Joonas Suuronen

Käyttöliittymä autonomisten ajoneuvojen valvonnan simulaattoriin

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tieto- ja viestintätekniikan tutkinto-ohjelma
Insinöörityö
27.10.2022

Tiivistelmä

Tekijä:	Joonas Suuronen
Otsikko:	Käyttöliittymä autonomisten ajoneuvojen valvonnan simulaattoriin
Sivumäärä:	51 sivua
Aika:	27.10.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaaja:	Lehtori Miikka Mäki-Uuro

Insinöörityön tarkoituksena oli suunnitella ja kehittää autonomisten ajoneuvojen valvonnan simulaattoriin graafinen käyttöliittymä. Sovellus kehitettiin Unity-pelimoottorilla ja C#-ohjelmointikielellä. Muut opiskelijat kehittivät yhteistyössä itse autonomisten ajoneuvojen simulaation käyttämällä SUMO-simulaattoria ja sen yhteydessä toimivaa TraCI-rajapintaa. Sovellus on osa Metropolia Ammattikorkeakoulun SAM-hanketta (Smart Autonomous Mobility). SAM-hanke pyrkii edistämään liikenteen tutkimus- ja innovaatiotoimintaa.

Tavoitteena oli tehdä sovellus ja simulaatio, jolla voidaan mitata, kuinka useaa ajoneuvoa yksittäinen autonomisten ajoneuvojen valvoja pystyy valvomaan, ennen kuin fysiologiset ja kognitiiviset rajat tulevat vastaan. Valvojalle luodaan rasitetta erilaisilla ongelmatilanteilla, joita hänen tulee voida havaita ja ratkoa käyttöliittymän kautta. Autonomisia, eli itsenäisesti toimivia, ajoneuvoja sovelluksessa ovat linja-autot, logistiikasta vastaavat rekat, ruohonleikkurit, valvontadroonit ja vesiajoneuvot. Simulaation autonomisten ajoneuvojen toiminta-alue rajoitettiin Helsingin Vuosaareen.

Insinöörityön tuloksena saatiin suunniteltua ja toteutettua toimiva käyttöliittymä simulaatiosovellukseen. Lisäksi käyttöliittymään tehtiin lukuisia parannuksia, eikä kehitystyötä lopetettu minimivaatimuksiin. Sovelluksen muut osat ovat vielä hieman vajaita autonomisten ajoneuvojen valvojien testaamiseen, joten kehitystyötä jatketaan yhä. Myös graafista käyttöliittymää päivitetään sovelluksen jatkokehityksessä vastaamaan uusia tarpeita.

Avainsanat: käyttöliittymä, automaatio, simulaatio, Unity

Abstract

Author:	Joonas Suuronen
Title:	User interface for surveillance application of simulation of autonomous vehicles
Number of Pages:	51 pages
Date:	27 October 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Game applications
Supervisor:	Miikka Mäki-Uuro, Senior Lecturer

The topic of the thesis was to design and develop a graphical user interface for an autonomous vehicle monitoring simulator. The application was developed with the Unity game engine and C# programming language. In collaboration, other students developed the simulation of autonomous vehicles using the SUMO-simulator and the TraCI-programming interface. The application is a part of Metropolia University of Applied Sciences' SAM-project (short for Smart Autonomous Mobility). The SAM-project aims to develop transportation research and innovation.

The goal was to develop an application and simulation that could be used to measure how many vehicles one supervisor could supervise, before their physiological and cognitive limits would come into play. The supervisor shall be burdened with various problems, which they should be able to detect and solve through the user interface. The autonomous vehicles in the application include buses, logistics' trucks, lawnmowers, surveillance drones and water vehicles. The simulation's operating area of the autonomous vehicles was limited to a part of Helsinki called Vuosaari.

The result of the thesis was a successfully designed and developed user interface for the application. Additionally, the interface received numerous improvements and the one developed did not seize at minimal functionality. Other parts of the application are still lacking for the testing of the vehicle supervisors, so additional development is still required. The graphical user interface will also be updated in the future development to meet any new requirements.

Keywords: User interface, automation, simulation, Unity

Sisällys

Lyhenteet

UI: *User interface*. Käyttöliittymä tarkoittaa laitteen tai ohjelmiston osaa, jonka kautta käyttäjä käyttää tuotetta.

UX: *User experience design*. Käyttökokemussuunnittelu on prosessi, jossa pyritään luomaan mahdollisimman intuitiivinen ja hyvä vuorovaikutus käyttäjän ja tuotteen välille.

SUMO: *Simulation of Urban Mobility.* SUMO on avoimen lähdekoodin paketti ohjelmia ja koodia, ja sillä voidaan simuloida realistista liikennettä etenkin kaupunkiympäristöihin.

SAM: *Smart Autonomous Mobility*. Älykkään autonomisen liikenteen tutkimus- ja innovaatiotoiminnan kehittämistä edistävä hanke.

Inspector: Unityn yhteydessä käsite inspector tarkoittaa kehitystyökalua, jolla voidaan tarkkailla ja käsitellä peliobjektien komponentteja ja ominaisuuksia.

1 Johdanto

SAM-hanke (engl. Smart Autonomous Mobility) on älykkään autonomisen liikenteen tutkimus- ja innovaatiotoimintaa edistävä hanke. Yksi SAM-hankkeen päätavoitteista on selvittää, kuinka useaa autonomista ajoneuvoa yksi valvoja pystyy valvomaan samanaikaisesti ilman liiallista henkistä tai fyysistä kuormitusta. Ensisijaisesti ajoneuvot ovat automaattisia henkilöliikenteen sähköbusseja. Myös erilaisia logistiikka-ajoneuvoja sekä viher- ja vesialueilla toimivia valvontadrooneja kuuluu projektiin. Ajoneuvot ovat pääosin autonomisia, eli esimerkiksi tarkkoja ohjausliikkeitä tai nopeutta ei tarvitse säätää sovelluksesta.

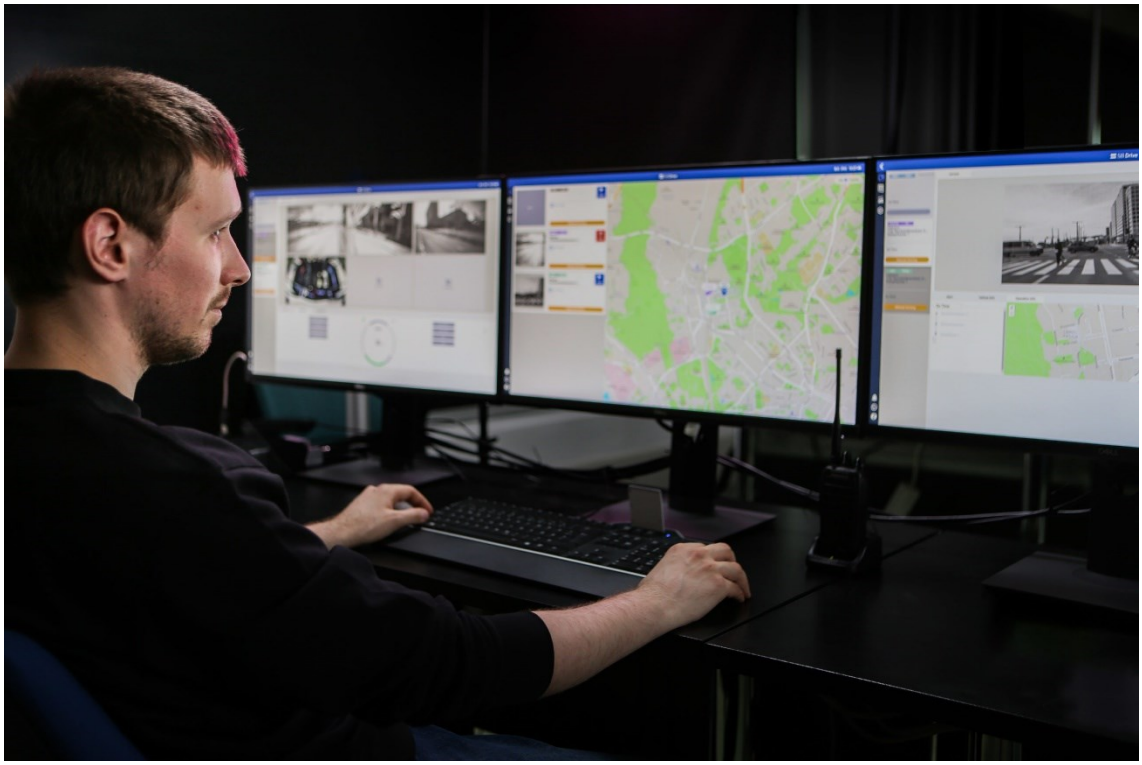
On kuitenkin välttämätöntä, että autonomisia ajoneuvoja valvotaan erilaisten ongelmatilanteiden varalta. Esimerkkejä simuloiduista ongelmatilanteista ovat meluavat tai häiriköivät matkustajat, estynyt kaista tai tie ja häiriö ovien sulkeutumisessa. Tämän projektin osuus ei ole suunnitella itse ajoneuvoja, vaan oletetaan, että ajoneuvoissa on erilaisia sensoreita, kuten desibelimittareita ja kameroita, joilla saadaan havaittua poikkeukselliset tilanteet.

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa käyttöliittymä autonomisten ajoneuvojen simulaattoriin. Yhteistyössä muut opiskelijat kehittivät itse simulaation ja ongelmatilanteet. Käyttöliittymän on tarkoitus antaa käyttäjälle tietoa ja mahdollistaa erilaiset käskyt simulaation ajoneuvoille.

Opinnäytetyön raportin rakenne on seuraava. Aluksi selvennetään SAM-hanketta ja mikä tämän insinöörityön osuus on siinä. Seuraavaksi käsitellään yleisesti käyttökokemuksen ja käyttöliittymän suunnittelua. Sen jälkeen tarkennetaan sovelluksen aloituspaneelien käyttöliittymän toteutusta ja päänäkymän toteutusta. Lopuksi avataan, mitä parannuksia käyttöliittymä sai projektin edetessä ja minkä takia näihin parannuksiin päädyttiin.

2 SAM-hanke

Ennen kuin syvennytään itse käyttöliittymän suunnitteluun ja toteuttamiseen, on tärkeää ensin tarkentaa, mistä SAM-hankkeessa on kyse. Tulevaisuudessa nähdään paljon enemmän autonomisia ja älykkäitä ajoneuvoja tai liikennejärjestelmiä. Esimerkiksi Suomessakin näkee Espoossa Starship-yrityksen valmistamia autonomisia kuljetusrobotteja, jotka kuljettavat ruokaa tavallisille ihmisille kotiovelle asti. (1.) Maailmanlaajuisesti etenkin isompien autonomisten ajoneuvojen valvontaratkaisut ovat todella alkeellisessa vaiheessa. Yksi tärkeistä SAM-hankkeen tarkoituksista on selvittää, kuinka useaa autonomista ajoneuvoa yksi valvoja pystyy valvomaan, ennen kuin fysiologiset ja kognitiiviset rajat tulevat vastaan. Kuvassa 1 on esimerkki siitä, miltä autonomisten ajoneuvojen valvontatila voisi näyttää.



Kuva 1. Sensible 4 -yrityksen esittämä valvontapiste autonomisten ajoneuvojen valvomiseen. (2.)

Ajoneuvojen autonomian voi jakaa kuuteen tasoon J3016-standardin mukaan. Tasot 0–2 ovat lievää helpotusta ajamiseen, kuten kuolleen kulman merkkivalo, automaattinen hätäjarrutus tai kaistan automaattinen keskittäminen. Tasolla 3 auto pystyy ajamaan suurimmaksi osaksi automaattisesti, mutta tarpeen mukaan ihminen saattaa joutua ohjaamaan. Tasolla 4 auto on täysin autonominen, eli ohjauspyörää tai polkimia ei tarvita. Automaatiotason 4 ajoneuvo ei kuitenkaan pysty ajamaan kaikissa paikoissa ja olosuhteissa. Tason 5 ajoneuvo on sama kuin tason 4, mutta ilman rajoituksia paikan ja olosuhteen mukaan. (2.)

Suurin osa niin sanotusti itseajavista ajoneuvoista, joita on esimerkiksi Teslalla, on tason 3 autonomisia ajoneuvoja. Lisäksi ne ovat lähes poikkeuksetta henkilöautoja eivätkä linja-autoja tai logistiikasta vastaavia rekkoja. SAM-hanke pyrkii edistämään tason 4 tai jopa tason 5 autonomisia ajoneuvoja. SAM-hankkeen tavoittelemat ajoneuvot eivät siis tarvitse kuljettajaa tai ohjauspyörää, vaan mahdollisuus lähettää ohjeita etäältä riittää. Vastaavan haasteeseen lähteneitä projekteja tai yrityksiä on hyvin vähän. (3.)

2.1 SAM-hankkeen henkilöstö ja työnjako

SAM-hanke on jaettu neljään eri työpakettiin. Työpaketti 1 vastaa projektin perusasioista, kuten henkilöstöstä ja kaikenlaisista paperitöistä. Työpaketti 2, johon insinöörityön kirjoittajakin kuuluu, on itse simulaation ja sovelluksen kehittäminen. Työpaketti 3 vastaa lähinnä käyttäjäkokeista. Käyttäjäkokeissa tallennetaan tietoa vapaaehtoisista, joten on tärkeää tietää, mitä tietoa tallennetaan, missä sitä säilytetään ja kuka siihen pääsee käsiksi. Työpaketissa 3 tarvitaan siis lakiteknistä osaamista ja taitoa kerätä fysiologista dataa vapaaehtoisilta. Työpaketin 4 tarkoitusta ei ole kerrottu, mutta se mahdollisesti liittyy käyttäjäkokeiden tulosten tulkintaan.

Työpaketissa 2 on ollut keväästä 2022 lähtien kuusi neljännen vuoden opiskelijaa. Syksyllä 2022 osa opiskelijoista on valmistunut ja lähtenyt projektista, mutta tilalle on saatu uutta henkilöstöä vastaamaan valmistuneiden opiskelijoiden osuutta. Kirjoittajan pääosuus oli suunnitella ja toteuttaa

käyttöliittymä. Lisäksi kirjoittaja teki paljon opiskelijoiden ohjaamista. Kokouksissa piti aina varmistaa, että kaikilla on sopivasti tehtävää eikä useampi opiskelija tee vahingossa samaa asiaa. Kirjoittaja osallistui myös ongelmatapauksissa esitettävien videoiden kuvaamiseen aina, kun sille oli tarvetta. Muut opiskelijat vastasivat pitkälti taustajärjestelmistä, kuten SUMO:sta, TraCI-rajapinnasta, Unityn ja SUMO:n toimivuudesta yhdessä ja tietokannasta.

2.2 Ongelmatilanteet

SAM-hankkeen tarkoitus on siis varsin laaja. Osuus, johon insinöörityön tekijä ja viisi muuta opiskelijaa osallistuivat, oli luoda autonomisten ajoneuvojen simulaatio. Seuraavaksi lisättiin erilaisia simuloituja ongelmatilanteita, jotka vaativat käyttäjältä toimia käyttöliittymän kautta. Tarkoituksena on tehdä realistisen tuntuinen simulaatio, jolla voidaan luoda rasitetta ja painetta koehenkilöille. Ongelmatilanteiden suunnitelma ja toteutus on vaihdellut projektin edetessä, mutta ainakin seuraavat ongelmatilanteet löytyvät simulaattorista jossain muodossa:

- Este reitillä, jossa käyttäjän tulee hyväksyä tai valita uusi reitti.
- Este kaistalla, jossa käyttäjän tulee simuloidun ajoneuvon kameranäkymän perusteella valita, odotetaanko esteen poistumista vai ohitetaanko se.
- Ovien sulkeutumisessa viivettä, jossa käyttäjän tulee selvittää, minkä takia ovet eivät sulkeudu normaalisti. Ovet saa jälleen toimimaan normaalisti painamalla manuaalisesti ovet auki ja jälleen kiinni sovelluksen käyttöliittymän ohjauspaneelista.
- Ajoneuvon lisäys, jossa käyttäjän tulee yksinkertaisesti hyväksyä käyttöliittymään ilmestyvästä ponnahdusikkunasta uuden reitin aloittaminen.
- Ajoneuvon poistaminen, jossa käyttäjän tulee pysäyttää auto ja ohjata se varikolle.

Ongelmatilanteet voidaan jakaa neljään eri asteeseen. Ensimmäinen aste, jonka värikoodi on vihreä, on ilmoitustyyppinen viesti. Ensimmäisen asteen ilmoitus ei tyypillisesti vaadi toimia käyttäjältä, mutta on hyvä tietää. Esimerkki ensimmäisen asteen tapahtumasta on uuden ajoneuvon onnistunut vuoron aloittaminen. Toinen aste, jonka värikoodi on keltainen, on varoitustyyppinen ilmoitus, joka saattaa tarvita toimia käyttäjältä. Esimerkki toisen asteen tapahtumasta ovat metelöivät tai häiriköivät matkustajat, jolloin käyttäjä saattaa joutumaan tekemään toimia. Kolmannen asteen ilmoitus, jonka värikoodi on punainen, tarkoittaa poikkeuksellista merkittävää tapahtumaa, johon käyttäjän tulee lähes poikkeuksetta reagoida. Esimerkki kolmannen asteen ilmoituksesta on moottorivikailmoitus ajoneuvosta. Neljäs ilmoitusaste on hieman poikkeuksellinen. Muut asteet tekevät ilmoituksen tapahtumalokiin ja jaotellaan tärkeyden mukaan. Neljäs aste on ponnahdusikkuna, joka vaatii käyttäjältä hyväksynnän napin painalluksella. Esimerkki neljännen asteen tapahtumasta on hyväksyntä uuden ajoneuvon vuoron aloittamisesta (josta tulee vielä ensimmäisen asteen ilmoitus, kun aloitus on onnistunut).

2.3 Simulaation testaaminen

Kun simulaatio, käyttöliittymä ja ongelmatilanteet on kehitetty riittävälle tasolle, voidaan luoda noin puolitoista tuntia kestävä skenaario, jonka tarkoitus on testata koehenkilöön kohdistuvaa fysiologista ja kognitiivista rasitetta. Tämän kokeen aikana koehenkilöstä mitataan erilaisia fysiologisia piirteitä, kuten sydämen sykettä, hengitystaajuutta ja mahdollisesti kortisolia syljestä. Koehenkilö täyttää myös lopuksi kyselyn auttamaan arviota fysiologisista ja fyysisistä tekijöistä kokeen ajalta. Käyttöliittymään liittyvää palautetta otetaan myös vastaan.

Simuloitu skenaario on jokaiselle koehenkilölle samanlainen. Aluksi skenaariossa on niin sanottu alkulämmittely, jossa käyttäjä saa totutella rauhassa käyttämään sovellusta ja ymmärtämään sen toimintaa. Tämän jälkeen ongelmatilanteita alkaa esiintymään rauhallista tahtia. Reilun puolivälin jälkeen nousee kuvitteellinen myrskysää. Tässä vaiheessa ongelmatilanteiden tahti nopeutuu merkittävästi ja koehenkilö kokee oletettavasti suurimman

kuormituksen. Lopuksi ongelmatilanteiden määrä alkaa harvenemaan, kunnes tilanne on täysin rauhallinen ja hallinnassa, jolloin käyttäjän tulee lopettaa kuvitteellinen työvuoro ja kirjoittaa loppuraportit.

Simulaatiota testataan kymmenillä vapaaehtoisilla, projektiin kuulumattomilla henkilöillä. Ennen koetta pyritään saamaan hieman ymmärrystä heidän taustoistaan. Esimerkiksi kokeneet strategiapelien pelaajat ovat mahdollisesti tottuneet hallitsemaan useampaa virtuaalista yksikköä tai ajoneuvoa samanaikaisesti, ja myös tottuneet tähän liittyvään fysiologiseen kuormaan. Simulaation skenaariota tulee todennäköisesti tasapainottaa useampaan kertaan lisäämällä tai vähentämällä ongelmatilanteita ja ajoneuvojen määrää. Tämän tarkoitus on löytää sopiva fysiologisen ja kognitiivisen paineen määrä, jota keskiverto käyttäjä pystyy hallitsemaan siedettävästi.

Projektiin tehtiin myös tietokanta, jonne tallennetaan tietoa simulaatiossa tapahtuvista asioista. Esimerkiksi jokainen ongelmatilanne, napin painallus ja mahdollisesti hiiren liikkeet tallennetaan. Näin voidaan tutkia muuan muassa, kuinka kauan käyttäjillä menee reagoida ongelmatilanteisiin ja painelevatko he vahingossa vääriä nappeja. Lisäksi saadaan tietoa siitä, onko käyttöliittymässä parantamisen varaa, mikäli käyttäjien pitää jatkuvasti liikuttaa hiirtä merkittäviä etäisyyksiä näytöllä, tai ovatko käyttöliittymän osat tarpeeksi helposti löydettävissä.

3 Käyttökokemuksen suunnittelu

Seuraavaksi käsitellään projektin suunnitteluvaihetta. Ensin selitetään termit käyttökokemus (UX, engl. user experience) ja käyttöliittymä (UI, engl. user interface) sekä näiden kahden välinen yhteys ja merkitys ohjelmistokehityksessä.

Käyttöliittymä tarkoittaa laitteen tai ohjelmiston osia tai osaa, joka yhdistää käyttäjän ja tuotteen. Useimmiten käyttöliittymä välittää käyttäjälle tietoa ja mahdollisuuden antaa tuotteelle syötettä. Esimerkiksi käyttäessä älypuhelimen laskinsovellusta, käyttöliittymä älypuhelimen ja käyttäjän välillä on

kosketusnäyttö, josta käyttäjä saa tietoja ja antaa syötettä kosketuksilla. Tässä esimerkissä on toisaalta myös toinen käyttöliittymä, nimittäin laskinsovelluksessa on useita virtuaalisia painikkeita, joilla ohjata sovelluksen toimintaa. Lisäksi laskin tarvitsee tekstikentän käyttöliittymäänsä viestiäkseen käyttäjälle laskutoimituksen tuloksen. Itse laskinsovelluksen käyttöliittymästä voidaan käyttää nimitystä graafinen käyttöliittymä (GUI, engl. graphical user interface), sillä se on virtuaalinen näytölle piirretty käyttöliittymä.

Termi käyttökokemus on vaikea määritellä, sillä se vaihtelee paljon kohteen mukaan. Yleisesti käyttökokemus tarkoittaa käyttäjän kokemusta tuotteen tai palvelun kokonaisuudesta ennen sen käyttöä, käytön aikana ja käytön jälkeen. Esimerkiksi älypuhelimen laskinsovelluksessa, käyttökokemus alkaa jo, kun käyttäjä huomaa tarvitsevansa laskinta. Hänelle tulee aiempien kokemusten perusteella mielikuva siitä, miten hän voi käyttää tuttua kätevää sovellusta laskutoimitukseen. Kun sovellus on avattu, looginen, selkeä ja helppokäyttöinen käyttöliittymä mahdollistaa koneen tarkan vastauksen saamisen nopeasti. Tämän esimerkin käyttökokemus päättyy aika nopeasti, kun käyttäjä on saanut vastauksensa ja sulkee sovelluksen.

Laskimen käyttö on tyypillisesti aika yksinkertainen ja nopea käyttökokemus, mutta vaikkapa kaupassa käydessä käyttökokemus kestää huomattavasti kauemmin. Poikkeuksellisen huonot ja hyvät käyttökokemukset jäävät myös usein käyttäjien mieleen käyttökokemuksen jälkeen. Hyvä käyttökokemus saa asiakkaat tai käyttäjät palaamaan. Huono käyttökokemus puolestaan saa asiakkaat tai käyttäjät etsimään vaihtoehtoisia tuotteita tai palveluita. Opinnäytetyön sovelluksessa hyvällä käyttökokemuksella on kuitenkin hieman erilainen prioriteetti. Vaihtoehtoisia sovelluksia ei ole, mutta hyvä käyttökokemus tarvitaan, jotta koehenkilöille ei tule ylimääräistä rasitetta epäintuitiivisen ja huonon käyttöliittymän vuoksi. Koehenkilön kokeman rasitteen tulisi siis syntyä autonomisten ajoneuvojen valvomisesta ja ongelmien ratkaisemisesta eikä uuden käyttöliittymän opettelusta.

3.1 Käyttökokemuksen suunnitteluperiaatteet

Käyttökokemussuunnittelu voi olla yllättävän haastavaa, sillä usein luodaan jotain uutta ja valmiiksi luoduista ratkaisuista ei välttämättä ole apua. Sekin voi olla hankalaa tutkia, mitä asioita kannattaa hyödyntää valmiista ratkaisuista ja kuinka paljon. Käyttökokemus on ainakin osittain subjektiivista, eli jokaisella on omat mieltymyksensä, eikä ole mahdollista tehdä tuotetta, joka vastaa täydellisesti kaikkien makua ja tarpeita. Usein käyttökokemukset sisältävät samoja tapoja ja periaatteita, jotka ovat valtaenemmistölle mieluisia ja intuitiivisia, sillä käyttäjät ovat tottuneet niihin entuudestaan muista tuotteista. Esimerkiksi käyttöliittymissä on aina tärkeimmät, eniten käytetyt napit ja elementit päänäkymässä, mutta harvemmin tarvittavat osat, kuten äänenvoimakkuuden säädin, löytyy napin tai kahden alta asetuksista.

Tilanteita, joissa käyttökokemussuunnittelua tarvitaan, on oikeastaan loputtomasti, joten tarkkaa metodologiaa on vaikeaa käyttää. On kuitenkin useampia eri lähestymistapoja hyvän käyttökokemuksen suunnitteluun. Tässä projektissa suunnitteluvaiheessa nähtiin parhaaksi lähestymistavaksi suunnitteluajattelu (engl. Design Thinking). Se on hyvä lähestymistapa, kun luodaan jotain täysin uutta jo valmiiksi olemassa olevien tuotteiden parantelun sijaan. (4.)

> Suunnitteluajattelu on yleisesti määritelty analyyttisenä ja luovana prosessina, joka sitoo ihmisen mahdollisuuksia kokeilla, luoda ja tehdä prototyyppejä, kerätä palautetta ja uudelleensuunnitella. (5.)

Suunnittelunajattelun osuus on kuitenkin vain saada tehokas ja looginen pohja sovellukselle. Tämän jälkeen vaihdetaan lähestymistapaa. Koska suunnitteluajattelu auttoi tekemään hyvän pohjan sovellukselle, nyt sitä tulee vielä hioa ja parantaa. Olemassa olevien tuotteiden parantamiseksi sopivampi lähestymistapa on ihmiskeskeinen suunnittelu (HCD, engl. Human-Centered Design). Kun suunnitteluajattelussa keskityttiin tehokkaaseen käyttöliittymään, joka ajaa asiansa tehokkaasti, ihmiskeskeinen suunnittelu pyrkii parantamaan tuotteesta tavalliselle ihmiselle sopivamman ja helppokäyttöisemmän. Tyypillisiä ihmiskeskeisen suunnittelun lähestymistapoja on kerätä palautetta

käyttäjäkokeiluista ja iteroida käyttökokemusta useita kertoja, niin että tehdään aina uusia parannuksia. Usein parannukset saattavat olla hyvinkin pieniä, mutta suuri määrä pieniäkin parannuksia johtaa lopulta merkittävästi parempaan lopputulokseen. Parannuksia käsitellään lisää luvussa 6.

3.2 Suunnittelun aloittaminen

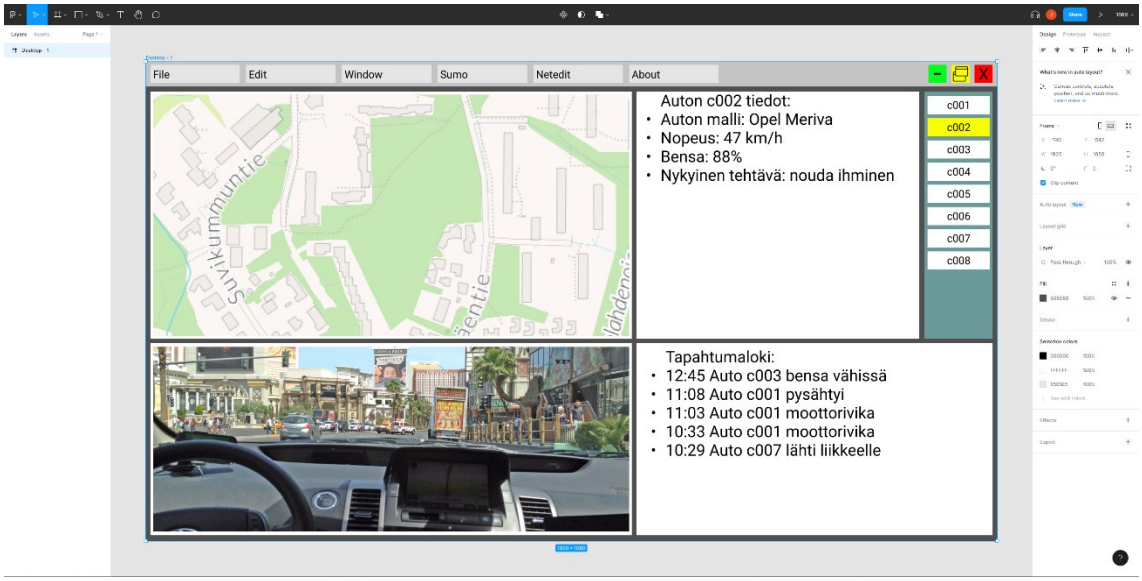
Suunnitteluajattelu on pitkälti maalaisjärjen käyttöä. Looginen ja rationaalinen ajattelukyky on siis tärkeää hyvässä käyttökokemus- ja käyttöliittymäsuunnittelussa. Aiemmasta kokemuksesta on paljon hyötyä. Useimmiten kannattaa myös ottaa mallia jo valmiista ratkaisuista, mikäli sellaisia on. Autonomisten ajoneuvojen valvontatyökalut ovat kuitenkin varsin uusi asia globaalisti. Valmiita ratkaisuja on rajatusti, eikä niissä ole kaikkia tähän käyttötarkoitukseen tarpeellisia elementtejä.

Sovellusta piti siis lähteä suunnittelemaan tyhjästä. Luonnosten tekemiseen ja kokeiluun on erilaisia työkaluja. Insinöörityöhön valittiin ilmainen, verkossa käytettävä Figma-niminen prototyyppityökalu, sillä se oli työn tekijälle hieman entuudestaan tuttu. (6.) Figmassa on helppo kokeilla erilaisia käyttöliittymäelementtien asetteluja, ja kaikki pysyy siistinä, symmetrisenä ja loogisena. Käyttöliittymän suunnittelun aloitusvaiheessa tiedettiin, että ainakin seuraavat elementit ovat tarpeellisia:

- kartta, josta näkee myös ajoneuvot
- ajoneuvolista, joka sisältää kaikki aktiiviset ajoneuvot ja mahdollisuuden valita ainakin yksittäinen ajoneuvo
- tietopaneeli, joka kertoo tietoja valitusta ajoneuvosta. Myöhemmin tämä todettiin turhaksi ja se jätettiin pois ainakin toistaiseksi
- kameranäkymä, joka näyttää kuvaa valitun ajoneuvon kamerasta
- tapahtumaloki, jonne tulee ilmoituksia ajoneuvoihin liittyvistä tapahtumista
- yläpaneeli, josta saa auki erilaisia alivalikoita. Tämä osa oli hieman epävarma, sillä ei tiedetty vielä, tuleeko sille olemaan tarvetta.

Pieni aiempi kokemus yhdistettynä Figman hyvään käyttökokemukseen mahdollisti työkalun oppimisen nopeasti. Vajaassa viikossa oltiin tyytyväisiä

varhaiseen Figmassa luotuun käyttöliittymän prototyyppiin. Tämä prototyyppi näkyy kuvasta 2.



Kuva 2. Figma-prototyyppi autonomisen ajoneuvosimulaation valvontatyökalusta. Prototyypissä näkyy pääosin tutut osat tutuissa paikoissa. Merkittävin muutos prototyypissä on jatkuvasti näkyvä kameranäkymä, joka siirrettiin myöhemmin kartan alle. Lisäksi ohjauspaneeli puuttuu kokonaan prototyypissä.

Prototyyppi on tehty yleisimpään tietokoneiden resoluutioon, eli 1920 x 1080, joka on 16:9-kuvasuhteessa. Itse oikea sovellus pyritään saamaan toimimaan myös muilla resoluutioilla, jotta sitä voidaan käyttää ainakin tableteilla, mutta se ei ole tärkeä prioriteetti. Vasemmalla ylhäällä on isoin käyttöliittymäelementti, eli kartta. Kartta on yksi tärkeimmistä osista työkalua, joten se on kookkain. Kartan suuren koon syynä on myös se, että se mahdollistaa hyvän kokonaiskuvan näkemisen, kun mahdollisimman moni ajoneuvo voi olla näkyvissä samanaikaisesti. Kartan alapuolella on toiseksi isoin elementti, ajoneuvon kameranäkymä. Se on tarkoituksella tehty leveäksi, jotta kamerakuvasta näkee mahdollisimman paljon leveyssuunnassa. Maa-ajoneuvoliikenteen luonteen takia leveyssuuntainen näkyvyys on paljon tärkeämpää kuin pituussuunnan näkyvyys. Oikealla ylhäällä sijaitseva ajoneuvolista on yksinkertainen ja selkeä. Valittu ajoneuvo on merkitty keltaisella. Ajoneuvolistan vierellä näkyy tietoja valitusta ajoneuvosta. Oikealla alhaalla on tapahtumaloki, jonne tulee

dynaamisesti simulaation aikana ilmoituksia ajoneuvojen tapahtumista. Viimeisimmässä versiossa on aika paljon lisäyksiä ja muutoksia tähän prototyyppiin verrattuna, mutta siitä lisää luvussa 6.

Prototyypissä on geneerinen kuva jostain kartasta ja autonäkymästä havainnollistamaan, miltä sovellus näyttäisi tulevaisuudessa. Tekstikentissä on myös paikanpitäjinä karkeasti jotain sellaista tekstiä, mitä voisi kuvitella olevan lopullisessa versiossa. Suunnitelma siitä, tarvitaanko yläpaneelia ja mitä siinä tulisi olemaan, oli vielä epäselvää prototyypin vaiheessa. Aiemman kokemuksen perusteella tiedettiin kuitenkin, että on helpompi poistaa käyttöliittymän osia kuin luoda uusia. Tämä johtuu siitä, että uudelle osalle pitää tehdä tilaa muista käyttöliittymän osista. Se ei ole aina mahdollista, sillä joissain tapauksissa muut käyttöliittymän osat jäävät liian pieneksi sulavan käyttökokemuksen kannalta. Kun poistetaan turhaksi todettua käyttöliittymän osaa, se tarkoittaa vain lisää tilaa muille osille, mikä johtaa yleensä parempaan käyttökokemukseen.

Figma-prototyypin valmistumisvaiheessa saatiin idea siitä, että sovelluksen paneeleita pystyisi siirtämään ja skaalaamaan suoritusaikana. Ideasta keskusteltiin tekijäryhmässä. Tultiin kuitenkin pian yksimieliseksi siitä, että tämä ei olisi kovin hyvä idea. Koehenkilöt käyttävät sovellusta noin 90 minuuttia, joten heidän tulisi tottua käyttöliittymään nopeasti. Jos sovellus on suunniteltu sellaiseksi, että sitä on helppo muokata, käyttäjät saattavat kokeilla erilaisia asetelmia sen sijaan, että tottuisivat valmiiseen loogiseen asetelmaan. Siirreltävissä paneeleissa tulisi myös paljon uusia ongelmia, kuten miten kaikki paneelit saadaan aina olemaan täysin näkyvissä. Jos osa paneeleista on muiden paneelien alla, käyttäjältä saattaa jäädä tärkeää tietoa tai painikkeita tavoittamattomiin ilman, että koehenkilö edes huomaa asiaa. Muokattavat paneelit ovat siis liian kaoottinen ja monimutkainen idea simulaatioon, mutta sitä ei ole suljettu täysin pois. Etenkin jos sovellus etenee oikeaan käyttöön autonomisten ajoneuvojen valvontaan, lienee hyvä antaa vakituisten työntekijöiden muokata työkalujaan juuri omaan käyttöön sopiviksi.

3.3 Siirtyminen käyttöliittymän toteutukseen

Kun prototyyppi oli valmis, tuli seuraavaksi haasteeksi itse käyttöliittymän toteuttaminen ja sitten yhdistäminen simulaatioon. Projektiryhmässä todettiin, että Unity-pelimoottori olisi erinomainen alusta tehdä sovellus. Pääsyyt Unityn valitsemiseen olivat, että kaikki olivat jo valmiiksi hyvin kokeneita Unityn kanssa ja itse ajoneuvojen simulaatioon käytetty SUMO sisältää valmiit integraatioratkaisut Unityyn. Lisäksi Unity tarjoaa mahdollisuuden kääntää sovelluksen yli 20:lle eri alustalle. Tulevaisuudessa ilmaantuisi mahdollisesti tarvetta kääntää sovellus Windows-, WebGL-, Android-, Apple- tai Linux-alustoille, jotka ovat kaikki tuettuja Unityssä. (7.) Unity-pelimoottoria voi ohjelmoida usealla eri ohjelmointikielellä. Yhteiseksi ohjelmointikieleksi valittiin C#, sillä se on Unityssä oletuksena, eli ylimääräistä säätämistä ei tarvita. Lisäksi kaikilla ryhmässä on kokemusta Unityn ja C#:n yhdistelmästä. Unityyn löytää myös eniten apua ja dokumentointia C#-kielellä.

Vaihtoehtoja sovelluksen kehittämiseen oli useampia. Kaikissa ratkaisuissa on hyvät ja huonot puolensa. Unityssä oleellisia heikkouksia ei kuitenkaan ole paljoa. Unityyn verrattuna suoritusteho olisi varmasti parempi monessa muussa vaihtoehdossa, kuten työkalun ohjelmoimisessa alusta asti C++-ohjelmointikielellä ja avoimilla koodikirjastoilla. Suoritusteho ei kuitenkaan ole ongelma, sillä sovellus tulee olemaan tarpeeksi kevyt myös selaimille ja tableteille. Sovellukseen jää luultavasti paljon optimointimahdollisuuksia, sillä ryhmä keskittyy ensisijaisesti perustoiminnallisuuksiin, ja optimointiin keskitytään vain, jos sille tulee tarvetta tai päätoiminnallisuudet ovat valmiita.

Epäoptimaalisen suoritustehon lisäksi Unityn mahdollisena haittapuolena on se, että Unityn tekemä käännös saattaa viedä hieman enemmän levytilaa kuin olisi välttämätöntä. Tätä on vaikea arvioida, sillä ei ole muita käännöksiä, joihin verrata. Sovelluksen viemä levytila on joka tapauksessa enintään muutaman gigatavun, eikä tätä haittapuolta voi pitää mitenkään merkittävänä.

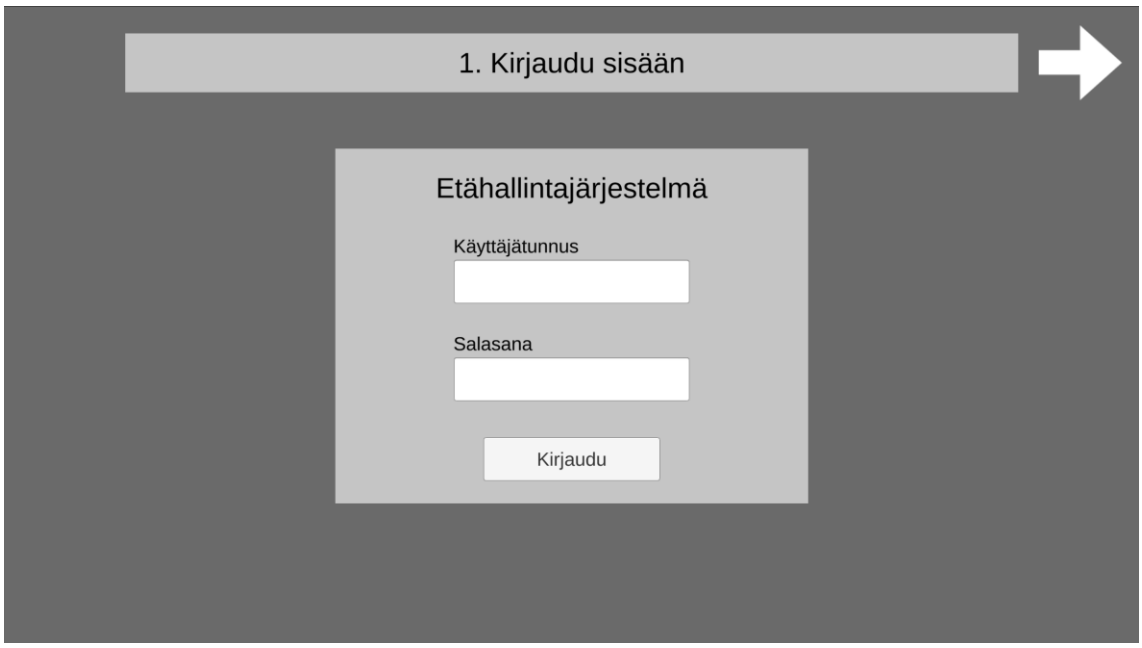
4 Aloituspaneelien toteutus

Suunnitelma, prototyyppi ja kehitysalustan valinta olivat valmiita. Tässä luvussa käsitellään itse sovelluksen kehittämistä. Ryhmä valitsi yksimielisesti projektiin tavallisen 3D-pohjan sisäänrakennetulla grafiikkaliukuhihnalla (engl. built-in render pipeline). Hienommat grafiikkaliukuhihnat eli universaali grafiikkaliukuhihna (engl. URP, Universal Render Pipeline) ja teräväpiirtografiikkaliukuhihna (engl. HDRP, High Definition Render Pipeline) ovat olemassa lähinnä parempaa 3D-grafiikkaa varten, eli tässä käyttöliittymäpainotteisessa projektissa niistä ei olisi hyötyä. Projekti tehtiin 3D-sovelluksena, koska se tarjoaa mahdollisuuksia, joita 2D-pohja ei mahdollista. Yksi idea oli 3D-visualisointi simulaatiosta, mutta siitä luovuttiin myöhemmin. Projektissa käytettiin Unityn versiota 2021.2.9f1. Versionhallintaan käytettiin GitLab-palvelua.

Seuraavaksi käsitellään aloituspaneelien teknistä toteutusta. Aloituspaneelit tarkoittavat ensimmäisiä sivuja, jotka käyttäjä näkee avatessaan sovelluksen. Jokaisessa aloituspaneelissa on otsikko ja nuolet edelliseen ja seuraavaan paneeliin (mikäli sellaisia on olemassa). Nuolet toimivat painikkeina navigointiin alkupaneelien välillä.

4.1 Kirjautuminen

Sovelluksessa päädyttiin käyttämään viittä erillistä aloituspaneelia, jotka on tarkoitus käydä läpi ennen itse valvontatyökaluun etenemistä. Käyttäjällä on kuitenkin myöhemminkin mahdollisuus palata takaisin aloituspaneeleihin varsinaisesta valvontatyökalusta. Ensimmäisessä paneelissa käyttäjän on tarkoitus kirjautua sisään (kuva 3).



Kuva 3. Kirjautumispaneelissa on erilliset kentät käyttäjätunnukselle ja salasanalle sekä nappi sisäänkirjautumista varten.

Käyttäjätunnuksella ja salasanalla on sovelluksessa kaksi tarkoitusta. Ensinnäkin data, jota kerätään sovelluksen käytön aikana, on helppo yhdistää käyttäjään käyttäjätunnuksen perusteella. Toinen tarkoitus on parantaa immersiota, sillä oikeassa autonomisten ajoneuvojen valvontatyökalussa täytyy olla jonkinlainen kirjautumisjärjestelmä turvallisuussyistä. Tässä tarkoituksessa käyttäjätunnuksen ja salasanan käsittelyyn riittää todella yksinkertainen ratkaisu, vaikka se ei olekaan turvallinen. Ratkaisun perusidea esitetään koodiesimerkissä 1.

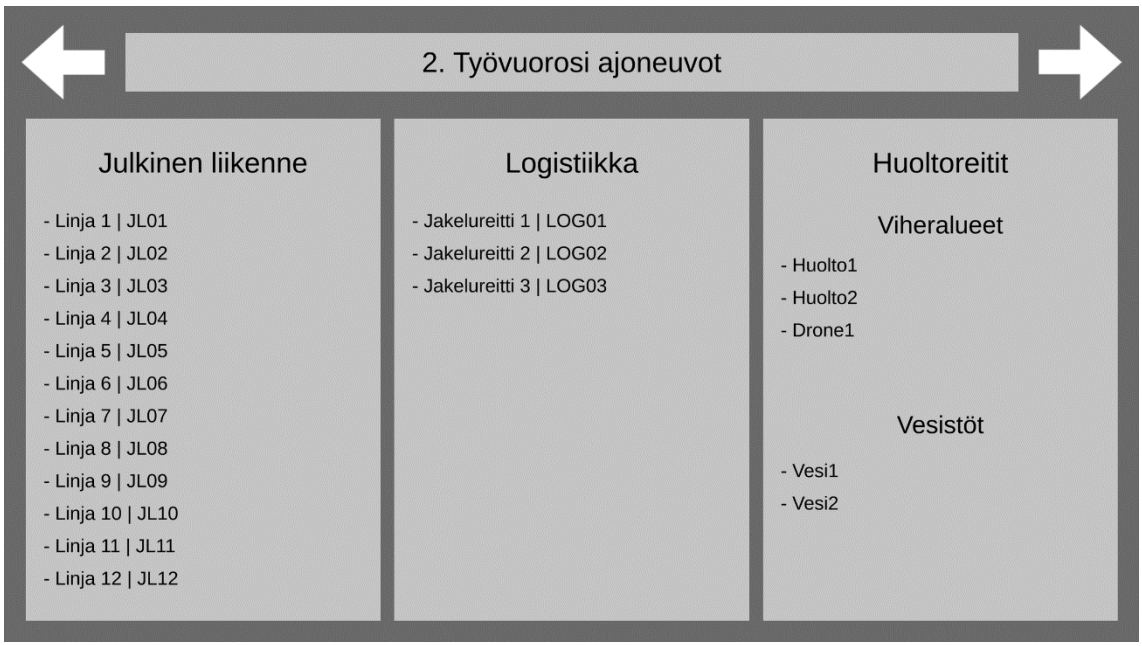
```
public void CheckValidID()
    {
        if (usernameField.text == "käyttäjätunnus 1" &&
passwordField.text == "salasana 1")
        {
            loginAllowed = true;
        }
        else if (usernameField.text == "käyttäjätunnus 2" &&
passwordField.text == "salasana 2")
        {
            loginAllowed = true;
        }
        else
        {
            loginAllowed = false;
        }
    }
```

Esimerkkikoodi 1. Jokaiselle käyttäjätunnus-salasanayhdistelmälle kirjoitetaan oma ”if”- tai ”else if” -lauseke. Lausekkeen parametreissä tarkistetaan, vastaako salasana käyttäjätunnusta. Mikäli vastaa, sisäänkirjautuminen sallitaan. Muuten evätään sisäänkirjautuminen.

Oikeassa valvontatyökalussa pitäisi salata käyttäjätunnukset ja salasanat, jotta niitä ei saisi suoraan luettua. Tyypillinen käytäntö on myös piilottaa näytöltä salasanakenttään kirjoitettu teksti, jotta muut ihmiset, jotka saattaisivat nähdä näytön, eivät kuitenkaan näe suoraan salasanaa. Liian monesta väärästä salasanasta tulisi myös lukita kirjautumisyritykset hetkeksi, jotta ”brute force” -hakkeroinnista tulisi huomattavasti hankalampaa. On myös yleistä varmentaa kirjautuminen ulkoiselta palvelimelta, mikä muuan muassa mahdollistaa salasanojen ja käyttäjätunnusten vaihtamisen helpommin myös silloin, jos tunnukset ovat unohtuneet. Lisäksi palvelimella turvattu kirjautuminen lisää turvallisuutta. Mitään näistä ei kuitenkaan tarvittu tässä vaiheessa projektia.

4.2 Työvuoron ajoneuvot

Toisen paneelin oli tarkoitus listata kaikki ajoneuvot, joita tässä työvuorossa on aktiivisena. Ajoneuvot on jaettu kolmeen kategoriaan: julkinen liikenne, logistiikka ja huoltoreitit. Huoltoreitit-kategoria sisältää myös kaksi alakategoriaa, jotka ovat viheralueet ja vesistöt. Jokaisella kategorialla on oma paneeli ja otsikko, jotka näkyvät samassa muodossa kahdessa aloituspaneelissa ja päänäkymässä. Toinen aloituspaneeli näkyy kuvassa 4.



Kuva 4. Listat on luokiteltu ajoneuvon tyypin mukaan. Jokaisella kategorialla on oma, dynaaminen vieritettävä listansa.

Yksi SAM-hankkeen päätarkoituksista on selvittää, kuinka montaa ajoneuvoa yksi ihminen pystyy valvomaan kerrallaan, joten tarvitaan dynaaminen ajoneuvolista. Tämä tarkoittaa sitä, että listattujen ajoneuvojen määrää pitää voida vähentää ja lisätä. Tämän voisi toteuttaa kahdella eri tavalla. Ensimmäinen tapa olisi tehdä yksi tekstikenttä, jossa jokainen ajoneuvo on omalla rivillään. Toinen tapa on tehdä useampi tekstikenttä, joista jokainen vastaa yhtä ajoneuvoa. Työssä nähtiin toinen tapa oikeastaan kaikilla tavoilla paremmaksi ja helpommaksi. Dynaaminen selattava lista erillisiä tekstikenttiä on oikeastaan aika helppo tehdä, sillä se tarvitsee vain Unityn valmiita UI-komponentteja. Ohjelmointia tarvitsee tällaisessa listassa vain, jos haluaa vaihtaa listan sisältöä suoritusaikana.

Vaikka dynaaminen selattava lista on helppo tehdä, sen rakenne voi olla hieman sekavan tai monimutkaisen oloinen. Dynaamista selattavaa listaa tarvitaan useammassa osaa tätä insinöörityötä, joten käydään se läpi huolellisesti. Listan tärkein ja monimutkaisin osa on ”vieritettävä lista” -komponentti (engl. Scroll Rect). Vieritettävän listan voi luoda, kun luo uuden peliobjektin ja valitsee vieritettävän näkymän (engl. Scroll View) UI-alavalikosta.

Tämä vieritettävä näkymä -peliobjekti luo myös muutaman peliobjektin lapsekseen, jotka ovat kaksi kahvaa, joilla voi liikuttaa näkymää horisontaalisessa ja vertikaalisessa suunnassa. Lisäksi vieritettävä näkymä luo ”viewport”-nimisen peliobjektin, jonka lapsena on ”Content”-niminen peliobjekti. Viewport-peliobjekti tarkoittaa aluetta, jonka läpi näkee näkymän sisällön. Content-peliobjekti on taas itse sisältö, jonka halutaan olevan vieritettävässä näkymässä.

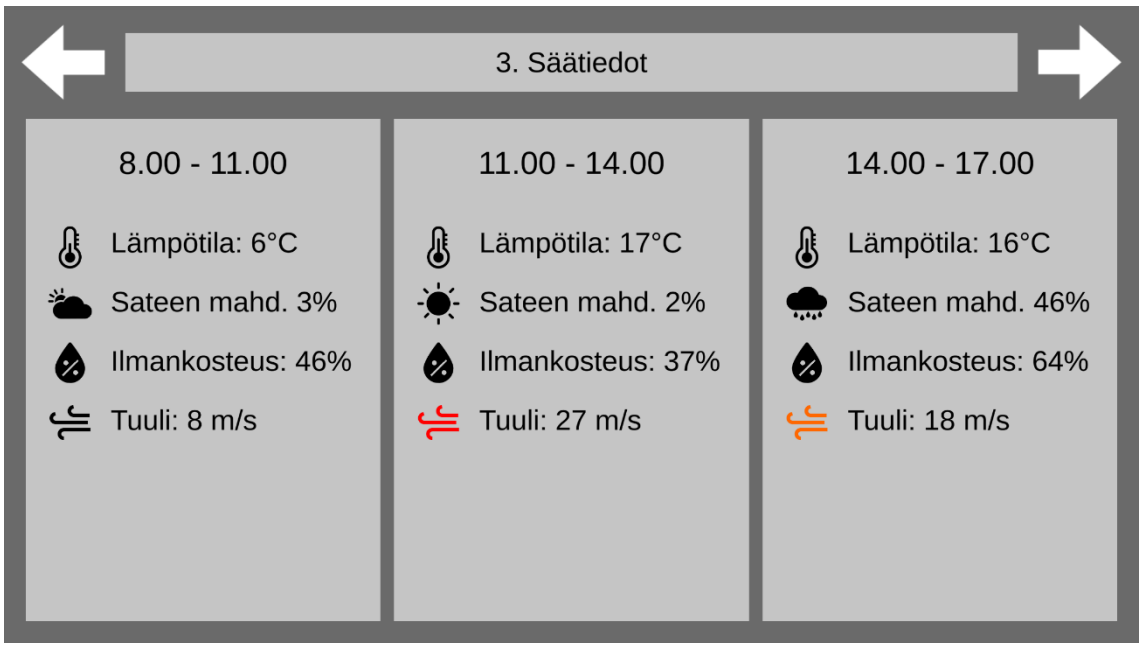
Äsken luotu vieritettävä näkymä tarvitsee seuraavaksi hieman muutoksia, jotta siitä saa vertikaalisesti vieritettävän dynaamisen listan. ”Scroll view” -peliobjektin ”Scroll rect” -komponentissa tulee poistaa horisontaalinen liikkuvuus, jotta lista liikkuu ainoastaan ylös ja alas. Lisäksi haluttiin, että vertikaalinen vierityspalkki tulee esille vain silloin, jos kaikki listan sisältö ei mahdu yhdelle sivulle ja horisontaalisen vierityspalkin voi piilottaa kokonaan. ”Scroll rect” -komponentissa ”Horizontal Scrollbar” -asetuksen ”Visibility”-asetus tulee asettaa ”Permanent”-asetukselle. ”Viewport”-peliobjektiin ei tarvitse tehdä muutoksia. ”Content”-peliobjektiin pitää lisätä kaksi uutta komponenttia. ”Vertical Layout Group” on komponentti, joka asettelee käyttöliittymän objektit tasaisesti halutulle alueelle vertikaalisessa suunnassa.

”Padding”- ja ”Spacing”-asetuksia pitää aina säätää hieman käyttötarkoituksen mukaan, jotta sisältö näyttää hyvältä ja symmetriseltä. Lopuksi tulee vielä lisätä ”Content Size Fitter” -komponentti ”Content”-peliobjektiin, jotta vieritettävän näkymän koko muuttuu automaattisesti sopivan kokoiseksi. ”Content Size Fitter” -komponentti toimii huomattavasti paremmin täyssä käytössä, kun ”Horizontal Fit”- ja ”Vertical Fit” -asetukset laitetaan ”Preferred Size” -asetukseen. Nyt ”Content”-peliobjektin lapseksi voi laittaa mitä tahansa käyttöliittymäelementtejä, kuten tekstikenttiä, ja vieritettävä näkymä muuttuu aina juuri oikean kokoiseksi. Myös vieritysrulla mukautuu sopivan kokoiseksi, tai se katoaa kokonaan, mikäli kaikki käyttöliittymän osat näkyvät kohdealueella.

4.3 Säätiedot

Yksi autonomisten ajoneuvojen valvonnassa oleellinen asia on sää. Jäisellä asfaltilla ajoneuvot ovat alttiimpia kolareihin tai ulosajoihin, ja autonomisen ajoneuvojen sensorit saattavat peittyä esimerkiksi lumella, mikä altistaa monenlaisiin ongelmiin. Tämän takia on todella tärkeää, että valvontatyökalu sisältää oman osan tiedottamaan sääennustetta ja nykyistä säätä. Koehenkilöiden rasituskokeen käsikirjoituksen mukaan testin aikana nousee kuvitteellinen myrsky, joka pitää saada viestittyä käyttäjälle. Luvussa 5 näemme, että itse sovelluksen päänäkymässä on oma pieni sääosuutensa, mutta myös alkupaneeleissa on hyvä tiedottaa, millainen sää on tulossa.

Jatkokehitysideana saatetaan hankkia oikea sääpaneeli esimerkiksi Forecalta. Tässä vaiheessa projektia halutaan kuitenkin pitää koehenkilöiden kokeessa aina käsikirjoituksen mukainen sää, joten reaaliaikainen sääennuste ei ole mahdollinen. Yksimielinen ratkaisu oli erotella muutaman tunnin välein sään päivitys. Tätä varten säätiedot-paneelissa on muutama erillinen palkki, jossa näkyy kellonaika ja säätietoja. Oleellisiksi säätiedoiksi nähtiin lämpötila, sateen mahdollisuus (tässä tulee myös ilmi esimerkiksi, onko pilvistä tai aurinkoista), ilmankosteus ja tuulennopeus. Pieni informaation määrää lisäävä ja käyttökokemusta parantava idea oli lisätä kullekin säätiedon kohdalle symboli. Esimerkiksi lämpötilan kohdalla on merkki lämpömittarista, josta on muutama eri versio sen mukaan, onko lämpötila korkea tai matala. Sateen mahdollisuuden symboli esittää hyvin sen, onko taivaalla kirkas aurinko vai sadepilviä. Säätietojen symbolit voidaan myös värittää. Tarkoituksena on näin korostaa kohokohtia, esimerkiksi poikkeuksellisen korkeissa tuulissa esiintyvä punainen tuulisymboli kiinnittää käyttäjän huomion tehokkaasti. Säätiedot esittävä paneeli on kuvassa 5.



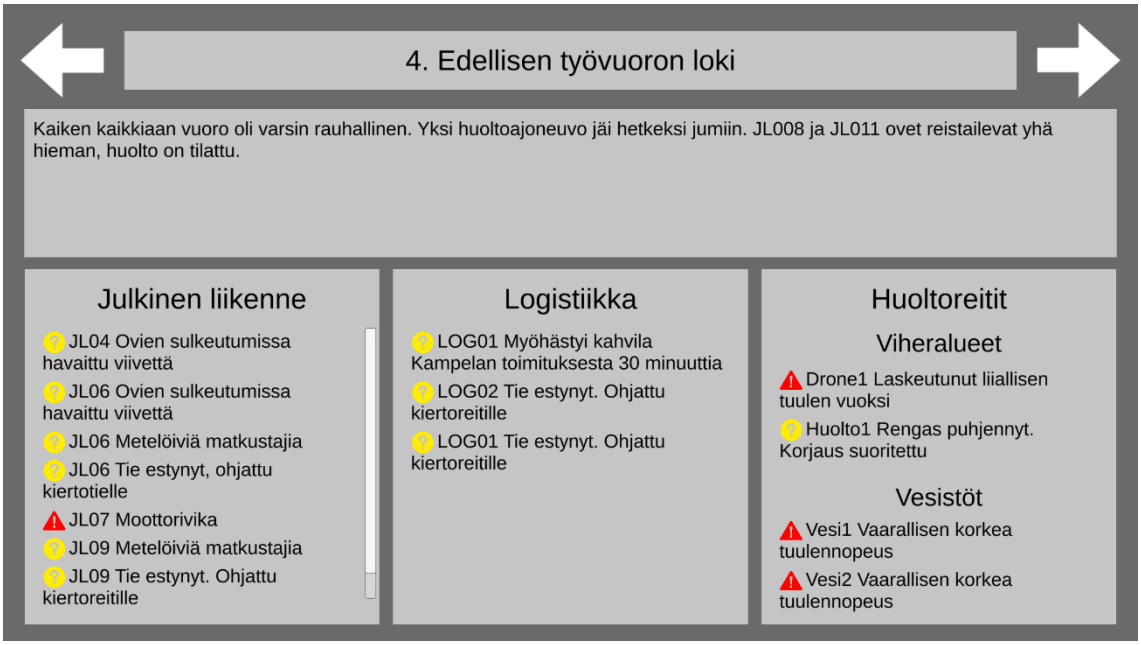
Kuva 5. Säätiedot-aloituspaneeli antaa hyvän yleiskuvan siitä, millainen sää tulee olemaan työvuoron aikana. Poikkeuksellisen rajut sääolosuhteet, kuten myrskytuulet, on helpompi havaita värikoodauksen ansiosta.

Säätietopaneeli ei tarvinnut yhtään ohjelmointia, pois lukien sivujenvaihtojärjestelmän. Paneelin rakenne on hyvin samankaltainen muidenkin paneelien kanssa. Yläotsikko ja napit ovat identtisiä jokaisessa aloituspaneelissa (pois lukien nappien funktiokutsut ja otsikon teksti). Sääpalkit saadaan järjesteltyä symmetrisesti ”Horizontal Layout Group” -komponentilla. Tällöin on myös todella helppo lisätä tai vähentää sääpalkkien määrää. Jokainen sääpaneeli on kopioitu samasta pohjasta, jossa aikamääre on erillinen elementti. Itse säätiedot on ”Vertical Layout Groupin” avulla aseteltu sopivan väljiksi ja symmetrisiksi. Sääsymbolien kohdalla tärkeä huomio on, että lähteenä käytetyn kuvan väri on valkoinen. Tällöin sääsymbolien väriä on helppo vaihtaa suoraan inspector-näkymästä tai koodista (koodiesimerkki 5 sivulla 34). Vakiona sääsymbolit on värjätty mustaksi inspector-näkymästä, koska musta erottuu hyvin valkoiselta taustalta. Myös korostusväreinä käytetty oranssi ja punainen erottuvat selkeästi taustasta.

4.4 Edellisen työvuoron loki

Neljännen aloituspaneelin tarkoitus on tiedottaa käyttäjälle edellisen työvuoron tapahtumia ja luonnetta. Paneeli koostuu edellisen työntekijän vapaista muistiinpanoista, joita hän on voinut kirjoittaa työvuoron aikana ja vuoron lopettamiseen tarkoitetussa paneelissa. Neljännessä aloituspaneelissa tulee lisäksi lista kaikista keltaisen ja punaisen asteen ilmoituksista, joita esiintyi edeltävällä työvuorolla. Simulaation käsikirjoitetun kokeen aikana sekä muistiinpanot että edellisen vuoron ilmoitukset ovat aina samat. Tämä johtuu siitä, että voidaan luoda uskottavaa taustatarinaa käyttäjäkokeelle. Lisäksi edellistä työvuoroa tai koetta ei välttämättä ollut tai se oli merkittävästi erilainen. Kokeiden jälkeen on kuitenkin tarkoitus hakea ilmoitukset automaattisesti.

Työntekijän tarvitsee nähdä muistiinpanoja ja ilmoituksia edelliseltä työvuorolta siitä syystä, että jotkin tapahtumat voivat vaikuttaa myös tulevaan työvuoroon. Esimerkiksi, jossain ajoneuvoissa voi olla toistuvia ongelmia vaikkapa ovien kanssa. Tällöin alkavan vuoron työntekijä pystyy kiinnittämään tähän ongelmaan huomiota ja tekemään huoltopyynnön (tai muita toimenpiteitä), mikäli ongelma jatkuu. Mahdollisuus vapaisiin muistiinpanoihin on myös tärkeää tästä syystä. Edellisen vuoron työntekijä on voinut kiinnittää huomiota poikkeukseen, mutta päättänyt seurata tilannetta ennen jatkotoimenpiteitä. Edellisen työvuoron lokipaneeli on kuvassa 6.



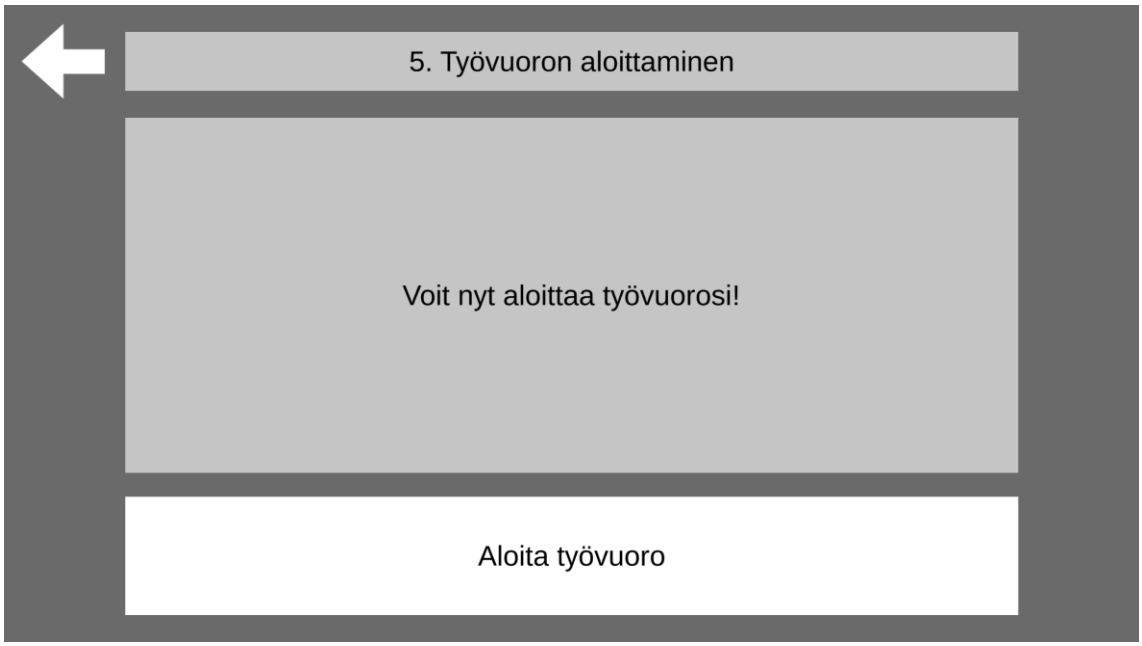
Kuva 6. Otsikon jälkeen nähdään edellisen työntekijän vapaat muistiinpanot. Suurimman osan paneelin tilasta vievät listat ajoneuvokategorioiden keltaisista ja punaisista ilmoituksista.

Tämän aloituspaneelin toteutuksessa käytetyt tekniikat ovat pitkälti samoja kuin edellisissäkin paneeleissa. Muistiinpanot ovat omassa tekstikentässään yksinkertaisessa staattisessa paneelissa. Kategoriat ovat jälleen aseteltu ”Horizontal Layout Groupin” avulla tasaisesti. Kategorioiden sisällä olevat dynaamiset vieritettävät listat toteutettiin samalla tavalla kuin aloituspaneelissa 2 (ks. luku 4.2).

4.5 Työvuoron aloittaminen

Viides eli viimeinen aloituspaneeli on ikään kuin aulatila itse sovelluksen päänäkymään. Mikäli useampi valvoja on valvomassa ja ohjaamassa ajoneuvoja, voi syntyä useita eri ongelmia. Esimerkiksi jos kaksi eri valvojaa yrittää korjata ongelmatilannetta samanaikaisesti eri tavoilla, ajoneuvon ongelma voi vain pahentua ongelman korjaantumisen sijaan. Halutaan siis, että valvontatyökalussa vain yksi käyttäjä kerrallaan voi hallita autonomisia ajoneuvoja. Tämän takia tarvitaan odotustila, josta pääsee päänäkymään vasta, kun edellinen työntekijä on lopettanut työvuoronsa. Kun edellinen työntekijä on

lopettanut vuoronsa, odotustilassa odottavalle seuraavalle työntekijälle aktivoituu nappi, josta pääsee päänäkymään valvomaan ja ohjaamaan ajoneuvoja. Viimeinen aloituspaneeli on kuvassa 7.



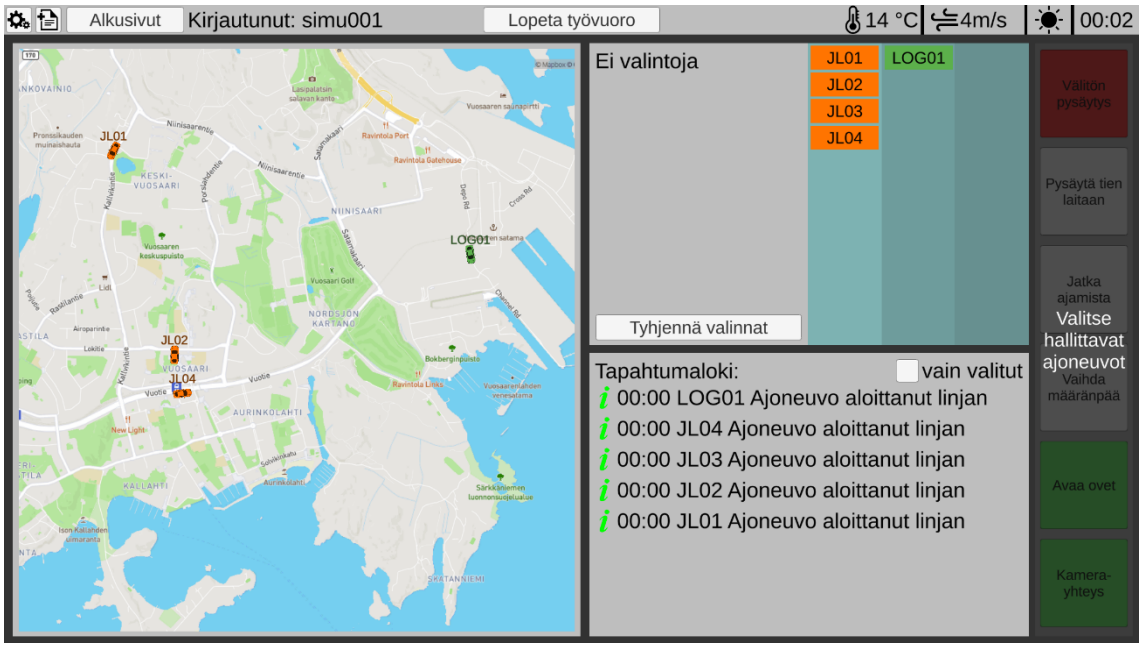
Kuva 7. Työvuoron aloituspaneeli, kun edellinen valvoja on lopettanut vuoronsa.

Paneeli on rakenteeltaan hyvin yksinkertainen. Aiempien aloituspaneelien tapaan ylhäällä on paneelin otsikko ja nappi edelliseen paneeliin. Tässä sivussa ei ole nappia seuraavaan paneeliin, koska seuraavaa paneelia ei ole. Paneelin keskellä on vain iso tekstikenttä, jossa lukee ”Odotetaan edellisen valvojan lopettavan työvuoronsa”. Kun edellinen valvoja lopettaa työvuoronsa, paneelin teksti muuttuu ja tekstikentän alapuolelle ilmestyy iso nappi työvuoron aloittamiseen. Koevaiheessa edellisen työntekijän vuoron lopettaminen korvattiin yksinkertaisesti tietyn sekuntimäärän odotuksella. Mikäli työntekijä palaa vielä aloituspaneeleihin myöhemmin, tekstiksi vaihtuu ”Voit nyt jatkaa työvuoroa” ja napissa lukee ”Jatka työvuoroa”.

5 Päänäkymän toteutus

Aloituspaneelien jälkeen käyttäjä pääsee itse valvomisen päänäkymään. Päänäkymä on sovelluksen tärkein paikka, sillä se viestii käyttäjälle jokaisen

ajoneuvon sijaintia ja tilaa sekä antaa käyttäjälle mahdollisuuden tehdä ajoneuvoihin liittyviä valintoja ohjauspaneelin kautta. Uusimman version päänäkymästä näkee kuvassa 8.



Kuva 8. Sovelluksen päänäkymä. Vasemmalla on kartta, josta näkee ajoneuvojen sijainnin. Kartan oikealla puolella ovat ajoneuvolistat ja tapahtumaloki. Aivan oikeassa laidassa on ohjauspaneeli. Sovelluksen yläreunassa on niin sanottu yläpaneeli.

Päänäkymä sisältää useita erimuotoisia ja erikokoisia elementtejä. Esimerkiksi kartalle halutaan enemmän tilaa kuin muille osille. Tämän takia ”Layout Group” ei ole sopiva komponentti tässä käytössä. ”Layout Group” antaa tärkeimpänä etuna helpon asettelun ja symmetrisyyden, mikäli elementtien määrää vaihdetaan. Päänäkymän elementit ovat kuitenkin ennalta suunniteltuja, joten ”Layout Group” -komponentista ei olisi mitään hyötyä. Parempi vaihtoehto on manuaalisesti luoda, asetella ja skaalata jokainen paneeli sopivaksi. Symmetrisyys onnistuu yhä helposti seuraamalla ”Rect Transformien” kokoa ja käyttämällä Unityn ”Snap”-ominaisuutta.

5.1 Karttanäkymä

Päänäkymän pohjan valmistuttua oli seuraava looginen askel lähteä toteuttamaan jokaisen elementin sisältöä ja toimintoja. Kartan haluttiin olevan interaktiivinen, eli käyttäjän tulisi voida zoomata ja liikutella kartan näkymää. Kartta on keskitetty alueelle, jossa autonomiset ajoneuvot liikkuvat. Tämä alue on merkitty oransseilla reunoilla. Alueen ulkopuolista karttaa voi myös nähdä, kun zoomaa taakse. Ulkopuolista karttaa näkee myös, jos vierittää karttaa tarpeeksi reunoille, mutta tällöin näkymä vierii takaisin oleelliselle alueelle. Kartat saadaan avoimelta yhteistyöprojektilta nimeltä ”OpenStreetMaps”. OpenStreetMapsilta saa helposti hyvälaatuiset kartat ilmaiseksi. Lisäksi OpenStreetMapsilla on paljon integrointimahdollisuuksia, ja tämänkin hankkeen ajoneuvosimulaatiosta vastaava SUMO toimii OpenStreetMapsin karttojen avulla. (8.)

Hyväntuntuinen ja intuitiivisesti liikuteltava karta onnistuu helpoiten käyttämällä jälleen vieritettävää näkymää. Tässä tarkoituksessa vieritettävä näkymä jätetään liikuteltavaksi sekä horisontaalisessa että vertikaalisessa suunnassa. Tällöin karttaa voi liikuttaa mihin tahansa suuntaan niin, että näkee myös alueet, jotka ovat jääneet zoomaamisen hetkellä ulkopuolelle.

Vieritettävän näkymän ”Mask”-komponentti on ikään kuin ikkuna, joka näyttää sen lapsena olevan sisällön. Tämän takia ”MapMaskiksi” nimetty peliobjekti skaalataan siihen kohtaan ja muotoon, missä halutaan kartan näkyvän päänäkymässä. ”MapMask”-peliobjektin lapsena on toinen peliobjekti, joka kantaa kuvakomponenttia, jonka lähdekuvana on korkealaatuinen kartta ajoneuvojen toiminta-alueesta. Lisäksi tässä peliobjektissa on skripti, joka käsittelee kartan zoomaamisen. Kartan zoomaamista käsittelevä skripti sisältää muuttujat zoomauksen aloituskokoon, zoomauksen ylä- ja alarajaan ja zoomausnopeudelle. Lisäksi skriptissä on bool-muuttuja, joka seuraa, onko käyttäjän hiiri kartan päällä.

Zoomausta rajoittavat muuttujat ovat suojaamassa käyttäjiä itseltään tai vahingoilta. Jos zoomaa liikaa ulos, kartan ulkopuolella näkyy vain mustaa

reunaa eikä ajoneuvoista saa enää mitään selkoa. Jos zoomaa liikaa sisään, ei näe oikein mitään. Syyt zoomauksen rajoittamiseen on esitetty kuvassa 9.

Kuva 9. Vasemmalla on demonstraatio liiallisesta sisään zoomaamisesta ja oikealla liiallisesta ulos zoomaamisesta. Tavallisesti sovelluksessa on paljon rajatummat arvot zoomaamiseen.

Zoomaamista käsittelevä skripti lukee jatkuvasti syötettä hiiren rullasta. Mikäli rullasta tulee jokin muu arvo kuin 0 ja hiiri on kartan päällä, voidaan kutsua ChangeZoom() -funktiota. Tämä toimintaperiaate näkyy vielä koodiesimerkissä 2.

```
private void ChangeZoom(float scrollWheel)
    {
        float rate = 1 + zoomRate * Time.unscaledDeltaTime;
        if (scrollWheel > 0)
        {
            SetZoom(Mathf.Clamp(transform.localScale.y / rate,
minSize, maxSize));
        }
        else
        {
            SetZoom(Mathf.Clamp(transform.localScale.y * rate,
minSize, maxSize));
        }
    }

    private void SetZoom(float targetSize)
    {
        transform.localScale = new Vector3(targetSize, targetSize, 1);
    }
```

Esimerkkikoodi 2. Zoomauksesta vastaavat funktiot. Perusidea on se, että karttaa skaalataan X- ja Y-akseleilla hiiren rullan syötteen mukaan.

Kartta koostuu useammasta eri kuvasta. Zoomausta käsittelevä skripti on pääkartassa, jonka lapsena ovat loput kartat. Tämän seurauksena lapsiobjekteina olevat muut kartat perivät pääkartan skaalan eli kaikki kartat zoomaantuvat identtisesti pääkartan kanssa. Laivueen alueen merkitsevät oranssit reunat toimivat samalla periaatteella. Reunat koostuvat yksinkertaisesti neljästä manuaalisesti asetellusta tyhjästä kuvasta. Koska lähdekuvat ovat tyhjänä täysin valkoisia, niiden väri on helppo määritellä sopivaksi inspector-näkymästä.

5.2 Tapahtumaloki ja ponnahdusikkunat

Kun simulaatiossa ajoneuvoille tapahtuu jotain, se viestitään käyttäjälle tapahtumalokin tai ponnahdusikkunoiden avulla. Tapahtumalokille on oma alueensa päänäkymässä. Ponnahdusikkunat tulevat osittain kartan päälle vasempaan yläkulmaan. Tapahtumalokin tulee olla vieritettävä, dynaaminen ja lajiteltu niin, että uusin viesti on ylimpänä. Tapahtumalokissa on myös nappi asetukselle, joka näyttää vain valittuihin ajoneuvoihin liittyvät tapahtumat. Ponnahdusikkunoita pitää voida olla useampia näkyvillä samaan aikaan, ja niiden pitää sisältää nappi, joka hyväksyy ja poistaa ponnahdusikkunan.

Tapahtumaloki

Tapahtumalokille on varattu sopiva, manuaalisesti rajattu alue oikeassa alakulmassa päänäkymää. Tapahtumalokin alueen yläreunassa on otsikko, joka ilmaisee alueen olevan tapahtumaloki. Otsikon lisäksi yläreunassa on nappula, joka määrittelee, mikäli lokissa tulisi näkyä vain valittujen ajoneuvojen tapahtumat. Todellisuudessa tämä nappula aina poistaa kaiken näkyvästä listasta ja luo uudelleen halutut viestit listasta, joka pitää kirjaa kaikista viesteistä, joita suoritusaikana on tapahtunut. Tämä näkyy koodiesimerkissä 3.

```
private void Toggle()
    {
        onlySelectedVehicle = selectedVehicleToggle.isOn;
        ClearLog();
        GetEventMessages();
    }
    // Haetaan kaikki valintojen mukaiset viestit listasta
    public void GetEventMessages()
    {
        ClearLog();
        foreach (EventLogMessage message in eventLogMessages)
        {
            if (onlySelectedVehicle == true &&
!selection.selections.Contains(message.carId) && message.carId != ""
)
            {
                continue;
            }
            InstantiateLogMessage(message.carId, message.message,
message.importance);
        }
    }
```

Koodiesimerkki 3. Tapahtumalokin nappulan toiminnon tärkeimmät funktiot. Nappula kutsuu Toggle-funktiota. InstantiateLogMessage() luo näkyvään tapahtumalokiin uuden tapahtuman.

Itse tapahtumalokin visuaalinen lista on toteutettu tutulla dynaamisella vieritettävällä listalla. Tällä kertaa siinä on kuitenkin yksi pieni muutos. Halutaan, että listan uusin ilmoitus ilmestyy aina ylimmäksi listaan. Tähän tarkoitukseen on ”Layout Group” -komponenteissa valmis asetus nimeltä ”Reverse Arrangement”. Asetuksen ollessa päällä ”Layout Group” esittää aseteltavana olevat peliobjektit käänteisessä järjestyksessä. Kun tapahtumalokiin lisätään uusi tapahtuma, lisätään uusi peliobjekti, jossa on tekstikenttä. Tämä uusi peliobjekti on ”Vertical Layout Groupin” vaikutuksen alla, ja ”Reverse

Arrangement” -asetuksen avulla tämä uusi peliobjekti menee heti listan päällimmäiseksi. Tämä on hyödyllinen järjestely, jotta käyttäjän on helppo nähdä aina uusin ilmoitus listan ylimpänä tapahtumana eikä sitä tarvitse etsiä muiden tekstien seasta. Tämä nopeuttaa merkittävästi käyttäjän mahdollisuutta toimia ja tekee käytöstä paljon mukavampaa.

Ponnahdusikkunat

Käyttöliittymä on aika täysi eikä ylimääräistä tilaa ole, joten ponnahdusikkunoiden kohdalla ensimmäiseksi piti päättää, minne ponnahdusten tulisi ilmestyä ja kuinka iso ikkunan tulisi olla. Loogisin paikka ponnahdusikkunoille päätyi olemaan vasemmalla ylhäällä kartan päällä. Tähän on kaksi pääsyytä. Ensinnäkin vasen yläkulma on monesta muusta sovelluksesta tuttu paikka uusille ponnahdusilmoituksille. Toiseksi mikään tärkeä nappi tai tekstikenttä ei peity niin, että sitä ei voisi nähdä tai käyttää. Tietenkin osa kartasta peittyy ponnahdusikkunalla, mutta karttaa voi liikuttaa niin, että ponnahdusikkunan piilottama alue näkyy jälleen. Kuva 10 esittää karttanäkymän, joka peittyy osittain ponnahdusikkunalla.

Kuva 10. Karttanäkymä ponnahdusikkunalla. Mikäli ponnahdusikkunoita on useampia, ne kasaantuvat alkuperäisen ponnahdusikkunan alapuolelle.

Ponnahdusikkunat toimivat aika yksinkertaisesti. Luodaan uusi peliobjekti prototyyppiobjektin (engl. Prefab) pohjalta. Uusi peliobjekti asetetaan valmiiksi luodun peliobjektin lapseksi, joka määrittelee uudelle ponnahdusikkuna-peliobjektille sopivan sijainnin käyttäen ”Rect Transform” -asetuksia. Tämä tulee tarkemmin ilmi koodiesimerkissä 4. Ponnahdusikkunoita pitelevä peliobjekti sisältää myös ”Vertical Layout Group” -komponentin, joka tekee listasta dynaamisen niin, että ponnahdusikkunoita voi olla useampia samanaikaisesti ilman, että ne menevät päällekkäin. Vieritettävää näkymää ei tällä kertaa tarvita,

sillä ponnahdusikkunoita on yleensä 0–2 samanaikaisesti, eli kaikki ovat näkyvissä ilman vierittämisen tarvetta. Tämä järjestely mahdollistaa myös sen, että ponnahdusikkunoissa oleva nappi voi muiden toimenpiteiden jälkeen tuhota koko ponnahdusikkuna peliobjektin. Tällöin ”Vertical Layout Group” tuo seuraavat ponnahdusikkunat ylöspäin sopivalle sijainnille.

```
public GameObject PopupMessage(string description)
    {
        GameObject newPopup = Instantiate(popupGO);
        newPopup.transform.SetParent(popupContainer);
        newPopup.transform.localScale = new Vector3(1, 1, 1);

        TextMeshProUGUI popupText =
newPopup.GetComponentInChildren<TextMeshProUGUI>();
        popupText.text = description;

        return newPopup;
    }
```

Koodiesimerkki 4. Uuden ponnahdusikkunan luominen.

Uusi ponnahdusikkuna voidaan luoda yksittäisellä funktiokutsulla, jolle annetaan parametriksi teksti, joka halutaan ponnahdusikkunan sisälle. ”PopupMessage()”-funktio kutsun kolmas rivi asettaa ponnahdusikkuna-peliobjektin skaalaksi 1 jokaiselle akselille. Tämä korjaa ongelmia, joita tulee, kun rivillä 2 uusi peliobjekti asetetaan toisen lapseksi. Mikäli vanhempana toimivan peliobjektin skaalaa tai muita asetuksia säädetään, se aiheuttaa uuden ponnahdusikkuna-peliobjektin perimään näitä asetuksia, mikä usein aiheutti ponnahdusikkunan näkymään väärässä sijainnissa tai koossa. Toinen huomion arvoinen asia on se, että projektin jokainen tekstikenttä on tehty käyttäen TMP:tä (engl. Text Mesh Pro), koska sillä saa huomattavasti parempilaatuista tekstiä ja enemmän tekstiin liittyviä asetuksia. TMP on huomattava parannus Unityn alkuperäiseen tekstinkäsittelyratkaisuun, joten se on integroituna oletuksena Unityyn uudemmissa Unity-versioissa.

Ponnahdusikkunan prototyyppiobjekti on aika yksinkertainen rakenteeltaan. Siinä on kaksi taustapaneelia, tekstikenttä ja nappi. Taustapaneeleita on kaksi, joista taaempi on iso ja tumma. Etummaisempi taustapaneeli on pienempi ja vaalea. Täten saadaan ponnahdusikkunalle reuna, joka näyttää paljon paremmalle ja ikkunan erottaa helposti muusta näkymästä.

Prototyyppiobjektissa on kiinni oma pieni skriptinsä, jossa on funktio koko ponnahdusikkunan poistamiseksi. Ponnahdusikkunassa oleva nappi kutsuu tätä funktiota. Lisäksi skriptin ”Awake()”-funktio sisältää monimutkaista koodia, joka varmistaa, että ikkunan napista aiheutuvat ääniefektit kutsutaan sovelluksen ainoasta äänilähteestä (äänistä lisää luvussa 5.4.5).

Yksi parannusidea suorituskyvyn kannalta olisi käyttää objektivarastokuviota (engl. Object Pooling) eli luoda ikkunat valmiiksi piiloon ja siirtää ne oikeaan paikkaan oikealla tekstillä, kun niitä tarvitsee. Tämä parantaisi suorituskykyä, sillä uusien objektien luominen on suorituskyvyltään raskasta ja valmiiksi luotujen objektien siirtäminen tai muokkaaminen on kevyttä. (9.)

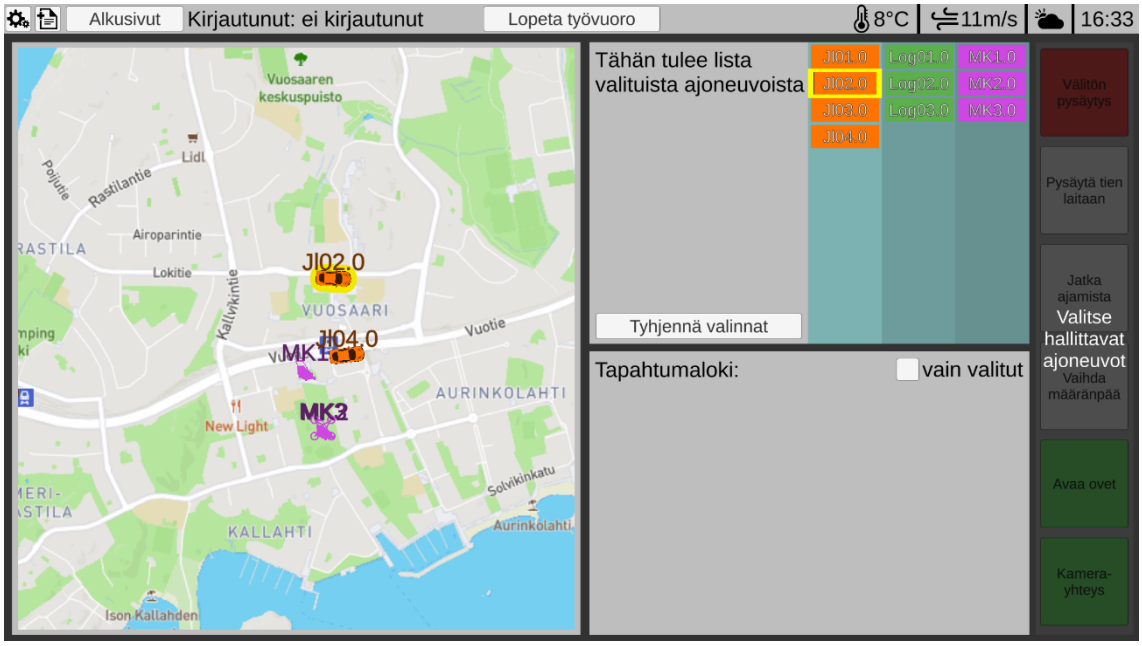
5.3 Ajoneuvolistat, kartan symbolit ja ajoneuvojen valinta

Ajoneuvot ovat kolmessa eri kategoriassa, ja jokaiselle kategorialle on oma listansa. Listan lisäksi jokaisella ajoneuvolla on kartalla liikkumassa oma symbolinsa, joka ilmaisee ajoneuvon sijainnin, nimen ja ajoneuvotyypin ja mahdollistaa ajoneuvojen valitsemisen. Ajoneuvolistat, kartalla liikkuvat ajoneuvon symbolit ja ajoneuvojen valintajärjestelmä liittyvät kiinteästi toisiinsa, joten on loogista käsitellä ne samassa luvussa.

Ajoneuvolistat ovat kolme dynaamista vieritettävää listaa, eli jokaiselle kategorialle on oma. Listat ovat aluksi tyhjiä, mutta kun ajoneuvo luodaan, sille luokitellaan muun muassa ajoneuvokategoria. Kun uuden ajoneuvon kategoria tiedetään, se on helppo lisätä oikeaan listaan. Lisäksi kategorian mukaan listassa olevalle napille ja kartalla pyörivälle symbolille voidaan määritellä sama väri, mikä helpottaa ajoneuvojen tunnistamista. Uuden ajoneuvon luonnissa lisätään myös kartalle uusi peliobjekti, jossa on kuva- ja nappikomponentit. Tämä kartan ajoneuvo yhdistetään SUMOon, joka liikuttaa sitä simulaation mukaisesti.

Kun uudesta ajoneuvosta on nappi listassa ja kartalla, ne tarvitsevat yhteyden toisiinsa. Pääsyy yhteyden tarpeeseen on valitsemisjärjestelmä. Kun käyttäjä vie hiirensä listalla tai kartalla olevan ajoneuvonapin päälle, molempien nappien

tulee korostua värillä, jotta käyttäjä näkee selkeästi, mitä on valitsemassa. Lisäksi käyttäjä voi helposti löytää tietyn ajoneuvon kartalta viemällä hiirensä listassa olevan napin päälle, sillä kartalla oleva vastaava auto korostuu selkeästi. Tämä on helppoa toteuttaa hyödyntämällä ajoneuvojen tunnistetta. Jokaisella ajoneuvolla on oma uniikki kirjaimista ja numeroista koostuva tunniste. Kun nappi luodaan ja sille etsitään oikea kartan symboli, oikean kartan symbolin löytää etsimällä niitä pitävän peliobjektin lapset läpi. Jokaisen lapsi-peliobjektin kohdalla tarkistetaan, vastaako tämän peliobjektin nimi haluttua tunnistetta, kunnes oikea ajoneuvo löytyy. Ajoneuvojen ja ajoneuvolistan nappien korostuminen on näytetty kuvassa 11.



Kuva 11. Ajoneuvon korostuminen, kun hiiren vie ajoneuvolistan napin tai kartan symbolin päälle. Kuvassa käyttäjä pitää hiirtä ajoneuvolistan päällä ja vastaava ajoneuvo korostuu myös kartalla.

Kun ajoneuvojen karttasymbolit ja listan napit on luotu ja yhdistetty, on helppoa toteuttaa valintajärjestelmä. Ajoneuvon voi valita painamalla sen karttasymbolia tai nappia listalta. Mikäli haluaa valita useamman ajoneuvon, käyttäjä voi painaa Vaihto- tai CTRL-näppäimen pohjaan ja valita useamman ajoneuvon. Vaihto- tai CTRL-näppäimen painaminen myös mahdollistaa ajoneuvon valinnan poiston, mikäli on valinnut useamman ajoneuvon ja haluaa poistaa osan valinnoista.

Valinnat voi tyhjentää myös B-näppäimestä tai käyttöliittymässä olevasta ”tyhjennä valinnat” -napista.

Valintajärjestelmä pitää listaa valittujen ajoneuvojen tunnisteesta, ja toinen lista pitää listaa valittujen ajoneuvojen kartan peliobjekteista. Aina kun käyttäjä muuttaa valintojaan, kutsutaan omaa funktiota, joka päivittää korostukset ja luettelon valituista ajoneuvoista. Tämä funktio myös aktivoi ohjauspaneelin piilottavan läpinäkyvän palkin, joka estää ohjauspaneelin käytön. Ohjauspaneeli määrittelee käskyt valituille ajoneuvoille, eli sitä ei pidä voida käyttää, mikäli mitään ajoneuvoa ei ole valittu. Kun käyttöliittymän käyttämistä halutaan rajoittaa päälle laitettavilla palkeilla, on tärkeä tarkistaa, että päällä olevassa palkissa on ”Raycast Target” -asetus päällä. Tämä asetus tarkoittaa sitä, että kyseinen käyttöliittymän osa ottaa huomioon hiirestä tulevan säteenseurannan ja täten myös estää sen takana olevia käyttöliittymäelementtejä toimimasta hiiren säteenseurannasta. (10.) Takana oleviin käyttöliittymäelementteihin voi silti yhä päästä käsiksi käyttöliittymän navigaatiojärjestelmällä, mikäli asiaa ei huomioi.

5.4 Muut päänäkymän osat

Tärkeimmät päänäkymän osat on nyt käsitelty edeltävissä luvuissa. Päänäkymässä on kuitenkin vielä useita pieniä ja yksinkertaisia, mutta tärkeitä elementtejä, joita on syytä käsitellä. Esimerkiksi ohjauspaneeli, asetukset ja yläpaneelin sisältö saavat tässä luvussa selvityksen. Ohjauspaneeli on toistaiseksi hieman keskeneräinen, sillä ei vielä tiedetä tarkalleen, mitä kaikkia käskyjä käyttäjän tulee voida määritellä ajoneuvoille. Asetukset sisältävät toistaiseksi vain äänenvoimakkuuden säätimen, sillä simulaation kokeiluihin ei haluta ylimääräisiä tekijöitä, jotka saattaisivat harhauttaa käyttäjää tai aiheuttaa muita mahdollisia ongelmia.

Päänäkymän säätiedot ja kello

Päänäkymän oikeassa yläreunassa on tunnit ja minuutit ilmaiseva digitaalinen kello. Kellon vasemmalla puolella ovat järjestyksessä yleisen säätilan ilmaiseva

symboli, tuulen nopeus symbolilla ja lämpötila symbolilla. Hieman paremman selkeyden vuoksi jokaisen kohdan erottelee pieni musta palkki. Jokainen näistä alueista pysyy karkeasti samankokoisena, eli jokaisen alueen voi määritellä ennalta sopivan kokoiseksi ja sopivaan kohtaan. Itse symbolit ja numerot voivat vaihtua, mutta ne pysyvät aina lähes samankokoisena.

Simulaation kokeisiin halutaan joka kerta samanlainen sää. Tämän takia säätiedot pitää voida määritellä etukäteen käsikirjoituksen mukaan. Simulaation kokeiden käsikirjoitusta ohjaa yksittäinen skripti, joka pitää kirjaa ajasta sekuntimäärän mukaan. Tämän takia voidaan määritellä helposti, että tietyn sekuntimäärän jälkeen vaihdetaan säätietoja halutulla tavalla. Symbolien eli peliobjektin kuvakomponentin lähdekuva on helppoa vaihtaa koodista. Lisäksi jokainen symbolien lähdekuva on täysin valkoinen, jotta niiden väriä on helppo vaihtaa suoraan koodista tai inspector-näkymästä koodiesimerkin 5 esittämällä tavalla.

```
using UnityEngine;
using UnityEngine.UI;

public class Esimerkki : MonoBehaviour
{
    public Image kuvaObjekti;
    public Sprite uusiLähdekuva;
    public Color uusiVäri = new Color(255, 0, 0);

    void Update()
    {
        if (Input.GetKey(KeyCode.Q))
        {
            kuvaObjekti.sprite = uusiLähdekuva;
            kuvaObjekti.color = uusiVäri;
        }
    }
}
```

Koodiesimerkki 5. Esimerkkiluokka, joka vaihtaa peliobjektin kuvakomponentin lähdekuvaa ja kuvan väriä käyttäjän painaessa Q-näppäintä.

Koodiesimerkin 5 mukainen järjestelmä toimii kaikkiin päänäkymän säätietojen kuvakkeisiin. Lisäksi lämpötila ja tuulennopeus ovat helppoa vaihtaa int-muuttujatyypistä tekstiksi käyttämällä ToString()-funktiota.

Ohjauspaneeli

Ohjauspaneeli on yksi sovelluksen tärkeimmistä osista. Ajoneuvojen seuraamisesta ja ongelmien havaitsemisesta ei ole paljoa hyötyä, mikäli niille ei voi lähettää käskyjä ongelmien korjaamiseksi. Ohjauspaneelin näkee esimerkiksi kuvien 11 ja 12 oikeassa reunassa. Insinöörityöraporttia kirjoittaessa, on kuitenkin yhä hieman epävarmaa, mitä kaikkea ohjauspaneeliin tarkalleen tarvitaan. Muutama asia tiedetään varmasti tarpeelliseksi, kuten ovien avaamiseen ja kamerayhteyden avaamiseen tarvittavat napit. Ohjauspaneelissa tulee myös ottaa huomioon se, että eri ajoneuvoluokat tarvitsevat eri toimintoja. Esimerkiksi valvontadroonit tai ruohonleikkurit eivät tarvitse toimintoa ovien avaamiseen, mutta lähes kaikkiin ajoneuvoihin tarvitaan mahdollisuus kamerayhteyteen.

Ohjauspaneelin tulisi olla erityisen selkeä ja intuitiivinen, jotta käyttäjät eivät lähetä käskyjä vahingossa. Toisaalta ohjauspaneelista tulee voida lähettää käskyjä nopeasti ja helposti, mikäli sille tulee tarvetta. Näiden syiden takia ohjauspaneelin tyyppiset käyttöliittymäelementit kannattaa sijoittaa johonkin näytön reunaan ja pitää ne mahdollisimman staattisina, jotta käyttäjä tottuu helposti niiden sijaintiin ja muotoon. Yleensä ohjauspaneelin kaltaiset käyttöliittymäelementit löytyvät näytön alareunasta. Tässä projektissa muut käyttöliittymänelementit kuitenkin tarvitsevat paljon vertikaalista tilaa, jota on muutenkin vähemmän yleisimmissä kuvasuhteissa tietokoneella tai tableteilla. Tämän takia looginen valinta oli sijoittaa ohjauspaneeli päänäkymän oikeaan reunaan.

Ohjauspaneelin perusrakenne on muidenkin osien tapaan aika yksinkertainen tehdä Unityllä, eikä omaa koodia tarvinnut kirjoittaa paljoa. Pohjapaneelin ja nappien symmetrinen asettelu vie jo pitkälle, sillä Unityn inspector-näkymästä voi määritellä funktiokutsut nappien painalluksista tai osoittamisesta hiirellä. Luvussa 5.3 käsitelty valintajärjestelmä kertoo, minkälaisia ajoneuvoja on valittu eli mitä toimintoja ohjauspaneelissa tulee olla käytettävissä ja mitä ajoneuvoja napeilla lähetetyt käskyt koskevat. Lisäksi ovien avausnappi ja kamerayhteyden avausnappi ovat ”toggle”-tyyppisiä. Tämä tarkoittaa sitä, että ensimmäisellä napin painalluksella nappi aktivoituu ja se pysyy aktivoituna, kunnes sitä painaa uudelleen. Tämä toimintaperiaate esitetään koodiesimerkissä 6.

```
public void ToggleCameraView()
    {
        if (camOpen)
        {
            camOpen = false;
            EnableCameraView(false);
            camButtonText.text = "Avaa kamera-yhteys";
            camImage.color = openColor;
        }
        else
        {
            camOpen = true;
            EnableCameraView(true);
            camButtonText.text = "Sulje   kamera-yhteys";
            camImage.color = closeColor;
        }
    }
```

Koodiesimerkki 6. Koodipätkä, jota kutsutaan kamerayhteyden avaamisnapista.

”Toggle”-tyyppinen funktio on helppo ohjelmoida if-lauseella ja bool-muuttujalla, joka seuraa sitä, onko ”toggle” päällä vai pois päältä. Lisäksi napin teksti ja värikoodaus vaihdetaan tilanteen mukaan sopivaksi. Ovien avaus käyttää vastaavaa koodipätkää.

Asetukset ja muistiinpanot

Kuten luvun 5.4 alussa on mainittu, asetuksia ei tarvita tässä vaiheessa muuta kuin äänenvoimakkuuden säätöön. Asetukset-ikkunalla on kuitenkin toisena käyttötarkoituksena näyttää pikanäppäimet käyttäjälle, kuten kuva 12 osoittaa. Muistiinpanot-ikkunan tarkoitus on se, että käyttäjä voi kirjoittaa muistiinpanoja, joista saadaan lopuksi vuoron tiivistelmä seuraavalle käyttäjälle. Testivaiheessa käyttäjä näkee kuitenkin aina samat muistiinpanot alussa, mutta muistiinpanojen keräämiseen on muita käyttötarkoituksia.

Kuva 12. Päänäkymä, jossa on avattu asetukset-ikkuna. Muistiinpanot-ikkuna on samankaltainen, mutta sen sisällä on vain tekstikenttä, jonne voi kirjoittaa muistiinpanoja.

Asetuksista löytyvä äänenvoimakkuuden säädin toimii niin kuin odottaisi, eli säätimen ollessa ylhäällä äänenvoimakkuus on täysillä ja säätimen ollessa alhaalla kokonaan mykistetty. Lisäksi säätimen yläpuolella oleva kaiuttimen symboli toimii vaimennusnappina. Vaimennusnappi myös tallentaa, mikä äänenvoimakkuus oli ennen mykistystä, joten painamalla mykistystä uudelleen äänenvoimakkuus palaa edelliseen arvoonsa. Symbolin kuva vaihtuu sen mukaan, millä tasolla äänenvoimakkuus on. Säätimen alapuolella on tekstikenttä, joka kertoo tarkan äänenvoimakkuuden, ja siihen voi kirjoittaa tarkan haluamansa arvon.

Tätä toteutusta monimutkaisti merkittävästi se yhdistelmä, että äänenvoimakkuuden säätö halutaan tehdä logaritmisella kaavalla sekä säätimen että tekstikentän kautta. Ihmisen kuulo ei ole lineaarinen, joten lineaarinen äänenvoimakkuuden säätö ei tunnu oikealta. Lineaarinen äänenvoimakkuuden säädin menee liian hiljaiselle nopeasti, ja se tuntuu yleisestikin todella huonolta ja kömpelöltä. Perusidea logaritmisessa äänenvoimakkuuden säätimessä on se, että säädin palauttaa arvon väliltä

0,001 ja 1,000. Tämä arvo sijoitetaan Mathf.Log10-funktioon, ja sen palauttama arvo kerrotaan vielä 20:llä. (11, 12.) Tämä kaava näkyy esimerkkikoodissa 7.

```
public void AdjustVolume(float sliderValue)
    {
        currentMasterVolume = Mathf.Log10(sliderValue) * 20;
        masterMixer.SetFloat("MasterVolume", currentMasterVolume);
        volumeInputField.text = Mathf.RoundToInt(sliderValue *
100f).ToString();
        UpdateVolumeIcon();
    }
```

Koodiesimerkki 7. Logaritminen äänenvoimakkuuden säätö. ”sliderValue” on säätimen palauttama 0,001 ja 1,000 välinen arvo. Oikean äänenvoimakkuuden säätämisen jälkeen vielä päivitetään tekstikenttä ja kaiuttimen symboli.

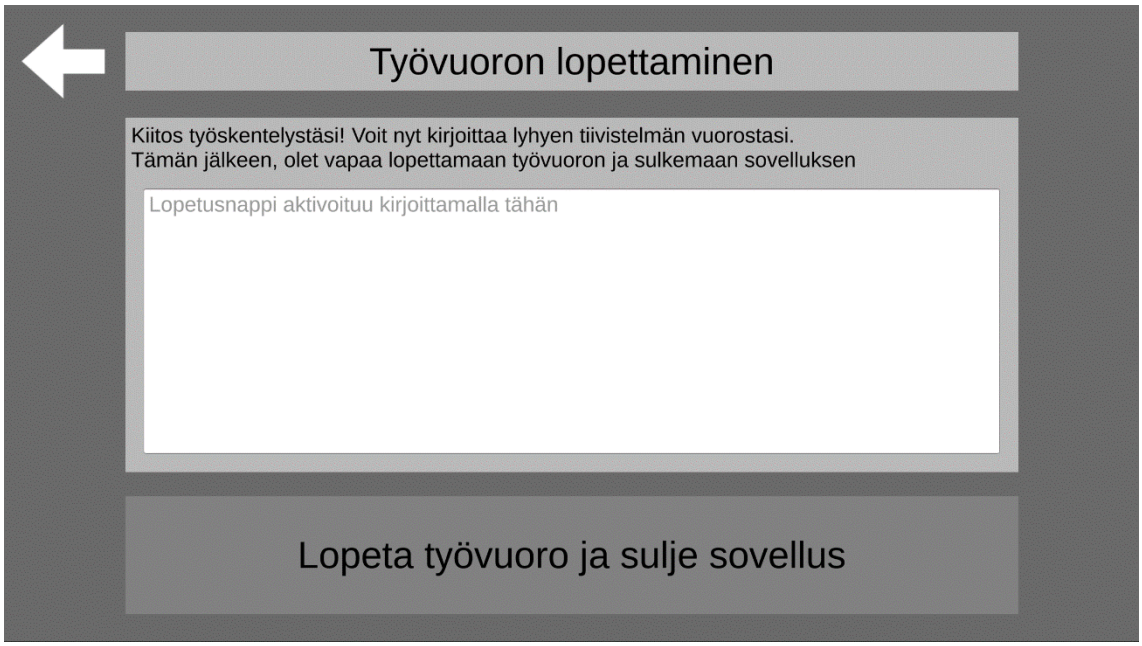
Logaritminen äänenvoimakkuuden säätö käsittelee vasta säätimellä säädetyn äänenvoimakkuuden. Toinen funktio käsittelee äänenvoimakkuuden, mikäli arvo syötetään tekstikentän kautta. Käyttäjä haluaa kirjoittaa äänenvoimakkuuteen arvon 0:n ja 100:n väliltä, joten se pitää ensin muuttaa ja rajoittaa 0,001 ja 1,000 välille. Tämän jälkeen arvo voidaan sijoittaa koodiesimerkin 7 osoittamaan logaritmiseen kaavaan. Lopuksi vielä äänenvoimakkuuden liukusäädin päivitetään vastaavaan paikkaan tekstikentän numeron mukaan.

Tekstikenttä, liukusäädin ja logaritminen kaava käsittelevät arvoja eri muuttujatyyppeinä, joten niitä pitää välillä kääntää koodissa. Mykistysnappina toimiva kaiutinsymboli on helppo päivittää sopivaksi äänenvoimakkuuden mukaan. Symboleita on neljä äänenvoimakkuuksille ja yksi mykistykselle, joten äänenvoimakkuus voidaan kategorioida aina 25 %:n välein.

Vuoron lopetuspaneeli

Kun työvuoro on lopussa, käyttäjä voi painaa päänäkymän yläosan keskeltä löytyvää ”Lopeta työvuoro” -nappia. Se avaa aloituspaneelien kaltaisen sivun. Sivulla on iso tekstikenttä, jonne on kopioitu muistiinpanot, joita käyttäjä saattoi kirjoittaa jo työvuoron aikana päänäkymän muistiinpanoikkunaan. Tässä vaiheessa on siis mahdollista viimeistellä muistiinpanoista työvuoron tiivistelmä, joka tulisi tavallisesti seuraavan käyttäjän aloituspaneeleissa vastaan. Käyttäjäkokeiden aikana aloituspaneelien tiivistelmä on aina sama. Kun

tiivistelmään on kirjoitettu jotain, käyttäjälle avautuu näkymän alareunaan nappi, joka lopettaa vuoron. Mikäli tiivistelmän tekstikenttä on tyhjä, tämä nappi on peitetty pois käytöstä tuttuun tapaan harmaalla kuvapeliobjektilla. Tilanne on esitetty kuvassa 13.



Kuva 13. Työvuoron lopetuspaneeli tyhjässä tiivistelmän tekstikentässä.

Lisäksi lopetuspaneelissa on aloituspaneelien kaltainen paluunappi. Tämä nappi vie takaisin päänäkymään. Tärkeä yleinen käytäntö on, että jokaisesta valikosta tai paneelista pääse vielä takaisin edelliseen näkymään. Muuten käyttäjät saattavat vahingossa painaa itsensä jonnekin ilman mahdollisuutta palata. Tässä tapauksessa paluunapin puute tarkoittaisi sitä, että vahinkopainalluksen takia käyttäjän tulisi sulkea sovellus ja aloittaa koko simulaatio alusta.

Äänten implementaatio

Äänten osuus ei ole kovin monimutkainen tai tärkeä tässä projektissa. Nappeihin saa lisää tuntumaa ääniefektien avulla. Kun hiiren vie napin päälle, siitä kuuluu pieni naksahdus napin tummentumisen lisäksi. Tämä tekee käyttäjälle selkeämmäksi sen, että kyseessä on nappi. Ääniefekti ja napin

tummennus myös tekevät selkeämmäksi sen, että hiiri on mennyt napista toiseen, mikä voi estää vahinkopainalluksia. Muut ääniefektit sovelluksessa ovat ajoneuvojen ilmoitukset ja varoitukset. Kullekin ilmoituksen tärkeysasteelle on oma lyhyt ilmoitusäänensä. Pienemmän tärkeysasteen ilmoitusääni on ikään kuin pehmeä kilahdus, johon on saattanut tottua esimerkiksi puhelimien ilmoitusäänistä. Tärkeimmän tärkeysasteen ilmoituksessa taas on huomattavasti terävämpi, huomiota sitova kolahdus, jonka tarkoitus on kiinnittää käyttäjän huomio tehokkaammin.

Sovellukselle riittää yksi äänimikseri ja yksi äänilähde. Äänilähteeseen tulee säätää ”spatial blend” -asetus arvoon 0, jotta ääni kuuluu tasaisesti kaikista kaiuttimista tai molemmilta puolilta kuulokkeita. Tämän lisäksi tarvitaan vielä skripti lähettämään äänikäskyjä äänilähteelle. Tässä skriptissä on muutama AudioClip-muuttuja, joihin määritellään sopivat ääniefektit kullekin tapahtumalle.

Ääniskripti tarvitsee muutaman funktion jokaiselle eri ääniefektille, joita kutsutaan esimerkiksi napeista tai tapahtumalokin ilmoituksista. Kaikki äänitapahtumat kutsutaan audioSource.PlayOneShot()-funktion kautta, koska se mahdollistaa useamman samanaikaisen päällekkäisen äänen. Lisäksi tällöin ei tarvitse vaihtaa äänilähteen omaa AudioClip-muuttujaa. Napin painalluksen yhteyteen voi määritellä funktiokutsun, joka soittaa äänen ääniskriptistä. Napin ääniefekti, kun hiiren vie sen päälle, tarvitsee hieman erilaisen käsittelyn. Napille pitää lisätä ”Event Trigger” -komponentti. Tämä komponentti pystyy lähettämään funktiokutsun ääniskriptiin, joka luo ääniefektin, kun hiiren vie napin päälle.

6 Käyttöliittymän parannukset

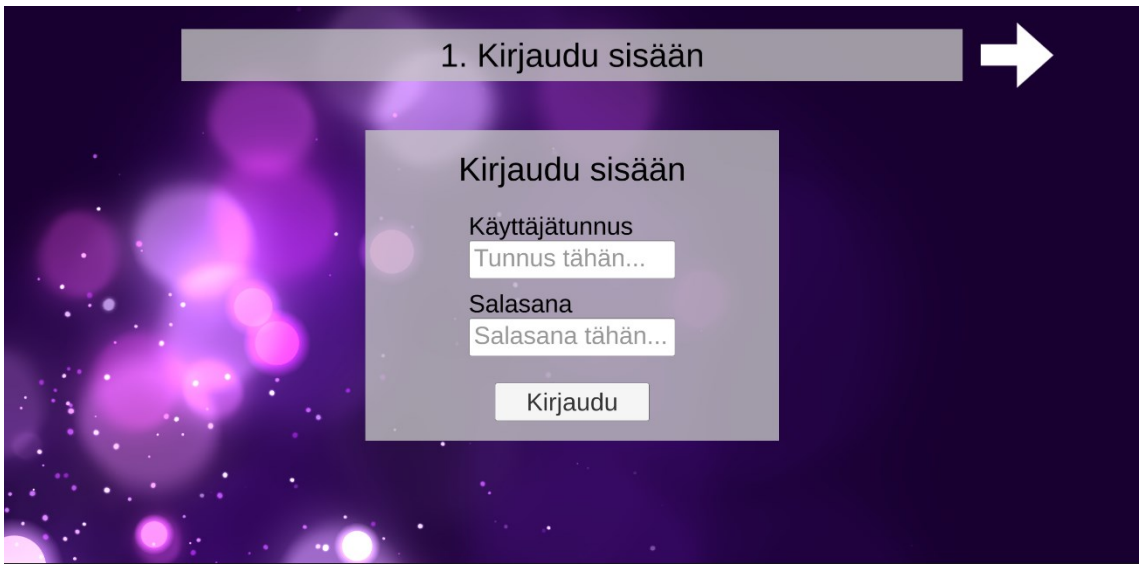
Mikäli haluaa hyvän, intuitiivisen ja loogisen käyttöliittymän, sitä pitää jatkuvasti testailla ja parannella. Pitää muistaa, että käyttökokemuksen- ja käyttöliittymän suunnitteluprosessi on jatkuva ja se elää projektin mukana. Usein parannukset saattavat olla todella pieniä, mutta isoissa määrissä pienetkin parannukset johtavat merkittävästi parempaan lopputulokseen. Tarpeen isoille ja merkittäville muutoksille voi usein karsia kokonaan tehokkaalla suunnittelulla ennen itse

toteutusta. Tämä piti paikkaansa tässä projektissa, sillä tarvetta merkittäviin muutoksiin ei esiintynyt missään vaiheessa pari viikkoa kestäneen suunnitteluvaiheen ansiosta. Myöhemmin projektissa tuli vielä paljon pienempiä muutoksia ja erilaisia vaihtoehtoja kokeiltiin, mutta ei mitään isoa.

Yleinen käytäntö on tehdä selkeitä iteraatioita sovelluksesta tai pelistä, ja seuraava iteraatio aina tuo joukon uusia muutoksia ja parannuksia. Tässä projektissa ei tähän nähty tarvetta, vaan se olisi mahdollisesti tuonut vain lisää säätöä ja vaivaa. Kun parannusideoita ja ehdotuksia esiintyi projektin edetessä, niitä pystyi kokeilemaan helposti välittömästi. Tämä mahdollisuus on yksi merkittävä hyöty siitä, että tekee kaiken kunnolla alusta asti, eli välttää kovakoodausta ja väliaikaisratkaisuja.

6.1 Aloituspaneelien parannukset

Aloituspaneelit kokivat jatkuvasti muutoksia projektin myötä. Perusidea kussakin paneelissa pysyi aina samana, mutta niiden ulkonäkö ja selkeys sai merkittäviä parannuksia projektin edetessä. Merkittävä osa näytöstä menee aloituspaneeleissa taustalle. Aluksi taustalla oli väliaikaisena ratkaisuna liukuväri. Liukuväri eri väriyhdistelmistäkokeiluista huolimatta tuntui hieman hassulta koko ryhmän mielestä. Sen takia suurimman osan projektin kehityksestä aloituspaneeleissa oli rauhallinen partikkeliefekti. Partikkeliefekti on Hovl Studion julkaisema. (13.) Alkuperäistä taustaefektiä muokattiin hieman, että se sopisi paremmin projektin käyttöön. Esimerkiksi efektin simulaationopeutta hidastettiin. Kuva 14 antaa viitettä siitä, miltä partikkeliefekti näytti.



Kuva 14. Ensimmäinen aloituspaneeli partikkeliefektitaustalla. Kuvasta ei voi saada täyttä käsitystä siitä, miltä elävä taustaefekti näyttää, sillä suoritusaikana partikkelit leijailevat vasemmasta alakulmasta hieman oikealle ylös.

Partikkeliefektin poistamiseen oli parikin syytä. Vaikka partikkeliefektin piti todella hitaana ja rauhallisena, pienikin liike taustalla häiritsee käyttäjää itse olennaisesta osasta paneelia. Lisäksi kun sovellus koottiin WebGL-yhteensopivaksi, efekti näyttää hieman pätkivältä ja yleisestikin huonolta. Tilalle laitettiin harmaa tausta. Tarkennettuna vielä harmaa tausta toteutetaan valitsemalla renderöivälle kameralle ”Clear Flags” -asetukseen ”Solid Color”, ja tähän pystyy valitsemaan sopivan harmaan värin. Lisäksi paneelien sisäiset ikkunat kannatti laittaa täysin läpinäkymättömiksi, sillä taustalla ei ollut enää efektiä, jonka tulisi näkyä osittain niiden läpi.

Jokaisella sivulla hienosäädettiin myös fonttikokoa. Aluksi ison tekstin ajateltiin olevan hyvä, koska jos tilaa on paljon, miksi ei käyttäisi sitä helposti luettavaan isoon tekstiin. Ryhmässä tultiin kuitenkin siihen tulokseen, että esimerkiksi kuvan 14 osoittamat tekstikoot ovat turhan isoja. Vaikka paneelit olivat jo ihan selkeitä ja hyvin luettavissa, pienempi fonttikoko sai niistä yhä rauhallisemmat ja helppolukuisemmat.

Lisäksi projektin käytettävyysasiantuntijana toiminut lehtori Liisa Seppänen suositteli käyttämään kerrointa 1,5–2,0 leipätekstin ja otsikoiden välillä. Kerroin

1,5 otsikoihin tuntui toimivan erinomaisesti tässä projektissa. Liisa Seppänen suositteli myös tiettyä pikselikokoa leipätekstin fonttiin, mutta tekstistä vastaava TextMeshPro-järjestelmä ei käsittele tekstin kokoa samalla tavalla kuin esimerkiksi Word tai PowerPoint. Insinöörityön tekijä joutui siis itse arvioimaan sopivampaa fonttikokoa. Kuva 15 esittää nykyisen kirjautumispaneelin parannukset.

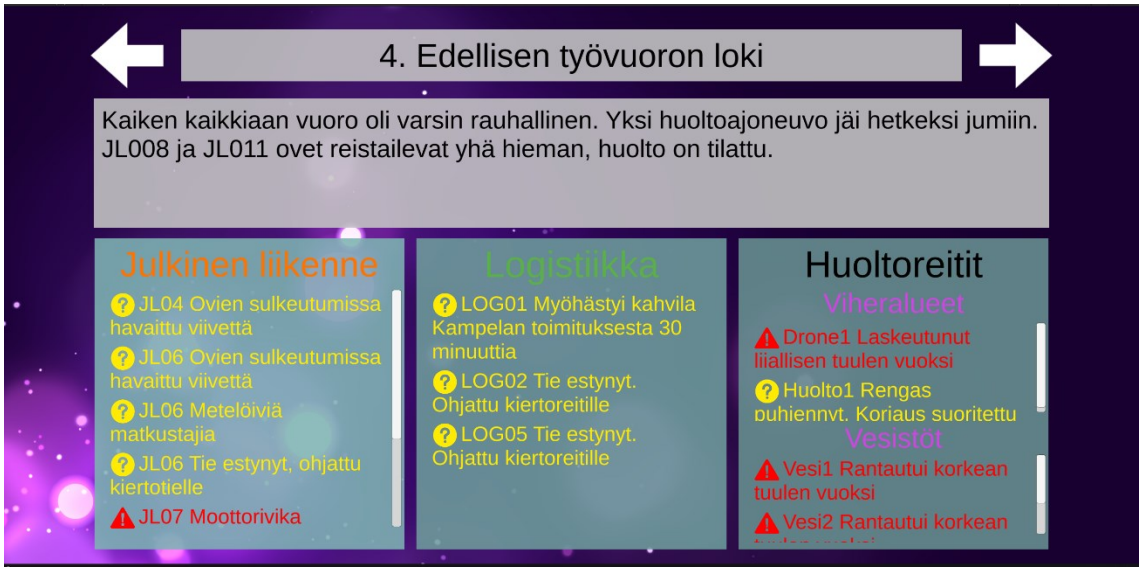
Kuva 15. Uusin versio ensimmäisestä aloituspaneelista.

Kuvassa 15 näkyy myös, että sisäänkirjautumisikkunan teksti vaihdettiin kuvaukseen siitä, mihin ollaan kirjautumassa sisään. Aloituspaneelin yläotsikko kuitenkin kuvaa jo sen, että paneelin tarkoitus on sisäänkirjautuminen.

Aloituspaneelit 2 ja 4 sisältävät ajoneuvolistat kullekin kategorialle. Yksi hyvältä kuulostanut parannusidea oli käyttää kategorioiden värikoodausta päänäkymässä ja aloituspaneeleissa. Kullekin kategorialle oli ideana laittaa tekstille, ajoneuvolistan napille ja kartalla pyörivälle ajoneuvolle oma värinsä. Jokaisella ajoneuvolistalla on myös omansävyisensä haalea tumman sininen väri taustalla. Itse päänäkymässä nämä värikoodaukset toimivat erinomaisesti. Aloituspaneeleista tuli kuitenkin vaikeammin luettavat ja hieman kaoottiset, kuten kuvasta 16 näkee.

Kuva 16. Toisen aloituspaneelin vanha versio. Värit tekevät tekstistä vaikeammin luettavaa ja hieman häiritsevää.

Kuvien 16 ja 17 osoittamat värikoodaukset olivat koko ryhmän mielestä yksimielisesti parannuksia vaille. Lopulta, eri kokeilujen jälkeen, tultiin kuitenkin siihen tulokseen, että värikoodaus ei ollut tarpeellista aloituspaneeleissa. Aloituspaneelien uusimmat parannellut versiot näkee luvusta 4, kuvat 3–7. Kuvassa 17 näkyy vielä neljäs aloituspaneeli ennen parannuksia.



Kuva 17. Neljännen aloituspaneelin vanha versio. Ajoneuvojen varoitus- ja huomautustekstit on myös värikoodattu.

Kuvan 17 osoittama neljäs aloituspaneeli sai myös hieman selkeyttä, kun ilmoitusten teksti muutettiin täysin mustaksi. Sen vierellä oleva värjätty symboli riittää helposti viestimään, minkä tärkeysasteen ilmoitus on kyseessä. Sama parannus tehtiin myös päänäkymän tapahtumalokiin.

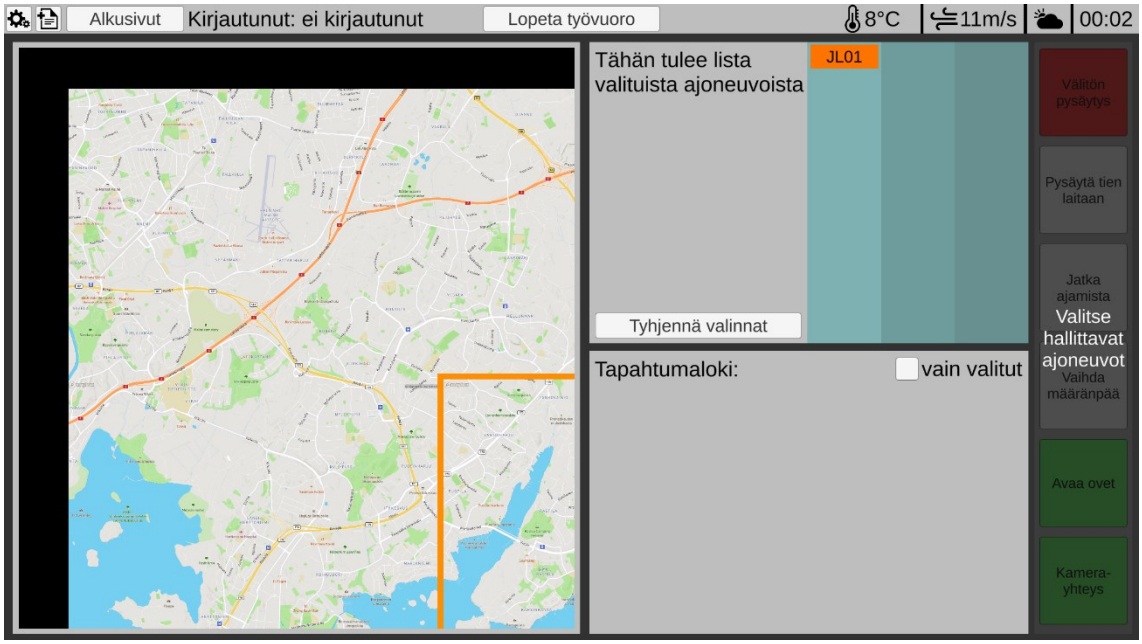
Aloituspaneeleihin tuli myös paljon muita pieniä muutoksia, kuten ikkunoiden parempaa asettelua ja se, että sisäänkirjautuminen vie välittömästi seuraavaan sivuun, kun käyttäjätunnuksen ja salasanan on laittanut oikein. Näitä muutoksia on kuitenkin niin paljon ja ne ovat niin pieniä, että niitä ei ole syytä listata. Lisäksi tulevaisuudessa tulee varmasti vielä lisää muutoksia projektin edetessä yhä pidemmälle.

6.2 Päänäkymän parannukset

Myös päänäkymään tehtiin useita muutoksia ja lisäyksiä projektin edetessä. Esimerkiksi alkuvaiheessa pitkään ylhäällä ei ollut niin sanottua yläpaneelia ollenkaan. Tämä johtui siitä, että sovellusta kehitettiin yksi osa kerrallaan, ja pitkään aikaan ei esimerkiksi ollut varmaa, tarvitaanko yläpaneelia ollenkaan. Kuitenkin esimerkiksi säätietojen ja asetusten takia yläpaneeli nähtiin tarpeelliseksi. Yläpaneeliin saa myös hyvin mahtumaan erilaisia valikoita ja alavalikoita, mikäli sille nähdään tulevaisuudessa tarvetta.

Kartan parannukset

Päänäkymän kartasta muodostui nopeasti toimiva versio, jota voi liikuttaa ja zoomata. Kartassa oli kuitenkin hieman ikävä ominaisuus. Kun karttaa zoomasi taakse nähdäkseen koko toiminta-alueen kerrallaan tai liikutti karttaa reunoille pois toiminta-alueelta, reunoilla näkyi runsaasti mustaa reunaa. Musta reuna oli siis aluetta, jonne kartta ei yltänyt. Tämä korjattiin lisäämällä kahdeksan karttaa, jotka ovat kukin yhtä isoja kuin toiminta-alueen kartta. Näin käyttäjä voi zoomata kauas tai vierittää näkymää reunalle, ilman että musta alue tulee näkyviin. Mustan alueen saa yhä näkyviin, jos sitä lähtee tietoisesti yrittämään, mutta tavallisessa käytössä sitä ei pitäisi tulla näkyviin. Tämä on näytetty kuvassa 18.



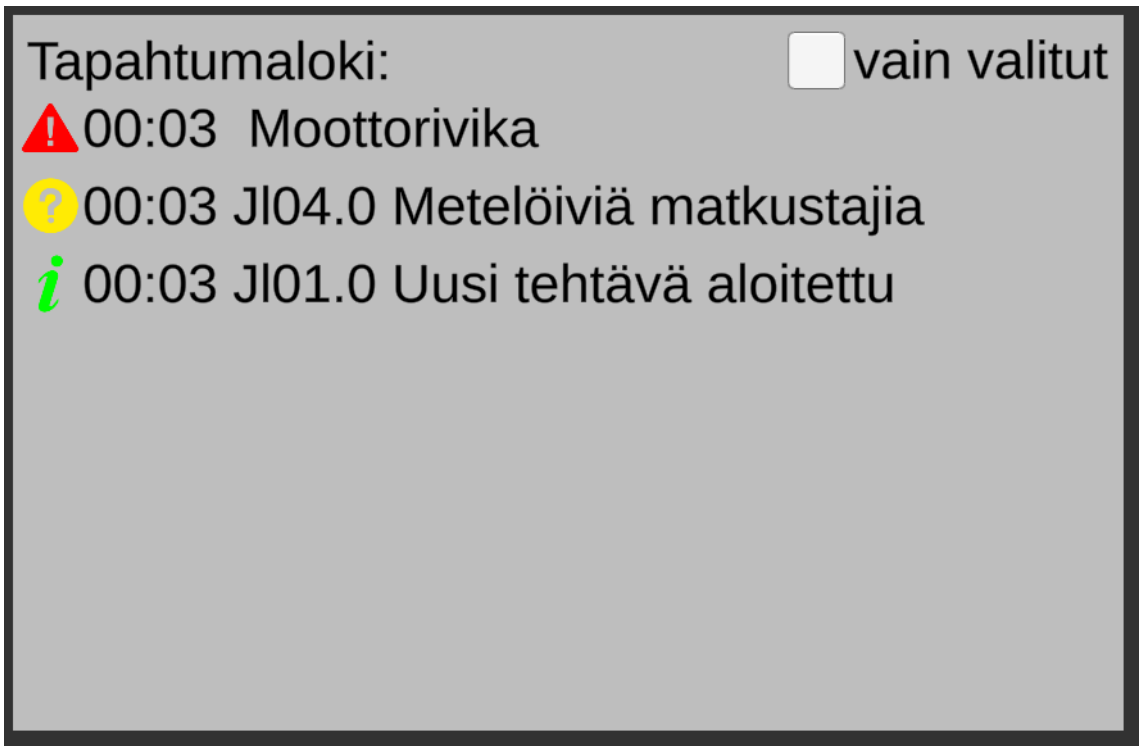
Kuva 18. Päänäkymä, jonka kartassa tulee näkyviin niin sanottu musta alue ja rajattu toiminta-alue.

Kuvan 18 näkymän saa replikoitua, mikäli zoomaa niin kauas kuin mahdollista ja vierittää kauas reunalle. Käyttäjän tulee myös pitää hiirtä pohjassa tai näkymä ponnahtaa keskitetyksi takaisin oransseilla reunoilla merkittyyn toiminta-alueeseen. Oranssit reunat lisättiin merkitsemään toiminta-alue tämän parannuksen yhteydessä. Kuvan 18 esittämää mustaa aluetta ei nähdä oleelliseksi ongelmaksi, mutta kartasta voi tehdä tulevaisuudessa vielä isomman, mikäli sille nähdään tarvetta.

Kartassa on tiedostettu olevan yksi hieman käytettävyyttä vaikeuttava ominaisuus. Kun karttaa zoomaa, zoomaus tapahtuu kohti kartan keskiosaa eikä siihen, mihin hiiri osoittaa. Tämä tarkoittaa sitä, että kun käyttäjä haluaa vaikka zoomata tietyn ajoneuvon kohdalle, hänen kameransa liikkuu kohti keskustaa zoomauksen yhteydessä. Tämä johtaa siihen, että käyttäjän tulee zoomauksen lisäksi vierittää karttaa zoomauksen kohdetta päin. Tämä pieni ongelma on ollut alusta asti, mutta sitä ei nähty niin oleelliseksi, että olisi kannattanut käyttää aikaa ja vaivaa sen korjaamiseen. Jotkin ongelmat voivat olla aika mitättömiä, mutta niiden korjaaminen vaatisi todella paljon vaivaa. Tulevaisuudessa asialle saatetaan kuitenkin tehdä jotakin.

Tapahtumalokin ja valintajärjestelmän parannukset

Tapahtumaloki sai muutaman parannuksen projektin edetessä. Nopeasti tapahtumalokin toteutuksen jälkeen huomattiin, että uusimman viestin tulisi näkyä ylimpänä. Tämän parannuksen sai tehtyä vaihtamalla yhden muuttujan inspector-näkymässä, jotta lista tulisi käänteisessä järjestyksessä ja täten näyttäisi uusimman viestin ylimpänä. Parannus oli siis helppo toteuttaa, mutta se paransi käyttökokemusta merkittävästi. Toinen merkittävä parannus tapahtumalokiin oli lisätä kullekin tärkeysasteelle oma symboli ja värikoodaus. Kuvassa 19 näkee, miltä tapahtumaloki näyttää muutamalla ilmoituksella.



Kuva 19. Tapahtumaloki, jossa on yksi ilmoitus jokaisesta tärkeysasteesta. Kuvan ilmoitukset ovat paikanpitäjiä eivätkä oikeita ilmoituksia. Tässä esimerkissä punaista ilmoitusta ei ole yhdistetty mihinkään ajoneuvoon kehityssyistä.

Ääniefekti, ilmoituksen symboli ja väri välittävät viestin tehokkaasti käyttäjälle. Kun kaikki ilmoitukset olivat vain mustaa tekstiä ilman symbolia tai ääntä, käyttäjä ei välttämättä ymmärtänyt asian tärkeyttä tarpeeksi nopeasti. Esimerkiksi vihreä ilmoitus on tyypillisesti hyvä tietää, mutta punainen ilmoitus

tarvitsee lähes poikkeuksetta välittömiä toimia. Pitkään myös teksti oli tärkeysasteen mukaan värikoodattu, mutta teksti päädyttiin pitämään mustana, jotta se olisi hieman helpompi lukea.

Valintajärjestelmä oli sen tekemisestä asti aina käyttökelpoinen, mutta se sai vielä muutaman päivityksen helpottamaan käyttämistä. Esimerkiksi valittujen autojen keltainen korostus paransi paljon käytettävyyttä, koska valittujen autojen havainnollistaminen kartalta parani todella paljon. Lisäksi päätettiin lisätä ajoneuvolistojen viereen näppäinpainalluksen lisäksi nappi, joka tyhjentää kaikki valinnat. Myöhemmin päätettiin myös se, että jos on valinnut vain yhden ajoneuvon ja sitä painaa uudelleen, valintajärjestelmä poistaa sen valinnan ja näin tyhjentää listan kokonaan. Tämä yksityiskohta ei ollut merkittävä parannus, mutta se oli todella helppoa toteuttaa, ja täten muutos kannatti tehdä.

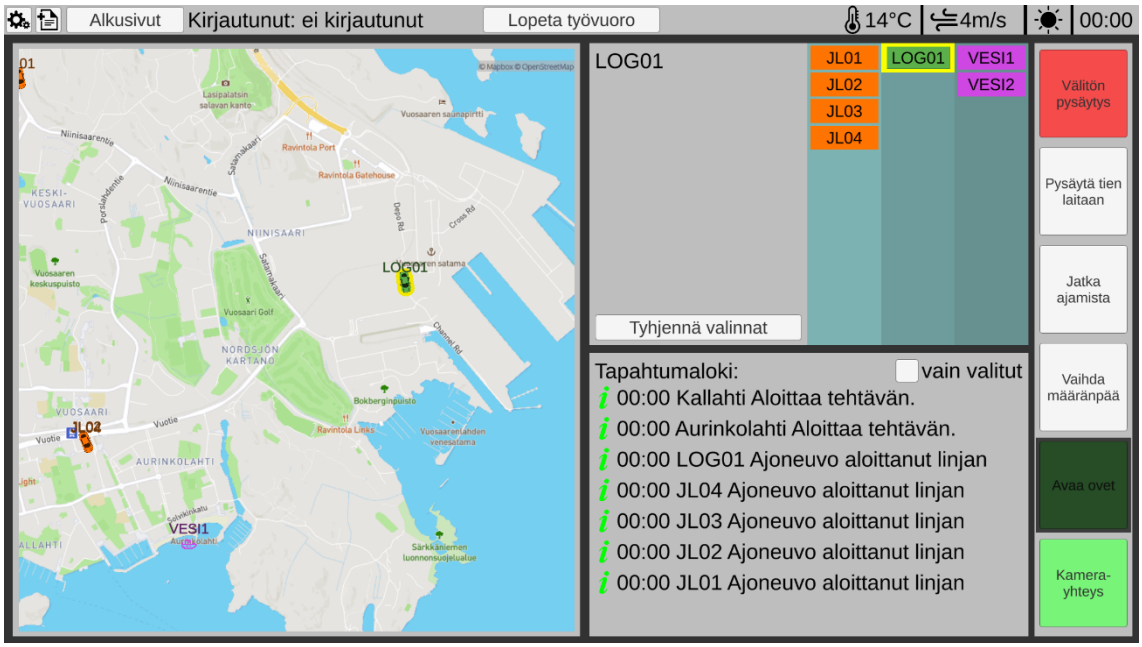
6.3 Muita parannuksia

On vielä muutama parannus, jotka ovat käsittelyn arvoisia. Aiemmin mainittiin, miten koko sovelluksen kaikkia tekstejä on hieman muuteltu eri tavoin. Yleinen hyväksi todettu piirre teksteihin oli se, että ne muutettiin mustaksi. Värikoodaus on hyvä asia, mutta teksteissä se tuntui yleisesti liian häiritsevältä, vaikka kokeiltiin useita erilaisia värisävyjä. Merkittävä parannus tuli siitä, kun ajoneuvolistoissa olevien nappien ajoneuvon nimiteksti muutettiin kiinteäksi mustaksi. Aiemmin nämä tekstit olivat valkoisia mustalla reunalla, mutta luettavuus oli silti hankalampaa kuin täysin mustalla tekstillä. Kuvassa 11 on vielä valkoinen teksti ja kuvissa 12, 18 ja 20 näkyy kyseinen parannus.

Lisäksi käytettävyysasiantuntijana toimineelta Liisa Seppäseltä tuli värikoodaukseen liittyvää palautetta. Hänen mukaansa värikoodaus on hyvä asia, mutta asian ei koskaan tulisi yksin jäädä sen varaan. Tämän takia päätettiin lisätä symbolit tapahtumalokiin värin lisäksi. Myös kartalla liikkuville ajoneuvoille on tarkoitus lisätä kullekin ajoneuvolle vastaava symboli. Autoilla, valvontadrooneilla ja ruohonleikkureilla on omansa, mutta bussit ja logistiikka-ajoneuvot esimerkiksi käyttävät tällä hetkellä samaa symbolia.

Kun tultiin siihen päätökseen, että käyttäjän tulee vielä kirjoittaa vuoron tiivistelmä lopussa, se toiminto lisättiin. Yksi ongelma sovelluksessa on kuitenkin se, että käyttäjällä tulee helposti hieman pitkästyttävää valvonnan aikana. Lopuksi työvuoron lopetuksen yhteydessä käyttäjä saattaa kirjoittaa tiivistelmän nopeasti ilman kunnon ajatusta. Tämän takia nähtiin yksimielisesti hyväksi päätökseksi lisätä päänäkymään ikkunan, johon voi jo valmiiksi kirjoittaa muistiinpanoja tai tiivistelmää. Nämä muistiinpanot tai tiivistelmä kopioituvat suoraan vuoron lopetuksesta vastaavaan näkymään. Jatkoideana tuli myös, että tapahtumalokin tapahtumia voisi esimerkiksi hiiren oikealla painalluksella lisätä näihin muistiinpanoihin suoraan.

Ohjauspaneelin tarkka sisältö on hieman keskeneräinen vielä tässä vaiheessa. Projektin edetessä tuli esille kuitenkin yksi välttämätön parannus ohjauspaneeliin. Ohjauspaneelin pitää mukautua sen mukaan, mitä ajoneuvoja on valittu. Esimerkiksi autonominen ruohonleikkuri ei tarvitse nappia ovien avaamiseen, koska näissä ruohonleikkureissa ei edes ole ovia. Kuva 20 esittää vastaavan tilanteen.



Kuva 20. Päänäkymä, jossa ohjauspaneelin nappi on piilotettu, koska logistiikkakategorian ajoneuvo on valittu.

Parannus on todella tärkeä käytettävyyden kannalta. Napit, jotka eivät tee mitään, olisivat vain hämmentämässä käyttäjää. Lisäksi käyttäjä saattaisi luulla toiminnon olevan mahdollinen valitulle ajoneuvolle, mikä voisi johtaa erilaisiin väärinymmärryksiin. Tämä parannuksen toteutus oli kuitenkin helppo. Ensin jokaiselle napille piti lisätä oma piilottava objekti napin päälle. Tämän jälkeen voi kysyä valintajärjestelmältä, minkä kategorian ajoneuvo on valittu, ja sen mukaan aktivoida tai deaktivoida näitä piilottavia objekteja. Tällä hetkellä on päätetty, että mikäli useampi ajoneuvo on valittu, käskyjä ei voi lähettää. Pääsyynä on se, että käyttäjä saattaisi vahingossa lähettää käskyn useammalle ajoneuvolle.

7 Yhteenveto

Kokonaisuudessaan autonomisten ajoneuvojen valvonnan simulaatiosovellus onnistui kohtuullisesti. Sovelluksen päätoiminnot ja perusideat toimivat kuten haluttua. Kuitenkin immersiivinen simulaattori, joka voisi todentuntuisesti testata käyttäjien stressinsietokykyä, tarvitsee vielä hieman työtä. Tämä ei välttämättä ole niinkään käyttöliittymään pohjautuva ongelma, vaan simulaatio tarvitsisi hieman lisää erilaisia ongelmatilanteita ja monimutkaisempia ratkaisuja ongelmiin. Lisäksi projektia jatketaan yhä ainakin syksyn 2022 ajan itse käyttäjäkokeisiin asti.

Unity-pelimoottori oli varsin toimiva ratkaisu projektin toteutukseen. SUMO-liikennesimulaattori ja siihen liittyvä TraCI-rajapinta on tarkoitettu hieman erilaiseen käyttöön, mutta niillä sai toteutettua simulaatiota ja ongelmatilanteita pienen lisätyön avulla. Niissä on taustalla jotain hieman kyseenalaisia kiertotapoja toteuttaa ongelmatilanteita, esimerkiksi este kaistalla -ongelmatilanne saa kohdeajoneuvon pysähtymään asettamalla sen eteen SUMO-simulaatiossa toisen auton, jota ei kuvata Unityn puolelle.

Itse käyttöliittymän kanssa työskentely oli sujuvaa suurimmalta osalta. Käyttöliittymän suunnittelu ei alkuun vaikuttanut työläältä. Vaikutti siltä, että on helppoa ja nopeaa vain sijoittaa tarvittavia osia näytölle ja saada toimiva ratkaisu ilman merkittävää työtä. Pienen tutkimuksen jälkeen tultiin kuitenkin

siihen tulokseen, että kannattaa käyttää tarpeeksi aikaa suunnitteluun ennen toteuttamisen aloittamista. Tällä voi pitkällä aikavälillä säästää aikaa ja saada huomattavasti paremman lopputuloksen. Opiskeluajan kursseilta ja kokemuksesta tiedettiin, että käyttökokemus- ja käyttöliittymäsuunnitteluun on erilaisia lähestymistapoja. Suurin osa näistä lähestymistavoista kuulosti kuitenkin melko samanlaisilta. Sen lisäksi, että lähestymistavat kuulostivat samalta, ne eivät anna selkeitä ohjeita, vaan kaikki ovat käytännössä arkijärjen käyttämistä suunnitteluvaiheeseen. Voidaan vahvasti suositella kaikille käyttöliittymästä vastaaville oman suunnitteluvaiheen pitämistä ennen toteuttamista. Monet käyttökokemus- ja käyttöliittymäsuunnitteluperiaatteet ovat myös hyödyllisiä, mutta erilaisten lähestymistapojen tutkiminen ja vertailu tuntui lopulta lähinnä ajan hukkaamiselta.

Autonomisten ajoneuvojen valvonnan simulaatiosovelluksen käyttöliittymään jäi paljon parannettavaa ja jatkokehitysideoita, mutta se on erittäin hyvin toimiva ja riittävä nykyiseen vaiheeseen. Alusta asti oli kuitenkin ollut tiedossa, että projekti mahdollisesti jatkuu pitkään ja myös käyttöliittymää pitää päivittää muiden parannuksien yhteydessä. Autonomisten ajoneuvojen simulaatiosovelluksen ja täten myös käyttöliittymän viimeinen versio jää siis toistaiseksi odottamaan tulevaisuuteen.

Lähteet

1 Nurmi, Minea. 2022. Some täyttyi sekoilevista kuljetusroboteista – näin ne oikeasti toimivat. Verkkoaineisto. Alma Media. <https://www.iltalehti.fi/iltvuutiset/a/39bff293-914f-40d1-a650-a093e3c95156>. Luettu 10.7.2022.

2 Shuttleworth, Jennifer. 2019. Levels of Driving Automation. Verkkoaineisto. SAE International. <https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic>. Luettu 12.7.2022.

3 Meeting the Big Challenges in Autonomous Driving With Remote Control. 2022. Verkkoaineisto. Sensible 4. <https://sensible4.fi/technology/articles/meeting-the-big-challenges-in-autonomous-driving-with-remote-control/>. Luettu 12.7.2022.

4 Clarifying human-centered design thinking once and for all. 2022. Verkkoaineisto. Hotjar. <https://www.hotjar.com/design-thinking/vs-human-centered-design/>. Luettu 5.7.2022.

5 Razzouk, Rim. 2012. What Is Design Thinking and Why Is It Important? Verkkoaineisto. AERA. <https://www.researchgate.net/publication/258183173_What_Is_Design_Thinking_and_Why_Is_It_Important>. Luettu 7.7.2022.

6 Kopf, Ben. The power of Figma as a Design Tool. Verkkoaineisto. Toptal. <https://www.toptal.com/designers/ui/figma-design-tool>. Luettu 12.10.2022.

7 Build once, reach billions. 2022. Verkkoaineisto. Unity Technologies. <https://unity.com/solutions/multiplatform>. Luettu 4.10.2022.

8 OpenStreetMap. 2022. Verkkoaineisto. OpenStreetMap. <https://www.openstreetmap.org/about>. Luettu 5.9.2022.

9 Why You Should Avoid Runtime Instantiation and Destruction in Unity. 2021. Verkkoaineisto. Noveltech. <https://www.noveltech.dev/why-avoid-runtime-instantiation-unity/>. Luettu 12.10.2022.

10 Raycast target on UI elements. 2015. Verkkoaineisto. Unity Technologies. <https://answers.unity.com/questions/1099030/raycast-target-on-ui-elements.html>. Luettu 14.9.2022.

11 Pigeon, Stéphane. The non-linearities of the Human Ear. Verkkoaineisto. AudioCheck.net. <https://www.audiocheck.net/soundtests_nonlinear.php>. Luettu 27.9.2022.

12 Leonard, John. 2018. Unity Audio: How to make a UI volume slider (the right way). Verkkoaineisto. Youtube. <https://www.youtube.com/watch?v=xNHSGMKtlv4>. Luettu 27.9.2022.

13 Horobets, Vladyslav. 2021. Background Effects. Verkkoaineisto. Hovl Studio. <https://assetstore.unity.com/packages/vfx/particles/background-effects-198996>. Luettu 28.9.2022.