Ari-Pekka Åttman

Satama-alueen työn suunnittelutyökalu

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Kesäkuu 2019

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Kesäkuu 2019	**Tekijä/tekijät** Ari-Pekka Åttman
Koulutusohjelma Tietotekniikka		
Työn nimi Satama-alueen työn suunnittelutyökalu		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 21
Työelämäohjaaja Jari Isohanni		

Tämän opinnäytetyön tavoitteena oli tehdä prototyyppi satama-alueen työn suunnittelutyökalusta sekä kertoa työnvaiheista. Suunnittelutyökalulla voidaan suunnitella, mihin satama-alueelle tulevat rahtilaivat menevät lastausta tai purkua varten. Työkalu on tarkoitettu satamille, joissa on rahtilaivaliikennettä ja kiinnostusta digitalisoida sataman toimintaa. Työn tavoite on selkeyttää ja nopeuttaa suunnittelua sekä mahdollistaa tiedon jakamisen reaaliajassa useammalle näytölle. Työssä käytetään työkaluna Unity-pelimoottoria, Visual Studio Codea ja ohjelmointikielenä C#-kieltä.

Asiasanat
Prototyyppi, Unity, GUI, UI

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Centria University of Applied Sciences	**Date** June 2019	**Author** Ari-Pekka Åttman
Degree programme Information Technology		
Name of thesis The planning tool for the work in the harbour area		
Instructor Sakari Männistö		**Pages** 21
Supervisor Jari Isohanni		

The aim of this thesis was to create prototype of the planning tool for the work in the harbour area and to explain the stages of the work. With the planning tool the user will be able to plan where the next freighter should dock for the loading or unloading. The planning tool is meant for the industrial harbors that have freighters docking at their dock and are interested in digitalizing their business. The prototype's main objective was to make planning more agile and clearer, and to allow sharing of the planning information with multiple different places in realtime. Tools that were used to make this prototype included Unity game engine and Visual Studio Code as code editor. C# was used as the programming language.

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Key words
Prototype, Unity, GUI, UI

KÄSITTEIDEN MÄÄRITTELY

GUI, Graphic User Interface

JSON, JavaScript Object Notation

UI, User Interface

1 JOHDANTO

Projektin toimeksianto tuli Centria-ammattikorkeakoulun TKI:n kautta, ja se on osa Biline-hanketta. Projektin tarkoituksena oli luoda teollisuussatama-alueille suunnittelutyökalun prototyyppi, jolla teollisuussatamat voivat suunnitella saapuvien laivojen paikat satamalaiturilla sekä luoda ympäristön, jossa on mahdollista suunnitella laivojen lastaukseen käytettävien koneiden ja laivaan tai laivasta lastattavan materiaalin sijoittamista ja määrää. Kaikki tämä tehdään, jotta mahdollistetaan parempi työnsuunnittelu, turvallisuus ja tehokkuus. Vastaavanlaisia toteutuksia on olemassa konttisatamiin mutta ei bulkkimateriaalin käsittelyyn. Sovellukseen halutaan graafinen käyttöliittymä, jota on helppo lukea ja käyttää. Sovelluksessa näkyvää suunnittelutietoa tulee jakaa useampaan paikkaan samaan aikaa.

Prototyypissä käytettävän koodin tulee olla suurimmaksi osaksi omaa, ja sovelluksen suunnittelu tehdään yhteistyössä TKI:n kanssa. Tästä syystä sovelluksen luonti toteutetaan Centria-ammattikorkeakoulun tiloissa, jolloin kommunikointi TKI:n ja minun välillä helpottuu. Aluksi tulee valita sovelluksen luontiin sopivat työkalut haluttujen ominaisuuksien perusteella. Sovelluksen tulee antaa käyttäjälle kuva sovelluksen käyttötavoista, käyttömahdollisuuksista sekä sen tuomista hyödyistä.

Sovelluksen tulee siis olla helppokäyttöinen, ettei sen opetteluun tarvitse kuluttaa suuria määriä työtunteja, ja lisäksi sen tulee näyttää käyttäjille, kuinka helppoa ja nopeaa sillä suunnitellun tiedon käyttö voi olla.

2 SOVELLUKSESSA KÄYTETYT OHJELMAT JA TYÖKALUT

Työn satama-altaan suurpiirteinen muoto otettiin Kokkolassa sijaitsevan tehdasalueen satama-altaasta. Suunnittelutyökalu todettiin hyväksi ideaksi, koska satama-alueella on hyödyllistä olla työkalu, jolla näyttää toteutukseen halutut suunnitelmat reaaliajassa. Tämä ohjelman tarkoitus on nopeuttaa suunnitelmien luomista ja mahdollistaa niiden näyttäminen useammassa paikassa yhtä aikaa.

Prototyyppiä varten oli kaksi eri työkaluvaihtoehtoa, Unity ja Unreal Engine. Molemmissa työkaluissa on mahdollista toteuttaa kolmiulotteinen, reaaliajassa oleva suunnittelutyökalu. Näistä kahdesta päädyttiin Unityyn sen helppokäyttöisyyden ja laajan koodidokumentaation vuoksi.

Unityssä ohjelmointikielivaihtoehtoina on C#, Java sekä Boo. Päädyimme C#:n käyttöön, koska siitä on eniten aiempaa kokemusta, sekä selkeät dokumentaatiot ja paljon tietoa foorumeilla.

2.1 Unity

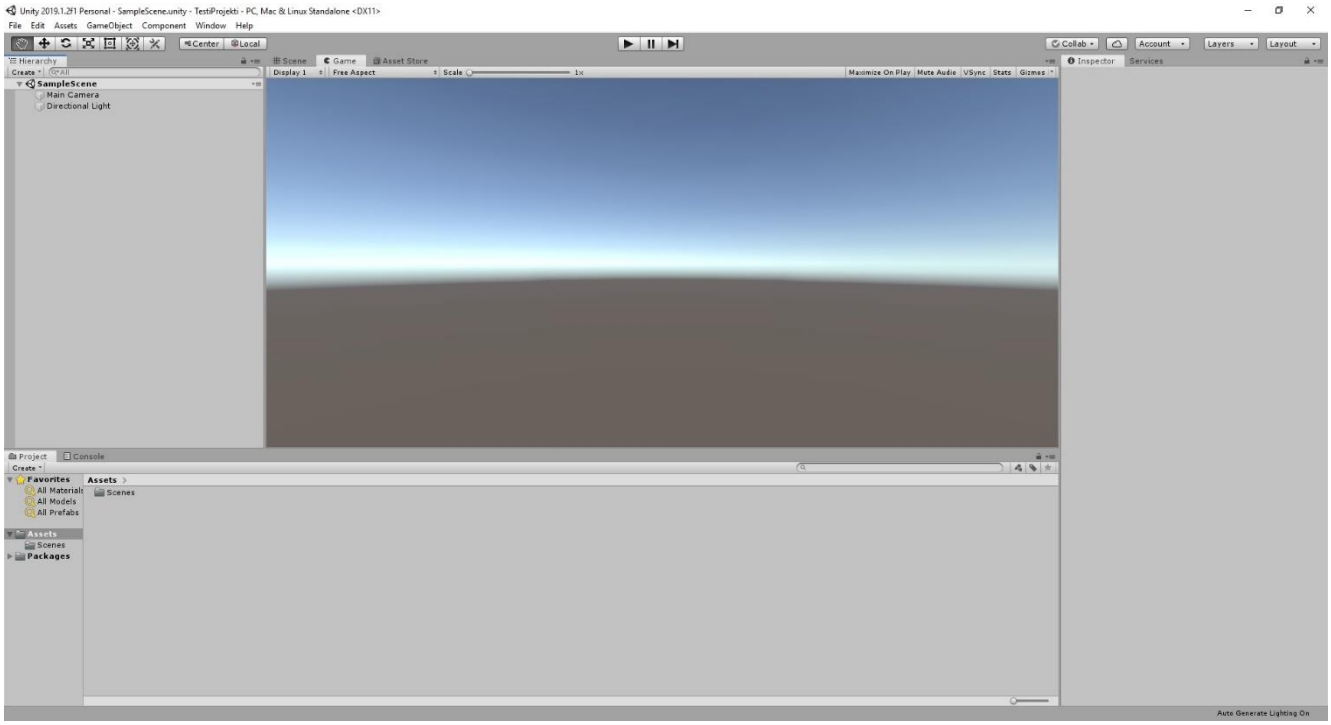
Unity on monialustainen pelimoottori, josta on ilmainen ja kaupallinen lisenssi. Ilmaisella lisenssillä saa tehdä kaksi- ja kolmeulotteisia pelejä tai sovelluksia, jos niitä tehdään ei-kaupalliseen käyttöön tai jos yritys tai organisaatio, joka käyttää kyseistä lisenssiä, on tuottanut alle 100 000 USD edellisellä tilikaudella. Unityn avulla pelit ja sovellukset voidaan tehdä useammalle eri alustalle, kuten esimerkiksi Windowsille, Android-puhelimille sekä webselaimille (Unity.)

Unityssa on valmiina erillinen fysiikkamoottori nimeltään PhysX, jonka on tehnyt Nvidia. Fysiikkamoottorin avulla on mahdollista käyttää reaaliaikaista fysiikkojen simulointia, kuten esimerkiksi painovoiman simuloinnissa. Jos sovellus on käynnissä ja sisältää jonkinlaisen fysiikkakomponentin niin sille tehdään koko ajan fysiikan simulointia (Unity.)

Unity mahdollistaa nopean ja selkeän käyttöliittymän suunnittelun sekä ympäristön rakentamisen valmiiden komponenttien avulla. Ohjelmaan on myös helppo tuoda itse tehdyt 2D- ja 3D-mallit ja käyttää niitä valmiiden komponenttien sijaan. Unitystä on myös selkeät dokumentaatiot sekä harjoituksia, joita pidetään ajan tasalla uusimman Unity version kanssa. Dokumentaation avulla uusikin käyttäjä voi nopeasti tehdä omia pelejä tai sovelluksia. Unity tarjoaa käyttäjilleen myös foorumin, jonka kautta pyytää

apua muilta käyttäjiltä, jos dokumentaatiosta ei löydy vastausta haluttuihin kysymyksiin (Unity Documentation.)

Unityllä voidaan testata tehtyä sovellusta suoraan Unityssä, ja tämän ominaisuuden avulla on nopea testata uusia muutoksia sekä korjata mahdollisia virheitä. Sovellusta ajettaessa sille voi tehdä erilaisia testejä, joiden avulla voi seurata esimerkiksi käyttäjän tekemiä toimintoja, kuten tietyn painikkeen painallusta. Nämä ominaisuudet tekevät Unitystä hyvän työkalun nopeaan erilaisten prototyyppien luomiseen. Kuvassa 1 näkyy esimerkki miltä uusi Unity-projekti näyttää ennen kuin sinne lisätään objekteja tai skriptejä.



KUVA 1. Uuden Unity-projektin näkymä

2.2 Microsoft Visual Studio

Microsoft Visual Studio on Microsoftin tekemä ohjelmankehitysympäristö. Tämä ohjelmankehitysympäristö on luotu helpottamaan ohjelmoijien työtä monilla eri ohjelmointikielellä. Visual Studiossa on useita ohjelmointia helpottavia ominaisuuksia, kuten IntelliSense, sisäänrakennetut ohjelmointikielen kääntäjät sekä useita erilaisia liitännäisiä. IntelliSense auttaa käyttäjää täydentämällä koodia käytetyn ohjelmointikielen ja ympäristön perusteella. Visual Studio tarjoaa myös eri versionhallintaohjelmien

käytön, kuten Git ja Subversion. Tämä mahdollistaa koodin helpon jakamisen ja hallinnoinnin useamman käyttäjän välillä. Visual Studio tarjoaa myös erilaisia tiimipalveluita, jotka auttavat projektien aikataulutuksessa, tehtävien jaossa sekä kehityksen seurannassa (Visual Studio.)

Visual Studio tarjoaa myös tekstieditorin mikä on tehty avoimella lähdekoodilla, mikä mahdollistaa ohjelman räätälöinnin käyttäjän tarpeiden mukaan. Tämän tekstieditorin nimi on Visual Studio Code, ja se sisältää hieman samoja ominaisuuksia kuin muut Visual Studion ohjelmat, kuten automaattisen koodin täydentämisen ja tuen virheenkorjaukselle. Code ei tue vakiona kaikkia ohjelmointikieliä, mutta Visual Studio Marketplacesta löytyy tuhansia eri laajennuksia, joiden avulla saadaan käyttöön automaattisen koodin täydentämisen tai muita Visual Studion tarjoamia ominaisuuksia (Visual Studio.)

Unity versiosta 2018.1 eteenpäin Unity asentaa vakiokoodieditorina Visual Studio 2017 Community MonoDevelop-Unity sijasta. Aiemmissa versioissa jouduttiin erikseen asetuksista asettamaan koodieditoriksi Visual Studion, jos ei haluttu käyttää MonoDevelop. Visual Studio Coden käyttöönotto vaatii edelleen sen asettamisen vakiokoodieditoriksi Unityn asetuksista. Codeen tulee myös asentaa vähintään kaksi liitännäistä, C# Extension sekä Unity Debugging Extension. (Lukasz Paczkowski 2018.)

Unity Debugging Extension tarjoaa Unity-koodin virheiden jäljittäjän. Sen avulla voi seurata koodin toimivuutta asettamalla pysäytyspisteitä tiettyyn kohtaan koodia, jolloin sovellus pysähtyy siihen kohtaan ja jolloin voi rauhassa katsoa mitä sillä hetkellä tapahtuu. C# Extension taas tarjoaa C#-ohjelmointia helpottavia ominaisuuksia, kuten aiemmin mainitun IntelliSensen ja koodin korostuksen. On mahdollista ohjelmoida myös ilman näitä liitännäisiä, mutta silloin ei saada käyttöön näitä helpottavia ominaisuuksia. (Unity Development with Visual Studio Code.)

2.3 C#

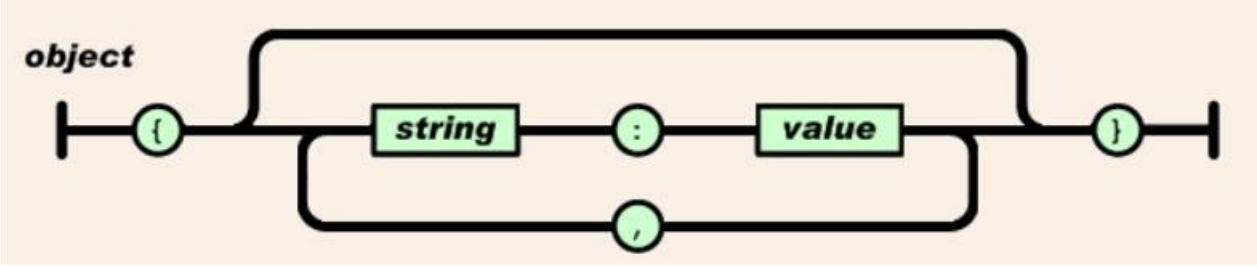
C# on moderni, tyyppiturvallinen ja olioperustainen ohjelmointikieli. C# on suunniteltu käyttämään Microsoftin .Net Frameworkia, joka tarjoaa ajonaikaisen testaamisen ja luokkakirjastoja, jotka yksinkertaistavat modernien komponenttipohjaisten ohjelmien luonnin ja jakamisen. C#-ohjelmointikielen kääntäjä ja .NET Framework julkaistiin tammikuussa 2002, mutta niiden kehitystä jatketaan edelleen. (C# IN NUTSHELL, 3)

C# mahdollistaa oliopohjaisen ohjelmoinnin sisältäen kaikki yleiset oliopohjaisuutta tukevat rakenteet, jotka löytyvät myös C++:sta ja Javasta. C# tukee kuten C++ ja Java olio-ohjelmoinnin yleisiä rakenteita. Toisin kuin monet muut ohjelmointikielet, C# ei itsessään sisällä ajonaikaisia kirjastoja vaan käyttää .NET Frameworkissä olevaa luokkakirjastoa, joka sisältää esimerkiksi konsoli I/O, tietoverkon sekä tiedonkäsittelyn. (C# IN NUTSHELL, 5)

C# on luotu sillä tarkoituksella, että sillä voidaan luoda kestäviä ja vakaita ohjelmistoja, ja se tuo sitä esille usealla eri tavalla. Ensinnäkin C# on tyyppiturvallinen ohjelmointikieli eli sillä luotuja ohjelmia estetään käyttämästä objekteja väärin. Kaikki koodi ja data ovat tyyppisidonnaisia eli voidaan käsitellä vain objekteja, joilla on sama tyyppi. Esimerkiksi, jos käyttäjä yrittää käyttää integraalilukuja merkkijonon kanssa, annetaan hänelle ilmoitus väärän tyypin käytöstä. C# tarjoaa myös automaattisen muistinhallinnan, jota kutsutaan usein automaattiseksi roskienkeräykseksi. Sillä tarkoitetaan tarpeettomien osoittimien, muistivuotojen sekä epäsuorien viittausten poistoa. (C# IN NUTSHELL, 6.)

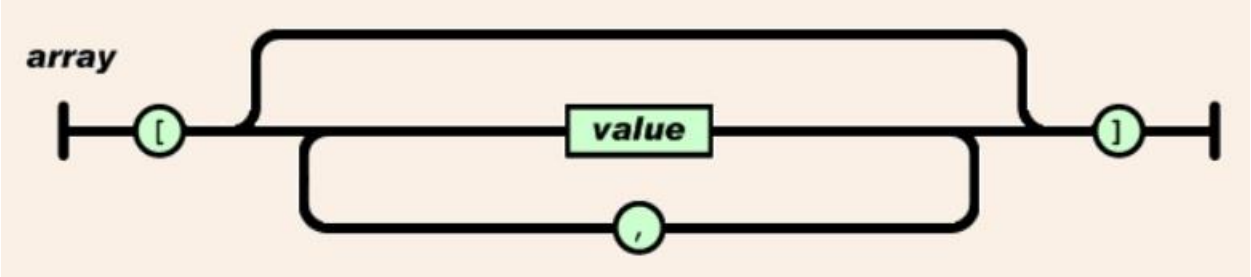
2.4 JSON

JSON on kevyt tiedostoformaatti, joka on ihmiselle selkeästi luettavaa ja helposti kirjoitettavassa muodossa. JSON-esitysmuoto hyödyntää JavaScript-syntaksia mutta on täysin riippumaton siitä, joten se sopii muihinkin ohjelmointikieliin, kuten C#- tai C++-kieliin. JSON-tiedosto voidaan muodostaa olioista ja taulukoista. Objekti sisältää Avain-arvopareja, joissa avain tulee olla merkkijonotietotyyppiä, mutta arvo voi olla useampaa eriä, kuten numeroita tai objekti tai totuusarvoja. Objektin arvo voi myös sisältää järjestetyn taulun tai toisen objektin. Avain-arvoparin avain ja arvo erotetaan toisistaan kaksoispisteellä. Objekti erotetaan aaltosulkeilla. (Introducing JSON.)



KUVA 2. Avain-arvoparin rakenne (mukaillen Introducing JSON)

Järjestetyssä taulussa on vain arvoja, jotka erotetaan toisistaan pilkulla. Taulussa olevat arvot tulee laittaa hakasulkujen sisälle, kuten alla olevassa kuvassa. (KUVA 3)



KUVA 3. Järjestetyn taulun rakenne (mukaillen Introducing JSON)

Projektissa hyödynnetty JSON-tiedosto koostuu taulukoista. Taulukon sisällä oleva data on objektimuodossa. Käytän projektissa SimpleJSON-nimistä valmista helppokäyttöistä JSON-jäsentäjää/rakentajaa, koska ei ole ajallisesti järkevää rakentaa omaa jäsentäjää vain tätä prototyyppiä varten. SimpleJSON:n avulla pääsee nopeampaa kehittämään itse sovellusta. SimpleJSON on MIT-lisensoitu, eli kaikki saavat käyttää sitä ilmaiseksi, miten haluavat, mutta tekijä ei ota vastuuta, jos siitä aiheutuu käyttäjälle harmia. SimpleJSONin ohjelmoinut henkilö löytyy GitHubista käyttäjänimellä Bunny86. (SimpleJSON.)

3 TOIMEKSIANTO

Toimeksianto tuli Centria-ammattikorkeakoulun TKI:ltä, kun kävin pyytämässä ehdotuksia opinnäytetyöksi. TKI:ltä kerrottiin minulle, että satama-alueelle olisi hyvä toteuttaa laivojen sijoittamista helpottava sovellus. Sovelluksen haluttiin kykenevän esittämään tietoa kolmiulotteisesti, jotta siitä saataisiin selkeä niin suunnittelijalle kuin muillekin käyttäjille. Ohjelma oli myös hyvä saada näkyviin useampaan pisteeseen yhtäaikaisesti, jotta suunnitelmat saataisiin toteutukseen mahdollisimman nopeasti. Halutiin myös mahdollisuus panostaa enemmän sovelluksen graafiseen puoleen, ja tämän takia ehdotin työkaluksi Unity-pelimoottoria. Se mahdollistaisi aiemmin mainittujen ominaisuuksien toteuttamisen ja mahdollisti minulle C#-ohjelmointikielen käytön.

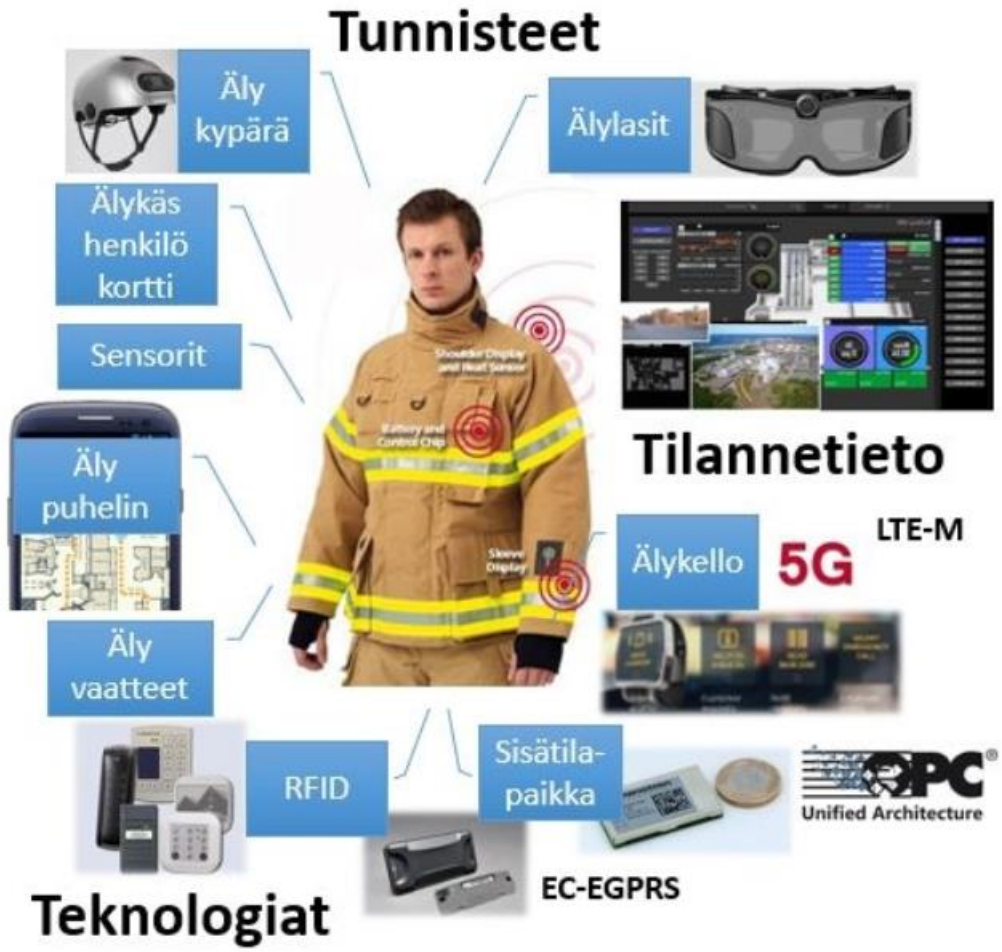
TKI:n mielestä ehdotus oli hyvä, ja aloimme miettiä tarkempia vaatimuksia projektille. Päädyimme tekemään vain prototyypin kyseisestä sovelluksesta, koska projekti olisi muuten liian laaja yhdelle henkilölle opinnäytetyöhön. Projekti sisälsi laivojen luonnin graafisen käyttöliittymän avulla, niiden liikuttamisen oikeille paikoille sekä oikeiden laivojen paikkatietojen hakemisen. Kaikki projektissa käytetyt kolmiulotteiset objektit olivat Unitystä suoraan saatavia valmiita palasia. Suurin osa projektissa käytettävästä koodista tuli tehdä itse, mutta graafiseen puoleen ei tulisi panostaa liikaa aikaa. Prototyyppi tuli kuitenkin rakentaa niin, että sitä on halutessa selkeä jatkokehittää.

3.1 TKI-BILINE

> BILINE-hankkeen idea on luoda Kokkolan suurteollisuusalueelle digitaalinen turvallisuuden kokonaiskuva, johon kuuluvat alueen ihmiset, koneet kuin laitteetkin. Hankkeen toteutusaika on vuodesta 2016 vuoteen 2019 asti, tuona aikana on tarkoitus tuottaa digitaalisia ratkaisuja, joilla helpotetaan työntekoa työturvallisuutta parantaen. Digitaalisten ratkaisujen toteutuksessa käytetään erilaisia teknologioita, kuten sensoriverkkoja, paikannusjärjestelmiä ja mobiiliohjelmistoratkaisuja. Näissä kaikissa hyödynnetään teollisen internetin viimeaikaisimpia innovaatioita. Innovaatiot tuodaan todelliseen teollisuusympäristöön, jossa niistä kehitetään käytännöllisiä sovellutuksia, jotka yritykset voivat ottaa käyttöön tai jatkokehitykseen.

Tarkoituksena on luoda alueen yritysten kannalta parhaat ratkaisut parhailla mahdollisilla teknologioilla. Jotta tähän tavoitteeseen päästään, vaatii se hankkeen kautta ja maailmalta tulevien uusien innovaatioiden tutkimista ja vertailua. Uuden teknologian on toimittava aidossa ympäristössä helposti, luotettavasti

ja sujuvasti, sen vuoksi käytetyn tekniikan ympärille rakennetaan sovellukset, jotka on räätälöity kohdeyrityksen tarpeiden mukaan. Hankkeen tuloksien tarkoituksenmukaisuus toteutetaan sovelluksille tehdyillä testeillä niin laboratorio- kuin kenttäolosuhteissa. Alla olevassa kuvassa 4 näkyy BILINE-hankkeessa käytettäviä tunnisteita ja teknologioita. (Toivanen 2016, 8.)



KUVA 4. BILINE-hankkeen tunnisteet ja teknologiat (Toivanen 2016, 8)

Hankkeessa rakennetaan ympäristö digitaalisen tunnistusteknologian ympärille, jossa kokonaisturvallisuutta edistäviä digitaalisia palveluita voidaan kokeilla ja kehittää. Tässä ympäristössä kerätään ja tuotetaan tietoa, josta jalostetaan reaaliaikaista tilannetietoa ennakoivaan tilannekuvaan. Tilannetietoa tuotetaan tutkimalla kaksisuuntaisen, interaktiivisen tiedon tuottamista ja sen hyödyntämistä. Digitalisaatioista tehdään teollisille toimijoille ja loppukäyttäjille konkreettista luomalla tietopankkeja ja informaatiosisältöjä, ja integroimalla tietoa (Toivanen 2016, 9.)

4 TOTEUTUS

Työkalussa on yksi satama-allas, jonka reunoilla on ankkurointipisteitä laivoille, laivan luontisäätimet ja luonti- sekä poistonappi. Säätimillä pystyy määrittämään laivan haluttu korkeus, pituus ja leveys. Alukset voi myös siirtää oikeille paikoille valitsemalla aluksen hiirellä ja sitten valitsemalla halutun ankkurointipisteen. Graafista puolta työssä ei vielä ole muuta kuin Unityssa lisätyt suorakulmaiset palikat ja pallot, jotka esittävät satamalaituria, laivoja sekä ankkuripisteitä. Työhön on myös lisätty laivojen paikannusjärjestelmä, joka on Centria ammattikorkeakoulun TKI:n tekemä. Prototyyppi toimii tällä hetkellä vain hiirellä, mutta siihen voi lisätä kosketusnäyttöohjauksen.

Lopulliseen tuotteeseen olisi mahdollista lisätä tarkempi satama-alue esimerkiksi Unityn oman mapboxin avulla. Myös laivojen luomisen voi toteuttaa niin, että kun aluksen koordinaatit tulevat tietyn koordinaatiston sisään, se luodaan ohjelmaan, josta sen voi siirtää haluamalleen paikalle. Lopullinen tuote voisi myös pitää sisällään lastattavien tuotteiden tietoja sekä lastauslaitteistoihin liittyviä tietoja, kuten sijainnin ja onko se käytössä vaiko ei. Suunnitelman muuttaminen useammalla laitteella kerralla olisi myös mahdollista, mutten tiedä, onko se kaikista järkevin ratkaisu.

Aluksi tulee laatia yleinen suunnitelma prototyypin toteutusta varten. Tässä vaiheessa tulee päättää sovelluksen ominaisuudet sekä niiden toteutustavat. Sovelluksen yleinen ulkonäkö olisi hyvä olla valmiina tässä vaiheessa, vaikka jätettäisiinkin vielä tilaa muutoksille.

Laivojen luonti toteutetaan ensimmäisenä, koska siitä on alustava suunnitelma ja toteutustavasta on jo kokemusta peliohjelmointiopintojen ansiosta. Vaikkei prototyypissä haluta panostaa ulkonäköön, tulee satama-alueen muodostus toteuttaa mahdollisimman aikaisessa vaiheessa, koska sen saa nopeasti toteutettua Unityn valmiista komponenteista. Tämän jälkeen voi keskittyä hankalempiin asioihin, kuten kuinka siirtää laivoja luodussa ympäristössä ja kuinka hyödyntää TKI:n tekemää laivojen paikannusjärjestelmää.

4.1 Suunnitteluvaihe

Aluksi suunnitellaan, mitä ominaisuuksia prototyypille halutaan ja kuinka ne kannattaisi toteuttaa. Laivojen luonti oli jo mietitty valmiiksi, ja se haluttiin toteuttaa graafisen käyttöliittymän avulla, jossa käyttäjä voi nappia painamalla poistaa ja luoda laivoja. Graafiseen käyttöliittymään tuli myös säätimet laivan pituudelle, leveydelle ja korkeudelle.

Tämän jälkeen suunniteltiin, miten satama-alueen ja sen ankkuripisteiden kuvaaminen toteutetaan. Päädyttiin toteuttamaan se Unityn valmiilla palikoilla ja palloilla, joista palikat kuvaavat laivoja ja laituria ja pallot ankkuripisteitä. Laivoilla ja laiturilla on käytössä eri materiaalit, jotta ne on helpompi erottaa toisistaan. Käyttäjä ei voi lisätä itse ankkuripisteitä tai muuten muokata satamaa, vaan ainoastaan luoda uusia laivoja ja siirrellä niitä halutuille paikoille.

Laivojen siirtely haluttiin toteuttaa hiiren avulla, joten päädyttiin tapaan, jossa halutun laivan voi valita siirtämällä kursorin laivan päälle ja painamalla hiiren vasenta nappia. Valitun laivan erottaa siitä, kun laivan väri muuttuu sinisestä valkoiseksi. Laivan valinnan voi peruuttaa painamalla mihin tahansa alueella olevaan tyhjään tilaan tai laituriin. Kun laiva on valittu, sen voi siirtää halutulle paikalle painamalla jotain satama-alueella olevista ankkuripisteistä, jolloin laiva siirtyy ankkuripisteen päälle niin, että ankkuripiste on keskellä laivan keskiosaa. Laiva myös kääntyy ankkuripisteelle asetetun kulman suuntaisesti.

Centria-ammattikorkeakoulu TKI:n laivanpaikannusjärjestelmän data löytyy internetsivulta, joten ohjelmassa tulee toteuttaa tiedon haku tältä. Data on tiettävästi JSON-muodossa, joten Unity-projektiin joudutaan lisäämään JSON-jäsentäjä, jotta saadaan haettua JSON:sta tarvittava tieto. Datan käyttöönottoa varten tulee pohtia koordinaatiston luontia, joka mahdollistaa laivan luonnin silloin, kun laivan on tiettyjen koordinaattien sisäpuolella. Haku tulee suorittaa heti ohjelman käynnistyessä ja tämän jälkeen vain tietyin väliajoin, ettei järjestelmää kuormiteta turhaan.

Viimeisenä suunniteltavana asiana tulee reaaliaikaisen suunnitelman jakaminen. Tiedon jakaminen haluttiin toteuttaa luomalla prototyypistä eräänlainen moninpeli, mikä mahdollistaisi useamman laitteen liittämisen sovellukseen ajon aikana. Tässä tulee ottaa huomioon, ketkä käyttäjät saavat muokata suunnitelmaa. Tämä asia on viimeisenä toteutuslistalla, sillä se vaatii kaikista eniten työtä niin ohjelmoinnin kuin suunnittelun osalta.

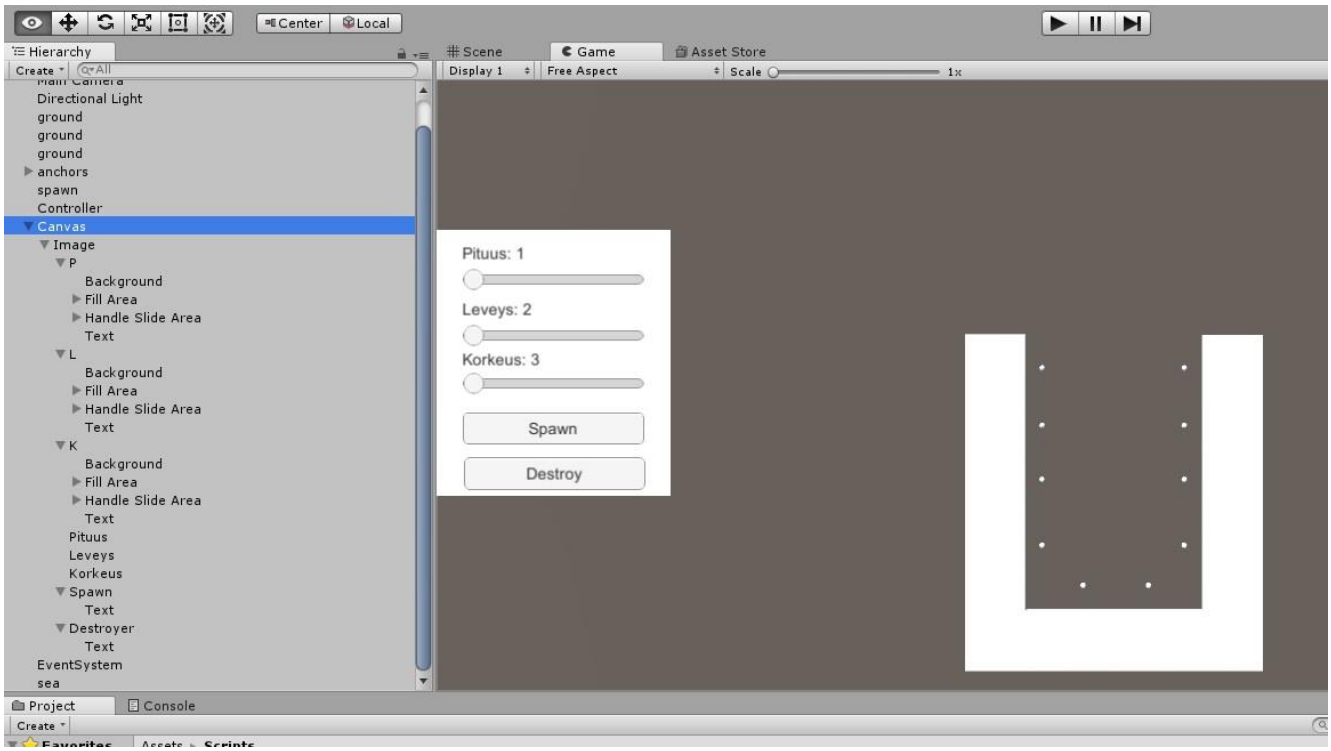
4.2 Satama-alueen pohjan muodostaminen

Satamaa varten luodaan Unityssä uusi kolmiulotteinen ”maisema” johon tulee lisätä kaikki, minkä käyttäjä näkee. Satama-alueen pohja kopioitiin kokkolalaisen satama-alueen suurpiirteisen muodon pohjalta. Unityssä tehtiin U-muotoinen laiturialue, jonka sisälle tuli ankkuripisteitä. Ankkuripisteet ovat Unityssä valmiina olevia palloja, joita voi sijoittaa haluamalleen paikalle. Ankkuripisteitä ei voi luoda käsin sovelluksen kautta vaan niitä tulee lisätä tarpeen mukaan Unityn kautta.

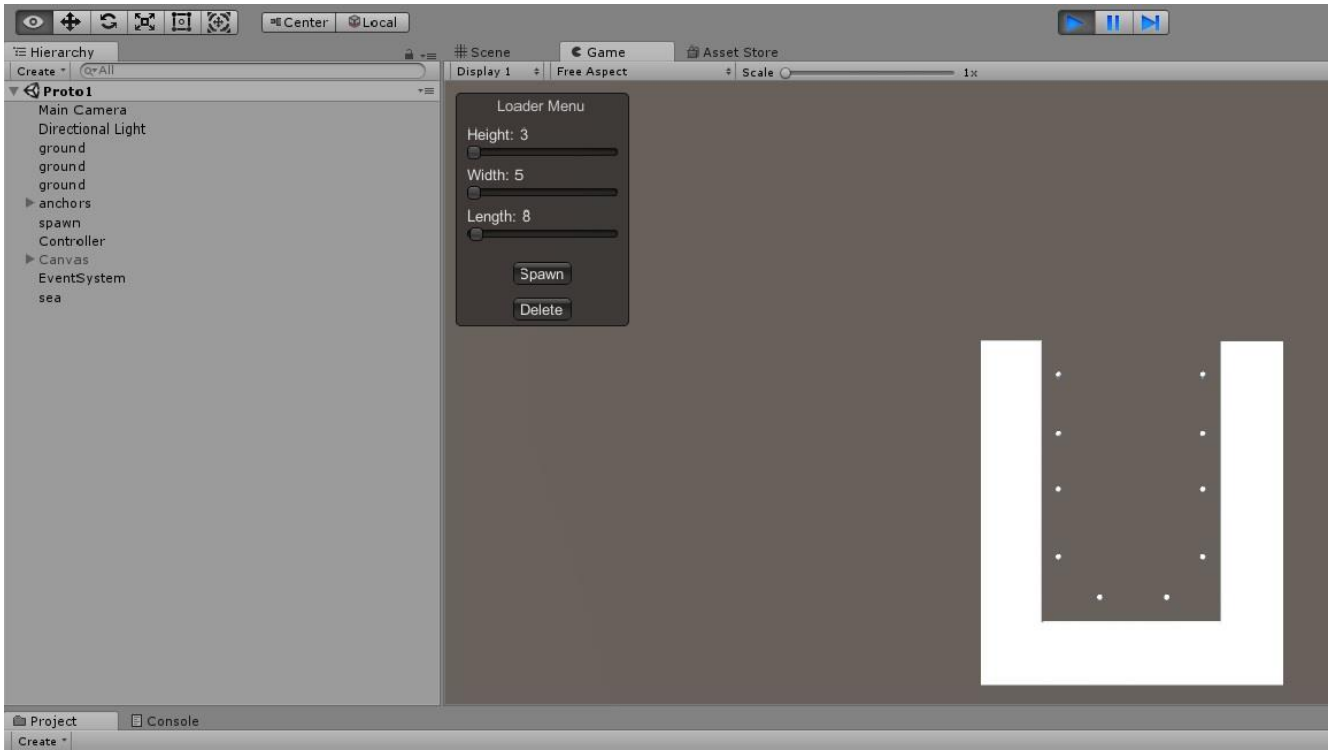
Sovellukseen selvitettiin realistisen satama-alueen pohjan muodostamisen mahdollisuutta eri kartoitusrajapintojen avulla, ensiksi ArcGISin avulla ja myöhemmin suoraan Unityyn asennettavan kartoitusrajapinnan Mapbox avulla. ArcGIS ei lopulta ollut varteenotettava vaihtoehto, koska sen liittäminen tähän sovellukseen oli liian haastavaa, ja koska Unity tarjosi mahdollisuuden käyttää suoraan heidän ohjelmaansa luotua rajapintaa tuli Mapboxista parempi vaihtoehto. Lopulta päädyttiin kuitenkin jättämään satama-alue alkuperäiseen muotoon, koska sovelluksen graafista puolta ei nähty yhtä tärkeäksi, kuin muuta toiminnallisuutta (Mapbox).

4.3 Laivan luonti

Laivan luonti toteutettiin graafisen käyttöliittymän avulla, koska se tekee siitä selkeän suunnittelijalle. Näin suunnittelija pystyy helposti kokeilemaan erikokoisten laivojen sijoittelua satama-alueella. Aluksi käyttöliittymä luotiin Unityn UI:n valmiista palikoista, mutta sitten päädyttiin käyttämään Unityn GUI:ta, koska sen avulla saatiin ohjelmoida hieman lisää ja kokeilla uuttaa tapaa tehdä käyttöliittymän. Käyttöliittymiä voi vertailla alempana (KUVA 5 ja KUVA 6).



KUVA 5. UI-versio projektin näkymästä



KUVA 6. GUI-versio projektin näkymästä

UI:lla pystyttiin tehdä käyttöliittymä valmiista palikoista ja vähemmällä ohjelmointi määrällä. UI:n avulla on myös todella helppo vaikuttaa sovelluksen ulkonäköön lisäämällä nappeihin esimerkiksi väriä

tai kuvia ja muutokset nähdään välittömästi Unityssä. Sama onnistuu myös GUI:lla, mutta muutokset nähdään vasta, kun laittaa sovelluksen päälle, koska sen osat luodaan koodin kautta.

UI-elementeillä luotua käyttöliittymää varten Unity-projektin maisemaan tulee lisätä piirtoalue, jolle UI-elementit lisätään. Ilman piirtoaluetta UI-elementit eivät tule näkyviin. Tällaisessa elementeistä rakennetussa käyttöliittymässä elementtien sijoittelu onnistuu Unityn käyttöliittymän kautta hiirellä siirtämällä. Lisäsin piirtoalueen sisälle tyhjän kuvan, joka toimii säätimien taustana. Kuvan päälle lisättiin kaikki halutut säätimet, niiden tekstit sekä napit. Säätimien päällä olevat mittatiedot tulevat säätimen senhetkisestä numerosta. Kun napit ja säätimet olivat halutuilla paikoilla, ohjelmoitiin napeille logiikkaa. Laivojen luontiin ja poistoon tarkoitetun koodin näkee alla (KUVA 7).

SpawnController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class SpawnController : MonoBehaviour {
    public GameObject ship;
    public GameObject spawnPoint;
    public int shipCount;
    public Button destroy, spawn;
    public Slider length, width, height;
    public Text pituus, leveys, korkeus;

    // Use this for initialization
    void Start() {
    }

    // Update is called once per frame
    void Update() {
        pituus.text = length.value.ToString();
        leveys.text = width.value.ToString();
        korkeus.text = height.value.ToString();
    }

    public void Spawn() {
        GameObject s = ship;
        s.transform.localScale = new Vector3(height.value, width.value, length.value);

        Instantiate(s, spawnPoint.transform.position, spawnPoint.transform.rotation);
    }

    public void DeleteShip()
    {
        if (ShipController.selectedShip != null)
            Destroy(ShipController.selectedShip);
    }
}
```

KUVA 7. Laivan luonti/poisto-koodi

Laivojen luonnissa käytettävät liukusäätimet tehtiin siis kahteen kertaan: ensiksi valmiista paloista ja toisella kertaa koodin avulla, mikä tuntui paljon paremmalta vaihtoehdolta niin kehittymisen kannalta

kuin ulkonäöllisesti. Laivan luontia varten käyttäjä valitsee haluamansa mitat ja painaa luontipainiketta, jolloin säätimien arvojen kokoinen alus luodaan hieman kuvassa näkyvän satama-alueen ulkopuolelle, josta käyttäjä saa valita sen ja siirtää haluamalleen paikalle. Tarpeettoman laivan käyttäjä voi poistaa valitsemalla sen ja painamalla ”Delete”-nappia. Alla näkyy kuva koodista, jolla loin GUI-näkymän (KUVA 8).

SpawnController.cs ● GUIScript.cs × ShipJSONData.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GUIScript : MonoBehaviour {

    float heightValue = 3.0f;
    float widthValue = 5.0f;
    float lengthValue = 7.5f;

    string height, width, length;

    void OnGUI() {
        height = "Height: " + heightValue.ToString();
        width = "Width: " + widthValue.ToString();
        length = "Length: " + lengthValue.ToString();

        heightValue = GUI.HorizontalSlider(new Rect(20, 55, 130, 30), Mathf.Round(heightValue), 3.0f, 10.0f);
        widthValue = GUI.HorizontalSlider(new Rect(20, 90, 130, 30), Mathf.Round(widthValue), 3.0f, 10.0f);
        lengthValue = GUI.HorizontalSlider(new Rect(20, 125, 130, 30), Mathf.Round(lengthValue), 5f, 30.0f);

        GUI.Box(new Rect(10, 10, 150, 200), "Loader Menu");
        GUI.Label(new Rect(20, 35, 130, 30), height);
        GUI.Label(new Rect(20, 70, 130, 30), width);
        GUI.Label(new Rect(20, 105, 130, 30), length);

        if (GUI.Button(new Rect(60, 155, 50, 20), "Spawn")) {
            GameObject spawnPoint = GameObject.Find("spawn");
            GameObject s = Instantiate(Resources.Load("Prefabs/laiva", typeof(GameObject)), spawnPoint.transform.position, spawnPoint.transform.rotation) as GameObject;
            s.transform.localScale = new Vector3(heightValue, widthValue, lengthValue);
        }

        if (GUI.Button(new Rect(60, 185, 50, 20), "Delete")) {
            if(ShipController.selectedShip != null)
                Destroy(ShipController.selectedShip);
        }
    }
}
```

KUVA 8. GUI-luontikoodi

4.4 Laivojen logistinen organisointi

Prototyypissä haluttiin laivojen organisoinnin olevan käyttäjälle mahdollisimman helppoa ja selkeää. Tästä syystä vain laivoja voi siirrellä, tällöin vähennetään mahdollisia käyttäjävirheistä aiheutuvia ongelmia ja pidetään sovelluksen käyttö yksinkertaisena.

Laivan valitseminen toteutettiin säteensuuntausmenetelmällä. Unityssä on mahdollista merkata objekteja eri merkinnöillä. Näiden merkintöjen avulla voidaan erotella objektit toisistaan, jolloin säteensuuntausmenetelmän avulla tiedetään mihin objektiin osutaan ja miten osuman tuomaan tietoon reagoidaan. Hiiren vasenta nappia painaessa lähetetään säde suoraan hiiren osoittimesta kameran osoittamaan suuntaan, joka objektiin osuessa palauttaa tietoa, kuten esimerkiksi osutun objektin merkki tai osumakohdan

sijainti. Laivoille sekä ankkuripisteille on annettu niitä kuvaavat merkinnät, joiden avulla toteutetaan koodissa eri toimintoja.

Laivan siirtäminen oikealle paikalle onnistuu painamalla alusta ja sen jälkeen painamalla haluttua ankkuripistettä. Kun käyttäjä valitsee luodun laivan painamalla hiiren vasenta nappia osoittimen ollessa aluksen päällä, se muuttuu sinisestä valkoiseksi, ja tämän jälkeen käyttäjä painaa jostakin ankkuripaikkoja merkitsevästä pallosta, ja alus siirtyy sen kohdalle ja aluksen väri vaihtuu taas siniseksi. Käyttäjä voi myös perua valintansa painamalla mihin vain tyhjään tilaan, jolloin alus muuttuu takaisin siniseksi. Alla olevassa kuvassa näette, kuinka laivat valitaan sekä miten laivan väri vaihdetaan valinnan yhteydessä. (KUVA 9)

ShipController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ShipController : MonoBehaviour {

    public static GameObject selectedShip;
    MeshRenderer meshRenderer;
    public bool selected;

    // Use this for initialization
    void Start () {
        selectedShip = null;
        selected = false;
    }

    // Update is called once per frame
    void Update () {
        if (Input.GetMouseButtonUp(0)) {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            Physics.Raycast(ray, out hit);

            if (hit.collider.CompareTag("Ship")) {
                if (!selectedShip) {
                    selectedShip = hit.collider.gameObject;
                    meshRenderer = selectedShip.GetComponent<MeshRenderer>();
                    meshRenderer.material.color = Color.white;
                    selected = true;
                } else {
                    ChangeColor();
                    selectedShip = hit.collider.gameObject;
                    meshRenderer = selectedShip.GetComponent<MeshRenderer>();
                    meshRenderer.material.color = Color.white;
                    selected = true;
                }

            }
            else if (hit.collider.CompareTag("Anchor") && selected) {
                selectedShip.transform.position = hit.collider.transform.position;
                selectedShip.transform.rotation = hit.collider.transform.rotation;
                ChangeColor();
            }
            else {
                if(selected)
                    ChangeColor();
            }
        }

        if(Input.GetKeyUp(KeyCode.Escape)){
            Application.Quit();
        }
    }

    void ChangeColor() {
        meshRenderer.material.color = Color.blue;
        selectedShip = null;
        selected = false;
    }
}
```

KUVA 9. Laivan valitseminen ja värin vaihto

4.5 SimpleJSON laivojen paikannusjärjestelmä

SimpleJSON otettiin prototyypin tekemiseen mukaan, koska Centria-ammattikorkeakoulun TKI:n tekemä paikannusjärjestelmä antoi niihin liittyvän datan JSON-muodossa. Aluksi yritettiin itse tehdä jäsentäjä, jotta TKI:n järjestelmän tuottama data saataisiin purettua, mutta päädyttiin nopeasti siihen tulokseen, ettei kannata tehdä jotain niin haastavaa itse, kun se löytyy jo jonkun toisen tekemänä ilmaisella lisenssillä. Ongelma jäsentäjän itse tekemisessä oli se, ettei sellaisen tekemisestä ollut aiempaa kokemusta ja data oli vaikeasti jäsennettävässä muodossa.

SimpleJSON käyttöönotto oli niin helppoa, että riitti, kun ladattiin siihen kuuluvat tiedostot tekijän jakamasta GitHub-säilytyspaikasta ja lisättiin ne Unityn prototyypin liitännäiset-kansioon ja kirjoitettiin koodiin referenssin SimpleJSONiin. Sen avulla säästettiin siis paljon aikaa ja samalla pystyttiin katsomaan SimpleJSONin avulla, kuinka jatkossa pystyy suunnittelemaan ja ehkä jopa rakentamaan jäsentäjän itse omaa tarkoitusta varten.

Kun SimpleJSON oli otettu käyttöön, tein Unityyn uuden C#-koodin, jonka sisällä hain TKI:n tekemästä paikkatieto-JSON:sta tarvittavat tiedot. Alla olevasta kuvasta 10 näkee haun suorittavan C#-koodin. Tämä koodi palauttaa JSON-listalla ensimmäisenä olevan laivan koordinaatiotiedot. Kyseinen koodi ajetaan 5min välein, jottei järjestelemä kuormitu liikaa haun takia. Ajastinta voidaan muokata pienemmäksi tai suuremmaksi helposti, mutta ohjelmassa ei haluttu antaa käyttäjälle mahdollisuutta muokata sitä, ettei aiheudu ongelmia. Tämä ominaisuus lisättiin prototyyppiin lopulta vain jatkokehitystä ajatellen, koska koordinaatiston luonti sekä käyttöönotto olisivat vaatineet liikaa työtä siihen nähden, että prototyyppi on alustavasti itse sovelluksen demoamista varten. Haettua tietoa ei siis käsitellä vielä millään tavoin, mutta tämä paikkatiedon haku haluttiin tehdä, että sitä voi esitellä jatkokehitysmahdollisuutena.

SpawnController.cs ● GUIScript.cs ShipJSONData.cs ×

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using SimpleJSON;

public class ShipJSONData : MonoBehaviour {
    public string JSONString;
    public string url = "http://bilineapiapp.azurewebsites.net/Vehicles/Ship";
    // Use this for initialization
    void Start(){
        StartCoroutine(UpdateJSON());
        StartCoroutine(Timer());
    }

    IEnumerator Timer(){
        yield return new WaitForSecondsRealtime(300.0f);
        StartCoroutine(UpdateJSON());
    }

    IEnumerator UpdateJSON(){
        using (WWW www = new WWW(url)){
            yield return www;
            JSONString = www.text;

            var N = JSON.Parse(JSONString);
            var deviceid = N["value"][0]["deviceid"];
            var lat = N["value"][0]["lat"];
            var lon = N["value"][0]["lon"];
        }
    }
}
```

KUVA 10. Laivan paikkatietojen haku SimpleJSON avulla

5 POHDINTA JA JOHTOPÄÄTÖKSET

Työkalua tehdessä suurin ongelma oli itselleni asettamani liian suuret vaatimukset, koska kyseessä oli kuitenkin prototyyppi, jolloin ei olisi ollut tarve suunnitella ja yrittää toteuttaa graafista puolta ja muita lopulliseen tuotteeseen liittyviä ominaisuuksia. Itse prototyypin ohjelmointi aiheutti myös hieman ongelmia, kuten valitun laivan selvästi esittäminen väriä vaihtamalla. TKI:n paikannusjärjestelmän JSON-muotoinen data oli myös vaikea saada toimimaan, joten turvauduttiin valmiiseen SimpleJSON-jäsentäjään.

Tehtävä itsessään oli hyvin mielenkiintoinen, koska en ollut aiemmin ajatellut Unityn mahdollisuuksia muussa kuin pelien tekemisessä. Tätä työtä tehdessä aloin ajatella Unityä aivan uudelta kannalta, eikä pelien kehittäminen ollutkaan enää se kaikista mielenkiintoisin urapolku, vaan erilaisten hyötyohjelmistojen luominen. Tämän työn avulla olen myös saanut muutaman kutsun haastatteluihin ja työn aihe on ollut monien haastattelijoiden mielestä mielenkiintoinen.

Prototyyppi tässä muodossa ei ole käytettävissä, mutta pienellä jatkokehityksellä se pitäisi saada nopeasti käytettäväksi. Lisäämällä jonkinlaisen koordinaatiston prototyyppiin saisi otettua käyttöön laivojen paikkatietohin perustuvan laivojen luonnin. Tämän avulla voisi luoda alukset sovellukseen heti, kun ne ovat tietyllä alueella, ja samalla poistamaan, kun ne poistuvat alueelta. Sovellusta voisi myös jatkossa parantaa graafisesti lisäämällä laivoille omat kolmiulotteiset mallit sekä muokkaamalla satama vastaamaan oikeaa kohdetta. Käyttäjälle olisi myös hyvä lisätä työkalut, jotka mahdollistavat satama-alueen ja sen ankkuripisteiden muokkauksen. Sovellukseen tulisi miettiä useampi mahdollinen tapa toteuttaa suunnitelman jakaminen. Kyseessä on kuitenkin pelimoottori, joten jonkinlainen kirjautumisjärjestelmä voisi olla paras tapa toteuttaa suunnitelman jaon, mutta myös esimerkiksi sovellusta ajavan tietokoneen ruudun näkymää voisi jakaa. Satama-alueen ankkuripisteille tulisi myös luoda jonkinlainen tarkistus, mahtuuko kyseinen alus halutulle paikalle. Myös satama-alueen muiden työkalujen seuraaminen ja mallintaminen olisi mahdollista, mikä taas auttaisi sijoittamaan oikeat työkalut laivojen tarpeiden ja lastaussijainnin perusteella. Myös SimpleJSONsta voisi päästä eroon tekemällä itse juuri tähän sovellukseen soveltuvan JSON-jäsentäjän. Haluaisin myös perehtyä tarkemmin mihin kaikkeen Unity oikeasti pystyisi muissa logistiikka-alan sovelluksissa, kuten varastoissa tai logistiikkaterminaaleissa.

Opin prototyyppiä tehdessä hieman ohjelmoinnista C#-kielellä ja JSONin käyttämistä eri projekteissa. Kehityin myös projektin suunnittelussa, koska alussa tein todella paljon virheitä asioiden tärkeysjärjestyksissä sekä keskityin yleisesti monta kertaa ihan vääriin asioihin, kuten laivojen mallintamiseen tai muuhun graafiseen puoleen, johon ei alun perin haluttu edes panostaa. Se myös herätti mielenkiinnon työskennellä erilaisilla aloilla, jossa ei välttämättä ole niin kehittyneitä sovelluksia, mutta joissa on paljon potentiaalia.

LÄHTEET

Albahari, B., Drayton, P. & Neward, T. 2002. C# IN A NUTSHELL. O'Reilly Media.

Introducing JSON. Saatavissa: http://json.org/ Luettu 6.5.2019.

Mapbox. Saatavissa: https://www.mapbox.com/unity/ Luettu 14.6.2019.

Paczkowski, L. 2018. Replacing MonoDevelop-Unity with Visual Studio Community starting in Unity 2018.1. Saatavissa: https://blogs.unity3d.com/2018/01/05/discontinuing-support-for-monodevelop-unity-starting-in-unity-2018-1/ Luettu 7.5.2019.

SimpleJSON. Saatavissa: https://github.com/Bunny83/SimpleJSON. Luettu 6.5.2019.

Toivanen L. 2016. BILINE Use Cases. Saatavissa: http://projekti.centria.fi/data/liit-teet/36ce44f328a746f1be1292276ebb840a.pdf Luettu 6.5.2019.

Unity. https://unity3d.com/unity?_ga=2.31774775.88792374.1556447357-353291592.1556447357 Luettu 6.5.2019.

Unity Development with Visual Studio Code. https://code.visualstudio.com/docs/other/unity Luettu 7.5.2019.

Unity Documentation. https://docs.unity3d.com/Manual/in-dex.html?_ga=2.32290999.88792374.1556447357-353291592.1556447357 Luettu 6.5.2019.

Visual Studio. https://visualstudio.microsoft.com/vs/ Luettu 12.5.2019.