Miika Montonen

VAIHDETTAVIEN HAHMOJEN LUONTI MONINPELIIN

Opinnäytetyö
Tietojenkäsittely

2017

Kaakkois-Suomen
ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Miika Montonen	Tietojenkäsittely	Marraskuu 2017
Opinnäytetyön nimi Vaihdettavien hahmojen luonti multipeliin		33 sivua
Toimeksiantaja Työväen Sivistysliitto TSL / Mikkelin ammattikorkeakoulu		
Ohjaaja Jukka Selin		

Tiivistelmä

Pelillistäminen on lisääntynyt huomattavasti viime vuosina ja sitä on alettua kokeilemaan moniin eri aihealueisiin. Tätä on edusauttanut varsinkin Unity-pelimoottorin tuleminen ilmaiseksi kokonaisuudessaan vuonna 2015.

Tässä opinnäytetyössä esittelen, kuinka käyttää Unity-pelimoottoria ja toteuttaa sen Unet komponentilla pelihahmon ulkonäön vaihto peliympäristössä ajonaikana. Aloitan kertomalla, mikä on moninpeli, kuvailemalla on Unityn ja sen toimintoja liittyen ominaisuuksiin, joita tarvittiin käytännöntyössä. Tämä sama myös Unet-komponentin osa-alueessa.

Opinnäytetyöni viimeisessä osa-alueessa käyn läpi toimeksiantoni käytännön työtä ja sen tuloksia. Toimeksiantajana toimivat Työväen Sivistysliitton ja toimeksiantoni oli osa heidän hanketta nimeltä digiosaajaksi työelämään. Mikkelin ammattikorkeakoulu toteutti tähän hankkeeseen moninpelattavan pelin, joille tein käytännön työni.

Asiasanat

Unity, Unet, peliohjelmointi, pelillistäminen, moninpeli

Author (authors)	Degree	Time
Miika Montonen	Bachelor of Business Administration	November 2017
Thesis Title Creating changeable characters for multiplayer		33 pages
Commissioned by Työväen Sivistysliitto TSL / Mikkeli University of Applied Sciences		
Supervisor Jukka Selin		

Abstract

Gamification has increased considerably over the last few years and its uses have been experimented on with many different themes. Unity game engine becoming wholly free in 2015 has especially contributed to this.

In this thesis, I introduce how to use the Unity game engine and its component Unet to implement a change of appearance for the game character in the game environment, while in runtime. I start by telling what is a multiplayer game, and by describing Unity and its functions related to the features needed in the theory section. This was the same with the Unet component section.

In the final part of the thesis I go through the practical work I i did regarding the theme the theme of this thesis. The work was commissioned by Työväen Sivistysliitto TSL and it was part of their project called digiosaajaksi työelämään. Mikkeli University of Applied Sciences implemented a multiplayer game for this project to whom I worked for during the practical work.

Keywords

Unity, Unet, game programming, gamification, multiplayer

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä perehdyn pitkäkestoisen pelikehityksen alkuvaiheisiin suunnittelun, toteutuksen ja jatkokehityksen kannalta. Toimeksiantajana toimii Digiosaajaksi työelämään-hanke. Hanketta oli toteuttamassa Mikkelin ammattikorkeakoulu, Itä-Suomen Yliopiston ja Työväen sivistysliitto. Tarkoituksena oli suunnitella ja toteuttaa prototyyppi kehitettävän työelämäpelin palvelutilanteita kuvaavasta osa-alueesta. Pelin tarkoituksena on toimia osana kurssien sisältöä.

Hankkeen tavoite on parantaa aikuisten, joilla on heikot perustaidot, tieto- ja viestintäteknisiä taitoja (TVT-taidot) ja sitä kautta parantaa Etelä-Savon työllisyystilannetta. Hankkeessa kehitetään työelämälähtöisiä pedagogisia menetelmiä tieto- ja viestintäteknisten taitojen opetukseen aikuisväestölle, jolla on heikko peruskoulutus.

Kohderyhmä on yli 30-vuotiaat työttömät tai työttömyysuhan alaiset, joilla ei ole perusasteen jälkeistä tutkintoa tai joiden tutkinto ei vastaa nykyisen työelämän vaatimuksia. Näillä vähän koulutetuilla ryhmillä on usein heikot tai olemattomat muuttuvan työelämän vaatimat TVT-taidot. Kohderyhmään kuuluvat myös yksinyrittäjät. Maantieteellinen toiminta-alue on Etelä-Savossa Mikkelin, Savonlinnan ja Pieksämäen seutukunnat.

Hankkeessa toteutetaan kussakin toimintakaupungissa noin puolen vuoden mittaisia monimuotokoulutuksia, joihin rekrytoidaan osallistujiksi yllä kuvattua kohderyhmää. Koulutus alkaa TVT-testillä ja oppimisvalmiuksien kartoituksella. Alkukartoitusten pohjalta oppijat jaetaan tasoryhmiin, joissa otetaan oppijoiden yksilölliset tarpeet huomioon.

Koulutuksissa käytetään osallistavia menetelmiä, joiden tavoitteena on osallistujien voimaannuttaminen ja rohkaiseminen uskomaan itseensä oppijana ja työntekijöinä. Erilaisen oppijan tarpeet otetaan pelin kehittämisessä ja ohjauksessa huomioon. Pelin visuaalisuus ja tarinankerronta tukevat erilaisia oppijoita. Opiskelu tapahtuu yhteistoiminnallisesti ryhmissä, joille annetaan yhdessä ratkaistavia tehtäviä. Ryhmissä opitaan työelämässä vaadittavia vuorovaikutustaitoja ja tiimityötä.

Mikkelin ammattikorkeakoulu toteuttaa hankkeessa moninpelattavan työelämäpelin, joka tarjoaa vaihtoehtoisen tavan oppia työelämän TVT-taitoja, työelämän pelisääntöjä, alaistaitoja ja vuorovaikutusta työelämässä. Työelämäpeli tuo hankkeeseen merkittävän uutuusarvon ja uudenlaisen lähestymisen kohderyhmän tarpeisiin vastaamisessa. Peliä kehitetään jatkuvana käyttäjälähtöisenä prosessina ja kohderyhmät osallistuvat vahvasti sen kehittämiseen. Tämä osaltaan lisää kohderyhmän oppimismotivaatiota ja itseluottamusta.

Varsinaisen pelin lisäksi tuloksena syntyy ns. jatkokehitysalusta, työkalu, joka huomioi erilaiset oppijat ja tulevaisuuden kehitystarpeet. Se tuottaa osaltaan myös sisältöjä peliin. Hanke julkaisee raportin yhteyteen kehitysblogin, johon kaikki hanketoimijat voivat osallistua.

Itä-Suomen yliopisto Aducate, toteuttaa hankkeessa jatkuvaa TVT-opetuksen ja pelin kehittämisen arviointia ja antaa opettajille ja pelikehittäjille jatkuvaa palautetta sähköisten kyselyjen ja mittausten avulla prosessin eri vaiheissa. Aducate osallistuu osallistujien taitojen alkukartoitukseen. Se arvioi opettajien digi- ja verkkopedagogisten taidot ja jatkokoulutustarpeet. Arviointien tuloksena koulutusprosessi ohjautuu jo hankkeen aikana kohderyhmää parhaiten palvelevaksi ja siten koulutus jatkossa vastaa kohderyhmään kuuluvien koulutustarpeeseen hyvin. Arviointi paljastaa sellaisia koulutustarpeita, joihin vastaamalla voidaan jatkossa yksinkertaisemmin tukea aikuisten digiosaamista.

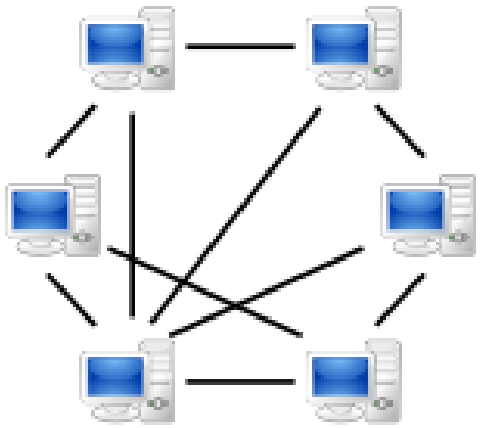
Keskityn opinnäytetyössäni selittämään tekemäni osan moninpeliin, joka antoi pelaajalle mahdollisuuden vaihtaa pelihahmonsa ulkonäköä ja sen mahdollisen hahmon vaatteiden väriä.

Pohjustan myös moninpelien verkkotaustaa, koska se oli oleellinen osa hahmon ulkonäköön vaikuttavan järjestelmän luomisessa, sekä yleinen selitys käytetystä pelimoottorista Unitystä. Tätä seuraten kerron miten kyseisessä pelimoottorissa asiat toimivat ja rakentuvat, miten ohjelmointi ja objektien luonti tapahtuvat peliympäristöön.

2 MONINPELI

Moninpeli on peli, jossa kaksi tai useampi pelaaja voivat osallistua samaan peliin samanaikaisesti. Pelaamisen alkutaipaleilla tämä tapahtui saman laitteen kautta, mutta on nykyaikana siirtynyt melkein täysin toimimaan verkon yli.

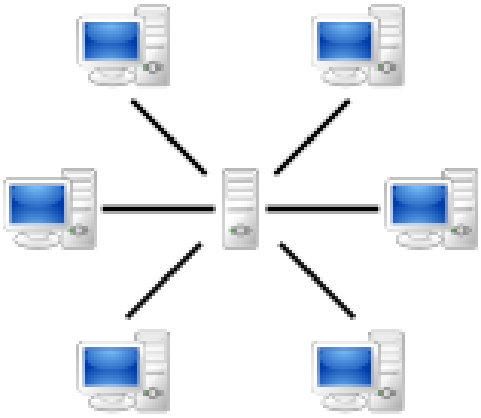
Moninpelit jakaantuvat yleisemmin kahteen kategoriaan, joista ensimmäinen on pelaaminen toisiaan vastaan tavalla tai toisella ja toinen on, että pelaajat pyrkivät yhteistyöllä saavuttamaan annetun lopputuloksen.



Kuva 1. Vertaisverkko

Vertaisverkko-malli oli ensimmäisten moninpelien toimintapohjana. Tässä toimivat kaikki pelaajat serverinä, sekä klienttinä jakaen tietoa kaikilla osanottajille. Tämä kuitenkin aiheuttaa helposti monia ongelmia.

Pelin ollessa käynnissä tilanteen pitäminen samanlaisena kaikille pelaajille on erittäin hankalaa vertaisverkkoa hyväksikäyttäen ja pienikin heitto pelaajien välisessä yhteydessä saa pelimaailman tapahtumat hiukan erilaisiksi ja näiden kasaantuessa pelaamisesta tulee suorastaan mahdotonta. Pelaajien arvoa ei verkossa määritellä erikseen, joten tästä johtuen ei ole mahdollista löytää peli-tilannetta, johon kaikkia pelaajia voisi verrata ja tasata tilanteen.



Kuva 2. Palvelin-asiakas-verkko

Näistä ongelmista opittuna alettiin siirtyä käyttämään palvelin-asiakas–yhteyttä, jossa kaikki pelaajat ovat klienttejä ja keskustelevat vain serverin kanssa. Pelin toiminnot toteutettiin vain serverillä, joka otti vastaan pelaajien tekemiä inputeja, eli mahdollisia komentoja näppäimistöltä tai muulta tämänkaltaiselta laitteelta. Näiden inputtien perusteella serveri pystyi luomaan uuden pelitilanteen sekä lähdettämään sen kaikille pelaajille ja luomaan identtisen pelitapahtuman kaikille pelaajille.

Koska yhä useammin siirryttiin verkon yli toimintaan, niin tämä toi mukanaan sen, että järjestelmä toimi hyvin, kunhan verkkoyhteyden aiheuttama latenssi pysyi pienenä pelaajien ja serverin välillä. Saadakseen latenssin pysymään pienenä alettiin osa pelin tapahtumista, eli koodista suorittamaan paikallisesti.

Tähän tuotiin vielä mukaan järjestelmä, joka ennusti klientillä pelaajan käyttäytymistä ja lähetti sen perusteella serverille tietoja. Tämä pienensi latenssin vaikutusta, koska klientin ei tarvinnut enää odottaa inputtien lähettämistä serverille ja serverin vastausta. Koettiin, että ennustamisen lisäksi tarvittaisiin tapa korjata tilanteet, joissa klientti olisi väärässä siitä mitä pelaaja tulisi tekemään. Klientti kirjoittaa inputit ja niiden tapahtuma-ajan muistin puskuriin ja käyttää näitä ennustaessaan tilannetta.

Päätilana ennustukselle klientti käyttää viimeisintä serveriltä tullutta tietoa. Jos tapahtuu ristiriitatilanne, klientti käyttää serveriltä tullutta uutta tietoa ja poistaa

muistin puskurista olevat tiedot, jolloin klientti katsoo pelin tilanteet serverin lokista taaksepäin viimeisimpään suoritettuun pelitilanteeseen. Näin saadaan vältettyä tilanteet, joissa toinen pelaa pompahtelee paikasta toiseen äkillisten muutoksien aiheutuessa ja väärästä ennustuksesta serverin oikeaan pelitilanteeseen. (Glenn. 2010).

3 UNITY

Unity on moni-alustainen pelimoottori, joka tukee 2D- ja 3D-grafiikkaa, vedä ja pudota toiminnallisuutta, sekä C#-ohjelmointia. Unity on yksi tämän hetken suosituimmista pelinkehitysalustoista ja pelimoottoreista. Syynä tähän on sen laaja ilmainen perusversio, jonka kuka tahansa voi ottaa käyttöön täysin maksutta. Yhtenäisyys on merkittävä kyky kohdistaa pelejä useille alustoille.

Unityn on Unity Technologiesin kehittämä ja sitä käytetään lähinnä videopelien ja simulaatioiden kehittämiseen tietokoneille, konsoleille ja mobiililaitteille. Se ilmoitettiin ensimmäiseksi vain OS X: lle, joka julkaistiin vuonna 2005 Applen Worldwide Developers Conference-tapahtumassa, ja sitä on sen jälkeen laajennettu 27 alustalle.

Unity 5:n julkaisu toi mukanaan Personal Editionin, joka sisältää lähes kaiken sen, minkä entinen maksullinen Pro-versiokin. Unityn ilmaisversiolla on myös mahdollista julkaista projektejaan ilman lisenssimaksuja, kunhan tilin ja projektin omistaja ei tienaa yli 85 000 € vuodessa.

Unityn ilmaisversio takaa sen, että uudet kehittäjät uskaltavat kokeilla alustaa ja sen myötä päätyvät sitä monesti käyttämään. Unityllä on myös oma kauppapaikka, josta kehittäjät voivat hankkia lisäosia Unityyn tarpeen mukaan.

Unity käyttää ohjelmointikielinään UnityScriptiä, C#-kieltä ja myös muita, mutta ne ovat käytössä harvinaisia. Näistä suosituin on C#, jonka suurin osa Unityn käyttäjäkunnasta on ottanut ohjelmointikielekseen. UnityScript joka muistuttaa JavaScriptiä, on toiseksi suosituin käytetyistä ohjelmointikielistä.

Unityn oma koodi referenssi viittaa ainoastaan näihin kahteen suositumpaan, mikä aiheuttaa sen, että muiden harvinaisten kielten käyttö on erittäin vähäistä.

Unityn koodi referenssin lisäksi Unity tarjoaa referenssin kanssa yhdessä toimivan manuaalin sekä Learn-osion sivustollaan, mistä löytyy huomattava määrä ohjeita erilaisten pienten malliprojektien toteuttamiseen.

Näiden lisäksi Unityn suuren käyttäjämäärän takia, internet on täynnä erilaisia ohjeita projekteihin ja omissa projekteissa esiintyviin ongelmiin. Unityn käyttäjien foorumi tarjoaa myös paljon apua kehittäjille.

Unityn suosio perustuu myös sen laajaan alustatukeen. Kuten aikaisemmin mainittu, tällä hetkellä Unity tukee yli 27 eri alustaa aina pöytätietokoneesta älytelevisioon. Helppokäyttöisyys ja hyvä tukijärjestelmä ovatkin Unityn parhaita myyntivaltteja.

3.1 Gameobject prefab Unityssä

GameObject eli peliobjekti on Unityssä kaikki sceneen (pelitapahtuma) ladatut pelin osat. Jokainen pelihahmo, valo, tausta tai käyttöliittymän nappula, ovat peliobjekteja.

Prefab on siihen liitetyn peliobjektin malli kyseiselle objektille. Peliobjektisteista voidaan luoda prefab kääntämälle haluttu peliobjekti prefab muotoon Unityn sisällä, jolloin kaikki peliobjektin alkuperäiset tiedot tallentuvat prefab-mallitiedostoon sisältäen myös siihen liitetyt komponentit ja niiden arvot. Tämän jälkeen prefab-mallia voi muokata vaikuttamatta alkuperäiseen peliobjektiin ja näiden mallien käyttö on helppo tapa esimerkiksi luoda samanlaista objektia useampi vaikuttamatta alkuperäiseen peliobjektiin.

Prefab-tiedostojen käyttö on yleistä myös sen takia ettei, jokaista peliobjektia tai sen osaa olisi ladattava omana objektinaan erikseen, eikä selkeänä pakettina. Tällöin prefabbien käyttö, myös keventää pelien latausaikoja.

Prefabeja voi hyödyntää hyvin monissa erilaisissa tilanteissa, eikä niiden käyttö rajoitu pelkästään pelimaailmassa sijaitseviin objekteihin. Prefabeja on mahdollista siirtää myös projektien välillä, mikä mahdollistaa myös niiden lataamisen Unityn kauppapaikasta. Myös Unity itse tarjoaa muutamia valmiita prefab-paketteja kehittäjien käyttöön.

3.2 Script-koodi Unityssä

Unityn tukemalla ohjelmointikielellä kirjoitetut Script-tiedostot ovat komentosarjoja, joilla tuodaan peliobjekteihin toiminta. Ne liitetään yleensä osaksi peliobjektia, josta sen komponentit ja objekti itse saavat käskynsä. Lähes kaikki mitä pelin sisällä tapahtuu, on lähtöisin scripteistä.

Unity tarjoaa käyttäjilleen valmiin kirjaston Script-tiedostoja varten, joka sisältää useita hyödyllisiä funktioita. Yksi tärkeimmistä Unityn tarjoamista luokista on MonoBehaviour. Tämä sisältää muun muassa Start()- ja Update()-funktiot. On mahdollista määrittää toimintoja peliobjekteille, kun ne ladataan sceneen ensimmäistä kertaa (Start()-funktio) tai jokaisen ruudunpäivityksen yhteydessä (Update()-funktio).

4 UNET

Unity ilmoitti vuonna 2014 julkaisevansa uuden komponent nimeltä Unet. Unet on moninpelirajapinta, joka on tarkoitettu moninpelien tueksi Unity pelimoottoriin. Unet tuo yksinkertaisuutta yhteen vaikeimmista asioista toteuttaa Unityllä: moninpeli.

Sen tarkoitus on tuoda itsensä lisäksi myös automaattisia rakennuspalikoita mukanaan, eli komponentteja Unityyn, joiden avulla moninpelin luominen helpottuu ja tulee mahdolliseksi, jopa ilman aikaisempaa kokemusta verkkopeleistä tai ohjelmoinnista.

Unetin uusi verkkorajapinta hoitaa yleiset yhteysongelmat, jotka muodostuvat palomuureista. Unet tarjoaa tekijälle yksinkertaisen keinon luoda helposti servereitä peliin eri mahdollisuuksilla esim. Unityn matchmaker-serverit tai P2P (Peer to Peer) tyylisen, jossa omalle PC:lle voi avata julkisen serverin, eli palvelin-asiakas-verkko.

Unetin tarjoamat verkko-ominaisuudet ovat integroitu Unityn pelimoottoriin, siten myös sen editoriin luoden näiden muokkaamisen helpoksi, joko Unityn graa-

fisella liittymällä tai script-koodin puolella. Tästä esimerkkeinä kaksi Unityn ensimmäiseksi mainitsemaa asiaa virallisessa dokumentaatiossansa. Verkkoidentiteetti (NetworkIdentity), jolla voidaan viestiä kuinka peliobjektien tulisi toimia verkon yli halutuille käyttäjille, sekä verkkokäyttäytyminen (NetworkBehaviour), joka voidaan periyttää scripteille antaen niille näkyvyyden verkkossa oleville verkkoindentiteeteille. (Unity Technologies 2015).

4.1 UNET verkkopeli unityssä

Suurimmaksi osaksi Unityllä luodut verkkopelit toimivat yhden serverin ja usean klientin periaatteella. Kun käytössä ei ole omaa erillistä serveriä, yksi klienteistä toimii serverin ylläpitäjänä. Tällöin yksi pelaajista toimii, sekä palvelun tarjoajana, että klienttinä. Tämä pelaaja yhdistää itsensä luomaansa serveriin omalla paikallisella klienttillä, kun taas muut serverille yhdistyvät pelaajat toimivat etäklienttien kautta.

Unetin pyrkimyksenä on käyttää yhtä ja samaa koodia sekä paikallisesti, että etäklienttien kautta, jotta kehittäjien tarvitsisi miettiä vain yhdenkaltaista klienttiä jokaiseen tilanteeseen ja olosuhteeseen.

Peliobjektien luonti Unityssä pelinäkymään tapahtuu komennolla Instantiate, jos objekti halutaan näkyviin myös verkkopuolelle, se täytyy luoda verkossa ja antaa sille identiteetti (NetworkIdentity). Tämän voi tehdä vain serverillä, jonka seurauksena objekti luodaan peliin yhdistyneiden klienttien näkymään myös.

Normaalisti luotujen objektit ja verkossa luodut objektit ovat samanlaisia ja ne tottelevat samoja peliloopien (gameloop) käytäntöjä. Vaikka objektit ovat samankaltaisia, niillä on silti verkkopuolella eri toimintoja ja ominaisuuksia. Nämä toiminnot vaativat juuri verkkoidentiteetin toimiakseen.

Peliobjekteille välitetään scriptien komennot verkon välityksellä. Pelaaja voi kutsua scriptissä annettuja komentoja vain hänellä sallitulle peliobjektille.

Kun peliin liittytään verkon kautta, Unity ja Unet verkkorajapinta toimivat yhdessä ja objektistaan luodaan paikallinen objekti kyseiselle klientille. Tässä tapauksessa käytetään komentoa isLocalPlayer, jonka avulla kyetään tarpeen vaatiessa tutkia esim. kuka hallitsee kyseistä peliobjektia.

Muitakin boolean tarkastuksia tulee Unetin mukana, kuten isServer, isClient sekä hasAuthority. Unityn versio 5.2 mahdollisti klient-auktoriteetin antamisen pelaajasta riippumattomille objekteille.

4.2 Unet hallinta ja animaatio

NetworkManager on Unetin komponentti, joka hallitsee verkkomoninpelin eri tiloja. Se on yksi osa Unityyn tulevista Unet komponenteista.

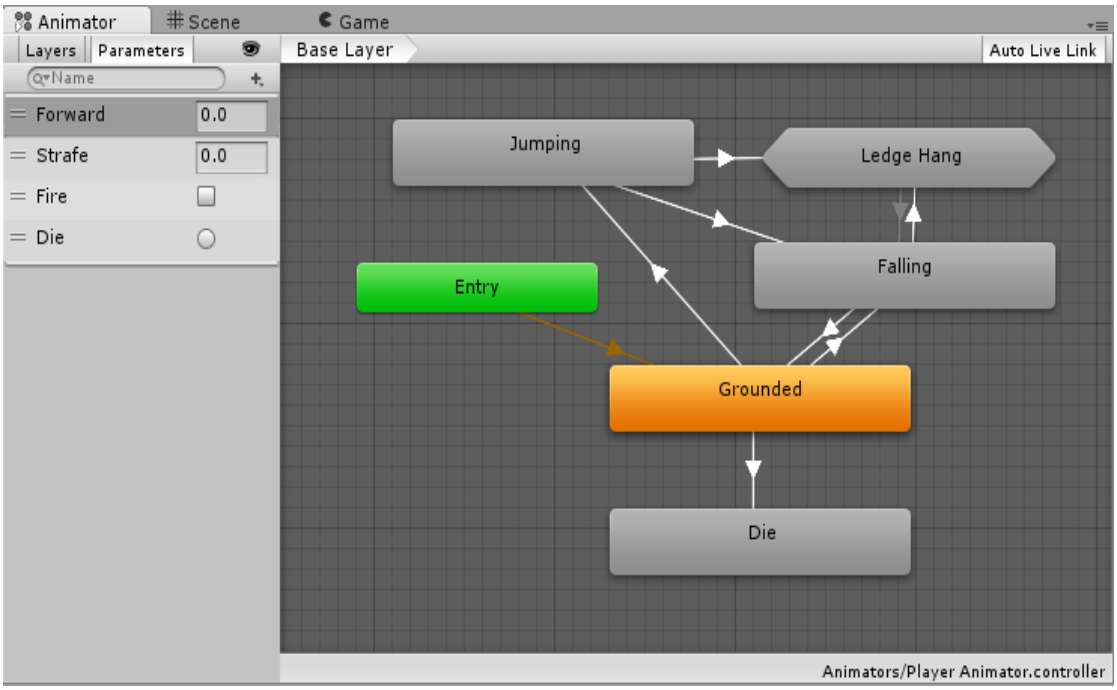
NetworkManager tuo mukanaan myös pelinäkymän puolelle NetworkManager-HUD:n. Se on käyttöliittymä, jonka kautta pelin ollessa ajossa voi vaihtaa pelin ylimmäntason verkkoasetuksia. Graafisen käyttöliittymänsä vuoksi NetworkManagerin käyttö ei vaadi välttämättä lainkaan ohjelmointia tai scriptejä.

Unityn animaatio-ominaisuuksiin kuuluu uudelleenkohdennettavia animaatioita, täydellistä animaatiopainojen hallintaa ajon aikana, tapahtumakutsut animaation toistossa, kehittyneitä tilakonehierarkioita ja siirtymiä.

Kun animaatiossa on klippejä valmiina käytettäväksi, on käytettävä Animator-ohjainta yhdistämään ne. Yksikössä luodaan Animator Controller-arvoja, ja näin voidaan helposti ylläpitää joukkoa animointeja hahmolle tai objektille.

Kuva 3. Animator komponentti

On useasti normaalia, että käytössä on useita animaatioita ja niiden välillä vaihdellaan, kun tietyt tilanteet pelissä ilmenevät. Voit esimerkiksi vaihtaa kävelystä hyppyyn helposti. Kuitenkin vaikka sinulla olisi vain yksi animaation leike, sinun on sijoitettava se animatorin ohjaimeen, jotta sitä voi käyttää pelissä.



Kuva 4. Animator kontrolleri kaavio-ikkuna

Ohjain hallitsee animaatiotiloja ja niiden välisiä siirtymiä käyttämällä niin sanottua State Machine-laitetta, jota voidaan miettiä virtauskaavioksi, jossa luodaan siirtymät toisiin animaatioihin.

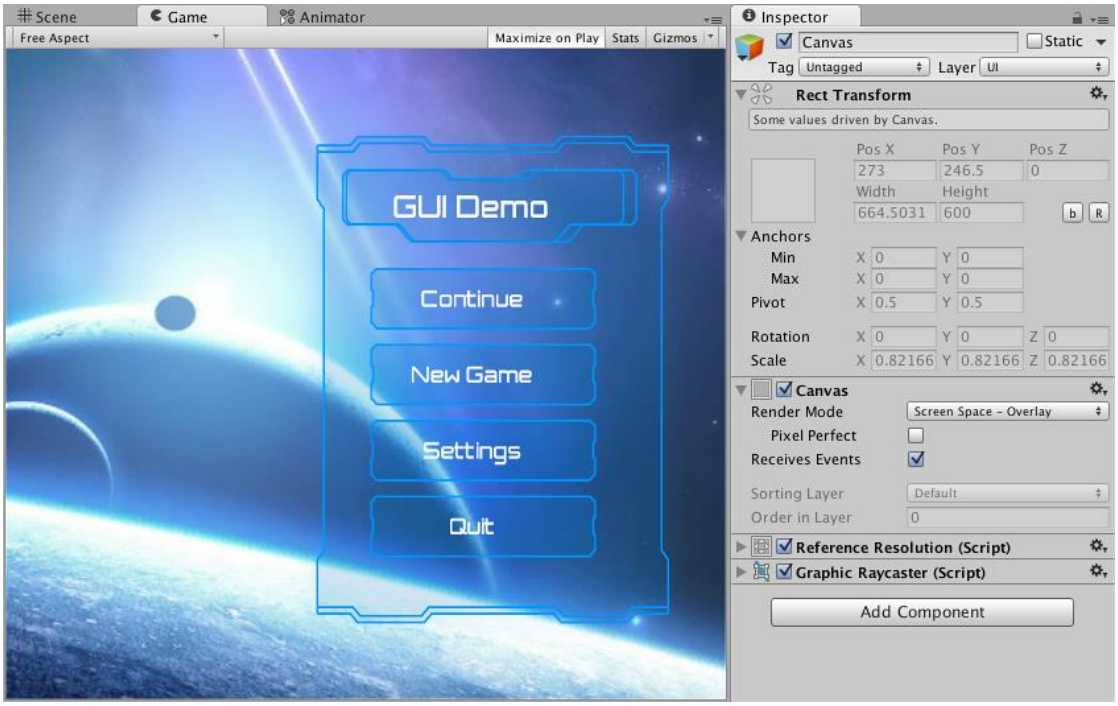
Tämä myös ajaa siihen, että Unity animaatiossa ei ole automaattista tarkastusta onko animaatio loppunut tai alkanut, vaan se on tutkittava tilakoneen tapahtumien avulla. Näin voidaan saada helposti aikaan erittäin laaja ja monimutkainen tilakone varsinkin, kun haetaan sulavia siirtymiä liikkeistä toisiin. Tilakoneeseen tulee useita hyppyjä kohdasta toiseen tutkien mitä pitäisi tehdä tiettyjen tilanteiden aikana, joka tekee siitä vaikeasti hallittavan.

4.3 Canvas

Canvas on alue, joka pitää sisällään kaikkia UI-elementtejä. Canvas-komponentti on peliobjekti ja kaikki UI-elementit ovat tällaisen Canvasin lapsia. Canvas-alue ilmenee muokkausnäkymässä suorakulmioksi. Tämä helpottaa UI-elementtien sijoittamista ilman, että pelinäkymä on näkyvillä kaiken aikaa.

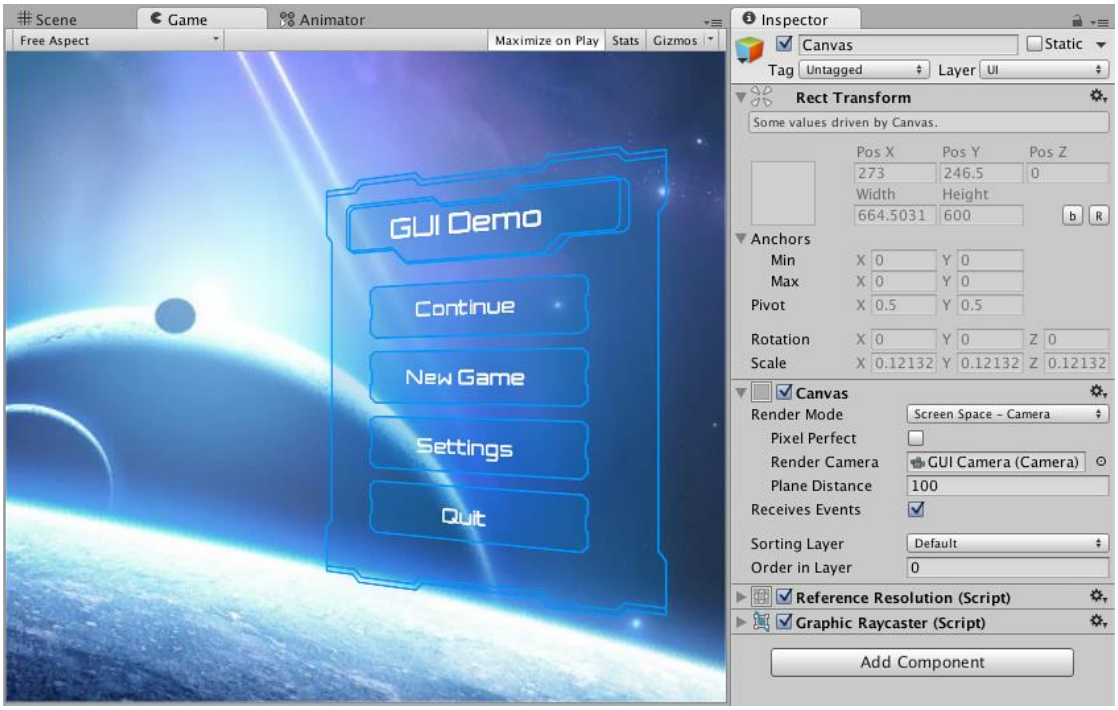
Canvasin UI-elementit piirretään samaan järjestykseen kuin ne näkyvät hierarkiassa. Ensimmäinen lapsi piirretään ensin, toinen lapsi seuraavaksi ja niin

edelleen. Järjestystä voidaan myös ohjata scripteillä käyttämällä näitä komentoja: SetAsFirstSibling, SetAsLastSibling ja SetSib-lingIndex. Canvaksella on Render Mode-asetus, jota voidaan käyttää tekemään se screen space tai world space. Nämä meinaavat, joko ruudulle suoraan piirtyvä UI tai maailmassa erillisenä oleva esim. teksti.



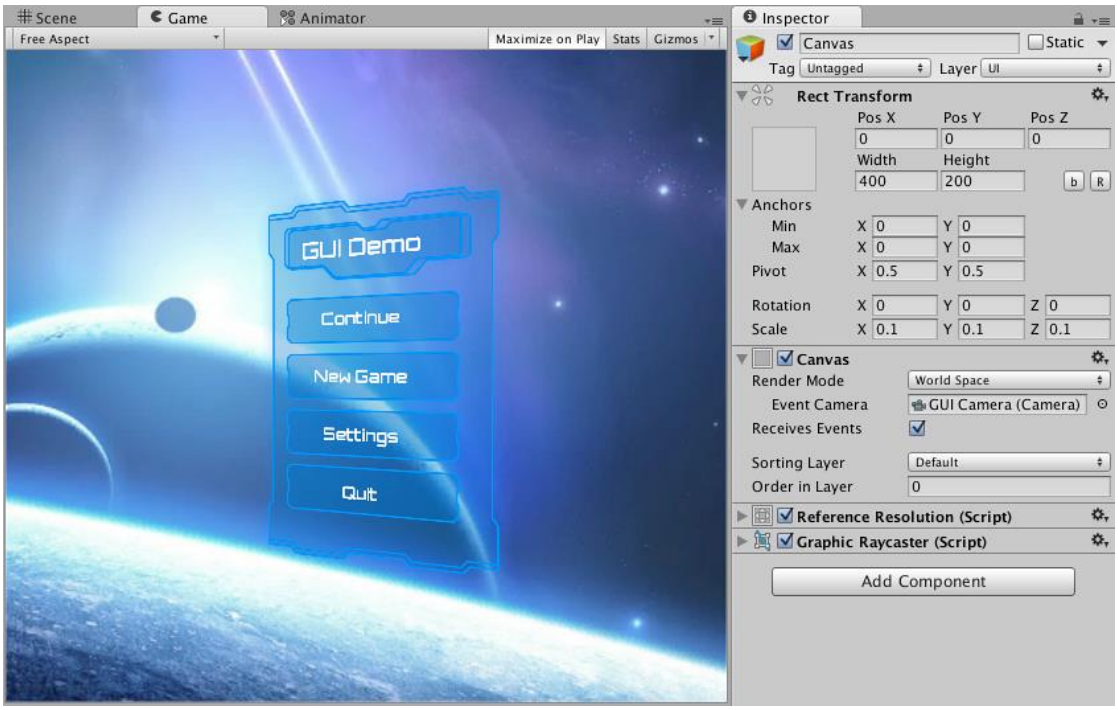
Kuva 5. Canvas-näkymä kamerasta

Samanlainen kuin Screen Space, Screen Space – Camera piirtää Canvasin tiettyyn etäisyyteen kameran eteen. Tämän kameran piirtää UI-käyttöliittymän elementit, mikä tarkoittaa, että kameran asetukset vaikuttavat käyttöliittymän ulkoasuun.



Kuva 6. Canvas näkymä editor-näkökulmasta

Jos Kamera on asetettu Perspektiiviin, UI-elementit näytetään perspektiivinä ja perspektiivisen vääristymän määrää voidaan ohjata kameran näkökentän avulla. Jos näytön resoluutiota muutetaan, tai kameran kuvakulma muuttuu, kangas vaihtaa automaattisesti kokoa vastaavaksi.



Kuva 7. Canvas näkymä pelimaailmassa.

World Space piirustustilassa Canvas toimii, kuten mikä tahansa muu peliobjecti. Canvas-koko voidaan asettaa manuaalisesti ja UI-elementit tulevat 3D-sijoittelun perusteella esille muiden kohteiden edessä tai takana. Paikka-koordinaatistoa voi muuttaa pelimaailmassa samanlailla, kuin muidenkin peliobjetien (kuva 7).

4.4 Ajan hallinta unityn sisällä

Update-funktiolla voi seurata syötteitä (input) ja muita tapahtumia säännöllisesti scriptistä ja toimia sen mukaan tarvittaessa. Esimerkiksi siirtää hahmoa, kun "eteenpäin" -näppäintä painetaan.

```
//C# script example
using UnityEngine;
using System.Collections;

public class ExampleScript : MonoBehaviour {
    public float distancePerFrame;

    void Update() {
        transform.Translate(0, 0, distancePerFrame);
    }
}


//JS script example
var distancePerFrame: float;

function Update() {
    transform.Translate(0, 0, distancePerFrame);
}
```

Kuva 8. Koodi esimerkki objektin siirtämisestä

Tärkeä asia muistaa käsitellessä aikapohjaisia toimintoja kuten tämä on, että pelin kuvataajuus (framerate) ei ole vakio, eikä myöskään aika Update-funktio kutsujen välillä. Eri suorituskykyinen tietokone toteuttaa käskyt nopeammin tai hitaammin, josta johtuen asiat tapahtuvat eri aikaan. Esimerkkinä tästä on siirtää objektin asteittain eteenpäin, yhden kehyksen kerrallaan.

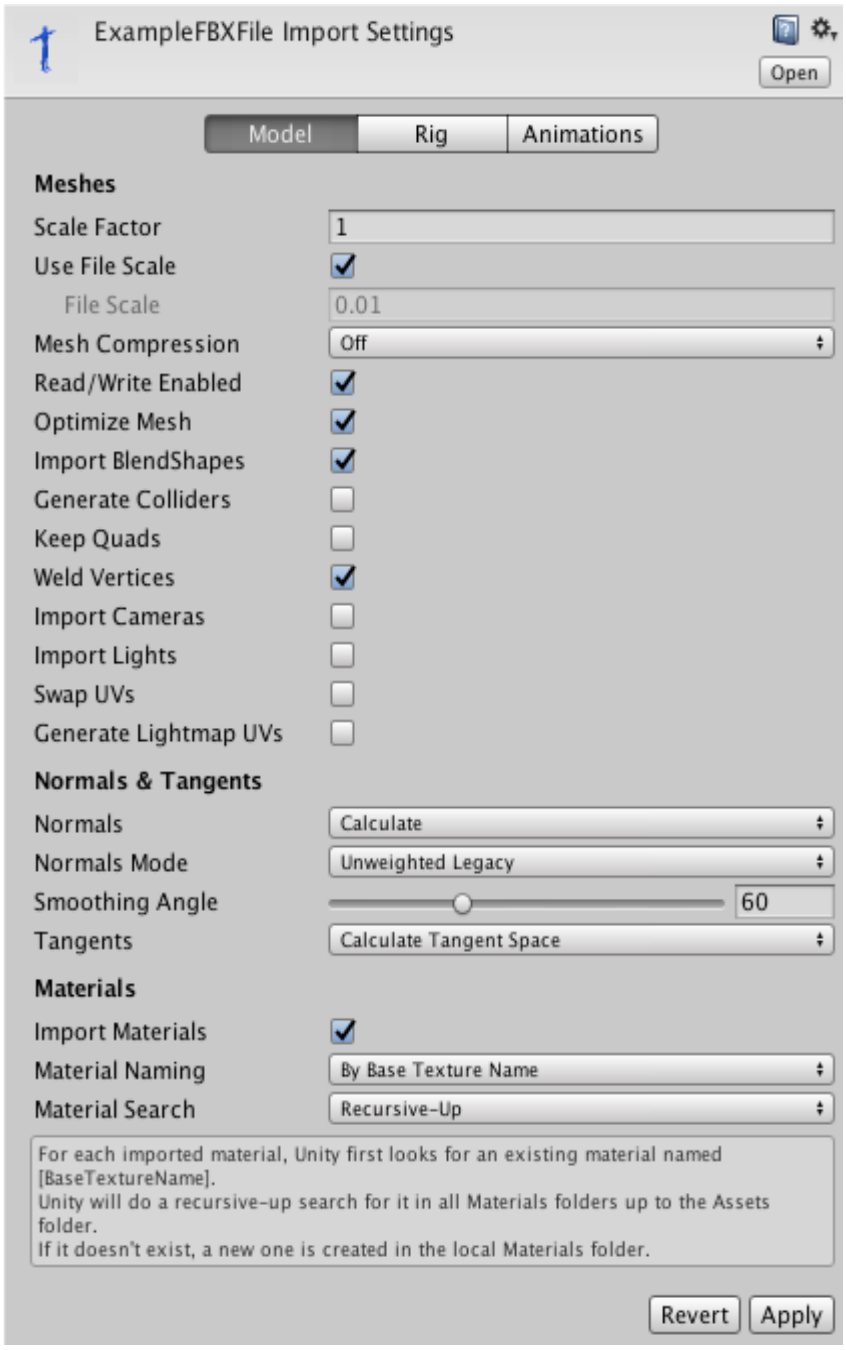
```
//C# script example
using UnityEngine;
using System.Collections;

public class ExampleScript : MonoBehaviour {
    public float distancePerSecond;

    void Update() {
        transform.Translate(0, 0, distancePerSecond * Time.deltaTime);
    }
}


//JS script example
var distancePerSecond: float;

function Update() {
    transform.Translate(0, 0, distancePerSecond * Time.deltaTime);
}
```

Kuva 9. Koodi esimerkki objektin siirtämisestä Time.deltaTime avulla

Objekti näyttää liikkuvan epätasaisella nopeudella, jos kuvataajuus ei ole vakio. Jos kuvataajuus on 10 millisekuntia, kohde etenee eteenpäin distancePerFramella sata kertaa sekunnissa, mutta jos kuvataajuus kasvaa 25 millisekunnin ajan (CPU:n kuorman vuoksi), kohde etenee vain neljäkymmentä kertaa sekunnissa. Ratkaisu on skaalata koko sen liike ajan mukaan, joka toteutuu Time.deltaTime-ominaisuudella.

5 HAHMON LIIKKUVUUS JA MODELI

Unity tukee 3D-malleja, kuten rakennuksen tai huonekalun. Kun mallin tuo Unityyn se tulee useina kohteina, kuten myös mallinnusohjelmassa. Tämä ilmenee hyvin projekti-ikkunassa, jossa itse mallin alle on luotu jokainen kappale lapsiobjektina. Tästä johtuen on järkevää luoda Prefab-objekti mallista ja tuoda se projektiin.



Kuva 10. Modelin tuonti-ikkunan asetukset

Mallitiedosto voi sisältää myös animaatiodataa, jota voidaan käyttää tämän mallin tai muiden mallien animointiin. Animaatiotiedot tuodaan yhdeksi tai useammaksi Animaatio-leikkeeksi riippuen käytetystä mallinnusohjelmasta. Mallitiedoston tuontiasetukset näkyvät FBX-ikkunan Model-välilehdessä, kun malli on valittu.

5.1 Syncvar

SyncVars ovat muuttujia NetworkBehavior-komentosarjoista, jotka synkronoidaan serveriltä asiakkaisiin. Kun kohde on luotu maailmaan tai uusi pelaaja liittyy peliin, lähetetään niille kaikkien SyncVarsin viimeisin tila suhteessa verkkossa oleviin peliobjekteihin, jotka ovat näkyvissä niille. Jäsenmuuttujat tehdään SyncVars-palveluun käyttämällä [SyncVar]-määriteltyä attribuuttia.

```
class Player : NetworkBehaviour
{

    [SyncVar]
    int health;

    public void TakeDamage(int amount)
    {
        if (!isServer)
            return;

        health -= amount;
    }
}
```

Kuva 11. Sycvar esimerkki koodissa

SyncVarsin tilaa sovelletaan kohteisiin asiakkaille ennen kuin OnStartClient() kutsutaan, joten objektin tila on taattu olevan ajantasainen sisällä OnStartClient().

SyncVars voi olla perustyyppejä, kuten kokonaislukuja, merkkijonoja ja kelluk-keita. Ne voivat olla myös Unity-tyyppejä, kuten Vector3 ja käyttäjän määrittämiä rakenteita, mutta päivitykset Struct SyncVars-ohjelmistoille lähetetään monoliittisina päivinä, ei inkrementaalisin muutoksin, jos kenttien sisällä olevat kentät muuttuvat. Yhdessä NetworkBehavior-komentosarjassa voi olla jopa 32 SyncVarsia - tämä sisältää SyncListit. SycnVar-päivitykset lähetetään automaattisesti serveriltä, kun SyncVarin arvo muuttuu.

5.2 Synclist

SyncListit ovat kuin SyncVars, mutta ne ovat arvojen luetteloita yksittäisten arvojen sijasta. SyncList-sisältö sisältyy SyncVar-tilaan alkuperäisissä tilan päivityksissä. SyncListit eivät vaadi SyncVar-attribuutteja, vaan ne ovat erityisiä luokkia. Perustietyyppeihin on sisäänrakennetut SyncList-tyypit: SyncListString, SyncListFloat, SyncListInt, SyncListUInt ja SyncListBool

```
public class MyScript : NetworkBehaviour
{
    public struct Buf
    {
        public int id;
        public string name;
        public float timer;
    };

    public class TestBufs : SyncListStruct<Buf> {}
    TestBufs m_bufs = new TestBufs();

    void BufChanged(SyncListStruct<Buf>.Operation op, int itemIndex)
    {
        Debug.Log("buf changed:" + op);
    }

    void Start()
    {
        m_bufs.Callback = BufChanged;
    }
}
```

Kuva 12. Synclist esimerkki koodissa

Näihin myös kuuluu SyncListStruct, jota voidaan käyttää käyttäjien määrittämien rakenteiden luettelointiin. Syntyperäinen SyncListStruct-luokka voi sisältää perustyyppisiä jäseniä, taulukoita ja yhteisiä Unity-tyyppejä. Ne eivät voi sisältää monimutkaisia luokkia tai geneerisiä säiliöitä.

5.3 Network script

Ennen Unet APIn käyttöä scripteissä, pelaajahahmo oli mahdollista luoda peliruudulle, mutta sitä ei voitu liikuttaa. Tämä johtuu siitä, että scriptien metodeja ei välitetty pelaajaklienttien kautta servereille. Unet tarjoaa valmiita scriptejä, joiden liittäminen peliobjektiin antaa sille verkkoidentiteetin.

Verkkoidentiteetti mahdollistaa muun muassa pelaajan liikuttamisen reaaliajassa serverillä, jolloin kaikki yhdistyneet klientit näkevät liikkumisen samaan aikaan. Kuvassa (Kuva 13) on inspectorin kautta päälle laitettava pelaajan verkkoidentiteetti scripti. Scriptille voi määrittää joko ServerOnly tai Local Player Authority ominaisuuden.

Server Onlyn ollessa päällä, vain itse serverillä on oikeus peliobjekttiin. Tällöin pelaaja itse ei voi esimerkiksi hallita hahmonsa liikkumista. Jotta kontrollit toimisivat oikein, täytyy LocalPlayerAuthority ottaa käyttöön. Tällöin objektin hallinta on aina paikallisen koneen klientillä.

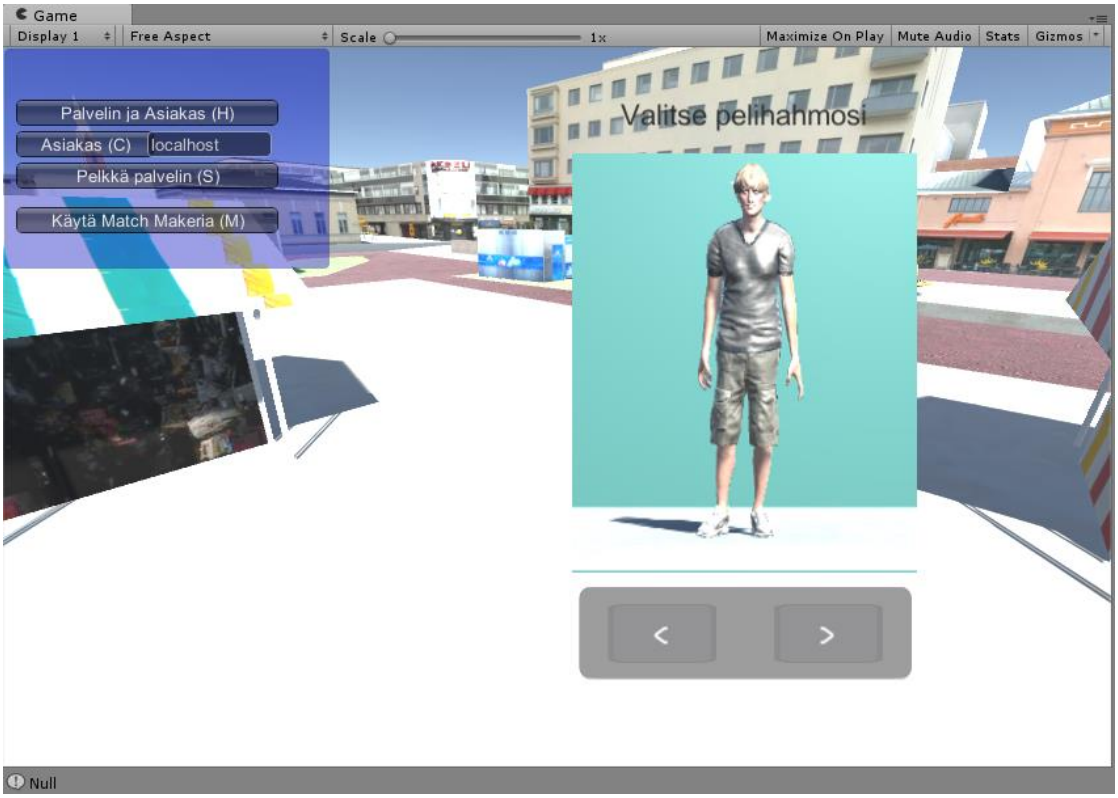
Kuva 13. Unetin verkkoidentiteetti scripti

Kaikki scriptit, joiden halutaan toimivan verkossa, tulee tehdä muutoksia sitä varten. Näille täytyy lisätä using.UnityEngine.Networking, jotta niille voidaan periyttää NetworkBehaviour. Periytyksen avulla scripteissä voidaan käyttää erilaisia RPC-komentoja (Remote ProcedureCalls), jotka tosin ovat vaapumassa unholaan, sekä käskyjä (Commands).

Saadakseen pelin perusominaisuudet (liikkuminen, hyppiminen) toimimaan verkossa, pelaajahahmolle tulee luoda Player_network scripti. Näin varmistetaan paikallisen klientin liityttäminen peliin, että se kykenee hallitsemaan oman hahmonsa toimintoja oikein. (Unity Technologies 2015.)

6 HAHMON VALINTA

Pelihahmon valintajärjestelmä sijaitsee pelin aloitusruudussa. Tämän avulla voi valita hahmon ulkonäön neljästä ennalta määritellystä modelista (kuva13). Valinta toimii nuolinäppäimien avulla, jota kontrolloi hahmopainettu-scripti (kuva 14).



Kuva 13. Hahmo valinta aloitusruudusta

Nuolinäppäimiä painettaessa scripti havaitsee painalluksen ja pyörittää laskurin avulla sitä takaisin ensimmäiseen modeliin saavuttaessaan viimeisimmän. Scripti on sijoitettu nuolipainikkeisiin (kuva 13) ja tarkkailee niiden tapahtumia, jos painallusta tapahtuu hiirellä niiden päällä, niin se lisää tai miinustaa valinnasta riippuen laskurin numeroa ja modeli vaihtuu toiseen.

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

public class hahmoButtonPainettu : MonoBehaviour, IPointerClickHandler
{
    public bool plussa, miinus = false;

    public void OnPointerClick(PointerEventData data)
    {
        if(hahmoButtonEvent.malliNumero > 4)
        {
            hahmoButtonEvent.malliNumero = 3;
        }
        if (hahmoButtonEvent.malliNumero < 1)
        {
            hahmoButtonEvent.malliNumero = 2;
        }
        if (plussa)
        {
            hahmoButtonEvent.malliNumero = hahmoButtonEvent.malliNumero + 1;
        }
        if (miinus)
        {
            hahmoButtonEvent.malliNumero = hahmoButtonEvent.malliNumero - 1;
        }
    }
}
```
Kuva 14. Hahmon valitan koodia

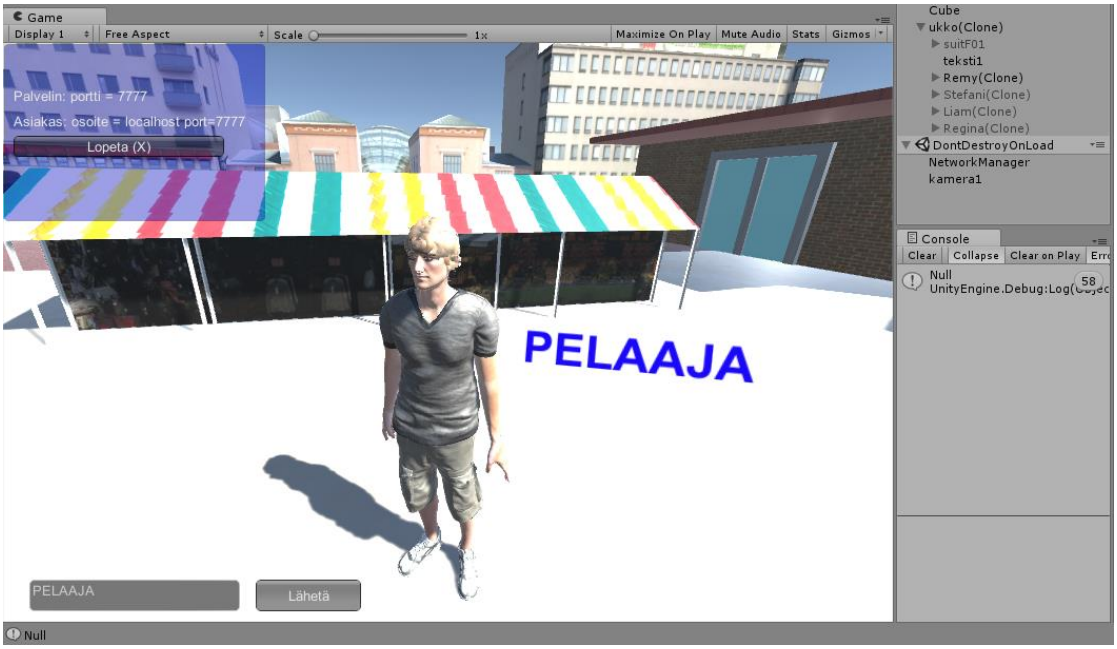
Scripti myös tallentaa muistiin hahmoButtinEvent.mallinumero muuttujaan modelin numeron (kuva 14). Tätä numeroa käytetään myös, kun muille pelaajille pitää lähettää tieto siitä, että mitä modelia tarkalleen pelaaja käyttää tapauskohtaisesti NetworkIdentityn avulla. (kuva 17)

6.1 Hahmon luonti pelimaailmaan.

Peli luo pelaajalle kloonin alkuperäisestä hahmosta maailmaan hänen sinne liittyessään ja näin tapahtuu, joka kerta kun uusi pelaaja sinne liittyy. Klooni siirtyy pelaajan hallintaan NetworkManagerin avulla hänelle annetun numeron (Network Identity) perusteella. Tämä helpottaa hahmojen hallintaa ja resurssien kulutusta, kun ei tarvitse luoda uutta peliobjektia aina uuden pelaajan yhdistyessä pelipalvelimelle vaan alkuperäisestä kopioidaan uusi itsenäinen prefab.

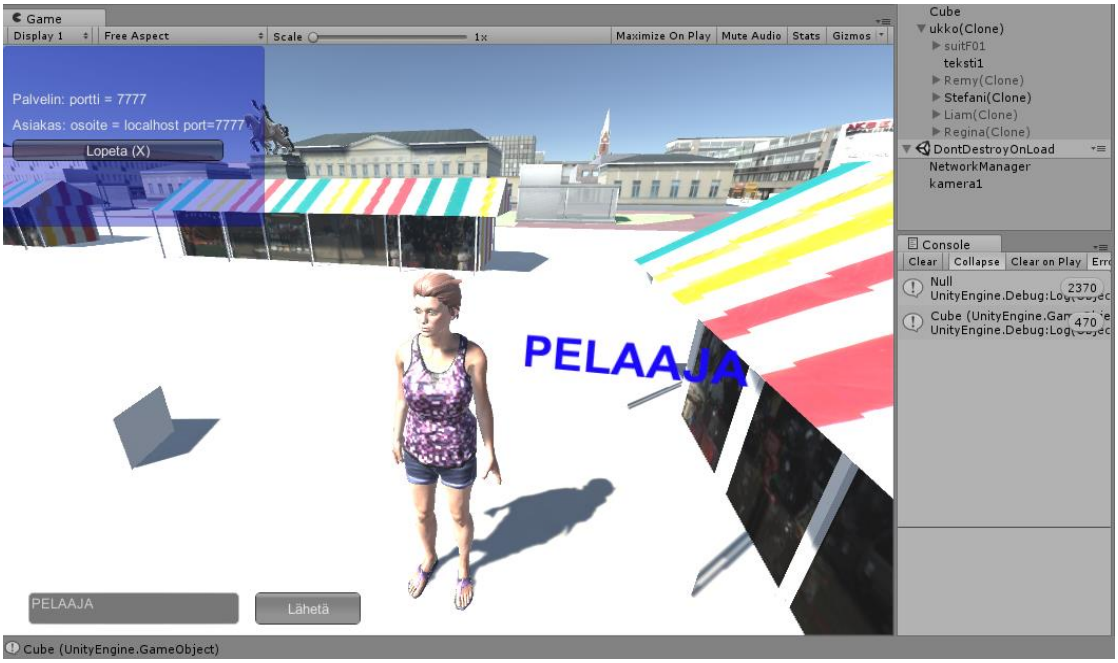
Kaikki modelit ovat piirretty pelin käynnistyessä alkuperäisen pelihahmon peliobjektiin, josta ne kytketään päälle mallinumeron perusteella. Toiset modelit eivät ole piilossa vaan kokonaan pois pelimaailmasta, ennen kuin mallinumero muuttuu, jolloin ne kytketään päälle ja vanha pois. Tämä auttaa pienentämään resurssien kulutusta.

Aloitusruudun modelin valinnan perusteella hahmon modeliksi pelimaailmaan katsotaan valittu. Tässä tapauksessa on mieshahmo päässyt myös peliin saakka aloitusruudusta (kuva 15). Tässä vaiheessa myös muutkin peliin yhdistyneet tarkastavat mikä mallinumero liittyneellä pelaajalla on ja heillekin piirtyy tämä valittu modeli.



Kuva 15. Pelihahmon eri modelit kloonin alla oikealla ylhäällä kuvassa

Halutessaan pelaaja voi lopettaa pelaamisen ja liittyä uudestaan peliin. Tällöin myös voi vaihtaa vapaasti hahmoa, kuten esimerkki kuvassa on vaihdettu naishahmoon mieshahmosta (kuva 16). Peli kohtelee lähtemistä ja uudelleen liittymistä aina uutena tapahtumana pyyhkii lähteneen hahmon kloonin ja siihen liittyvät tiedot maailmasta kokonaan.



Kuva 16. Pelihahmo on vaihdettu naishahmoon

Hahmon valinta käyttää hyväkseen Unityn ominaisuutta, että voidaan kytkeä pois päältä peliobjekteja kokonaan ja halutessa kytkeä niitä taas päälle. Esimerkki script (kuva 17) näyttää miten tämä tapahtuu mallinumero tarkastamalla.

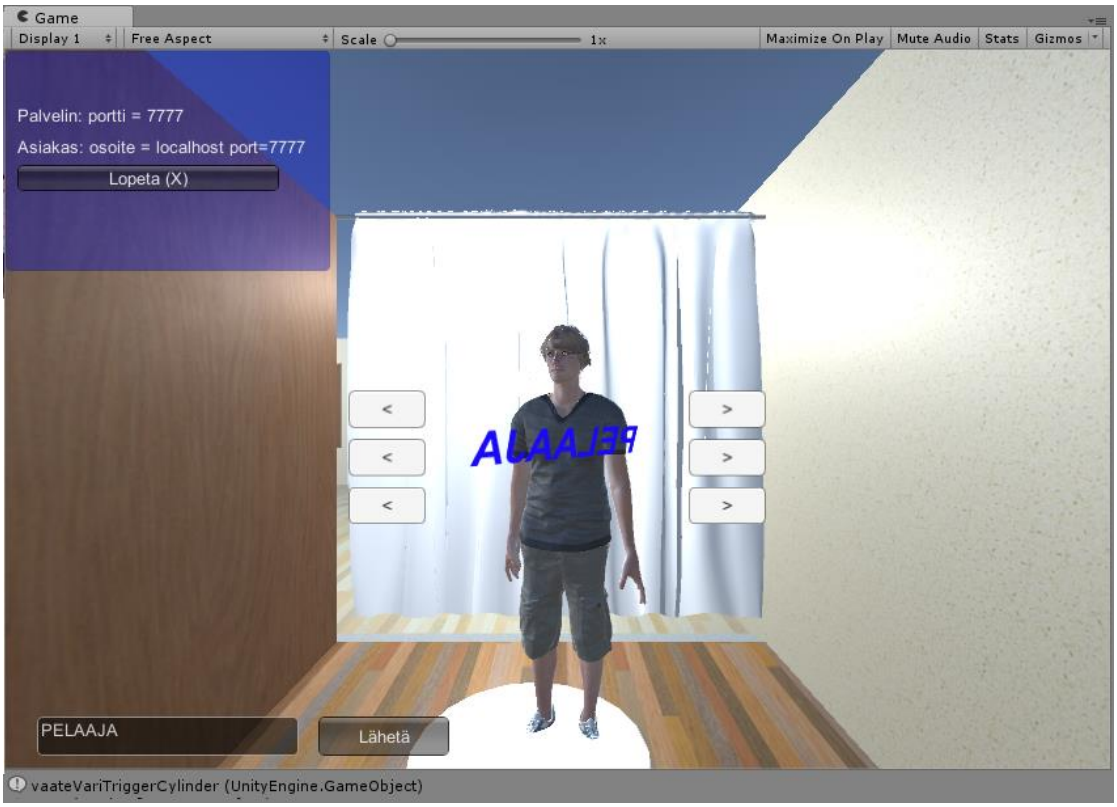
Mallinumeron ollessa tietty, niin kaikki muut kytketään pois ja vain saatu modeli sallitaan aktivoida mallinumeron perusteella.

```
5   public class hahmoButtonEvent : MonoBehaviour {
6
7       public static int malliNumero = 1;
8       public GameObject model1;
9       public GameObject model2;
10      public GameObject model3;
11      public GameObject model4;
12
13      void Start()
14      {
15
16      }
17      void Update()
18      {
19          if (malliNumero == 1)
20          {
21              setActive(model1, model2, model3, model4);
22          }
23          if (malliNumero == 2)
24          {
25              setActive(model2, model1, model3, model4);
26          }
27          if (malliNumero == 3)
28          {
29              setActive(model3, model2, model1, model4);
30          }
31          if (malliNumero == 4)
32          {
33              setActive(model4, model2, model1, model3);
34          }
35      }
36
37
38      public void setActive(GameObject m1, GameObject m2, GameObject m3, GameObject m4)
39      {
40          m1.SetActive(true);
41          m2.SetActive(false);
42          m3.SetActive(false);
43          m4.SetActive(false);
44      }//setAcve
45  }
```

Kuva 17. Koodi esimerkki hahmon vaihdosta

6.2 Hahmon värin vaihtaminen

Pelin sisällä sinne liittyessä pystyy halutessaan vaihtamaan valitun modelin vaatteiden väriä. Värejä, kuten myös modeleja on tässä tapauksessa rajattu määrä, joka toimii myös laskurilla, kun nappeja painaa (kuva 20). Napit kontrolloivat ylhäältä alas katsottuna eri vaateosien väri. Kuten (kuva 18) näkyy vaikuttaa ylimmät napit paitaan, keskimmäiset housuihin ja alimmat kenkien väriin.



Kuva 18. Värinvaihto näkymä

Värien vaihto tapahtuu ennalta määritellyssä paikassa, joka löytyy pelimaailmasta sinne mallinnetun kaupan sisältä. Kaupassa on sovitushuone johon kävelemälle kamera muuttaa kuvakulmaansa peilin suuntaisesti. Kuvassa näkyvälle (kuva 18) pyöreälle valkoiselle alueella kävellessä verho sulkeutuu takana, jonka jälkeen hahmo pakotetaan peilin tiettyyn paikkaan ja se käännetään katsomaan peliä, eli kameraa.

Sovitushuone käyttää hyväkseen scriptissään, joka on sijoitettu valkoiseen pyöreään alustaan Unityn omaa toimintoa nimeltä Ontrigger. Tämä toiminto tarkastelee, jos sen laukaisemiseksi määritelty peliobjekti jakaa alueen sen kanssa mihin peliobjektiin script on kytketty ja toteuttaa koodin sen sisällä. Tämä on niin

verhon sulkeutuminen, kameran kuvakulman muuttaminen ja värinvaihto käyttöliittymän piirtäminen canvaksen kautta vain sen pelaajan ruudulle.

```
4
5   public class vaateKauppaTrigger : MonoBehaviour {
6       GameObject hahmo;
7
8       public GameObject verho;
9       public GameObject triggerCanvas;
10      public GameObject triggerCylinder;
11      public GameObject triggerKamera;
12
13      int t;
14
15      Vector3 keskipiste;
16
17      bool verhoAvautuu=false;
18      bool verhoSulkeutuu = false;
19
20
21      // Update is called once per frame
22      void FixedUpdate()
23      {
24      }
25
26      //otetaan kuka astuu triggerin päälle talteen ja pistetään oikeaan suuntaan ja kohtaan.
27      void OnTriggerEnter(Collider other)
28      {
29          verhoAvautuu = true;
30          verhoSulkeutuu = false;
31          hahmo = other.gameObject;
32          hahmo.transform.rotation = Quaternion.identity;
33          hahmo.transform.position = keskipiste;
34          triggerCanvas.SetActive(true);
35          triggerKamera.SetActive(true);
36      }
37
38      void OnTriggerExit(Collider other)
39      {
40          verhoAvautuu = false;
41          verhoSulkeutuu = true;
42          triggerCanvas.SetActive(false);
43          triggerKamera.SetActive(false);
44      }
45
```

Kuva 19. Esimerkki värinvaihto alueella tulemisesta.

Kaikkien eri vaateosien värienvaihto hoituu saman scriptin kautta. Tämä on saavutettu antamalla boolean arvoja, joita voi kytkeä päälle Unity Inspector-ikkunasta. Tämä sama scripti on osa kaikkia nappeja, joista on määriteltyä mihin sen pitää vaikuttaa.

```
25
26          if (kenkaBool)
27          {
28              ukkokoodi1.variValintaKenka = plusMinusTarkistus(ukkokoodi1.variValintaKenka);
29              ukkokoodi1.variValintaKenka = tarkastin(ukkokoodi1.variValintaKenka);
30
31          }//kenkaBool
32      }//OnPointerClick
33
34      //katsotaan onko luku yli 4 tai alle 1, että kiertää kehää
35      int tarkastin(int variValinta)
36      {
37          if (variValinta > 4)
38          {
39              variValinta = 0;
40          }
41          if (variValinta < 0)
42          {
43              variValinta = 4;
44          }
45          return variValinta;
46      }
47
48      //tuleeko miinusta vai plussaa. Mitä tapahtuu kun nappia painetaan
49      int plusMinusTarkistus(int variValintaPM)
50      {
51          if (plussa)
52          {
53              variValintaPM = variValintaPM + 1;
54          }
55          if (miinus)
56          {
57              variValintaPM = variValintaPM - 1;
58          }
59          if (miinus && plussa)
60          {
61              Debug.Log("+1-1=0, Plus ja Miinus eivät saa olla yhtä aikaa päällä.");
62          }
63          return variValintaPM;
64      }
65  }
```

Kuva 20. Esimerkki miten valitaan värinvaihdon kohde

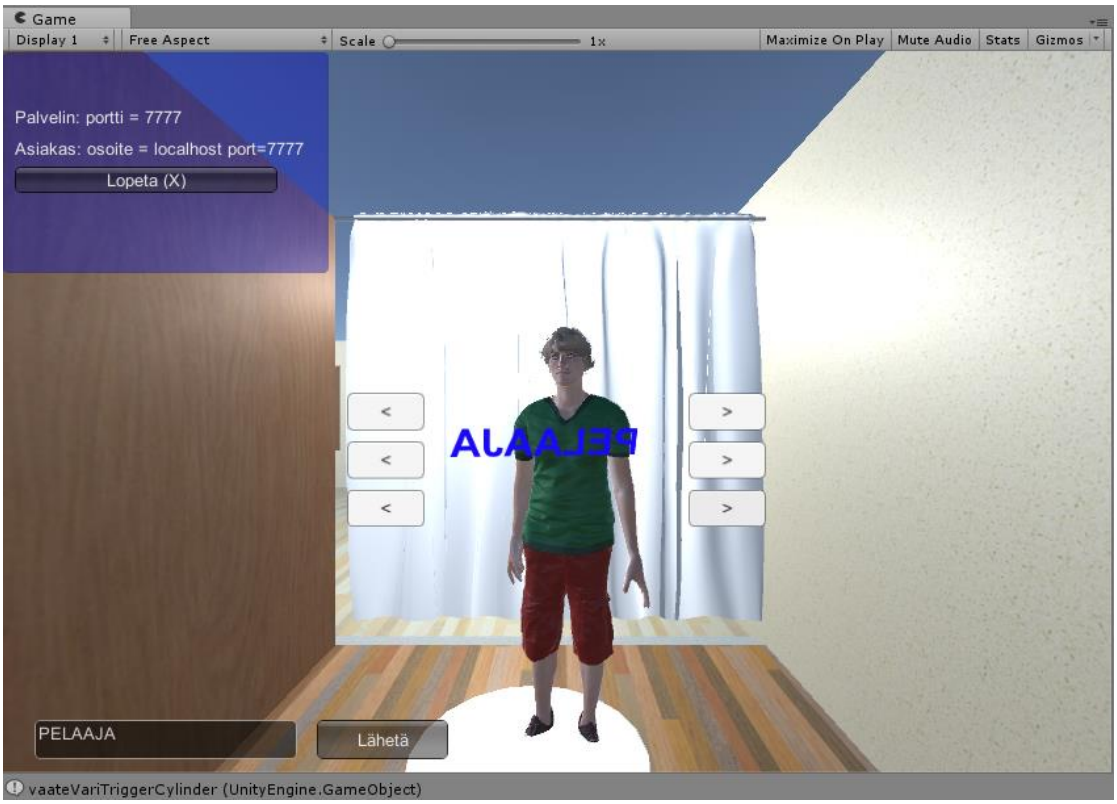
Värinvalinta-tiedot määritellään eri script-tiedostossa, kuin missä niiden vaihto tapahtuu. Värien vaihto katsotaan myöskin mikä boolean on valittu kyseiseen nuoleen, jonka perusteella script katsoo kaikki lapsi objektit pelihahmosta läpi ja poimii sieltä tietyn nimisen ja tekee siihen muutokset nuolien painalluksien mukaan (kuva 21).

Tässä script-tiedostossa on myös mukana tarkistus siitä mitä pelihahmo alueella seisoo. Tällä saadaan värit vaihtumaan vain halutulle pelaajalle, eikä kaikille pelissä oleville. Sen mukana tulee pelihahmo peliobjektin tiedot ja se vertaa niitä käyttäen Unityn omia toimintoja onko kohteen tiedot samat, kuin sen alueen jakavan peliobjektin.

```
            }
            if (this.variHousu == 2)
            {
                vaateVariHousu(0, 1, 0, 1);
            }
            if (this.variHousu == 3)
            {
                vaateVariHousu(1, 0.92f, 0.016f, 1);
            }
            if (this.variHousu == 4)
            {
                vaateVariHousu(1, 1, 1, 1);
            }

            //kenkien värit vaihtuvat tässä
            if (this.variKenka == 0)
            {
                //kenka.GetComponent<Renderer>().material.color = alkuVariKenka;
            }
            if (this.variKenka == 1)
            {
                vaateVariKenka(0, 1, 1, 0);
            }
            if (this.variKenka == 2)
            {
                vaateVariKenka(0, 1, 0, 1);
            }
            if (this.variKenka == 3)
            {
                vaateVariKenka(1, 0.92f, 0.016f, 1);
            }
            if (this.variKenka == 4)
            {
                vaateVariKenka(1, 1, 1, 1);
            }
        }//if
    }

    void OnTriggerEnter(Collider other)
    {
        model = other.gameObject;
        hahmo = model;
        if (!tarkistus&&model!=vanhaModel)
        {
            vanhaModel = model;
        }

        tarkistus = true;
    }

    //otetaan talteen kaikki paidat ja vaihdetaan niitten väriä annetuilla parametreillä.
    void vaateVariPaita(float vari0, float vari1, float vari2, float vari3)
    {
        Transform[] kukkeli = variModelTiedot.hahmo.transform.GetComponentsInChildren<Transform>(true);
        foreach (Transform a in kukkeli) if (a.gameObject.name == "Tops")
                a.GetComponent<Renderer>().material.color = new Color(vari0, vari1, vari2, vari3);
    }//vaateVariPaita
```

Kuva 21. Esimerkki värinvaihdosta

Saatuaan halutut värit pelihahmonsa päälle voi pelaaja yksinkertaisesti vain kävellä alueelta pois ja verho avautuu hänen takanaan ja kamera palautuu kolmanteen persoonaan pelihahmon taakse. Näitä värivalintoja ei tallenneta, vaan kuten pelihahmon kloonikin pelaajan lähtiessä pelistä tuhotaan kaikki tiedot niistä.



kuva 22. Onnistunut värinvaihto

7 LOPPUSANAT

Työn tavoitteena oli tutustua Unity3D-pelikehitysympäristöön ja Unet-verkkorajapinnalla moninpelin tekoon, sekä saada aikaan toimiva osa moninpeli projektiin. Tavoitteeseen päästiin ja aikataulussa ja vaatimuksissa pystyttiin pysymään. Unity3D-ympäristöön ja varsinkin Unet-verkkorajapintaan kuitenkin vielä paljon opiskeltavaa.

Ohjelmointikielenä käyttämäni C# oli minulle kielenä melkein uusi, mutta aikaisemmat opinnot olivat valmistaneet minut hyvin siihen. Sekä Java-kieltä / C käyttäneenä se ei ollut ongelma. UI:n toteutus vain klientti kohtaisesti, ettei näy kaikille samaan aikaan oli kaikista hankalin osa ja eniten aikaa vievä.

En ollut tehnyt tai harrastanut peliohjelmointia aikaisemmin, mutta muilta osin pelin tekeminen osoittautui yksinkertaisemmaksi, kuin luulin. Tähän auttoi varmasti Unityn oleminen alustana.

Aikaa pelikehitys tosin vei ja toisinaan vaikeaksi luultu toteutus saattoi ratketa yksinkertaisesti. Vastapuolena kuitenkin pienen ja yksinkertaiselta näyttävän asian tekeminen saattoi osoittautua vaikeaksi. Itse opinnäytetyön kirjoittaminen yllätti minut paljon aikaa vieväksi.

Lopputulokseen olen tyytyväinen ja sain myös positiivista palautetta. Aina voisi näyttää visuaalisesti paremmalta, mutta asia toimii ja on helppo käyttää.

LÄHTEET

David, H. 2014. Creating E-Learning Games with Unity. E-kirja. Packt Publishing. Saatavissa: https://www.packtpub.com [viitattu 2.11.2017].

Jeff, W. 2008. What is a Game Engine?. WWW-dokumentti. Päivitetty 29.4.2008. Saatavissa: https://www.gamecareerguide.com/features/529/what_is_a_game_.php [viitattu 2.11.2017].

Unity Technologies. 2017. WWW-dokumentti. Päivitetty 10.6.2017. Saatavissa: https://docs.unity3d.com/Manual/UNetOverview.html [viitattu 2.11.2017].

Glenn. F. 2010. What Every Programmer Needs To Know About Game Networking. WWW-dokumentti. Päivitetty 10.6.2017. Saatavissa: https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/ [viitattu 9.11.2017].

Unity Technologies. 2017. WWW-dokumentti. Päivitetty 10.6.2017. Saatavissa: https://docs.unity3d.com/Manual/TimeFrameManagement.html [viitattu 2.11.2017].

Unity Technologies. 2017. WWW-dokumentti. Päivitetty 10.6.2017. Saatavissa: https://docs.unity3d.com/Manual/UICanvas.html [viitattu 2.11.2017].

Unity Technologies. 2017. WWW-dokumentti. Päivitetty 10.6.2017. Saatavissa: https://docs.unity3d.com/Manual/Animator.html [viitattu 2.11.2017].

How To Make My First Person Controller turn left and right in unity 5. 2016. Blogi. Päivitetty 6.6.2016. Saatavissa: https://gamedev.stackexchange.com/questions/116073/how-to-make-my-first-person-controller-turn-left-and-right-in-unity-5 [viitattu 19.11.2017].

Unity Technologies. 2017. WWW-dokumentti. Päivitetty 10.6.2017. Saatavissa: http://docs.unity3d.com/ScriptReference/Network.html [viitattu 2.11.2017].

Unity Technologies. 2017. WWW-dokumentti. Päivitetty 10.6.2017. Saatavissa: http://docs.unity3d.com/Manual/UNetManager.html [viitattu 2.11.2017].