SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

VIRTUAALIVANHUSPELIN MODERNISOINTI WEB- JA VR-YMPÄRISTÖÖN

TEKIJÄ: Henri Hyyryläinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Henri Mikael Hyyryläinen	
Työn nimi Virtuaalivanhuspelin modernisointi web- ja VR-ympäristöön	
Päiväys 26.5.2020	Sivumäärä/Liitteet 20
Toimeksiantaja/Yhteistyökumppani(t) Savonia AMK	

Tiivistelmä

Tässä opinnäytetyössä modernisoitiin Savoniassa toteutettu Virtuaali Vanhus (VIVA) peli (2012). Peli oli toteutettu Unityllä webselaimessa pelattavaksi, mutta nykyään selaimet estävät pluginien suorittamisen selaimessa, joten peliä ei voi pelata. Opinnäytetyön tavoitteena oli pelin kääntäminen nykyisissä selaimissa toimivaksi. Tavoitteena oli myös, että peli toimii VR-laseissa (Vive, Oculus, Microsoft Hololens, Microsoft MR-lasit). Lisäksi työhön kuului viva.savonia.fi -sivuston uusiminen siten, että peli kommunikoi suoraan taustajärjestelmän kanssa (pelin loki).

Sovelluksen modernisointi aloitettiin kääntämällä aiemman version lähdekoodit uusimmalle versiolle pelimoottorista. Tämän jälkeen peli käännettiin nykyaikaiseksi WebGL-versioksi. Mikäli kaikki tämä oli sujunut oikein, aloitettiin eri VR-laitteiden tunnistamisen ja näiden tarjoamien kirjastojen integrointi sekä VR-laitteille optimointi niin motoriikan kuin käyttöliittymien suhteen.

Toteutuksen aikana tavoite muuttui. Modernisoitujen ja alkuperäisten kooditiedostojen ero osoittui uusien Unity-versioiden ominaisuuksien rinnalla liian suureksi stabiilin alkuperäisen pelin emuloinnin onnistumiseksi. Projektin tavoite siirtyi luomaan nykyaikainen runkokehikko tukemaan tulevien ominaisuuksien ja tasojen luontia.

Lopputuloksena syntyi huomattavasti suunniteltua suppeampi käännöstyö, mutta uusi runkokehikko tukee lupaavasti uusien ominaisuuksien lisäämistä.

Avainsanat
Unity, videopelit, modernisointi, selain, runkokehikko

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Henri Hyyryläinen			
Title of Thesis Modernization of a virtual old game for the web and VR environment			
Date	26 May 2020	Pages/Appendices	20
Client Organisation /Partners Savonia University of Applied Sciences			

Abstract

The topic of this thesis was the modernization of the VirtualElderly (VIVA) game implemented in Savonia (2012). The game was implemented on Unity to be played in a web browser, but today browsers block plug-ins from running in the browser, so the game cannot be played. The aim of the thesis was to make the game work in current browsers. The goal was also for the game to work in VR glasses (Vive, Oculus, Microsoft Hololens, Microsoft MR glasses). In addition, the work included updating the viva.savonia.fi website so that the game communicates directly with the background system (game log).

The modernization of the application began by translating the source code of the previous version to the latest version of the game engine. The game was then translated into a modern version of WebGL. If all this had gone well, the integration of the various VR devices and the integration of the libraries they provided, as well as the optimization of the VR devices in terms of both motor and user interfaces, began.

During implementation, the goal changed. The difference between the upgraded and the original code files, along with the features of the new Unity versions, proved to be too great to succeed in emulating a stable original game. The goal of the project shifted to creating a modern framework to support the creation of future features and levels.

The result was a much smaller translation work than planned, but the new framework promises to support the addition of new features.

Keywords

Unity, Video games, modernization, browser, framework

SISÄLTÖ

1 JOHDANTO

Virtuaalitodellisuuden käyttötarkoitukset ovat tietotekniikan kärjessä olivat ne peli-, oppimis-, demonstraatio- tai mallinnustarkoituksessa.

Opinnäytetyön toimeksiantaja on Savonia-ammattikorkeakoulu. Työn lähtökohtana on vuonna 2012 Savoniassa Unityllä toteutettu Virtuaali Vanhus (VIVA) web-selainpeli, https://viva.savonia.fi/.

Network Plugin Application Programming Interface (NPAPI) (Wikimedia Foundation, Inc., 2019) on käytöstä poistunut sovellusohjelmointirajapinta (API), joka sallii selainlisäosien kehittämisen. Se kehitettiin aluksi Netscape selaimiin (Netscape Navigator 2.0) 1995. Myöhemmin se lisättiin muihin selaimiin, mutta HTML5:n kehityttyä useammat selainvalmistajat ovat poistaneet tuen tämän API:n suhteen. Google poisti kyseisen tuen kokonaan vuoden 2014 loppupuolella. Chrome 37 ilmestyttyä pluginin enablointi estettiin jopa manuaalisesti heinäkuussa 2014 (Google LLC, 2014). FireFox puolestaan poisti tuen vuoden 2016 lopussa (Mozilla Foundation, 2015).

Uusi Unity WebGL korvaa heidän NPAPI:iin perustuvan Unity WebPlayerin. (Unity Technologies, 2019). Kysyinen WebGL on HTML5:n tekniikoita käyttävä Javascript API, joka renderoi 2D ja 3D grafiikkat web selaimessa. (Unity Technologies, 2019)

Koska VIVA peli oli toteutettu Unityllä webselaimessa pelattavaksi vuonna 2012, nykyään selaimet estävät tarpeellisten pluginien suorittamisen selaimessa ja peliä ei voi pelata. Työhön kuuluu pelin kääntäminen nykyisissä selaimissa toimivaksi sekä kaikilla kuluttaja VR-laseissa (Vive, Oculus, Microsoft Hololens, Microsoft MR-lasit) toimivan version tekeminen. Lisäksi työhön kuuluu viva.savonia.fi -sivuston uusiminen siten, että peli voi kommunikoida suoraan taustajärjestelmän kanssa (pelin loki).

Tarkoituksena on myös löytää vastauksia nykyisen webselaimessa toimivan käännöksen triggereiden outoon käytökseen sekä lisätä osaamista Unityllä toteutetusta VR-pelistä.

Opinnäytetyön lopullinen tarkoitus on olla käyttökelpoinen sairaanhoitajaopiskelijoiden käytössä simuloimassa kotihoitoa vanhusten kanssa. Websovelluksen on tarkoitus toimia itsenäisesti ilman erillisiä lisälaitteita. Virtuaalitodellisuuslaseja tukemalla pelikokemuksesta saa immersiivisemman.

Työssä kehitetyn sovellusrakenteen tavoitteena oli mahdollistaa helpompi uuden sisällön tuottaminen peliin.

2 TYÖN TOTEUTUS

2.1 Alkuperäinen suunnitelma työn toteuttamiselle

Alkuperäinen versio sovelluksesta on toteuttu Unity:llä, joten sovelluksen modernisointia tehdään myös saman ohjelmiston nykyisillä versioilla.

Unity on tehokas järjestelmäriippumaton pelin kehitysympäristö ja ajonaikainen pelimoottori tukien eri sovellusalustoja. Kehittäjät pystyvät ottamaan käyttöön pelinsä mobiili-, konsoli ja tietokonealustoille. Se tukee jopa Webplayeria voidaakseen pyöriä selaimilla. (Gregory, 2014)

Sovelluksen modernisointi voidaan aloittaa kääntämällä aiemman version lähdekoodit uusimmalle versiolle pelimoottorista. Tämän jälkeen peli käännetään nykyiseksi WebGL-versioksi. Mikäli kaikki tämä on sujunut oikein, aloitetaan eri VR-laitteiden tunnistamisen ja näiden tarjoamien kirjastojen integrointi ja VR-laitteille optimointi niin motoriikan kuin käyttöliittymien suhteen.

Samanaikaisesti uusitaan viva.savonia.fi -sivusto ja pistetään WebGL-versio kommunikoimaan suoraan taustajärjestelmän kanssa.

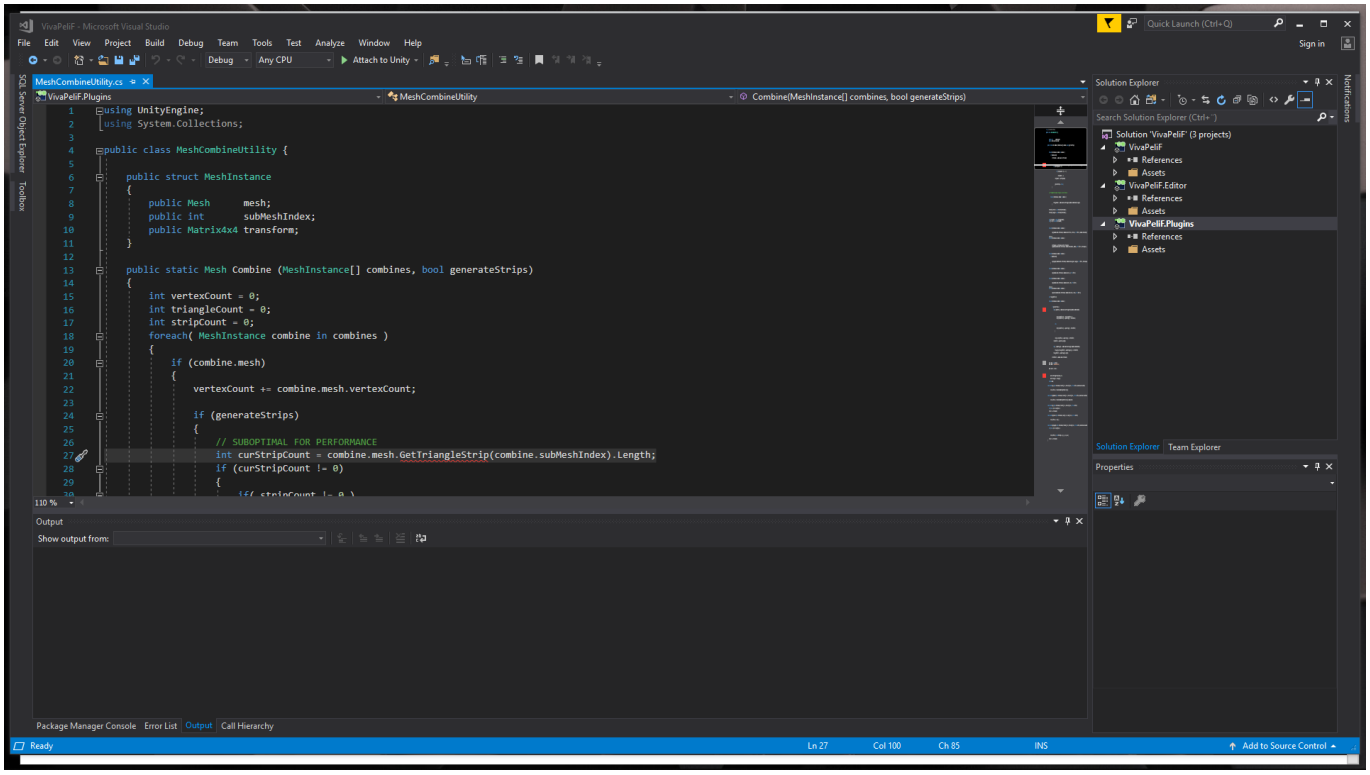
2.2 Skriptien kääntäminen C#:lle

Unity versiot alle 5.6 tukivat koodaamista JavaScriptissa, C#:lla ja Boo-kielellä (Gregory, 2014). Sen jälkeiset versiot ovat pudottaneet tuen JavaScriptille. Näin ollen vanha versio ViVasta ei toimi suoraan uudemmassa versiossa.

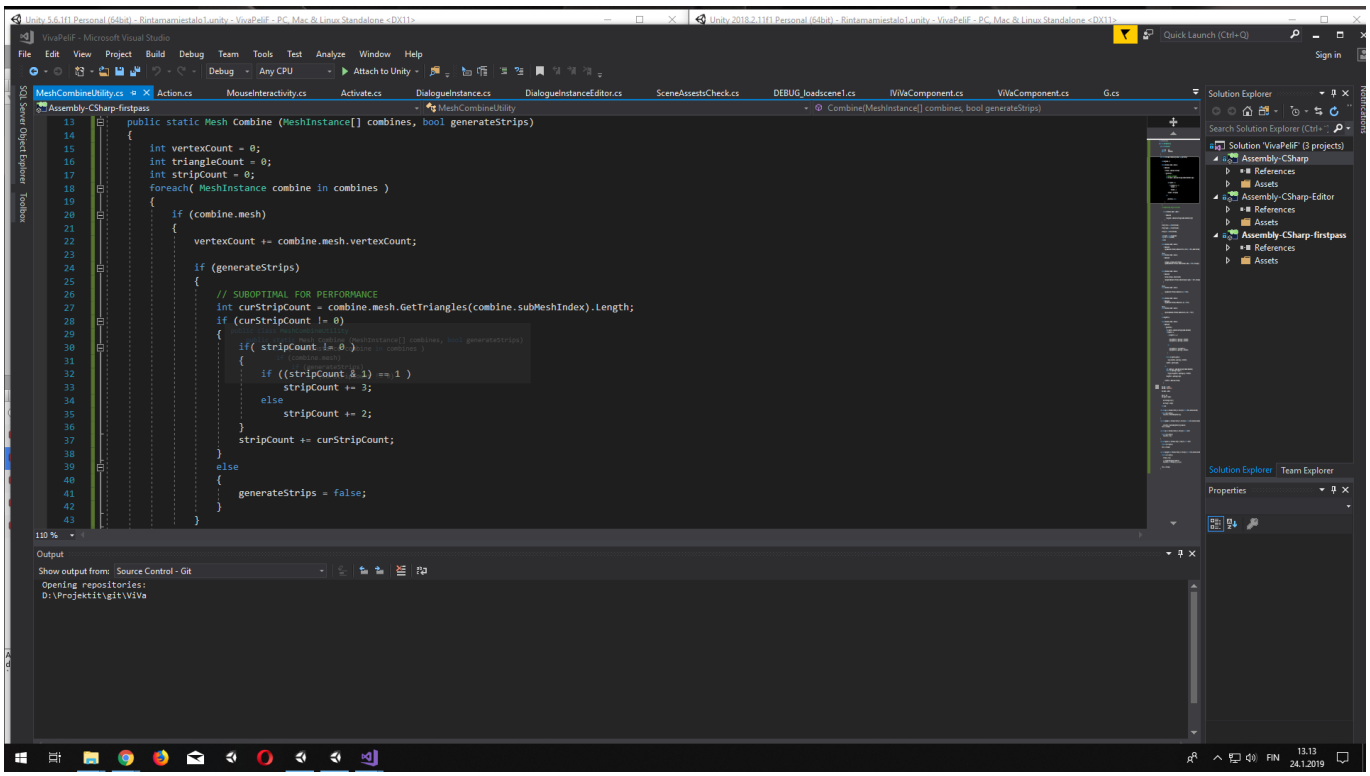
Ensimmäisenä JavaScript-koodit tuli tiedosto tiedostolta muuttaa C#-tiedostoiksi. Näille täytyi sen jälkeen tehdä muokkauksia, jotka toimivat C#:n ja Javascriptin eroina. Näihin kuului mm. boolean-tietotyypin tunnistaminen, mikäli ei-boolealla ei ollut alustettua arvoa, ajonaikaisen koodin lukeminen ja suorittaminen merkkijonosta, listojen käyttäytyminen ja tyyppimuutokset ja tunnistamiset. Myöskin Unityssä jokainen Javascript-objekti oli peritty komponenttiluokasta, jota nykyisissä versioissa ei enää ole tai ei ainakaan pidä sisällään samoja attribuutteja ja metodeja. Tätä varten jouduttiin kirjoittamaan oma ViVaComponent:ksi nimetty luokkarakenne, joka peri tämän Unityn Monobehaviour-luokan laajentaen näihin attribuutteihin. Näin ollen uudet tästä rakenteesta perityt luokkarakenteet tunnistettiin ViVaComponenteiksi. Tietenkin eroavaisuutena C# ja JavaScriptin välillä, C# ei kyennyt kutsumaan uusia methodeja lapsiluokista ilman ViVaComponentti-olion tyyppimuutosta lapsiluokaksi.

Seuraavaksi täytyi tehdä muutokset koodeihin, kuinka ne kutsuivat (eng. invoke) uusia Unityn ominaisuuksia API:n kautta. ViVa-peli oli tehty vanhemmalla versiolla Unitya, jossa viittaukset peliobjekteihin kiinnitettyihin scripteihin tapahtui suoremmin. Suurin osa methodikutsuista oli myös täysin muuttunut tai oli otettu pois käytöstä nykyisistä versioista. Kuten esimerkiksi yhdistäessään mesheja yhdeksi, verteksien tunnistaminen peliobjektin renderoinnissa eroaa modernisoidusta versiosta havainnollistettu alla olevissa kuvissa (Kuva 1) (Kuva 2).

Nykyinen GetTriagles-methodi hakee viitatun peliobjektin alimeshen kolmiot luetteloituna. Palautettu luettelo on kokonaisluku verteksi-indekseistä. Näitä indeksejä käytetään aloittamaan meshen verteksitaulu. Vanhemman GetTriangeStrip -metodin tarkempaa toimintaa ei ole tiedossa, sillä Unity ei säilytä enää dokumentaatio vanhemmille kuin 5.4-versioille. Nimestä voidaan kuitenkin päätellä sen palauttaneen samankaltaisesti meshen verteksiluvun. Katsoen, kuinka helposti muutos korjattiin tuottamaan saman tuloksen, todennäköisintä on, että metodin nimi on muutettu tarkemmin kuvaavammaksi. Myös suurin osa peliobjektien ominaisuuksista oli kapseloitu turvaamaan jäsenmuuttujiaan ja ne voitiin hakea viittaamalla tiettyihin uusiin methodeihin.

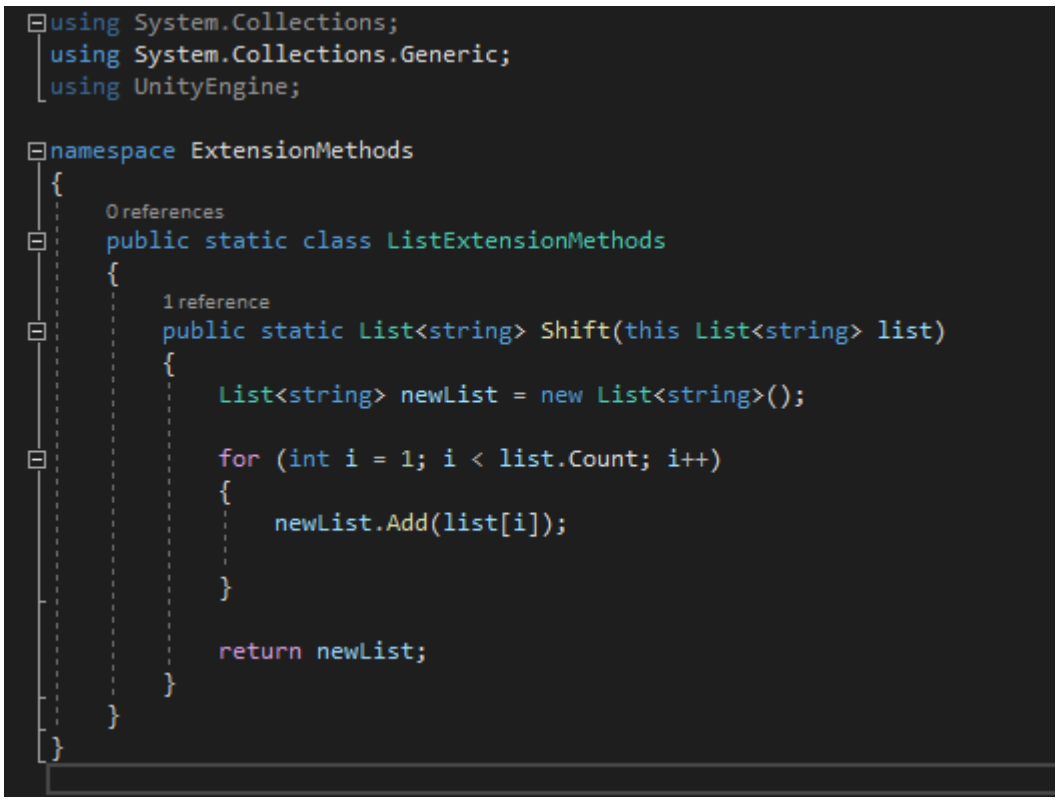


Kuva 1: Meshejen yhdistäminen vanhassa scriptissä.



Kuva 2: Meshejen yhdistämisen modernisointi uudessa scriptissä

Useimpia Javascriptin funktioita ei lyötynyt C#sta täysin sellaisenaan kuin ne alkuperäisessä kielessä toimivat. Näin ollen, mahdollistaakseen mahdollisimman tarkan samankaltaisen toiminnan kyseiset funktiota haluttiin luoda itse emuloimaan toimintamalliltaan aiempia funktioita. C#:n omaa Extension-methods-ominaisuutta käytettiin luodakseen samanlaisen tyylin skriptien kesken vähentäen refaktorointia ja nimen korjailuja. Näin jokainen lisäyksensä koottiin omaan sekä omaan nimiavaruuteensa, että omaan kansiorakenteeseensa. Kuten alla olevassa kuvassa (Kuva 3) on esitetty.



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

namespace ExtensionMethods
{
    0 references
    public static class ListExtensionMethods
    {
        1 reference
        public static List<string> Shift(this List<string> list)
        {
            List<string> newList = new List<string>();

            for (int i = 1; i < list.Count; i++)
            {
                newList.Add(list[i]);

            }

            return newList;
        }
    }
}
```

Kuva 3: Vanhojen Javascript-funktioiden emulointi lisäyksillä

Myöskin yhden vaikeuden scriptien kääntämisessä toi vanha attribuutti tässä vanhassa komponenttiluokassa, joka määritteli tarkan järjestyksen, jossa koodi tulisi suorittaa. Tämä ominaisuus oli poistunut uudesta Unity-versiosta, mutta pelin rakenteen toiminta oli riippuivainen tästä. Nykyisin scriptien toistaminen tapahtuu järjestyksessä missä ne ovat kiinnitetty kysyiseen peliobjektiin.

2.3 Toimintarungon hahmoittaminen

Alkuperäisen version kaikkia suunnitteludokumenttejä ei ollut saatavilla ja pelin toiminnallisuuksia sekä logiikkaa tuli selvittää vanhan version ja lähdekoodien perusteella.

Suurin osa vanhoista JS (Javascript) -skripteistä olivat staattisia instansseja. Ne eivät näkyneet nykyisellä suositulla tavalla Unityn inspector-ikkunassa. Näin ei kyetty hyödyntämään pelimoottorin ominaisuutta esittää tilanne ylimalkaisesti. Tämä yleensäkin toimii normaalissa ohjelmoititilanteessa, missä scriptien määrä ja ohjelmiston rakenne ovat suhteellisen pieniä. Pelikehityksen kohdalla on erityisen tärkeää hallita ja hahmottaa ohjelman kokonaisuutta, sillä muutosten tekeminen moniin skripteihin kooditasolla on sekä työlästä että aikaavievää.

Kääntäessäessä skriptejä C#: lle pelin toiminnallisesta rakenteesta pääsi jollain tasolla selkeyteen. Staattinen G-skripti pyöritti taustalla kaikkea. G-skripti alustui automaatisesti jokaisen pelikerran alussa. Jokaiseen peliobjektiin oli liitetty Action-scripti, joka peri aiemmin mainitun JS: lle ominaisen perusobjektin. Action-scriptiin oli myös liitetty erinainen listallinen peliobjekteja. Listat oli merkitty

riippuen mitä G-skriptin haluttiin tekevän näille. Nämä saattoivat vaihdella enabloimisesta, poistamisesta tai uudelleen aktivoinnista. Kun tälle skriptille annettiin Run-komento, jokaisen skriptin lista silmukoitiin G-skriptissä, joka listan mukaan teki, kuinka käskettiin jokaisessa peliobjektissa. Tällöin G myös tallensi pelitilanteen kirjoittaen tilanteeseen johtavat komennot string-objektiin.

Unityn päivittämisen myötä jokainen JS-skripti lakkasi Unityn näkökulmasta olemasta ja jokainen skripti piti päivittää inspector-ikkunaan uuteen vastaavaan käännettyyn C#-skriptiin manuaalisesti verraten vanhempaan versioon. Tämä tapa on varsin työläs, aikaavievä ja virhealtis.

2.4 Tarinarungon hahmoittaminen

Vaikka kaikista tarinoista oli aikanaan kohtuullisen tarkka juoni, nämä dokumentit eivät olleet saatavilla tämän työn aikana. Näin ollen, pelitapahtumien kulku oli hankala hahmoittaa. Se koottiin pelaamalla vanhempia versioita itse pelimoottorissa, koska viimeinkin uudet selaimet olivat jopa estäneet pluginien manuaalisen päälle laiton. Jossain tilanteissa kuitenkin jopa nämä vanhemmat versiot lakkasit pyörittämästä tarinaansa eteenpäin.

Myöskään pelistä tehdyt dokumentaatiot ja pöytäkirjat eivät avanneet tilannetta enempää ja uudelleen rakennetun pelin tarina vain pysähtyy.

Dokumentaation ja muistioiden perusteella tilanteesta ei saanut yksiselitteistä vastausta tarinan kulusta. Modernisointityössä jouduttiin tekemään arvauksia käytettävissä olevien tietojen pohjalta.

2.5 Emuloinista siirtyminen runkokehikkoon

Toteutuksen aikana tavoite muuttui. Modernisoitujen ja alkuperäisten kooditiedostojen ero osoittui uusien Unity -versioiden ominaisuuksien rinnalla liian suureksi stabiilin alkuperäisen pelin emuloinnin onnistumiseksi. Projektin tavoite siirtyi luomaan nykyaikainen runkokehikko tukemaan tulevien ominaisuuksien ja tasojen luontia.

Tämä alkoi luomalla GameManager-luokka, joka valvoi pelin lataamista ja pelaaja-prefabin valintaa viittaamatta staattisiin vanhoihin koodeihin. Tämä myös toi koodit objektinäkymään mistä kokonaisuuden hahmoittaminen oli helpompaa. Tämä luokka tuli myös valvomaan pelin moodia.

2.6 GameManager

GameManager-objekti valvoo pelin nykyisen iteration aloitusta. Ladatessa kyseisen kohtauksen se tarkistaa määritellyn pelimoodin ja valitsee pelaajaobjektin sen mukaisesti. Kyseiset pelaajat on määritelty objektiin liitetyssä PlayerManager-skriptissä. Kukin pelaajahahmo luetaan määritellystä prefab-objektista asianmukaisesti ja sijoitetaan PlayerManagerissa määriteltyyn koordinaattiin.

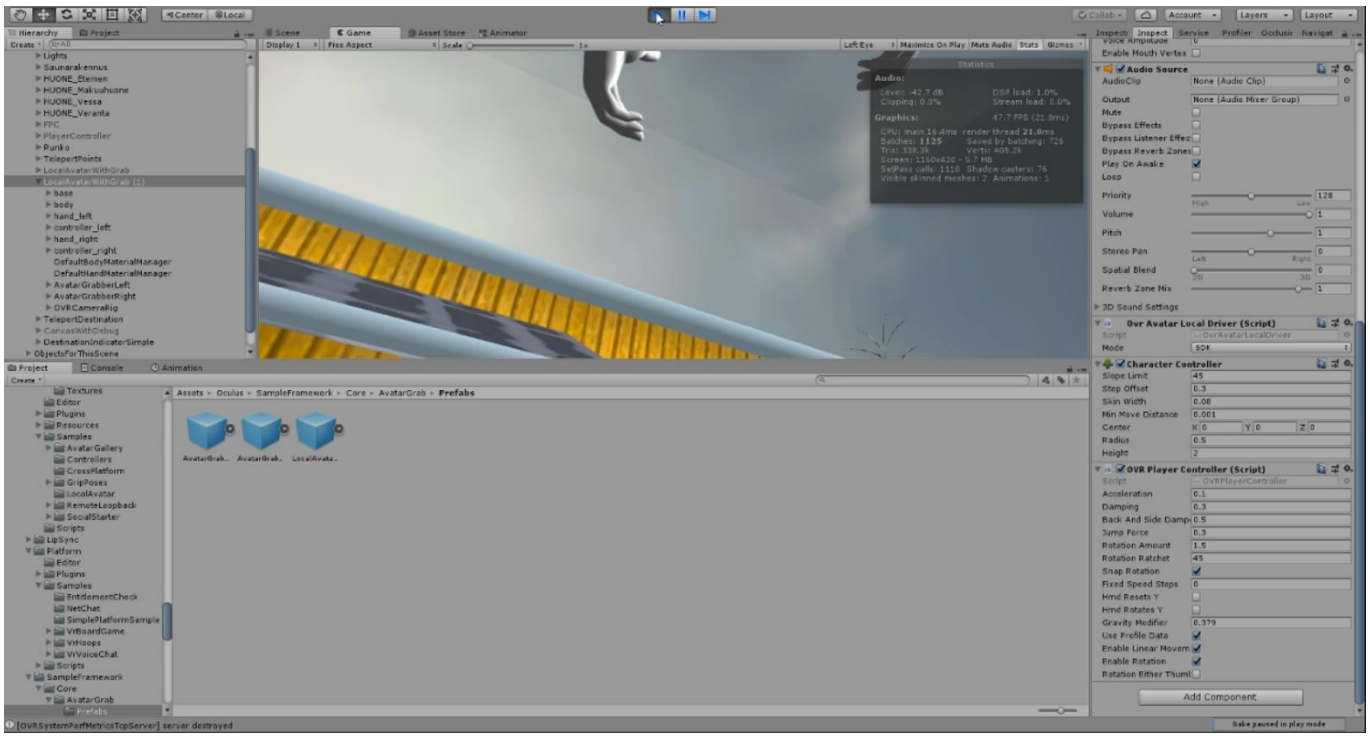
2.7 Pelaaja normaalisti

Aiemmin pelaajalle oli rakennettu liikkumiskriptit alusta alkaen. Päivittääkseen pelaajahahmon nykypäivään nämä kyseiset arvot kopioitiin Unityn perusmallien mukana tulevaan Character Controlleriin. Unityn Character Controller on 3D-fysiikka pohjainen komponentti, jota yleensä käytetään ensimmäisen tai kolmannen persoonan hahmon kontrolloinnissa. Nämä hahmot eivät käytä hyväkseen Unityn Rigidbody-fyysikoita.

Nykyisessä pelihahmossa on samallalailla liitettynä mukaan pelaajahahmon dialogi. Suurimmaksi osaksi dialogi edelleen noudattaa samaa vanhaa dialogijärjestelmää, joka oli koottu uudelleen jostain kolmannen osapuolen tarjoamasta JS-pohjasta.

2.8 Pelaaja VR:ssä

Koska koodia haluttiin käyttää uudelleen tehokkaasti, pelaajaa myös VR-moodissa liikuteltiin saman CharacterController-objektin avulla. Oman ongelmansa tästä toi, mikäli VR-mallin luovaan LocalAvatar-objectiin lisäsi kyseisin CharacterControllerin LocalAvatar menetti tietonsa, kuinka korkealla sen lattiaraja oli. Tämän takia se sai itse VR-mallinnuksen leijumaan pelaajan ylipuolella (Kuva 4). Ratkaisuna oli siirtää LocalAvatar CharacterControllerin hierarkian lapseksi ja kiinnittää se sen sisäiseen TrackingSpace-objektiin ja ottaakseen CharacterControlleriin liitetyn OVRPlayerController-luokan Y-akselin resetointi pois päältä. (Facebook Technologies, LLC, 2019)



Kuva 4: Local Avatar ei tunnista lattiaansa oikein, mikäli se on suoraan kiinnitetty CharacterControlleriin

2.9 StoryScriptManager

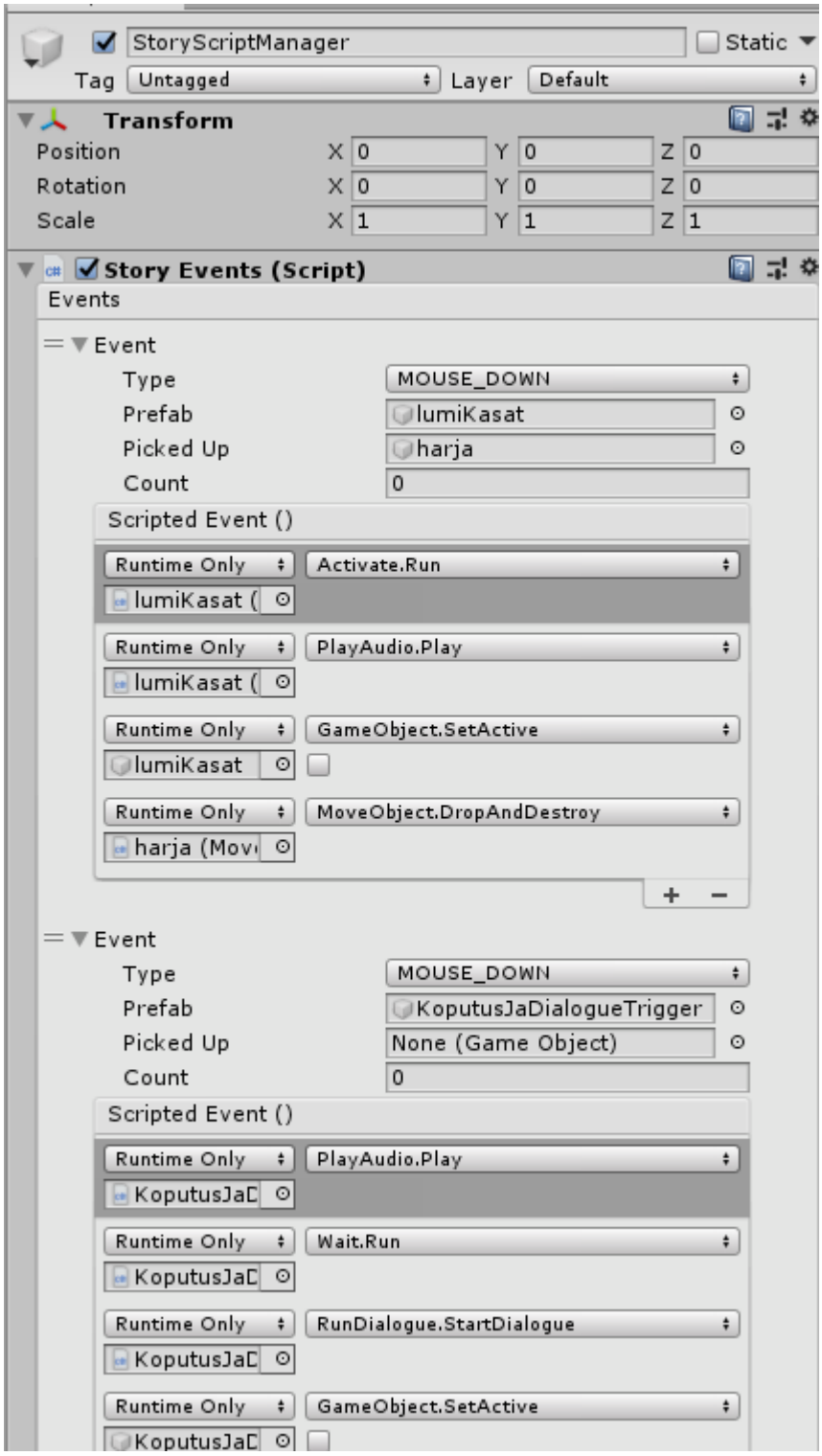
Nykyistääkseen pelirunkoa nykyaikaisemmalle pelimoottoriohjelmoinnille haluttiin jokainen G-skriptin alaisuudessa hajallaan olevat skriptit tuoda yhteen paikkaan helpommin käytettäväksi koodissa. Koska jokainen oma Action- ja muu skriptinsä oli omassa peliobjektissaan, pelirungon kokonaishahmoittaminen ja ylläpitäminen olivat erittäin hahkalaa, jopa kyseiseen peliin tottuneeseen henkilölle. StoryScriptManager oli skripti, joka liitettiin GameManager-objektin alaisuuteen. Sen vastuulla oli luoda havainnostava runkorakenne koko nykyiselle pelisessiolle ja helpottaa jopa ei-ohjelmoijan uuden tarinaskenaarioiden lisäämistä yksinkertaisella käyttöliittymällä. Käyttöliittymän ohjelmointi mahdollistettiin Unityn oman Editor-luokkien avulla, jolla inspector-ikkunaan upotettuihin skripteihin pystyy ohjelmoimaan muokatun käyttöliittymän tarjoten erilaista funktionaalisuutta.

StoryScriptManagerin avulla pystyy luomaan loogisesti toisiaan seuraavia tapahtumia ja niiden seurauksia. Jokaisessa peliobjektissa on edelleen omat skriptinsä, mutta skriptien hallinta tapahtuu StoryScriptManagerin kautta, joka on koonnut kaikki yheen selkeään järjestykseensä. Manageriin pystyy luomaan uusia StoryEvent-listauksia. Kun uuden Eventin luo, siihen liitettään peliobjekti, jonka kanssa halutaan vuorovaikuttaa. Kun peliobjekti on lisätty, valitaan vuorovaikutuksen tyyli. Tämä tyylivaihtelee niin hiirellä klikkaamiseen kuin objektiin liitetyn collider-olion kanssa yhteentörmäämiseen. Eventissä on myös mahdollista vaatia, että pelaaja pitää tiettyä peliobjektia kädessään pickedUp-parametrin avulla.

Kun peli on alustunut StoryScriptManager lisää näihin kyseisiin objekteihin listener-event-objektin, joka tiedottaa takaisin onko kyseiset asetetut parametrit täyttyneet. Kun tiedotus on tapahtunut se alkaa käydä lävitse sen scriptedEvent-listaa. Jokainen scriptedEvent on UnityEventistä periytyvä tapahtuma, jonka pystyy triggeroimaan helposti muualta käsin ja jopa upottamaan käyttöliittymä elementteihin. Tämän takia se on erittäin yhteensopiva tarkoitusperien kanssa.

Jokainen scriptedEvent vastaa teon seuraukseen ja jokaiselle eventille voidaan lisätä rajaton määrä eri seuraamuksia. Nämä seuraamukset voivat vaihdella eri objektien näkyvyydestä, poistosta ja dialogin ja äänen toistosta.

Täydennetty StoryScriptManager koostuu lisäkkäisistä listoista, jotka parantavat koko tilanteen yhteistä hahmoitusta kuten kuvassa (Kuva 5) on esitetty.



Kuva 5 : Tarinan pelieventit tuotuna Manageriin

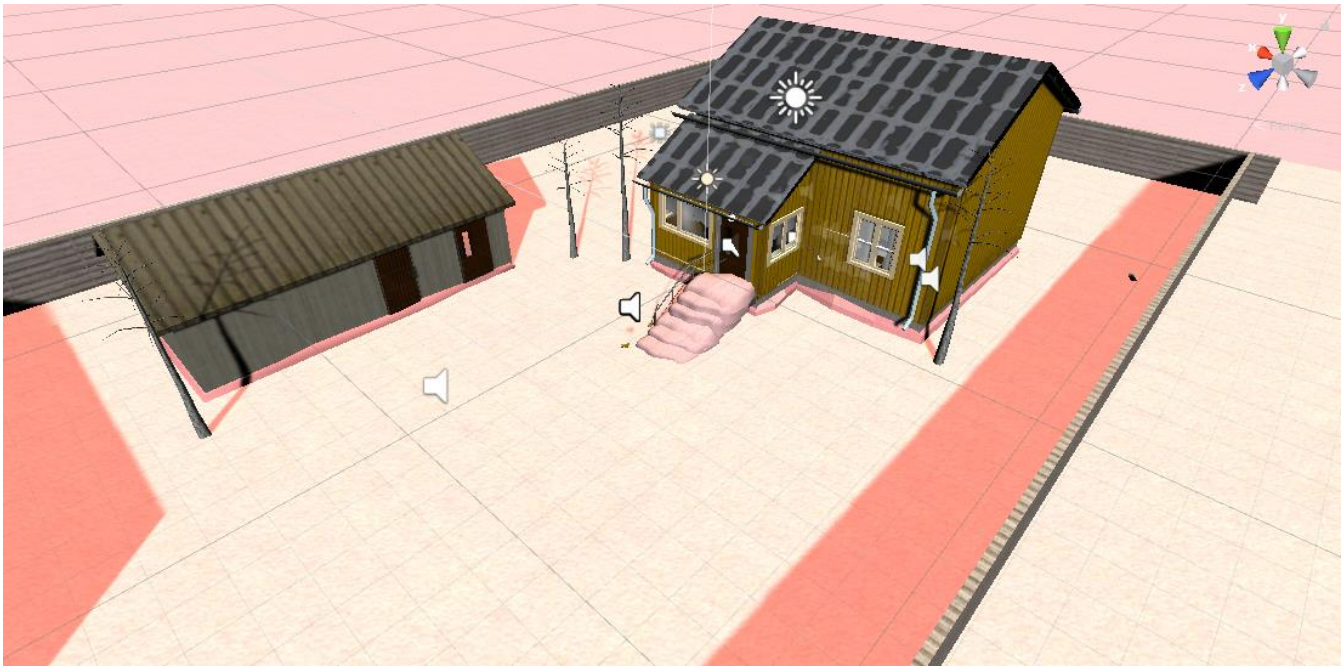
2.10 Valaistusmuutokset Unity 2019.1:ssa

Pelikehityksessä valokartat on yleensä määritelty esirenderöityihin tekstuureihin, jotka pitävät sisällään valonlähteiden vaikutukset kohtauksen staattisiin esineisiin pikselitasolla. Nämä valokartat ovat asetettu päällekkäin pelikohtauksen geometrian päälle saadakseen aikaan vaikutuksen valaistuksesta. Unityssa valaistus hallitaan pääosin Lightmapper-työkalun avulla. Se kykenee leipomaan (engl. bake) valokartat kohtauksen geometrian ja valolähteiden perusteella. Toinen valaistuksessa nykyään käytetty tekniikka on valoanturit (engl. light probes). Ne tallentavat tietoa siitä, kuinka valo kulkee tilan lävitse pelikohtauksessa. Tietyllä määrällä valoantureita voidaan parantaa liikkuvien esineiden valaistusta ja staattista Level of detail (LOD)-horisonttimaisemaa kyseisessä tilassa. (Gregory, 2014) (Unity Technologies, 2019)

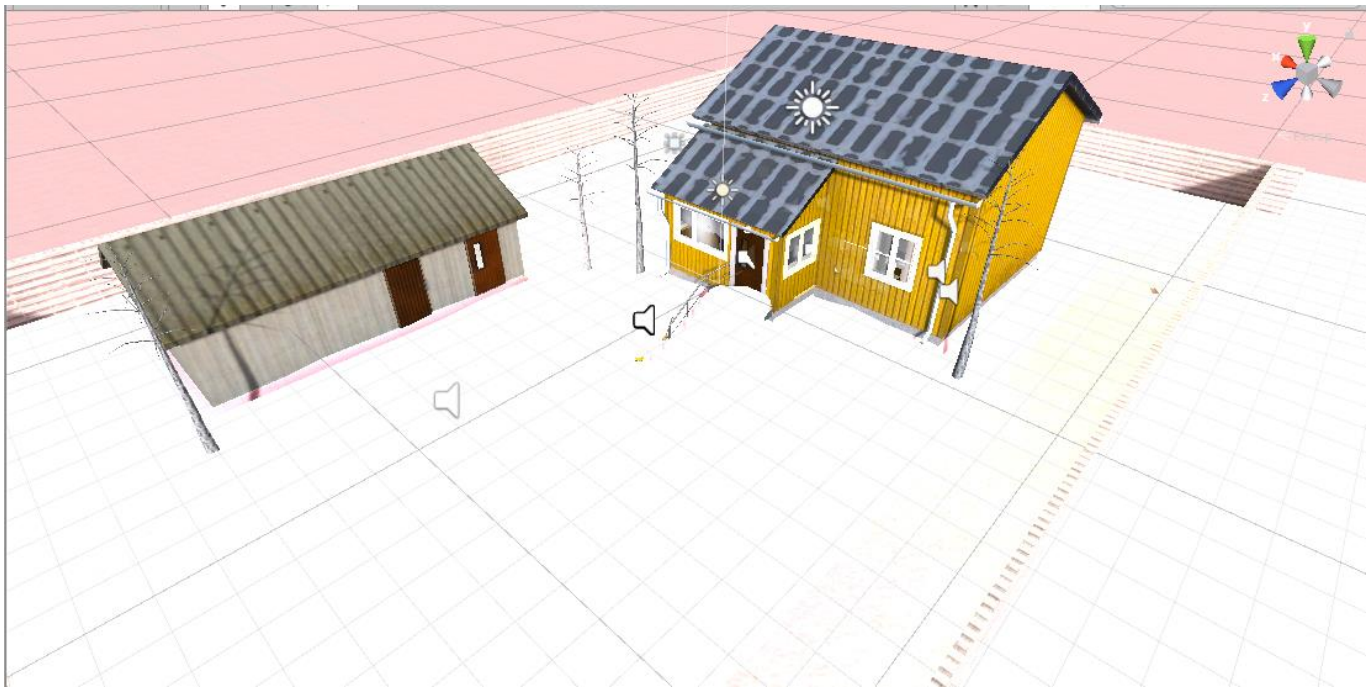
Unity 5:sta alkaen Unity on käyttänyt valaistusjärjestelmänään Geomericsin Enlightenia valokarttojen kartoittamiseen ja reaaliaikaiseen maailmanvalaistukseen. Ennen versiota 2019.1:ta epäsuora intensiteetin liukusäädin vaikutti ainoastaan valokarttoihin, kun Progressive Lightmapper-työkalua käytetään. Enlighten hallinnoi molempia sekä valokarttoja että valoantureita. Nyt kaikki back-endit vaikuttavat epäsuoraan intensiteetin arvon sekä valokarttoihin että valoantureihin. Tämän takia leipoessaan uudelleen valaistusta osa antureista saattaa näyttää kirkkaammilta, mikäli epäsuoraa intensiteettiä on muutettu sen oletusarvosta. Tämän takia muuttaessaan versioiden väliä on hyvä poistaa leivotut tiedot ennen valokarttojen uudelleen leiposta päivityksen jälkeen. (Unity Technologies, 2020)

Tämän takia kohtauksien materiaalit voivat vaikuttaa erilaisilta samoja arvoja käyttäen. Versiossa 2019.1 Unity aloitti kehittämään uusia sisäänrakennettuja jälkiprosessointieffektejä sekä High Definition Scriptable Render Pipeline (HDRP)-pohjassaan sekä Light Weight Render Pipeline (LWRP)-pohjaansa muuttaen shadereitaan vanhoista versioista. (Unity Technologies)

Graafisen ja renderointi API:n muutosten takia myös vanhemmista versioista tuotujen asset-objektien heijastusarvot eivät näytä odotetulta kuten alla olevasta kuvasta (Kuva 6) voi nähdä. Kun valokartat on leivottu uudestaan, voidaan nähdä emission intensiteetin muutokset sekä aiempien valaistun moottorin että shaderien muuttuneiden emissio-ominaisuuksien kautta alla olevassa toisessa kuvassa (Kuva 7).



Kuva 6 : Rintamiestalo1-kohtaus ennen uudelleen leivottuja valaistuksia päivityksen jälkeen Unity 2019.2.15f1:ssa



Kuva 7 : Rintamiestalo1-kohtaus uudelleen leivottuja valaistusten jälkeen Unity 2019.2.15f1:ssa

3 JATKOKEHITYS

Suosituksena projektin tulevaisuudelle olisi järkevintä rakentaa kyseinen peli uudestaan modernissa pelimoottoriympäristössä pohjautuen kootun runkokehikon kaltaiseen ratkaisuun. Vaikka nykyinen runkokehikko onkin ajanmukaisesti parempi ratkaisu kuin aiempi, aiemmat suunnitteluratkaisut ovat edelleen integroitu nykyiseen järjestelmään.

Nykyinen dialogijärjestelmä vaatisi vielä uudelleen kirjoitusta. Nykyään dialogistringit ja sekä äänet asustelevat omissa paikoissaan ja niitä haetaan vertailemalla keskenään silmukoissa etunumerointia keskenään. Jonkin asteinen nykyaikainen node-pohjainen järjestelmä toteutteutettuna Unity Editorilla toisi selkeyttä myös ei-ohjelmoijalle.

Aiempi tallennusjärjestelmää ei ollut toteutettu, sillä se oli toteutettu JS-pohjaisessa koodissa kirjoittamalla erilaisia ohjelmointikomentoja tekstitiedostoon tallennettavaksi. Ladatessaan tämä tekstiluettiin ja koodi toteutettiin reaaliaikaisesti. C#:ssa ei ole mitään tämän kaltaista. Nykyisessa runkokehityksessä jokaiselle StoryEventille kykenee antamaan id:n/indeksin. Tämän indeksin tallentaminen ja lataaminen olisi varmasti helpoin ja tilaasäästävin vaihtoehto toteuttaa tallentaminen.

Myöskään kommunikointia palvelimen kanssa ei oltu implentoitu.

4 YHTEENVETO

Projektin alkuperäinen tavoite oli vuonna 2012 Savoniassa toteutetun Virtuaali Vanhus (VIVA) pelin, https://viva.savonia.fi/, modernisointi. Peli on toteutettu Unityllä webselaimessa pelattavaksi, mutta nykyään selaimet estävät pluginien suorittamisen selaimessa ja peliä ei voi pelata. Työhön kuuluu pelin kääntäminen nykyisissä selaimissa toimivaksi sekä kaikilla kuluttaja VR-laseissa (Vive, Oculus, Microsoft Hololens, Microsoft MR-lasit) toimivan version tekeminen. Lisäksi työhön kuuluu viva.savonia.fi -sivuston uusiminen siten, että peli voi kommunikoida suoraan taustajärjestelmän kanssa (pelin loki).

Pelin skriptien kääntäminen oli aikaa vievää manuaalista työtä. Erot versioiden välillä vaihtelivat erinäisistä nimeämuutoksista kokonaisten ominaisuuksien poistoon tai siirtymättömyyteen nykyisenään.

Toteutuksen aikana tavoite muuttui. Modernisoitujen ja alkuperäisten kooditiedostojen ero osoittui uusien Unity versioiden ominaisuuksien rinnalla liian suureksi stabiilin alkuperäisen pelin emuloinnin onnistumiseksi. Projektin tavoite siirtyi luomaan nykyaikainen runkokehikko tukemaan tulevien ominaisuuksien ja tasojen luontia.

Lopputuloksena syntyi huomattavasti suunniteltua suppeampi käännöstyö, mutta uusi runkokehikko tukee lupaavasti uusien ominaisuuksien lisäämistä. Tämä pitää sisällään visuaalisemman pelitapahtumien lisäilyn ei-ohjelmoijille sekä modernimman näkökulman pelimoottorin hallintaan.

5 LÄHDELUETTELO

Facebook Technologies, LLC. 2019. Attaching Local Avatar to Player Controller in Unity. *Oculus Developer Forums.* [Online] Facebook Technologies, LLC, 2019. [Viitattu: 13. 3 2019.] https://forums.oculusvr.com/developer/discussion/49444/attaching-local-avatar-to-player-controller-in-unity.

—. 2019. OVRInput. *Oculus Developer Center.* [Online] Facebook Technologies, LLC, 2019. [Viitattu: 14. 3 2019.] https://developer.oculus.com/documentation/unity/latest/concepts/unity-ovrinput/.

Google LLC. 2014. Update on NPAPI Deprecation. *Chromium Blog.* [Online] Google LLC, 2014. [Viitattu: 14. 3 2019.] https://blog.chromium.org/2014/05/update-on-npapi-deprecation.html.

—. 2019. Use or fix Flash audio & video. *Google Chrome Help.* [Online] Google LLC, 2019. [Viitattu: 14. 4 2019.] https://support.google.com/chrome/answer/6258784?hl=en&visit_id=636908551800903202-1408687936&rd=1.

Gregory, Jason. 2014. *Game engine architecture -- Second edition.* s.l. : CRC Press, Taylor & Francis Group, LLC, 2014. ISBN 978-1-4665-6001-7.

HONT. 2019. [转]Unity: make your lists functional with ReorderableList. *HONT blog.* [Online] 2019. [Viitattu: 18. 3 2019.] https://www.cnblogs.com/hont/p/5458021.html.

Mozilla Foundation. 2015. NPAPI Plugins in Firefox. *The Mozilla Blog.* [Online] Mozilla Foundation, 8. 10 2015. [Viitattu: 14. 4 2019.] https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/.

Unity Technologies. 2020. *Unity - Manual: Upgrading to Unity 2019.1.* [Online] 29. 04 2020. [Viitattu: 03. 05 2020.] https://docs.unity3d.com/Manual/UpgradeGuide20191.html.

—. Getting started with LWRP | Package Manager UI website. [Online] Unity Technologies. [Viitattu: 15. 5 2020.] https://docs.unity3d.com/Packages/com.unity.render-pipelines.lightweight@5.10/manual/getting-started-with-lwrp.html.

—. 2019. Unity - Manual: Glossary. *Unity.* [Online] Unity Technologies, 2019. [Viitattu: 14. 4 2019.] https://docs.unity3d.com/Manual/Glossary.html#WebGL.

—. 2019. Unity Webplayer – NOT SUPPORTED. *Unity.* [Online] Unity Technologies, 2019. [Viitattu: 15. 4 2019.] https://unity3d.com/webplayer.

Wikimedia Foundation, Inc. 2019. NPAPI. *Wikipedia.* [Online] Wikimedia Foundation, Inc., 2019. [Viitattu: 14. 4 2019.] https://en.wikipedia.org/wiki/NPAPI.