UNITY JA UNET VERKKORAJAPINTA

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittely

Visamäki, kevät 2016

Joona-Mikael Mäkinen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

VISAMÄKI
Tietojenkäsittely
Systeemityö

Tekijä Joona-Mikael Mäkinen **Vuosi** 2016

Työn nimi Unity ja Unet verkkorajapinta

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli kehittää verkossa toimiva PC moninpeli Unityllä. Työn toimeksiantajana toimi HAMK ja Tommi Lahti. Työn tavoitteeseen päästiin ja valmiina tuotoksena syntyi kahden pelaajan verkkopeli. Peli toteutettiin ensin yhdelle pelaajalle, jonka jälkeen Unet verkkorajapintaa käyttäen siihen lisättiin verkko-ominaisuudet. Pelin kaikki sisältö on tuotettu itse.

Koulutöiden ja vapaa-ajalla tehtyjen projektien ansiosta ohjelmointitaidot olivat melko vakaalla pohjalla ennen työn aloitusta. Pelinkehitykseen oltiin myös tutustuttu, mutta muutamaa projektia lukuunottamatta, projektit ovat jääneet ajatustasolle. Työtä aloittaessa Unity työskentely-ympäristönä oli melko tuntematon.

Työ on jaettu teoria- ja käytännönosuuteen. Teoriaosioissa käydään läpi Unityn ja Unetin ominaisuuksia, kuten Unityn käyttöliittymää ja Unetin verkkokonseptia. Käytännönosuudessa käydään koodiesimerkkien ja ruutukaappausten avulla tarkemmin läpi pelin sisältöä ja miten se toimii.

Opinnäytetyötä tehdessä Unity ja Unet todettiin melko helposti omaksuttavaksi kehitysympäristöksi. Ilmenneistä ongelmista selvittiin Unityn kattavan dokumentoinnin avulla. Pelin suunniteltuihin tavoitteisiin päästiin, mutta sen kehitystä on tarkoitus jatkaa harrastemielessä. Seuraavia jatkokehityskohteita ovat esimerkiksi uuden käyttäjäliittymän ja äänien liittäminen peliin.

Avainsanat pelikehitys, Unity, Unet, verkkopeli

Sivut 27 s.

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

VISAMÄKI
Business Information Technology
System development

Author Joona-Mikael Mäkinen **Year** 2016

Subject of Bachelor's thesis Unity and Unet

ABSTRACT

Goal of this Bachelor's thesis was to develop an online game for PC using Unity game engine. The thesis was commissioned by HAMK University of Applied Sciences and Tommi Lahti. Goals of the thesis were achieved and fully working, two player online game was made. The game was first developed as a single-player game. After that the online features were added using Unet. All of the material used in the game were made independently.

At the start of the project, Unity as a game development tool was fairly new to me. The thesis is divided into two parts: theory and practical-part. In theoretical section of thesis Unity's and Unet's features, such as Unity's interface and Unet's networking concept were discussed in detail. In practical-part the game itself was looked into using code examples and screen captures.

During the process of this thesis, Unity and Unet were found to be approachable options for game development. Some minor problems were solved by reading and studying the Unity documentation. All of the intended features were made for the game, but the development of the game is planned to continue as a hobby. Some of the features for future development include new user interface for the game and sound implementation.

Keywords game development, Unity, Unet, online game

Pages 27 p.

TERMISTÖ

Scripti on koodi, joka käsittelee pelin eri ominaisuuksia kooditasolla.

Kaksiulotteinen (2D) peli toimii x ja y-akseleilla, vaaka- ja pystysuunnassa.

Luokka määrittelee scriptissä objektien rakenteen.

Metodi on scriptissä suoritettava komento.

Sprite on kuva, joka toimii pelaajan hahmon grafiikkana.

Spawn tarkoittaa jonkin peliobjektin luontia tai “synnyttämistä” pelinäkymään

Local Host on paikallisessa verkossa hetkellinen serverin ylläpitäjä.

Local Client on paikallisessa verkossa ylläpitäjän serveriin yhteyttä ottava pelaaja.

SISÄLLYS

1 JOHDANTO

Opinnäytetyön valmis tuotos tulee olemaan kaksiulotteinen verkkopeli sekä kyseisen pelin kattava raportointi opinnäytetyössä. Opinnäytetyön pohjalta ei kuitenkaan voi suoranaisesti, tutoriaalin tavoin tehdä valmista verkkopeliä. Verkkopeli toteutetaan Unity-pelinkehityssovelluksella sekä sen verkkorajapinnalla, Unetillä. Työssä käydään läpi Unityn ja Unetin ominaisuuksia sekä tutustutaan tarkemmin, kuinka ne toimivat pelinkehityksessä. Opinnäytetyön aiheen keksin itse, joten työn toimeksiantajana ovat HAMK ja Tommi Lahti. Toimeksiannon vaatimuksena oli perusominaisuuksiltaan toimiva peli, johon liitettäisiin verkko-ominaisuuksia. Työtä ei varsinaisesti tehty opetusaspekti mielessä, mutta kattavan raportoinnin ja valmiin projektin ansiosta sitä voisi käyttää Unityn ja Unetin perehdytykseen. Opinnäytetyön käytännönosuutta on tarkoitus lukea Unity projektin rinnalla, sillä ne tukevat toisiaan hyvin paljon.

Valitsin aiheen, koska pelit ja niiden kehitys ovat olleet aina minulle mielenkiintoisia. Toivomuksenani on päästä työskentelemään pelialalle tulevaisuudessa, joten opinnäytetyö aiheesta voisi toimia mahdollisesti valttikorttina työnhaussa. Ennen työn aloitusta minulla oli hyvin vähän kokemusta Unitystä. Ohjelmointitaitoni olivat kuitenkin melko vakaalla pohjalla erilaisten kouluprojektien ansiosta. Pelien kehityksessä olen ennen opinnäytetyötä ollut lähinnä ajatustasolla, mutta olen kuitenkin tutustunut eri materiaalilähteisiin ennen työn aloitusta. Työstä syntyvään verkkopeliin on tarkoitus tehdä kaikki itse. Jos jollekkin osa-aluille ei löydy aikaa tai taitoa niiden tekemiseen, vähemmän tärkeät asiat jätetään opinnäytetyön ulkopuolelle tai niihin käytetään verkosta saatavia ilmaismateriaaleja. Itse materiaalien, kuten äänen tai grafiikan luomiseen työssä ei keskitytä.

Pelit ovat nykyään hyvin valtavirtaviihdettä ja niiden myyntiluvut nousevat kokoajan suuremmiksi. Vuonna 2015 pelkästään digitaalisten latausten myynti nousi 8 prosenttia vuodesta 2014. Digitaalisia latauksia tehtiin 61 miljardin dollarin edestä. (DiChristopher 2016.) Jo pelkästään näiden tuloksien perusteella näen pelialalla yhä kasvavaa tulevaisuutta ja tahdon olla osa sitä.

Opinnäytetyön tutkimuskysymyksiä ovat:

- Mitkä ovat Unity ja Unet?
- Miten kaksiulotteinen peli toteutetaan Unityllä?
- Miten verkkopeli toteutetaan käyttäen Unettiä?

2 UNITY

Unity on usealle alustalle tarkoitettu pelikehitysmoottori, jolla voidaan tehdä muun muassa kaksi- tai kolmiulotteisia pelejä tietokoneelle, konsoleille, verkkosivuille sekä mobiililaitteille. Pelimoottori on tehty C- sekä C++-ohjelmointikielillä ja se tukee C# sekä Javascriptia. (Unity Technologies 2015l.)

2.1 Historia

Vuonna 2004 David Helgason, Nicholas Francis ja Joachim Anten perustivat Unity Technologies yrityksen, joka kehitti Unity-pelimoottorin. He keskittyivät epäonnistuneen peliprojektin jälkeen luomaan pelimoottorin, joka tekisi pelinkehityksestä kaikille siitä kiinnostuneille helpommin lähestyttävän. Unityn ensimmäinen versio julkaistiin kesäkuun 8. vuonna 2005. (Brodkin 2013.)

Vuonna 2008 Unity oli ensimmäisiä pelimoottoreita, joilla oli täysi tuki iPhonelle ja sen sovelluskehitykselle. Tämän myötä Unity sai enemmän julkisuutta ja yhä useammat isommat yritykset ja yhteisöt ottivat pelimoottorin käyttöön. Nykyään se tukee hieman yli kahtakymmentä alustaa, joihin kuuluu muun muassa PlayStation 4, Xbox sekä Linux. (Brodkin 2013.)

Unityn ensimmäinen ilmainen versio julkaistiin vuonna 2009. Tätä ennen ohjelmasta joutui ostamaan lisenssin, jotta sitä pystyi järkevästi hyödyntämään. Ennen versiota 4.3 Unity oli pääasiassa tarkoitettu kolmiulotteisten pelien tekoon. Ohjelma ei sulkenut kaksiulotteisen grafiikan käyttöä pois, mutta siihen vaadittiin Unityn Asset Storesta ostettuja lisäosia. Versio 5.0 toi Unityn ennen maksullisia olleita ominaisuuksia ilmaisen lisenssin omistajille. Ominaisuuksiin kuuluvat muun muassa äänimikseri, tehokkaampi animaattori sekä uusi fysiikkamoottori, PhysX 3.3. Versiossa 5.1 Unity esitteli uuden tavan toteuttaa verkko-ominaisuuksia sen omalla verkkorajapinnalla, Unetillä. (Unity Technologies 2015d.)

Unityllä on tällä hetkellä noin 4.5 miljoonaa rekisteröitynyttä käyttäjää, joista noin miljoona ovat aktiivisia. Sillä on nopeasti kasvava markkinarvo ja tällä hetkellä se hallitsee määrällisesti kolmiulotteisesti tehtyjä mobiilipelejä. (Unity Technologies 2015a.)

2.2 Ominaisuudet

Unity on kattava pelimoottori ja sen tarjoamilla ominaisuuksilla on mahdollista kehittää toimivia pelejä tai sovelluksia. Ominaisuuksiin kuuluu muun muassa animaatio- sekä sprite-editorit, kaksi- ja kolmiulotteisen grafiikan tuki, erilaiset scriptimahdollisuudet sekä äänimikserit. Unity tukee myös monia eri kehittämisalustoja, kuten iOS, Android, Tizen, Windows Phone 8, Windows, Mac, Linux, web-selaimet,

PS3, PS4, PSVita, Xbox360, Xbox One, Wii U, Android TV, Samsung SMART TV, Oculus Rift, Microsoft Hololens, PS Vr sekä WebGL. (Unity Technologies 2015e.)

Maaliskuun 3. 2015 Unity julkaisi tähän mennessä heidän suurimman päivityksensä, Unity 5.0:n. Se lisäsi muun muassa 64-bittisen käyttöjärjestelmän tuen, paremmat valaistusmahdollisuudet, päivitetyt äänityökalut sekä uuden fysiikkamoottorin. WebGL:n eli web-selain pohjaisen pelaamisen tuki ilman erinäisiä selainlaajennuksia lisättiin Unityyn sen uudessa versiossa. (Unity Technologies 2015c.)

Vaikka Unity 5.0 teki monista ominaisuuksista ilmaisia ja kaikille käyttäjille mahdollisia, siihen tarjotaan myös professional-lisenssiä. Maksullinen lisenssi mahdollistaa muun muassa mukautettavan aloitusruudun pelissä, ryhmälisenssin, analyysityökalut sekä beta-pääsyn Unityn tuleviin versioihin. (Unity Technologies 2015j.)

Unity koostuu pääasiassa kahdesta eri osasta: graafisesta käyttöliittymästä, Unity Editorista sekä käyttäjän valitsemasta IDE:stä. Editorista käyttäjä löytää työkalut, näkymä- ja peliruudun sekä hierarkia, projekti ja objektien tarkastelu ikkunan. Näkymäruudussa käyttäjä voi hallita peliobjekteja visuaalisesti ja sijoitella niitä haluttuille paikoille. Näkymäruudun objektit tulevat myös peliruudulle, mutta sen puolella niiden käyttäytymistä ja ominaisuuksia ei voi enää muokata. Projekti-ikkunasta löytyvät kaikki projektille tarkoitetut assetit eli äänet, grafiikat ja scriptit. Sen kautta niitä voidaan siirtää näkymäruudulle tai jonkin objektin ominaisuuksiksi. Hierarkiaikkuna hallinnoi pelin objekteja ja listaa ne halutussa järjestyksessä. Järjestyksellä ei sinänsä ole väliä, mutta toisen objektin siirto toisen alle voi vaikuttaa sen toimivuuteen. Objektin tarkasteluikkunasta voi muuttaa tai lisätä sen ominaisuuksia, kuten scriptejä tai objektin sijoitusta näkymä- ja peliruudussa. (Unity Technologies 2015k.)

2.3 Ohjelmointiympäristö, API ja IDE

Unity tukee kolmea eri ohjelmointikieltä: C#, Javascriptin kaltaista Unity Scriptiä sekä Boo:ta. Kaikkia Unity-projektia varten tehtyjä koodeja kutsutaan scripteiksi. Unityn editorianalyysien mukaan Boo:n käyttäjämäärä on kuitenkin niin pieni, että sen varsinainen tarjoaminen ohjelmointikielenä mahdollisesti lopetetaan. Projektit, joissa on käytetty Boota, kuitenkin toimivat vielä tämänkin jälkeen. Unity Scripting API löytyy heidän verkkosivuiltaan ja se on toteutettu lähes kuten muidenkin ohjelmointiympäristöjen APIt. Se on jaettu kolmeen eri luokkaan: UnityEngine, UnityEditor sekä Other. Jokainen näistä on jaettu niiden omien luokkien mukaan. Luokkien alta löytyvät vielä niiden muuttujat ja mahdolliset funktiot. Muuttujien ja funktioiden helpottamiseksi Unity Scripting API tarjoaa pieniä esimerkkikoodeja tai tarkempia selityksiä niiden toiminnasta. Scriptien tekoon Unity tarjoaa pääasiassa kahta eri vaihtoehtoa, Visual Studio integraatiota tai Unityn MonoDevelopia. Ne toimivat lähes samalla tavalla, vaikka niiden ulkonäkö onkin erilainen.

Scriptien kirjoittamiseen voi käyttää muitakin ohjelmia, mutta nämä kaksi ovat suosituimpia. (Unity Technologies 2015l.)

Unity on tehnyt scriptien ja niiden muuttujien muokkaamisesta hyvin helppoa. Peliobjektille liitetyn scriptin ominaisuuksia voidaan Unity Editorista muuttaa objektin tarkasteluikkunasta, mikäli ominaisuudet ovat asetettu julkisiksi scriptissä. Muutoksia voi tehdä myös pelin aikana, mutta ne toimivat vain sen hetkellisen pelin ”ajon” aikana. (Unity Technologies 2015k.)

2.4 Gameloop Unityssä

Gameloopilla tarkoitetaan pelin sisäistä kiertoa, joka käy kokoajan läpi sen komponentteja. Yleisesti se koostuu pelaajan antamien komentojen, kuten näppäinpainallusten päivittämisestä, pelin tilan (esimerkiksi onko pelaaja kuollut vai ei) päivityksestä sekä grafiikan päivityksestä tai piirtämisestä. (Pothix 2012.)

Unityssä gameloop on hieman monimutkaisempi ja kattavampi. Gameloop saa alkunsa editorin Reset eli nollaustoiminnosta, jossa scriptien ominaisuudet alustetaan ja asetetaan objektille, kun Reset-toiminto ajetaan tai scripti liitetään ensimmäisen kerran objektiin. Näkymän (Scene) aloituksen aikana ajetaan metodit Awake, OnEnable ja OnLevelWasLoaded. Awakea käytetään alustamaan muuttujia tai pelitiloja ja sitä kutsutaan vain kerran scriptin elinkaaren aikana. OnEnablea kutsutaan, kun jokin pelin objekti aktivoidaan. Sillä voidaan esimerkiksi tulostaa viesti ruudulle pelaajan synnyttyä (Spawn) peliruudulle. OnLevelWasLoaded kutsutaan, kun pelin taso (Level) ladataan. (Unity Technologies 2015f.)

Jos scriptissä on Start-funktio, sitä kutsutaan ennen ensimmäisen kuvaruudun (Frame) päivitystä. Tämä ei ole kuitenkaan pakollinen osa scriptiä. Se on korvattu joissakin tapauksissa Awakella. Start ei päde objekteihin, jotka alustetaan ja luodaan pelin aikana. Se koskee vain niitä objekteja, jotka on lisätty ennen pelin aloitusta editorissa näkymään (Scene). Kuvaruutujen päivitysten välissä on mahdollista kutsua OnApplicationPause. Tätä kutsutaan, kun kuvaruudun lopussa havaitaan tauko, esimerkiksi pelaajan keskeytettyä pelin painamalla tauko-painiketta. (Unity Technologies 2015f.)

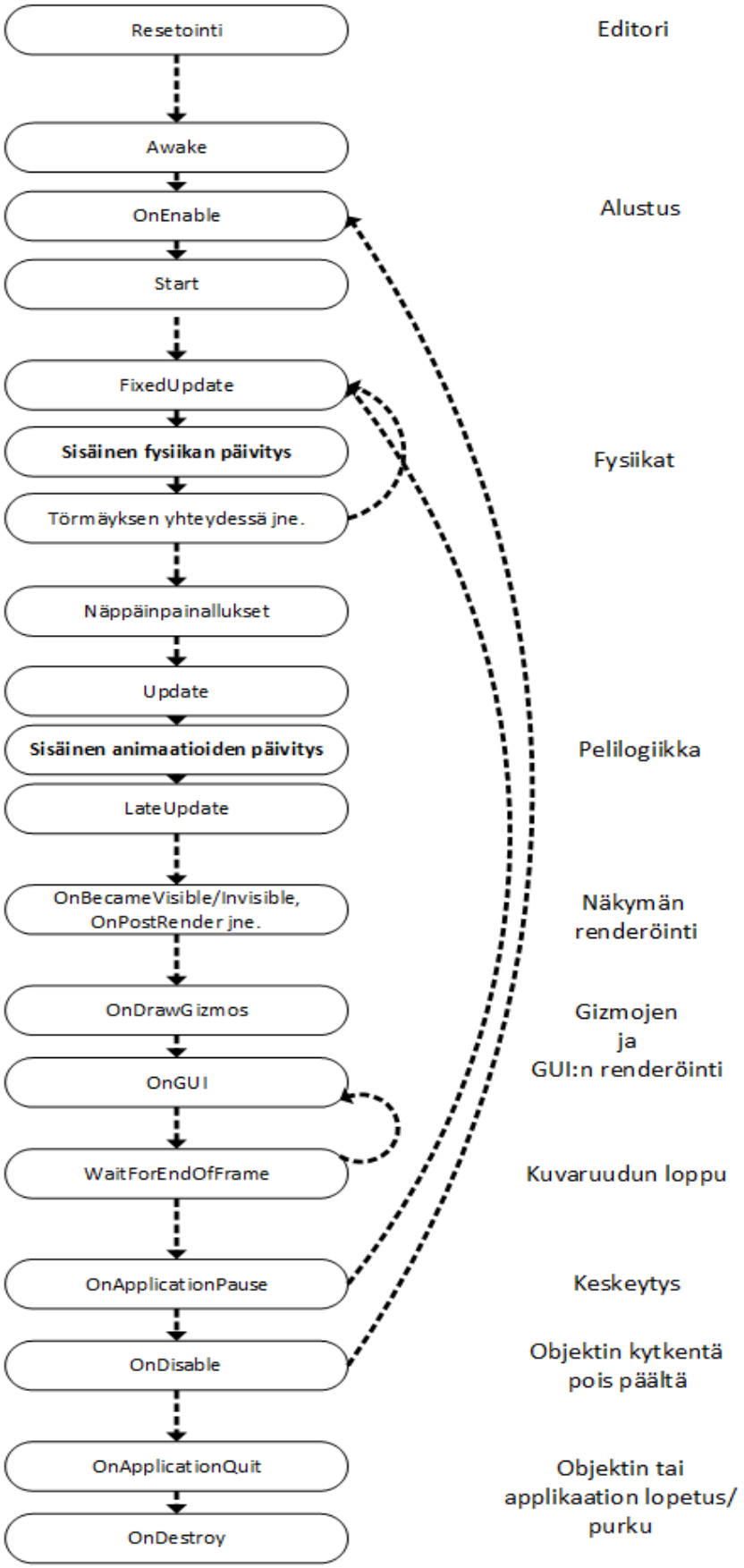
Pelin logiikan tai esimerkiksi animaatioiden seurantaan Unityssä on mahdollista käyttää useampaa päivitysvaihtoehtoa. Yleisin näistä on Update-funktio, jota kutsutaan kerran per kuvaruutu (Frame). FixedUpdatea kutsutaan useamman kerran kuvaruudun aikana. Sen sisällä kaikki fysiikkalaskennat, esimerkiksi pelaajan hyppyakseleiden laskenta, tapahtuvat samantien. FixedUpdate käyttää myös omaa ajastinta, joka on riippumaton kuvataajuudesta (Frame rate).

LateUpdate-funktio kutsutaan kerran per kuvaruutu, kun scripti on käynyt Updaten läpi. Kaikki mahdolliset laskennat, joita on tehty Updatessa, ovat

ehtineet valmistua, kun LateUpdate alkaa. Tämän ansioista esimerkiksi kolmannen persoonan kamera ohjaus voidaan toteuttaa helposti sen sisällä. Hahmon liikkuminen on voitu tehdä turvallisesti Updaten aikana, jonka jälkeen LateUpdate hoitaa kameran liikkumisen. LateUpdate tässä tapauksessa varmistaa, että hahmo on ehtinyt liikkumaan, ennen kuin kameraa seuraa sitä.

Unityssä on myös kameran toimintaan ja grafiikkaan liittyviä päivityksiä, renderointeja (Render). Renderoinnilla tarkoitetaan tässä tapauksessa grafiikan luomista ja piirtämistä peliruudulle. OnBecameVisible tai OnBecameInvisible kutsutaan, kun peliobjekti tulee näkyviin kameralle tai sen poistuessa kameran näköpiiristä. OnPreRender kutsutaan ennen kuin kamera aloittaa pelinäkymän (Scene) renderoinnin. Kehitysvaiheessa yksi hyödyllinen funktio on OnDrawGizmos. Sen avulla voidaan piirtää pelinäkymään normaalisti näkymättömiä objekteja, esimerkiksi pelaajan törmäyksen tarkistuksia. OnDrawGizmos ei näy lopputuotoksessa (Build). Coroutine-pohjaiset päivitykset ajetaan, kun Update palauttaa jotakin tietoa. Niiden avulla voidaan tehdä esimerkiksi päivityksiä, jotka toimivat vasta, kun Update on käynnyt kaikki omat tehtävänsä läpi. (Unity Technologies 2015f.)

OnDestroy-funktio ajetaan, kun objektin viimeinen päivitysruutu (Frame) on ajettu. Funktiolla voidaan esimerkiksi tuhota objekteja pelistä. Lopetukseen liittyvät päivitykset ajetaan kaikkien pelin objektien läpi. OnApplicationQuit kutsutaan ennen pelin sulkemista. Editorissa se tapahtuu, kun pelin testaus pysäytetään ja Unityn web playerissä, kun webnäkymä suljetaan. OnDisable-funktio kutsutaan, kun jokin pelin objekteista otetaan pois päältä tai siitä tulee toimeton. (Unity Technologies 2015f.) Kuvassa (1) on hieman yksinkertaistettu versio Unityn gameloopista. (Unity Technologies 2015f.)



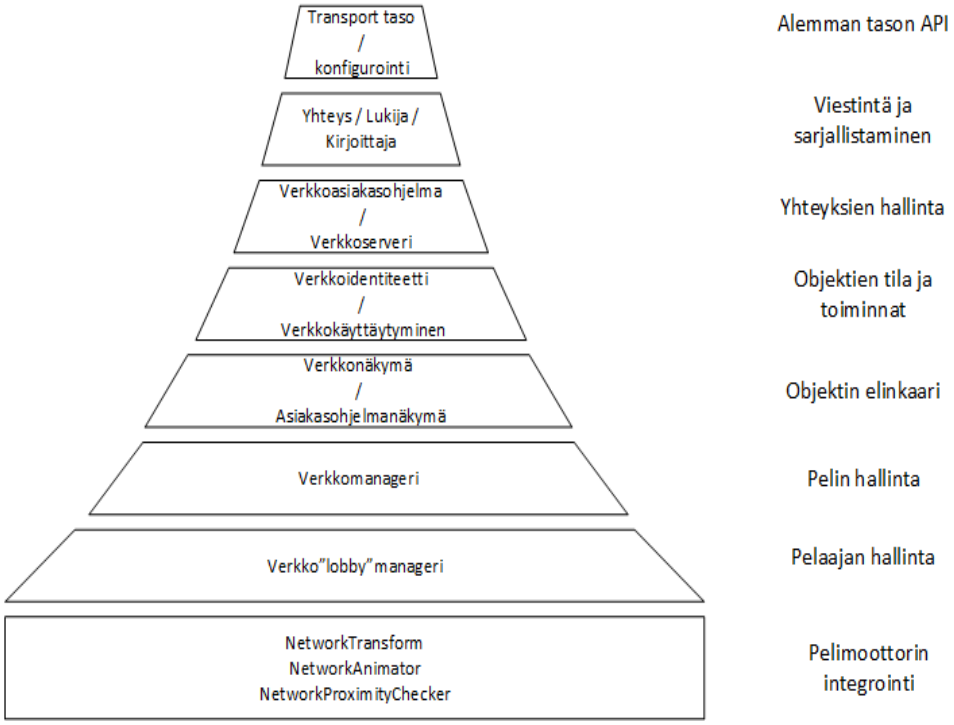
Kuva 1 Gameloop Unityssä

3 UNET–VERKKORAJAPINTA

Unity kertoi vuonna 2014 Unite Asia -konferenssissa julkaisevansa uuden moninpelirajapintansa, Unetin. Sen pääominaisuudet ovat korkean suorituskyvyn tarjoava UDP (User Datagram Protocol) eli yhteydettömän protokollan alusta eri pelityypeille, alemman tason API (LLAPI), jolla voidaan luoda yksinkertaisia yhteyksiä pelaajien välille sekä korkean tason API (HLAPI), joka mahdollistaa helpon ja turvallisen yhteyden asiakasohjelman (Client) ja serverin välille. Unetissä on myös ominaisuutena matchmaker-serverit, joiden avulla käyttäjä voi luoda pelihuoneita useammalle pelaajalle. Uusi verkkorajapinta ratkaisee myös pelaajien yhteysongelmia, joiden syynä ovat olleet palomuuriasetukset. Verkko-ominaisuudet on Unityssä integroitu pelimoottoriin sekä Unity editoriin, mahdollistaen objektien verkkoasetusten editoinnin sekä graafisella että scripti puolella. Esimerkkejä tästä ovat verkkoidentiteetti (NetworkIdentity), jonka avulla objektit saadaan kulkemaan verkon kautta useammalle pelaajalle, sekä verkkokäyttäytyminen (NetworkBehaviour), joka voidaan periyttää scripteille antaen niille verkkotoimivuuden. (Unity Technologies 2015g.)

Unetin alemman tason API, toiselta nimeltään Transport Layer API, on ohut toimintataso rakennettuna käyttäjän käyttöjärjestelmän paikallisen portti- ja verkkojärjestelmän päälle (socket-based networking). Se lähettää ja vastaanottaa viestejä, jotka ovat muotoiltuna taulukkotavuina (byte arrays). Alemman tason APIn tarkoituksena on keskittyä yhteyksien joustavuuteen ja suorituskykyyn. Se tukee perustoimintoja verkkokommunikoinnissa, johon kuuluu yhteyksien luominen, yhteyshallinta, statistiikka ominaisuudet sekä mahdollinen kommunikaatio Unityn relay-serverin kautta. Transport Layeriä käyttäessä perus toimintakaava on seuraavanlainen: Transport tason alustus, verkkotopologian konfigurointi, hostin luonti, verkkokommunikaation aloitus eli viestien lähetys ja vastaanotto sekä tason sammutus käytön loputtua. (Unity Technologies 2015m.)

Opinnäytetyötä varten tehdyssä pelissä on käytetty korkean tason APIa. Se käsittää kaikki yleisimmät vaatimukset useamman pelaajan pelille. Unity API manuaalin ”The high level API”-osuudessa korkeamman tason APIn kerrotaan koostuvan useammasta tasosta, jotka tukevat toinen toisiaan. Kuva 2 demonstroi näiden eri tasojen järjestystä ja niiden tarkoituksia. (Unity Technologies 2015h.)



Kuva 2 HLAPIn tasot

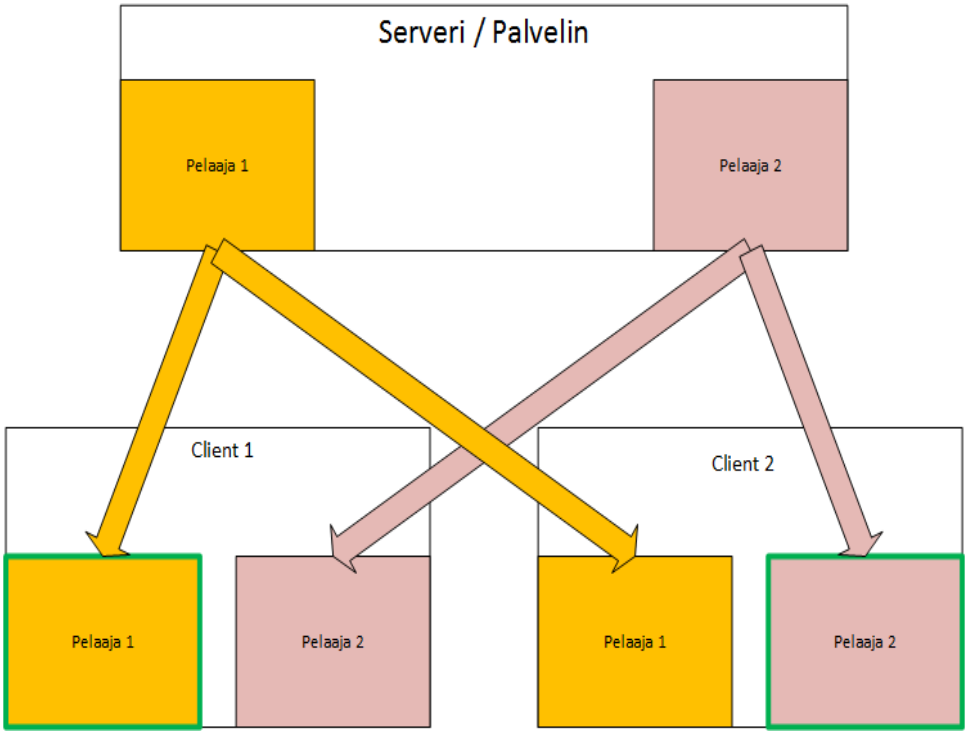
3.1 Verkkokonsepti Unityssä

Unityllä tehdyt verkkopelit käyttävät käytännössä yhtä serveriä ja useaa clienttiä (asiakasohjelma). Kun käytössä ei ole dedicated serveriä (serveri, jossa pelaaja ”lainaa” valmiiksi tehtyä serveriä, jossa ei itse pelaa), yksi clienteistä toimii serverin ylläpitäjänä, hostina. Host toimii tällöin sekä palvelun tarjoajana että clienttinä ja käyttää paikallista clienttiä (LocalClient), kun taas muut serverillä pelaajat toimivat etäclienttien (RemoteClient) kautta. Unityn verkkosysteemin pyrkimyksenä on käyttää yhtä ja samaa koodia sekä Local- että RemoteClientissa, jotta kehittäjien tarvitsisi miettiä aina vain yhden kaltaista clienttiä. (Unity Technologies 2015n.)

Normaalisti Unityssä objektien luonti pelinäkymään tapahtuu komennolla Instantiate. Jos objekti halutaan näkyviin myös verkkopuolelle, se täytyy ”spawnata” aktiiviseksi verkossa. Tämän voi tehdä vain servereillä, jonka seurauksena objekti luodaan peliin yhdistettyjen clienttien näkymään. Normaalisti luotujen objektien ja spawnattujen objektien elinkaari on hyvin samanlainen ja ne tottelevat samoja gameloopin käytäntöjä. (Unity Technologies 2015n.)

Vaikka objektien elinkaari on samankaltainen, niillä on silti verkkopuolella erilaisia toimintoja ja ominaisuuksia. Jokaiselle pelaajalle on verkkopelissä määritelty oma objekti, jota he ohjaavat. Verkon välityksellä näille objekteille välitetään scriptien komennot. Pelaaja voi kutsua komentoja vain omalle, mutta ei toisen pelaajan objektille. Esimerkiksi pelaajan kuoleminen moninpelissä tapahtuu kuolevan pelaajan komentojen kautta, vaikka kuolema johtuisikin toisen pelaajan ampumasta panoksesta. Kun pelaaja liittyy verkkopeliin, hänen

objektistaan luodaan paikallinen objekti kyseiselle clientille. Tämä tapahtuu kaikilla peliin liittyvillä pelaajilla. Scripti puolella tässä tapauksessa käytetään boolean komentoa isLocalPlayer, jolla voidaan tarvittaessa tutkia esimerkiksi ohjaako pelaaja itse jotakin objektia vai hoitaako objektin ohjaamisen serverin toinen pelaaja. Kuvassa 3 tämä asia todetaan yksinkertaisesti.



Kuva 3 Oman pelaajan sijoittuminen clienteissä

Vain pelaajalla itsellään voi olla oman objektinsa isLocalPlayer boolean asetettuna todeksi. Unet APIssa on muitakin boolean tarkastuksia isLocalPlayerin lisäksi, kuten isServer, isClient sekä hasAuthority. (Unity Technologies 2015n.)

Unityn versio 5.2 mahdollisti client-auktoriteetin antamisen pelaajasta riippumattomille objekteille. Tämän voi tehdä kahdella eri komennolla: NetworkServer.SpawnWithClientAuthority tai NetworkIdentity.AssignClientAuthority. Molemmat näistä antavat kuitenkin saman lopputuloksen. Pelaajasta riippumattomia objekteja voi olla esimerkiksi tietokoneälyn ohjaama vihollinen. (Unity Technologies 2015n.)

3.2 NetworkManager

NetworkManager on Unetin komponentti, joka hallitsee verkkomoninpelin eri tiloja. Se on implementoitu Unityyn käyttäen Unet HLAPIa, joten se on täysin kehittäjien muokattavissa.

NetworkManagerin käyttö ei vaadi välttämättä lainkaan koodausta tai scriptejä, sillä siitä löytyy graafisen käyttöliittymän puolelta oma

kontrolleri. NetworkManager tuo mukanaan myös pelinäkymän puolelle NetworkManagerHUD:n (head-up display). Se on oletus käyttäjäliittymä, josta pelin käynnissä olon ajan pelaaja voi vaihtaa pelin perus verkkoasetuksia. Näihin asetuksiin kuuluu LAN (local area network) pelin isännöinti, LAN-osoitteen muuttaminen ja kyseiseen osoitteeseen liittyminen, pelkän LAN-serverin ylläpito sekä matchmaker-servereiden asetukset. (Unity Technologies 2015o.)

NetworkManagerin voi periyttää koodin puolella ja muokata sen asetuksia tarkemmin, esimerkiksi oman HUD:n luonnissa. Komponentti sisältää seuraavat ominaisuudet: pelin tilan hallinnointi, objektien spawnauksen hallinnointi, näkymän hallinnointi, verkkotiedon debuggaus, matchmaking sekä kaikkien ominaisuuksien muokkaus. (Unity Technologies 2015o.)

4 2D-PELIN TOTEUTUS UNITYSSÄ

Toimeksiantona oli toteuttaa yksinkertainen peli, jossa on verkko-ominaisuuksia. Pelin peruskomponentit toteutettiin ensin ilman verkko-ominaisuuksia, mutta järkevintä olisi ollut toteuttaa peli suoraan verkkopuolelle, sillä niin moni scripteistä tuli muuttumaan lähes täysin verkkovaiheeseen siirryttäessä. Kaikki sisältö on tuotettu peliin itse joten ilmaisiin materiaalilähteisiin verkossa ei tarvinnut turvautua. Pelin grafiikat on toteutettu Photoshop-sovelluksella.

4.1 Peliobjektit

Ensimmäisenä peliin lisättiin hahmo, kenttä ja sen tasot sekä kaikki tarvittavat ominaisuudet näille objekteille. Grafiikoiden tuominen Unityyn on tehty hyvin yksinkertaiseksi. Niille voi luoda projektinäkymään oman kansion Assets-kansiojuureen. Kun grafiikat on tuotu projektiin mukaan, Unity antaa mahdollisuuden muuttaa joitakin niiden asetuksia. Asetuksiin kuuluu muun muassa tekstuurityyppi, Sprite-tila, kuvan pivot-piste sekä sen suodintila. Suodintilan avulla voidaan määrittää, tarkentaako Unity grafiikan pikselin tarkkuudella vai pyöristetäänkö kulmat. Unitystä löytyy myös sprite-muokkain, jonka avulla voi esimerkiksi erotella spritesheetistä hahmoanimaatioiden eri kuvaruudut.

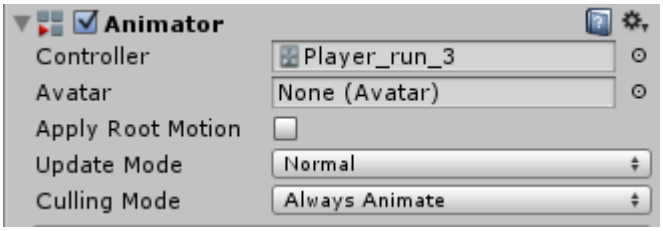
Pelihahmolla on sen objekteina paikkatietojen lisäksi Sprite Renderer, animaattori, Rigidbody 2D, Box Collider 2D sekä erinäiset tarvittavat scriptit. Sprite Rendererin (Kuva 4) liittäminen peliobjektiin mahdollistaa sen piirtämisen pelinäkymään.

Kuva 4 Sprite Renderer komponentti

Unity lisää grafiikkaobjektille automaattisesti Sprite Rendererin, jos kyseinen objekti vedetään pelinäkymään projekti-ikkunasta. Sprite Renderer saa piirrettävän objektin kuvan Sprite-ominaisuudesta, mutta se ottaa myös huomioon Material-ominaisuudesta esimerkiksi valaistukseen ja varjostukseen liittyviä ominaisuuksia. Order in Layer -ominaisuuden avulla voidaan vaihtaa spriten näkyvyyttä pelinäkymässä suhteessa muihin Sprite Renderer komponentin omaaviin objekteihin.

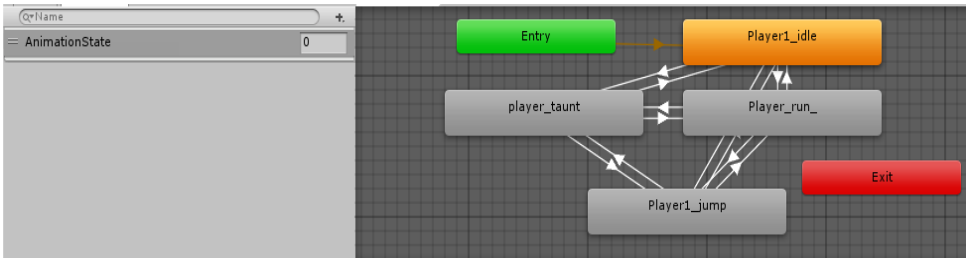
Objektin animaattorikomponentti vastaa sen animaatiosta. Komponentti vaatii viittauksen animaatiokontrolleriin toimiakseen oikein. Jos pelissä

käytetään 3D-animointia, animaattorille voidaan myös antaa oma avatar, joka toimii ikään kuin hahmon luurankona.



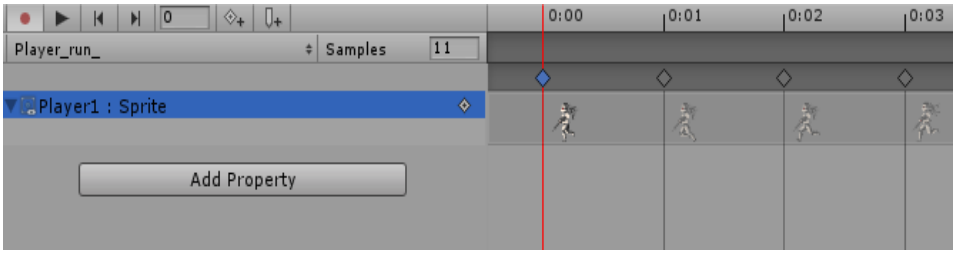
Kuva 5 Animaattori komponentti

Animaatiokontrolleri luodaan Unityn graafisessa käyttöliittymässä käyttäen sen Animator-työkalua. Sen avulla voidaan ohjata eri animaatiotiloja ja milloin hahmon tulisi noudattaa niitä. Animaattori-työkalu toimii ikään kuin tapahtumapuuna animaatioille. Animaatiotilojen vaihtaminen tapahtuu yleensä jonkin pelitilan toteuduttua. Hahmo voi esimerkiksi vaihtaa juoksu-animaatiosta hyppy-animaatioon, kun pelaaja painaa välilyöntinäppäintä. Erinäisiä animaatiotiloja sekä animaatioklippejä on mahdollista hallinnoida scriptien puolella.



Kuva 6 Animaattori työkalu

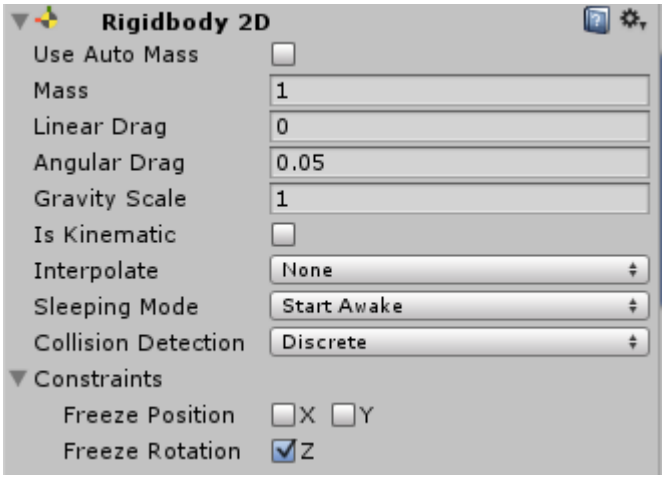
Itse animaatiotilat koostuvat useasta eri spritestä tai spritesheetistä erotelluista kuvaruuduista. Ne asetellaan animaatio-ikkunassa aikajanalle oikeaan järjestykseen, jonka jälkeen yksittäisiä animaatioita voidaan käyttää animaattorissa.



Kuva 7 Animaatioikkuna

Objektin RigidBody2D-komponentti antaa sille fysiikkamoottorin ominaisuudet. Se eroaa RigidBody:stä ainoastaan sen akselimäärien perusteella, sillä RigidBody2D käyttää vain X ja Y-akseleita, kun taas RigidBodyssä on myös lisäksi Z-akseli. Tästä johtuen kaksiulotteinen pelihahmo voi vain liikkua X- ja Y-akseleilla. RigidBody2D-komponentin omaavaa objektia voidaan liikuttaa pelinäkymässä kohdistamalla siihen eriasteisia voimia tai nopeuksia. Kun objektiin on liitettynä myös

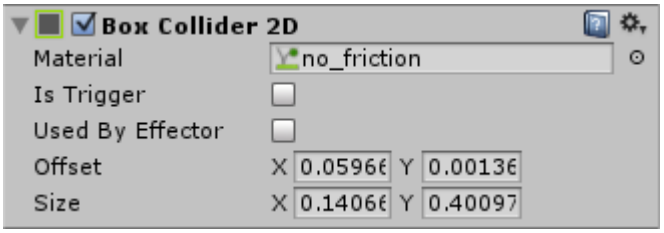
törmäystä tarkistava komponentti, kyseinen objekti käyttäytyy fysiikkamoottorin mukaisesti törmätessään muihin liikkuviin objekteihin. (Unity Technologies 2015p.)



Kuva 8 Rigidbody 2D komponentti

RigidBody2D:n ominaisuus Is Kinematic mahdollistaa painovoiman ja törmäysefektin poiston objektilta. Yleensä tätä käytetään vain hetkellisesti mainittujen ominaisuuksien poistamiseen, jonka jälkeen ne palautetaan scriptissä takaisin objektille. Is Kinematicin avulla voidaan esimerkiksi simuloida räjähdyksen aiheuttamaa paineaalto-efektiä. (Unity Technologies 2015p.)

Box Collider 2D komponentti toimii objektin törmäyksen tarkistajana. Se on nimensä mukaisesti neliön muotoinen ja se on automaattisesti kohdennettuna annettuun objektiin. Colliderin rajoja voi kuitenkin muokata tarkemmiksi tarpeiden mukaan. Se tottelee myös kaksiulotteisia akseleita eli sen mittasuhteet toimivat X ja Y-koordinaatistossa. Colliderille voidaan määritellä myös oma materiaali, jota se noudattaa. Materiaalia muokkaamalla voidaan esimerkiksi määritellä objektin törmäyksessä ilmentyvää kitkaa sekä kimmoisuutta. Colliderin Is Trigger ominaisuudella voidaan määrittää collider toimimaan jonkin pelin tilan tarkastajana. Silloin Box Collider 2D esimerkiksi tarkastaa, onko jokin muu objekti sen törmäystarkistusrajojen sisällä ja tuhoaa itsensä näin tapahtuessa. Unitystä löytyy Box Colliderin lisäksi ympyrä-, kulma- sekä polygoni-collider. (Unity Technologies 2015q.)



Kuva 9 Box Collider 2D objekti

Platformien eli taso-objektien komponentit ei juuri eroa pelaajan komponenteista. Nekin tarvitsevat Sprite Rendererin näytölle piirtoa

varten ja jonkin collidereista törmäyksen tarkistukseen. Kun objekti on luotu valmiiksi, siitä voidaan luoda Prefab. Prefabit ovat objekteista luotuja valmiita elementtejä, joita voidaan myöhemmin käyttää drag-and-drop – menetelmällä pelin luonnissa. Prefabeilla on valmiiksi niille asetetut komponentit ja asetukset. Ne tallennetaan projektinäkymän Prefab-kansioon.

4.2 Hahmon scriptit

Pelaajan hahmolle on liitettynä ohjaamista, ampumista, elämiä sekä kuolema-tilaa hallinnoivat scriptit. Hahmon liikuttaminen vaakatasossa tapahtuu antamalla sen RigidBody2D komponentille nopeutta x-akselilla. Nopeutta voidaan antaa joko positiivisessa tai negatiivisessa suunnassa. Nopeutta kuvaa scriptissä RigidBody2D:n variaabeli velocity. Se on RigidBody2D:n lineaarinen nopeus. Unityn 5.0 versioista eteenpäin RigidBody2D on alustettava scripteissä ennen sen käyttöä, tehden sen käytöstä paljon käytännöllisempää ja helpompaa. Esimerkissä 1 arvo float toimii RigidBody2D:lle annettavana nopeutena. Float antaa nopeudelle tasaisemman arvon ja sulavamman liikkumisen verrattuna integer-tyyppiseen arvoon.

```
RigidBody2D rb;

public float speedF = 5f;

void Start(){

        rb = GetComponent<RigidBody2D>();
}

void Update(){

        if (Input.GetKey(KeyCode.LeftArrow))
            {
                rb.velocity = new Vector2(-speedF, rb.velocity.y);
            }

        if (Input.GetKey(KeyCode.RightArrow))
            {
                rb.velocity = new Vector2(speedF, rb.velocity.y);
            }
}
```

Esimerkki 1 Näppäinkomentojen tunnistus ja hahmon liikuttaminen

Scriptissä oli myös syytä tarkistaa, liikkuuko hahmo ollenkaan. Yksinkertaisissa tapauksissa se ei ole tarpeellista, mutta liikkumattomuuden tarkistuksessa voidaan antaa esimerkiksi hahmolle eri animaatiotiloja. Kun hahmoa ei haluttu liikuttaa, sen liikkumisnopeudeksi asetettiin nolla.

```
void Update(){
    else {
        rb.velocity = new Vector2(0, rb.velocity.y);
    }
}
```

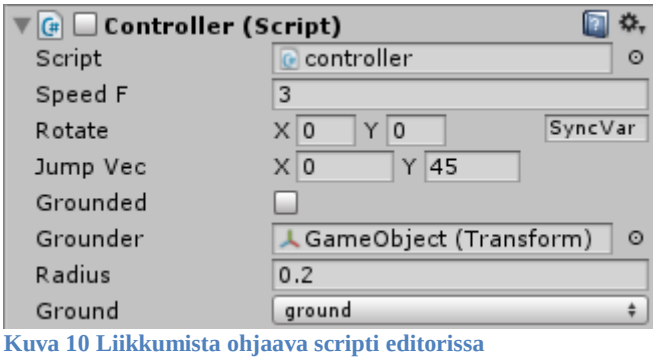
Esimerkki 2 Liikkumaton hahmo

Jotta pelaajalle voitaisiin lisätä hyppy, tarvittiin scriptiin boolean tarkistus, jolla todettiin onko hahmo maassa vai ei. Luokan Physics2D:n funktiolla OverlapCircle voitiin luoda hahmolle näkymätön ympyrä, joka tarkistaa, onko sen rajojen sisäpuolella jokin collider. Scriptissä määriteltiin myös LayerMask attribuutti. LayerMaskilla objekteille voidaan antaa tietty tasotunnus, jonka avulla tarkastellaan esimerkiksi, osuuko kyseiseen tasoon jonkin objektin collider. (Unity Technologies 2015r.)

```
public float radius;
public LayerMask ground;
public Transform grounder;
public bool grounded;
public Vector2 jumpVec;
void Update(){
        grounded = Physics2D.OverlapCircle(grounder.transform.position, radius, ground);
        if (Input.GetKey(KeyCode.UpArrow) && grounded == true)
                {
                   rb.AddForce(jumpVec, ForceMode2D.Force);
                }
}
```

Esimerkki 3 Maan tunnistus ja hyppy

Hyppy on toteutettu antamalla Rigidbody2D:lle sen Y-akselilla voimaa ForceMode2D.Force –komennolla. Voiman intensiivisyys riippuu voimalle annettavan objektin massasta (mitä suurempi massa, sitä enemmän voimaa tarvitaan objektin liikuttamiseen). Jotta hyppy toimisi oikein, pelaajaobjektille täytyy editorin puolelta määrittää objekti grounder sekä tarkistettava LayerMask ground.



Kuva 10 Liikkumista ohjaava scripti editorissa

Koska attribuutit määritettiin julkisiksi scriptissä, niitä voidaan muokata myös editorin puolella. Myös platformeille ja muille tarvittaville

peliobjekteille on määritettävä oikea LayerMask niiden objektien tarkasteluikkunasta.

Kuva 11 Objektin LayerMask

Pelaajan ampuminen toteutettiin käyttäen Raycastia. Raycast on Unityn luokka, joka luo oletuksena näkymättömän viivan pisteestä A pisteeseen B samalla tarkistaen ja raportoiden kaikki objektit, jotka osuvat viivalle. Tämä palauttaa RaycastHit objektin, jota voidaan suoraan kutsua scriptissä. Raycastin aloituspisteenä käytettiin tyhjää peliobjektia, joka sijoitettiin editorissa hahmon eteen sopiviin koordinaatteihin. Näin varmistettiin, että raycastin luoma näkymätön viiva ei tule osumaan itse pelaajaan. Attribuuttien ja tarvittavien alustusten jälkeen tehdään if-lauseen sisällä null-tarkistus. Lauseessa tarkistetaan, onko aloituspistettä asetettu editorissa. Kuvassa on kyseinen tarkistus. (Unity Technologies 2015s.)

```
public LayerMask whatToHit;

public RaycastHit2D hit;

public Transform Shootpoint;

void Awake() {

        Shootpoint = transform.FindChild("Shootpoint");

        if (Shootpoint == null)
        {
                Debug.Log("Lähtöpistettä ei asetettu.");
        }
}
```

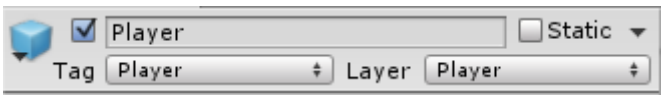
Esimerkki 4 Ampumispisteen alustus

Itse ampumistoiminto toteutettiin samalla periaatteella kuin liikkuminen: jos pelaaja painaa jotakin nappia, pelaaja luo hetkellisen raycastin ja tarkistaa, osuiko viivalle objekteja. Ampumisessa käytetään Unityn valmiita Input-määrityksiä, toisin kuin hahmon liikkumisessa, jossa näppäinkomennoksi annettiin tietty painike. Input.GetButtonia käytettäessä scriptissä ei annata tiettyä painiketta, vaan tapahtumanimi, jolle on jo määritetty painike. Näitä määrityksiä voi editorin puolella muuttaa.

```
if (Input.GetButtonDown ("Fire1"))
{
    Shooting();
}

void Shooting() {

    Vector2 ShootPointPos = new Vector2(Shootpoint.position.x, Shootpoint.position.y);
    Vector2 ShootDirection = (Vector2)transform.position - ShootPointPos;

    RaycastHit2D hit = Physics2D.Raycast(ShootPointPos, ShootDirection.normalized, -100, whatToHit);

    if (hit.transform.tag == "Player")
       {
          // Tee vahinkoa kohteeseen
       }
}
```

Esimerkki 5 Raycastin asetus

Esimerkissä 5 RaycastHit2D:lle annetaan sen lähtöpiste, raycast-viivan suunta, viivan pituus sekä LayerMask, jonka objekteihin raycastin on tarkoitus vaikuttaa. Tämän jälkeen scripti tarkistaa, jos raycast osuu ”Player”-nimikkeen omaavaan objektiin, jolloin kyseiseen objektiin tehdään vahinkoa. Objektien nimikkeet voidaan määrittää editorissa objektien tarkasteluikkunasta (Kuva 12).



Kuva 12 Objektin nimike, Tag

Raycastin LayerMask täytyy myös määrittää editorissa, jotta raycast tietää, mitkä objektit sen täytyy rekisteröidä.

Hahmon animaatioiden vaihtaminen scriptissä toteutettiin animaattori-luokan avulla. Kun animaattori on alustettu scriptissä, sen kaikkia osia voidaan muokata tarpeiden mukaan. Tässä tapauksessa hahmon animaatiotiloiksi oli asetettu vaihtuva integer-arvo, jota muutetaan aina näppäinpainallusten ja hahmon suunnan mukaisesti. Esimerkissä (Esimerkki 6 Animaatiotilojen vaihto scriptissä) asetetaan animaatiotilan integer yhteen (1), kun pelaaja on painanut vasenta nuolta.

```
public Animator anim;

void Start() {

        anim = GetComponent<Animator>();
}

void Update() {

        if (Input.GetKey(KeyCode.LeftArrow))
        {
                anim.SetInteger("AnimationState", 1);
        }
}
```

Esimerkki 6 Animaatiotilojen vaihto scriptissä

5 VERKKOPELIN LUONTI

Vaikka peli tehtiin ensin komponenteiltaan lähes valmiiksi ennen verkko-ominaisuuksien lisäämistä, tarvittavien Network-objektien lisääminen ja niiden liittäminen jo entisiin objekteihin ja scripteihin oli melko haastavaa. Haastetta lisäsi hieman myös ohjeiden puuttuminen suurilta osin kaksiulotteisen verkkopelien luomiseen.

5.1 Tarvittavat objektit

Jotta pelin hahmoja, yleistä pelilogiikkaa sekä scriptien ominaisuuksia voitiin siirtää Unetin APIn avulla verkon puolelle, tarvittiin muutama uusi peliobjekti. Peliin lisättiin tyhjä objekti, jolle annettiin NetworkManager sekä NetworkManagerHUD scriptit. NetworkManager on korkeamman API tason luokka, joka hallitsee kaikkia verkkopelin tiloja. Se mahdollistaa editorin puolelta hallinnan esimerkiksi portti- ja IP-asetuksille, prefabien spawnaukselle sekä pelinäkymien vaihtamisen eri verkkopelin tilojen välillä. NetworkManagerista löytyy myös viiveen simulointi, jolla voidaan antaa verkkopelille keinotekoista liikennettä. Viiveen simuloinnin avulla pelin käyttäytymistä voi testata eri viive-tiloissa. NetworkManagerHUD luo peliin oletus käyttäjäliittymän (GUI), jonka avulla pelaaja voi hostata tai liittyä peleihin. Käyttäjäliittymä on tarkoitettu kehitysvaiheeseen ja sitä ei ole tarkoitus käyttää pelin valmiissa julkaistussa versiossa. (Unity Technologies 2015o.)

5.2 Network scriptit

Ennen Unet APIn käyttöä scripteissä, pelaajahahmo oli mahdollista luoda peliruudulle, mutta sitä ei voitu liikuttaa. Tämä johtuu siitä, että scriptien metodeja ei välitetty pelaajaclienttien kautta servereille. Unet tarjoaa muutaman valmiin scriptin, joiden liittäminen peliobjektiin antaa sille verkkoidentiteetin. Verkkoidentiteetti mahdollistaa muun muassa pelaajan liikuttamisen reaaliajassa, jolloin kaikki clientit näkevät liikkumisen samaan aikaan. Kuvassa (Kuva 13 Unetin verkkoidentiteetti scripti) on pelaajan verkkoidentiteetti scripti. Scriptille voi määrittää joko Server Only tai Local Player Authority ominaisuuden. Server Onlyn ollessa päällä, vain itse serverillä on pääsy peliobjekttiin. Tällöin pelaaja itse ei voi esimerkiksi hallita hahmon liikkumista. Jotta kontrollit toimisivat oikein, täytyy Local Player Authority ottaa käyttöön. Tällöin objektin hallinta on aina paikallisen koneen clientillä.

Kuva 13 Unetin verkkoidentiteetti scripti

Kaikkiin scripteihin, joiden halutaan toimivan verkossa, täytyy tehdä pieniä muutoksia. Scripteille täytyy lisätä using.UnityEngine.Networking, jotta ne voidaan periyttää NetworkBehaviourista. Periytyksen avulla scripteissä voidaan käyttää erilaisia RPC-komentoja (Remote Procedure Calls) sekä käskyjä (Commands). Jotta pelin scriptien perusominaisuudet (liikkuminen, ampuminen, jne.) saatiin toimimaan verkossa, pelaajahahmolle luotiin Player_network scripti. Tämä scripti varmistaa, että paikallisen clientin liityttyä peliin, se pystyy hallitsemaan oman hahmonsa scriptejä oikein. Esimerkissä 7 on Player_network scriptin sisältö. Siinä käytetään Unet APIn lisäämää isLocalPlayer boolean lausetta, joka tarkistaa, onko clientyhteys paikallinen. Jos yhteys on paikallinen, se ottaa käyttöönsä scriptille annetut komponentit. (Unity Technologies 2015t.)

```
using UnityEngine;
using System.Collections;
using UnityEngine.Networking;


public class Player_Network : NetworkBehaviour {

   void Start () {

        if (isLocalPlayer)
        {
            GetComponent<controller>().enabled = true;
            GetComponent<Shoot>().enabled = true;
            GetComponent<hp>().enabled = true;
            GetComponent<NetworkAnimator>().SetParameterAutoSend(0, true);
        }
}
```

Esimerkki 7 Pelaajan scriptien päälle laitto

Jotta hahmo liikkuisi oikein jokaisen pelaajan ruudulla, sen liikkumis-scriptiin oli lisättävä käsky hahmon kääntymiselle. Käsky kääntää hahmon sen X-akselilla riippuen siitä, painaako pelaaja oikeaa vai vasenta nuolta. Verkkokäskyt erotellaan normaaleista metodeista [Command]-merkinnällä. Metodin nimi vaatii myös liitteen ”Cmd”, jotta [Command] tunnistaisi sen käskyksi. (Unity Technologies 2015u.)

```
[Command]
public void CmdFlip(bool flip) {

      if (flip)
      {
          rotate = new Vector2(0, 180);
      }
      else
      {
          rotate = new Vector2(0, 0);
      }
}
```

Esimerkki 8 Hahmon kääntö-komento

Kääntymiseen käytettävän Vector2 attribuutin luomiseen käytettiin SyncVaria. Se on variaabeli, joka synkronoidaan servereiltä clienteille. Kun objekti, jolla on SyncVar-attribuutti, spawnataan pelinäkymään,

kyseisen attribuutin viimeisin tila lähetetään serverille. Tällöin attribuutin arvo on aina päivitettynä kaikille pelaajille. (Unity Technologies 2015v.)

```
[SyncVar(hook="UpdateRotate")]public Vector2 rotate;

void UpdateRotate(Vector2 newRotate) {

    transform.eulerAngles = newRotate;
}
```

Esimerkki 9 SyncVar attribuutti

Käskyjen kutsu ei poikkea normaalien metodien kutsumisesta.

```
if (isLocalPlayer) {

        if (Input.GetKey(KeyCode.LeftArrow))
        {
            CmdFlip(false);
        }
        else if (Input.GetKey(KeyCode.RightArrow))
        {
            CmdFlip(true);
        }
}
```

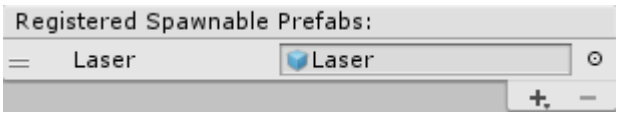
Esimerkki 10 Paikallisen pelaajan kääntö verkon välityksellä

Ampumisen yhteydessä luotava projektiili käyttää samaa käsky-menetelmää. Instantiate on prefabien eli objekteista luotujen mallien luonnissa käytettävä funktio. Se luo alkuperäisestä prefabista kloonin, jota käytetään pelinäkymässä. Klooni luodaan aina joka kerta, kun pelaaja painaa ampumiseen tarkoitettua painiketta. Sille annettavat arvot ovat luotava objekti, sen lähtöpiste sekä lähtöpisteen suunta. Networkser.Spawn luo sille annetun objektin kaikkiin päällä oleviin clientteihin samanaikaisesti. Käskyä kutsutaan, kun pelaaja painaa ampumispainiketta.

```
[Command]void CmdEffect() {

    GameObject bullet = (GameObject)Instantiate(Laser.gameObject,
    Shootpoint.position, Shootpoint.rotation);

    NetworkServer.Spawn(bullet);
}
```

Esimerkki 11 Projektiilin spawnaus

Projektiilin prefab täytyy lisätä editorissa NetworkManageriin rekisteröityjen prefabien listalle. Ilman lisäystä projektiili näkyy vain ampujalle.



Kuva 14 Rekisteröidyt, spawnattavat prefabit

Jotta ampumisella olisi jokin merkitys, sen täytyy tehdä vahinkoa. Ampumisesta vastaava scripti hakee toisesta scriptistä metodin, joka vähentää osutun kohteen elämiä. Metodia kutsutaan scriptin käskyssä.

```
[Command]
void CmdTellServerWhoWasShot(string uniqueID, int dmg) {

      GameObject go = GameObject.Find(uniqueID);
      go.GetComponent<hp>().lowerHealth(dmg);
}
```

Esimerkki 12 Käsky vahingon teolle

Esimerkin 12 käskyssä käytetään uniqueID-merkkijonoa. Koska serverin jokainen pelaaja luodaan samasta prefab-mallista, pelaajille täytyy antaa oma uniikki nimi. Sen luonnissa käytetään prefabin tai sen kloonin alkuperäistä nimeä, jonka perään lisätään yhteyksien mukaan joko numero 1, 2, 3… ja niin edelleen. Alkuperäisen nimen löytämiseen scriptissä käytetään transform.name muuttujaa. Jokaisella pelin objektilla on transform. Se varastoi kyseisen objektin tietoja, kuten paikkaa pelin koordinaatistossa, skaalaa, suuntaa sekä nimeä. Kun alkuperäinen nimi on haettu ja nimeä luodaan paikalliselle clientille, scripti hakee NetworkInstanceId:n avulla pelaajalle annetun tunnisteen. Siitä muutetaan NetworkIdentity, josta oteaan käyttöön vain sen netId. NetId antaa objekteille aina arvon integereinä ykkösestä ylöspäin, riippuen objektien määrästä. Kun yksilöivä id on haettu, se asetetaan merkkijonona pelaajan nimen perään. Metodin GetNetID [Client]-attribuutti (Esimerkki 13 Uniikin tunnisteen luonti pelaajalle) varmistaa, että uniikin nimen luonti tehdään clientin luodessa yhteyden serveriin, eikä ennen sitä. Pelaajalle annetun uniikin nimen avulla jokainen clientti voidaan helposti erotella toisistaan, jolloin eri pelaajille voidaan antaa eri komentoja ja ominaisuuksia.

```
[SyncVar]
public string playerUnique;
private NetworkInstanceId pNetID;
private Transform ntransform;

public override void OnStartLocalPlayer() {
    GetNetID();
    SetID();
}

void Awake () {
    ntransform = transform;
}

void Update () {

    if(ntransform.name == "" || ntransform.name == "Player(Clone)")
    {
        SetID();
    }
}

[Client]
void GetNetID() {
    pNetID = GetComponent<NetworkIdentity>().netId;
    CmdTellServerMyID(MakeUniqueID());
}

void SetID() {

    if (!isLocalPlayer)
    {
        ntransform.name = playerUnique;
    }
    else
    {
        ntransform.name = MakeUniqueID();
    }
}

string MakeUniqueID() {

    string uniqueName = "Player" + pNetID.ToString();
    return uniqueName;
}

[Command]
void CmdTellServerMyID(string name) {

    playerUnique = name;
}
```

Esimerkki 13 Uniikin tunnisteen luonti pelaajalle

Verkkopeleissä on varauduttava clienttien välillä tapahtuvaan viiveeseen, latenssiin. Sillä tarkoitetaan sitä aikaa, mikä yhteydeltä kuluu matkata lähettäjältä vastaanottajalle ja takaisin. Aikaan voi vaikuttaa esimerkiksi clienttien välinen fyysinen matka tai tietokoneella taustalla pyörivät, verkkoyhteyttä käyttävät ohjelmat. Viivettä voidaan helpottaa scripteissä lineaarisella interpoloinnilla. Sillä tarkoitetaan prosentuaalista arvoa kahden eri arvon välillä. Esimerkiksi numeroiden 1 ja 10 välille voitaisiin tehdä 50 prosentin interpolointi, jolloin arvoksi saataisiin 5. Unityssä interpolointiin käytetään funktiota Lerp. Opinnäytetyön verkkopelissä Lerpiä käytetään tutkimaan hahmon liikettä. Liikkeestä tallennetaan Vector2-koordinaatiston pisteet, jotka lähetetään serverille. Scriptiin on luotu kaksi eri lerp-väliä, normaali ja nopea. Lerp funktion arvot on annettu float arvoina sulavamman liikkumisen takaamiseksi. Pelin ajon

aikana scripti tarkastelee yhteyksien latenssia. Jos se nousee liian korkeaksi, scripti valitsee tallennetuista Vector2-koordinaatisto pisteitä edelliset pisteet, interpoloi niiden välin ja siirtää hahmon sulavasti interpoloinnista saatuun pisteeseen. Scripti (Esimerkit 14 ja 15) ei ole kovin joustava ja se toimiikin vain tietyissä latensseissa. Jos latenssi nousee liian korkeaksi, lerp-scriptin arvoja täytyy muokata sopivimmiksi. Korkeampiin latenssiarvoihin voi myös varautua muuttamalla editorista NetworkManagerin timeout arvoja. Timeoutille voi laittaa latenssiylärajan, jolloin serveri katkaisee clientiltä yhteyden, jos sen latenssi ylittää timeoutille annetun rajan. Scriptissä käytettävä Unet-attribuutti [ClientCallback] toimii samalla tavalla kuin aiemmin mainittu [Client], mutta ei generoi varoitus- tai huomautusviestejä Unityn debug-konsoliin. (Unity Technologies 2015i.)

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

[NetworkSettings(channel = 0, sendInterval = 0.033f)]
public class Player_SyncPos : NetworkBehaviour {

    [SyncVar (hook = "SyncPosition")]
    private Vector2 syncPosition;

    [SerializeField]
    Transform mtransform;
    [SerializeField]
    float lerpRate;
    float normalLerp = 16;
    float fasterLerp = 24;

    private Vector2 lastPosition;
    private float threshold = 0.1f;

    private NetworkClient nworkClient;
    private int latency;
    private Text latencyTxt;

    private List<Vector2> syncPositionList = new List<Vector2>();
    [SerializeField]
    private bool usePastLerp = false;
    private float close = 0.2f;

    void Start() {
        nworkClient = GameObject.Find("NetworkManager").GetComponent<NetworkManager>().client;
        latencyTxt = GameObject.Find("Latency").GetComponent<Text>();
        lerpRate = normalLerp;
    }
        void FixedUpdate () {
        TransmitPos();
        LerpPosition();
        Latency();
        }

    void LerpPosition() {
        if (!isLocalPlayer)
        {
            if (usePastLerp)
            {

            }
            else
            {
                NormalLerping();
            }
        }
    }
```

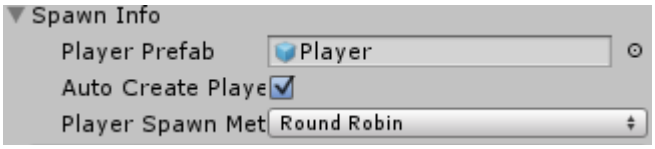
Esimerkki 14 Lerp-scripti (1/2)

```
    [Command]
    void CmdProvidePosToServer(Vector2 pos) {
        syncPosition = pos;
    }

    [ClientCallback]
    void TransmitPos() {
        if (isLocalPlayer && Vector2.Distance(mtransform.position, lastPosition) > threshold)
        {
            CmdProvidePosToServer(mtransform.position);
            lastPosition = mtransform.position;
        }
    }

    void SyncPosition(Vector2 latestPos) {
        syncPosition = latestPos;
        syncPositionList.Add(syncPosition);
    }

    void Latency() {
        if (isLocalPlayer)
        {
            latency = nworkClient.GetRTT();
            latencyTxt.text = latency.ToString();
        }
    }

    void NormalLerping() {
        mtransform.position = Vector2.Lerp(mtransform.position, syncPosition, Time.deltaTime * lerpRate);
    }

    void PastLerping() {
        if (syncPositionList.Count > 0)
        {
            mtransform.position = Vector2.Lerp(mtransform.position, syncPositionList[0], Time.deltaTime * lerpRate);

            if (Vector2.Distance(mtransform.position , syncPositionList[0]) < close)
            {
                syncPositionList.RemoveAt(0);
            }

            if (syncPositionList.Count > 10)
            {
                lerpRate = fasterLerp;
            }
            else
            {
                lerpRate = normalLerp;
            }
        }
    }
}
```

Esimerkki 15 Lerp-scripti (2/2)

Pelaajan hahmon animaatioiden välittämiseen serverille on käytetty Unetin komponenttia Network Animator. Siihen löytyy valmis scripti, jonka voi liittää tarvittaville objekteille. Network Animatorille täytyy liittää aiemmin Kuva 6 Animaattori työkaluluotu animaatiokontrolleri, joka hallitsee animaatioiden vaihtumista hahmolla.

Pelaajien luominen järkevästi peliruudulle ei ole kovin vaikeaa. Pelaajien käyttämä prefab on lisättävä editorissa NetworkManagerin spawn infoon. Pelinäkymään on myös sijoitettu kaksi näkymätöntä objektia, jotka toimivat pelaajien aloituspisteinä. Objekteille on lisätty Unet-scripti Network Start Position. Scripti sijoittaa automaattisesti NetworkManageriin rekisteröidyn prefabin haluttuun aloituspisteeseen. Jotta hahmot eivät aloittaisi samasta pisteestä, NetworkManagerille annettiin kiertävä spawnaus menetelmä, Round Robin (Kuva 15). Se kierrättää aloituspistettä peliin liittyvien clientien perusteella, sijoittaen ensimmäisen pelaajan vasemmalle sekä toisen oikealle ja niin edelleen.



Kuva 15 NetworkManagerin spawn-info

6 YHTEENVETO JA JATKOKEHITYS

Unity on melko helposti lähestyttävä vaihtoehto aloittelevalle pelikehittäjälle. Sen kehitysmoottori on tarpeeksi tehokas vaativien pelien kehitykseen, mutta sillä voi myös tehdä yksinkertaisia tuotoksia. Unitystä löytyy kattava dokumentointi APIn sekä manuaalin puolelta. Unity tarjoaa myös valmiita tutoriaaleja aloittelijoille, ja sen kehitysfoorumitkin ovat suhteellisen aktiivisia. Markkinnoilla ei ole tällä hetkellä kovinkaan montaa vastaavanlaista kehitysalustaa, jos otetaan huomioon Unityn useat ja laadukkaat ominaisuudet. Sen ilmainen lisenssi tarjoaa lähes kaikki samat ominaisuudet, kuin maksullinenkin versio. Ainoat erot tulevat julkaisuprosesseihin, joissa parempi tuki löytyy maksulliselta puolelta.

Opinnäytetyössä päästiin paneutumaan melko syvällisesti verkkopelin kehitykseen ja kuinka se tapahtuu Unityssä. Kaikki materiaali, mitä pelissä käytettiin, toteutettiin itse. Grafiikan tuottamisessa käytettiin Adobe Photoshoppia sekä Asespriteä. Näiden ohjelmien ja grafiikan tuottamisen raportointi jätettiin kuitenkin pois, sillä niiden tuottama lisäarvo työhön olisi ollut hyvin pieni. Opinnäytetyössä ei myöskään ollut tarkoitus tutustua pelimateriaalien tuottamiseen.

Valmis tuotos saatiin aikaiseksi, mutta sen kehitystä on vielä työnkin jälkeen tarkoitus jatkaa harrastemielessä. Pelistä jäi peruskomponentillisesti puuttumaan vain äänet. Pelin julkaisuprosessia ei tarkasteltu paljoa, sillä se ei ollut varsinaisen työn tarkoitus. Esimerkiksi oman vakaan serverin rakentaminen verkkopelin alle olisi suotavaa ennen pelin julkaisua, sillä pelin servereiden yhteydet kulkevat tällä hetkellä Unityn kautta. Peliin on myös tarkoitus rakentaa oma käyttäjäliittymä Unetin tarjoaman oletusversion tilalle. Scripteissä voisi myös olla hieman siivottavaa, mutta kaikki kuitenkin toimivat tällä hetkellä niin kuin pitääkin. Opinnäytetyön aikana pelikehityksestä opittiin paljon ja työstä saaduilla taidoilla on hyvä jatkaa kehitystä jatkossa.

LÄHTEET

Brodkin Jon. 2013. Dice. How Unity3D Became a Game-Development Beast. Viitattu 5.1.2016. http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

Corazza Seraphina. 2013. History of the Unity Engine [Freerunner 3D Animation Project]. Viitattu 5.1.2016. https://seraphinacorazza.wordpress.com/2013/02/14/history-of-the-unity-engine-freerunner-3d-animation-project/

DiChristopher Tom. 2016. Digital Gaming sales hit record $61 billion in 2015: Report. Viitattu 7.3.2016. http://www.cnbc.com/2016/01/26/digital-gaming-sales-hit-record-61-billion-in-2015-report.html

Haas John. 2014. Worcester Polytechnic Institute. A History of the Unity Game Engine. Viitattu 10.1.2016. https://www.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf

Juhl Erik. 2014. Announcing Unet – New Unity Multiplayer Technology. Viitattu 20.1.2016 http://blogs.unity3d.com/2014/05/12/announcing-unet-new-unity-multiplayer-technology/

Pothix. 2012. Gamedev Glossary: What Is the ”Game Loop”? Viitattu 3.2.2016. http://gamedevelopment.tutsplus.com/articles/gamedev-glossary-what-is-the-game-loop--gamedev-2469

Unity Technologies. 2015a. Viitattu 6.2.2016. https://unity3d.com/public-relations

Unity Technologies. 2015b. Viitattu 7.2.2016. https://unity3d.com/5

Unity Technologies. 2015c. What’s new in Unity 5.0. Viitattu 10.2.2016. https://unity3d.com/unity/whats-new/unity-5.0

Unity Technologies. 2015d. What’s new in Unity 5.1 Viitattu 11.2.2016. https://unity3d.com/unity/whats-new/unity-5.1

Unity Technologies. 2015e. Viitattu 11.2.2016.http://unity3d.com/unity/multiplatform

Unity Technologies. 2015f. Viitattu 3.2.2016. http://docs.unity3d.com/Manual/ExecutionOrder.html

Unity Technologies. 2015g. Viitattu 14.2.2016. http://docs.unity3d.com/Manual/UNetOverview.html

Unity Technologies. 2015h. Viitattu 14.2.2016. http://docs.unity3d.com/Manual/UNetUsingHLAPI.html

Unity Technologies. 2015i. Viitattu 2.3.2016. https://unity3d.com/learn/tutorials/modules/beginner/scripting/linear-interpolation

Unity Technologies. 2015j. Viitattu 11.2.2016. https://unity3d.com/get-unity

Unity Technologies. 2015k. Viitattu 11.2.2016. https://unity3d.com/learn/tutorials/topics/interface-essentials

Unity Technologies. 2015l. Viitattu 12.2.2016. http://docs.unity3d.com/ScriptReference/

Unity Technologies. 2015m. Viitattu 14.2.2016. http://docs.unity3d.com/Manual/UNetUsingTransport.html

Unity Technologies. 2015n. Viitattu 15.2.2016. http://docs.unity3d.com/Manual/UNetConcepts.html

Unity Technologies. 2015o. Viitattu 29.2.2016. http://docs.unity3d.com/Manual/UNetManager.html

Unity Technologies. 2015p. Viitattu 28.2.2016. http://docs.unity3d.com/ScriptReference/Rigidbody2D.html

Unity Technologies. 2015q. Viitattu 28.2.2016. http://docs.unity3d.com/Manual/class-BoxCollider2D.html

Unity Technologies. 2015r. Viitattu 28.2.2016. http://docs.unity3d.com/Manual/Layers.html

Unity Technologies. 2015s. Viitattu 29.2.2016. http://docs.unity3d.com/ScriptReference/Physics2D.Raycast.html

Unity Technologies. 2015t. Viitattu 30.2.2016. http://docs.unity3d.com/ScriptReference/Network.html

Unity Technologies. 2015u. Viitattu 1.3.2016. http://docs.unity3d.com/ScriptReference/Networking.CommandAttribute.html

Unity Technologies. 2015v. Viitattu 2.3.2016. http://docs.unity3d.com/ScriptReference/Networking.SyncVarAttribute.htm