Emil Virolainen

Liikennesimulaattorin aliohjelmien käyttö ja visualisointi 3D-pelimoottorissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

6.9.2022

Tiivistelmä

Tekijä:	Emil Virolainen
Otsikko:	Liikennesimulaattorin aliohjelmien käyttö ja visualisointi 3D-pelimoottorissa
Sivumäärä:	32 sivua
Aika:	6.9.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaajat:	Lehtori Miikka Mäki-Uuro Lehtori Antti Laiho

Insinöörityön tarkoituksena oli kehittää lisäominaisuuksia SUMO-liikennesimulaattorin avulla toimivaan liikenteenvalvontasovellukseen. Sovelluksen kehitystyö toteutettiin Unity-pelimoottorilla ja C#-ohjelmointikielellä käyttäen SUMO-simulaattorin mukana tulevaa TraCI-rajapintaa.

Autonominen ajoneuvo on ajoneuvo, joka suoriutuu ajotehtävästä ilman kuljettajaa. Autonomisille ajoneuvoille voidaan luokitella eri autonomian tasoja. SUMO-simulaattori kykenee simuloimaan autonomista eli itsenäistä liikennettä rajatulla alueella. Simulaattoria käytettiin liikenteenvalvontasovelluksen pohjana, johon kehitettiin lisäksi simuloituja liikenteen ongelmatilanteita käyttäjän testaamista varten.

Sovellukseen kehitettiin ensin oikean maailman liikennettä kuvaava liikennealue käyttämällä liikennesimulaattoria. Liikenteeseen kehitettyihin erilaisiin satunnaisesti ilmestyviin ongelmatilanteisiin kuului muun muassa yksittäisen ajoneuvon tekninen ongelma tai tieosuuden paikallinen tai kokonainen tukkiutuminen. Sovellusta varten kehitettiin graafinen käyttöliittymä, joka sisältää aktiivista liikennekuvaa näyttävän karttanäkymän, listan aktiivisista liikenteessä olevista ajoneuvoista sekä painikkeita, joilla yksittäisille ajoneuvoille voi antaa komentoja. Käyttöliittymään kuului myös kameranäkymä, joka kuvasi valitun ajoneuvon tilannetta toistamalla erilaisia videoita näkymässä.

Insinöörityön lopputulos oli valvontasovellus, jolla voi valvoa autonomisista ajoneuvoista koostuvaa liikennettä. Sovellus toteutettiin osana Metropolia Ammattikorkeakoulun SAM-hanketta (lyhennys sanoista Smart Autonomous Mobility). Hanke ja sen pohjalta kehitetty insinöörityö tukevat oikean maailman itsenäisen liikenteen kehitystyötä simuloimalla oikean maailman liikennejärjestelyjä autonomisilla ajoneuvoilla ja tarjoamalla toimivan alustan niiden vaatimusten kartoittamiselle, joita itsenäisen liikenteen valvontaan sisältyy.

Avainsanat: liikennesimulaattori, itsenäinen liikenne

Abstract

Author:	Emil Virolainen
Title:	Application and visualization of a traffic simulator's subroutines in a 3D game engine
Number of Pages:	32 pages
Date:	6 September 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Game Applications
Supervisor:	Miikka Mäki-Uuro, Senior Lecturer Antti Laiho, Senior Lecturer

---

The purpose of this final year project was to further develop a traffic control application for controlling autonomous vehicles. The application uses the SUMO traffic simulator and was developed in the Unity game engine using C#, and the traffic simulator programming interface TraCI.

An autonomous vehicle can drive without the help of a human driver. They can be classified into different levels of autonomy. The SUMO traffic simulator can simulate autonomous traffic in a limited area. It was used as a basis for the traffic control application.

An area with autonomous traffic was simulated in the traffic control application. The area was then augmented with artificial problem scenarios affecting the traffic flow. A single problem scenario could be a technical malfunction in a vehicle, or the blockage of a road. A graphical user interface was also developed for the application, which consisted of a map, a list of vehicles, and buttons for controlling the vehicles.

The result of the project was a functioning autonomous traffic control application as part of Metropolia's Smart Autonomous Mobility, or SAM project, which aims to study a user's ability to supervise autonomous vehicles using a traffic control application. The project also supports development of autonomous mobility by simulating real world traffic arrangements with autonomous vehicles.

Keywords: traffic simulator, autonomous mobility

Sisällys

1 Johdanto

Insinöörityö tehtiin osana autonomisen liikenteen valvontasovelluksen kehitystyötä. Työn tarkoituksena oli kehittää erilaisia liikenteessä ilmeneviä ongelmatilanteita liikennesimulaatiota ja liikenteenvalvontasovellusta varten. Tämän saavuttamiseksi oli tarkoitus hyödyntää simulaattoriin kuuluvan rajapinnan komentoja, joilla simulaatiota ja sen kulkua saattoi muokata reaaliajassa.

Sovellus kehitettäisiin liikennesimulaattorista, jota ajetaan yhdessä kolmiulotteisen pelimoottorin kanssa. Näin simulaattorissa oleva tieto voidaan visualisoida käyttäjää varten pelimoottorissa, jolla graafisen käyttöliittymän kehittäminen on helppoa. Pelimoottori myös mahdollistaa videoiden toistamisen ja reaaliaikaisen liikennesimulaatiota kuvaavan kartan näyttämisen käyttäjälle. Itse sovelluksen päämäärä olisi tukea hanketta, jonka tarkoituksena olisi kerätä tietoa siitä, kuinka monta ajoneuvoa yksi käyttäjä kykenee tehokkaasti valvomaan samanaikaisesti autonomisen liikenteen valvontasovelluksella.

Insinöörityö voitaisiin toteuttaa ohjelmoimalla erilaisia lisäominaisuuksia liikennesimulaattoriin käyttämällä C#-ohjelmointikieltä, jota liikennesimulaattorin rajapinta ja Unity-pelimoottori käyttävät.

Insinöörityö perustuu Metropolia Ammattikorkeakoulun SAM-hankkeeseen, jonka tarkoituksena on kehittää autonomisten ajoneuvojen ja liikenteenvalvonnan teknologiaa ja liiketoimintaa. Hankkeesta on tehty muitakin insinööritöitä. Nämä työt keskittyvät muun muassa pelimoottorin ja liikennesimulaattorin väliseen integraatioon, valmiin sovelluksen graafiseen käyttöliittymään tai sovelluksessa käytettyyn tietokantaan. Tämä insinöörityö keskittyy liikenteenvalvontasovellusta varten kehitettyihin liikenteen ongelmatilanteisiin, niiden toteutustapoihin ja visualisointiin graafisessa käyttöliittymässä.

Luvussa 2 esitellään autonomisen liikenteen määrittelyjä. Luvuissa 3 ja 4 perehdytään projektissa käytettyyn liikennesimulaattoriin ja pelimoottoriin. Luvussa 5 käydään läpi liikenteenvalvontasovelluksen kehitystyötä, siinä käytettyjä

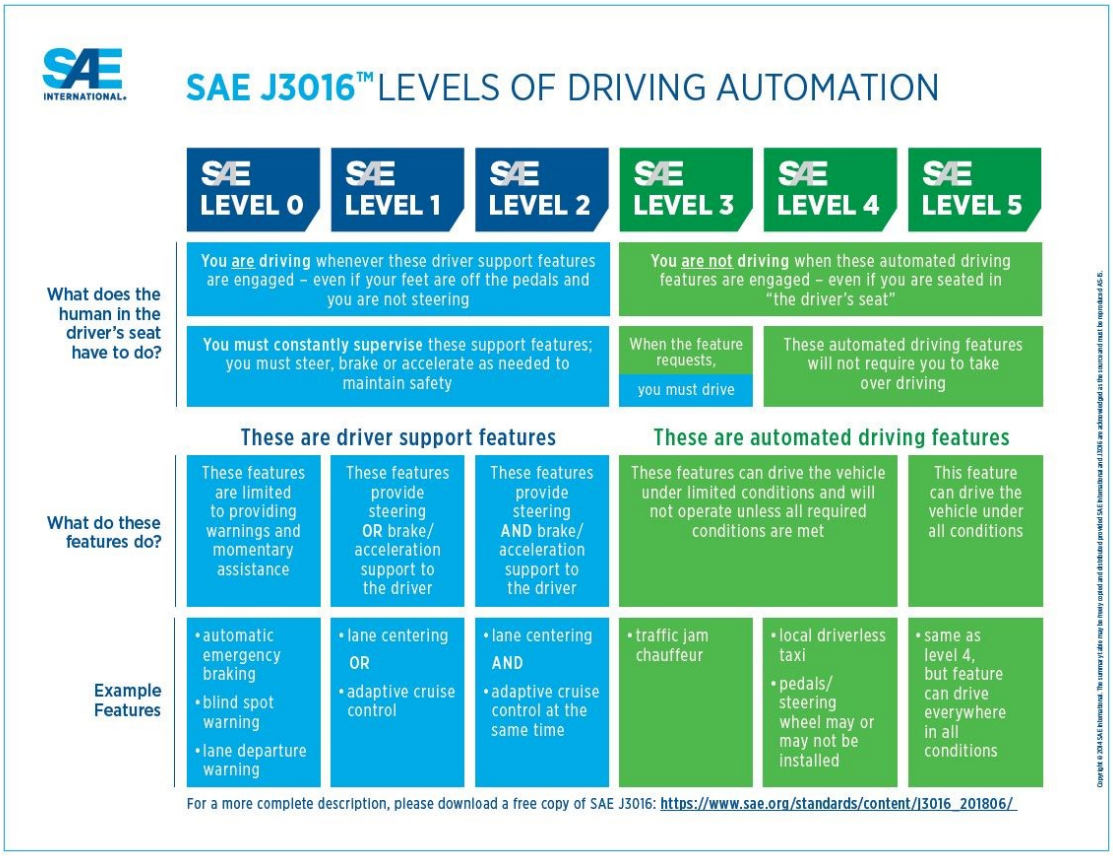
työskentelymenetelmiä ja liikennesimulaattorissa ja pelimoottorissa tapahtunutta kehitystyötä.

2 Autonominen liikenne

Suomen liikenne- ja viestintävirasto Traficom määrittelee autonomisen ajoneuvon seuraavasti: ”Autonominen ajoneuvo (engl. autonomous vehicle) kykenee suoriutumaan ajotehtävästä ilman kuljettajaa ja ilman yhteyttä muihin ajoneuvoihin tai infrastruktuuriin” (1). Liikenne, joka insinöörityössä simuloitiin SUMO-nimisellä liikennesimulaattorilla, kuvaa täysin autonomista liikennettä eli liikennettä, jossa kaikki ajoneuvot ovat Traficomin määritelmään sopivia autonomisia ajoneuvoja.

SAE International -nimisen insinöörialojen standardienkehitysyhdistyksen julkaiseman standardin J3016 (kuva 1) mukaan itseajaville ajoneuvoille on määritelty kuusi eri autonomian tasoa (2). Ensimmäisessä kolmessa tasossa vaaditaan, että ihmisajaja seuraa aktiivisesti liikenneympäristöä, kun taas kolmessa jälkimmäisessä tasossa automatisoitu ajamisjärjestelmä suorittaa liikenneympäristön seuraamisen. Kuusi eri tasoa ovat lyhyesti selitettynä seuraavanlaiset: Tasossa 0 ajoneuvon järjestelmät eivät hallitse ajoneuvoa, mutta voivat antaa varoituksia ja väliaikaisesti puuttua tilanteeseen. Tasossa 1 ajaja ja automatisoitu järjestelmä jakavat ajoneuvon hallinnan. Esimerkiksi vakionopeudensäädin tai kaistanpitoapu ovat automatisoidun tason 1 ominaisuuksia. Tasossa 2 automatisoidulla järjestelmällä on ajoneuvon täysi hallinta, eli kiihdytys, jarrutus ja kääntäminen. Ihmisajajan täytyy kuitenkin olla valmiina ottamaan hallinta välittömästi, jos automatisoitu järjestelmä lakkaa toimimasta. 3. taso on ensimmäinen, jossa ajajan huomiota ei vaadita. Ajoneuvo reagoi itsenäisesti kaikkiin tilanteisiin, kuten hätäjarrutusta vaativiin tilanteisiin. Ajajan pitää olla valmis vastaamaan ajamisesta tietyissä tilanteissa määritellyn ajan kuluttua. Tasossa 4 ajajan huomiota ei vaadita ollenkaan. Automatisoitu järjestelmä vastaa kaikesta ajamiseen liittyvästä toiminnasta, ja ajaja voi esimerkiksi nukkua ajon aikana. Tason 4 automatisoitu järjestelmä toimii vain rajoitetuilla, sen tason automatisoiduille ajoneuvoille tarkoitetuilla alueilla. Tason 5 täysin autonominen ajoneuvo ajaa

itsenäisesti kaikkialla eikä ajoneuvo esimerkiksi edes vaadi ohjauspyörää. Näin ollen projektissa simuloidun liikenteen ajoneuvot voivat lukeutua joko tason 4 tai jopa tason 5 autonomisen ajoneuvon määritelmään.

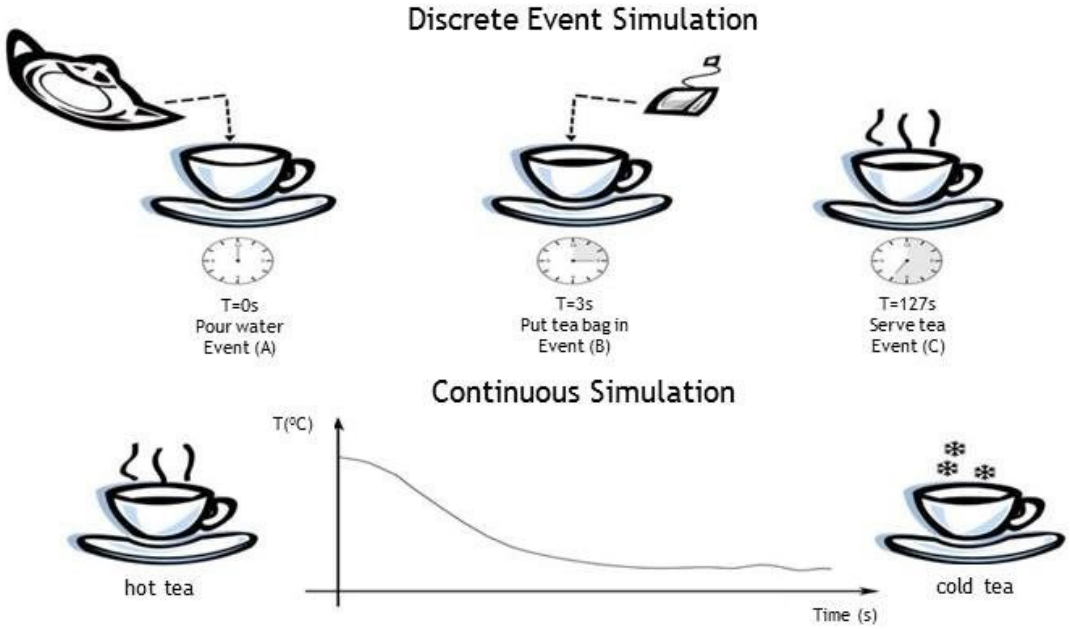


Kuva 1. SAE Internationalin taulukkografiikka ajoneuvon automaation eri tasoista (3).

3 SUMO-liikennesimulaattori

Teknillisen korkeakoulun (nyk. Aalto-yliopisto) liikennetekniikan professorin Matti Pursulan vuonna 1999 kirjoittaman liikennesimulaatioita käsittelevän yleiskatsauksen (4) mukaan liikenteensimulointiohjelmien sovelluksia voidaan luokitella monella tavalla. Simulaatioiden perusluokitteluja ovat muun muassa mikroskooppiset ja makroskooppiset sekä jatkuvan ja diskreetin ajan lähestymistavat (kuva 2). Mikroskooppisessa simulaatiossa käsitellään yksittäisten ajoneuvojen vuorovaikutusta, kun taas makroskooppisissa simulaatioissa simulaation

kohteena ovat suuremmat ajoneuvoryhmät. Jatkuvan ajan lähestymistavassa simulaatio kulkee tietyllä vakionopeudella, jossa simulaatiossa tapahtuvia tapahtumia seurataan kuluvan ajan mukana. Diskreetin ajan lähestymistavassa seurataan simulaation kulkua tapahtumapainotteisesti.



Kuva 2. Diskreetin ja jatkuvan ajan simulaation erot esitettynä teenkeittämisesimerkin avulla (5).

Ongelma-alueena simulaatiossa voi olla risteys, tieosuus tai kokonainen tieverkosto. Pursula selittää, että erityisalueita liikennesimulaatioissa ovat liikenneturvallisuus ja edistyksellisen liikenneinformaation ja ohjausjärjestelmien vaikutukset sekä erilaisten tarpeiden arviointi mikroskooppisten simulaatioiden avulla.

Liikenteen simulointi toteutettiin insinöörityössä Eclipse SUMO (lyhennys sanoista Simulation of Urban MObility) -nimisellä kaksiulotteisella avoimen lähdekoodin liikennesimulaattorilla (6). SUMO on mikroskooppinen jatkuvan ajan liikennesimulaattori, joka kykenee useamman ajoneuvon samanaikaiseen simulointiin rajatun kokoisella kartalla. Simulaattorilla on oma graafinen käyttöliittymä, mutta sitä voi myös ajaa suoraan komentoriviltä ilman graafista käyttöliittymää. Simulaattorin kehityksestä vastaa DLR:n (saks. Deutches

Zentrum für Luft- und Raumfahrt) eli Saksan ilmailukeskukseen kuuluva liikennejärjestelmien instituutti (saks. Institut für Verkehrssystem-technik) (7).

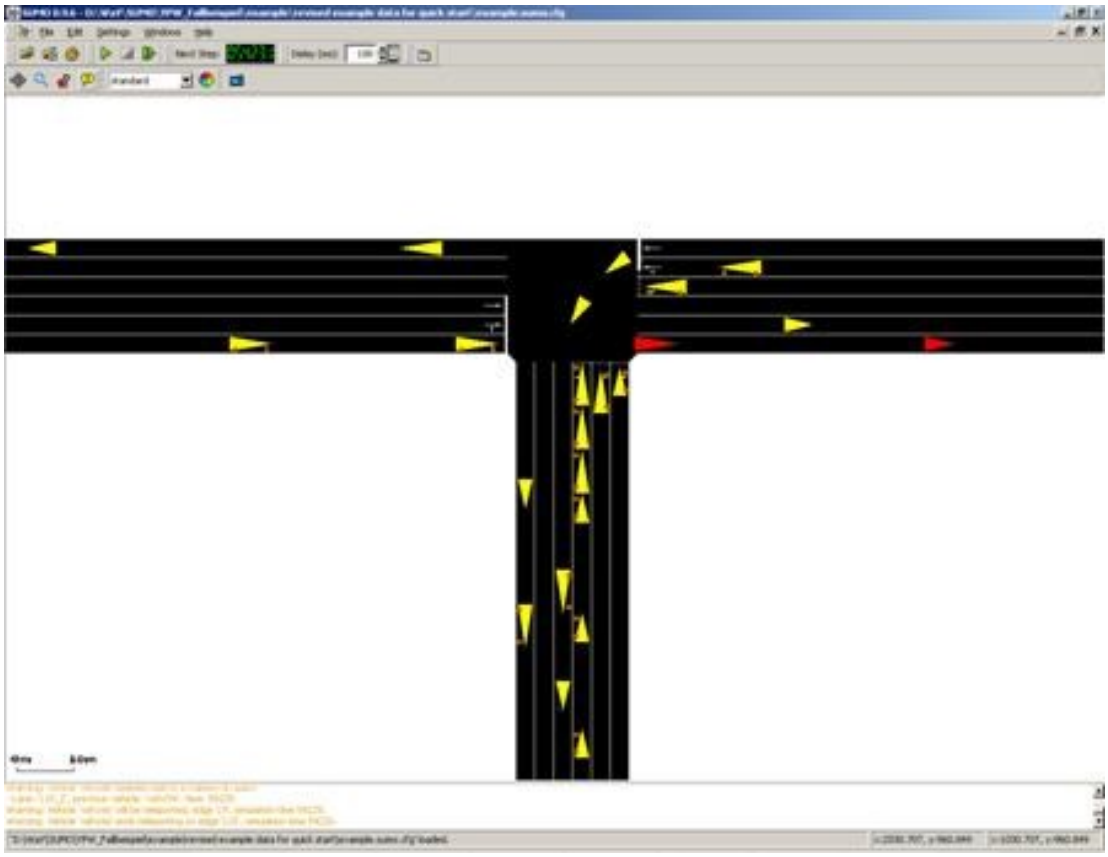
3.1 Rajapinta

Projektissa käytettiin TraCI (Traffic Control Interface) -nimistä rajapintaa, joka on suunniteltu SUMO-liikennesimulaattorin reaaliaikaista käsittelyä varten (8). SUMO:n dokumentaation mukaan TraCI antaa pääsyn käynnissä olevaan tieliikenteen simulaatioon ja mahdollistaa simuloitujen kohteiden arvojen noutamisen ja niiden käsittelyn. TraCI:n dokumentaation mukaan TraCI käyttää TCP-pohjaista asiakas-palvelinarkkitehtuuria, joka tarjoaa pääsyn SUMO-simulaatioon (8). Näin ollen SUMO toimii palvelimena, joka voidaan käynnistää komentoriviltä lisävaihtoehdoilla. Jos esimerkiksi palvelimen portti annetaan käynnistysvaihtoehtona, SUMO kuuntelee tätä porttia saapuvia yhteyksiä varten. Tällöin SUMO valmistelee simulaation, minkä jälkeen se odottaa, että ulkopuoliset sovellukset yhdistävät siihen ja ottavat hallinnan.

TraCI-rajapinta sisältää valmiita komentoja, joita voi kutsua suoraan ohjelmakoodista. Komennot vaativat viittauksen olemassa olevaan SUMO-simulaatioon, jolle komennot suoritetaan. Komennoilla voi mm. pysäyttää ajoneuvoja, muuttaa ajoneuvojen nopeutta, reittiä tai päämäärää tai muuttaa tieosuuksien nopeusrajoituksia.

3.2 Graafinen käyttöliittymä

SUMO-ohjelmaa voi käyttää sen omalla graafisella käyttöliittymällä (kuva 3), sumo-gui:lla (9). Tämä mahdollistaa simulaation helpomman tarkkailun, sillä käyttöliittymässä näkyy simulaatioon asetettu tieliikenneverkosto ja siinä toimiva liikenne. Käyttöliittymällä voi säätää erilaisia asetuksia reaaliajassa, kuten liikenteen kulun nopeutta tai erillisten ajoneuvojen muuttujien arvoja.



Kuva 3. sumo-gui-ohjelman graafinen käyttöliittymä (10).

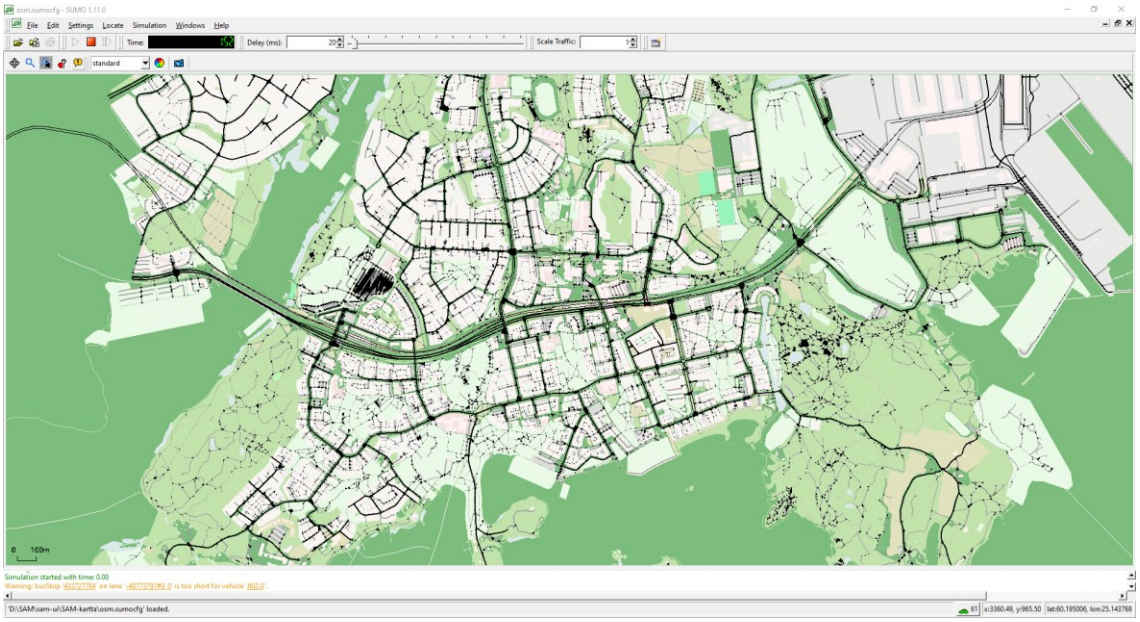
3.3 Simulaation osat

Ajoneuvot ovat tärkeä osa simulaatiota. Ajoneuvoille on simulaatiota varten määritelty monta eri abstraktia luokkaa, jotka kuvaavat erilaisia ajoneuvoja liikenteessä. Oletusarvoisesti eri ajoneuvoluokkia on 26 (11), esimerkiksi ”passenger”; eli henkilöautoluokka, joka on simulaation oletusajoneuvoluokka. Myös jalankulkijat, kevyt liikenne ja julkinen liikenne on kuvattu eri ajoneuvoluokkien avulla. Ajoneuvolla on monta eri muuttujaa, joiden arvoja voi simulaation ajon aikana muuttaa TraCI:n komentojen avulla.

Simulaation kartta koostuu tieosuuksista, joita simulaatiossa kutsutaan englanniksi nimellä edge. Tieosuuksien lisäksi kartassa on risteyksiä (junction), jotka eivät ole tieosuuksia, vaan oma erillinen osa simulaatiota omine muuttujineen. Tieosuuksien käsittely simulaation ajon aikana oli iso osa ongelmatilanteiden toteutuksia.

Jokaisella simulaatiossa olevalla ajoneuvolla on myös oltava reitti. Reitti koostuu listasta peräkkäisiä tieosuuksia. Reitti voi olla minkä tahansa pituinen, myös vain yhden tieosuuden pituinen. Ongelmatilanteissa käytettävillä esteajoneuvoilla on muun muassa yhden tieosuuden mittainen reitti. Ajoneuvon reittiä voi muuttaa ajon aikana, joko määrittelemällä uuden reitin tekemällä listan siihen kuuluvista tieosuuksista ja antamalla sen ajoneuvolle tai käyttämällä TraCI:n uudelleenreitityskomentoa, jolle annetaan parametriksi valitun ajoneuvon haluttu päämäärä. Komento laskee ajoneuvolle automaattisesti uuden reitin päämäärään.

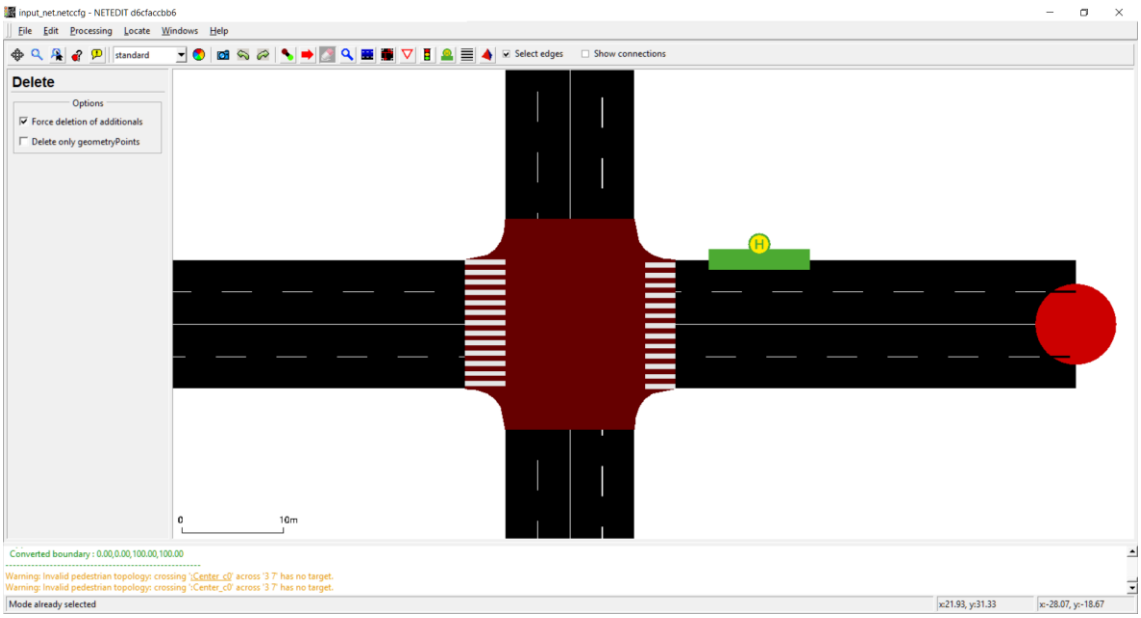
Liikennesimulaattoria ajetaan sitä varten luodussa liikenneverkostossa (engl. network). Projektia varten mallinnettiin versio oikean maailman Vuosaaren liikenneverkostosta (kuva 4). Verkostoon mallinnettiin tiet, risteykset ja julkinen liikenne oikean maailman mukaan. Mallinnusta varten käytettiin OpenStreetMap-nimistä avoimen lisenssin karttapalvelua, joka toimi yhdessä SUMO:n netconvert-nimisen tieverkostogeneraattorin kanssa. OpenStreetMap-palvelua (12) käyttävässä SUMO:n työkaluihin kuuluvassa OSMWebWizardissa valittiin alue, joka haluttiin mallintaa. Alue tallentui osm-muotoisena tiedostona, josta netconvert muunsi sen xml-tiedostoksi SUMO:a varten. Näiden työkalujen avulla generoitiin liikenneverkosto mallinnettuna suoraan oikean elämän tieverkostosta.



Kuva 4. Osa Vuosaaresta mallinnetusta alueesta sumo-gui-ohjelmassa.

3.4 Verkostonmuokkaus

SUMO:n ohjelmistopakettiin kuului myös netedit-niminen graafinen tieverkostonmuokkausohjelma (kuva 5). Ohjelmalla voi muokata ja luoda tieverkostoja graafisen käyttöliittymän avulla. Itse tiedostot ovat xml-muodossa ja valmiina SUMO:n luettavaksi. Projektissa netedit-työkalua käytettiin netconvert-työkalun generoiman kartan siistimiseen, sillä tietyissä kohdissa netconvertin käyttämä liikenneverkoston generointialgoritmi ei tuottanut risteyksiä tai tieosuuksia sellaisina, kuin ne olivat oikeassa maailmassa.



Kuva 5. Risteyksen muokkausta netedit-ohjelmassa. Tien sivussa oleva vihreä alue on linja-autopysäkki. (13.)

4 Unity-pelimoottori

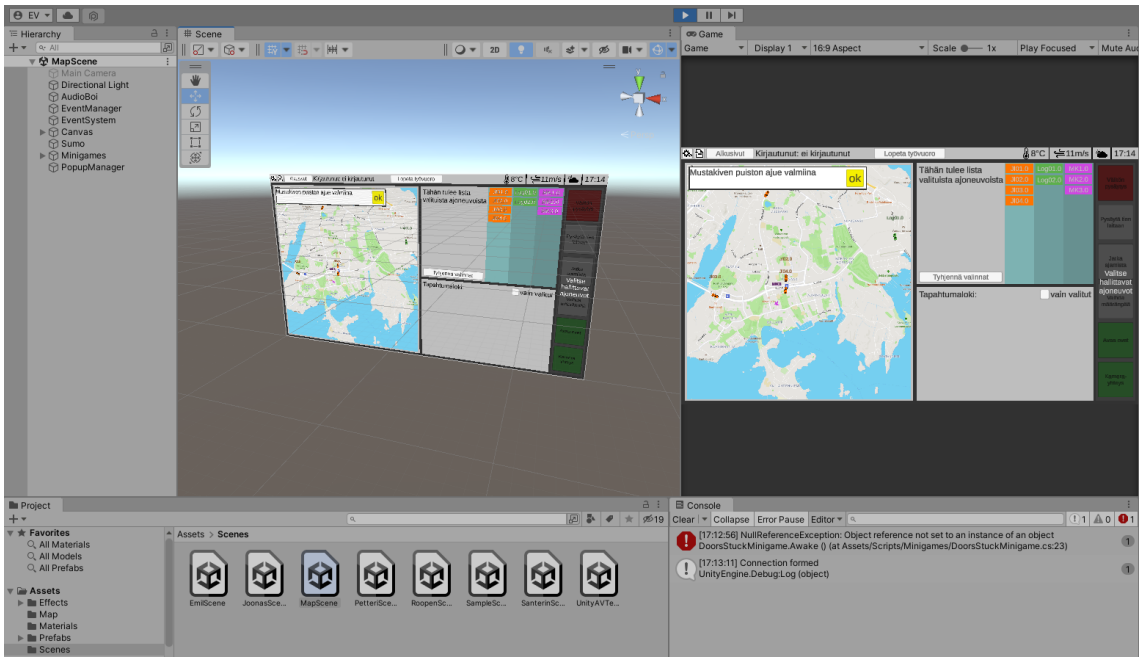
Insinöörityönä tehty sovellus ja sen graafinen käyttöliittymä toteutettiin Unity-pelimoottorin versiolla 2021.2.9f1. Pelimoottori tukee yli 25:tä eri alustaa, mukaan lukien PC-tietokoneita. Moottori on ilmainen ja erityisen suosittu itsenäisten ja mobiilipelikehittäjien keskuudessa (14). Tämä pelimoottori valittiin sovelluksen toteutusta varten, koska projektin jokaisella kehittäjällä oli aikaisempaa kokemusta moottorin käytöstä. Pelimoottorilla kehitetty sovellus oli kaksiulotteinen.

Pelimoottorissa kehitys toimi C#-ohjelmointikielen ja moottorin oman käyttöliittymän avulla.

Pelimoottorin sisällä kehitys toimi niin, että jokaisella kehittäjällä oli oma ”scene” eli näkymä Unityssa, johon muutoksia tehtiin (kuva 6). Unityssa näkymä on erillinen instanssi kehitysympäristöstä, johon kehitystä tehdään. Unityn dokumentaatiossa selitetään, että näkymät sisältävät pelin objektit ja että niihin voi luoda valikon, erillisiä pelitasoja ja mitä tahansa muuta (15).

Pelimoottorin kehitys pohjautuu niin kutsuttuihin peliolioihin (engl. gameobject). Unityn oman dokumentaation mukaan peliolio on pohjaluokka kaikille kokonaisuuksille Unityn näkymissä (16). Peliobjekteihin liitetään komponentteja, joihin voi tallentaa mitä tahansa peleihin liittyvää tietoa tai rakenteita.

Jos kaksi eri kehittäjää on tehnyt muutoksia samaan näkymään ja näitä muutoksia yritetään lisätä samaan aikaan samaan versionhallintahaaraan, syntyy niin sanottu yhdistyskonflikti (engl. merge conflict), jolloin vain yksi joukko kehitettyjä muutoksia voidaan säästää. Tämän ongelman välttämiseksi kaikki kehittäjät tekivät kehitystyötä omiin näkymiinsä, joista ne aina välillä yhdistettiin erilliseen päänäkymään.



Kuva 6. Unityn graafinen käyttöliittymä. Vasemmalla puolella "scene"-näkymä. Alhaalla luettelo eri näkymätiedostoista.

5 Liikenteenvalvontasovelluksen kehitys

5.1 SAM-hanke

Älykäs autonominen liikenne, eli Smart Autonomous Mobility, SAM, on Metropolia Ammattikorkeakoulun hanke, jonka tavoitteena on hankkeen hakemuksen mukaan ”luoda uutta liiketoimintaa sekä kehittää jo olemassa olevaa liiketoimintaa autonomisten ajoneuvojen ja liikenteen valvonnan alalle” (17). Hankkeen tavoitteisiin kuuluu myös kestävän liikenneinfrastruktuurin ja sen suunnittelun edistäminen sekä autonomisten ajoneuvojen ja koneiden käytön saavutettavuuden tarkastelu. Hankeen olennainen osa on tuottaa etävalvomojärjestelmä, jolla voi tehdä valvomotoimintaa testausympäristössä. Tämä testausympäristö on se osa hanketta, jota insinöörityö koski, eli liikenteenvalvontasovellus.

5.2 Työmenetelmät

Projektin teossa jäsenten tekemää työtä dokumentoitiin laajasti. Microsoftin Teams-ohjelmaan tallennettiin viikoittain jokaisen jäsenen tehtävä kullakin viikolla. Lisäksi projektista pidettiin yllä pelisuunnitteludokumenttia, joka sisälsi kaikki valmiin projektin ominaisuudet sekä selitteet niiden toiminnasta.

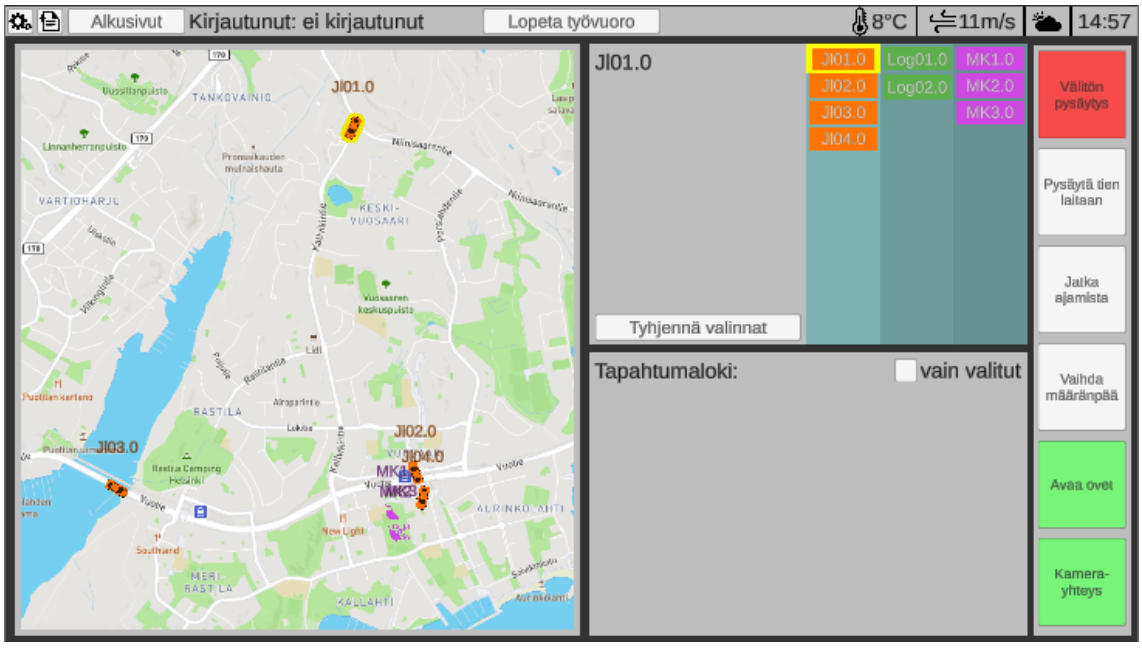
Projektin kehityksessä käytettiin GitHub-versionhallintatyökalua. Työkalun avulla kaikilla projektin kehittäjillä oli sama versio projektista. Työkalu myös mahdollisti projektin jakamisen eri haaroihin. Pääasiassa uudet lisäykset vietiin kehityshaaraan, josta ne vietiin edelleen päähaaraan. Tämä varmisti sovelluksen toimivuuden siinä tapauksessa, että jokin kehityshaaraan tehty muutos olisi aiheuttanut virheen tai ongelman versionhallintatyökalussa tai kehitettävässä sovelluksessa.

Projektin työskentely aloitettiin perehtymällä liikennesimulaattorin toimintaan ja käyttämiseen. SUMO:n sekä neteditin käyttöä opeteltiin SUMO:n verkkosivuilta löytyvillä opetustehtävillä. Samalla TraCI:n käyttäminen selkeytyi, kun tehtävien tietyt osa-alueet vaativat omaa Python-skriptiä, jossa käytettiin TraCI:n komentoja. Opetustehtävien aiheena oli muun muassa verkostojen luonti manuaalisesti netedit-ohjelmaa käyttämällä tai OSMWebWizardin avulla automaattisesti luotuna. Osa opetustehtävistä keskittyi tarkempiin aiheisiin, kuten liikennevalojen ohjaus tai julkisen liikenteen luominen.

Jotta SUMO toimisi oikein Unity-pelimoottorin kanssa, piti molempien ohjelmien käyttää samaa ohjelmointikieltä. SUMO:n TraCI:sta löydettiin avoimen lähdekoodin C#-pohjainen toteutus (18), joka integroitiin Unityyn jatkokehitystä varten. Uuden ohjelmointikielen ja alustan takia TraCI:n komentoja piti opetella uudestaan.

5.3 Käyttöliittymä

Pelimoottorilla toteutettu liikenteenvalvontasovellus ja sen graafinen käyttöliittymä (kuva 7) koostuivat monesta osasta. Käyttöliittymän ensisijaisin osa oli itse karttanäkymä, joka kuvasi reaaliaikaisesti kaikkien ajoneuvojen liikettä liikenteessä ja jossa näkyi myös eri ajoneuvojen tyypit. Simulaatiossa aktiivisina olevista ajoneuvoista oli myös lista, josta ajoneuvon nimeä painamalla sai valittua ajoneuvon, jotta sille sai annettua eri komentoja.



Kuva 7. Liikenteenvalvontasovelluksen graafinen käyttöliittymä.

Kun sovelluksen käynnistää, se vie käyttäjän sisäänkirjautumissivulle. Sovellukseen kirjaudutaan käyttäjätunnuksella ja salasanalla, jotka syötetään niille tarkoitettuihin kenttiin. Kirjautumissivun jälkeen käyttäjälle näytetään lista kaikista ajoneuvoista, jotka ovat aktiivisina simulaatiossa sen ajamisen ajan (kuva 8). Ajoneuvot on jaettu kolmeen eri ryhmään, kuten simulaation ajon aikaisessa graafisessa käyttöliittymässä. Ryhmät ovat julkisen liikenteen, logistiikan ja huoltoreittien ryhmiin kuuluvat ajoneuvot. Ajoneuvolistan jälkeen käyttäjälle annetaan sääennustus sekä tietoa simulaation aikana vallitsevista sääolosuhteista. Niihin kuuluvat lämpötila, sateen mahdollisuus, ilmankosteus ja tuulen

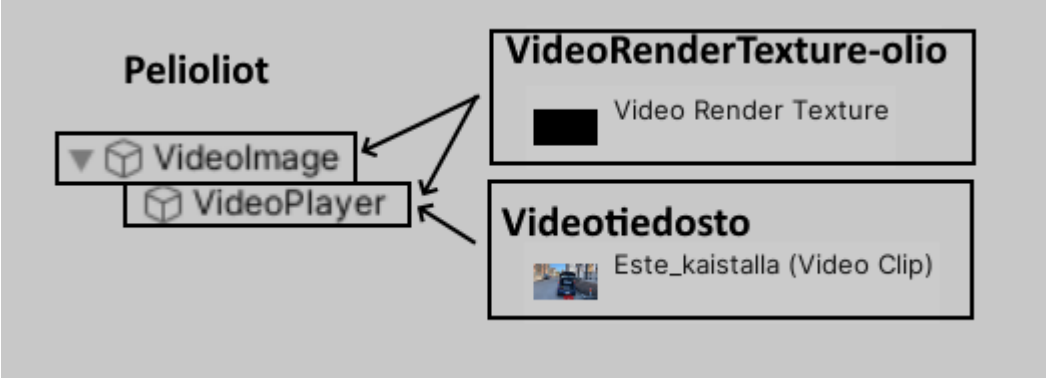
nopeus. Ennen varsinaiseen simulaatioon pääsemistä käyttäjälle näytetään vielä edellisen käyttäjän työvuoron loki, johon edellinen käyttäjä on kirjannut tärkeitä tapahtumia tai vikoja ajoneuvoissa. Suurin osa tästä käyttäjälle annettavasta informaatiosta oli insinöörityötä kirjoitettaessa vain esimerkkinä, ja se tullee korvatuksi oikealla tiedolla, joka haetaan erilaisista rajapinnoista projektin edetessä.

Kuva 8. Kirjautumissivun jälkeen näkyvä lista työvuoron aikana olevista seurattavista ajoneuvoista. Ajoneuvot on luokiteltu kolmeen eri ryhmään.

Sovelluksen käyttöliittymä sisältää painikkeita, joilla annetaan komentoja ajoneuvoille. Painikkeet ovat nimiltään ”välitön pysäytys”, ”pysäytä tien laitaan”, ”jatka ajamista”, ”vaihda määränpää”, ”avaa ovet” ja ”sulje ovet” sekä ”kamerayhteys”. Painikkeet eivät vaikuta mihinkään, ellei käyttäjä ole valinnut ongelmatilanteessa olevaa ajoneuvoa, jolle painikkeiden komennot annettaisiin. Ongelmatilanteissa on tietty kokoelma painikkeita, joita käyttäjän täytyy painaa ratkaistakseen ongelmatilanteen.

Kamerayhteyspainikkeen painaminen eroaa muista painikkeista siten, että se avaa myös kamerayhteyspaneelin käyttöliittymässä. Paneeli toteutettiin pelimoottorin VideoPlayer-komponentilla, johon voi liittää videotiedoston, jota

komponentti toistaa. Itse videontoistopeliolio, jolla pelimoottorissa sai videot toistumaan, koostui kahdesta eri pelioliosta (kuva 9). Yhteen peliolioon liitetään kuvaolio (VideoImage), jonka texture-muuttujan arvoksi asetetaan VideoRenderTexture-olio. VideoRenderTexture on kaksiulotteinen tekstuuriolio, jonka avulla voi projisoida videokuvaa. Toinen peliolio, joka on asetettu ensimmäisen olion lapsiolioksi, sisältää varsinaisen VideoPlayer-komponentin. Tämä komponentti sisältää Target Texture -nimisen muuttujan, johon VideoRenderTexture asetetaan. Lopuksi VideoPlayer-komponentille asetetaan videotiedosto, joka halutaan toistaa. Jokaiselle ongelmatilanteelle on omat videotiedostonsa, jotka kuvaavat käyttäjälle tarkemmin, mikä varsinainen ongelma on kussakin ongelmatilanteessa. Alun perin kaikki videotiedostot, joita videokomponentti käytti, olivat sisällytettynä sovelluksen tiedostorakenteessa, mutta myöhemmin toiminnallisuus tulisi toteuttaa antamalla komponentille linkki videotiedostoon, joka itsessään olisi ladattu pilveen.



Kuva 9. Videontoistotoiminnallisuuden rakenne. Viittaus VideoRenderTexture-olioon annetaan molemmille peliolioille.

Kun uusi ongelmatilanne alkaa, siitä annetaan huomautus käyttäjälle niin kutsutulla pop-up-viestillä. Viesti ilmestyy ruudun vasempaan yläreunaan, ja se sisältää lyhyen selityksen ongelmatilanteesta sekä painikkeen, josta viestin voi poistaa. Käyttöliittymä sisältää myös tapahtumalokin, johon ilmestyy tärkeää tietoa simulaation kulusta, muun muassa ajoneuvojen tilanteista ongelmatilanteissa. Tapahtumalokin voi asettaa suodattamaan viestejä vain valitulle ajoneuvolle.

5.4 Komennot

Avoimen lähdekoodin C#-pohjainen TraCI-toteutus oli vajaa, sillä se ei sisältänyt kaikkia komentoja alkuperäisestä TraCI:sta. Suuria ongelmia komentojen puute ei aiheuttanut, mutta tietyissä tilanteissa piti löytää oma toteutus ongelmalle, jota ei olisi ollut, jos rajapinta olisi sisältänyt kaikki alkuperäiset komennot. Eräs vakavimmista puutteista oli komento, jolla sai selvitettyä tietyn ajoneuvon pysähdystilan eli sen, onko ajoneuvo pysähtynyt vai ei. Yleensä tämän ongelman sai kierrettyä hakemalla ajoneuvon nopeuden. Jos nopeus oli 0, oli ajoneuvo sillä hetkellä pysähtynyt.

TraCI:n komentoja käytettäessä ohjelmakoodissa on aina verrattava ”TraCIClient”-nimisen luokan olioon, jolle komennot annetaan. TraCIClient-luokka kuvaa TraCI:n instanssia ja niitä on pääohjelmassa aina vain yksi. Tietyt ohjelman metodit vaativat viittausta tähän TraCI-instanssiin toimiakseen. TraCIClient-luokan oliota kutsutaan koko ohjelmassa nimellä ”client”, kuten esimerkkikoodissa 1 näytetään.

```
TraCIClient client = SumoConnect.Instance.Client;
string veh = GetRandomVeh(”pt_bus”);
string edge = ”23278350”;
client.Vehicle.ChangeTarget(veh, edge);
```

Esimerkkikoodi 1. Satunnaisen ”pt_bus”-tyyppisen simulaatiossa olevan ajoneuvon kohteeksi asetetaan uusi tieosuus. TraCI:n ”ChangeTarget”-metodi laskee ajoneuvolle automaattisesti uuden reitin.

Tärkeitä ominaisuuksia, joita ongelmatilanteissa tarvittiin, olivat muun muassa ajoneuvon pysäytys, uudelleenreititys, pysäytys tien sivuun sekä tietyn tieosuuden käytön estäminen. Ajoneuvon pysäytys oli yksinkertainen nopeusmuuttujan arvon asettaminen 0:ksi. Tällöin ajoneuvo hidasti asetettuun nopeusarvoon, eli tässä tilanteessa kokonaan pysähtyneeksi. Pysäytys tien sivuun, josta muu liikenne voisi ohittaa ajoneuvon, toimi asettamalla ajoneuvolle uusi pysähdyskohta. Uuden pysähdyskohdan parametreiksi annetaan pysähdyskohdan paikka, aika, pysähdyksen kesto ja tyyppi. Pysähdystyyppejä ovat esimerkiksi linja-autopysäkki tai pysäköinti. Tässä tilanteessa pysäköinti asettaa ajoneuvon tien sivuun pois muun liikenteen tieltä. Uudelleenreititykselle oli hyvin

yksinkertainen komento, joka laskee valitulle ajoneuvolle uuden reitin ottamalla nopeimman matkustusajan tieosuudet ajoneuvon päämäärän huomioon ottaen. Nopein matkustusaika lasketaan simulaatiossa kaavalla tieosuuden pituus jaettuna tieosuuden keskiarvonopeudella.

Tietyt ongelmatilanteiden toteutukset vaativat yhden tieosuuden käytön estämistä liikenteeltä. Tämä toteutettiin asettamalla uusi ns. esteajoneuvo tieosuudelle sekä asettamalla tieosuuden nopeusrajoituksen arvoksi 0. Tällöin ajoneuvot, joiden reitillä tie oli, pysähtyivät ennen saapumista tieosuudelle, josta ne suurimmassa osassa tapauksista pystyi uudelleenreitittämään.

5.5 Liikenteen ongelmatilanteet

Seuraava vaihe projektissa oli kehittää ongelmatilanteet, joilla simuloitiin erilaisia poikkeustilanteita, joita oikean elämän liikenteessä saattaisi ilmaantua, muun muassa erilaisia tieosuuksien tukkeutumisia tai teknisiä vikoja ajoneuvoissa. Ensin ongelmatilanteet kehitettiin pelkkään SUMO-simulaatioon, josta ne jälkeenpäin siirrettiin pelimoottoriin. Tämä prosessi vaati tiettyjen ongelmatilanteiden kohdalla ohjelmakoodin uudelleenkirjoitusta, sillä tietyt toteutustavat, jotka toimivat liikennesimulaattorissa itsessään, eivät toimineet pelimoottoriin mukautetussa liikennesimulaattorin toteutuksessa.

Valmiissa sovelluksessa ideana oli, että koko simulaation keston aikana on tiettyjä hetkiä, jolloin ongelmatilanteita aiheutetaan käyttäjälle ohjelmallisesti. Simulaation alussa ja lopussa ongelmatilanteita ilmenee vähän ja ne ovat yksinkertaisempia, kun taas simulaation keskellä ongelmatilanteita ilmenee runsaammin ja ne ovat monimutkaisempia. Käyttäjän kokema rasitus on siis voimakkain puolessavälissä simulaation kestoa. Simulaation suunniteltu kesto oli 1,5 tuntia. Kehitysvaiheessa ongelmatilanteiden aloittaminen toimi kuitenkin painamalla tiettyjä näppäimiä näppäimistöllä testaamisen ja vianetsinnän helpottamiseksi.

5.5.1 Tehtävienhallinnointi

Sovelluksen toteutuksessa käytettiin apuna erillistä tehtävänhallintaluokkaa (sovelluksessa nimellä ”TaskManager”), joka hallinnoi sitä, miten käyttäjän tuli ratkaista eri tehtävät sovelluksessa. Tehtävänhallintaluokan tueksi kehitetyt task- ja subtask-luokat helpottivat ongelmatilanteiden ratkaisun toteuttamista käyttöliittymää varten. Subtask-olio joko vertaa johonkin käyttöliittymän painikkeeseen tai sille annetaan odotusaika. Task-olioon voi lisätä subtask-olioita haluttuun järjestykseen, kuten esimerkkikoodissa 2 tehdään. Kun käyttäjä suorittaa yhden subtask-tehtävän, se poistetaan task-olion subtask-listasta. Kun task-olion subtask-lista on tyhjä, tehtävä on suoritettu. Yleensä ongelmatilanne on tämän jälkeen ratkaistu.

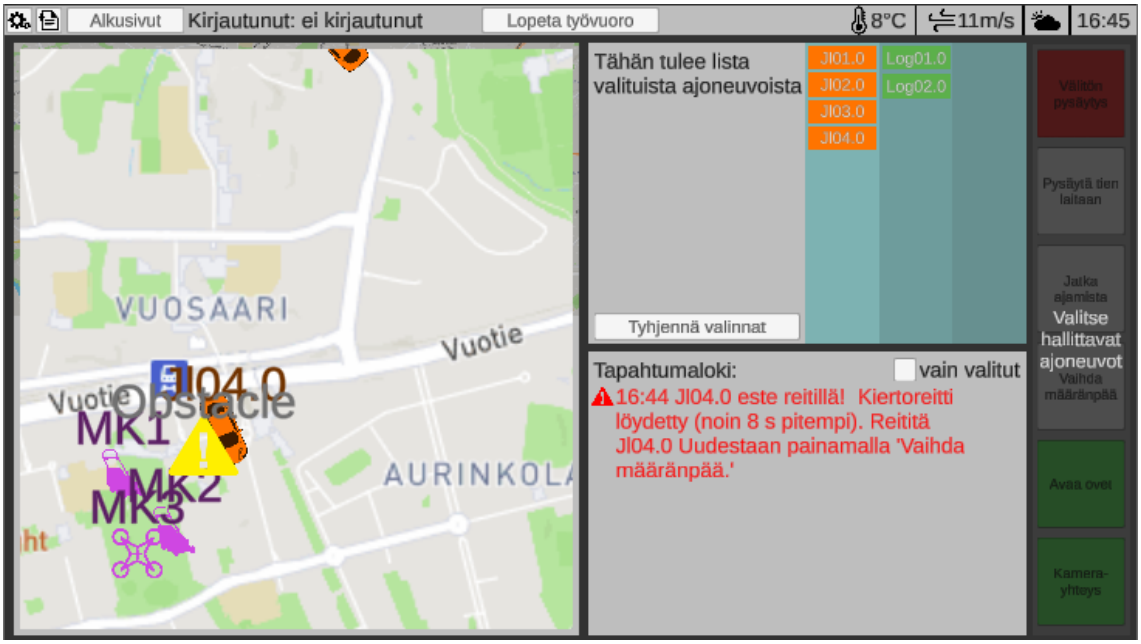
```
ButtonSubtask ct = new ButtonSubtask(taskManager.cameraButton);
ButtonSubtask bt = new ButtonSubtask(taskManager.changeRouteButton);
WaitSubtask wt = new WaitSubtask(1);
Subtask[] s = { ct, bt, wt };
taskManager.IssueTask(veh, new Task(s, 1));
```

Esimerkkikoodi 2. Task-olion luominen taskmanager-luokkaan este reitillä-ongelmatilanteen koodissa. Task-olion subtask-taulukolle asetetaan kolme subtask-oliota.

Jokaisen ongelmatilanteen kooditiedosto periytyy pääongelmatilannekooditiedostosta nimeltään ”Minigames.cs”. Tässä tiedostossa määritellään viittaukset tärkeisiin sovellusta hallitseviin luokkiin jokaisen ongelmatilanteen kooditiedostoa varten. Näitä luokkia ovat ”EventManager”, jonka avulla voi kutsua erilaisia metodeja lokiviesteihin liittyen, ”AudioHandler”, joka hallinnoi sovelluksen ääniefektejä ja niiden aktivointia, ”Canvas” (yleisnimitys Unityssa kaksiulotteisten käyttöliittymien pohjaobjekteille), jonka kautta voi kutsua erilaisia käyttöliittymään liittyviä metodeja, ”SelectionManager”, joka pitää jatkuvasti listaa valituista ajoneuvoista sekä ”TaskManager”, joka hallinnoi aktiivisia Task-olioita. Minigames-skriptissä määritellään myös tukimetodeja, joita monen ongelmatilanteen koodi tarvitsee, kuten satunnaisen ajoneuvon valintaa tai esteolion luomista kartalle.

5.5.2 Este reitillä

Ajallisesti eniten kehitysaikaa kului niin sanotun ”Este reitillä” -ongelmatilanteen kehittämiseen (kuva 10). Ongelmatilanteen päämäärä oli saattaa kokonainen tieosuus käyttökelvottomaksi liikenteelle koko matkalta niin, että sitä käyttävät ajoneuvot joutuisivat kiertämään tieosuuden. Ongelmatilanteessa oli siis kaksi osaa: tieosuuden tukkiminen ja sitä käyttävien ajoneuvojen reittien uudelleenlaskeminen. Ongelmatilanne oli käyttäjän kokeman rasituksen kannalta raskain ja myös ohjelmallisesti laajin kehitetty ongelmatilanne.



Kuva 10. Este reitillä -ongelmatilanne. Ajoneuvo Jl04.0 on pysähtynyt keltaisella varoituskuvalla ja "Obstacle"-tekstillä merkityn tukitun tieosuuden eteen. Kuvan purppurat ”MK”-ajoneuvot kuvaavat ruohonleikkureita ja lennokkia.

Tieosuuden tukkimista varten täytyi ensin määritellä tieosuus, joka tukitaan. Valitseminen toimi alun perin antamalla ohjelmalle ajoneuvo, jonka reitti tuli tukkia. Valitun ajoneuvon reitiltä haettiin ajoneuvon seuraavat tieosuudet, joille ajoneuvo ajaisi. Näistä valittiin sopiva tukittava tieosuus seuraavin määritelmin: Tieosuuden matkustusaika-muuttujan arvon täytyy olla vähintään kaksi, eli tieosuuden on oltava tarpeeksi pitkä. Jos tieosuus olisi liian lyhyt, se voisi johtaa tieosuuden valitsemisessa tilanteeseen, jossa valittu tieosuus olisi esimerkiksi

pieni pala risteykseen johtavaa kaistaa, jossa ei olisi tarpeeksi tilaa esteajoneuvoille. Tieosuudella ei pidä myöskään olla muita ajoneuvoja, eikä se voi olla valitun ajoneuvon reitin heti seuraava tieosuus. Jos tieosuudella olisi muita ajoneuvoja, se vaikeuttaisi esteajoneuvojen asettamista tai voisi johtaa vaikeampiin ongelmiin. Valittu tieosuus ei voi olla ajoneuvon reitin seuraava, koska ajoneuvolla ei välttämättä olisi tarpeeksi aikaa pysähtyä juuri ilmestyneiden esteajoneuvojen eteen.

Esimerkkikoodissa 3 näytetään kyseinen estetieosuuden hakemisprosessi. Esimerkissä käydään foreach-silmukan avulla läpi lista ”obstEdges”, joka sisältää kaikki valitun ajoneuvon reitin tieosuudet. Esimerkin if-lausekkeessa tarkistetaan jokaisen tieosuuden (edge) kohdalla järjestyksessä, että sen matkustusaika on enemmän kuin 2 sekuntia, että tieosuudella ei ollut yhtäkään ajoneuvoa viimeisimmällä aika-askeleella ja että tieosuus ei ole ajoneuvon seuraava. Jos kaikki ehdot toteutuvat, asetetaan tieosuus valituksi estetieosuudeksi (ohjelmassa nimellä ”bestEdgeID”). Tämän jälkeen silmukan läpikäynti lopetetaan.

```
foreach (var edge in obstEdges)
{
    if (client.Edge.GetTraveltime(edge).Content > 2.0 &&
    client.Edge.GetLastStepVehicleNumber(edge).Content == 0 &&
    client.Vehicle.GetRoute(vehID).Content.IndexOf(edge) !=
    client.Vehicle.GetRoute(vehID).Content.IndexOf(
    client.Vehicle.GetRoadID(vehID).Content + 1))
    {
    bestEdgeID = edge;
    break;
    }
}
```

Esimerkkikoodi 3. Sopivan estetieosuuden hakeminen.

Kun ajoneuvon reitiltä on löytynyt määritelmät täyttävä tieosuus, tarkistetaan tieosuuden kaistojen määrä. Jokaiselle kaistalle lisätään uusi ajoneuvo, jonka nopeudeksi asetetaan 0. Nämä uudet ajoneuvot toimivat esteinä muille tieosuutta käyttäville ajoneuvoille. Valitun tieosuuden nopeusrajoitukseksi täytyy myös asettaa 0, koska silloin simulaatio antaa muille ajoneuvoille etukäteen tiedon tieosuuden käyttökelvottomuudesta, jolloin sitä käyttävät ajoneuvot pysähtyvät ennen saapumistaan tieosuudelle. Tämä antaa mahdollisuuden reitittää ajoneuvot

uudelleen ilman ongelmia. Projektin myöhemmässä kehitysvaiheessa tukittava tieosuus oli ennalta määritelty ohjelmakoodissa, koska tukkimiseen soveltuvia tieosuuksia oli liian vähän simulaatioon valitussa kartassa. Tällä toimenpiteellä ennaltaehkäistiin monta pientä ongelmaa ja varmistettiin mahdollisuus jatkokehittämiselle. Sovelluksen graafisessa käyttöliittymässä estettä kuvaa keltainen varoituskolmiografiikka, jonka yläpuolella lukee ”Obstacle” eli este englanniksi. Este luodaan Minigames-luokasta perityn ”CreateObstacle”-metodin avulla, jota kuvataan esimerkkikoodi 4:ssä. Metodissa luodaan uusi Unity-ohjelman käyttämä peliolio (GameObject), joka asetetaan sovelluksessa olevalle kartalle. Peliolio on keltainen varoituskolmiokuvake. Uuden peliolion sijainti kartalla saadaan hakemalla esteajoneuvon sijainti simulaatiosta ja muuntamalla se Unityn karttaan sopivaksi.

```
public GameObject CreateObstacle(string obstVeh_)
{
    GameObject obst = Instantiate(obstacleObj,
    SumoConnect.Instance.map.MapTransform);

    obst.transform.localPosition = SumoMap.SumoToUnityPos(
    SumoConnect.Instance.Client.Vehicle.GetPosition3D(
    obstVeh_).Content);

    return obst;
}
```

Esimerkkikoodi 4. Esteobjektin luominen simulaatioon.

Ajoneuvon pysähtyessä tukitun tieosuuden eteen pystyi sen uudelleenreitittämään yhdelle korkeintaan kolmesta uudesta reittivaihtoehdosta. Tätä varten täytyi kehittää aliohjelma, joka laskee ajoneuvolle uuden reitin kolme kertaa, tai niin monta kertaa kuin mahdollista, niin että jokainen reitti kiertää tukitun tieosuuden eri suunnasta. Tämä toiminnallisuus saavutettiin niin, että aina, kun yksi uusi reitti oli laskettu käyttämällä TraCI:n uudelleenreitityskomentoa, asetettiin reitin ensimmäisten tieosuuksien nopeusrajoitus väliaikaisesti nollaksi. Näin uudelleenreitityskomentoa kutsuessa uudestaan sen polunetsintäalgoritmi joutui laskemaan uuden reitin käyttämättä aikaisempia tieosuuksia, koska niiden kautta kulkiessa olisi kulunut ääretön määrä aikaa. Koska algoritmi etsii aina nopeimman reitin kulkea, algoritmi ei käyttänyt näitä tieosuuksia. Kun kaikki

reittivaihtoehdot oli laskettu, käyttäjä sai valita niistä yhden, joka asetettiin ajoneuvolle.

Lopullisessa ohjelmassa ei reittivaihtoehtoja kuitenkaan ollut, koska graafinen pelimoottoriin rakennettu käyttöliittymä tuki heikosti eri reittivaihtoehtojen näyttämistä käyttäjälle. Myös tietyn reitin valitseminen käyttäjän toimesta olisi ollut haasteellista toteuttaa käyttöliittymässä. Näin ollen lopullisessa ongelmatilanteessa ajoneuvo reititetään uudelleen ensimmäiselle uudelleenreitityskomennon palauttamalle kiertoreitille painamalla painiketta käyttöliittymässä.

Kun ajoneuvolle on annettu uusi reitti, ohjelma tarkistaa, onko kyseinen ajoneuvo linja-auto. Jos on, ohjelma tallentaa kiertoreitin ja antaa sen seuraaville samaa linjaa ajaville linja-autoille, jos ne saapuvat tukitun tieosuuden eteen ennen ongelmatilanteen päättymistä. Tämän ansiosta käyttäjän ei tarvitse turhaan reitittää saman linjan linja-autoja uudelleen.

Kun mikä tahansa ajoneuvo saapuu estyneelle tieosuudelle, aloitetaan ongelmatilanteen ohjelmakoodissa uusi task-olio, jolle annetaan subtaskeiksi kamerayhteyspainikkeen painaminen, määränpäänvaihtopainikkeen painaminen ja lopuksi sekunnin mittainen odotus. Tämän jälkeen task-olion subtaskit on suoritettu ja ajoneuvo jatkaa matkaansa uudella kiertoreitillä.

Jokaisessa ongelmatilanteessa vaaditaan kamerayhteyspainikkeen painamista, koska se kuuluu käyttäjälle vaadittuun käytäntöön. Kamerayhteyspainikkeen piti olla vaihtoehtoista, mutta ohjelma ei sallinut sitä, sillä jotta painiketta voisi painaa, siihen pitää liittyä aktiivinen subtask-olio. Muuten painikkeen painaminen ei tee mitään. Tämän takia lisättiin jokaisen ongelmatilanteen subtask-listaan kamerayhteyspainikkeen painaminen, jotta kamerayhteys-toimintoa voitiin ylipäätään käyttää.

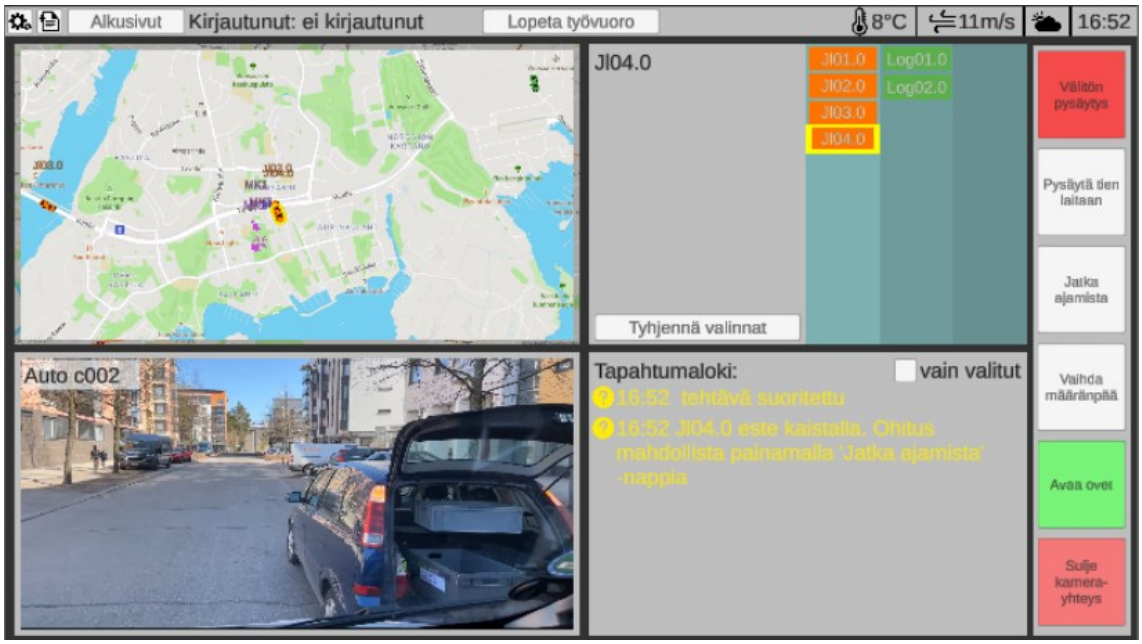
Kun ongelmatilanteen alkamisesta on kulunut ennalta määritelty aika, ongelmatilanne puretaan ja tilanne palautetaan tavalliseksi. Tämä tarkoittaa sitä, että tukitun tieosuuden nopeusrajoitus palautetaan siihen, mikä se oli ennen ongelmatilanteen alkua. Myös ongelmatilanteen alussa tieosuudelle asetetut uudet

esteajoneuvot asetetaan ajamaan niin sanotulle varikolle, eli ennalta määritetylle tieosuudelle, johon päästyään ne poistetaan simulaatiosta.

5.5.3 Este kaistalla

Vaikka este kaistalla -ongelmatilanne nimensä mukaan kuulostaa hyvin samanlaiselta kuin este reitillä -ongelmatilanne, niiden erot ovat silti suuret. Este kaistalla -ongelmatilanteessa (kuva 11) yhden ajoneuvon ajamalla kaistalla havaitaan ajoneuvon edessä este, joka on toinen pysähtynyt ajoneuvo. Käyttäjällä on kaksi vaihtoehtoa: joko odottaa, että pysähtynyt ajoneuvo jatkaa matkaa, tai ohittaa pysähtynyt ajoneuvo. Yksikaistaisella tiellä vastaantulevan liikenteen kaistaa voi myös käyttää ohittamiseen, jos kaista on tyhjä ohitushetkellä.

Pelimoottoriin toteutetussa ongelmatilanteessa valitaan siis kohdeajoneuvo samalla tavalla kuin este reitillä -ongelmatilanteessa. Ongelmatilanne kohdistetaan tähän ajoneuvoon asettamalla sen eteen kohdeajoneuvon samalle tai seuraavalle tieosuudelle pysähtynyt ajoneuvo. Estynyttä kaistaa kuvastaa estegrafiikka, joka ilmestyy kartalle. Koska tieosuuden nopeusrajoitusta ei aseteta nollaksi kuten este reitillä -ongelmatilanteessa, ajoneuvo ei pysähdy ennen tukittua tieosuutta. Ajoneuvo ajaa niin pitkälle kuin mahdollista, eli tässä tapauksessa tien tukkivan ajoneuvon perään, johon se pysähtyy. Tämän jälkeen käyttäjälle annetaan ilmoitus esteestä ajoneuvon kaistalla. Käyttäjä voi olla tekemättä mitään ja odottaa tilanteen purkautumista tai painaa käyttöliittymässä olevaa ”Jatka ajamista” -painiketta, mikä yksinkertaisesti poistaa esteajoneuvon ja käyttöliittymässä näkyvän estegrafiikan.



Kuva 11. Este kaistalla -ongelmatilanne, jossa kamerayhteys on avattu. Ruudun vasemman alalaidan video näyttää esteen ohitustilanteen.

SUMO:ssa on olemassa oleva ohitustoiminnallisuus myös vastaantulevien kaistaa käyttäen, mutta tämän toteuttaminen olisi vaatinut karttatiedoston kaikkien tieosuuksien manuaalista muokkausta niin, että jokaiselle tieosuudelle asetetaan uusi muuttujan arvo, joka sallii vastaantulevan liikenteen kaistan käyttämisen ohitustilanteessa. Tätä muokkausta ei kuitenkaan tarvinnut toteuttaa, sillä ohituksen kuvaaminen käyttäjälle toimii yksinkertaisesti poistamalla estegrafiikka kartasta. Varsinaista ohittamista ei tarvitse kuvata graafisesti, koska esteajoneuvoa ei näy kartalla ollenkaan.

Este kaistalla -ongelmatilanteen subtask-lista on lähestulkoon identtinen Este reitillä -ongelmatilanteen kanssa. Lista eroaa siten, että määränpäänvaihtopainike on korvattu ajonjatkamispainikkeella. Kun ajoneuvo on ohittanut esteen, este poistuu, eli ongelmatilanne ei ole samalla tavalla pitkäkestoinen kuin este reitillä. Valmiissa sovelluksessa voisi ilmetä siis monta este kaistalla -ongelmatilannetta samaan aikaan, eikä se olisi käyttäjälle liian kuormittavaa.

5.5.4 Ovien sulkemisessa viivettä

Ovien sulkemisessa viivettä kuvaa sellaista ongelmatilannetta, jossa yksittäinen ajoneuvo on pysähtynyt ja avannut ovensa. Ovet eivät kuitenkaan sulkeudu normaalisti ja käyttäjän on pakko manuaalisesti korjata ongelma. Ongelmatilanteesta tulee käyttäjälle ilmoitus, minkä jälkeen käyttäjän täytyy ensin avata kamerayhteys ajoneuvoon painamalla kamerayhteyspainiketta. Tästä käyttäjä näkisi ovien olevan auki tai jumissa. Ongelmatilanne ratkeaa painamalla ”avaa ovet” -painiketta, odottamalla sekunnin ja tämän jälkeen painamalla ”sulje ovet” -painiketta. Ajoneuvo jatkaa ajamista normaalisti reitillään ja ongelmatilanne on ratkaistu.

Ongelmatilanteen subtask-oliot ovat ratkaisujärjestyksessä kamerayhteyspainikkeen painaminen, ovien avauspainikkeen painaminen, sekunnin odottaminen ja lopuksi ovien sulkupainikkeen painaminen.

Ongelmatilanteen toteutus alkoi suoraan pelimoottorin kehitysympäristössä, eli sitä ei tehty ensin pelkkään riippumattomaan liikennesimulaattoriin, kuten moni muu ongelmatilanne. Ohjelmallisesti ongelmatilanne ”aiheutetaan” valitsemalla satunnainen ajoneuvo. Ongelmatilanne alkaa vasta, kun auto pysähtyy seuraavan kerran, esimerkiksi linja-autopysäkille tai risteykseen.

Satunnaisen ajoneuvon simulaatiosta valitsemista varten täytyi kehittää oma metodi. Metodi toimi hakemalla ensin kaikki simulaatiossa aktiivisena olevat ajoneuvot, minkä jälkeen niistä poimitaan kaikki halutun tyypin ajoneuvot, esimerkiksi linja-autot. Tämän jälkeen listasta valitaan yksi ajoneuvo käyttämällä satunnaisfunktiota. Metodi on kuvattu esimerkkikoodissa 5. Metodissa otetaan yhteen listaan ensin talteen kaikki määritellyn tyypin ajoneuvot, jotka ovat aktiivisina simulaatiossa. Tämän jälkeen käytetään Unityn MonoBehaviour-luokkaan kuuluvaa satunnaistamisfunktiota ”Random.Range”, joka arpoo satunnaisen kokonaisluvun määriteltyjen lukujen välillä. Satunnaista lukua käytetään ajoneuvolistasta yhden ajoneuvon poimimiseen satunnaista lukua käyttäen. Jos määritellyn tyypin ajoneuvoja ei ole simulaatiossa aktiivisena, metodi palauttaa null-arvon.

```
public string GetRandomVeh(string vehType)
{
    TraCIClient client = SumoConnect.Instance.Client;
    List<string> allVehs = client.Vehicle.GetIdList().Content;
    List<string> vehs = new List<string>();
    foreach (var veh in allVehs)
    {
        if (client.Vehicle.GetTypeID(veh).Content == vehType)
        {
            vehs.Add(veh);
        }
    }
    if (vehs.Count == 0)
    {
        return null;
    }
    string randVeh = vehs[Random.Range(0, vehs.Count)];
    return randVeh;
}
```

Esimerkkikoodi 5. Satunnaisen ajoneuvon valitseminen.

5.5.5 Ajoneuvon lisäys

Osaa suunnitelluista ongelmatilanteista ei ehditty aikarajan sisällä kehittää, mutta hankkeen jatkuessa nekin on tarkoitus myöhemmin lisätä sovellukseen.

Yksi näistä ongelmatilanteista oli ajoneuvon lisäys -niminen ongelmatilanne. Vaikka ongelmatilanne luokittuikin samaan ryhmään aliohjelmia muiden ongelmatilanteiden kanssa, se ei sisältänyt varsinaista ongelmatilannetta. Tehtävän tarkoitus oli kertoa käyttäjälle siitä, että uusi ajoneuvo on ilmestymässä simulaatioon. Käyttäjä painaa painiketta viestistä, joka ilmoittaa uuden ajoneuvon lisäyksen, ja simulaatio jatkuu tavallisesti.

Ongelmatilanteessa seurattaisiin SUMO:n simulaatiota, ja kun uuden ajoneuvon lisäys havaitaan, annettaisiin käyttäjälle ”popup”-viesti uudesta ajoneuvosta. Käyttäjän tarvitsee vain painaa painiketta viestistä, eikä tilanne vaadi sen enempää syötettä käyttäjältä.

Ongelmatilanne olisi vaatinut jonkin tavan seurata, milloin SUMO-simulaatioon lisätään uusi ajoneuvo. Tällaisen toiminnallisuuden toteuttaminen ei ollut yksinkertaista, sillä itse liikenteenvalvontasovelluksessa, kun yhden linjan linja-auto saapuu päätepysäkilleen, sen nimen indeksi vain vaihtuu. SUMO:ssa kuitenkin

linja-auton saavuttuaan päätepysäkille se poistetaan simulaatiosta ja sen tilalle asetetaan uusi samantyyppinen linja-auto. Sovelluksessa siis ajoneuvon nimi vain vaihtuu, vaikka simulaatiossa koko ajoneuvo vaihtuu. Tämän ongelmatilanteen kehittäminen ei siis ollut etusijalla, koska sovelluksessa kehityksen aikana kuvattiin vain linja-autoja eikä esimerkiksi myös henkilöautoja, joille tämä toiminnallisuus olisi vaadittu.

5.5.6 Ajoneuvon käytöstä poisto

Ajoneuvon käytöstä poisto -ongelmatilanteen skenaario oli, että ajoneuvossa ilmenee tekninen vika, esimerkiksi renkaan puhkeaminen. Käyttäjä ohjaa ajoneuvon tien sivuun painamalla ”pysäytä tien laitaan” -painiketta. Kun ajoneuvo on pysähtynyt tien sivuun, sen voi ohjata varikolle painamalla ”jatka ajamista” -painiketta. Tämän jälkeen ajoneuvo ajaa ennalta määritellylle varikkotieosuudelle ja poistetaan simulaatiosta sen saavuttua päämääräänsä.

Ongelmatilanne käytti epätavanomaisesti kahta eri task-oliota eri kohdissa ongelmatilanteen aliohjelman koodia. Ensimmäisen task-olion subtask-oliot olivat kamerayhteyspainike, tienlaitaanpysäytyspainike ja sekunnin odotus. Ohjelma valvoi, milloin nämä subtask-oliot olivat suoritettu niiden task-oliossa, minkä jälkeen kutsutaan ajoneuvon pysäytysmetodia, joka pysäyttää ajoneuvon tien sivuun ja asettaa seuraavan task-olion. Toisen task-olion subtask-oliot olivat ajonjatkamispainike ja sekunnin odotus. Esimerkkikoodissa 6 näytetään, miten ohjelma sisälsi yksinkertaisen muuttujan, jonka arvoa vertaamalla varmistettiin, että ohjelma seurasi oikean task-olion etenemistä.

```
if (taskStarted)
{
    if (drivingToDepot)
    {
        if (taskManager.taskList[veh] == null)
        {
            DriveToDepot();
        }
    }
    else if (taskManager.taskList[veh] == null)
    {
    StopVehicle();
    }
}
```

Esimerkkikoodi 6. Kahden eri task-olion subtask-listojen tarkistaminen. drivingt-ToDepot-muuttuja määrittelee ohjelmalle, kumpaa subtask-listaa käsitellään.

Ajoneuvon käytöstä poisto -ongelmatilanteesta jäi kehittämättä toiminnallisuus, jossa ajoneuvo täysin poistetaan sovelluksesta. Täysin valmista ja toimivaa ajoneuvon poistometodia ei ollut kehitetty. Vaikka ajoneuvon poisto itse SUMO-simulaatiosta on hyvin suoraviivaista, olisi ajoneuvo ja kaikki siihen liittyvät viittaukset pitänyt tyhjentää sovelluksesta ja hallinnoivista luokista. Tämä toiminnallisuus jäi kehittämättä sovellukseen, minkä vuoksi ongelmatilanne ei ole käytössä sovelluksen ongelmatilannelistassa.

5.6 Jatkokehitys

Liikenteenvalvontasovelluksen toteutuksessa on varaa lisäominaisuuksille, jotka voitaisiin toteuttaa jatkokehityksessä. Eräs tärkeä ominaisuus, joka jäi kehittämättä sovellukseen insinöörityötä varten, oli käyttäjän toimintojen ohjelmallinen seuraaminen. Muun muassa käyttäjän reaktioaika, valinnat ja toimintojen järjestys antaisivat tärkeää tietoa käyttäjän suorituskyvystä ja rasituksesta sovelluksen käytön aikana.

Ohjelmakoodia myös kommentoitiin laajasti kehityksen aikana, mikä varmisti koodin helppolukuisuuden. Jos projektiin liittyy myöhemmässä vaiheessa lisää kehittäjiä, on kehitystyö heille helpompaa, kun ohjelmakoodin kommenteissa on selvennetty kooditiedoston eri osia ja toimintoja sekä niiden käyttötarkoituksia. Yleisesti ottaen jokaisen luokan, metodin ja joskus jopa muuttujan jälkeen

kirjoitettiin lyhyt ja yksinkertainen kommentti, joka kertoo, mitä kyseinen tietorakenne tekee tai sisältää.

Myös tietokannan integrointia sovellukseen oli suunniteltu. Tietokantaan tallentuisivat kaikki käyttäjän antamat komennot aikaleimoineen ja kaikki tapahtumalokin tapahtumat. Tietokannasta voisi myöhemmin tutkia ja verrata kerättyä dataa sekä analysoida yksittäisten käyttäjien kuormitusta ja kykyä suoriutua tehtävistä ja ongelmatilanteista.

5.7 Loppuanalyysi

Toteutettu sovellus sopi tarkoitukseensa hyvin autonomisen liikenteen valvontatyökaluna. SUMO-simulaattori toimi hyvin tarkoitukseensa valmiin sovelluksen liikennesimulaattorina, ja sen muokkaaminen ja käyttämisen oppiminen onnistui riittävän helposti. Valmiin simulaattorin käyttämisessä oli se hyöty, ettei omaa simulaattoria tarvinnut kehittää erikseen, mikä säästi paljon aikaa ja työtä projektin kehitysvaiheessa.

Ongelmatilanteiden toteutus onnistui lähtökohtaisesti myös hyvin, vaikka tiettyjen ongelmatilanteiden kohdalla osa ohjelmakoodista vaati uudelleenkirjoitusta. Ongelmatilanteiden yksittäiset vaatimukset ja niin kutsutut käsikirjoitukset olivat selkeästi dokumentoitu ennen niiden varsinaista kehittämistä, mikä auttoi ja selkeytti kehitystyötä.

Projektin aloittamisessa aikaa meni hieman liikaa netedit-verkostonmuokkausohjelman opettelemiseen, eikä ohjelmaa loppujen lopuksi käytetty projektin kehityksessä kovin laajasti. Ongelmatilanteiden kehittämisen olisi voinut aloittaa jo aikaisemmin, kunhan niiden käsikirjoitukset olisivat valmistuneet nopeammin. TraCI-rajapinnan käyttö onnistui hyvin ja lähestulkoon ongelmitta. Komentojen oppiminen ja sisäistäminen oli nopeaa, ja niiden käyttäminen ongelmatilanteiden kehityksessä oli ratkaisevaa kehityksen sujuvuuden kannalta.

Osa suuremmista ominaisuuksista tietyissä ongelmatilanteissa osoittautui tarpeettomiksi tai liian epävarmasti toimiviksi, joten niiden kehitykseen hukkaantui

turhaan aikaa. Näiden ominaisuuksien käyttämättä jättäminen kuitenkin tuki kehitysprosessia nopeuttaen vianetsintää ja kehitysvauhtia tietyissä ongelmissa.

6 Yhteenveto

Autonominen ajoneuvo on käsite, jota on määritelty monella tavalla. Määritelmä kuitenkin kiteytyy muutamaan ominaisuuteen: ajoneuvo ei vaadi ihmiskuljettajalta mitään ohjausta eikä sen tarvitse olla yhteydessä muuhun liikenteeseen. Autonominen ajoneuvo ajaa itseään autonomisesti kaikkialla.

Liikenteen simulointia voi suorittaa monella eri tavalla, mutta työssä käytetty simulaattori suoritti mikroskooppista jatkuvan ajan simulaatiota. Toisin sanoen simulaatio keskittyi yksittäisiin ajoneuvoihin, ja se simuloitiin reaaliajassa niin, että jokaisen sekunnin aikana tapahtuneet tapahtumat kuvattiin simulaatiossa. Tällainen toiminnallisuus oli tarpeellinen sovelluksen käyttötarkoitukseen.

SUMO-liikennesimulaattorin mukana tulleet aputyökalut auttoivat kehitystyötä merkittävästi. Työkaluihin kuului TraCI-rajapinta, jolla käsiteltiin simulaattoria ajon aikana, simulaattorin graafinen käyttöliittymä, joka auttoi hahmottamaan simulaattorin tapahtumia, OSMWebWizard-niminen automaattinen liikenneverkostogeneraattori simulaation kartan luomista varten, sekä netedit-verkoston-muokkaustyökalu, joka auttoi kartan tarkemmassa muokkaamisessa.

Varsinainen liikenteenvalvontasovellus rakennettiin Unity-pelimoottorilla. Sovelluksen graafinen käyttöliittymä toimi hyvin ongelmatilanteiden kehitystä varten, mutta vaati joitakin kompromisseja tietyiltä ongelmatilanteiden ominaisuuksilta helpon käytettävyyden takaamiseksi. Pelimoottoriin integroitiin SUMO-simulaattori, jonka avulla sovellukseen saatiin toimiva oikeaa elämää kuvastava liikenneympäristö.

Projekti oli osa suurempaa Metropolia Ammattikorkeakoulun SAM-hanketta. Projektin kehitystyötä dokumentoitiin kehityksen aikana monella tavalla, ja kehityspalavereita pidettiin kaksi kertaa viikossa.

Ongelmatilanteiden kehittäminen onnistui hyvin, mutta tarkemmalla ja pidempikatseisella suunnittelulla olisi voitu välttää turhien käyttämättä jääneiden ominaisuuksien kehittäminen. Vaikkei kaikkia suunniteltuja ongelmatilanteita saatu kehitettyä kokonaan, tärkeimmät niistä silti saatiin toteutettua toimivasti sovellukseen.

Jatkokehityksen huomioon ottaen kommentointia olisi voinut olla laajemminkin tietyissä kooditiedostoissa, sillä jotkin operaatiot tai kohdat voivat olla epäselviä kehittäjälle, joka liittyy projektiin kesken sen kehitystä. Projektin tiedostorakenne ja tiedostojen nimet projektin sisällä kuitenkin selventävät eri osien toiminnallisuutta.

Insinöörityön päätavoite, eli ongelmatilanteiden kehitys, niiden visualisointi ja käyttö 3D-pelimoottorissa, saavutettiin aikarajan sisällä. Kehitystyö oli sujuvaa, ja lopputuote soveltui käyttötarkoitukseensa hyvin.

Lähteet

1 Verkottunut ja automatisoituva tieliikenne. 2021. Verkkoaineisto. Traficom. <https://www.traficom.fi/fi/liikenne/liikennejarjestelma/verkottunut-ja-automatisoituva-tieliikenne>. Luettu 28.6.2022.

2 Automated driving. 2021. Verkkoaineisto. SAE International. <https://web.archive.org/web/20180701034327/https://cdn.oemoffhighway.com/files/base/acbm/ooh/document/2016/03/automated_driving.pdf>. Luettu 28.6.2022.

3 Shuttleworth, Jennifer. 2019. Levels of driving automation. Verkkoaineisto. SAE International. <https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic>. Luettu 1.7.2022.

4 Pursula, Matti. 1999. Simulation of Traffic Systems – An Overview. Verkkoaineisto. Journal of Geographic Information and Decision Analysis. <https://publish.uwo.ca/~jmalczew/gida_5/Pursula/Pursula.html>. Luettu 4.7.2022.

5 Machado, Tatiane. 2017. Computer simulation of a photovoltaic system for electricity generation at residencies and condominiums. Graduation Thesis. Universidade Candido Mendes. ResearchGate. <https://www.researchgate.net/publication/322530271_Computer_Simulation_of_a_Photovoltaic_System_for_Electricity_Generation_at_Residencies_and_Condominiums> Luettu 14.7.2022.

6 Eclipse SUMO. Verkkoaineisto. SUMO. <https://www.eclipse.org/sumo/>. Muokattu 4.1.2022. Luettu 30.5.2022.

7 Institut für Verkehrssystemtechnik. Verkkoaineisto. DLR. <https://www.dlr.de/TS>. Luettu 4.7.2022.

8 TraCI. Verkkoaineisto. SUMO. <https://sumo.dlr.de/docs/TraCI.html>. Luettu 30.5.2022.

9 sumo-gui. Verkkoaineisto. SUMO. <https://sumo.dlr.de/docs/Tutorials/quick_start.html>. Luettu 1.7.2022.

10 sumo-gui. Verkkoaineisto. SUMO. <https://sumo.dlr.de/docs/sumo-gui.html>. Luettu 14.6.2022.

11 Abstract Vehicle Class. Verkkoaineisto. SUMO. <https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html#abstract_vehicle_class>. Luettu 28.6.2022.

12 OpenStreetMap. Verkkoaineisto. SUMO. <https://sumo.dlr.de/docs/Networks/Import/OpenStreetMap.html>. Luettu 23.8.2022.

13 Netedit common edit modes. Verkkoaineisto. SUMO. <https://sumo.dlr.de/docs/Netedit/editModesCommon.html>. Luettu 5.7.2022.

14 Dealessandri, Marie. 2020. What is the best game engine: is Unity right for you? Verkkoaineisto. gamesindustry.biz. <https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you>. 16.1.2020. Luettu 7.7.2020.

15 Scenes. 2022. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/Manual/CreatingScenes.html>. 9.7.2022. Luettu 13.7.2022.

16 GameObject. 2022. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/ScriptReference/GameObject.html>. 9.7.2022. Luettu 18.7.2022.

17 Lindblom, Pekko. Älykäs autonominen liikenne - Smart Autonomous Mobility. Verkkoaineisto. Metropolia Ammattikorkeakoulu. <https://www.metropolia.fi/fi/tutkimus-kehitys-ja-innovaatiot/hankkeet/sam>. Luettu 23.8.2022.

18 CodingConnected.Traci. Verkkoaineisto. SUMO. <https://github.com/CodingConnected/CodingConnected.Traci>. Luettu 14.6.2022.