YRKESHÖGSKOLAN
NOVIA

Visuell simuleringsmiljö för trafik

Philip Miemois

Examensarbete för ingenjörs (YH)-examen

El- och automationsteknik

Vasa 2022

EXAMENSARBETE

Författare:	Philip Miemois
Utbildning och ort:	El- och automationsteknik, Vasa
Inriktning:	Automation
Handledare:	Roger Mäntylä

Titel: Visuell simuleringsmiljö för trafik

Datum: 9.6.2022 Sidantal: 28

Abstrakt

Examensarbetet gick ut på att skapa en simuleringsmiljö för trafik som klarar av att anpassa sig till olika upplägg samt mäta bränsleförbrukning i systemet. Simuleringen har byggts upp i programmet Unity.

Syftet med detta arbete var att skapa en simuleringsmiljö där det ska vara möjligt att utveckla nya lösningar för att förbättra flödet i trafiken, samt konstatera vilka utmaningar som finns i mera krävande trafiksituationer såsom inne i centrum i större städer. För att trafikflödets effektivitet ska vara mätbart har det beräknats ett medeltal av bränsleförbrukningen. Simuleringen krävde ett visuellt användargränssnitt för att ge en bättre förståelse för vilka delar av trafiken som trafiken mindre effektiv.

Arbetet bestod till största del av programmering och av att skapa funktionerande koncept som skapar en grund för hur resten av simuleringen ska fungera. En återkommande utmaning under arbetet var att hitta på metoder och implementera metoderna för att kunna skapa så litet behov av förhandsinställningar som möjligt, men samtidigt också undvika att ta ner på simuleringens dynamiska kapacitet. En stor del av konceptet baserar sig på att göra så få beräkningar som möjligt medan simuleringen kör och i stället göra beräkningarna i början av simuleringen.

Förverkligandet från koncept till fungerande simulering lyckades och de viktiga detaljerna blev implementerade enligt plan. Simuleringen är inte validerad, vilket betyder att det inte finns några bevis på att den motsvarar verkligheten.

Språk: Svenska
Nyckelord: trafik, simulering, Unity

OPINNÄYTETYÖ

Tekijä:	Philip Miemois
Koulutus ja paikkakunta:	Sähkö- ja automaatiotekniikka, Vaasa
Suuntautumisvaihtoehto:	Automaatiotekniikka
Ohjaaja:	Roger Mäntylä

Nimike: Visuaalinen simulointiympäristö liikenteelle

Päivämäärä 9.6.2022 Sivumäärä: 28

Tiivistelmä

Opinnäytetyön tarkoitus oli luoda liikenteelle simulaatioympäristö, joka sekä pystyy sopeutumaan eri järjestelmiin että mittaamaan liikenteen polttoainekulutuksen. Simulaatio on tehty Unity-ohjelmassa.

Tämän työn tarkoituksena oli luoda simulaatioympäristö, jossa on mahdollista kehittää uusia ratkaisuja liikenteen sujuvuuden parantamiseksi ja selvittää, mitä haasteita voi esiintyä vaativammissa liikennetilanteissa, kuten kaupunkien keskustassa isommissa kaupungeissa. Liikennetehokkuuden mittaamiseksi laskettiin polttoaineen kulutuksen keskiarvo. Simulointi vaati visuaalisen käyttöliittymän, jotta saataisiin parempi käsitys siitä, mitkä liikenteen osat aiheuttavat yleisen tehokkuuden heikkenemistä.

Työ koostui pitkälti ohjelmoinnista ja toiminnallisten konseptien luomisesta. Tällä luotiin pohja sille, miten muun simulaation pitäisi toimia. Työn aikana toistuva haaste oli keksiä menetelmiä ja toteuttaa niitä siten, että esiasetettuja arvoja tarvitaan mahdollisimman vähän, mutta samalla vältetään simulaation dynaamisen kapasiteetin pieneneminen. Suuri osa konseptista perustuu siihen, että laskelmia tehdään mahdollisimman vähän simulaation ollessa käynnissä ja sen sijaan tehdään laskelmat simulaation alussa.

Toteutus konseptista toiminnalliseen simulaatioon onnistui ja tärkeät yksityiskohdat toteutettiin suunnitelmien mukaan. Simulaatiota ei ole validoitu, mikä tarkoittaa, että ei ole näyttöä siitä, että se vastaa todellisuutta.

Kieli: ruotsi
Avainsanat: liikenne, simulointi, Unity

BACHELOR'S THESIS

Author:	Philip Miemois
Degree Programme:	Electrical Engineering, Vasa
Specialisation:	Automation
Supervisor:	Roger Mäntylä

Title: Visualized Traffic Simulation

Date 9.6.2022 Number of pages: 28

Abstract

In this thesis, the respondent has chosen to devote his thesis work to creating a simulation environment for traffic that is able to adapt to different layouts and measure fuel consumption in the system. The respondent has made the simulation with the program Unity.

The purpose of this thesis is to create a simulation environment where it is possible to develop new solutions to improve traffic flow, and to establish which challenges may exist in more demanding traffic situations such as in the city center in larger cities. In order to measure how effective the traffic flow is, an average of the fuel consumption is calculated. The simulation required a visual user interface to provide a better understanding of which parts of the traffic are causing a reduction in the overall efficiency.

The work consisted largely of programming and of creating functional concepts to create a basis for how the rest of the simulation should work. A recurring challenge during the work was to invent methods and implement the methods to create as little need for preset values as possible, but at the same time avoid reducing the dynamic capacity of the simulation. A large part of the concept is based on making as few calculations as possible while the simulation is running and instead of doing the calculations at the beginning of the simulation.

The realization from concept to functional simulation was successful and the important details were implemented according to plan. The simulation is not validated, which means that there is no evidence that it corresponds to reality.

Language: Swedish
Key words: Traffic, Simulation, Unity

Innehållsförteckning

1 Inledning

Detta examensarbete handlar om att skapa en simuleringsmiljö för trafik. Simuleringen innehåller element som hjälper användaren att visuellt förstå var trafikflödet är snabbare eller slöare. Simuleringen består av en karta med information som hjälper trafiken att navigera från startdestination till en angiven slutdestination. I denna simulering placeras även trafikljus vid korsningarna. Trafikljusens intervall och instruktioner går att justera genom att ändra på deras startvärden. Trafiken söker sig runt simuleringsmiljön genom att navigera till specifika punkter som definieras slumpmässigt. För att ge trafikflödet ett ämnesriktat flöde används punkter som anger in och ut flöde ur simuleringen. Trafiken mäter användningen av bränsle i realtid per fordon, vilket sedan används för att beräkna medelsförbrukningen i simuleringsmiljön. Simuleringen har skapats i Unity och programmerats i C#.

2 Teori

I detta kapitel tas det upp bland annat om vad programmet, Unity, som är använts. Dessutom har det valts att skriva om trafikteoretiska faktorer för att öka läsarens förståelse varför det valts att göra denna simulering.

2.1 Unity

Unity är en spelmotor som släpptes 2005. Unity används för att skapa både spel och realtids applikationer (Haas, 2014). Programmet använder sig av objektorienterad programmering och majoriteten av koderna skrivs i C#. Spelmotorn klarar av at skapa både 2D- och 3D-applikationer. Unity innehåller även färdigbyggda funktioner som den objektorienterade delen av programmeringen. I detta arbete har funktionerna 2D Sprites, 2D Tilemaps, Collider 2D och Scriptable objects använts (Unity Documentation, 2021). Funktionen 2D Sprites är en funktion som skapar programmerbara objekt av illustrationer. 2D Tilemaps funktionen används som bas för struktureringen av simuleringsmiljön och denna funktion ger möjligheten att använda 2D Sprites som byggblock. Spelmotorn har en flexibel kostnadsstapel vilket gör att den kan användas gratis så länge den inte överskrider en viss kommersiell gräns.

2.2 C#

Programmeringsspråket C# är ett objektorienterat programmeringsspråk som innehåller funktionalitet som hjälper programmeraren att skapa robust och hållbar kod. C# tillåter leverantörer av andra programmeringsapplikationer att integrera sina bibliotek vilket gör att programmeringsspråket kan användas för programmering av flertal olika ändamål. I arbetet finns en del C# termer i arbetet för att beskriva hur en del data behandlats.

2.2.1 Register

Ordet register förekommer i arbetet och syftar på att ett Dictionary har använts för att skapa registret som variablerna sparas i. Ett Dictionary är en metod som används för att kunna söka upp en variabel. Metoden använder sig av två variabler var den ena variabeln fungerar som nyckel och har använts för att returnera den andra variabeln. Båda variablerna matas in i registret samtidigt där nyckelvärdet bör vara unikt. I texten

förekommer det meningar där det har nämnts att register har använts. Det syftas direkt på ett specifikt register i text har klamrar använts för att hjälpa läsaren förstå att det är ett Dictionary. I de fallen är det frågan om att metoden Dictionary har använts för att kunna söka upp relevanta variabler med hjälp av en unik nyckel (C# Dictionary, 2021).

2.2.2 Array

Array är en datatyp som förekommer i detta arbete. Denna typ av data kan tänkas som ett koordinatsystem där en Array med en dimension visas som en linje. Vid varje punkt på linjen kan ett värde placeras. För att sedan söka upp ett specifikt värde används ett nummer för att få ut vilket värde som finns vid den distansen. I texten används en tredimensionell Array som kan representeras ett tredimensionellt koordinatsystem var varje värde är distansen för X, Y och Z axeln. I detta fall ges tre värden till Array:n för att få tillbaka ett värde (C# Arrays, 2021).

2.2.3 Class

Typen Class används för att kombinera olika datatyper för att sedan kunna referera dem till ett gemensamt namn. In i en Class definieras Children som kan vara olika datatyper som hör till den klassen. I detta arbete används Class i kombination med metoden Dictionary för att lagra flera variabler under samma nyckel. Termen Children används i objektorienterad programmering för att förklara att en grupp variabler hör ihop i en gemensam kombination (C# Classes, 2021).

2.3 Trafikstockningar

Faktorer som påverkar att trafikstockningar uppstår är vägarnas kapacitet, antalet fordon som används på grund av brist på allmänna transportmedel, vägtullspunkter, aggressiva accelerationer och retardationer, prioriteringar av olika fordon, dåligt optimerade trafikljus och fordon som söker parkeringsplatser (Xerox). Anslutningstyperna till motorvägar och distansen mellan korsningar har också en stor inverkan på hur ofta trafikstockningar uppstår. Trafikstockningar och flödet av trafiken kan beräknas med ett trafikstockningsindex genom formeln:

$$\text{Trafikstockningsindex} = \frac{\text{Resetid vid rusningstrafik}}{\text{Resetid utan trafik}}$$

Detta index kan användas för att beräkna ifall trafikstockningar uppstår på en viss väg och hur intensiv trafikstockning är. För större system går det att kartlägga vad indexet är för varje väg för att lättare kunna identifiera var problemet ligger (AutoNavi, 2018).

2.4 Trafikregler

Denna teori kommer att behandla trafikregler som gäller i Finland, eftersom dessa trafikregler redan är bekanta. Trafikregler varier från land till land men är oftast väldigt noggrant formulerade. Detta gör att trafiken har varierande funktionssätt beroende på land.

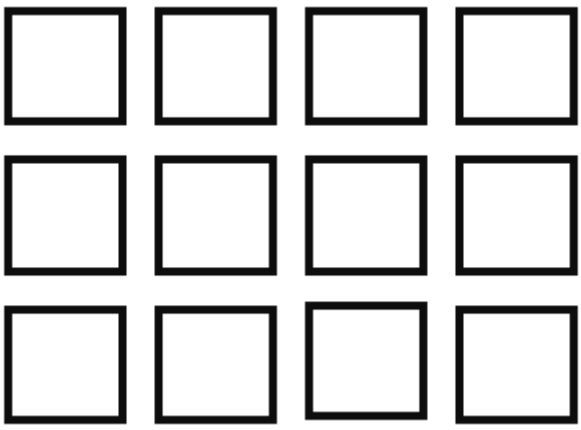
Enligt de finska lagarna bör fordon som svänger, gruppera sig till filen som ligger längst åt det hållet dit den har som intention att svänga. I situationer där fordonet korsar ett annat körfält är det svängande fordonet som måst väja ifall det inte finns skyltar eller ljussignaler som ger det svängande fordonet förkörsrätt. Bestämmelser om säkerhetsavstånd mellan fordon i trafiken gäller endast utanför tätort eller i förhållande till spårvagnar. I korsningar med trafikljus där det är trafikljus separat placerade för varje körfält bör fordonet följa det trafikljus som är ovanför det körfält som fordonet befinner sig i (Kommunikationsministeriet, 2018).

3 Koncept

I denna del beskrives grunden till vilket koncept som simuleringen baserar sig på, samt tas det upp vilka funktioner de olika delarna av systemet har och deras funktioner.

3.1 Basen för simuleringen

Som det syns i *Figur 1*, så består simuleringen av ett rutmönster. Rutmönstret används som bas för simuleringen och används för att planera hur systemet ska fungera från ett trafikperspektiv. Varje ruta räknas som en position och har unika värden. Dessa unika värden innehåller information om hur trafiken får röra sig i systemet och används sedan i trafiken för att möjliggöra navigeringen.

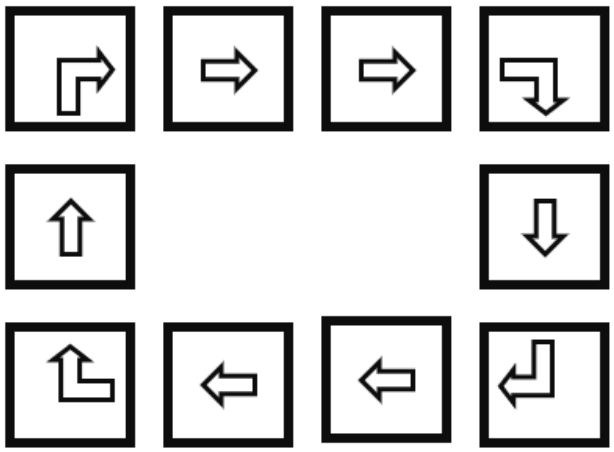


Figur 1. Rutmönster.

3.2 Navigationsval

Navigeringen av trafiken består av instruktioner som sätts in i simuleringsbasen. Beroende på vilken ruta som fordonet befinner sig i kommer fordonet att ha olika möjligheter, den kan välja att svänga endera till höger, vänster eller köra rakt fram. I detta exempel som syns

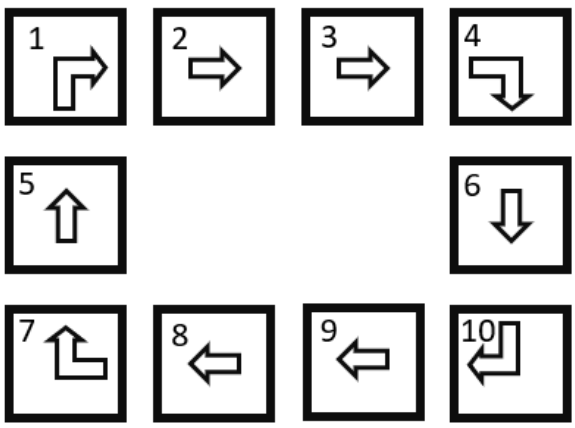
i *Figur 2,* tvingas trafiken att köra runt eftersom den inte har något annat alternativ.



Figur 2. Rutmönster med riktningsrutor.

3.3 Numrering av rutor

Alla rutor i simuleringen som kan användas för navigering. De tilldelas ett nummer för att det ska gå att urskilja dem från varandra, vilket kan ses i *Figur 3*. Denna numrering sker systematiskt med början från vänster och går mot höger. Detta gör att alla rutor har en egen identitet och all data som sedan kopplas till de individuella rutorna kopplas till dessa nummer.



Figur 3. Numrerade rutor i rutmönstret.

3.4 Distanser

För varje navigationsruta beräknas sedan den distans som de har till de andra rutorna. Till exempel när distansen räknas från ruta ett till ruta fem blir distansvärdet nio för rutorna som är i *figur 3*. Detta beror på att distansen inte räknas med skillnaden på koordinater i systemet men i stället som en längd på hur fordonen kommer behöva röra sig fram. I distansräkningen från ruta ett till ruta fem kommer fordonet behöva navigera via rutorna två, tre, fyra, sex, tio, nio, åtta och sju innan fordonet kommer fram till ruta fem. Ett annat exempel är att distansvärdet från ruta ett till ruta ett blir noll eftersom ingen rörelse krävs för att nå ruta ett från ruta ett.

3.5 Trafikljussystem

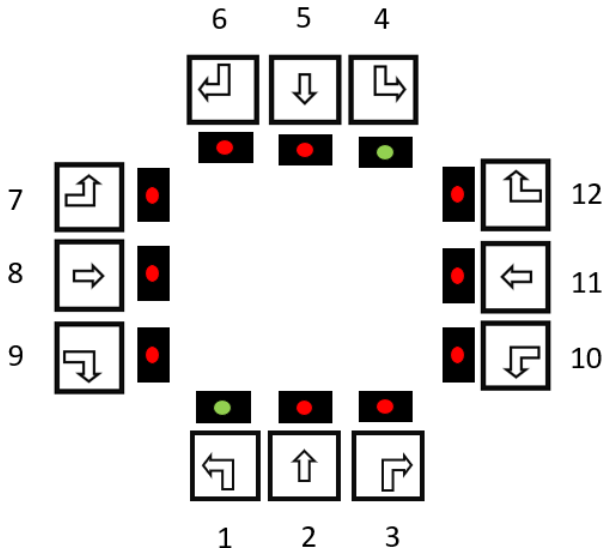
Varje individuellt trafikljus påverkar rutan bakom den så att de kan meddela trafiken deras logisk nivå, hög nivå eller ett betyder att ljuset är grönt och låg nivå eller noll betyder att ljuset är rött. Trafikljussystemet som ett har tio olika funktioner som det kan variera mellan och dessa funktioner beskriver vilka körfält som får använda korsningen. Ordningen av funktionerna och hur länge som de är aktiva går att variera genom att ändra storleken på trafikljussystemets startvariabler. De olika lägen ser till att ingen trafik behöver korsa ett annat aktivt körfält eller ge väjningsplikt åt mötande trafik.

Figur 4 förklarar positionerna av de individuella trafikljusen och *tabell 1* förklarar vilken logisk nivå som varje individuellt trafikljus har i varje funktion. I *tabell 1* betyder siffran ett att trafikljuset visar grönt ljus och siffran noll betyder att trafikljuset visar rött.

Tabell 1. Trafikljussystemets funktioner och ljus nivåer.

	Position 1	Position 2	Position 3	Position 4	Position 5	Position 6	Position 7	Position 8	Position 9	Position 10	Position 11	Position 12
Funk. 0	0	0	0	0	0	0	0	0	0	0	0	0
Funk. 1	0	0	0	1	1	1	0	0	0	0	0	0
Funk. 2	0	0	0	0	0	0	0	0	0	1	1	1

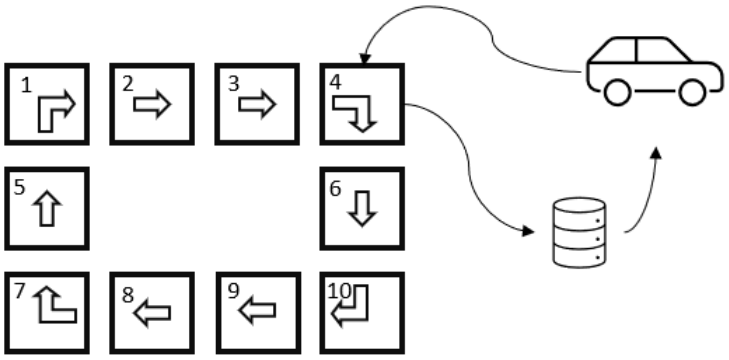
Funk. 3	1	1	1	0	0	0	0	0	0	0	0	0
Funk. 4	0	0	0	0	0	0	1	1	1	0	0	0
Funk. 5	0	1	1	0	1	1	0	0	0	0	0	0
Funk. 6	0	0	0	0	0	0	0	1	1	0	1	1
Funk. 7	0	0	1	0	0	1	0	0	1	0	0	1
Funk. 8	0	0	0	0	0	0	1	0	0	1	0	0
Funk. 9	1	0	0	1	0	0	0	0	0	0	0	0



Figur 4. Trafikljussystemtes positioner där var position fyra och ett har hög nivå vilket motsvarar funktion nio.

3.6 Navigering

Navigeringen är beroende av fordonets position i simuleringen. Fordonet frågar simuleringen vilken ruta den befinner sig på genom att använda sig av sin X, Y position i miljön. Systemet svarar då åt fordonet vilken ruta som motsvarar den positionen. Därefter får fordonet veta vilka alternativ det har och vilka längderna till dess destination är. Fordonet väljer den kortaste rutten till destinationen och efter det kör den till nästa ruta och gör likadant igen. I *Figur 5* har denna process visualiserats.



Figur 5. Informationsflöde för navigeringen.

3.7 Ämnesriktadtrafik

För att undvika att skapa ett system med helt slumpmässig trafik används in- och utgångar i simuleringsmiljön. Dessa punkter placeras i systemet som rutor i rutnätet och agerar som motorvägskopplingar i den simulerade staden. Den ämnesriktade ruttplaneringen väljer slumpmässigt starten eller slutet av navigeringen någonstans i staden. Sedan blir den andra ändan av rutten antingen på in- eller ut punkten. Orsaken till att trafiken rör sig mellan dessa olika punkter är för att skapa en mer trovärdig miljö, men samtidigt försöka hålla den slumpmässig.

Figur 6. De gröna rutorna visualiserar att trafik kan komma från dessa punkter in till staden och de röda rutorna visualiserar att trafik kan komma köra till dessa punkter för att åka ut ur staden.

3.8 Skapande av trafiken

Trafiken som fordonet rör sig i blir skapad beroende på hur mycket trafik som redan finns i simuleringen för tillfället. Den kan således definiera hur mycket trafik som får vara aktiv i simuleringen. Den skapade trafiken tilldelas en startposition och därefter en

slutdestination. Både start- och slutdestinationen följer systemet för ämnesriktad trafik som kan tas del av i kap. 3.7.

3.9 Konsumtionen

Trafikens bränslekonsumtion mäts i g/s för varje fordon. När fordonet nått sin destination rapporteras bränslekonsumtionen samt den sträcka som fordonet rört sig till konsumtionshanteraren. Denna del innehåller en variabel som avgör hur många fordon som den ska räkna medelsförbrukningen för. Om denna siffra är till exempel 50 kommer funktionen att räkna medelsförbrukningen av de senaste 50 fordonen i simuleringen. Medelsförbrukningen anges som gram/ruta.

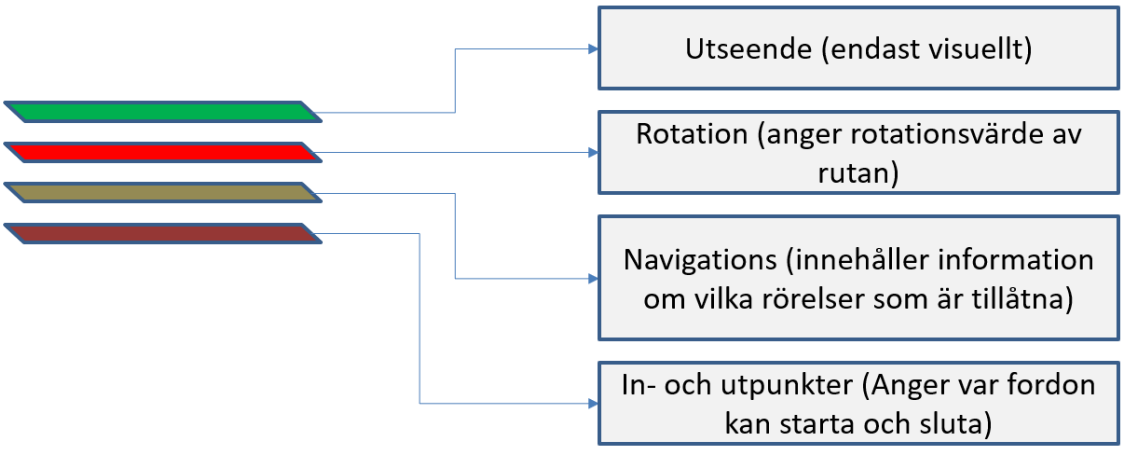
4 Utförande

I detta kapitel beskrivs hur de olika konceptdelarna har blivit implementerade i den verkliga simuleringen och hur de sedan används för att skapa ett enhetligt system.

4.1 Rutsystemet

Detta kapitel kommer att behandla rutsystemets uppbyggnad och vad de olika lagren i systemet har för funktion. Området som *figur 8*, *figur 9* och *figur10* tagits ifrån är samma område i denna konfiguration av simuleringen. Dessa figurer har avsiktligen tagits från samma område för att ge läsaren en möjlighet att förstå hur rutorna har placerats i de olika lagren och hur de placerats i förhållande till rutor i andra lager.

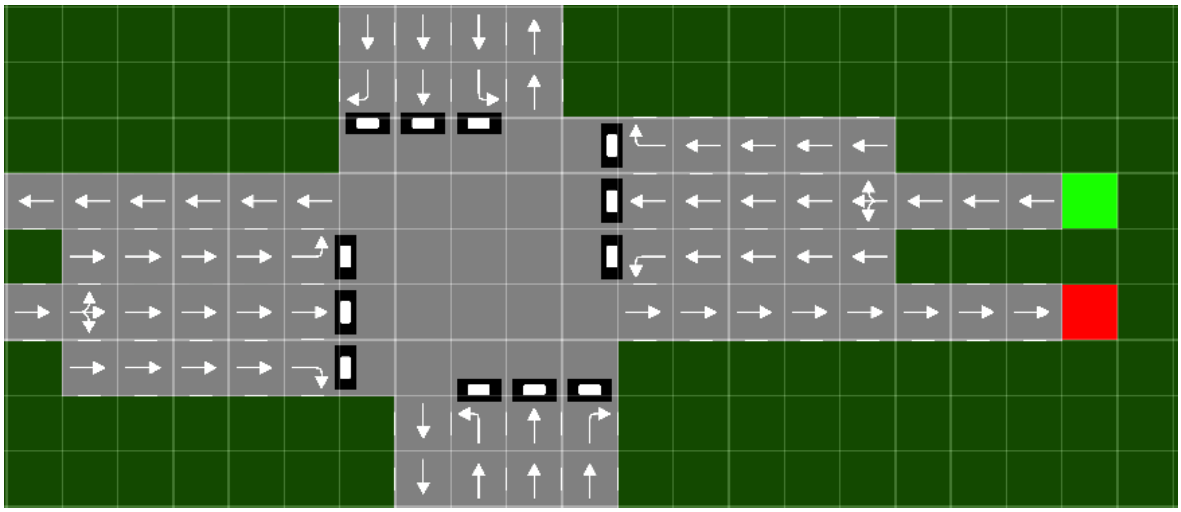
Rutsystemet i simuleringen, som ses i *Figur 1*, använder sig av Unitys egna 2D-objekt som heter ”TileMap -Rectangular”, se kapitel 2.1. Detta rutsystem tillåter användningen av ”sprites” det vill säga visuella element som kan kombineras med kod. Dessa objekt används för att skapa upplägget i simuleringsmiljön. I rutsystemet finns fyra olika system som definierar utseende, rotationen, navigationen samt in- och ut punkterna. De olika lagren placeras ovanpå varandra vilket gör att det översta lagret är det enda lager som syns när simuleringen körs. I *figur 7* visas det i vilken ordning som lagren placeras i rutsystemet.



Figur 7. Rutsystemets lager och funktion.

4.1.1 Utseende

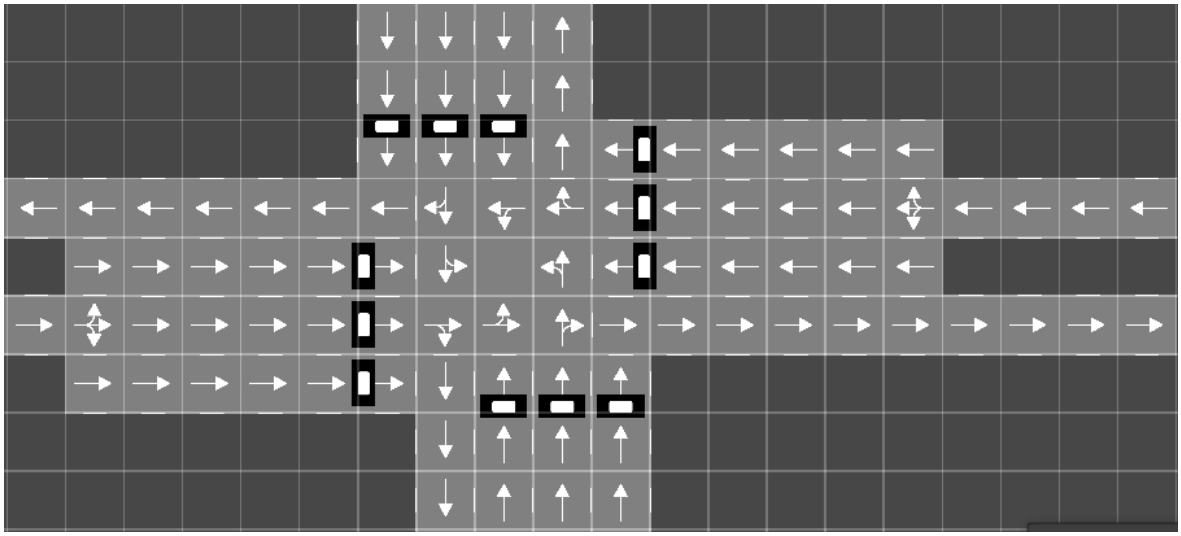
I det översta lagret ritas den slutliga miljön för hur simuleringen ska se ut när den är klar att köras. Det översta lagret har ingen funktion förutom att skapa den slutliga visualiseringen och att fungera som stöd för hur de andra rutorna ska placeras i de övriga lagren. Utseendelagret används som en visuell beskrivning för vad som är möjligt för trafiken men har ingen direkt koppling till simuleringens funktion. Utseendelagret har visualiserats i *figur 8*.



Figur 8. Utseendelagret.

4.1.2 Navigation

Det lager som syns i *Figur 9* styr navigationen och beskriver vilka rörelser som är tillåtna för fordonen. Rutan som tillåter att fordonen får köra rakt är visas som en pil framåt och betyder att rörelsen enbart kan ske rakt fram. Rutan som tillåter vänstersväng för fordonen visas som en pil som svänger till vänster. Denna ruta tillåter fordonen att enbart ändrar riktning mot vänster. Rutan för högersväng visas som en pil som svänger till höger och tillåter fordonen att enbart ändrar sin riktning mot höger. Rutan som beskriver att rörelse kan ske både framåt och till höger är en blandning mellan rutan som betyder rakt fram och rutan som betyder högersväng. Denna ruta tillåter fordonet att välja mellan att köra rakt eller att svänga till höger. Motsvarande möjlighet finns även för framåt och vänstersväng. Rutan som tillåter att fordonet kör framåt, åt höger eller åt vänster är en kombination av alla dessa tre. Denna ruta tillåter att fordonet kör rakt, till höger eller till vänster. Värdet för dessa rutor definieras som variabler där värdet ett tillåter en rörelse och värdet noll förbjuder en rörelse. Navigationslagrets uppbyggnad visas i *figur 9.*

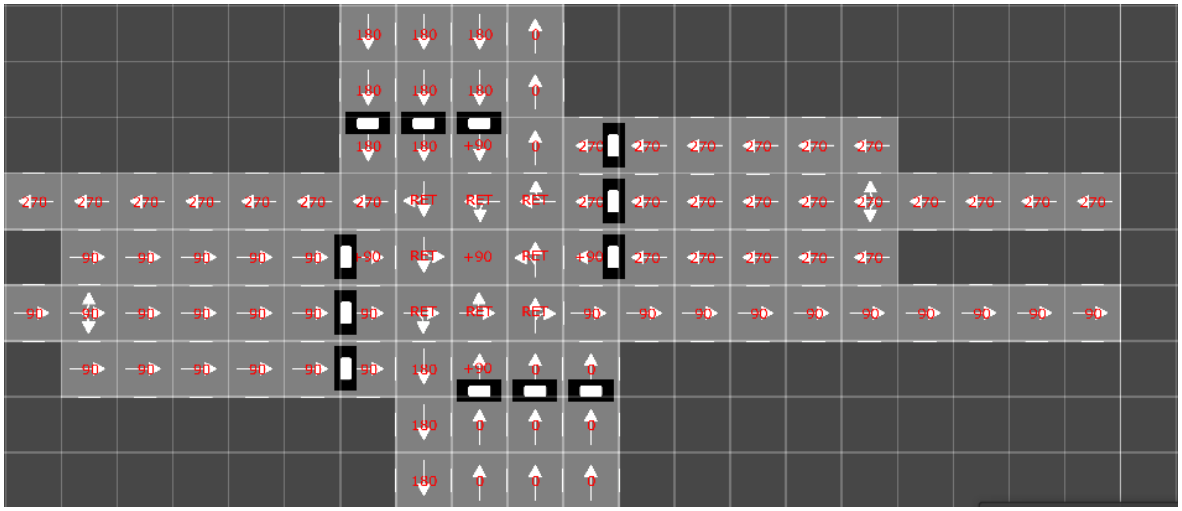


Figur 9. Endast navigationslagret är synligt.

4.1.3 Rotation

Detta lager används för att definiera hur de olika rutorna är roterade i lagret. Alla rutor i rotationslagret innehåller värden som används av simuleringen så att simuleringen förstår hur de olika rutorna har vridits när de placerats. Detta lager kan ha sex olika typer av

rotationer. Rutan med en nolla i mitten betyder att den är neutralt placerad och framåt rutan i navigationslagret pekar uppåt (se kapitel 4.1.2). Rutan med 90 i mitten betyder att riktningen av rutan är vriden 90 grader medsols, detta betyder alltså att framåt i navigationslagret pekar mot höger. Rutan med 180 i mitten betyder att riktningen av rutan är vriden 180 grader medsols, det vill säga att det betyder att framåt i navigationslagret pekar neråt. Rutan med 270 i mitten betyder att riktningen av rutan är vriden 270 grader medsols, det betyder att framåt i navigationslagret pekar mot vänster. Rutan med +90 i mitten används för att definiera att det sker en vänstersväng i en korsning. Denna ruta används i simuleringen för att göra en diagonal körning över trafikkorsningen. Rotationen RET används för att definiera att fordonet ska behålla samma navigeringsprincip och rotation som den tidigare haft, denna funktion används för att kunna navigera fordonet rakt genom korsningar. Rotationslagrets uppbyggnad visas i *figur 11*.

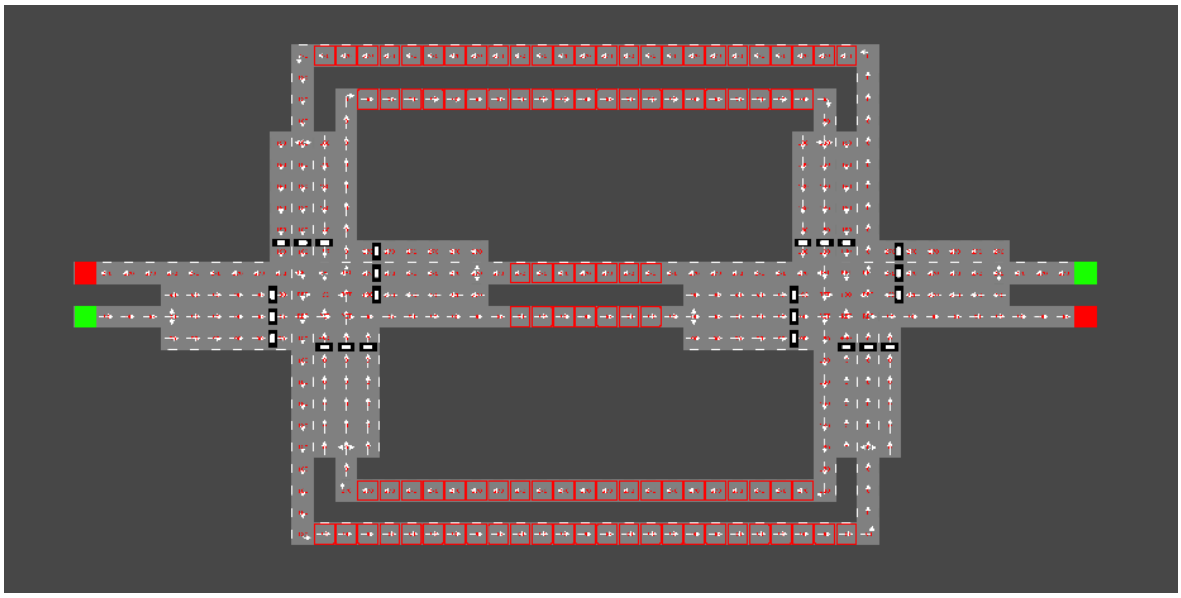


Figur 10. Den röda texten kommer från rotationslagret. Rutorna i rotationslagret innehåller endast röd text med en genomskinlig bakgrund. Detta för att navigationslagret syns bakom texten.

4.1.4 In- och utpunkter

Lagret för in- och utpunkter används för att planera var startpositioner och slutdestinationer för fordonet får befinna sig i systemet. Lagret anger även var in- och ut punkterna befinner sig i simuleringsmiljön. De rutor som har en röd ram runt sig anger att det är tillåtet att skapa ett fordon på den platsen eller att fordonets slutdestination kan vara på den rutan. De helt grönfärgade rutorna definierar in punkter i systemet och de helt

rödfärgade rutorna definierar utpunkter i systemet. Dessa rutor är nödvändiga för att skapa ett ämnesmässigt flöde i systemet, se kapitel 3.7.



Figur 11. Visualiseringen av lagret för in och ut punkter. I denna figur syns även navigationslagret och rotationslagret där var inga rutor för in och ut punkter placerats.

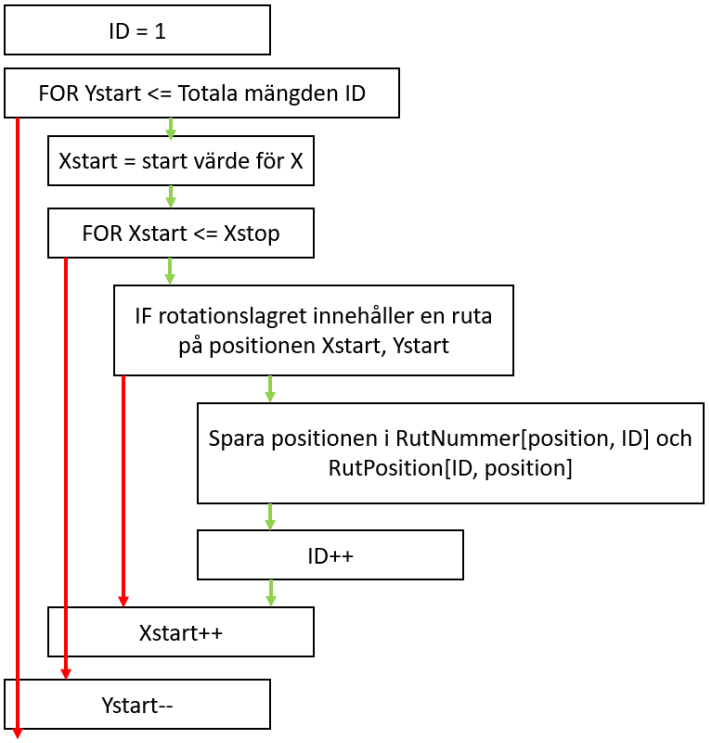
4.2 Skapande av data till simuleringen

I detta avsnitt behandlas den information som krävs för att få en fungerande simuleringsmiljö och hur det med hjälp av flödesscheman är möjligt att beskriva logiken i hur informationen skapats.

4.2.1 Numrering av rutor

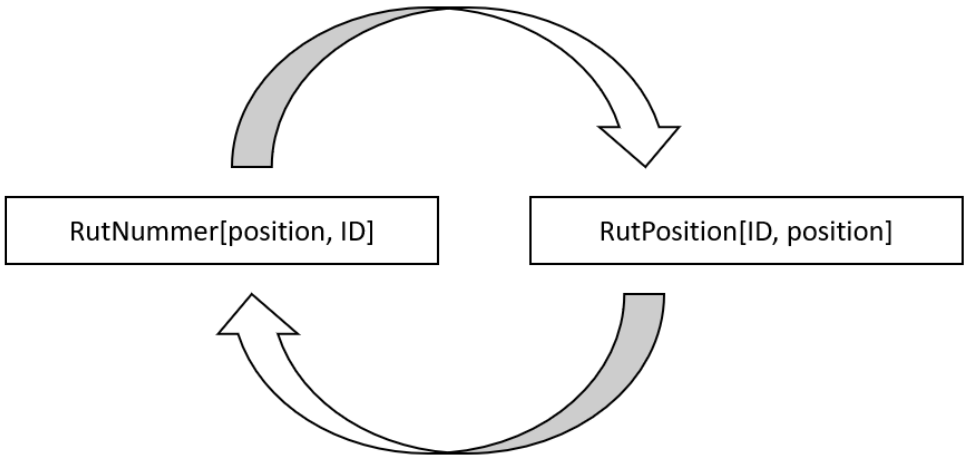
I denna process används startvärden och slutvärden för både X och Y som koordinater i systemet för att definiera storleken av simuleringen. Varje rutas position har en egen koordinat, och för att radnumreringen ska kunna räkna antalet rutor och ge rutorna ett individuellt ID måste området i systemet definieras som variabler. När denna funktion körs anger den ett individuellt ID åt alla rutor i rotationslagret. Mängden rutor i rotationslagret i förhållande till navigationslagret är 1:1. Funktionen söker igenom det på förhand definierade området och när funktionen hittar en ruta i rotationslagret sparas positionen av den rutan i RutPosition[] med ett nyckel-ID som används för att fråga registret var den

rutan finns. Den sparas också motsvarande i registret RutNummer[] som sedan används för att fråga vilket rutnummer som finns i den positionen.



Figur 12. Flödesschema för numrering av rutor i rotationslagret.

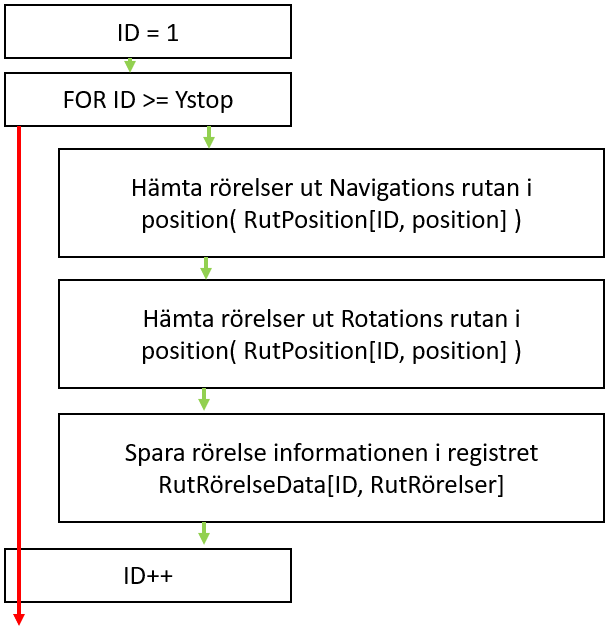
Dessa två register ger möjlighet att få veta rutans ID eller position beroende på vilken av de två nycklarna som finns till förfogande.



Figur 13. Sammankopplingen av de två register som används för att lätt slippa åt de olika rutornas positioner och ID:n.

4.2.2 Lagring av tillåtna rörelser

De rörelser som är tillåtna i simuleringen definieras som en egen ”Class”, som i detta fall heter RutRörelser[]. Denna ”class” får ”children”, som innehåller information om huruvida framåt, vänster- eller högersväng är tillåtna samt vilken rotation som rutan har. Genom att använda sig av en ”class” kan man skapa ett kombinerat register i stället för att hantera olika register för varje variabel.



Figur 14. Flödesschema för lagring av tillåtna rörelser.

4.2.3 Distansberäkning och -lagring

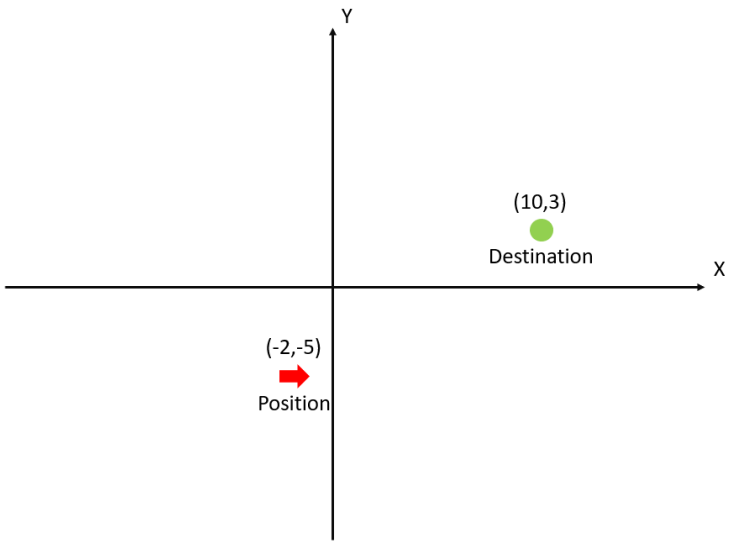
Beräkningen av distansen från en ruta till en annan sker när programmet startas. Denna beräkning sker för att definiera varje rutas och innehåller en kombination av rörelser som kan vara rakt fram, åt vänster eller åt höger. För varje riktning som finns räknas distansen till alla de övriga rutorna med start åt alla de håll som har tillåtits för den specifika rutan. Det vill säga om det finns totalt 100 rutor och rutan som den räknar från tillåter alla tre rörelser kommer den då att räkna distansen till alla rutor för att köra rakt fram, svänga vänster och svänga höger. Detta blir totalt 297 olika möjligheter eftersom det finns 99 möjligheter för varje riktning.

Funktionen för distanskalkylering prövar sig fram genom att använda de regler som finns i registret RutRörelseData[], se kapitel 4.2.2. När den har prövat fler antalet gånger än vad det finns rutor i systemet kommer den att returnera värdet -1 eftersom funktionen är då på en ruta som den tidigare varit på. Detta betyder att den inte hittade den destination som den sökte. Om den hittar destinationen som den sökte före den prövat högst så många gånger som det är tillåtet kommer den att returnera antalet försök som den gjort, vilket motsvarar mängden steg som den gjorde.

Värdena för hur långt det är från en punkt till en annan sparas i en tredimensionell Array som innehåller position, destination och riktning. Riktningsvärdet anges som ett värde mellan ett och tre, där ett motsvarar vänstersväng, två motsvarar rakt och tre motsvarar högersväng. Genom att sedan sätta in värdena för position, destination och riktning kan längden mellan positionen och destination för den riktningen fås fram. Distansen beräknas genom att göra antaganden av rutornas koordinater i rutsystemet. Den första rörelsen som kommer att göras från startrutan definieras av vilken riktningen som funktionen vill beräkna längden för. När det första försöket har gjorts kommer funktionen fortsätta genom att göra antaganden om vad som är det optimala alternativet att ta per ruta.

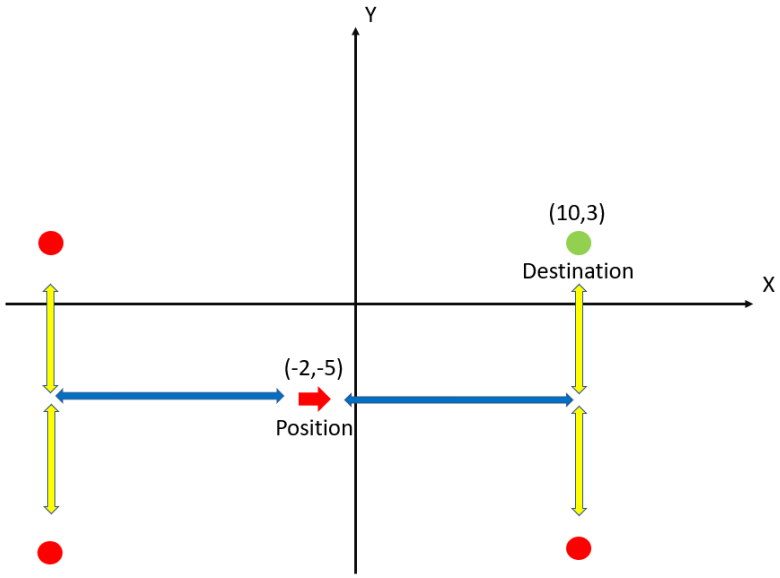
Före nästa steg görs jämförs förhållanden mellan distanserna på leden X och Y genom att beräkna skillnaden mellan positionens och destinationens koordinater. Denna jämförelse sker med absolutbeloppet för skillnaderna. På de positioner där alternativ finns väljer den sin riktning beroende på vilken rotation den har på sin nuvarande position och förhållanden mellan X och Y. Funktionen beräknar om destinationen är mera belägen mot sidorna eller framåt och beroende på vad skillnaden är gör den det steg som den tror är bäst. I situationer där destinationen är bakåt kommer den välja att svänga mot sidan. Funktionen svänger i detta fall åt det håll som är närmast. Ifall punkten är rakt bakom svänger den till höger. Ett exempel:

I *figur 16* har systemet riktningen 90 och är på position (-2, -5). Destinationen har positionen (10,3).



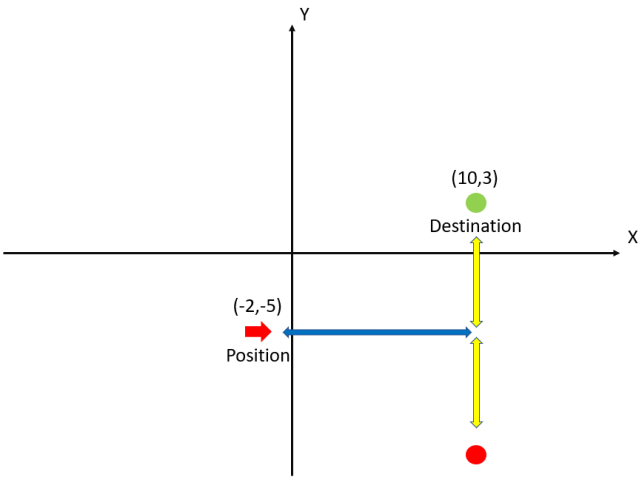
Figur 15. Befintliga positionen markerad som en röd pil och slut destinationen markerad som en grön punkt.

Programmet beräknar skillnaden mellan X – och Y-positionerna och jämför dem som absoluta värden. I *figur 15* är skillnaden i X – led större än skillnaden i Y – leden. Med denna beräkning kan programmet konstatera att destinationen finns i någon av positionerna i *figur 16* och att det är frågan om att svänga eller fortsätta rakt. Den första beräkningen sker i absolut format för att få den totala längden och lätt kunna jämföra skillnaden på X och Y mellan varandra. De röda punkterna i *figur 16* representerar positioner där det är möjligt att destinationen är i förhållande till positionen av funktionen. Den gröna punkten i *figur 16* är den destination som är den korrekta för funktionen. I detta fall ser funktionen på ifall skillnaden på X – led är positiv eller negativ. Denna beräkning görs för att identifiera ifall destinationen är bakåt eller framåt.



Figur 16. Befintliga positionen markerad som en röd pil och slut destinationen som en grön punkt. De tre röda punkterna är falska destinationer som inte uteslutits.

Systemet jämför sedan om skillnaden på X-koordinaterna är större eller mindre än skillnaden på Y-koordinaterna. I detta exempel som ses i *Figur 17,* konstateras att slutpunkten är rakt framåt och därför fortsätter funktionen framåt. Systemet har dock inte ännu konstaterat om slutpunkten efter det är mot vänster eller mot höger, men funktionen försöker alltid minska på den större axeln först.



Figur 17. Befintliga positionen markerad som en röd pil och slut destinationen som en grön punkt. Den röda punkten är en falsk destination som inte uteslutits.

När skillnaden i X – leden har minskat tillräckligt kommer systemet att försöka svänga till vänster så tidigt som möjligt och svängen kommer att ske så fort som en ruta i systemet tillåter det.

När systemet landar på rutor med rotationen RET eller +90 i rotationslagret sker ett undantag för att kunna styra trafiken så att fordonen inte sedan kan byta riktning olovligt i trafikkorsningar. Undantaget för RET är att den inte påverkas av navigations alternativen och får inte heller någon ny rotation. I detta fall fortsätter systemet att gå framåt med samma rotation som tidigare. Rotationen +90 gör att systemet söker sig tre steg framåt och två steg åt vänster. Orsaken till detta är att mittpunkten i korsningen är en gemensam ruta för alla vänstersvängar. Detta undantag hjälper dem att undvika varandra i simuleringen.

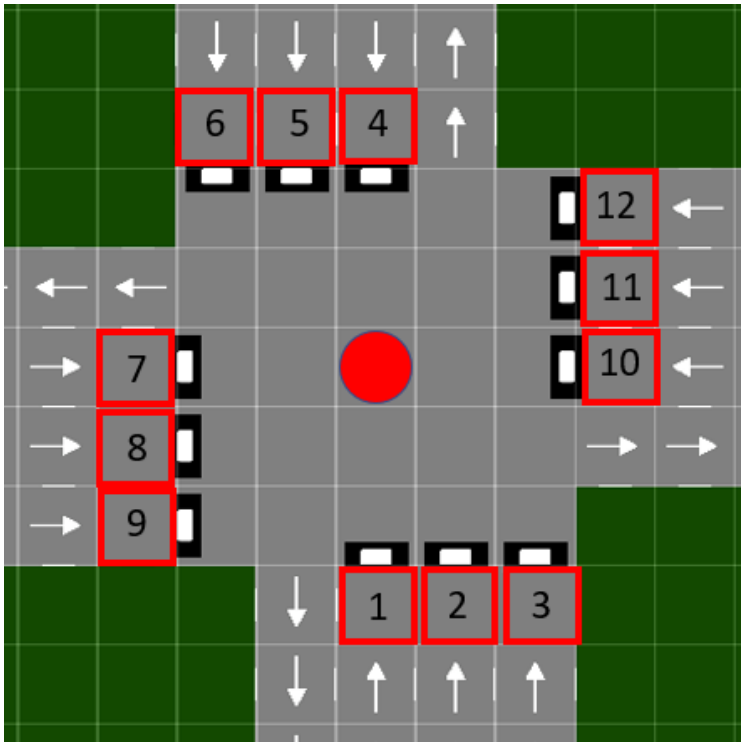
Efter att systemet gjort sitt steg framåt tar den värden för både rotationen och de tillåtna rörelserna för den rutan som den hamnat. Med dessa alternativ beräknar funktionen sedan vilken rörelse den ska göra för nästa steg.

Vid beräkningen av positionerna används koordinaterna för att på ett ungefär beräkna åt vilken riktning den ska fortsätta mot i stället för att gissa sig fram till alla de möjliga kombinationerna. När dessa distanser lagras i en Array kommer fordonet att kunna använda den kortaste möjliga riktningen för att smidigt söka sig fram till sin destination. Genom att göra denna beräkning i början av simuleringen klarar simuleringen av mer trafik eftersom systemet inte måste beräkna längderna i realtid.

4.3 Trafikljus

Trafikljussystemet är ett skilt objekt som placeras ovanpå rutsystemet men är inte knutet till rutnätets begränsningar. Trafikljussystemet använder sig av de tio olika lägena som nämndes i koncept delen, se kapitel 3.5. Trafikljussystemet har 20 startvariabler som används för att ställa in hur trafikljusen ska fungera. De nio första startvariablerna används för att definiera i vilken ordning som de olika funktionerna kommer i. För funktioner som inte används definieras deras respektive startvariabler till ett värde på noll. Trafikljussystemet skapar en roterande lista som innehåller de olika funktionera som trafikljuset ska använda sig av och i vilken ordning som de kommer. Ordningen skapas genom att ange det största värdet till startvariabeln för den funktion som ska visas först, det näst största värdet till den funktion som ska komma andra och så vidare. De nio andra

startvärdena som bör fyllas i är hur länge dessa funktioner ska vara aktiva. De två sista värdena som behövs är startfördröjning och körfördröjning. Startfördröjning definierar hur länge trafikljuset ska vänta i början av simuleringen före den börja köra igenom listan kontinuerligt. Denna funktion används för att ställa in nollpunkten för intervallet. Fördröjningen används för att definiera hur länge som trafikljussystemet är i funktion noll medan den byter från en funktion till en annan funktion. Trafikljussystemet skapar även ett register för hantering av placering av trafikljus i förhållande till rutsystemet i simuleringen, se kapitel 4.1. Detta register är gemensamt för alla trafikljus. Varje trafikljus tilldelas en individuell koordinat. Positionerna för vilken ruta som de ska påverka beräknas från mittpunkten av trafikljussystemet. För varje ändring av vilken funktion som är aktiv i trafikljussystemet uppdateras registret RutLjusHanterare[] där nyckeln som anges är positionen och värdet som anges är nivån av ljuset. Denna nivå anges som en siffra där ett är grönt och noll är rött.



Figur 18. Trafikljussystemets utseende och numrering.

4.4 Fordon

I detta kapitel beskrivs närmare hur fordonen navigerar genom simuleringsmiljön. kapitlet beskriver om hur fordonen hanterar andra fordon i simuleringsmiljön samt om hur bränslekonsumtion beräknas.

4.4.1 Navigation

Fordonen i denna simulering får ett startvärde som definierar till vilket nummer i systemet de ska navigera. Beroende på var dess startposition är kommer de att få data från rutan de befinner sig på om åt vilket håll de kan åka. Denna data får de genom att först fråga registret om vilket nummer den position de står på har. Beroende på vilka möjligheter som finns kommer fordonet att läsa distansen för varje tillåtna riktning från den array som skapades i kapitel 4.2.3. Då fordonet har valt den kortaste rutten kommer den att navigera till följande ruta. Efter att den nått den rutan kommer samma process börja om igen.

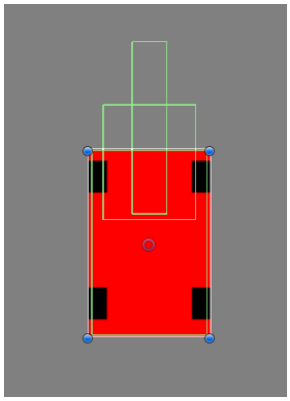
För positioner där rotationen är +90 eller RET används undantagsfunktioner som ändrar vart fordonet försöker röra sig, se kapitel 4.1.3. Ifall det är frågan om RET så kommer den ruta som fordonet rör sig mot att vara ett steg framför fordonet. När rotationen är +90 sätts den ruta som fordonet rör sig mot till tre steg framåt och två steg till vänster för att undvika samma konflikter som nämndes i kapitel 4.1.3. Rotationen av fordonet ändras till det som den varit +34 grader motsols eftersom den spetsvinkliga vinkeln mellan kateten och hypotenusan i en rätvinkligtriangel kan avrundas till 34 grader då längderna på kateterna är tre och två. Orsaken till att rotationen blir på detta vis är för att fordonet kommer röra sig diagonalt till vänster eftersom målet är diagonalt åt vänster.

Vid rutor som visar att både rakt, till vänster och till höger är det tillåtet för fordonet att göra en diagonal rörelse genom att tvinga den att sätta målet ett steg längre fram. Orsaken till att detta har gjorts är att den underlättar trafikstockningar som sker före trafikljusen.

4.4.2 Förhindring av kollision

Fordonen har tre olika kollisionsrutor som syns i *figur 19*. Dessa rutor används för att känna igen andra fordon. Rutan som täcker hela fordonet används för att låta andra fordon känna av den. Den långa smala rutan, används för att undvika fordonet framför, vilket sker genom att fordonet i fråga sakta ner. Den fyrkantiga rutan i fronten av fordonet anger att fordonet

behöver stanna för att undvika att röra ett annat fordon. För att syfta på denna kollisionsruta senare i texten har ordet stopruta använts.



Figur 19. Kollisionsrutorna för fordonet.

4.4.3 Rörelse

I normala fall gör fordonet endast steg på en ruta åt gången. Den enda rutan som kan tillåta att göra steg längre än en ruta åt gången är rotationsrutan +90. Distansen mellan mitten av fordonet och mitten av rutan som den rör sig mot jämförs hela tiden. När skillnaden mellan positionen av fordonet och mitten av rutan som den rör sig mot är innanför toleransen som angivits i startvariablerna kommer navigationsdelen att ge en ny koordinat som fordonet ska söka sig mot. Fordonet tar sin riktning från rotationsvärdet för den ruta som den rör sig mot. Den högsta hastigheten som fordonet får röra sig med är angivet i startvariablerna, samt att accelerationskoefficienten är angiven i startvariablerna.

Beroende på kollisionsrutornas logiska nivåer, kommer hastigheten av fordonet att ändras. Kollisionsrutornas logiska nivåer ändras när de kommer i kontakt med en annan kollisionsruta. I situationer där fordonet försöker sakta ner för att undvika fordonet framför kommer den att retardera om den är vid max hastigheten genom att halvera sin hastighet. Annars kommer den att accelerera med hälften av accelerationskoefficienten, som var given i startvariablerna. Ifall fordonets stopruta rör ett annat fordon kommer bilens hastighet att bli noll. Varje gång som fordonet når sitt mål uppdaterar den registren FordonsIDpositioner[] och FordonsPositioner[] för att resten av systemet ska veta var fordonen befinner sig. När fordonet når sin destination förstörs bilen och variablerna raderas ur registren FordonsIDpositioner[] och FordonsPositioner[].

4.4.4 Bränslekonsumtion

Bränslekonsumtionen beräknas kontinuerligt i de olika accelerationsfaserna och körfaserna. Formeln för bränslekonsumtionen har skapats genom att generalisera förbrukningen av 1,4 liters bensinmaskiner. Den data som använts för att skapa formeln har tagits ur ett tidigare arbete (Soborino, Hernandez, & Monzón, 2014). Den matematiska formeln för bränsleförbrukning som använts är:

$\frac{g}{km} = (x-70)^2 \times \sin(0.013) + 37$ där x är hastigheten i km/h.

Denna formel för att räkna bränslekonsumtion i g/s genom att använda formeln:

$\frac{g}{s} = \frac{(\mathrm{x}-70)^2 \times \sin(0.013) + 37}{\frac{\mathrm{x}}{1} \times 3600}$ där x är hastigheten i km/h.

Hastigheten i m/s beräknas genom att ta rörelsehastigheten gånger fyra. Orsaken till detta är att förhållandet mellan en enhet i simulering motsvarar ungefär 4 meter. För att konvertera till km/h tas hastighet gånger 3,6.

I de olika faserna där fordonet accelererar eller kör kontinuerligt beräknas bränslekonsumtionen om till g/s och adderas till den totala konsumtionen för fordonet. När fordonet accelererar multipliceras den tillfälliga konsumtionen med två och detta adderas sedan till den totala summan. När fordonet når sin slutdestination rapporteras den totala konsumtionen och sträckan till konsumtionshanteraren.

4.5 Trafikskapare

I detta kapitel tas det upp trafikskaparens startvariabler som innehåller information om hur många fordon som får vara aktiva samtidigt i simuleringsmiljön, vikter som påverkar funktionen skapar rutter för fordonen och med vilket intervall den ska skapar fordonen.

4.5.1 Hantering av aktiva fordon

Registret FordonsIDpositioner[] använder fordonets ID som nyckel och där förvarar den fordonets position. Registret FordonsPositioner[] använder sig av fordonets position som nyckel och förvarar fordonets ID. Dessa två register används för att undvika bilarna skapas på varandra när de kommer in i simuleringsmiljön. När de blir borttagna ur miljön

försvinner nyckel ID:t ur registret. Registren används även för att övervaka hur många ID:n, det vill säga fordon, som är aktiva.

4.5.2 Viktning av ruttyp

Trafiken skapas slumpmässigt från ett tal som landar mellan noll och tio. Detta tal jämförs sedan med den undre vikten där variabelns namn är InPunktTillStan. Ifall det slumpmässiga talet är mindre än variabeln InPunktTillStan kommer ett fordon att skapas från en in punkt med en destination vid en slumpmässig punkt i simuleringsmiljön, se kapitel 4.1.4.

Slutdestinationen väljs också slumpmässigt mellan ett och så många rutor som finns i simuleringsmiljön (maxantalet tas från största ID:t, se kapitel 4.2.1). Ifall rutan som blivit vald finns med i in och ut punkts lagret kommer denna att bli fordonets slutdestination. Om denna ruta inte finns med i in och ut punkts lagret kommer funktionen försöka skapa en ny rutt med en annan slumpmässig destination.

Ifall talet är större än startvariabeln StanTillUtPunkt kommer ett fordon att skapas i simuleringsmiljön med slutdestinationen vid en ut punkt. Startpunkten kommer att bli slumpmässigt vald på samma sätt som slutdestinationen blev vald i den tidigare varianten. Ifall det slumpmässiga talet landar mellan StanTillUtPunkt och InPunktTillStan kommer ett fordon bli skapat med både start- och slutpunkten in i miljön. Detta kräver att både start- och slutpunkten finns i in- och utpunktslagret.

4.5.3 Skapande av fordon

För varje ID som är ledigt i registret FordonsIDpositioner[], se kapitel 4.5.1, kommer systemet att försöka skapa ett nytt fordon och detta kommer att ske slumpmässigt varje gång. När den skapar ett fordon kommer den att kontrollera att det inte finns något annat fordon innanför ett område på 5x5 rutor. Detta gör den genom att samla data om positioner från registret FordonsPositioner[]. Om detta register ger som svar att det finns ett fordon innanför dessa rutor kommer den att avbryta skapandet av ett nytt fordon. Ifall det tillåts att skapa ett fordon kommer fordonet att få startvariabler som innehåller både start position och slut destination från funktionen, se kapitel 4.5.2, och ett ID. Samtidigt som fordonet skapas kommer det att sättas med i registren FordonsPositioner[] och FordonsIDpositioner[].

4.6 Konsumtionshanteraren

Funktionen för bränslekonsumtion får som startvärde antalet fordon som den ska erhålla medeltal av. Från detta skapas två arrayer av samma storlek. Den ena innehåller förbrukningen i gram och den andra innehåller sträckan i meter. För varje körning som rapporteras byter den position i arrayen. Totala värdet av arrayerna beräknas så att medelvärdet av den totala förbrukningen från de senaste körningarna kan beräknas. Denna siffra visas sedan för användaren i övre högra hörnet och ger en referens till om ändringarna har gjort flödet mera eller mindre effektivt.

5 Resultat

Simuleringen som skapats klarar av att simulera stora mängder trafik i olika system. Systemet kan bestå av flera olika korsningar med olika konfigurationer. För att skapa ett mer realistiskt flöde finns in- och ut punkter som används för att rikta trafiken. Dessa in- och ut punkter i in- och utpunktslagret kan användas för att representera motorvägsanslutningar eller liknande för att påvisa att trafiken främst kommer in eller åker ut via en viss punkt samt hur styrsystemet för trafikljusen påverkar effektiviteten av systemet. Genom att lägga olika vikter på hur trafiken skapas kan olika situationer simuleras som till exempel rusningstrafik där trafikens riktning är mer betonat åt ett håll.

Trafiksimuleringen fungerar genom att först skapa några databaser som innehåller alla värden färdigt beräknade som krävs för att kunna köra simuleringen. Genom att beräkna det som är möjligt före simuleringen startar, minskas beräkningarna under simuleringstiden. Detta gör att simuleringen klarar av att köra mer trafik utan att simuleringsmiljön saktas ner.

Rutsystemet som simuleringen bygger sig på tillåter simuleringen att konfigureras för olika ändamål. Detta gör att simuleringen kan användas för att simulera olika trafiklösningar och sedan se på hur konfigurationen påverkar bränsleförbrukningen i systemet. Genom att skapa ett register av färdigt konfigurerade trafikljus funktioner kan flödet av systemet smidigt justeras så att det är så nära optimalt som möjligt.

Själva simuleringen är inte kompilerad som en produkt eftersom den använder variabler och verktyg i Unity för att konfigureras.

6 Kritisk granskning och diskussion

Simuleringen använder sig av bara en bränsleförbrukningsmodell som generaliserats till en matematisk formell. I simuleringen kunde det ha tagits i beaktan större maskiner och olika typer av maskiner men i detta skede har bara en modell använts.

Simuleringen har inte validerats eftersom det kräver väldigt mycket data för att validera den. En funktion som saknas för att validera simuleringen är även ett sätt att återskapa samma trafikflöde ur ett register eller lista. I simuleringen har ett säkerhetsavstånd mellan bilarna på cirka två sekunder skapats under de flesta körningar. Den verkliga stadskörningen har det inte i detta examensarbete studerats eller implementerats i simuleringsmiljön.

Det finns även vissa funktioner när det kommer till trafikljus som är möjliga i verkligheten men som inte implementerats i denna simulering. En sådan funktion kunde vara att trafikljussystemets funktion skulle tillåta att både position ett och position nio skulle vara aktiva. Denna funktion förekommer i den verkliga värden men systemet som fordonen använder för att undvika varandra klarar inte av det eftersom fordonen tror att de kommer att kollidera och därför så stannar de. Detta gör att hela korsningen stannar och i teorin kunde det vara möjligt att hela trafiken står stilla. Fordonen i simuleringen klarar inte av att väja för varandra och därför kräver denna simulering trafikljus vid alla korsningar.

Om denna simulering skulle kunna valideras skulle det finnas möjlighet till att använda simuleringen för att planera ändringar i befintliga trafiksystem för att förbättra trafikflödet och för att minska utsläpp i trafiken.

7 Källor

AutoNavi. (2018). *2018 Q1 traffic analysis report of major cities in china.* Peking: amap.com.

C# Arrays. (den 1 10 2021). Hämtat från Microsoft: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/

C# Classes. (den 15 9 2021). Hämtat från Mircosoft: https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/classes

C# Dictionary. (den 15 9 2021). Hämtat från Microsoft: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-initialize-a-dictionary-with-a-collection-initializer

Haas, J. (2014). *A History of the Unity Game Engine.* Worcester: Worcester Polytechnic Institute.

Kommunikationsministeriet. (den 10 8 2018). *Vägtrafiklag 729/2018*. Hämtat från FINLEX: https://www.finlex.fi/sv/laki/alkup/2018/20180729

Soborino, N., Hernandez, S., & Monzón, A. (2014). *Reduced Carbon and Energy Footprint in Highway Operations: The Highway Energy Assessment (HERA) Methodology.* Madrid.

Unity Documentation. (2021). Hämtat från Unity: https://docs.unity3d.com/Manual/UnityManual.html

Xerox. (u.d.). *Make your city flow.*