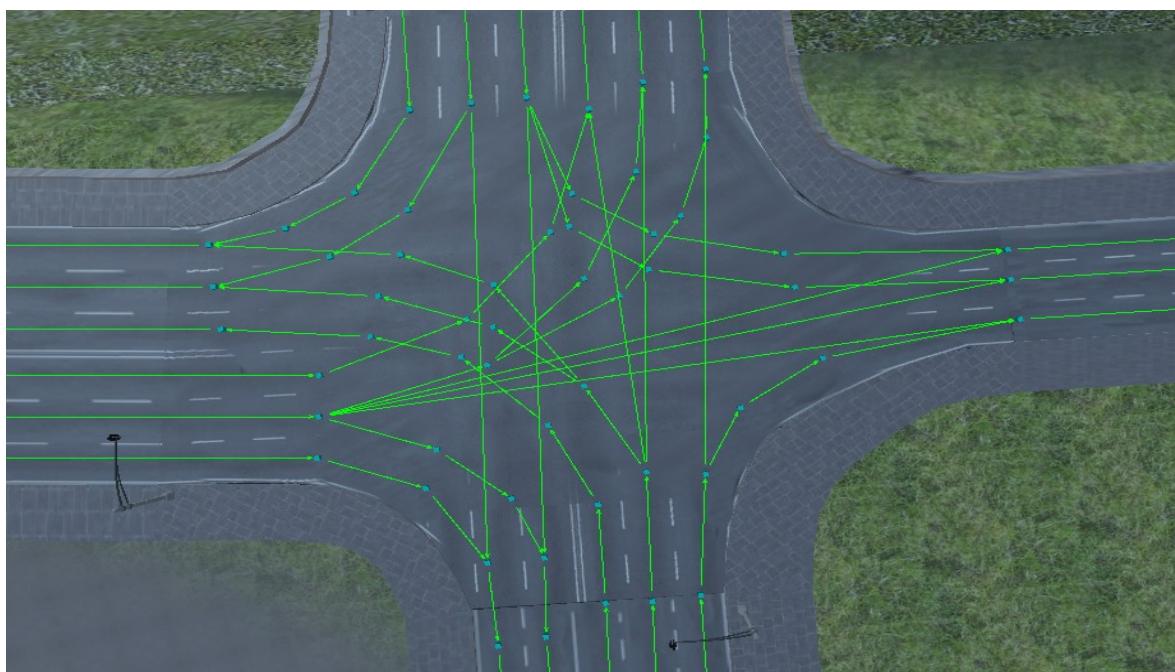


Figura 3.6: Exemplu de noduri folosite într-o intersecție

Structura unui nod:

- Succesori
 - reprezintă nodurile către care se poate deplasa un autovehicul;
 - fiecare nod are o probabilitate de a fi ales;
- Predecesori
 - reprezintă nodurile dinspre care poate veni un autovehicul;
 - fiecare nod are o anumită distanță care reprezintă distanța de la acel nod

la nodul curent;

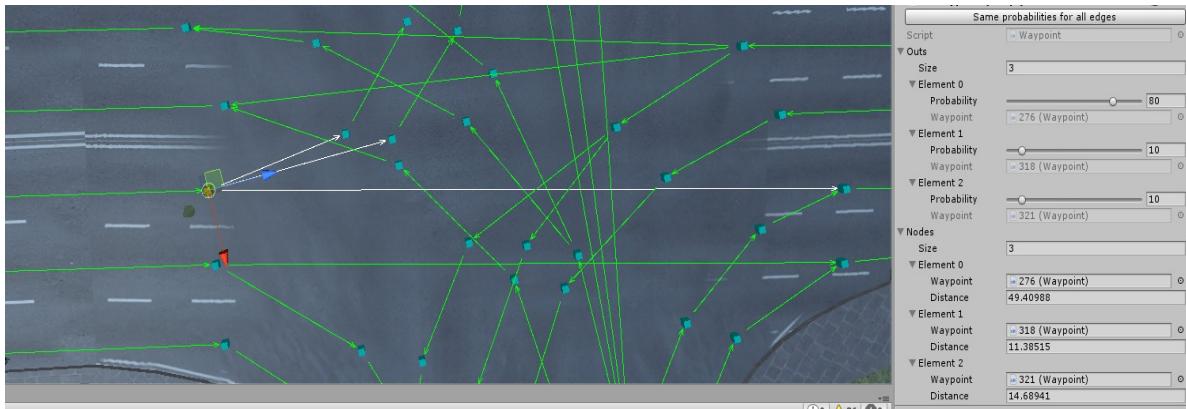


Figura 3.7: Nod cu un predecesor și trei succesi

3.2.1.1. Meniul pentru noduri

Pentru crearea și editarea caracteristicilor fiecărui nod s-a creat un meniu simplist care este asociat unui obiect invizibil, numit *WaypointEditor*. Acest meniu poate fi accesat din Unity doar de către dezvoltatorul aplicației.

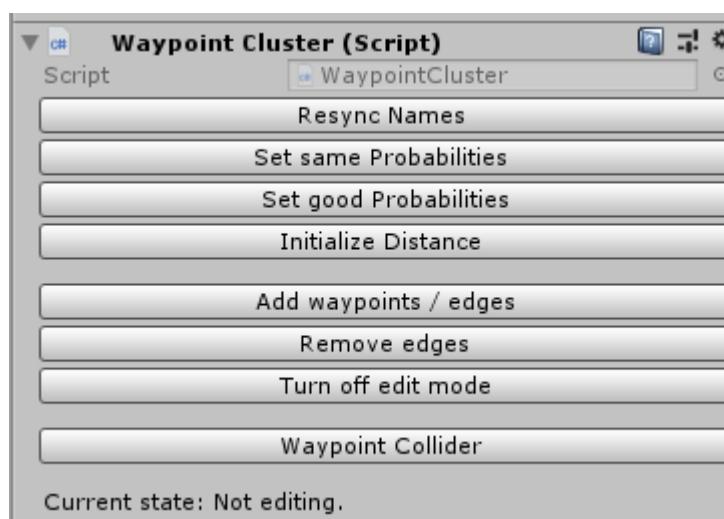


Figura 3.8: Meniu pentru noduri

Pentru crearea unui nod sau pentru crearea unei legături dintre două noduri se apasă pe *Add waypoints/edges* și selectează zona în care se dorește crearea nodului, respectiv se selectează primul nod și următorul care se dorește a fi nodul successor.

```
public Waypoint CreateWaypoint(Vector3 point)
{
    GameObject waypointAux = Resources.Load("Waypoint") as GameObject;
    GameObject waypointInstance = Instantiate(waypointAux) as GameObject;
    waypointInstance.transform.position = point;
```

```

waypointInstance.transform.parent = cluster.transform;
waypointInstance.name = currentID.ToString();
++currentID;
waypoints.Add(waypointInstance.GetComponent<Waypoint>());
waypointInstance.GetComponent<Waypoint>().SetParent(this);

return waypointInstance.GetComponent<Waypoint>();
}

```

```

public void LinkTo(Waypoint waypoint)
{
    if (Outs == null)
        Outs = new List<WaypointPercent>();

    if (waypoint.Ins == null)
        Ins = new List<Waypoint>();

    if (waypoint == this)
    {
        Debug.LogError("A waypoint cannot be linked to itself");
        return;
    }
    for (int i = 0; i < Outs.Count; ++i)
        if (waypoint == Outs[i].Waypoint)
            return;

    if (waypoint.Ins.Contains(this))
        return;

    Outs.Add(new WaypointPercent(waypoint));
    float distance = Vector3.Distance(transform.position, waypoint.transform.position);
    waypoint.Ins.Add(this);
    SetSame();
}

```

De asemenea, meniul prezintă și alte funcții utile precum:

- schimbarea numelor tuturor nodurilor, acestea fiind notate cu cifre de la 0 până la numărul total de noduri existente.

- setarea probabilităților de selectare a nodurilor. Aceste probabilități sunt de două tipuri: fie se aplică aceeași probabilitate tuturor nodurilor, fie se aplică o probabilitate în funcție de poziția acestora.

- inițializarea distanțelor dintre noduri se folosește după ce a fost produsă o schimbare în grupul de noduri, această funcție având rolul de a salva pentru fiecare nod în parte distanța dintre nodul precedent și el însuși.

3.3. Autovehiculele

Autovehiculele reprezintă participantul la trafic. Acestea primesc un nod inițial, urmând ca de fiecare dată să aleagă aleator următorul nod din graf. Selectarea următorului punct se face aleator pe baza probabilității nodului respectiv. Dacă următorul nod din graf este null, adică drumul s-a terminat, atunci mașina va fi distrusă și va fi eliberată zona de memorie.

```

public Waypoint GetNextWaypoint()
{
    int prob = Random.Range(0, 100);
    int sum = 0;
    for (int i = 0; i < Outs.Count; ++i)
    {
        sum += Outs[i].Probability;
        if (prob <= sum)
            return Outs[i].Waypoint;
    }
    SetSame();
    Debug.Log(this.name + " Probabilities has been corrected");
    return Outs[Outs.Count - 1].Waypoint;
}

```

Sunt prezente trei tipuri de autovehicule:

- mașini reale
- mașini simple
- mașina controlată

3.3.1. Mașina reală

Acest tip de autovehicul se deplasează prin aplicarea unei forțe asupra axului roților. De asemenea ține cont atât de gravitație cât și de coliziunile cu ceilalți participanți la trafic. Pentru a adăuga funcționalitățile dorite obiectului mașină, a fost creat un script *CarEngine.cs* care este atribuit mașinilor.

La un anumit timp (numit frame sau cadru) se apelează următoarea funcție care conține funcționalitățile mașinii de tip *mașina reală*.

```

private void FixedUpdate()
{
    if (_needBoost == true)
        speedBooster();
    getAngle();
    if (_distance < 50)
        setDirection();
    applySteer();
    lerpSteerAngle();
    drive();
    braking();
    accelerate();
    ();
    prioritySensors();
    if (ChangeLane && _distance < 30 && !isCoroutineExecuting)
        StartCoroutine(changeLaneLights());
    else
        turnOffSignLights();
    setTrafficDistance();
}

```

Pentru a se deplasa, mașina verifică de fiecare dată dacă trebuie să oprească sau să reducă viteza, ulterior aplicând forță necesară asupra axului roților.

```

private void braking()
{
    if (IsBraking)
    {
        _reduceSpeed = true;
        turnONBackLights();
        GetComponent<Rigidbody>().drag = 2;
        WheelFL.brakeTorque = BrakingTorque;
        WheelFR.brakeTorque = BrakingTorque;
        WheelBL.brakeTorque = BrakingTorque;
        WheelBR.brakeTorque = BrakingTorque;

        WheelFL.motorTorque = 0;
        WheelFR.motorTorque = 0;
    }
    else
    {
        _reduceSpeed = false;
        turnOFFBackLights();

        GetComponent<Rigidbody>().drag = 0;
        WheelFL.brakeTorque = 0;
        WheelFR.brakeTorque = 0;
        WheelBL.brakeTorque = 0;
        WheelBR.brakeTorque = 0;

        WheelFL.motorTorque = CoefAcc + MotorTorque;
        WheelFR.motorTorque = CoefAcc + MotorTorque;
    }
}

```

Atunci când mașina ajunge la o distanță suficientă față de nodul curent, aceasta stabilește care este următorul nod și actualizează distanța parcursă dintre nodul curent și predecesorul său. Distanța este definită ca timpul parcurs pentru a ajunge dintr-un nod în altul.

Pentru a se deplasa către nodul dorit, mașina actualizează la fiecare frame unghiul de rotație al roților.

```

private void applySteer()
{
    _distance = Vector3.Distance(FrontCar.transform.position,
    _currentWayPoint.transform.position);
    // Distance between car and the current point
    if (_distance < 1.5f)
        setWayPoints();

    Vector3 relativeVector =
    FrontCar.transform.InverseTransformPoint(_currentWayPoint.transform.position);

    // Adaptive steer
    float angle = (((WheelsMaxSteerAngle - _maxAngle) / MaxSpeed) * CurrentSpeed) +
    _maxAngle;
    float newSteer = (relativeVector.x / relativeVector.magnitude) * angle;

    TargetSteerAngle = newSteer;
    WheelFL.steerAngle = newSteer;
    WheelFL.steerAngle = newSteer;
}

```

```

private void setWayPoints()
{
    _previousWayPoint = _currentWayPoint;
    _currentWayPoint = _nextWayPoint;
    if (_currentWayPoint.Outs.Count == 0) && (_distance < 1))
    {
        Debug.Log("Destory");
        Destroy();
    }
    else if (_currentWayPoint.Outs.Count != 0)
    {
        _nextWayPoint = _currentWayPoint.GetNextWaypoint();
        if (_nextWayPoint.Outs.Count != 0)
            _nextNextWayPoint = _nextWayPoint.GetNextWaypoint();
    }
}

```

Pentru evitarea coliziunilor mașina este prevăzută cu nouă senzori poziționați astfel:

- Trei senzori în față, poziționați în continuarea direcției mașinii, paraleli între ei (stânga, centru, dreapta);
- Doi senzori în față, având un unghi de 15°, respectiv -15° față de senzorii anteriori (stânga, dreapta);
- Doi senzori în față pe lateral, la un unghi de 90°, respectiv -90° față de axul mașinii(stânga, dreapta);
- Doi senzori în spate, cu un unghi de 15°, respectiv -15° (stânga, dreapta);

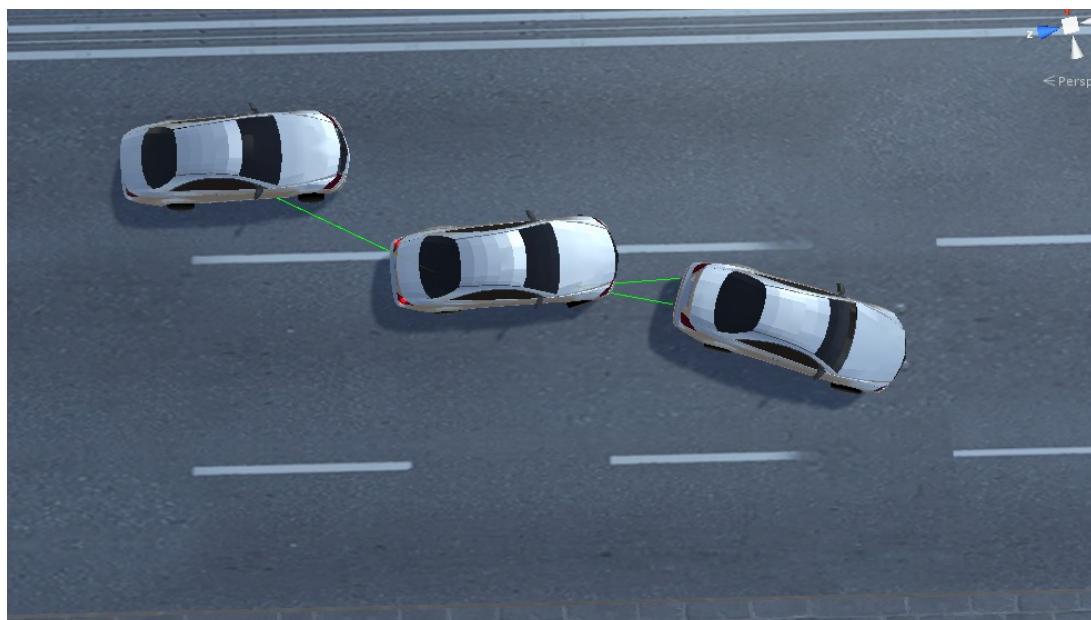


Figura 3.9: Exemplu de senzori activi

La fiecare cadru sunt verificăți toți senzorii și în cazul în care un obiect apare în fața acestora atunci mașina folosește funcția de setare a priorității pentru a afla dacă trebuie să oprească sau dacă are prioritate.

```

private void checkFrontCenterSensor(RaycastHit hit, Vector3 position)
{
    FrontCenterSensorPriority = true;
}

```

```

        Debug.DrawLine(position, hit.point, Color.green);
        CarAhead = hit.collider.gameObject.name;
        if (hit.collider.CompareTag("Player"))
            IsCarAhead = true;
        else if (hit.collider.gameObject.CompareTag("Car"))
        {
            Priority = getPriority(hit.collider.gameObject, hit);
            IsCarAhead = !Priority;
        }
    }
}

```

Spre exemplu, dacă senzorul din față găsește un obiect atunci mașina trebuie să oprească, mai puțin când aceasta se află într-o intersecție și are prioritate.

```

if (FrontCenterSensorPriority)
{
    if (Intersection)
    {
        if (GoLeft && car.IsRed)
            return true;
        if (GoLeft && ((car.GoLeft) && (car.CarAhead == name)))
            return true;
    }
    return false;
}

```

3.3.2. Mașina simplă

Spre deosebire de tipul precedent, autovehiculele de tip *Mașină simplă* nu țin cont de gravitație și nici nu există coliziuni, acestea putând să treacă prin alte obiecte. Pentru a evita trecerea prin alte obiecte, fiecare mașină este prevăzută cu trei senzori poziționați în fața mașinii (stânga, centru, dreapta). Dacă unul dintre senzori detectează un obiect mașina se oprește. În intersecții mașinile care merg înainte au prioritate.

```

if (FrontCenterSensorPriority && Intersection && (GoForward) && (car.CarAhead == name ))
    return true;

```

Pentru deplasarea mașinii nu mai acționează o forță asupra roților care să ducă la acest rezultat, ci, fiind un obiect simplu, mașina se deplasează prin schimbarea coordonatelor poziției la fiecare moment de timp.

```

private void drive()
{
    if (ForcedBraking || IsRed)
    {
        turnONBackLights();
        CurrentSpeed = 0;
    }
    else if (IsCarAhead)
    {
        turnONBackLights();
        CurrentSpeed -= (CurrentSpeed < 4) ? 0 : CurrentSpeed*(1 - MaxSpeed *
Time.deltaTime);
    }
    else
    {

```

```

        if (CurrentSpeed < MaxSpeed)
            CurrentSpeed += MaxSpeed * Time.deltaTime;
        _rb.MovePosition(transform.position + (transform.forward * CurrentSpeed * Time.smoothDeltaTime));
        Vector3 targetPosition = _currentWayPoint.transform.position;
        targetPosition.y = 20;
        var direction = targetPosition - transform.position;
        transform.rotation = Quaternion.Lerp(transform.rotation,
        Quaternion.LookRotation(direction), _turnSpeed * Time.deltaTime);

        turnOFFBackLights();
    }
}

```

Se folosește variabila `Time.deltaTime` pentru a obține o mișcare uniformă. Această variabilă reprezintă diferența dintre frame-ul curent și cel precedent. În general valoarea este ordinul 10^{-2} s, dar depinde foarte mult de performanțele calculatorului pe care este rulată aplicația. Cu cât diferența este mai mică, cu atât programul rulează mai bine.

Ambele tipuri de autovehicule sunt prevăzute cu aceeași logică de stabilire a direcției. Dacă un nod are mai mult de doi succesi, pentru a stabili dacă mașina face stânga, merge înainte sau face dreapta se folosește unghiul dintre nodul curent și succesi.

```

foreach (var waypoint in _currentWayPoint.Outs)
{
    if (waypoint.Waypoint != _nextWayPoint)
    {
        var angle = calculateAngle(waypoint.Waypoint);
        if (DirAngle < angle)
            left++;
        else if (DirAngle > angle)
            right++;
    }
}

```

După ce se stabilesc valorile pentru stânga (left) și dreapta (right), se calculează valorile care indică direcția de mers, acestea fiind publice, având astfel vizibilitatea și pentru ceilalți participanți la trafic.

```

if (left > right)
{
    ChangeLane = true;
    GoLeft = true;
    GoRight = false;
    GoForward = false;
}
else if (right > left)
{
    ChangeLane = true;
    GoRight = true;
    GoLeft = false;
    GoForward = false;
}
else
{
    ChangeLane = false;
}

```

```

        GoForward = true;
        GoLeft = false;
        GoRight = false;
    }
}

```

3.3.3. Mașina controlată

Acest tip de mașină poate fi controlată de către utilizator, având scopul de a observa cât mai bine simularea și de a interacționa cu participanții la trafic în scopul realizării unor evenimente precum crearea intenționată de blocaje în anumite zone și a accidentelor. Aceasta are gravitație și permite coliziunea cu restul obiectelor din simulare. Mișcarea se realizează prin aplicarea unei forțe asupra axului roților care duce la rotația acestora.

Forța aplicată este pozitivă sau negativă, în funcție de comanda dată de utilizator.

```

wheels.wheelFL.motorTorque = maxTorque * Input.GetAxis("Vertical") * gasMultiplier;
wheels.wheelFR.motorTorque = maxTorque * Input.GetAxis("Vertical") * gasMultiplier;
wheels.wheelRL.motorTorque = maxTorque * Input.GetAxis("Vertical") * gasMultiplier;
wheels.wheelRR.motorTorque = maxTorque * Input.GetAxis("Vertical") * gasMultiplier;

```

De asemenea, pentru schimbarea direcției roțile din față se vor învârti la stânga sau la dreapta în funcție de comanda dată de utilizator.

```

wheels.wheelFL.steerAngle = Mathf.Lerp(wheels.wheelFL.steerAngle, maxSteer *
    Input.GetAxis("Horizontal"), Time.deltaTime * 5);
wheels.wheelFR.steerAngle = Mathf.Lerp(wheels.wheelFL.steerAngle, maxSteer *
    Input.GetAxis("Horizontal"), Time.deltaTime * 5);

```

Mașina controlată poate determina și ruta optimă la un anumit punct. Această rută se calculează în timp real în funcție de traficul prezent și se actualizează de fiecare dată când mașina ajunge la o anumită distanță (100 metri) față de un nod care are mai mulți succesi.

```

if (_threadingIsReady)
{
    foreach (var point in _path)
    {
        if ((point.Nodes.Count > 1) && (getDistance(point.transform.position) < 100))
        {
            _pathRequested = true;
            StartPoint = point;
        }
    }
}

```

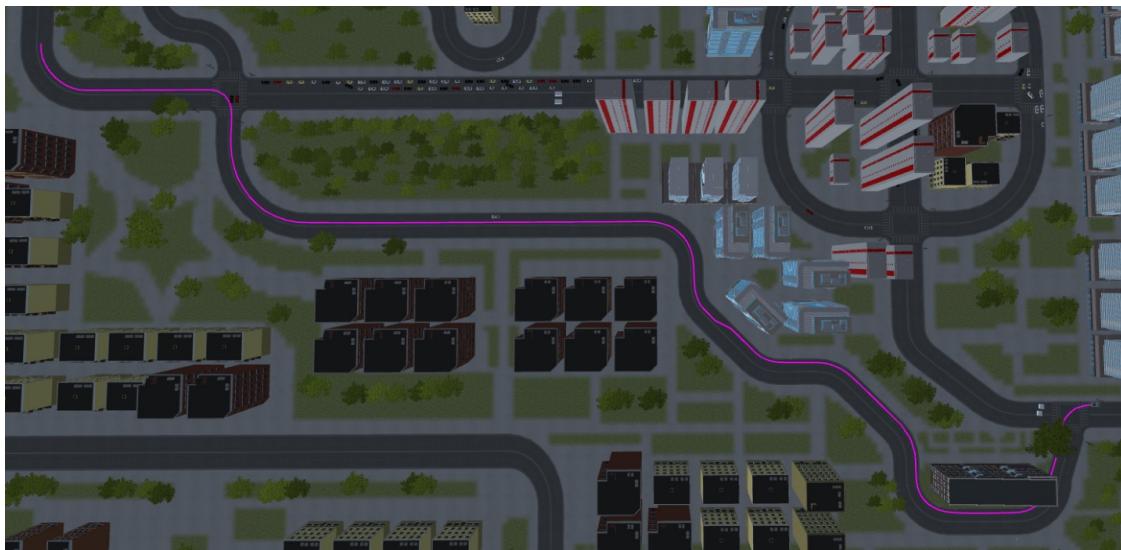


Figura 3.10: Mașina controlată – Drumul optim

3.4. Ruta optimă

Pentru calcularea rutei optime au fost implementați doi algoritmi, Dijkstra și A*, alegându-se cel mai bun în funcție de mediul pentru simulare. Pentru a folosi unul dintre algoritmi a fost creat scriptul PathRequestManager.cs care așteaptă de la utilizator o comandă cu nodurile de start și de final și tipul algoritmului care se dorește a fi folosit.

```
public static void RequestPath(PathRequest request)
{
    ThreadStart threadStart = delegate
    {
        _instance.pathFinding.FindPath(request, _instance.FinishedProcessingPath);
    };
    Thread thread = new Thread(threadStart);
    thread.Start();
}
```

Exemplu de apel al funcției:

```
PathRequestManager.RequestPath(new PathRequest(Type, StartPoint, EndPoint, OnPathFound))
```

3.5. Indicatoarele rutiere

Indicatoarele rutiere prezente în aplicație sunt obiecte care detectează autovehiculul și îi comunică ce indicații rutiere trebuie să respecte. Sunt implementate cinci indicatoare rutiere:

1. Intersecția
2. Curbă periculoasă
3. Cedează trecerea
4. Stop
5. Semafor

3.5.1. Intersecția

Intersecția este un obiect invizibil folosit pentru coliziune, numit collider. Acesta detectează de fiecare dată când o mașină se află în intersecție și îi trimite acesteia mesajul respectiv, prin setarea flagului Intersection pe *True*.

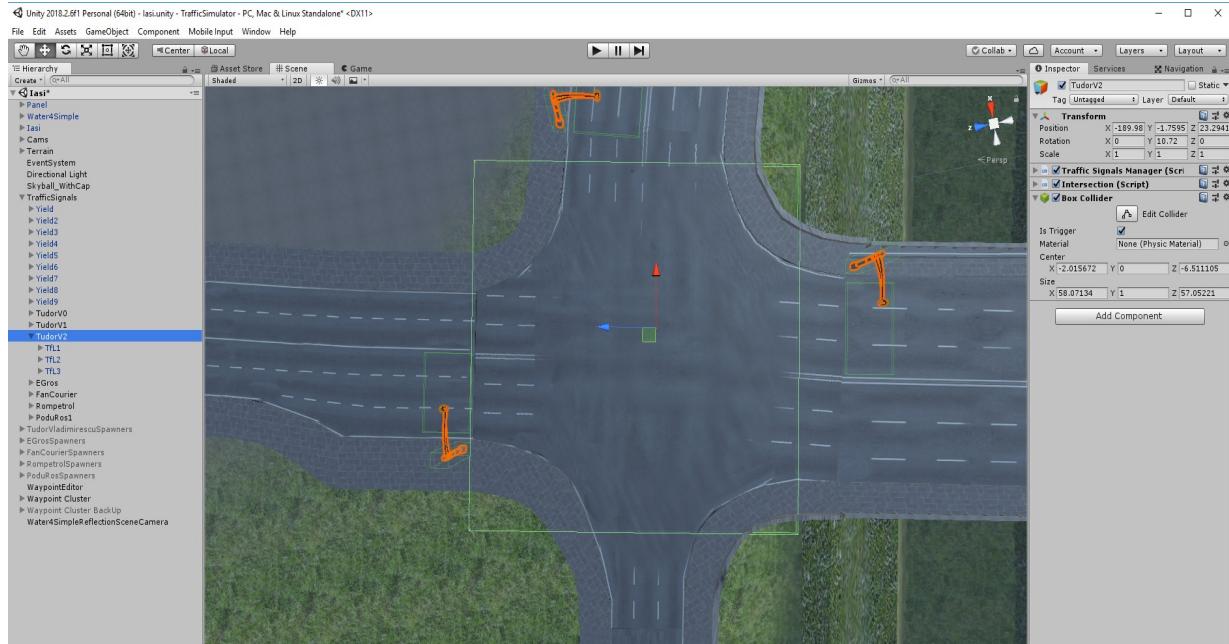


Figura 3.11: Intersecție – Obiectul de coliziune

Funcția care indică autovehiculelor faptul că se află într-o intersecție:

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.tag == "Car")
    {
        _nrOfCars++;
        _nrOfCarsIn++;
        other.GetComponent<CarEngine>().Intersection = true;
        other.GetComponent<SimpleCarEngine>().Intersection = true;
    }
}
```

Funcția care indică autovehiculelor faptul că au ieșit din intersecție:

```
private void OnTriggerExit(Collider other)
{
    if (other.gameObject.tag == "Car")
    {
        _nrOfCarsOut++;
        other.GetComponent<CarEngine>().Intersection = false;
        other.GetComponent<SimpleCarEngine>().Intersection = false;
    }
}
```

3.5.2. Curbă periculoasă

Acest indicator este plasat în curbele periculoase, unde autovehiculele pot ieși de pe stradă. Pentru a putea indica mașinilor ce acțiune trebuie să ia, acesta este prevăzut cu un obiect invizibil care detectează când o mașină se află în față indicatorului.



Figura 3.12: Indicator rutier – Curbă periculoasă

Funcția care indică autovehiculelor faptul că în față se află o curbă periculoasă

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Car") &&
    (other.gameObject.GetComponent<CarEngine>().enabled == true))
    {
        CarEngine car = other.gameObject.GetComponent<CarEngine>();
        car.IstCurveAhead = true;
    }
}
```

Dacă mașina detectează că în față se află curbă periculoasă și are o viteză mai mare decât 30km/h, aceasta va reduce viteza.

```
if (IstCurveAhead && (CurrentSpeed > 30))
    IsBraking = true;
```

3.5.3. Cedează trecerea

Indicatorul *Cedează trecerea* este format din două componente; una care detectează ce mașini se află în față acestuia și o componentă care reprezintă un senzor ce detectează autovehiculele care vin de pe drumul cu prioritate.

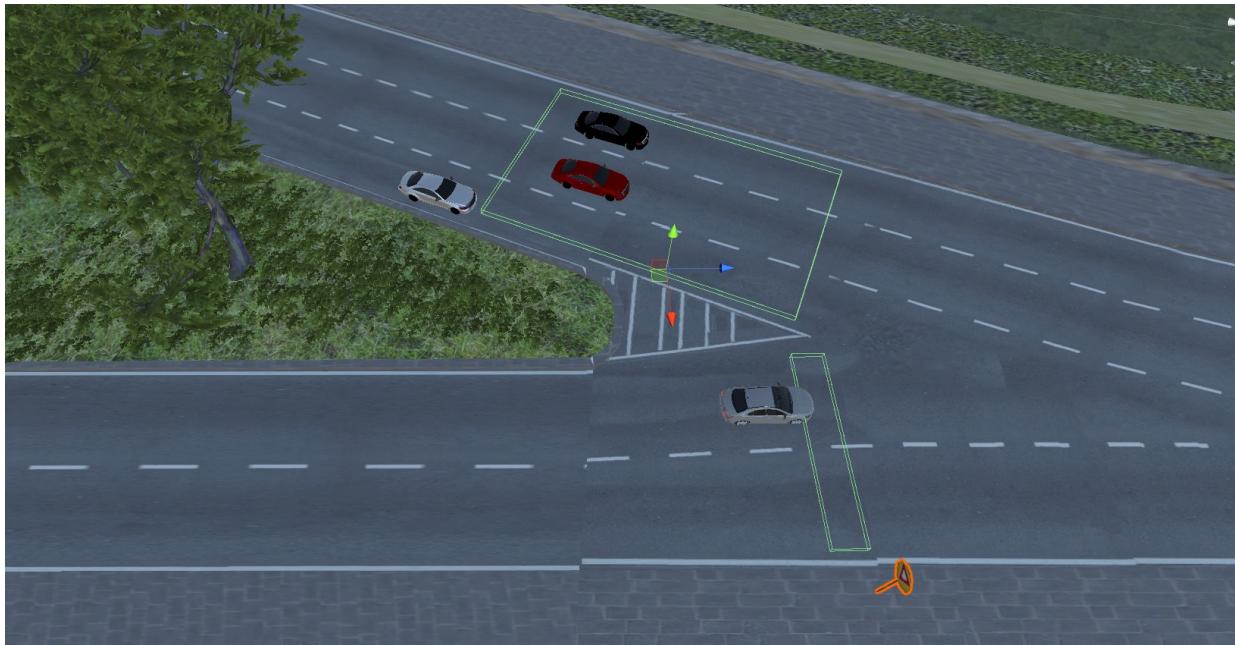


Figura 3.13: Indicator rutier – Cedează trecerea

Funcția care indică autovehiculelor faptul că pe drumul cu prioritate sunt și alte mașini

```
private void OnTriggerEnter(Collider other)
{
    _cars = Detector.GetComponent<CarDetector>().Cars;
    if (other.gameObject.CompareTag("Car"))
    {
        if (other.gameObject.GetComponent<CarEngine>().enabled == true)
        {
            CarEngine car = other.gameObject.GetComponent<CarEngine>();
            if (_cars != 0)
                car.IsRed = true;
            else
                car.IsRed = false;
        }
        else if (other.gameObject.GetComponent<SimpleCarEngine>().enabled == true)
        {
            SimpleCarEngine car = other.gameObject.GetComponent<SimpleCarEngine>();
            if (_cars != 0)
                car.IsRed = true;
            else
                car.IsRed = false;
        }
    }
}
```

Senzorul care detectează mașinile de pe drumul cu prioritate:

```
public class CarDetector : MonoBehaviour {

    public int Cars;
    private void Start()
```

```

{
    Cars = 0;
}

private void OnTriggerEnter(Collider other)
{
    Cars++;
}
private void OnTriggerExit(Collider other)
{
    Cars--;
}
}

```

3.5.4. Stop

Indicatorul stop este format dintr-un singur obiect care detectează autovehiculele din fața indicatorului și le transmite acestora informațiile necesare.



Figura 3.14: Indicator rutier – Stop

Funcție care oprește mașinile timp de două secunde atunci când se află în fața indicatorului Stop:

```

private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Car"))
    {
        if (other.gameObject.GetComponent<CarEngine>().enabled == true)
        {
            CarEngine car = other.gameObject.GetComponent<CarEngine>();
            car.IsRed = true;
            StartCoroutine(StopCar(car, 2));
        }
        else if (other.gameObject.GetComponent<SimpleCarEngine>().enabled == true)
        {
            SimpleCarEngine car = other.gameObject.GetComponent<SimpleCarEngine>();

```

```
        car.IsRed = true;
        StartCoroutine(StopCar(car, 2));
    }
}
```

3.5.5. Semafor

Întrucât aplicația software este pentru modelarea traficului rutier pe baza configurațiilor semafoarelor, obiectele *Semafor* sunt cele mai complexe indicatoare rutiere din aplicație. acest obiect este controlat de către un *TrafficLights Manager* atribuit fiecărei intersecții. Fiecare semafor detectează ce mașină se află la intrarea în intersecție și în funcție de culoarea activă (rosu, galben, verde) îi spune ce indicații rutiere trebuie să respecte.

3.5.6. Functionalitatea semaforului

La fiecare frame se verifică pentru fiecare semafor în parte dacă trebuie să schimbe semnalul luminos al acestuia, ulterior se verifică, care semnal luminos trebuie să fie activ și se pornește timer-ul pentru acesta. Timer-ul păstrează semnalul luminos activ o anumită perioadă de timp, după ce perioada de timp s-a scurt, atunci semaforul va fi actualizat cu următorul semnal luminos.

Partea de cod care verifică pentru fiecare semafor în parte dacă trebuie actualizat semnalul luminos:

```
if (_readyForUpdate)
    for (int i = 0; i < Dim; ++i)
        updateTrafficLights(gameObject.transform.GetChild(i), i);
```

Functia care verifică ce semnal luminos trebuie activat și pornește timer-ul pentru acesta:

```
private void updateTrafficLights(Transform trafficLights, int i)
{
    if (_change[i] == true)
    {
        if (_currentState[i] == TrafficLightState.Red)
            _timer[i] = RedLight(i);

        else if (_currentState[i] == TrafficLightState.Yellow)
            _timer[i] = YellowLight(i);

        else if (_currentState[i] == TrafficLightState.Green)
            _timer[i] = GreenLight(i);

        _currentState[i] = TrafficLightState.None;
        _change[i] = false;
        if (_timer[i] != null)
            StartCoroutine(_timer[i]);
    }
}
```

Trebuie precizat că *StartCoroutine* este o funcție specială pusă la dispoziție de Unity, execuția acesteia poate fi întreruptă în orice moment folosind instrucțiunea de *yield*. Când se utilizează o declarație *yield*, coroutine va întrerupe execuția și o va relua automat la următorul cadru.

Funcția pentru semnalul luminos roșu al semaforului:

```
IEnumerator RedLight(int i)
{
    RemainingTime[i] = RedTime[i];
    Red[i].enabled = true;
    _readyForUpdate = false;

    while (RemainingTime[i] > 0)
    {
        yield return new WaitForSeconds(1);
        RemainingTime[i]--;
        if (_smartIntersection == true) && (_intersectionIsEmpty == false))
            RemainingTime[i] = 1;
    }

    Red[i].enabled = false;
    _change[i] = true;
    _readyForUpdate = true;
    _currentState[i] = TrafficLightState.Green;
}
```

3.5.7. Comunicarea cu autovehiculele

Comunicarea cu autovehiculele este realizată prin intermediul scriptului *TrafficLight.cs*. Ca și indicatoarele prezентate, semafoarele conțin un obiect invizibil care are rolul de a detecta autovehiculele din fața acestora.

Atunci când o mașină este detectată și semnalul luminos al semaforului este de culoare roșie, acesta îi indică mașinii că trebuie să oprească. În caz contrar, dacă semnalul luminos este de culoare verde mașina este informată și poate să își continue deplasarea.

Funcția care indică autovehiculelor ce semnal luminos este activ:

```
private void OnTriggerEnter(Collider other):
{
    if (other.gameObject.CompareTag("Car"))
    {
        if (other.gameObject.GetComponent<CarEngine>().enabled == true)
        {
            CarEngine car = other.gameObject.GetComponent<CarEngine>();
            _listWithRealisticCars.Add(car);
            if (_yellow.enabled == true) || (_red.enabled == true))
                car.IsRed = true;
            else
                car.IsRed = false;
        }
        else if (other.gameObject.GetComponent<SimpleCarEngine>().enabled == true)
        {
            SimpleCarEngine car = other.gameObject.GetComponent<SimpleCarEngine>();
            _listWithSimpleCars.Add(car);
            if (_yellow.enabled == true) || (_red.enabled == true))
                car.IsRed = true;
            else
                car.IsRed = false;
        }
    }
}
```

```
    }
```

3.5.8. Intersecție inteligentă

Intersecțiile sunt prevăzute cu o funcție care calculează numărul de mașini prezente. Această funcție este utilă pentru a bloca semafoarele pe semnalul luminos respectiv, cu timer-ul setat pe valoarea 1, cât timp mai sunt mașini în intersecție, astfel evitând blocajul care se crează atunci când semafoarele își schimbă semnalul luminos în culoarea verde dar încă mai sunt mașini în intersecție.

Funcția care verifică numărul de mașini din intersecție

```
private void Update()
{
    if (_nrOfCarsIn <= _nrOfCarsOut)
    {
        _nrOfCarsIn = _nrOfCarsOut = 0;
        IntersectionIsEmpty = true;
    }
    else
        IntersectionIsEmpty = false;

    trafficFlowCalculator();
}
```

Partea de cod în care semafoarele rămân blocate cât timp mai sunt mașini în intersecție

```
while (RemainingTime[i] > 0)
{
    yield return new WaitForSeconds(1);
    RemainingTime[i]--;
    if (_smartIntersection == true) && (_intersectionIsEmpty == false))
        RemainingTime[i] = 1;
}
```

3.5.9. Salvarea configurațiilor

Pentru fiecare intersecție în parte este posibilă salvarea configurațiilor semafoarelor, acestea putând fi reutilizate într-o simulare viitoare.

```
<?xml version="1.0" encoding="UTF-8"?>
<TrafficManager>
    <TfL0>
        <Red time="23" value="False"/>
        <Yellow time="5" value="True"/>
        <Green time="10" value="False"/>
    </TfL0>
    <TfL1>
        <Red time="23" value="False"/>
        <Yellow time="5" value="True"/>
        <Green time="10" value="False"/>
    </TfL1>
    <TfL2>
```

```

<Red time="23" value="True"/>
<Yellow time="5" value="False"/>
<Green time="10" value="False"/>
</Tfl2>
</TrafficManager>

```

3.6. Camera

Pentru a putea observa cât mai bine comportamentul participanților la trafic au fost create două obiecte de tipul *camera*:

1. *Follow Camera* urmărește obiectul atașat, aceasta aflându-se în spate.

```

public void LookAtTarget()
{
    if (!ObjectToFollow)
        return;
    Vector3 _lookDirection = ObjectToFollow.position - transform.position;
    Quaternion _rot = Quaternion.LookRotation(_lookDirection, Vector3.up);
    transform.rotation = Quaternion.Lerp(transform.rotation, _rot, LookSpeed *
Time.deltaTime);
}

```

2. *Fly Camera* permite utilizatorului să observe fiecare zonă din scenă, acesta putându-i schimba poziția, înălțimea precum și rotația.

3.7. Probleme întâmpinate

1. Crearea mediului pentru simulare

O primă dificultate întâmpinată în dezvoltarea aplicației a fost crearea mediului necesar pentru simulare. Inițial s-a dorit crearea unui oraș pe baza datelor oferite de OpenStreetMap, dar acestea fiind insuficiente precum și nivelul de cunoștințe scăzut pentru modelarea unor obiecte în Unity folosind cod s-a decis folosirea unui software prietenos care este special pentru modelarea obiectelor 3D, anume Blender.

2. Determinarea mașinilor din apropiere

O altă problemă întâmpinată a fost detectarea mașinilor din apropiere în scopul opririi acestora înainte de coliziune sau în scopul determinării priorității. Inițial s-au folosit componentele *Box Collider* din Unity. *Box Collier* este o componentă a unui obiect care determină coliziunile cu un alt obiect care prezintă această componentă. Acestea au fost plasate în față și în spatele mașinilor, dar un număr foarte mare de *Box Collider* duce la o creștere ridicată a resurselor necesare, implicit la blocarea aplicației. Pentru a rezolva problema s-a trecut la ideea de *senzori*, unde un senzor reprezintă componența *RayCast*. *RayCast* este o linie trasată dintr-un punct dat, într-o direcție cu o anumită lungime specificată. Aceasta are proprietatea de a detecta toate obiectele pe care le întâlnește.

3. Crearea unei simulări având un număr mai mare de mașini

Fiecare mașină având 9 senzori (*RayCast*), pentru 100 de mașini funcțiile de Raycast sunt apelate de 900 de ori pe fiecare cadru. Pentru a rezolva această problemă a fost creat un nou tip de mașină și anume *Mașina Simplă* care nu necesită 9 senzori ci doar

3. Acest tip de mașină reduce apelul funcțiilor la 300 pentru 100 de mașini. Pentru a crește performanța și mai mult, apelul funcției *RayCast* de 3 ori pentru fiecare mașină în parte s-a înlocuit cu apelul funcției *RaycastCommand* care permite paralelizarea celor 3 apeluri a funcției *RayCast*, astfel reducând timpul de execuție.

4. Generarea rutei minime fără a bloca aplicația

Generarea rutei minime necesită un cost destul de ridicat, mai ales într-un mediu de simulare în care numărul de noduri este mare (peste 5000). Dacă apelarea unei astfel de funcții se face secvențial, atunci se riscă blocarea firului principal până la returnarea din funcție.

Pentru a rezolvarea acestei probleme s-a creat un *Thread Manager* care pornește de fiecare dată un nou fir de execuție pentru calcularea rutei, astfel evitând blocarea firului principal.

3.8. Funcționarea aplicației

Traffic Simulator este o aplicație software 3D de modelare a traficului care are ca scop fluidizarea traficului rutier prin configurarea semafoarelor.

Pentru crearea unei simulări este necesară alegerea unei hărți, urmată de generarea unui număr de autovehicule stabilit de utilizator. Desigur, este posibilă generarea altora și în timpul simulării. Fiecare autovehicul are propriul „creier” astfel încât să imite cât mai bine realitatea, acesta putând să ia o decizie în funcție de situația în care se află (la semafor, schimbarea de bandă, respectarea semnelor de circulație etc.). După ce autovehiculele au fost generate urmează ca utilizatorul să configureze semafoarele pentru a obține rezultate cât mai bune asupra fluidizării traficului. Aceasta are posibilitatea să salveze configurațiile în fișiere de tip XML, acestea putând fi reutilizate pe viitor.

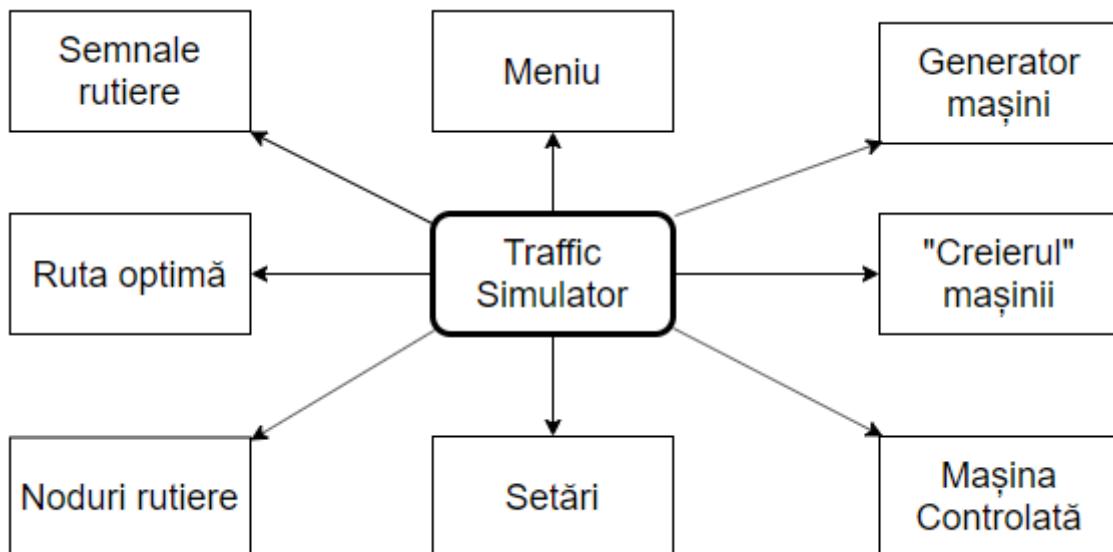


Figura 3.15: Funcționalitățile aplicației

3.9. Interfața cu utilizatorul

Aplicația prezintă mai multe posibilități prin care poate interacționa cu utilizatorul. Aceasta are la dispoziție un meniu principal în care poate începe o nouă simulare, opri temporar simularea, schimba setările aplicației sau închide aplicația, precum și un meniu “Tools” din care poate controla simularea. Trebuie menționat faptul că utilizatorul nu are acces la meniul creat pentru nodurile ce alcătuiesc rutele din mediul de simulare, acesta putând fi accesat doar de către dezvoltatorul aplicației.

3.9.1. Setările aplicației

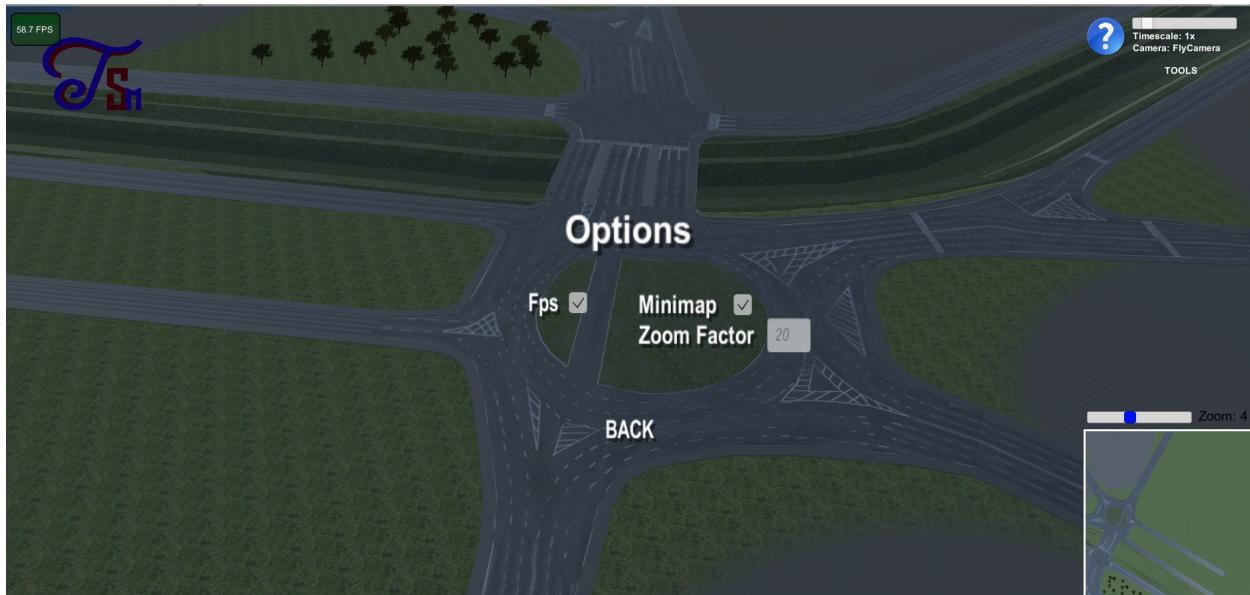


Figura 3.16: Traffic Simulator - Meniul „Options”

- a) *Fereastra FPS*: această fereastră arată numărul de cadre pe secundă. Este utilă pentru a vedea dacă sunt generate prea multe mașini sau dacă aplicația poate fi rulată în parametri optimi pe calculatorul respectiv.
- b) *Fereastra Minimap*: această fereastră este utilă pentru a vedea o anumită zonă din simulare cu camera principală și pentru a avea o viziune asupra intregii simulări.

3.9.2. Meniul „Tools”

- a) Realistic Car
- b) Simple Car
- c) Spawn cars on click
- d) Spawn cars on points
- e) Shortest path
- f) Select object
- g) Car controller

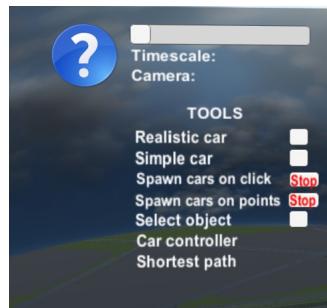


Figura 3.17: Traffic Simulator - Meniul „Tools”

- Realistic Car*: este un buton de comutare. Atunci când acest buton este apăsat mașinile generate vor fi de tipul *Realistic car*.
- Simple Car*: este un buton de comutare. Atunci când acest buton este apăsat mașinile generate vor fi de tipul *Simple car*.
- Spawn cars on click*: în urma apăsării acestui buton utilizatorul va trebui să introducă numărul de mașini pe care vrea să le genereze într-un anumit punct de pe drum și perioada de așteptare pentru generare. Acestea vor fi generate într-un interval cuprins între 2s și valoarea introdusă de utilizator. Zona va fi selectată prin apăsarea butonului stâng al mouse-ului. Mașinile se vor genera în cel mai apropiat punct al grafului față de punctul selectat.

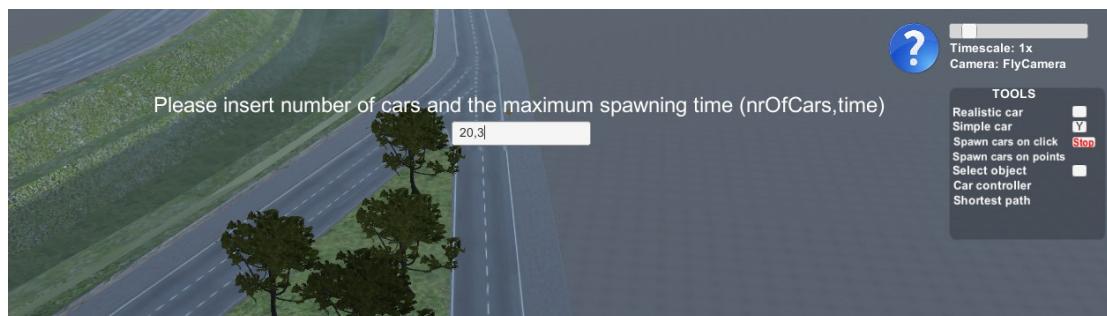


Figura 3.18: Introducere date pentru generarea autovehiculelor



Figura 3.19: Generare mașini după selectarea zonei

- d) *Spawn cars on points*: acest buton este asemănător cu cel prezentat mai sus, diferența fiind dată de faptul că mașinile nu vor mai fi generate într-o zonă selectată de utilizator ci în anumite puncte de pe hartă, din apropierea intersecțiilor.



Figura 3.20: Generare mașini în apropierea intersecțiilor

- e) *Shortest path*: permite utilizatorului să genereze ruta de cost minim. În urma apăsării acestui buton utilizatorul trebuie să selecteze algoritmul dorit (doar A* și Dijkstra sunt implementați) și să apese pe butonul *Start* sau poate alege *Test all* pentru a putea observa care algoritm este mai util pentru mediul de simulare selectat.



Figura 3.21: Shortest Path – utilizatorul folosește butonul *Test All*

Acesta va trebui să selecteze zonele de start și final pentru generarea rutei.



Figura 3.22: Shortest Path – Ruta generată

- f) *Select object*: este un buton de comutare care permite atât vizualizarea cât și editarea informațiilor obiectelor selectate. În urma selectării unui obiect va apărea o fereastră cu informațiile acestuia în stânga ecranului. Obiectele selectate pot fi de două tipuri:

- i. Mașină: atunci când o mașină este selectată apar următoarele informații: senzorul care este activ, mașina care se află în față și vteza mașinii care poate fi schimbată. Pentru fiecare mașină în parte există posibilitatea ștergerii acesteia de către utilizator, adăugării camerei Follow Camera sau setării ca centru pentru Minimap.



Figura 3.23: Meniu – Mașina selectată

- ii. Semafor: atunci când un semafor este selectat va apărea o fereastră cu informațiile despre toate semafoarele din intersecția care conține semaforul respectiv. Utilizatorul poate schimba configurațiile fiecărui semafor și anume: schimbare semnalului luminos activ, schimbarea timpului de staționare pentru fiecare semnal luminos în parte. De asemenea configurațiile făcute pot fi salvate pentru fiecare intersecție în parte, acestea putând fi încărcate în orice altă simulare care conține o intersecție cu aceeași denumire.



Figura 3.24: Meniu – Semafor Selectat

Fiecare intersecție conține butonul de comutare către o intersecție de tip „Smart intersection”. Acest tip de intersecție obligă semafoarele care au activ semnalul luminos roșu să nu-l schimbe atâta timp cât în intersecție încă sunt mașini.

- g) *Car controller*: atunci când acest buton este apăsat utilizatorul trebuie să aleagă în ce punct să fie generată o mașină pe care o poate controla. De asemenea și aceasta este prevăzută cu o fereastră informativă care îi permite ștergerea mașinii, setarea acesteia ca centru pentru minimap, schimbarea tipului de tracțiune, atașarea unei camere de tip Follow Camera și generarea rutei de cost minim până la un anumit punct.



Figura 3.25: Car Controller – Ruta minimă generată

Capitolul 4. Testarea aplicației

În acest capitol sunt prezentate cazurile de testare ale aplicației. După cum am menționat la început, aplicația a fost structurată pe componente, fiecare având o funcționalitate proprie. Așadar, pentru a asigura buna funcționalitate a aplicației este necesară o testare atât a fiecărei componente cât și a întregului sistem. Pentru testare s-a utilizat metoda de testare manuală, în timpul dezvoltării aplicației realizându-se teste pe fiecare componentă și, pe întreg sistemul la finalul procesului de dezvoltare. Fiind o aplicație dezvoltată pentru Windows, a fost creat un executabil care ulterior, a fost testat pe mai multe calculatoare.

4.1. Testarea componentelor

Fiecare componentă a fost testată imediat după realizarea acesteia, ceea ce presupune un avantaj, deoarece nu s-a început implementarea unei alte componente până când cea curentă nu funcționa conform așteptărilor astfel, nu există riscul de a folosi o componentă care oferă rezultate eronate.

4.2. Limitări

Întrucât fiecare autovehicul din aplicație are propriul „creier”, un număr mare al acestora duce la utilizarea procesorului la capacitate maximă. Acest lucru este din cauza funcționalității senzorilor utilizați pentru mașini deoarece atunci când acestea sunt generate într-un număr mare (peste 200), pe fiecare cadru sunt actualizați un de număr senzori cel puțin egal cu numărul total al participanților la trafic. Spre exemplu pentru 200 de mașini sunt activi cel puțin 200 de senzori și cel mult 600 pentru cele de tipul *Simple Car*, respectiv 1800 pentru cele de tipul *Realistic Car*.



Figura 4.1: 10 FPS - Simulare cu peste 200 de mașini

Concluzii

Este cunoscut faptul că întârzierile din trafic provoacă creșterea consumului de combustibil, poluare a aerului, pierderi de timp și de bani, accidente, disconfort și frustrare. Prin micșorarea întârzierilor, condițiile de mers ale șoferilor și situația ecologică sunt îmbunătățite. Din aceste motive, orice progres în această direcție reprezintă un câștig important economic și de mediu. Prin configurarea corectă și sincronizarea semafoarelor din trafic se reduce semnificativ întârzierile și consumul de combustibil. Semnalul sincronizat trebuie să poată reacționa diferit la volume de trafic și viteze diferite pe durata zilei.

Astfel, folosirea unei aplicații software pentru modelarea traficului rutier ar reduce semnificativ accidentele și timpul pierdut în trafic.

Traffic Simulator este o aplicație software de modelare a traficului rutier care poate duce la determinarea unor soluții în vederea optimizării fluidității traficului rutier. Scopul de bază al acesteia este de a observa rezultatele care pot apărea în trafic în urma configurării semafoarelor. De asemenea pentru ca rezultatele să fie cât mai clare, utilizatorul poate genera ruta de cost minim pe baza traficului curent. De asemenea, posibilitatea de a conduce un autovehicul în timpul simulării oferă utilizatorului un mod de interacțiune mai realist.

Această aplicație poate fi folosită și în alte scopuri precum: interacțiunea utilizatorului cu mediul de simulare pentru divertisment sau poate fi folosită ca suport pentru alte aplicații care necesită un simulator de trafic.

Acestea fiind spuse, aplicația prezentată face parte dintr-un proiect mai mare („Smart City”) care are ca scop crearea unui oraș inteligent, printre care se enumera următoarele funcționalități:

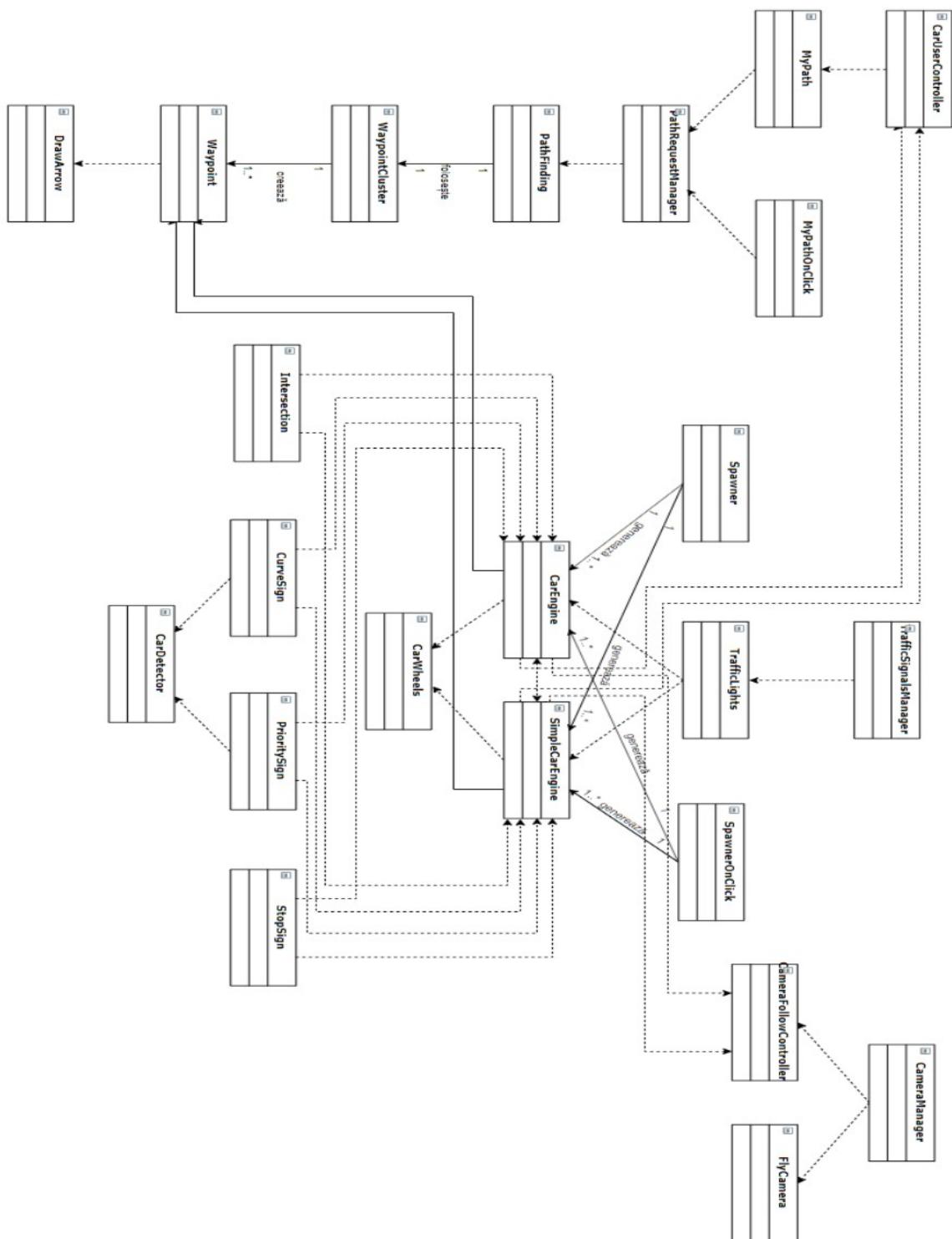
- Metode de optimizare a traficului rutier prin semafoare inteligente;
- Extinderea simulării pe tot Iașiul;
- Sincronizarea intersecțiilor;
- Adăugarea unor noi tipuri de vehicule precum: autobuze, poliție, salvare, pompieri și tramvaie;
- Adăugarea altor participanți la trafic precum: pietoni și bicliști;
- Adăugarea posibilității de a schimba condițiile meteorologice;

Bibliografie

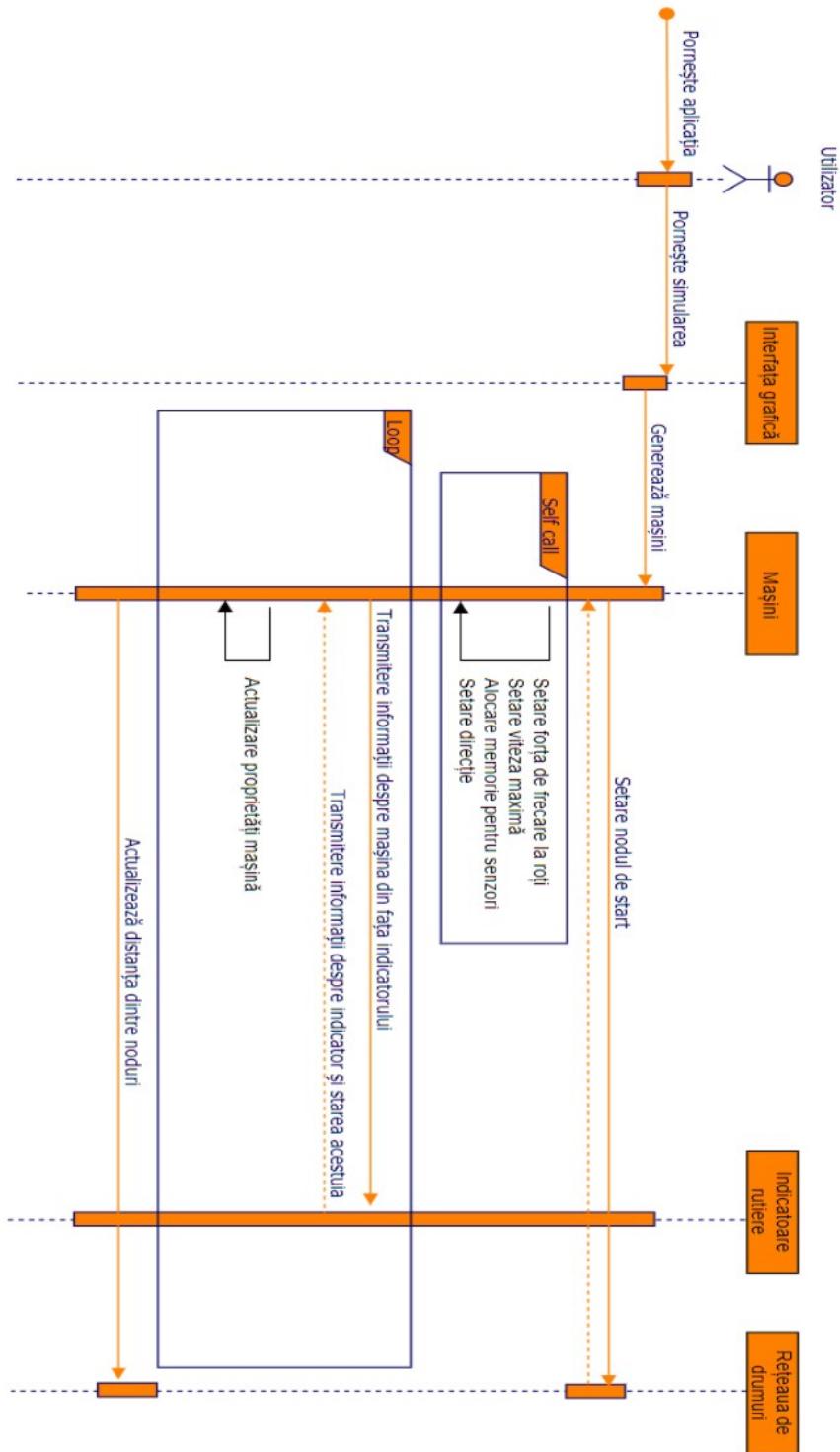
- [1] Documentația oficială, Unity3D, <http://docs.unity3d.com>.
- [2] Documentația oficială, Blender, <https://docs.blender.org/manual/en/latest/index.html>.
- [3] Florin Leon, http://florinleon.byethost24.com/lab_ip.htm, Ingineria programării, laboratoare 1, 2, 11.
- [4] Documentația oficială, OpenStreetMap, <https://ro.wikipedia.org/wiki/OpenStreetMap>.
- [5] Joe Mayo, “C# Succinctly”, 2015.
- [6] Aleksandra Jovic Vranes, Vesna Bjegovic Mikanovic, Jelena Milin Lazovic, Vladimir Kosanovic, „Journal of Transport & Health”, Volum 8, Martie 2018, Paginile 55-62.
- [7] Josh S, Car Sensors used in a Car Engine, 2019, <https://mechanicbase.com/engine/car-sensors>.
- [8] Alex Chumbley, Shortest Path Algorithms, <https://brilliant.org/wiki/shortest-path-algorithms>.
- [9] Vaidehi Joshi, Finding The Shortest Path, 2017, <https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e>.
- [10] Chung-Yang (Ric) Huang, Kwang-Ting (Tim) Cheng, 2009, Electronic Design Automation.
- [11] Stefan Edelkamp, Stefan Schrödl, 2012, Heuristic Search.

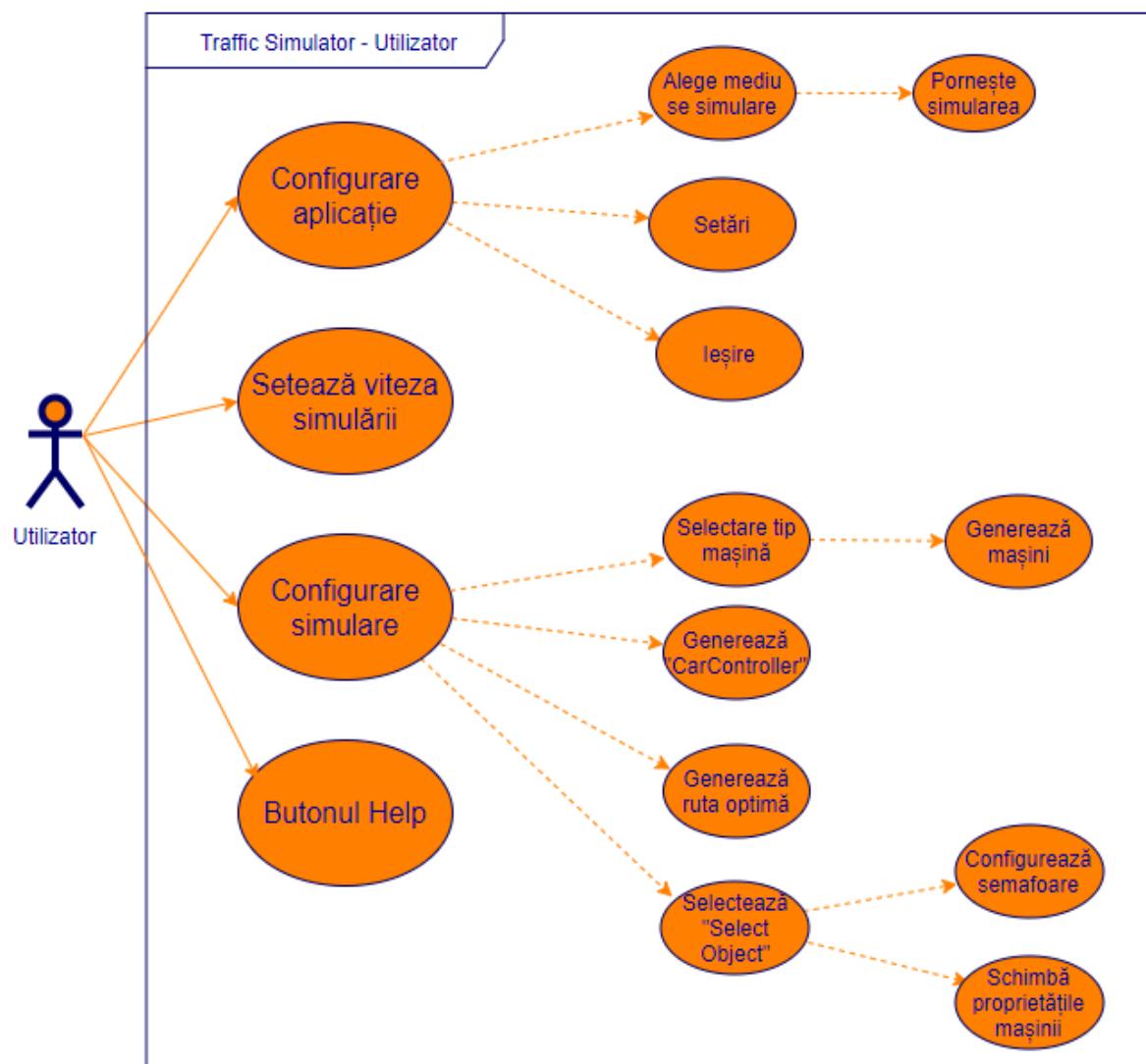
Anexe

Anexa 1. Diagrama UML de clase



Anexa 2. Diagrama UML de secvență a participanților la trafic



Anexa 3. Diagrama UML de utilizare

Anexa 4. Exemplu conținut fișier OSM

```

<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGIImap 0.6.1 (2055 thorn-02.openstreetmap.org)" 
copyright="OpenStreetMap and contributors"
attribution="http://www.openstreetmap.org/copyright"
license="http://opendatacommons.org/licenses/odbl/1-0/"/>
<bounds minlat="47.0948300" minlon="27.7457800" maxlat="47.0952700"
maxlon="27.7464700"/>
<node id="253609452" visible="true" version="1" changeset="381348" timestamp="2008-
03-23T19:46:36Z" user="sorein" uid="25906" lat="47.0945652" lon="27.7483385"/>
<node id="528559475" visible="true" version="2" changeset="3071914" timestamp="2009-
11-09T10:00:22Z" user="Cristian Haulica" uid="126375" lat="47.0950451"
lon="27.7460762"/>
<node id="408005222" visible="true" version="2" changeset="1260693" timestamp="2009-
05-20T10:23:11Z" user="Cristian Haulica" uid="126375" lat="47.0956459"
lon="27.7467041"/>
<node id="253609453" visible="true" version="3" changeset="3071914" timestamp="2009-
11-09T10:00:22Z" user="Cristian Haulica" uid="126375" lat="47.0949033"
lon="27.7455507"/>

<way id="23418301" visible="true" version="20" changeset="40624374" timestamp="2016-
07-10T07:00:30Z" user="LuluMOB" uid="169646">
<nd ref="253609452"/>
<nd ref="528559475"/>
<tag k="highway" v="residential"/>
<tag k="maxspeed" v="50"/>
<tag k="ref" v="DC44"/>
</way>
<way id="42342253" visible="true" version="6" changeset="40624374" timestamp="2016-
07-10T07:00:30Z" user="LuluMOB" uid="169646">
<nd ref="528559475"/>
<nd ref="408005222"/>
<tag k="bicycle" v="yes"/>
<tag k="cycleway" v="no"/>
<tag k="foot" v="yes"/>
<tag k="highway" v="residential"/>
<tag k="maxspeed" v="50"/>
<tag k="oneway" v="no"/>
<tag k="ref" v="DC44"/>
<tag k="sidewalk" v="none"/>
<tag k="smoothness" v="good"/>
<tag k="surface" v="asphalt"/>
</way>
<way id="430142583" visible="true" version="1" changeset="40624374" timestamp="2016-
07-10T07:00:29Z" user="LuluMOB" uid="169646">
<nd ref="528559475"/>
<nd ref="253609453"/>
<tag k="bicycle" v="yes"/>
<tag k="cycleway" v="no"/>
<tag k="foot" v="yes"/>
<tag k="highway" v="residential"/>
<tag k="maxspeed" v="50"/>
<tag k="oneway" v="no"/>
<tag k="ref" v="DC44"/>
<tag k="sidewalk" v="none"/>
<tag k="smoothness" v="good"/>
<tag k="surface" v="asphalt"/>
</way>
</osm>
```

Anexa 5. Algoritmul Dijkstra

```

public static List<Waypoint> Dijkstra(Waypoint start, Waypoint end)
{
    // We don't accept null arguments
    if (start == null || end == null)
        return null;

    // The final path
    List<Waypoint> path = new List<Waypoint>();

    // If the start and end are same node, we can return the start node
    if (start == end)
    {
        path.Add(start);
        return path;
    }

    // The list of unvisited nodes
    List<Waypoint> unvisited = new List<Waypoint>();

    // Previous nodes in optimal path from source
    Dictionary<Waypoint, Waypoint> previous = new Dictionary<Waypoint, Waypoint>();

    // The calculated distances, set all to Infinity at start, except the start Node
    Dictionary<Waypoint, float> distances = new Dictionary<Waypoint, float>();

    for (int i = 0; i < Graph.Count; i++)
    {
        Waypoint node;
        node = Graph[i];
        unvisited.Add(node);

        // Setting the node distance to Infinity
        distances.Add(node, float.MaxValue);
    }

    distances[start] = 0f;

    while (unvisited.Count != 0)
    {
        // Ordering the unvisited list by distance, smallest distance at start and largest
        at end
        unvisited = unvisited.OrderBy(node => distances[node]).ToList();

        // Getting the Node with smallest distance
        Waypoint current = unvisited[0];

        unvisited.Remove(current);

        // When the current node is equal to the end node, then we can break and return
        the path
        if (current == end)
        {
            // Construct the shortest path
            while (previous.ContainsKey(current))
            {
                // Insert the node onto the final result
                path.Insert(0, current);
            }
        }
    }
}

```

```
// Traverse from start to end
    current = previous[current];
}

// Insert the source onto the final result
path.Insert(0, current);
break;
}

// Looping through the Node connections (neighbors) and where the connection
// (neighbor) is available at unvisited list
for (int i = 0; i < current.Nodes.Count; i++)
{
    Waypoint neighbor = current.Nodes[i].Waypoint;
    float length = current.Nodes[i].Distance;

    // The distance from start node to this connection (neighbor) of current node
    float fullLength = distances[current] + length;

    // A shorter path to the connection (neighbor) has been found
    if (fullLength < distances[neighbor])
    {
        distances[neighbor] = fullLength;
        previous[neighbor] = current;
    }
}
return path;
}
```

Anexa 6. Algoritmul A*

```

public static List<Waypoint> AStar(Waypoint start, Waypoint end)
{
    // We don't accept null arguments
    if (start == null || end == null)
        return null;

    // The final path
    List<Waypoint> path = new List<Waypoint>();

    // If the start and end are same node, we can return the start node
    if (start == end)
    {
        path.Add(start);
        return path;
    }

    // the set of nodes to be evaluated
    List<Waypoint> openSet = new List<Waypoint>();

    // The set of nodes already evaluated
    HashSet<Waypoint> closedSet = new HashSet<Waypoint>(); /

    openSet.Add(start);

    while(openSet.Count > 0)
    {
        // Take from the open list the node node_current with the lowest f
        Waypoint currentNode = openSet[0];

        for (int i = 1; i < openSet.Count; ++i)
        {
            if ( (openSet[i].FCost <= currentNode.FCost) &&
                (openSet[i].HCost < currentNode.HCost) )
                currentNode = openSet[i];
        }

        // Move the current node from open set to closed set
        openSet.Remove(currentNode);
        closedSet.Add(currentNode);

        // When the current node is equal to the end node, then we can break and return
        the path
        if (currentNode == end)
        {
            path = CreatePath(start, end);
            return path;
        }

        // Update Gcost and HCost
        foreach (var neighbour in currentNode.Outs)
        {
            if (closedSet.Contains(neighbour.Waypoint))
                continue;
            int index = 0;
            float newCost = 0;
            for (index = 0; index < currentNode.Nodes.Count; ++index)

```

```
        if (currentNode.Nodes[index].Waypoint == currentNode)
            newCost = currentNode.GCost + currentNode.Nodes[index].Distance;

        if ( (newCost < neighbour.Waypoint.GCost) ||
            !openSet.Contains(neighbour.Waypoint) )
        {
            neighbour.Waypoint.GCost = newCost;
            neighbour.Waypoint.HCost = distance(currentNode, end);
            neighbour.Waypoint.ParentNode = currentNode;

            if (!openSet.Contains(neighbour.Waypoint))
                openSet.Add(neighbour.Waypoint);
        }
    }

    return path;
}

public static List<Waypoint> CreatePath(Waypoint startNode, Waypoint endNode)
{
    List<Waypoint> path = new List<Waypoint>();
    Waypoint currentNode = endNode;

    // Add parent nodes of current node to the path
    while (currentNode != startNode)
    {
        path.Add(currentNode);
        currentNode = currentNode.ParentNode;
    }

    // Ordering the path from start node to end node
    path.Reverse();
    return path;
}
```