Joonas Korhonen

PELISUUNNITTELU JA PELIN ENSIMMÄISEN PROTOTYYPIN TOTEUTTAMINEN

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Marraskuu 2016

KUVAILULEHTI

MAMK University of Applied Sciences	**Opinnäytetyön päivämäärä** 30.11.2016
Tekijä(t) Joonas Korhonen	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittelyn koulutusohjelma
Nimeke Pelisuunnittelu ja pelin ensimmäisen prototyypin toteuttaminen	

Tiivistelmä

Pelien kehitys on muuttunut paljon videopelien noin viiden vuosikymmenen pituisen historian aikana. Työkaluja videopelien kehittämiseen on tarjolla aiempaa enemmän ja ilmaisversiot alentavat pelikehityksen aloittamisen kynnystä.

Käyn opinnäytetyössäni läpi pitkäkestoisen peliprojektin aloittamiseen liittyviä osa-alueita pelin suunnittelun ja Unity3D-pelimoottorin osalta. Aloitan kertomalla pelin suunnittelusta, ensimmäisen tuotoksen kehittämisestä ja versiohallinnasta. Kerron Unity3D:n soveltuvuudesta nopeaan sisällön tuottamiseen sekä pohdin, kuinka jatko-kehitystä voidaan parantaa projektin hallinnan eri keinoin.

Opinnäytetyöni viimeisessä osa-alueessa käyn läpi toimeksiantoni käytännön työtä. Toimeksiantajana toimii Mikkelin ammattikorkeakoulu. Mikkelin ammattikorkeakoulu, Itä-Suomen yliopisto sekä Työväen sivistysliitto ovat toteuttamassa Digiosaajaksi työelämään -hanketta, johon sisältyy tieto- ja viestintäteknologiataitoja parantava työelämäpeli. Käytännön työni kertoo yhden peliin tulevan osa-alueen toteuttamisesta ketterän kehityksen keinoin. Peliprojekti jatkuu oman osallistumiseni jälkeen, joten työni keskittyy mekaniikoiltaan ja pelilogiikaltaan toimivan prototyypin luomiseen. Päätännössä tiivistän toteutuksen ja siinä ilmentyvien ongelmien ratkaisuja sekä pohdin projektin tulevaisuutta ja jatko-kehitystä.

Asiasanat (avainsanat)
Pelisuunnittelu, Unity3D

Sivumäärä 42	**Kieli** Suomi

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi Jukka Selin	**Opinnäytetyön toimeksiantaja** Digiosaajaksi työelämään -hanke

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** 30 November 2016
Author(s) Joonas Korhonen	**Degree programme and option** Business Information Technology
Name of the bachelor's thesis Game design and implementing the first prototype	

Abstract

Game industry has come a long way around its nearly five-decade-long journey. Game development doesn't necessarily require a huge budget or expensive tools anymore. With free to use tools a small team can easily start to develop a game.

This thesis studies the first steps of game development from the game design and the minimum viable product point of view. How to design a games first version in a way that the development can have a good starting point for the years to come? I started this by going through the steps of game design and the Unity3D game engine from the given perspective.

Last part of the thesis focused on the practical part of this study. The assignment was given to me by Mikkeli University of Applied Sciences. Mikkeli University of Applied Sciences, University of Eastern Finland and Worker's Educational Association are creating a serious game to improve the ICT and job application skills of the participants of the course. My practical work was one of the three final parts of the game. The assignment was to build a working minimum viable product that would serve as the ground work for the further development. In the final conclusion I review my work and reflect the results for the future of this larger project and the game industry itself.

Subject headings, (keywords) Game design, Unity3D	
Pages 42	**Language** Finnish
Remarks, notes on appendices	
Tutor Jukka Selin	**Bachelor's thesis assigned by** Digiosaajaksi Tyoelamaan Project

SISÄLTÖ

1 JOHDANTO

Peliteollisuus on kasvanut suuresti viime vuosikymmeninä ja vakiinnuttanut paikkansa yhtenä viihteen muodoista elokuvien ja kirjojen rinnalla. Pelikonsoleiden siirtyminen pelihalleista olohuoneisiin on muuttanut pelialaa sekä laajentanut pelien kirjoa suuresti. Tietotekniikan kehityksen saatossa pelien kehitys vapaasti käytettävin ja sitä varten suunniteltuin työkaluin on tullut mahdolliseksi myös kotioloissa. Opetuskäyttöön suunniteltuja niin sanottuja hyötypelejä on tehty pelialan historian aikana paljon, mutta viime vuosina eri opetusmenetelmien pelillistäminen on erityisesti nostanut päätään. Pelillisin elementein opetukseen saadaan variaatiota, ja ne voivat parhaimmillaan tukea eri oppimisvaikeuksia omaavien oppijoiden oppimista. Pelien kehitys on paljon aikaa ja työtä vaativa prosessi, joka sisältää runsaasti eri alojen osaamista ja yhteistyötä.

Tässä opinnäytetyössä perehdyn pitkäkestoisen pelikehityksen alkuvaiheisiin suunnittelun, toteutuksen ja jatkokehityksen kannalta. Toimeksiantajana toimii Digiosaajaksi työelämään -hanke. Hanketta on toteuttamassa Mikkelin Ammattikorkeakoulu, Itä-Suomen Yliopiston ja Työväen sivistysliitto. Tarkoituksena on suunnitella ja toteuttaa ensimmäinen prototyyppi kehitettävän työelämäpelin palvelutilanteita kuvaavasta osa-alueesta. Pelin tarkoituksena on toimia kursseilla käytettävänä opetustyökaluna, jossa erilaisten palvelutilanteiden ja niissä käytävien dialogien kautta opetellaan hyödyllisiä tieto- ja viestintäteknologiataitoja.

Käsittelen tässä opinnäytetyössä pelisuunnittelun ja ensimmäisen käytännön toteutuksen rakentamisen teoriaa sekä Unity3D-pelimoottorin soveltuvuutta ketterän kehityksen keinoihin. Käytännön osuus opinnäytetyön loppupuolella keskittyy pelin toteutukseen aiemmin kerätyn teoriatiedon tukemana.

2 PELISUUNNITTELU JA VERSIOHALLINTA

Käyn läpi suunnittelua, ensimmäistä tuotosta sekä jatkokehitystä versiohallinnan toteuttamisen näkökulmasta. Pelisuunnitteludokumentilla on suuri rooli vision kuljettamisessa johdonmukaisesti kehityksen eri vaiheiden läpi. Varsinkin suurissa kehitystiimeissä yhteistyö voi olla ajoittain vaikeaa ja myös omat tavoitteet voivat sumentua ke-

hityksen edetessä. Kehityksen alkuvaiheessa on tärkeää saada tuotos nopeasti ulos. Tällöin voidaan jo aikaisessa vaiheessa testata kantaako idea ja onko sillä minkäänlaista arvoa pelaajalle. Avaan käsitteitä kuten pelinsuunnitteludokumentti, MVP ja versiohallinta pelin kehityksen näkökulmasta.

2.1 Game design document – Pelinsuunnitteludokumentti

Pelin suunnittelu lähtee aina ideasta. Idea voi olla jokin mekaniikka, jonka ympärille peli rakentuu tai se voi olla määritelmä pelin teemasta tai suurpiirteisestä juonesta. Tärkeää on, että ideoita synnytetään mahdollisimman vapaasti eikä niitä tyrmätä suunnittelun tässä vaiheessa. Idea saattaa paperilla näyttää liian yksinkertaiselta tai epäloogiselta, mutta monet suuren suosioon nousseet pelisarjat ovat todistaneet toisin. Italialainen putkimies seikkailemassa sienivaltakunnassa prinsessaa pelastaen tai keltainen pisteitä syövä ja kummituksia pakeneva pallo eivät välttämättä ole paperilla niitä mielenkiintoisimpia pelin aiheita, mutta niin Pacmanista kuin Super Mariostakin on tullut yksiä peliteollisuuteen eniten vaikuttaneita pelisarjoja.

Pelin lajityypin eli genren valinnasta on hyvä lähteä liikkeelle, koska se luo heti mielikuvan pelin mekaniikoista ja rungosta, joiden avulla peliä voidaan alkaa luonnostella. Pelin kehityksessä on aina lainattu ideoita aiemmista peleistä ja se on yleensä hyvä lähestymistapa kehityksen alussa. Harvemmin idea on täysin uusi, mutta vanhojen esikuvien päälle voidaan rakentaa jotain täysin uutta, joka saattaa muuttaa täysin käsityksen kyseistä genrestä tai mekaniikasta. Nykypäivän tasohyppelypelit ovat edenneet kauas 80-luvun esikuvistaan, mutta niissä on silti yhä nähtävissä niiden jättämät jäljet. Omasta visiosta kannattaa kuitenkin pitää kiinni. Ilman uusia ideoita lajityypit tai peliala yleisesti eivät monipuolistu tai kehity eteenpäin. Suurilla budjeteilla tuotettujen pelien kehitys voi kestää useita vuosia ja työllistää jopa satoja ihmisiä, joten ajatus siitä mitä pelin tulisi olla niin graafiselta ulkoasultaan kuin pelillisiltä mekaniikoiltaan voi sumentua pitkän kehitystyön aikana. Tätä varten pelin suunnittelu- ja kehitystyöllä on hyvä olla tukena erilaisia työkaluja. Yksi tehokkaimmista suunnittelutyökaluista on niin sanottu Game design document.

Game design document eli suomalaisittain pelinsuunnitteludokumentti määrittää kaiken sen mitä suunnitteilla olevan pelin tulisi olla. Pelinsuunnitteludokumenttiin määritel-

lään pelin kannalta oleellisia osatekijöitä, kuten kohderyhmä, pelin tavoite ja päämekaniikat. Scott Rogers (2014) vertaa pelinsuunnitteludokumenttia chilin valmistukseen. Hyvän chilin kehittäminen vaatii reseptin, joka määrittää niin raaka-aineet, työtavat kuin tarvittavat työvälineet. Resepti ei pysy koko kehittelyvaiheen aikana samana vaan ainesosat ja niiden suhteet muuttuvat jatkuvasti parhaimman makuelämyksen saavuttamiseksi.

Pelinsuunnitteludokumentin pituus voi vaihdella yhden sivun pikakaaviosta aina monisatasivuiseen yksityiskohtaiseen dokumenttiin. Pituus kannattaakin valita projektin laajuuden mukaan. Dokumentin on oltava tarpeeksi yksinkertainen, jotta esimerkiksi kolmannet osapuolet saavat eheän kuvan pelin visiosta. Pelinsuunnitteludokumentti ei kuitenkaan saa olla liian suurpiirteinen, jotta pelin kehittämisessä mukana olevat tahot pystyvät työskentelemään samaan vision mukaisesti (Rogers 2014, 67). Dokumentin yksi tärkeimmistä tarkoitusperistä on siis se, että koko työyhteisö aina graafikoista ohjelmoijiin tietävät mihin suuntaan projektia ollaan viemässä. Riot Gamesin pelisuunnittelija Stone Librande tiivistääkin pelin vision niin, että se voidaan esittää esimerkiksi yhdellä mainosjulisteella. (Schell 2015, 428.)

PELINSUUNNITTELUDOKUMENTTI - YKSISIVUINEN DOKUMENTTI

Nimeke:	Digiosaajaksi Työelämään- työelämäpeli (Palvelutilanteet)
Julkaisu alustat:	Avoimenlähdekoodin tuotos, selain- ja työpöytäsovelluspeli
Kohderyhmä:	Yli 30- vuotiaat aikuiset, joilla heikot TVT- taidot.
Lajityyppi:	Hyötypeli opetustarkoitukseen.
Päämekaniikat:	Palvelutilanne osuudessa dialogipainotteisuus. Pelaajan valinnat
Työelämä- pelin juoni:	Pelaajan "matka" etenee Mikkelin kaupungissa erilaisissa palvelutilanteissa, joissa pelaaja suorittaa vielä määrittelemättömiä tavoitteta palvelutilanteesta riippuen
Palvelutilanteen juoni:	Pelaaja ottaa vuoronumeron ja omalla vuorollaan siirtyy tiskille. Tiskin dialogin läpäistyään pelaaja siirtyy toisessa tilassa olevaan pelin lopetus dialogiin. Viimeinen dialogi toimii palautteen keräämisenä pelaajilta. Pelaaja poistuu palvelutilanteesta takaisin kaupunkinäkymään.
Toissijaiset pelimekaniikat:	Erilaiset oppimistilanteiden rinnalle toteutettavat interaktiot.
Jatko-kehitys:	Ideoissa mm. moninpeli chat- toiminnalla, paljon erilaisia kurssille osallistuneiden kehittämiä palvelutilanteita. Jatko-kehitys jatkuu ainakin vuodelle 2017.

KUVA 1. Esimerkki yksisivuisesta pelinsuunnitteludokumentista

Pelinsuunnitteludokumentti ei ole työvälineenä kuitenkaan ongelmaton. Samalla hetkellä, kun dokumentin kirjoitus aloitetaan, on dokumentti jo käytännössä vanhentunutta tietoa. Tarkasti kirjoitettua dokumenttia olisi helppo katsoa kuin ohjekirjaa, jonka ohjeita noudattamalla peli voitaisiin kehittää valmiiksi, mutta todellisuudessa asia ei ole

yhtä mustavalkoinen. Pelinsuunnitteludokumentti toimii enemmänkin kokoelmana teorioita, joiden toimivuus todistetaan vasta käytännössä. Tämä puoli pelinsuunnittelusta voi aiheuttaa suuriakin ongelmia, jos suunnitelmista pidetään liian tiukasti kiinni tai uusia teorioita ja ideoita ei keksitä olemassa olevien lisäksi tai niiden tilalle. (Schell 2015, 426.)

Pelinsuunnitteludokumentilla yritetään helpottaa pelinkehitystä ja tavoitteen selkeyttämistä. Pelinsuunnitteludokumentti toimii ikään kuin suuren ihmisjoukon välisenä muistilappuna. Ihmisillä on luonnostaan rajoitettu muistikapasiteetti. Jos pelinkehityksessä yritetään vastata kymmeniin tai jopa satoihin kysymyksiin, on isonkin ihmisjoukon mahdotonta muistaa niitä kaikkia. Hyvä idea saattaa tuntua mahdottomalta unohtaa, mutta parin sadan työtunnin ja kymmenien uusien ongelmien jälkeen tuo hyväkin idea on luultavasti jo muuttunut, ellei täysin unohtunut mielestä. Jos ideat ja kysymykset on kirjattu ylös pelinsuunnitteludokumenttiin, ei samoihin kysymyksiin ja ongelmiin tarvitse keksiä vastausta useaan kertaan. (Schell 2015, 427.)

Toinen tärkeä osa-alue, jota pelinsuunnitteludokumentti yrittää helpottaa on kommunikointi työntekijöiden välillä. Jokaista kirjattua ideaa kohti tulee varmasti useita vastaväitteitä ja ehdotuksia, joten nopea tiedonvälitys työntekijöiden kesken on ehdotonta niin pienessä kuin isommassakin projektissa. Dokumentin saatavuus koko tiimille mahdollistaa sen, että kaikki pysyvät ajan tasalla ja muokkaaminen toimii saumattomasti uusien ideoiden myötä. Samaisesta syystä uskon esimerkiksi erilaisten pilvipalveluiden ja niissä käytettyjen työtilojen yleistyneen viime vuosina. (Schell 2015, 427.)

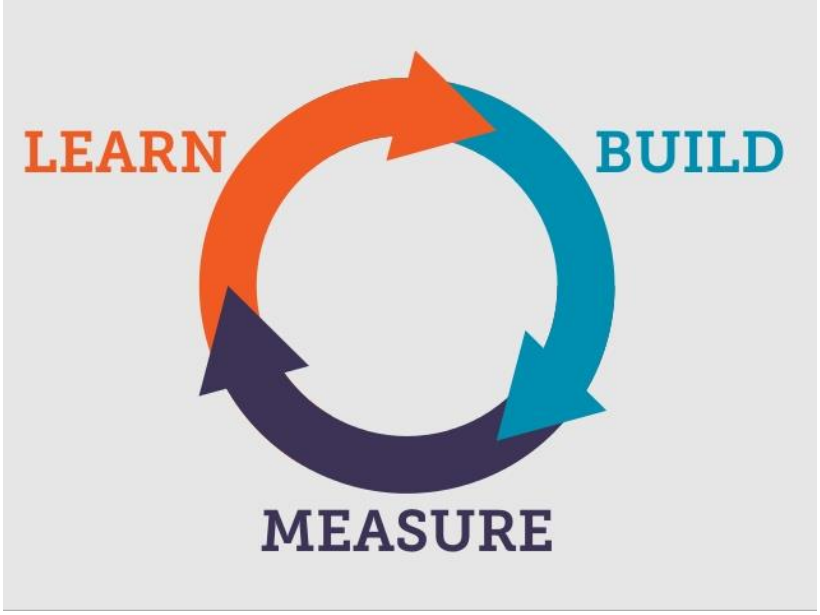
Pelisuunnitteludokumentin pohjalta voidaankin jo pelikehityksen alkuvaiheessa alkaa työstää ensimmäistä käytännön esimerkkituotetta, jonka tarkoitusperiin ja toteutukseen perehdyn seuraavaksi.

2.2 MVP - Minimum viable product

Termillä Minimum viable product eli lyhennettynä MVP:llä tarkoitetaan pienintä mahdollista toteutusta tuotteesta, jolla voidaan testata tuoteideaa erilaisilla ryhmillä. Kyseessä on eräänlainen prototyyppi, jolla voidaan esitellä ja testata tuotteen perusidean toimivuutta. Vaikka termi on alun perin lähtöisin nykyisen SyncDevin toimitusjohtaja Frank Robinsonin suusta (Minimum Viable Product, 2016) niin, termi minimum viable

product vakiinnutti asemansa Eric Riesin, Steve Blankin ja Alexander Osterwalderin ketterää Lean startup -ajatusta avaavien kirjojen myötä. Lean Startup perustuu ajatukseen, jossa tuotetta tai ideaa pitää lähteä testaamaan oikeiden asiakkaiden kanssa mahdollisimman aikaisin. (Setälä 2012) Vaikka malli on nykyisin hyvin yleinen kaikessa tuotekehityksessä, niin kyseinen malli on osoittautunut toimivaksi myös pelimaailmassa. Tulen tästä lähin viittaamaan termiin sen lyhennetyn muodon MVP:n kautta.

Pelisuunnitteludokumentti on loistava ja hyödyllinen tapa pelisuunnittelun alkuvaiheessa, koska kaikki tuoreet ideat saadaan kirjattua ylös ja visio pelin lopputuotteesta pysyy selkeänä. Kehityksen alkuvaiheessa on tärkeää, että pelin ydinmekaniikat ja tavoitteet olisivat toimivia. Tämä mahdollistaa sen, että niiden päälle voidaan lähteä rakentamaan lisää sisältöä. Usein ajatusten virta on voimakasta ja peliin ollaan lisäämässä suuria määriä ominaisuuksia jo ennen kuin peliä varten on kirjoitettu riviäkään koodia. MVP:n avulla kehitystiimi pääsee konkreettisesti kokeilemaan tuotteen perusideoita ja parhaassa tapauksessa myös testaamaan niitä ulkopuolisilla tahoilla. Pelin ydintä päästään hiomaan jo kehityksen ensimmäisinä viikkoina. Peliä konkreettisesti testaamalla huomataan ongelmia, joita ei pystytty paperilla havaitsemaan. Kehittäjät pääsevät näkemään mitkä pelin osa-alueet ovat kiinnostavia ja mitkä vain tuntuivat kiinnostavilta. Jotkin pelin mekaniikoista voivat yksinkertaisesti olla rikkinäisiä. Pelin tavoite tai eteneminen voivat myös olla puutteellisia. Testaamisen ja kehittämisen yhtäaikainen aloittaminen heti alussa voi parhaimmillaan säästää kymmeniä työtunteja sekä vahvistaa pelin visioita.



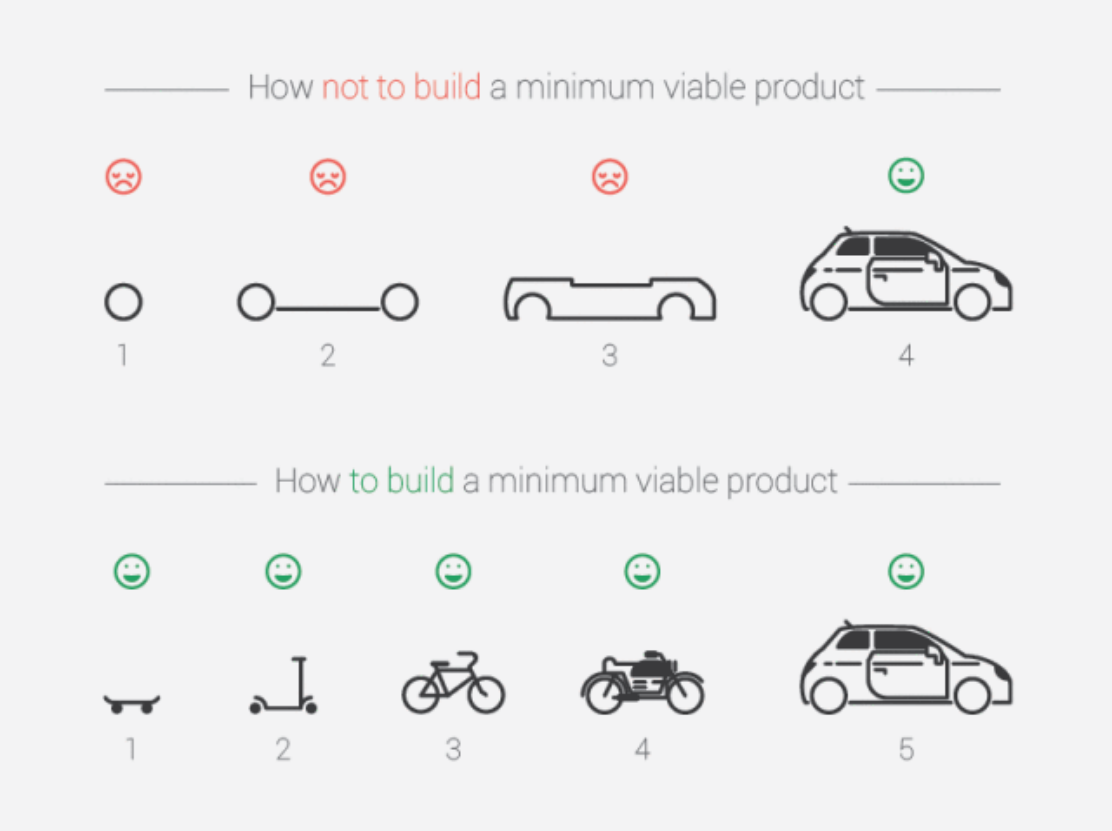
KUVA 2. Build, measure, learn -kaavio (Zaninotto 2016)

Kirjassaan The Lean Startup Eric Ries (2011) käyttää MVP:n kuvaamiseen Build-Measure-Learn kiertokaaviota, jonka yksi kiertokerta kuvaa sitä mikä MVP:n tulisi olla. Ideasta rakennetaan tuote, jolla mitataan jotain haluttua ominaisuutta. Saadulla tiedolla opitaan taas uutta ja uuden osaamisen pohjalta voidaan aloittaa uusi sykli kiertokaaviossa. Ensin rakennetaan prototyyppi, jonka avulla voidaan kerätä tarpeellista tietoa ja muokata sitä oikeaan suuntaan. (Ries 2011, 75-76.) Yksi vaikeimmista asioista MVP:tä luodessa on ominaisuuksien rajaaminen niin vähäiseksi kuin mahdollista, mutta kuitenkin saaden riittävästi tarpeellista tietoa tuotteen toimivuudesta. Ominaisuuksien rajaamisesta MVP:ssä voidaan havainnollistaa tasohyppelypelien avulla. Jos tasohyppelypelistä rajataan pois kaikki toissijainen mekaniikka, graafinen ulkoasu ja tarina, niin jäljelle jää hahmo ja sen erilaiset liikkumismuodot. Hahmo pystyy kävelemään, juoksemaan ja hyppimään. Kävely ja juoksu vaikuttavat hypyn korkeuteen ja sitä kautta pituuteen. Seuraavaksi tehdään ehdot pelaajan häviämiselle ja voittamiselle esimerkiksi lisäämällä peliin tasot, joilla hahmo voi liikkua sekä rotkoja, joihin tippua. Kentän loppuun voi asettaa maaliviivan, joka kertoo pelin onnistuneesta läpäisystä.

Jo näin vähäisellä interaktiolla pelin ja pelaajan välillä päästään kokeilemaan toimivatko pelin fysiikat ja mekaniikat toivotulla tavalla. Jos pelin kivijalka ei ole toimiva tai se ei herätä testaajassa mitään tunteita, on hyvin todennäköistä, että myöhemmin rakennetut elementit eivät palvele olemassa olevia elementtejä tai ne vaikuttavat päälle liimatuilta. MVP voi tietysti pelin tarkoituksesta riippuen sisältää paljon muutakin, mutta pääperiaatteena on testata ensimmäisenä pelin välttämättömät ydinmekaniikat. Yleinen tapa pelin MVP:lle on luoda pelille ensimmäinen taso tai eräänlainen hiekkalaatikko, jossa voidaan vapaasti testata olemassa olevia ominaisuuksia. (Extra Credits 2016.)

Pelillä täytyy olla jokin tapa mitata pelaajien palautetta jo pelin suunnitteluvaiheessa. Se voi tapahtua joko pelin sisällä tai esimerkiksi palautelomakkeilla tai -keskusteluilla. Jos pelissä olevat “kuolemat” voidaan liittää erilaisiin peliin asetettuihin rotkoihin esimerkiksi laskureilla, voidaan nähdä mitkä pelin esteistä olivat vaikeimpia. Peliin voidaan asettaa myös vaihtoehtoisia tapoja ylittää este ja siten mitata mitkä ongelmanratkaisutavat tulivat pelaajille luonnostaan. Edellä mainittu tieto on tärkeää, koska se miten pelin kehittäjät ratkaisevat pelissä ilmeneviä ongelmia ei aina mene yhteen pelaajien kanssa.

Tietoa keräämällä kehittäjät voivat oppia ja kehittää peliä eteenpäin. Edellä mainitun Build-Measure-Learn kierron suunta ja järjestys eivät kuitenkaan ole oleellisia vaan se, että kaikki kohdat toteutuvat kierron aikana. Usein kierron järjestys vaihtelee kehityksen eri vaiheista riippuen. Ensin voidaan päättää mistä osa-alueesta halutaan oppia, kuinka siitä voidaan kerätä hyödyllistä tietoa ja vasta sitten alkaa rakentaa seuraavaa versiota sen pohjalta. (Setälä 2012.)



KUVA 3. MVP:n rakentamisen hyviä ja huonoja tapoja kuvaava kaavio (Makabee 2015)

Kuva 3 esittää hyvin MVP:n idean. Tarkoituksena on, että jokaisesta kehityksen vaiheesta luotu prototyyppi on toimiva, mutta lopulliseen tuotokseen verrattuna yksinkertaistettu tai puutteellinen versio. Tarkoituksena on testata ennalta määrättyjä ominaisuuksia eikä toteuttaa rikkinäistä versiota lopullisesta pelistä. Pelimaailmassa MVP:n tärkeys on tullut viime vuosina hyvin esille erilaisten joukkorahoituskanavien kautta (esim. Kickstarter). Näillä kanavilla omalle pelille haetaan rahoitusta esittelyvideon, dokumentaation ja yleensä eräänlaisen MVP:n avulla. Tällaisella toimintaperiaatteella hyvin ideoidulla, minimaalisesti esitetyllä ja kehityksensä alkuvaiheessa olevalla tuotteella on hyvät mahdollisuudet kerätä rahaa tuotteen loppuun viemiselle. Ensimmäisen

MVP:n jälkeen pelin kehitystä voidaan alkaa syventää lisäämällä peliin uusia ominaisuuksia, haasteita ja variaatioita sekä muuttamalla sen graafista ulkoasua.

2.3 Versiohallinta

Versiohallinnalla tarkoitetaan sitä, kuinka kehityksen ajan eri vaiheita voidaan seurata eri versioiden kautta. Versiohallinta on tärkeä osa-alue mitä tahansa ohjelmistokehitystä, joten sama pätee myös pelien kehitykseen. Lopullinen tuotos on viimeinen versio kehitteillä olevasta tuotteesta ja jokaisella aiemmalla versiolla on ollut vaikutusta sen syntyyn. Aiemmin käsittelemäni MVP on yksi aikaisimmista versioista pelin kehityskaaressa, mutta versiohallinta on jo tuossa vaiheessa ollut suurella todennäköisyydellä käynnissä.

Versiohallinta voi yksinkertaisimmillaan tarkoittaa sitä, että aina uutta luotaessa kyseinen projekti tallennetaan erikseen, jolloin aiempi versio jää itsenäiseksi projektikseen. Tämä toteutuu esimerkiksi nimeämällä projektit numeroiden (projekti1, projekti2 jne.) tai päivämäärien mukaan (esimerkiksi projekti05touko2016). Tällä tavoin nykyistä ja edellistä versiota voidaan verrata ja kehityksen suuntaa on helpompi arvioida. Aiempiin versioihin voidaan myös palata, jos halutaan kokeilla vaihtoehtoista toteutustapaa tai seurata onko joku myöhemmin kohdattu ongelma ollut läsnä jo aiemmassa kehitysvaiheessa. (A Visual Guide To Version Control 2007.)

Pienissä kehitystiimeissä versiohallinta voidaan toteuttaa monella tavalla ilman erillisiä versiohallintajärjestelmiä (VCS, Version Control System). Projektia voidaan kopioida eri kehitystiimin jäsenille esimerkiksi muistitikun avulla tai käyttää erilaisia pilvipalveluja projektin kuljettamiseen. Monet ilmaista tallennustilaa sisältävät pilvipalvelut kuten Dropbox ja Google Drive sisältävät riittävästi tilaa pienille peliprojekteille, mutta informaation kulku kehityksen eri osa-alueista pitää hoitaa erikseen esimerkiksi sähköpostitse tai erillisellä tekstitiedostolla. Menetelmä toimii hyvin, jos tiimi on pieni ja kommunikaatiota voidaan pitää helposti yllä. Tällainen versiohallinta on kuitenkin ongelmallista, jos kehitystiimin tekemä työ kohdistuu toisiinsa vaikuttaviin osa-alueisiin tai kehitystiimin koko on suuri. Tiedostojen säilyttämiseen kehitetyt pilvipalvelut eivät pysty ilmoittamaan tehtyjen töiden keskenään synnyttämistä konflikteista eivätkä ne pidä automaattisesti kirjaa kehityksen eri versioista. (Ash 2016.)

Versiohallintaa varten on kehitetty useita versiohallintajärjestelmiä, jotka tarjoavat monia hyödyllisiä ominaisuuksia tukemaan versiohallintaa varsinkin, jos kehitystiimi koostuu useammasta henkilöstä. Erilaisia ilmaisia ja maksullisia versiohallintajärjestelmiä on paljon, mutta kerron seuraavaksi niille kaikille yhteisistä ominaisuuksia. Järjestelmät pitävät huolen eri versioiden tallentamisesta ja mahdollistavat aiempiin versioihin palaamisen vaivattomasti. Tällä tavoin projektista on myös palvelimella varmuuskopio niin viimeisimmästä kuin aiemmistakin versioista. Versioiden pysyessä ajan tasalla, jokainen kehitystiimin jäsen pääsee käsiksi uusimpana versioon ja näin kaikki tietävät missä vaiheessa eri osa-alueiden kehitys on. Tiimin jäsenillä on mahdollisuus kirjoittaa järjestelmään viesti aina uutta versiota lisätessä, joten kommunikaatio toimii sujuvasti eikä uusia tai korjattuja ominaisuuksia tarvitse itse lähteä etsimään tai kyselemään. (Ash 2016.)

Versiohallintajärjestelmät osaavat myös tulkita tiedostojen muutoksia, jos samaa tiedostoa on muokattu useamman henkilön toimesta. Kahden samaa ohjelmointiskriptiä työstäneen ohjelmoijan koodit yhdistyvät automaattisesti samaan tiedostoon. Samaan tiedostoon tehdyt erilliset muokkaukset voivat kuitenkin aiheuttaa konflikteja tiedoston sisällä. Tällaiset tilanteet voivat olla hyvin ongelmallisia, etenkin isoissa kehitystiimeissä, mutta versiohallintajärjestelmän ansioista ongelmasta ollaan kuitenkin tietoisia, sillä järjestelmä pystyy tunnistamaan syntyneet konfliktit. (A Visual Guide To Version Control 2007.)

Versiohallinta pelien kehityksessä tuo kuitenkin mukanaan tiettyjä ongelmia esimerkiksi verrattuna perinteiseen ohjelmistokehitykseen. Pelin kehityksessä käytössä on useita erilaisia tiedostomuotoja kehityksen eri osa-alueilta. Erilaisia tiedostomuotoja ovat muun muassa 3D-malleja, tekstuureja, äänitiedostoja, kuvia ja ohjelmointiskriptejä. Tiedostokoot voivat kasvaa hyvin suuriksi, mikä vaikeuttaa joidenkin varsinkin ilmaiseksi käytettävien versiohallintajärjestelmien käyttöä. Esimerkiksi ohjelmointiprojekteissa suosittu GitHub soveltuu hyvin ohjelmointiskriptien ylläpitoon, mutta suurempien tiedoston hallinta ei kokorajoitusten takia onnistu. Kyseiseen ongelmaan on suuremmissa pelitaloissa keksitty ratkaisuja, mutta ilmaisten työkalujen varassa olevien pienten kehitystiimien on keksittävä keinoja käsitellä suuria tiedostoja.
Pienissä tiimeissä toimiva ratkaisu voi olla versiohallintajärjestelmän käyttö pienille tiedostoille ja pilvipalvelussa olevan tallennustilan käyttö suuremmille. Ohjelmoin-

tiskriptit saattavat muuttua useaankin otteeseen kehityksen aikana, kun uusia ominaisuuksia syntyy tai vanhoja parannellaan. Lopullisten 3D-mallien sijaan voidaankin pitkälle kehityksen aikana käyttää erilaisia placeholdereita, joten 3D-mallien hallinta versiohallintajärjestelmän avulla ei ole mielestäni yhtä tarpeellista kuin esimerkiksi ohjelmointiskriptien kanssa. (Ash 2016.)

Myöhemmin käsiteltävä Unity3D-pelimoottorikin tarjoaa käyttäjilleen versiohallintatyökaluja, mutta niistä tehokkaimmat ovat käytössä vain maksullisissa lisenssiversioissa. Unity3D:llä on kuitenkin mahdollista pakata erilaisia objekteja niin sanottuihin paketteihin (Package), mutta niiden siirtäminen vaatii jonkin ulkopuolisen palvelun käyttöä.

3 UNITY3D JA CRADLE

Unity3D on Unity Technologiesin vuonna 2005 julkaistu pelimoottori. Unity3D:stä on tullut vuosien varrella yksi käytetyimmistä ilmaisversion omaavista pelimoottoreista. Kirjoitushetkellä rekisteröityneitä Unityn käyttäjiä on 5,5 miljoonaa ja tuotettujen pelien skaala on laaja. Varsinkin mobiilipelimarkkoilla Unity3D on todella suosittu ja vuoden 2016 ensimmäisellä vuosineljänneksellä 34 % tuhannesta eniten ladatusta ilmaisesta mobiilipelistä oli kehitetty Unity3D:llä. (Fast Facts 2016.)

Unity3D:stä on tarjolla erilaisia lisenssejä, joista yksi on täysin ilmainen. Ilmaisen lisenssin myötä sen käyttö varsinkin Indie-kehittäjien parissa on kasvanut suuresti, koska ilmaisversiolla tuotettuja pelejä voidaan käyttää kaupallisissa tuotteissa, kunhan vuosittainen liikevaihto ei ylitä sadantuhannen dollarin rajaa. (Unity Personal 2016.) Unity3D:n yksi suuri vahvuus on sen tuetut alustavaihtoehdot. Unity3D tukee kaikkia käytetyimpiä pelaamiseen tarkoitettuja alustoja kuten työpöytä- ja selainsovelluksia, älytelevisioita sekä mobiililaitteita. (Fast Facts 2016.) Vaikka Unity3D onkin suosittu Indie-kehittäjien parissa, niin Unity3D:llä tuotettuja suuren budjetin pelejä eli niin sanottuja AAA-pelejäkin on ilmestynyt markkinoille runsaasti viime vuosina. Hyvinä esimerkkeinä Unity3D:llä tuotetuista peleistä ovat mm. suomalaisen pelitalon Colossal Orderin kaupungin rakennuspeli Cities Skylines ja Square Enix Montrealin mobiilipeli

Lara Croft GO. (Made With Unity 2016.) Unity on myös vahvasti mukana pelien kehityksen uusimmissa innovaatioissa kuten virtuaalitodellisuudessa. Noin 53% Oculus VR:n Oculus Riftin peleistä on kehitetty Unity3D:llä. (Fast Facts 2016.)

Seuraavissa alaluvuissa tulen käsittelemään Unity3D:tä ketterän kehityksen näkökulmasta. Tarkoituksena ei ole syväluotaavasti käsitellä sitä mihin Unity3D:llä pystytään tai kohdentaa sen toimivuutta mihinkään tiettyyn pelityyppiin. Tarkoituksena on esitellä peruselementit, joilla luoda nopeasti pelin ensimmäiset raamit. Esittelen myös opinnäytetyöni käytännönosuudessa suuressa roolissa olevaa Twine-tarinatyökalua sekä sen ja Unity3D:n välille kehitettyä Cradle (aiemmin nimellä UnityTwine) GitHub-projektia. Aihealueen lopussa kokoan tiettyjä Unity3D:ssä käytettäviä kommunikaation ja johdonmukaisuuden parantamiseen vaikuttavia ominaisuuksia.

3.1 Unity ja ketterän kehityksen menetelmät

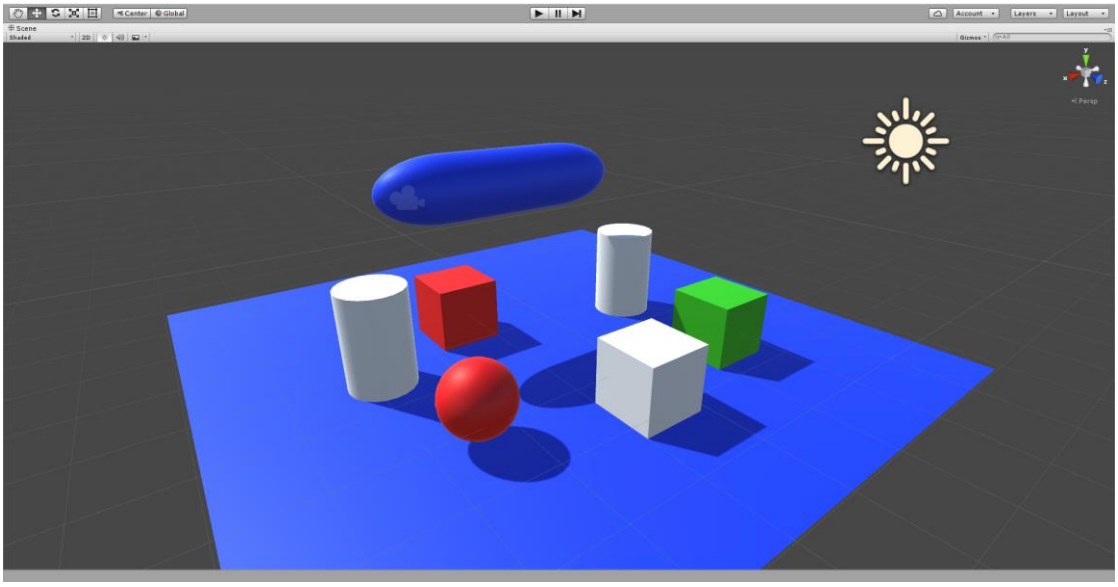
Pelin kehityksen aloittaminen Unity3D:ssä on tehty hyvin käyttäjäystävälliseksi. Selkeät editorin ikkunanäkymät ja laaja kirjo erilaisia sisäänrakennettuja ominaisuuksia helpottavat nopeaa sisällön tuottamista jo ennen ohjelmoinnin alkamista. Käyn tässä osiossa läpi Unity3D:n käyttöä nopeassa sisällön tuotannossa ensimmäistä MVP:tä ajatellen. Alaluvut etenevät pelimaailman luomisesta, toiminnallisuuden toteuttamisen kautta käyttöliittymään rakentamiseen. Lopuksi kerron pintapuolisesti ongelmanratkaisusta sekä kerään tietoperustaa sessionhallinnan ja ryhmätyön tehostamiselle.

3.1.1 Peliobjektit ja asset-tiedostot

Unity3D:ssä rakennetaan erilaisia Scenejä (Scenes 2016), jotka sisältävät kaikki sinne lisätyt peliobjektit. Scenet tallennetaan Scene-tyyppisiksi tiedostoiksi ja monien Scene-tiedostojen yhdistämisestä syntyy lopullinen pelin kokonaisuus. Scenen voidaan ajatella olevan esimerkiksi yksi pelin kenttä.

Asset (Asset Workflow 2016) tarkoittaa mitä vain Unity3D:n tukemaa tiedostotyyppiä, jota voidaan käyttää pelin kehityksessä. Assetit voivat olla esimerkiksi kuvatiedostoja, ohjelmointiskriptejä, ääntä, fontteja tai 3D- malleja. Assetteja voidaan tuoda projektiin suoraan, jolloin ne tallentuvat Asset-nimisen kansion sisään. Tämän kansion sisään voi-

daan luoda uusia kansiota ja onkin suositeltavaa jakaa erilaiset Assetit omiin kansioihinsa johdonmukaisuuden ylläpitämiseksi. Assetit eivät vielä itsessään ole peliobjekteja vaan ne ovat komponentteja, joista peliobjektit voidaan koostaa.



KUVA 4. Erilaisia Unity3D:n primitiivisiä peliobjekteja Scene-näkymässä. Osalle annettu materiaali-komponentin avulla oletusharmaasta poikkeava väri.

Ulkopuolisilla ohjelmilla tuotettujen 3D-mallien lisäksi Unity3D:ssä on sisään rakennettuna kokoelma erilaisia primitiivisiä objekteja, jotka voidaan luoda suoraan editorin kautta. Peliobjekti itsessään ei määrittele kuin oman sijaintinsa ja kokonsa suhteessa pelimaailmaan. Peliobjekti toimii eräänlaisena tyhjänä muottina, jonka sisään voidaan luoda toiminnallisuutta ja ulkoasua käyttämällä erilaisia Assetteja. Tällaisia primitiivisiä peliobjekteja Unity3D:ssä on mm. kuutio (cube), sylinteri (cylinder), pallo (sphere) ja taso (plane). Edellä mainittuja peliobjekteja voidaan käyttää esimerkiksi placeholderina ennen kuin ne korvataan lopullisilla 3D-malleilla tai materiaaleilla. Esimerkiksi taso-objekti toimii hyvin maanpintana ja kuutio vaikka pelihahmon korvikkeena. Tällä tavalla päästään mahdollisimman nopeasti rakentamaan pelille toiminnallisuutta ja sisältöä käyttäen minimaalisia resursseja pelin graafiseen ulkoasuun. (Primitive and Placeholder Objects 2016.)

Erilaisten Assettien käyttö palvelee myös hyvin ominaisuuksien ja elementtien uudelleen käyttöä. Luoduille peliobjekteille voidaan lisätä erilaisia komponentteja (Component, 2016). Komponentit voivat olla mitä tahansa Assetteja tai Unity3D:n sisäänra-

kennettuja komponentteja. Komponentit sisältävät erilaisia arvoja, joilla niiden ominaisuuksia voidaan säätää. Peliobjektille voidaan myös lisätä toinen objekti olemassa olevan peliobjektin sisälle. Tällöin peliobjekti muuttuu hierarkkiseen muotoon ja lisätystä peliobjektista tulee alkuperäisen objektin lapsiobjekti. Tällä tavoin esimerkiksi laajan 3D-mallin erilaisia osia voidaan hallita paremmin ja niiden muokkaus ei välttämättä vaadi ulkopuolisen ohjelman käyttöä.

Pelien kehityksessä tulee hetkiä, jolloin luotua peliobjektia tullaan käyttämään muuttumattomana useita kertoja samassa scenessä. Tätä varten Unityssa on käytössä mahdollisuus luoda tehdyistä peliobjekteista niin sanottuja Prefab-objekteja. Prefab eli valmisobjekti luodaan Asset-kansioon olemassa olevien Assettien rinnalle, jotta sitä voitaisiin uusiokäyttää tarvittaessa luomatta aina samaa peliobjektia uudelleen ja uudelleen. Peliobjektia voidaan kyllä vaivattomasti kopioida samaan pelinäkymään useaan otteeseen luomatta siitä erillistä valmisobjektia. Ongelmana on kuitenkin se, että kopioimalla jokainen peliobjekti säilyy itsenäisenä ja niille tehdyt muutokset eivät koske kaikkia identtisiä peliobjekteja. Valmisobjektin kautta peliobjektia voidaan luoda haluttu määrä pelinäkymään niin, että muutokset koskevat kaikkia valmisobjektista luotuja peliobjekteja. (Prefabs 2016.)

Unity3D:n yksi vahvuuksista on sen aktiivinen ja laaja yhteisö. Unity3D:n yhteisöstä löytyy keskustelufoorumi, jossa on palsta erilaisille kehitysongelmille ja niille löytyneille ratkaisuille. Ilmaisia Assetteja löytyy netistä runsaasti ja Unity3D:llä on myös oma kauppapaikka Asset store, josta löytyy paljon niin maksullisia kuin ilmaisiakin Assetteja. Asset storen käyttö varsinkin projektin alkuvaiheessa voi olla todella hyvä idea, koska esimerkiksi pelin perusmekaniikkoihin löytyy paljon valmiita kontrolleja liikkumista varten. Omat ohjelmointiskriptit mekaniikoille voidaan luoda myöhemmin, mutta mahdollisimman varhaisen testauksen vuoksi on hyvä, että valmiita rakennuspalikoita löytyy jo heti alkuvaiheessa. Asset storesta ladatut tuotteet tulevat editoriin pakettimuodossa (Package). Paketit ovat kokoelma Assetteja, jotka on pakattu yhteen tiedostoon. Paketin sisällä olevat tiedostot säilyttävät niille määrätyn kansiorakenteen sekä erilaiset määritellyt metatiedot. (Asset Packages 2016.)

3.1.2 Interaktiot ja ohjelmointi

Erilaisten peliobjektien avulla pelimaailmaa saadaan täytettyä, mutta tässä vaiheessa se on vielä hyvin eloton. Peliin on lisättävä interaktioita pelaajan ja pelin välille. Erilaisilla ohjelmointiskripti-komponenteilla pelille luodaan toiminnallisuutta ja logiikkaa. Ohjelmointiskriptit voivat toimia automaattisesti taustalla tai niitä voidaan kutsua pelaajan ja pelin välillä tapahtuvilla interaktioilla. Ohjelmointiskripti luodaan muiden Assettien tapaan Asset-kansiohierarkiaan ja sieltä se voidaan kiinnittää komponenttina erilaisiin peliobjekteihin.

Unity3D tukee natiivisti kahta erilaista ohjelmointikieltä. Tässä opinnäytetyössä käytän ohjelmointiin C#-ohjelmointikieltä. Unity3D tukee myös pelimoottoria varten kehitettyä Unityscript-ohjelmointikieltä, joka on muokattua Javascriptiä (Creating and Using Scripts, 2016). Unity3D:n asennuspaketti sisältää Unity3D:n integroidun kehitysympäristön MonoDevelopin, joka toimii oletuksena ohjelmoitiskriptien luomisessa. Käyttäjää ei kuitenkaan sidota MonoDevelopin käyttöön vaan vastaavia muita kehitysympäristöjä voidaan käyttää ja osa niistä onkin nykyään jo hyvin tuettuja Unity3D:n ominaisuuksien kanssa.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class kokeilu : MonoBehaviour {
5
6
7      // Use this for initialization
8      void Start () {
9
10     }
11
12     // Update is called once per frame
13     void Update () {
14         |
15     }
16 }
17
```

KUVA 5. MonoDevelopin luoma valmispohja ohjelmointiskriptille

Unity3D luo uudelle ohjelmointiskriptille automaattisesti pohjan, joka sisältää Unity3D:n sisäisiä toimintoja mm. sitä varten, että komponentti voidaan lisätä myöhemmin peliobjektiin. Uuteen ohjelmointiskriptiin luodaan myös kaksi valmista tyhjää metodia. Start-metodi on tarkoitettu erilaisia alustuksia varten ja sitä kutsutaan heti sen

omaavan peliobjektin synnyttyä. Update-metodia taas kutsutaan jokaisen framen välein tasaisin väliajoin. Update-metodi on hyvä tekemään seurantaa esimerkiksi pelaajan ja pelin välisistä interaktioista kuten esimerkiksi hahmon liikkumisesta. Samoin kuin muut Assetit, niin myös ohjelmointiskriptit eivät vaikuta pelimaailmaan ennen kuin ne on liitetty komponenttina johonkin peliobjektiin. (Creating and Using Scripts 2016.)

Komponentin arvoja voidaan muuttaa suoraan editorin kautta, jos ne vain on alustettu julkisiksi (Public) muuttujiksi. Tällä tavoin muuttujien arvoja voidaan säätää myös pelin ollessa käynnissä, mikä helpottaa eri arvojen testausta ilman, että peliä tarvitsee erikseen pysäyttää (Variables and the Inspector, 2016). Unity3D:ssä eri peliobjekteihin ja komponentteihin viittaaminen on tehty vaivattomaksi ohjelmointiskripteissä. Peliobjektien sisäinen hierarkia komponentteineen on esitetty johdonmukaisesti editorissa ja sama johdonmukaisuus näkyy myös ohjelmoinnin syntaksissa. Ohjelmointiskripti voi näiden viittauksien avulla muuttaa peliobjektin ja sen lapsiobjektien arvoja lennossa erilaisten interaktioiden kautta.

Peliobjektien ja ohjelmointiskriptien toiminnallisuuden välille on kuitenkin saatava jonkinlainen interaktio pelaajan ja pelin välillä. Peleissä on paljon interaktioita erilaisten peliobjektien välillä ja niiden aikaan saamat tapahtumat tekevät peleistä mielenkiintoisen. Interaktio voi olla jotain mitä pelaaja tekee itse, esimerkiksi näppäimistön välityksellä, se voi olla jotain pelihahmon ja sitä ympäröivän pelimaailman välillä tai jotain mitä tapahtuu vain peliobjektin ja sen arvojen kanssa.

Unity3D:ssä on sisäänrakennettuna fysiikkamoottorit niin kaksi- kuin kolmeulotteiselle pelimaailmalle. Vasta luodulla peliobjektilla ei kuitenkaan ole oletuksena mitään komponenttia, joka saisi sen reagoimaan fysiikkamoottorin kanssa. Pääkomponentti peliobjektin ja fysiikkamoottorin välillä on Rigidbody-komponentti. Rigidbodyn avulla peliobjekti tottelee fysiikkamoottorin luomia fysiikanlakeja. Ilman Rigidbodya erilaisia peliobjekteja voidaan liikuttaa ohjelmointiskriptien kautta suoraan muuttamalla niiden sijaintia ja rotaatiota suhteessa pelimaailmaan ja siten luoda toimivaa illuusiota esimerkiksi painovoimasta. Rigidbodyn avulla peliobjekteille voidaan fysiikkamoottorin avulla luoda kuitenkin realistisempia ja vaativampia interaktioita, kuten fysiikkaa hyödyntäviä törmäyksiä ja luonnollisemman oloista liikettä.

Peliobjektia ei Rigidbodyn liittämisen jälkeen liikuteta enää suoraan muuttamalla sen sijaintia pelimaailmassa vaan sitä voidaan liikuttaa kohdentamalla siihen liike-energiaa (Force), jonka laskennan fysiikkamoottori hoitaa pelaajan puolesta. Esimerkiksi pallon heittämisen simuloiminen onnistuu suoraan määrittämällä heittokulman, painovoiman vaikutuksen, pallon massan ja kohdistetun voiman määrän. Saman toiminnan toteuttaminen pelkän sijainnin ja rotaation muuttamisella vaatisi monimutkaisia laskutoimituksia ja pitkän määrän ohjelmointikoodia. Rigibodyn antamat fysiikkamoottorin vaikutukset voidaan kuitenkin myös ottaa pois vaihtamalla isKinematic-arvoa, jolloin fysiikkamoottori ei enää vaikuta peliobjektiin. Tällöin peliobjekti voi kuitenkin olla vuorovaikutuksessa muiden Rigidbodyn omaavien peliobjektien kanssa, joiden isKinematic-arvo ei ole käytössä. (Rigidbody Overview 2016.)

Yksi yleisimmistä tavoista, jolla interaktio synnytetään pelissä, on niin sanotut törmäykset (Collision). Törmäykset synnyttävät usein jonkinlaisen tapahtuman, joka vaikuttaa olennaisesti peliin ja niiden synnyttäminen tai niiltä välttyminen ovat usein iso osa pelin logiikkaa. Rigidbody mahdollisti peliobjektien reagoimisen fysiikkamoottorin kanssa, mutta peliobjektien väliset törmäykset vaativat erillisen Collider-komponentin. Collider-komponentti on itsessään näkymätön ja irrallinen oma objektinsa peliobjektin sisässä. Colliderin ei siis tarvitse olla samankokoinen kuin sen vanhempana toimivan peliobjektin. Sen tehtävä on vain havaita siihen kohdistuvat törmäykset. Colliderin omaavat peliobjektit voivat reagoida törmäyksessä fysiikkamoottorin mukaan ja niiden massat sekä liikenopeudet vaikuttavat silloin toisiinsa. Esimerkiksi autopeleissä autojen välinen törmäily halutaan usein toteuttaa niin, että peliobjektit reagoivat realistisesti toisiinsa fysiikan lakien mukaan. (Colliders 2016.)

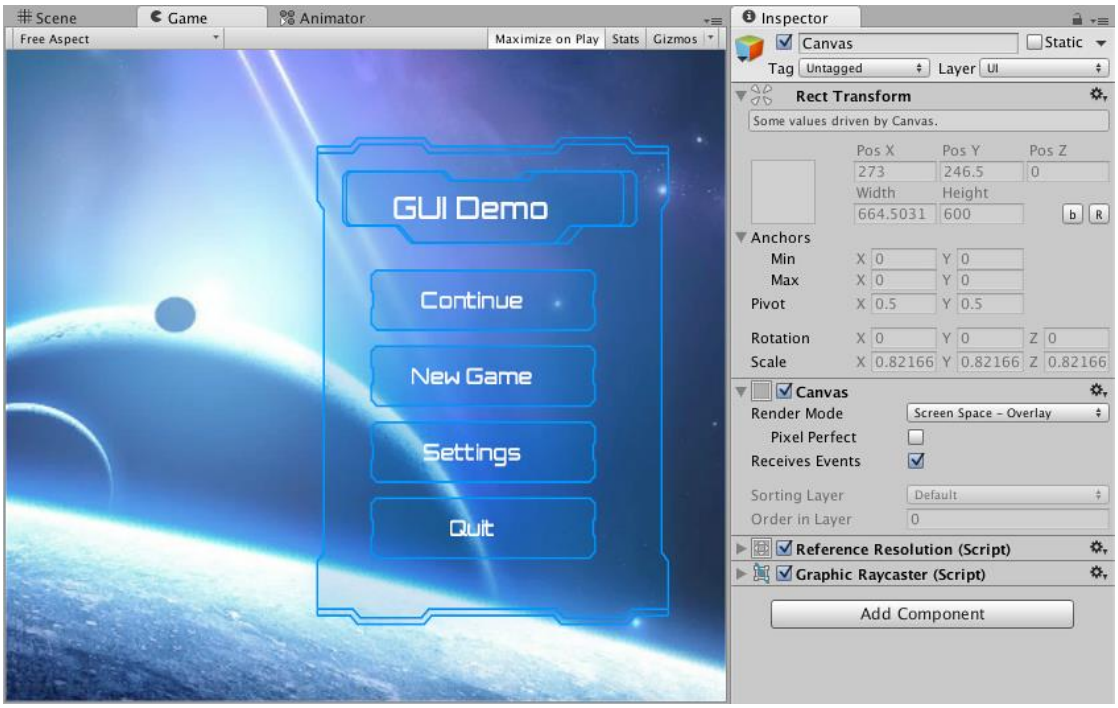
```
void OnTriggerEnter2D (Collider2D other)
{
    //Luodin törmäystunnistukset eri tag- ryhmien kanssa
    //Jos luoti osuu viholliseen
    if (other.gameObject.tag == "vihollinen") {

    //Tuhoa vihollispeliobjekti sekä luotipeliobjekti
        Destroy(other.gameObject);
        Destroy (this.gameObject);



    }

    //Jos luoti osuu tuhoutuvaan peliobjektiin esim. räjähdetynnyriin
    if (other.gameObject.tag == "rajahtava") {

    //Tuhoa räjähtäväpeliobjekti sekä luotipeliobjekti. Synnytä räjähdyskuva/animaatio
        Destroy(this.gameObject);
        GameObject tnt;
        tnt = Instantiate (explosionSprite, other.transform.position, other.transform.rotation) as GameObject;
        Destroy(tnt.gameObject,1);
    }

    //Jos luoti osuu seinäpeliobjektiin. Tuhoa pelkkä luotipeliobjekti
    if (other.gameObject.tag == "seina") {

        Destroy(this.gameObject);

    }
```

KUVA 6. Kuvakaappaus törmäyksentunnistuksesta

On kuitenkin tilanteita, joissa fysiikan vaikutuksien ei haluta vaikuttavan törmäykseen, vaan törmäyksen halutaan laukaisevan jokin tietty tilanne pelimaailmassa tai pelihahmossa. Esimerkiksi pelihahmon ja ensiapulaukun välinen törmäys on toteutettu kuvaamaan ensiapulaukun poimimista eikä kirjaimellista yhteentörmäystä. Tällaisia törmäysinteraktioita kutsutaan yleisesti Triggereiksi. Valitun Collider-komponentin voi muuttaa Triggeriksi, jolloin sen törmääminen toiseen peliobjektiin huomataan, mutta se ei automaattisesti synnytä minkäänlaista tapahtumaa. Esimerkiksi pelaajan ampuma luoti häviää osuessaan viholliseen ja vähentää vihollishahmolle määritettyjen terveyspisteiden määrä ilman, että luoti itsessään aiheuttaisi minkäänlaista törmäystä vihollishahmoon ja siten vaikuttaisi sen liike-energiaan tai fyysiseen olemukseen. Tämä mahdollistaa sen, että törmäyksiä tunnistamalla voidaan laukausta tilanteita pelimaailmassa. Triggerien laukaisemat toiminnallisuudet määritellään ohjelmointikripteissä onTriggerEnter-metodeilla, joiden sisään määritellään haluttu tapahtuma törmäyksen sattuessa. Esimerkiksi NPC-hahmon (non playable character) edessä voi olla näkymätön Collider-komponentti, johon törmäämällä pelihahmo laukaisee dialogi-ikkunan ilmestymisen pelaajan ruudulle. (Colliders 2016.)

3.1.3 Graafinen käyttöliittymä

Tässä vaiheessa pelimaailmassa on jo paljon erilaisia peliobjekteja, joille on luotu runsaasti toiminnallisuutta. Interaktioiden vahvistamiseksi peleihin tehdään erilaisia graafisia käyttöliittymiä. Kerättyjen pisteiden näkeminen pelin käyttöliittymässä vahvistaa pelaajan tavoitteellisuutta ja pelihahmon terveyden näkeminen vaikuttaa pelaajan ratkaisuihin pelin edetessä. Käyttöliittymillä myös neuvotaan pelaaja erilaisten vinkkien ja valikoiden avulla. Unity3D:n käyttäjäyhteisö toivoi pitkään uusia työkaluja käyttöliittymien rakentamiseen ja tammikuussa 2015 Unity3D:n 4.6 versio tarjosi uudistetun UI-järjestelmän (User Interface), joka helpotti käyttöliittymien luomista (Unity 4.6 2015).



KUVA 7. Unity3D:llä luotu graafinen käyttöliittymä

Unity3D:n UI-järjestelmän tarkoitus on antaa kehittäjälle työkalut luoda nopeasti erilaisia käyttöliittymiä. Käyttöliittymän kehittäminen Unity3D:ssä aloitetaan luomalla Canvas-elementti, johon kaikki myöhemmin luodut UI-elementit sijoitetaan. Canvas on peliobjekti siinä missä muutkin aiemmin mainitut elementit. Siihen on liitetty Canvas-komponentti, joka antaa erilaisia mahdollisuuksia muokata sen arvoja. Sen sisälle luodut erilaiset UI-elementit tulevat hierarkiaan Canvaksen lapsiobjekteiksi. Canvaksen sisälle luotu hierarkia vastaa sitä miltä itse käyttöliittymä tulee näyttämään. Hierarkian

pohjimmaisin peliobjekti näkyy päällimmäisenä elementtinä peittäen vanhemmat peliobjektit alleen. Canvas-komponentti pitää sisällään vain pari erilaista säätömahdollisuutta, mutta ne ovat sitäkin oleellisempia käyttöliittymän kannalta.

Komponentista valitaan yksi kolmesta Render modesta halutun lopputuloksen saavuttamiseksi. Render modeja ovat Screen Space-Overlay, Screen Space-Camera ja Word Space. Overlay ja Camera ovat toimintaperiaatteiltaan hyvin samanlaisia. Overlay-moodissa Canvas pysyy sijoitettuna Scene-näkymän yläreunaan ja se skaalaa itseään ruudun koon muutosten mukaisesti. Tässä moodissa kamera ja muut peliobjektit eivät vaikuta Canvaksen käyttäytymiseen. Tämä on hyvin yleinen tapa toteuttaa perinteinen käyttöliittymä, jossa Canvas on kuin piirtoheitinkalvo television ruudulla. Se pysyy muuttumattomana riippumatta pelin sisällöstä.

Camera-moodissa Canvas skaalautuu ruudun koon mukaan aivan kuin Overlay-moodissakin, mutta kamera vaikuttaa vahvasti Canvaksen käyttäytymiseen. Camera-moodissa valitaan kamera, jonka rotaation ja etäisyyden halutaan vaikuttavan Canvakseen. Kameran liike ja kulma saavat Canvaksen liikkumaan ja tällä tavoin siitä tulee dynaamisempi elementti. Näin voidaan luoda elävyyttä käyttöliittymään tekemällä siitä vähemmän staattinen. Kolmantena moodina Render mode -valikoimassa on World Space. World Space -moodissa Canvas käyttäytyy samoin kuin mikä tahansa muukin peliobjekti pelimaailmassa. Sen koko voidaan määrittää vapaasti ja se sijoittuu muiden peliobjektien eteen tai taakse riippuen kameran sijainnista. (Canvas 2016.)

World Space -moodissa voidaan luoda pieniä käyttöliittymiä pelimaailman sisään ja tällaista käyttöliittymää kutsutaan diegeettiseksi käyttöliittymäksi (Diegetic Interface). Diegeettisellä käyttöliittymällä tarkoitetaan käyttöliittymää, joka on osa jotain konkreettista, eikä vain pelaajaa varten luotua näkymätöntä esitystapaa. Hyviä esimerkkejä tällaisista käyttöliittymäratkaisuista ovat autopeleissä auton sisäpuolella kuvatut ajoneuvon omat nopeusmittarit, konkreettiset kompassit tai kartat, joita pelihahmo voi katsella ja scifi-peleissä usein käytetyt hologramminäytöt. Diegeettisillä käyttöliittymillä luodaan vahvaa immersiota pelimaailmaan staattisen pelivalikon tai HUD-systeemin (Head-up Display) sijaan. (Creating immersive experiences with diagetic interfaces 2010.)

KUVA 8. Diegeettinen käyttöliittymä Far Cry 2 -pelissä (Creating immersive experiences with diagetic interfaces 2010)

Diegeettiset käyttöliittymät ovat nousseet yhä merkittävämpään rooliin virtuaalitodellisuuden myötä. Perinteisemmät, ei diegeettiset, käyttöliittymät eivät sovellu VR-käyttöön (Virtual Reality), koska ihmisen silmien on mahdotonta pystyä tarkentamaan liian lähellä silmiä olevaan Overlay-tyyliseen näkymään. Pelimaailmassa näkyvien erilaisten informaatioiden tarkastelu virtuaalitodellisuudessa on paljon käyttäjäystävällisempää, koska liian lähellä silmiä olevat elementit voivat aiheuttaa pelaajalle huonovointisuutta. (User Interfaces for VR 2016.)

Canvaksen täyttö erilaisilla UI-elementeillä on tehty Unity3D:ssä helpoksi, sillä valmiita elementtejä kuten nappeja, liukusäätimiä, pudotusvalikkoja ja tekstikenttiä on saatavilla runsaasti. Kaikki elementit ovat myös vapaasti muokattavissa, joten oletuselementit sopivat hyvin placeholdereiksi primitiivisten peliobjektien tavoin. Unity3D:n valmiit UI-komponentit jaetaan visuaalisiin ja interaktiivisiin komponentteihin. Visuaaliset UI-komponentit (Visual Components 2016) ovat erilaisia teksti- ja kuvaelementtejä, jotka ovat osana toiminnallisuutta, mutta eivät laukaise niitä. Tekstiä ja kuvaa voidaan vetää suoraan editorista haluttuun Canvaksen kohtaan ja muokata niitä halutun kaltaisiksi. Unity3D tukee myös erilaisia fonttitiedostoja, joten omat fontit saa pelin teksteille vaivattomasti lisäämällä ne oman projektin Assetteihin.

Interaktiiviset UI-komponentit (Interaction Components 2016) taas laukaisevat jonkin tapahtuman ja niille on sisäänrakennettu omat oletuseventtinsä. Napille voidaan sen sisäänrakennetulla OnClick-eventillä suoraan määritellä mitä muutoksia napin painallus aiheuttaa pelimaailmassa tai sen käyttöliittymässä. Muita valmiita interaktiivisia UI-elementtejä ovat erilaiset liukusäätimet, scroll-alueet ja checkboxit. Kaikki nämä ovat nopeita keinoja luoda kehityksen alkuvaiheessa tapoja kommunikoida pelaajan ja käyttöliittymän välillä. Myöhemmin kaikki elementit voidaan myös korvata omilla kuvatiedostoilla ja rakentaa niille erilaisia toiminnallisuuksia suoraan ohjelmointiskripteistä, mutta tällä tavoin pelin logiikka ja mekaniikat päästään testaamaan ilman suurempaa panostusta.

3.1.4 Virheiden korjaus MonoDevelop- ja Unity3D-ympäristöissä

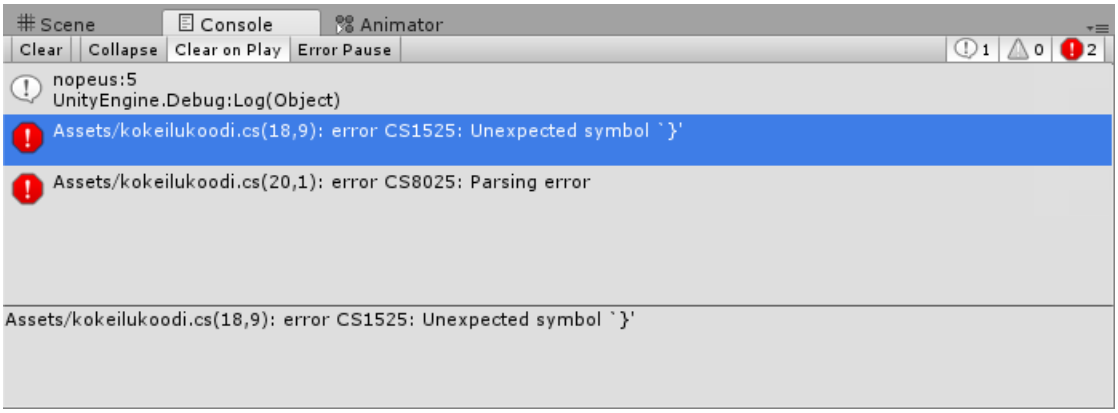
Pelinkehitys on jatkuvaa uuden luomisen ja vanhan testaamisen yhteistyötä. Olemassa olevien virheiden havaitseminen ja korjaaminen sekä mahdollisten tulevien virheiden ennaltaehkäisy mahdollistavat sen, että lopullinen peli on kokemuksena mahdollisimman nautinnollinen pelaajan näkökulmasta. Unity3D:n käyttöliittymän yläreunassa on työkalupalkki, josta pelin voi käynnistää sen nykyisessä muodossa testatakseen sen eri osa-alueita. Unity3D mahdollistaa myös erilaisten arvojen muuttamisen reaaliajassa pelin ollessa käytössä tehden testaamisesta mahdollisimman vaivatonta. Pelin ollessa käynnissä mitkään arvoihin tehdyt muutokset eivät kuitenkaan jää voimaan, vaan ne palautuvat niihin arvoihin, joissa ne olivat ennen pelin käynnistämistä. Silloin tällöin syystä tai toisesta peli ei kuitenkaan käynnisty olemassa olevien virheiden takia. Onneksi virheiden löytämiseen ja ratkaisuun Unity3D ja sen mukana tuleva ohjelmointiympäristö MonoDevelop tarjoavat erilaisia ominaisuuksia.

Unity3D:n ikkunat-valikosta (Window) löytyy Console niminen ikkuna, joka välittää virhe- ja varoitusviestejä käyttäjälle (Console Window 2016). Console-ikkuna voi myös näyttää muita Unity3D:n tai käyttäjän luomia viestejä. Peliä käynnistäessä peli ei välttämättä käynnisty vaan Console-ikkuna ilmoittaa ongelmista punaisella varoitusmerkillä varustetuilla virheviesteillä. Virheviestit kertovat siitä, että jokin ohjelmointikoodissa on synnyttänyt virheen, kun sitä on yritetty kääntää ohjelman toimesta. Virheilmoitus antaa kuvauksen siitä, mistä virhe on syntynyt sekä sen sijainnin ohjelmointiskriptissä, jos se vain on mahdollista. Unity3D ja MonoDevelop kommunikoivat toistensa välillä ja Unity3D osaa Console-ikkunassaan viitata MonoDevelopin löytämiin

virheisiin. Virheilmoitusta klikkaamalla käyttäjä ohjataan suoraan MonoDevelopin puolella olevan ohjelmointiskriptin riville, jossa virhe virhetarkistajan mukaan sijaitsee. On myös paljon tapauksia, jotka eivät johdu suoraan ohjelmointivirheestä ohjelmointiskriptistä. Tällöin ongelmanratkaisua joutuu tekemään tietoa hakemalla ja käymällä läpi jo luotuja skriptejä ja pelin logiikkaa. Tapauksissa, joissa virheisiin voidaan ohjata suoraan editorin kautta, päästään helposti tarttumaan olemassa olevaan ongelmaan ja korjaamaan se. (Debug.Warning 2016.)

Toinen yleinen Unity3D:n antama viesti Console-ikkunassa on varoitusviesti. Varoitusviestit eivät estä pelin käynnistämistä tai kääntämistä, mutta ne voivat johtua virheestä ja siten antaa apua Console-ikkunaan ilmestyneen virheen selvittämiseksi. Varoitusviesti voi myös olla jotain yleistä ilmoitusta siitä, että tekemäsi muutos koodissa ei ole Unity3D:n mielestä halutun kirjoitusasun mukaista. Unity3D:stä julkaistaan jatkuvasti uusia päivitettyjä versioita ja niin myös Unity3D:n käytännöt muuttuvat. Varoitusviesti voi esimerkiksi ilmoittaa, että kirjoittamasi koodi käyttää vanhentunutta tapaa alustaa muuttujia tai viitata olemassa oleviin komponentteihin. Varoitusviestit ilmoittavat myös muuttujista, joita ei käytetä projektissa. Tällaiset virheet eivät välttämättä ole vaarallisia, mutta ne on syytä tarkistaa. Varoitusviestit kannattaa kitkeä pois, ellei niiden tiedetä johtuvan jostain, jota tehdään tietoisesti esimerkiksi testauksen aikana. (Debug.Error 2016.)

Kolmas yleisimmistä viestityypeistä on käyttäjän luomat log-viestit. Unity3D:ssä käyttäjä voi itse synnyttää ohjelmointiskripteissä log-viestejä Console-ikkunaan. Tällä tavoin käyttäjä voi tarkistaa eri metodien toimivuutta tai eri muuttujien arvoja reaaliajassa pelin ollessa käynnissä. Käytäntö nopeuttaa testausta sekä vähentää arvailun määrää. Tällaiset log-viestit ovat yleinen toimintatapa perinteissä ohjelmointityössä, joten sen käyttö peliohjelmoinnissakin on hyvä ottaa tavaksi. Esimerkkitapauksena voidaan käsitellä vihollishahmon tuhoamista ampumalla. Panos osuu selkeästi viholliseen, mutta vihollinen ei häviä pelistä halutulla tavalla. Log-viestien avulla voidaan tarkastella panoksen ja vihollishahmon välillä olevaa törmäystä tai vaikka vihollishahmon terveyspisteiden määrää. Log-viestien avulla kehittäjän oletusten varaan jää vähemmän asioita, kun toiminnallisuus voidaan todistaa toimivaksi testauksen avulla. (Debug.Log 2016.)



KUVA 9. Unity3D:n Console-ikkuna

Console-ikkunan työkalupalkki tarjoaa monia esitystapaa muokkaavia vaihtoehtoja. Viestien määrä voi kasvaa testatessa ja vaikeuttaa entisestään virheiden löytämistä. Työkalupalkin Clear-vaihtoehdolla Consolen sisällön voi poistaa ja ClearOnPlay:lla sisällön tyhjentäminen tehdään automaattisesti aina pelin käynnistyksen yhteydessä. Tällä tavoin on helpompi tietää mitkä virheistä johtuvat meneillään olevasta testauksesta ja mitkä taas edellisistä käynnistyksistä. Clear-toiminto ei kuitenkaan poista kääntäjän huomaamia virheilmoituksia, joten toiminto on kätevä omien log-viestien poistamiseen. Collapse-vaihtoehdolla Console ilmoittaa toistuvasta ongelmasta vain kerran. Useat Update-metodien sisällä syntyvät virheet toistuvat useita kertoja sekunnissa, joten niiden tiivistäminen yhteen esityskertaan voi helpottaa Consolen seuraamista. (Console Window 2016.)

Työkalupalkin viimeinen vaihtoehto on Error Pause. Error Pause pysäyttää käynnissä olevan pelin virheeseen törmätessä. Käyttäjän tekemät log-viestit eivät kuitenkaan pysäytä peliä. Ominaisuus on hyödyllinen, jos käynnissä olevaa Sceneä haluaa tarkastella kohdatun virheviestin syntymishetkellä. Unity3D:n Consolesta pääsee käsiksi synnytettyihin log-tiedostoihin, jotka löytyvät myös koneen kiintolevyltä. Log-tiedostot pitävät sisällään enemmän tietoa, kuin mitä Console näyttää. Tällä tavoin ongelman selvittämistä voi yrittää ratkaista hieman pintaa syvemmältä. (Mt.)

Log-viestien lisäksi muuttujien arvoja pystytään seuramaan myös MonoDevelopin omalla debuggaus-ominaisuudella. Virheen ratkaisua varten voidaan rajata alue ohjelmointiskriptissä, jonka tietyssä pisteessä koodin kääntäminen pysähtyy. MonoDevelopin sisäänrakennettujen ominaisuuksien ansioista koodin kääntämisen pysähtyessä voi-

daan seurata kaikkien kyseisen ohjelmointiskriptin muuttujia. Tämä helpottaa ongelmanratkaisua, kun kaikki muuttujat ovat selkeästi esillä. Näin on myös kätevä seurata muuttujia testauksen kannalta jo ennen kuin mahdollisia virheitä on syntynyt. (MonoDevelop's Debugger 2016.)

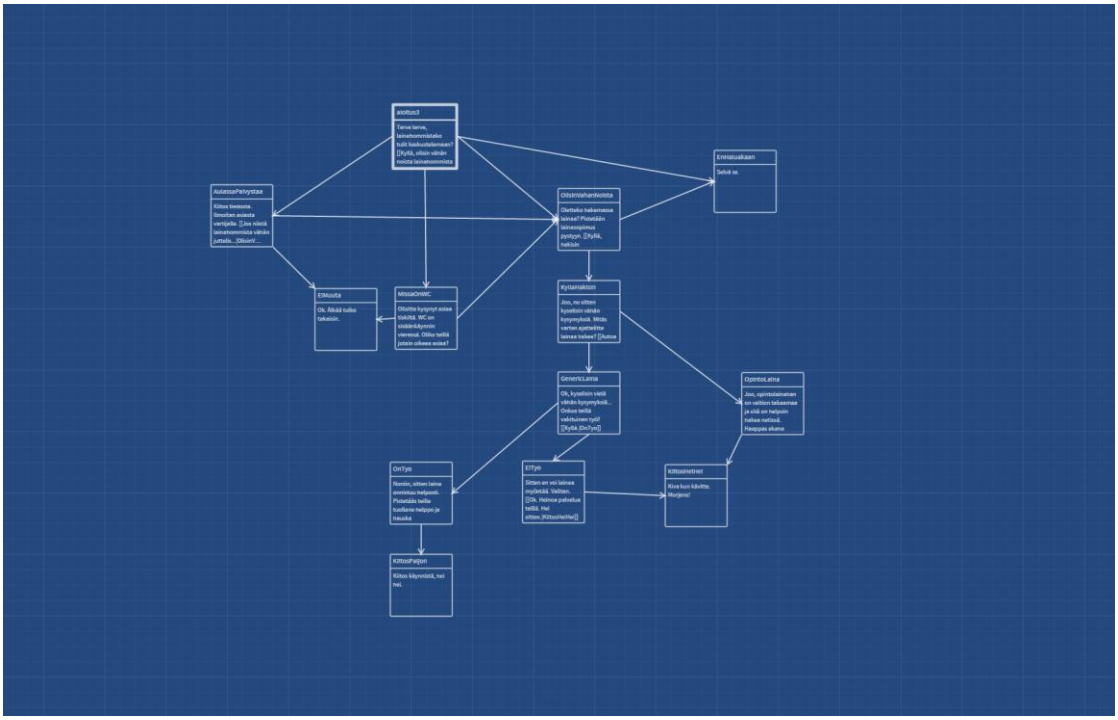
Kaikki edellä mainitut työkalut ovat kuitenkin vain ongelmanratkaisua tukevia keinoja eivätkä ne välttämättä aina ohjaa suoraan ongelman aiheuttaneen tekijän luo. Ongelmanratkaisu on taito, joka kehittyy siinä missä muutkin ohjelmointitaidot niitä harjoitellessa. Erilaiset työkalut onkin tehty tukemaan ongelmanratkaisua sekä sen kehittymistä, joten niiden käyttö kannattaa olla hallussa jo kehityksen alkuvaiheissa.

3.2 Cradle

Twine (Twine 2016) on Chris Klimasin vuonna 2009 kehittämä tarinankerrontatyökalu. Twine on avoimenlähdekoodin työkalu, jolla voidaan luoda dynaamisia tarinoita tai dialogeja. Twine-tarinoita voidaan julkaista suoraan HTML-muodossa ja niitä voidaan parannella käyttäen erilaisia Twinen sisäisiä muuttujia ja valokuvia. Twine tukee myös CSS:ää ja Javasciptiä. Twinellä voidaan tuottaa myös täysin lineaarista tarinaa, mutta parhaiten se pääsee oikeuksiinsa erilaisten hypertekstien avulla. Hyperteksti on tietokoneiden käyttämä käyttöliittymäperiaate, joka mahdollistaa automaattiset hyperlinkeiksi kutsutut ristiviittaukset eri dokumenttien välillä. Twinellä tuotettu tarina antaa käyttäjälle mahdollisuuden vaikuttaa suoraan tarinaan erilaisilla valinnoilla, mikä tekee tarinan kerronnasta interaktiivista ja dynaamista.

Twinea voi käyttää joko työpöytäsovelluksena tai suoraan selaimen kautta. Twinen käyttöliittymä on hyvin selkeä ja tarinaa työstettäessä syntyvä kaavio tukee hyvin haarautuvan tarinan seuraamista. Twinessä yksi tarina koostuu yhdestä Story Map -elementistä, jonka sisällä olevat Passage-elementit luovat lopullisen tarinan. Passage-elementit näytetään käyttäjälle yksi kerrallaan ja käyttäjän tekemät valinnat vievät Story Mapilla olevaa tarinaa eteenpäin Passage-elementtien linkittyessä toisiinsa. Passagelle annetaan otsikko, josta sen tunnistaa ja siihen voidaan viitata. Passagelle määritellään teksti, joka voi olla kerrontaa tai dialogia pelaajan ja pelin välillä. Passageen sijoitetaan vaihtoehtoja, joita valitsemalla tarina ohjautuu linkitysten kautta sitä vastaavaan Passageen. Näin tarinoista voidaan tehdä haarautuvia ja siten luoda monimutkaisia ja laajoja linki-

tyksiä eri elementtien välille. Twine sisältää myös muita erilaisia toimintoja, mutta keskityn tässä alaluvussa kertomaan Twinen ja Unity3D:n välillä kommunikoivasta lisäosasta. (Twine 2016.)



KUVA 10. Twine tarina. Passaget ja niiden linkitykset toisiinsa.

Twine soveltuu hyvin dialogien tekemiseen, koska helpoilla kaavioilla luodut elementit mahdollistavat nopean dynaamisten dialogien rakentamisen. Unity3D ei suoraan tue Twineä, mutta D. A. Terre on työstänyt vuodesta 2015 GitHubissa olevaa Cradle-lisäosaa (Cradle 2016) Unity3D:n ja Twinen välille. Opinnäytetyön käytännön osuudessa dialogit ovat suuressa roolissa, joten kyseinen lisäosa vaikutti hyvälle vaihtoehdolle projektin toteuttamisen kannalta. D. A. Terren luoma lisäosa Cradle (aiemmin nimellä UnityTwine) auttaa Unity3D:tä ymmärtämään Twinestä saatavaa tarinaformaattia. Cradle rakentaa Twinen Passageiden ja linkitysten perusteella UI-rakenteen, joka vastaa Twinen rakennetta.

Twinessä rakennettu tarina täytyy tuoda muodossa, jonka Cradle pystyy kääntämään C#-skriptiksi Unity3D:ssä. Cradle tukee Harlowe-, Sugarcane- ja Sugarcube-formaatteja. Twinen tarinasta muodostuu yksi tekstitiedosto, joka sisältää kaiken keskusteluun tai tarinaan luodun informaation. Seuraavaksi tekstitiedosto voidaan viedä Unity3D:hen, jossa Cradle luo automaattiseesti tiedostosta ohjelmointiskriptin. Erilaiset

merkit, kuten ääkköset, kannattaa toiminnan takaamiseksi muuttaa oikeaan merkistömuotoon ulkopuolisella ohjelmalla. (Cradle 2016.)

Cradle luo tekstitiedostosta ohjelmointiskriptin, joka tarvitsee toimiakseen kaksi peliobjektia. Ohjelmointiskriptille pitää luoda tyhjä ja Canvas-komponentilla varustettu peliobjekti. Tyhjä peliobjekti voidaan nimetä esimerkiksi keskustelun otsikon mukaan, jotta sen tunnistaminen helpottuu. Peliobjektiin lisätään Cradlen luoma ohjelmointiskripti ja määritellään mistä Passagesta keskustelun halutaan alkavan. Tämä kyseinen keskustelu pitää taas liittää Canvas-elementin sisältävään peliobjektiin, jotta näytölle piirrettävät elementit löytävät niille tarkoitetun sisällön. Cradle piirtää Passage-elementtien päätekstin tavallisena leipätekstinä dialogi-ikkunaan. Vastausvaihtoehdot taas luodaan omiksi linkeikseen. Tällä tavoin jokainen vastaus osaa luoda Twinen rakenteen mukaiset tekstielementit ruudulle vanhojen tilalle. (Mt.)

Erilaisiin dialogi-ikkunan tilanteisiin voidaan reagoida ohjelmointiskriptien kautta luomalla Passage-otsikoihin viittaavia metodeja. Cradlen toimintaan on sisäänrakennettu Enter- ja Exit-metodit. Näillä voidaan laukaista jokin kohta ohjelmointiskriptin koodista, kun uusi dialogi-ikkuna ilmestyy tai jokin tekstilinkeistä valitaan. Tällä tavoin dialogeihin voidaan sisällyttää erilaisia animaatioita, ääniä tai antaa muuttujille eri arvoja. Esimerkiksi keskustelun toinen osapuoli voi animaatioiden kautta reagoida selkeämmin pelaajan tekemiin valintoihin tai keskustelu voi muuttujien arvojen muutoksesta johtuen vaikuttaa tuleviin keskusteluihin.

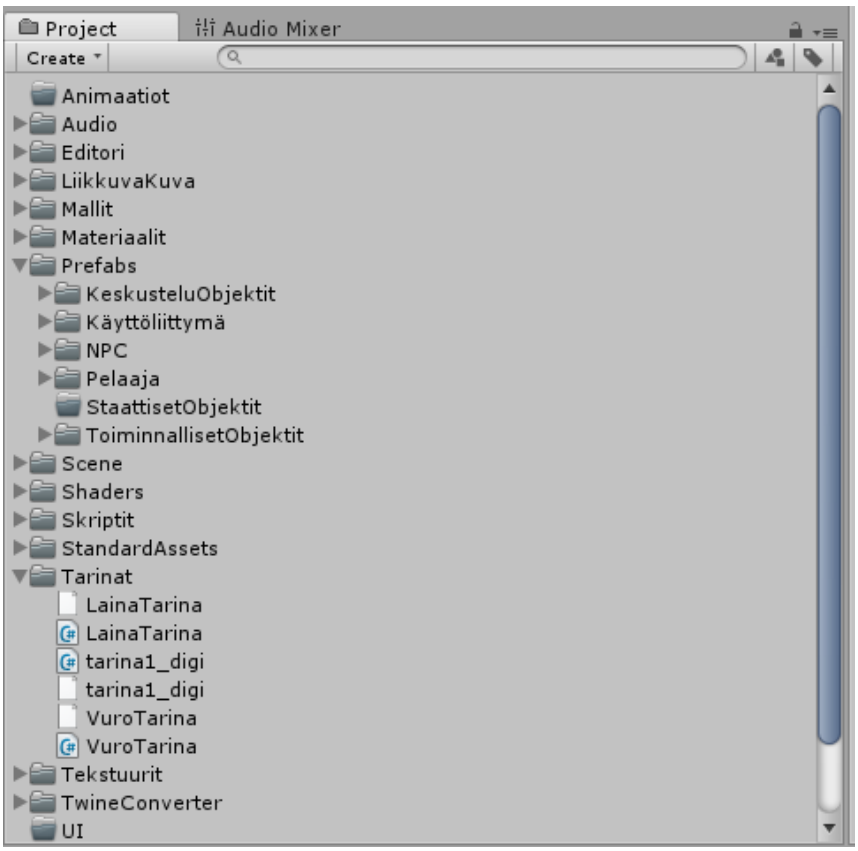
3.3 Käytäntöjä ryhmätyöskentelyn ja jatkokehityksen parantamiseksi

Peliprojektia aloittaessa on tärkeää saada nopeasti aikaan sisältöä testaamisen ja jatkokehityksen vuoksi. Projektin alkuvaiheessa on kuitenkin hyvä pitää mielessä sen selkeä luettavuus ja johdonmukaisuus, sillä mitä suuremmaksi projekti kasvaa, sitä tärkeämmäksi edellä mainitut asiat tulevat. Kehitystiimiin voi tulla uusia jäseniä tai siitä voi vastata täysin uusi ryhmä kehittäjiä. Oman opinnäytetyöni kannalta tämä on todella tärkeää, koska kyseinen peliprojekti jatkuu oman osallistumiseni jälkeen. Tämän takia luodun pohjan olisi hyvä olla mahdollisimman selkeä lukuinen, jotta seuraavat mukana olevat kehittäjät pääsevät mahdollisimman pienellä vaivalla jatkamaan pelin kehitystä. Hyviä käytäntöjä nimeämisen, projektin hallinnan ja ohjelmoinnin kannalta on monia, mutta nostan tässä esiin tämän projektin kannalta tärkeitä huomioita.

Assettien ja ohjelmointimuuttujien määrä kasvaa nopeaa vauhtia kehityksen edetessä. Tästä syystä on tärkeää antaa Asseteille kuvaavat nimet ja välttää moniselitteisten lyhenteiden käyttämistä. Kuvaavien nimien käyttäminen voi tuntua hyvin itsestään selvältä, mutta käytäntö nopeuttaa huomattavasti oman Asset-kirjaston käyttöä. Testausvaiheessa voi tuntua järkevältä käyttää esimerkiksi nappi1- ja nappi2-tyylistä nimeämistä, mutta sekaannuksen välttämiseksi tällaiset nimeämiset on muutettava viimeistään testaamisen jälkeen. Samat ongelmat tulevat pitkällä tähtäimellä eteen myös lyhenteiden käyttämisessä. Käytetyt lyhenteet pysyvät suurella todennäköisyydellä selkeinä sen kirjoittajalle, mutta ulkopuolisen silmin lyhenteet voivat aiheuttaa päänvaivaa. (Tulleken 2012.)

Kuvaavien nimien lisäksi nimeämisessä kannattaa käyttää johdonmukaista nimeämistapaa kuten esimerkiksi CamelCasea. Kyseissä nimeämiskäytännössä nimen ensimmäinen sana alkaa joko pienellä tai isolla kirjaimella, mutta kaikki seuraavat sanat alkavat isolla kirjaimella. Käytäntö on hyvin yleinen ohjelmointimaailmassa ja sitä näkee käytettävän silloin tällöin myös tuotemerkeissä kuten iPhone tai FedEx. Tärkeimpiä Assetteja voidaan myös korostaa kansiorakenteiden hierarkiassa laittamalla nimen eteen alaviiva esimerkiksi _GameManager. Näin nimetyt Assetit tulevat kansion järjestyksessä ensimmäiseksi, joten niiden löytäminen suuresta määrästä Assetteja helpottuu. (Top 10 Unity Best Practices: Naming Conventions 2015.)

Myös Asseteille määritellyt kansiot kannattaa pitää selkeinä. Yleismaailmalliset äänet, kuvat ja ohjelmointiskripit ovat hyviä kansioiden nimiä, mutta niiden sisään voidaan hyvin tehdä vielä tarkentavia kansioita kuten taustamusiikit, vihollistenÄänet jne. Tiivistettynä nimeämiskäytäntöjen pitäisi olla sellaisia, että tuntematon käyttäjä voi mahdollisimman helposti ymmärtää mistä on kyse. Kuvaava nimi kannattaa keksiä niin, että sitä ei tarvitse myöhemmin muuttaa ja tällä tavoin ennaltaehkäistä ongelmia jatkossa.
Projekteissa peliobjekteja on paljon, joten niiden hallinta tehokkaasti on tärkeää. Valmispeliobjekteja kannattaa luoda aina kun peliobjektin tarkoitus tiedetään. Samat peliobjektit voivat esiintyä pelissä satoja kertoja eri Scene-tiedostoissa ja valmispeliobjektien avulla ne pysyvät hallinnassa kehityksen edetessä. (Mt.)



KUVA 11. Esimerkki Asset-kansioiden rakenteesta ja nimeämisestä

Unity3D:ssä on sisäänrakennettuna myös tapa merkitä (Tag) peliobjekteja. Peliobjekteille voidaan luoda erilasia Tag-ryhmiä (Tags, 2016), joiden perusteella ne pystytään tunnistaa esimerkiksi ohjelmointiskripteissä. Pelihahmo voi esimerkiksi omata Player-Tagin ja vihollishahmot Enemy-Tagin. Tagien avulla editorissa voidaan saada nopeasti näkyviin tietyt peliobjektit tai varmistaa ohjelmointiskriptien viittaavan juuri oikean tyylisiin peliobjekteihin. Tagien lisäksi peliobjekteja voidaan jakaa erilaisiin Layer-ryhmiin (Layers, 2016), joiden avulla valaistus, kameran renderöinti tai törmäystunnisteet voidaan aktivoida vain tietyille peliobjekteille. Niin Tagien kuin Layerin avulla peliobjektien hallinta pysyy johdonmukaisena, kunhan uudet peliobjektit muistetaan asettaa omiin ryhmiinsä.

Jatkokehitystä ajatellen ohjelmointiskripteissä olevien muuttujien ja metodien täytyy olla ymmärrettävissä myös henkilöille, jotka eivät ole kirjoittaneet koodia. Nimeämiskäytäntöjen lisäksi omat koodit tulee kommentoida mahdollisimman selkeästi, jos ne eivät ole itsestään selviä. Kommenteissa olevat koodit eivät mene ohjelmointikääntäjän läpi, vaan ne jäävät käyttäjiä varten oleviksi kommenttiteksteiksi. Varsinkin pidempien metodien yläpuolelle kannattaa kommentoida sen toimintaa selkeyttävä esittelyteksti.

Ohjelmointiskripteihin kirjoitetut julkisiin luokkiin kuuluvat metodit näkyvät suoraan editorissa komponentin arvoissa. Arvoja voidaan siis muuttaa suoraan editorista pelin ollessa käynnissä ja näin testata eri arvoja parhaan lopputuloksen saavuttamiseksi. Muuttujille voidaan antaa työkaluvinkki (Tooltip Attribute 2016), joka näkyy editorissa hiiren ollessa muuttujan nimen päällä. Tätä varten muuttujaan on tehtävä ohjelmointiskriptissä vinkkiteksti, ja muuttujan on kuuluttava julkiseen luokkaan. Muuttujalle nimeltä PelaajanNopeus voidaan kirjoittaa esimerkiksi vinkkiteksti ”Syötä tähän pelaajan liikkumisnopeus”.

4 DIGIOSAAJAKSI TYÖELÄMÄÄN

Digiosaajaksi työelämään -hanke on Työväen sivistysliiton, Mikkelin ammattikorkeakoulun ja Itä-Suomen yliopiston toteuttama sekä Euroopan sosiaalirahaston ja Etelä-Savon Elinkeino-, liikenne- ja ympäristökeskuksen rahoittama hanke. Hankkeen tavoitteena on parantaa heikot perustaidot omaavien aikuisten tieto- ja viestintäteknisiä taitoja ja sitä kautta parantaa Etelä-Savon työllisyystilannetta. Hankkeen maantieteelliset toiminta-alueet ovat Mikkelin, Savonlinnan ja Pieksämäen seutukunnat. Hanke toteuttaa toimintakaupungeissa noin puolen vuoden mittaisia monitoimikoulutuksia, joissa edellä mainittuja taitoja kehitetään osallistavin oppimenetelmin. (Digiosaajaksi työelämään 2016.)

Mikkelin ammattikorkeakoulun rooli hankkeessa on toteuttaa työelämäpeli, joka tarjoaisi vaihtoehtoisen tavan oppia TVT-taitoja, työelämän pelisääntöjä, alaistaitoja sekä vuorovaikuttamista osana ryhmää. Pelillisyyden avulla voidaan myös tukea erilaisia oppijoita, joilla on esimerkiksi lukivaikeuksia tai muita oppimisen pulmia. Peliä on tarkoitus kehittää käyttäjäläheisenä prosessina, jonka toteuttamiseen kohderyhmät vahvasti osallistuvat.

Minun ja viiden muun Mikkelin ammattikorkeakoulun opiskelijan tehtävänä on luoda pohja edellä mainitulle työelämäpelille ja tarkoituksena on, että tulevaisuudessa toiset opiskelijaryhmät jatkaisivat pelin kehittämistä aina sen valmistumiseen saakka. Työelämäpelin toteuttaminen jakautui kolmeen erilaiseen projektiin, joita toteutettiin pareittain. Yhden ryhmän osa-alueena oli toteuttaa pelille pelillinen käyttöliittymä hyödyn-

täen Mikkelistä tehtyä 3D-mallia. Toinen ryhmä taas kehittää pelihahmon muokkauseditorin, jolla jokainen pelaaja voi luoda mieleisensä pelihahmon peliä varten. Kolmas projekti oli luoda pelille palvelutilanne, sen ympäristöt sekä interaktiot.

Minun ja parini Martti Honkalan tehtävänä oli luoda palvelutilanne ympäristöineen ja interaktioineen käyttäen hyväksi Unity3D-pelimoottoria sekä Twine-tarinatyökalua. Tavoitteena oli luoda raamit erilaisille palvelutilanteille, joita pelin lopullisessa tuotoksessa on useita. Palvelutilanteiden pääpainona on erilaiset dialogit, joita käymällä pelaajat oppivat hyödyllisiä taitoja, joita he voivat soveltaa oikeassa maailmassa. Immersion parantamiseksi palvelutilanteisiin toteutetaan ympäristöt 3D-malleilla ja ympäristöön luodaan oma äänimaailma. Peliin lisätään erilaisia interaktioita niin pelin tarinan tueksi kuin ylimääräisiksi pelaajaa viihdyttäviksi osa-alueiksi. Mikkelin kaupungin 3D-malli ja sen käyttäminen käyttöliittymänä sekä hahmonluominen ja sen pelissä hyödyntäminen tulevat myös osaksi lopullista peliä. Kerron kaupunkikäyttöliittymän ja palvelutilanteiden yhdistämisestä tämän opinnäytetyön loppupuolella.

4.1 Palvelutilanteen luominen

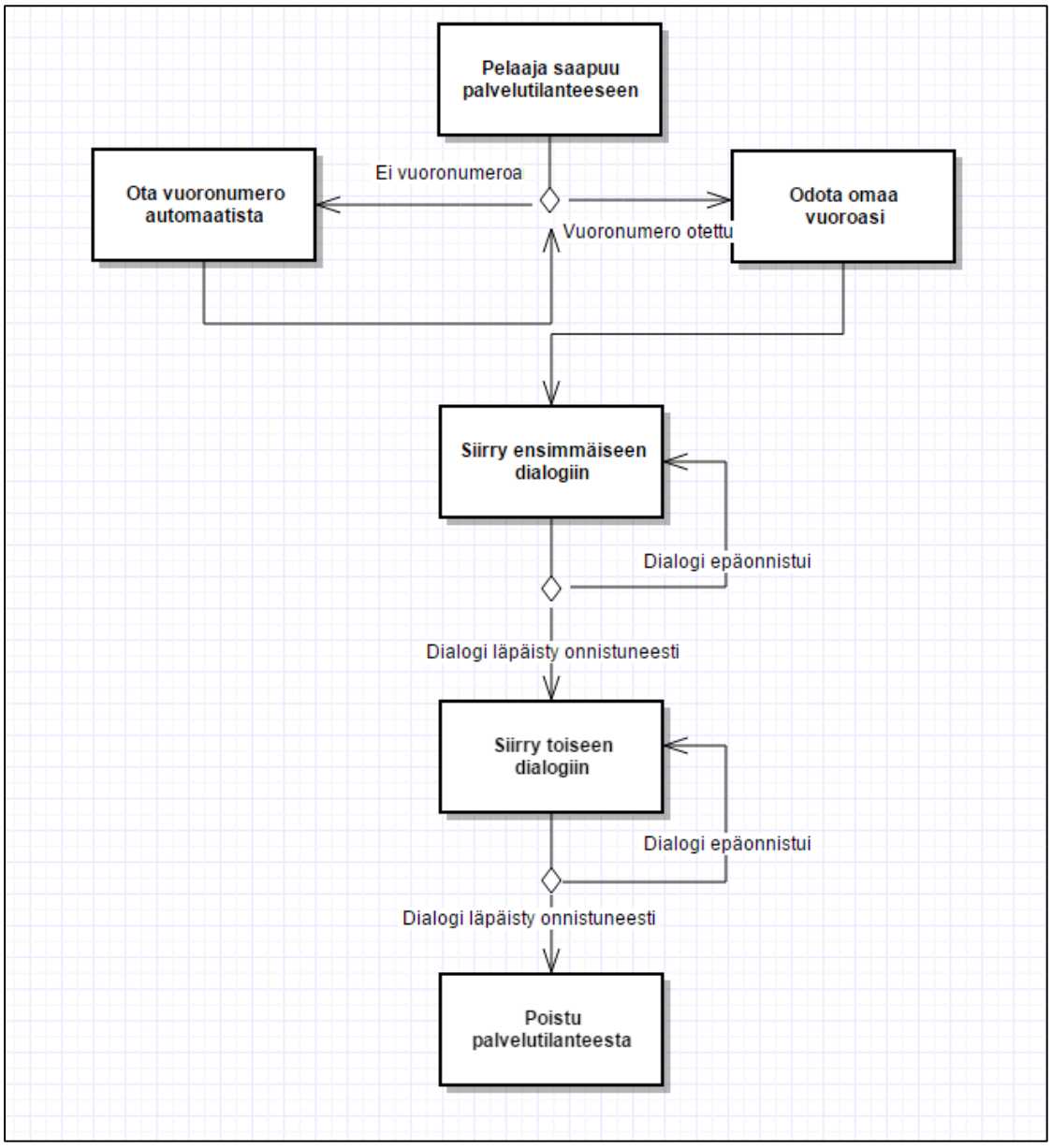
Tässä opinnäytetyöni osioissa kerron tarkemmin itse käytännön työn toteutuksesta. Tavoitteena oli luoda yleismaailmallinen palvelutilanne, jonka ominaisuuksia voidaan monistaa erilaisiin palvelutilanteisiin ja luoda niille ominaisia piirteitä. Palvelutilanteiden pääpainona on Unity3D:n ja Twinen yhdistävä Cradle ja sillä aikaan saadut dialogit. Dialogien ympärille luodaan erilaista juonta ja toiminnallisuutta piristämään oppimistilannetta. Palvelutilanteelle rakennetaan myös tilanteeseen sopiva ympäristö ja lopuksi tilanteeseen sijoitetaan palautekeskustelu pelaajia varten.

4.1.1 Suunnittelu

Pelin suunnittelu alkoi palvelutilanteen valitsemisella sekä pelinsuunnitteludokumentin luomisella. Tämän kokoisesta MVP:stä toteutettiin yksisivuinen pelinsuunnitteludokumentti, joka määritteli pelin pääpiirteet. Palvelutilanteen piti tukea kurssin aihe-alueita työelämän taitojen parantamiseksi. Hyväksi vaihtoehdoksi osoittautui Työvoimatoimisto ja siihen perustuva asiointitilanne. Työvoimatoimisto ympäristönä mahdollistaa monien immersiota lisäävän elementin käytön palvelutilanteessa. Vuoronumeroiden käyttö, palvelutiskit, odotushuoneen kalusteet sekä erillisissä huoneissa olevat toimistot

auttoivat rakentamaan hyvän juonen pelin tilanteille. Työvoimatoimisto sisältää myös yhtäläisyyksiä monien muiden työelämän ja sitä tukevien palveluiden kanssa, joten tilanteesta pyrittiin tekemään mahdollisimman monistettava.

Seuraavaksi pelinsuunnitteludokumenttiin voitiin alkaa listata pelin ominaisuuksia ja tavoitteita. Pelin pääpaino mekaniikkojen osalta on dialogijärjestelmässä. Sen ympärille rakennetut interaktiot mahdollistavat tapoja kuljettaa pelin juonta sekä antavat pelaajalle mahdollisuuksia tutkia ympäristöä. Pelin juoneksi kehittyi yksinkertainen palvelutilanne. Pelaaja ottaa vuoronumerokoneesta vuoronumeron ja siirtyy vapaalle palvelutiskille, kun peliympäristössä oleva vuoronumerotaulu näyttää saatua vuoronumeroa. Pelaaja tulee tiskille ja aloittaa dialogin tiskillä olevan työntekijän kanssa. Dialogin jälkeen pelaaja ohjataan kohti toimistoa, jossa on palvelutilanteen palautteen antokeskustelu toisen NPC:n kanssa.



KUVA 12. Pelin juonta kuvaava kaavio

Palvelutilanne on yksinkertainen, sillä se on kohdennettu testaamaan pelin pääpiirteitä. Tällä tavoin tietoa saadaan pelaajalta suorasti palautekeskustelulla ja epäsuorasti tarkastelemalla juonen etenemistä. Pelin on oltava niin selkeä ja johdonmukainen, että pelaaja onnistuu suorittamaan tapahtumaketjun. Jos pelaaja ei pysty tähän, on pelin mekaniikkoja selkeennytettävä lisää.

4.1.2 Tilan luominen ja törmäystunnistuksen testaus

Tässä vaiheessa suunnittelu eteni pelin tilan luomiseen. Tilaksi valittiin kuviteltu työvoimatoimisto, sillä siihen oli helppoa liittää pelissä tarvittavia ominaisuuksia. Pohjapiirustusta toteutettiin perinteisesti paperilla sekä erilaisilla kuvankäsittelyohjelmilla.

Tilan täytyy olla tarpeeksi suuri, jotta pelaaja pääsee vapaasti tutkimaan ympäristöä. Tilan pitää kuitenkin olla riittävän pieni, jotta pelaajan on helppoa löytää pelin kannalta oleelliset asiat. Pohjapiirustukseen tuli kaksi käytävää ja kaksi avointa tilaa. Pientä käytävää pitkin päästään pääaulaan, josta lähtee toinen käytävä, jonka varrelle sijoitettiin wc-tilat ja toimistot. Käytävälle sijoitettiin useita toimistoja, joista yksi on oikea tila, johon pelaaja pystyy siirtymään. Muiden toimistojen ovet toimivat vain rekvisiittana.

Googlen kuvahakua ja omia mielikuviamme käyttäen pohdimme asioita, joita työvoimatoimistosta yleensä löytyy. Teimme pohdinnoissamme esille tulleista asioista peliobjekteja. Peliobjektit jaoteltiin niiden käyttötarkoituksen mukaan staattisiin ja toiminnallisiin peliobjekteihin. Staattiset peliobjektit toimivat pelaajan eläytymistä vahvistavina tekijöinä ja ne pyrkivät luomaan pelistä mahdollisimman realistisen. Toiminnallisten peliobjektien päällimmäisenä tarkoituksena oli edistää pelin juonen kulkua.

Ensin toteutimme vain yhden huoneen, johon laitoimme tiskinä toimivan Unity3D:n primitiivisen kuutio-peliobjektin. Latasimme Asset Storesta kontrollerin ja 3D-mallin pääpelaajalle. Kontrollerin avulla hahmo pystyi liikkumaan, mikä helpotti tilan mittasuhteiden toimivuuden testaamista. Loimme samaa 3D-mallia käyttäen myös NPC-hahmon. NPC-hahmo asetettiin juuri luodun tiskin taakse, jossa se toimi toimistotyöntekijänä. NPC:lle lisättiin Collider-komponentti, joka reagoi pelaajan saapuessa tiskin eteen. Log-viestejä hyödyntäen tarkistimme törmäystunnistuksen toimivuuden.

Seuraavaksi tilassa olevat placeholder-peliobjektit täytyi korvata oikeita esineitä kuvaavilla 3D-malleilla. 3D-mallien tuli olla tyyliin sopivia ja ilmaisia käyttää. Päädyimme käyttämään Sketchup 3D -mallinnusohjelmalla toteutettuja malleja, sillä yhden ryhmän käyttämä Mikkelin kaupungin 3D-malli oli tehty kyseistä ohjelmaa käyttäen. Uskomme, että tämä helpottaa pitämään pelin eri osa-alueet yhtenäisinä. SketchUp-ohjelmasta löytyy 3D Warehouse 3D-mallikirjasto, johon ihmiset voivat lisätä tekemiään mallinnuksia toisten käyttäjien ladattaviksi. 3D-mallien käyttö omassa tapauksessamme oli SketchUp:n käyttäjäehtojen mukaista. (3D Warehouse Terms of Use FAQ 2016). Pelin tyyliin sopivia 3D-malleja löytyi hyvin 3D Warehouse -kirjastosta ja saimme tarvitsemamme mallinnukset sitä kautta. Valmiiden mallien käyttäminen palvelee ajatusta prototyypin nopeasta luomisesta.

Unity3D:n peliobjektit noudattavat sen omaa pituuden yksikköä, jossa yksi yksikkö vastaa metriä oikeassa maailmassa. Ensimmäinen ongelma valitsemiemme 3D-mallien yhteensopivuudessa Unity3D:n kanssa tuli esiin, kun malleja siirrettiin Unity3D:n editoriin. SketchUp:n mallien kokosuhteet eivät vastanneet Unity3D:n käyttämiä yksiköitä. Tämän seurauksena mallien mittasuhteet vääristyivät, kun ne siirretiin Unity3D:hen. Unity3D:ssä on malleille määriteltävä Scale Factor -arvo, jonka avulla pystyimme ratkaisemaan mallien kokosuhteiden muuttumisen aiheuttamat ongelmat. SketchUp:n käyttämien pituusyksiköiden yksi tuuma vastaa Unity3D:n käyttämää yhtä yksikköä. Syöttämällä Scale Factor -arvoksi 0.255 (Sketchup 2016) kaikki projektiin tuodut 3D-mallit skaalautuivat oikeaan suhteeseen.

KUVA 13. Palvelutilanteen tila erilaisine 3D- malleineen

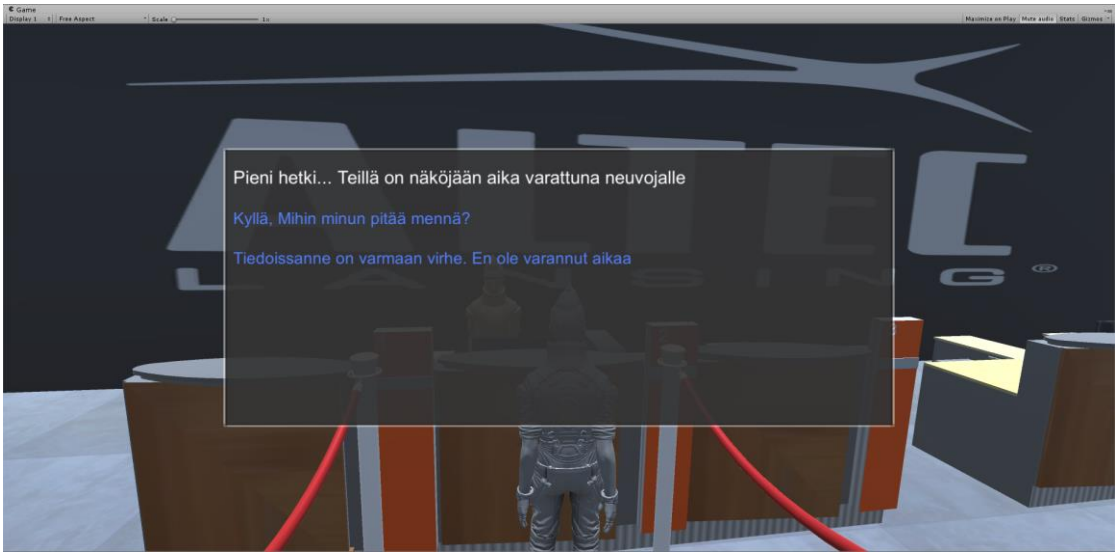
Pelimaailman lattiat ja seinät saivat materiaalikomponentit, joilla niihin lisättiin tarvittavat tekstuurit. Pelihahmo sekä käytettävät NPC:t jäivät Asset Storesta tuoduiksi oletushahmoiksi, koska kehitettävän MVP:n kannalta ne eivät olleet tärkeitä. Pelihahmo korvataan myöhemmin hahmolla, jonka pelaaja voi muokata haluamansa näköiseksi. Tämä on yhden peliprojektissa mukana olevan hahmonmuokkaus-ryhmän tehtävänä.

4.1.3 Dialogien toteuttaminen

Digiosaajaksi työelämään -hankkeen järjestämien kurssien tavoitteena on, että kurssille osallistuvat pääsevät suunnittelemaan ja toteuttamaan eri palvelutilanteita ja niissä käytettäviä dialogeja. Meidän tehtävänämme oli toteuttaa toiminnalliset valmiudet kokeiludialogilla sekä luoda palvelutilanteeseen palautekeskustelu. MVP:tä varten kehitetyn

dialogin oli päästä kokeilemaan erilaisia haarautuvia ja dynaamisia keskusteluraken-teita. Päätimme toteuttaa keskustelun, joka sisältää monia vastausvaihtoehtoja sekä kaksi erilaista polun päätöstä. Ensimmäinen polun päätöspiste olisi onnistunut keskustelu, joka ohjaa pelaajan seuraavaan ennalta luotuun palautekeskusteluun. Toinen keskustelu taas nollaa keskustelun ja pelaajan on suoritettava keskustelu uudestaan päästäkseen haluttuun lopputulokseen.

Dialogit luotiin Twine-tarinana ja luotu Twine-tarina käännettiin tekstitiedostoksi SugarCube-formaatissa. Unity3D:n ja Twinen yhteistyön mahdollistamiseksi projektiin täytyi tuoda GitHubin kautta Cradlen tiedostot. Cradlen avulla SugarCube formaatissa oleva tekstitiedosto voidaan kääntää C#-ohjelmointiskriptiksi. Dialogin toteutusta varten Unity3D:ssä luotiin kolme erilaista peliobjektia. Keskustelulle luotiin oma tyhjä peliobjektinsa, johon Cradlen luoma ohjelmointiskripti liitettiin komponenttina. Kyseinen peliobjekti sisältää Twinessä luodun tarinan sisällön. Toinen peliobjekti on Canvas-komponentin sisältävä peliobjekti, joka muodostaa pelin käyttämän graafisen käyttöliittymän. Käyttöliittymänä toimivaan peliobjektiin lisätään Cradlen asennuspaketista löytyvä Twine Text Player -ohjelmointiskripti. Tarinan sisältävä peliobjekti liitetään komponenttina Twine Text Player -ohjelmointiskriptin sisältävään peliobjektiin. Näiden kahden peliobjektin välille luodulla linkillä käyttöliittymään voidaan piirtää halutun tarinan tai keskustelun eri vaiheita.



KUVA 14. Dialogi-ikkuna palvelutilanteessa

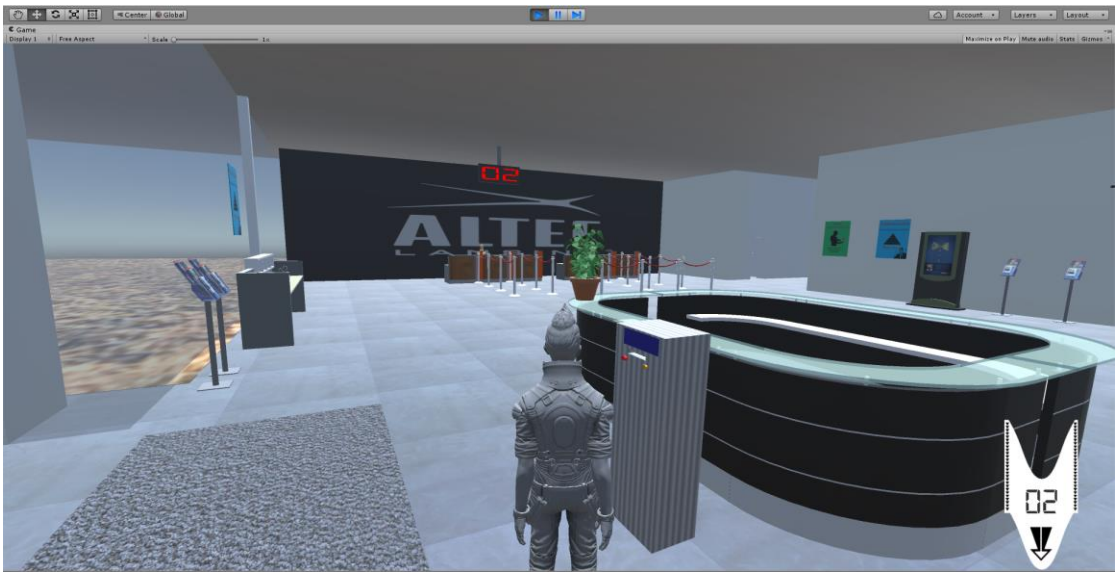
Käyttöliittymälle piirtyy luodun keskustelun ensimmäinen Passage vastausvaihtoehtoineen heti kun peli käynnistetään, joten kyseinen osa käyttöliittymän omaavasta peliobjektista täytyi laittaa toistaiseksi näkymättömäksi. Näin pelin käynnistyessä dialoginäkymä on jo olemassa, mutta käyttäjä ei pysty sitä näkemään. Dialogin näyttämiseen luotiin interaktio pelaajan ja pelimaailman välille. Aiemmin lisätyn tiskillä odottavan NPC-pelihahmon sisälle luotiin törmäystunnistusta varten uusi tyhjä peliobjekti. Törmäystunnistusta varten luotiin ohjelmointiskripti, joka näyttää dialoginäkymän pelaajalle, kun pelihahmo tulee tarpeeksi lähelle tiskiä. Dialoginäkymä näkyy pelaajalle niin kauan kuin törmäystunnistus tunnistaa pelihahmon olevan kantaman sisässä.

4.1.4 Interaktiot ja immersion luominen

Tässä vaiheessa käytännön työtä pelissä oli tarvittavat ominaisuudet, mutta niiden väliset juonen muodostavat interaktiot puuttuivat. Juonen mukaisesti ensimmäisellä ja toisella dialogilla on omat vaatimuksensa ennen kuin pelaaja pääsee niihin käsiksi. Ensimmäinen tiskillä käytävä dialogi NPC:n kanssa vaatii vuoronumeron ottamista sekä oikeaan aikaan tiskille saapumista. Toinen dialogi taas vaatisi ensimmäisen onnistunutta läpäisyä.

Vuoronumeron vastaanottaminen toteutettiin luomalla vuoronumerokone, johon ilmestyy ohjetekstillä varustettu nappi, kun pelihahmo saapuu tarpeeksi lähelle konetta. Vuoronumerokoneelle luotu ohjelmointiskripti arpoo numeron kahden ja viiden välillä ja kyseinen luku ilmestyy graafisen käyttöliittymän oikeaan alakulmaan. Vuoronumero ei voi koskaan olla numero yksi, koska pelaajan täytyy odottaa omaa vuoroaan. Vuoronumeron saaminen ja oman vuoron odottaminen vaativat tietysti myös sen, että palvelutiskien palvelutilannetta voidaan seurata vuoronumeron näyttävältä taululta. Vuoronumeronäyttö luotiin käyttämällä primitiivistä mustan väristä kuutiota, jonka pintaan liitettiin Text Mesh -komponentilla varustettu tekstielementti. Vuoronumeron ottaminen laukaisee ohjelmointiskriptissä olevan metodin, joka vaihtaa ajastettuna vuoronumeronäytöllä sijaitsevaa tekstiä eteenpäin numero kerrallaan. Kun vuoronumeronäytön numero vastaa pelaajan saamaa vuoronumeroa, kuuluu merkkiääni ja pelaajan odotetaan siirtyvän tiskille dialogin käymistä varten.

Teimme pelissä käytettävästä NPC-hahmosta valmisobjektin, koska sekä tiskin takana toimiva työntekijä, että toimistossaan oleva työntekijä ovat dialogipeliobjektia ja dialogin aloitusvaatimuksia lukuun ottamatta samanlaisia. Dialogin aloitusvaatimusten täyttymistä seurataan yksinkertaisella totuusarvolla eli sen arvo voi olla joko tosi tai epätosi. Cradlen tuomien lisäominaisuuksien ansiosta koodin avulla voidaan halutussa dialogin vaiheessa muuttaa eri peliobjektien ja komponenttien arvoja. Kun pelaaja ottaa vuoronumeron automaatista muuttuu totuusarvo NPC:n ohjelmointiskriptissä todeksi ja pelaaja voi siirtyä dialogin pariin. Samanlainen totuusarvon tarkistus on myös palvelutilanteen viimeisessä dialogissa. Dialogipeliobjektin vaihtaminen kahden dialogin välissä onnistuu myös ohjelmointiskriptin kautta, joten samaa graafista käyttöliittymää voidaan käyttää molempien keskustelujen piirtämiseen.



KUVA 15. Palvelutilanne. Kuvassa näkyvät pelaajan saama vuoronumero sekä tiskien yläpuolella sijaitseva vuoronumeronäyttö.

Palvelutilannetta varten toteutimme myös erilaisia juonen kannalta merkityksettömiä interaktioita ja peliobjekteja. Odotushuoneeseen luotiin radio, jonka pelaaja voi sammuttaa tai käynnistää, televisio, jossa pyörii siihen liitetty videotiedosto sekä erilaisia julisteita ja mainostauluja. Tällaisilla ominaisuuksilla ei ole merkitystä pelaajan tavoitteiden toteutumisen kanssa, mutta ne luovat tilaan erilaisia audiovisuaalisia elementtejä, joilla pelin immersiota voidaan kasvattaa.

4.1.5 Palautteen kerääminen pelaajalta

Pelin juonen viimeinen osio on toinen dialogi, jolla kerätään suoraa palautetta pelaajalta. Pelaajalta kysytään palvelutilanteeseen ja käyttökokemukseen liittyviä kysymyksiä, joiden avulla peliä sekä myös kurssin sisältöä voidaan kehittää. Palautteen keräämiseen tarvitaan niin kysymykset kuin niiden vastaukset. Palaute pitää myös toteuttaa niin, että se saadaan helposti talteen pelin testauksen jälkeen. Päätimme toteuttaa palautteen keräämisen XML-tiedostoa käyttäen. Päädyimme tähän valintaan, koska Unity3D tukee hyvin niiden lukemista ja niihin kirjoittamista. Kehityksen tässä vaiheessa palaute tulee jokaiselta pelaajalta omaan tiedostoonsa. Pelin testauksen järjestäjät keräävät saadut palautteet yhteen.

Kysymykset kerätään Twinessä luodusta Passage-otsikosta käyttäen Cradlen sisäänrakennettua CurrentPassageName-muuttujaa. Vastaukset taas kerätään käyttäen hyväksi Cradlen muodostamia linkkejä dialogi-ikkunassa. Jokaiselle linkille on lisätty OnClick-eventti, joka lisää sen hetkisen kysymyksen ja pelaajan valitseman vastauksen molemmille erikseen määriteltyyn listaan. Molemmat listat vastaavat järjestysnumeroillaan toisiaan, joten oikealle kysymykselle ja vastaukselle löytyy niiden indeksiä vertaamalla käyttäen oikea pari. Palautekeskustelun päättävälle Passagelle lisätään Cradlen sisäänrakennettu Enter-metodi. Metodin sisälle kirjoitimme koodin, joka koostaa molempien listojen sisältöjä hyödyntäen johdonmukaisen XML-tiedoston. Tiedosto erittelee kysymykset omiin elementteihinsä erotellen kysymystekstit ja vastaustekstit toisistaan. Keskustelun eteneminen on selkeästi luettavissa palautteen keräämistä varten.

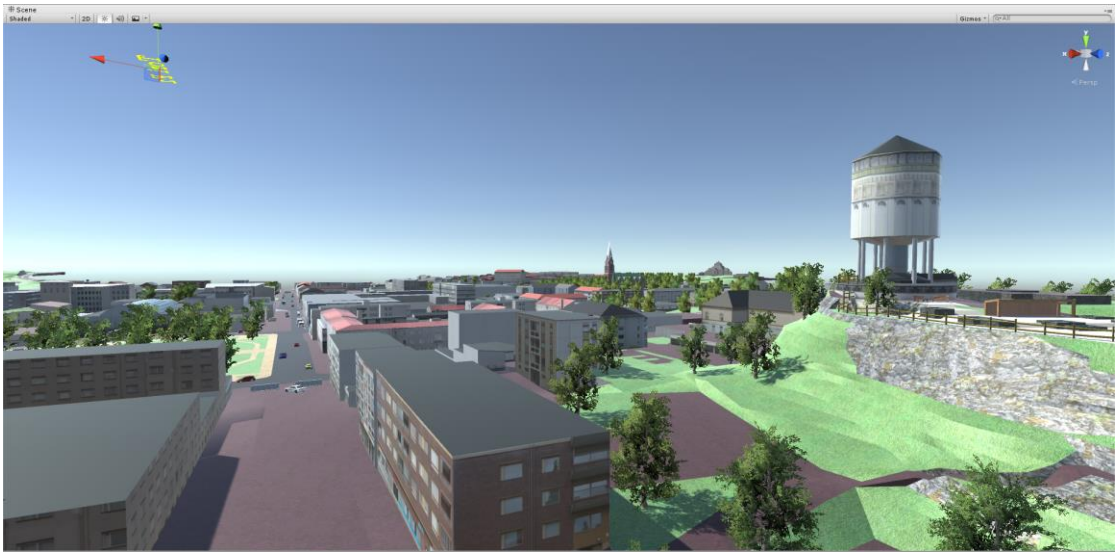
```
public void MihinMinunPitaa_Enter()
{
    lainakeskustelu = GameObject.Find("tormaystunnistus");
    lainakeskustelu.GetComponent<tormaysLainaneuvoja>().keskusteluvalmius = true;

    XDocument Palaute = new XDocument ();
    XElement KysymyksetElementti = new XElement("Kysymykset");

    int i;
    //Kysymys- ja vastauslistat käydään läpi ja niistä muodostetaan XML-tiedosto
    for (i = 0; i < Kysymykset.Count; i++) {
        XElement Kysymys = new XElement ("Kysymys");
        Kysymys.Add (new XElement ("Kysymysteksti", Kysymykset [i]));
        Debug.Log ("kokeilua : " + Kysymykset [i]);
        Kysymys.Add (new XElement ("Vastausteksti", Vastaukset [i]));
        KysymyksetElementti.Add (Kysymys);
    }
    Palaute.Add (KysymyksetElementti);
    Palaute.Save ("Palaute.xml");

}
```

KUVA 16. XML-tiedoston rakennus ohjelmointiskriptin avulla

Pelaaja on tässä vaiheessa päässyt palautekeskusteluun saakka pelin juonessa. Tämä toimii myös epäsuorana palautteena siitä, että edelliset juonen tapahtumat ovat onnistuneet pelaajalta ongelmitta. XML-tiedostoon voitaisiin myös kerätä epäsuoraa palautetta pelaajan toiminnoista. Erilaisten juonen kannalta merkityksettömien interaktioiden kerääminen XML-tiedostoon palautekeskustelun lisäksi voisi olla hyvä vaihtoehto pelin jatkokehityksen kannalta. Tässä vaiheessa toteutetun MVP:n idea oli kuitenkin testata palvelutilanteen ydinmekaniikkaa eli dialogien toimivuutta.

4.2 Kaupunkikäyttöliittymän ja palvelutilanteiden yhdistäminen

Kirjoitushetkellä lopullisen työelämäpelin hahmonmuokkauseditori ei ollut vielä valmis, joten eri osa-alueiden yhdistämiseen saatiin mukaan vain palvelutilanteita ja Mikkelin 3D-mallia kuvaavat projektit. Mikkelin 3D-mallista luotu käyttöliittymä toimii pelin aloitusmaailmana. Kaupungilla seikkaillessa pelaaja voi siirtyä eri puolilla kaupunkia sijaitseviin palvelutilanteisiin, joissa pelaaja pääsee suorittamaan erilaisia tehtäviä. Kahden erikseen kehitetyn projektin yhdistäminen vaatii tiedostojen tuomista projektista toiseen sekä eri Scene-tiedostojen yhdistämistä ohjelmoinnin avulla. Aion tästä lähin viitata projekteihin selkeyden vuoksi nimillä palvelutilanneprojekti ja kaupunkiprojekti.



KUVA 17. Kuva Mikkelin 3D-mallia hyödyntävästä kaupunkiprojektista

Tiedostojen siirtäminen projektista toiseen tapahtuu luomalla Unity3D:n käyttämiä paketteja. Tässä tapauksessa ei ollut väliä kumpaan projektiin toisen projektin tiedostot siirrettäisiin, koska lopullinen yhdistäminen vaatii vielä kolmannenkin projektin. Palvelutilanneprojekti oli kooltaan pienempi kuin kaupunkiprojekti, joten sen tiedostojen siirtäminen kaupunkiprojektiin oli nopeampi vaihtoehto. Scene-tiedosto sisältää kaikki siinä käytettävät objektit. Pakettia luodessa voi kuitenkin olla vaikeaa löytää kaikki tarvittavat assetit joita kyseinen Scene käyttää. Oikeaa hiiren näppäintä klikkaamalla halutun scene-tiedoston päällä voidaan valita select dependencies -vaihtoehto. Select dependencies valitsee kaikki assetit, joita valittu Scene käyttää. Tällä tavoin paketin luomista varten voidaan valita vain tarvittavat assetit, koko asset-kirjaston valitsemisen sijaan. Palvelutilanteeseen tarvittavista asseteista luotiin paketti, joka tuotiin kaupunkiprojektiin.

KUVA 18. Palvelutilanteen sisäänkäynti

Seuraavaksi kaupunkiprojektiin piti lisätä paikka, josta pelaaja pääsee siirtymään palvelutilanteeseen. Etsimme kaupunkiprojektista sijainnin, joka tässä tapauksessa toimi työvoimatoimiston sisäänkäyntinä. Lisäsimme oven eteen vihreän tason Unity3D:n primitiivistä tasopeliobjektia hyödyntäen sekä tekstin sen yläpuolelle. Taso toimii tässä vaiheessa placeholderina, joka on selkeästi nähtävissä pelaajan liikkuessa kaupungilla. Lisäsimme tason yläpuolelle myös näkymättömän peliobjektin Collider-komponentilla, joka laukaisee Scene-tiedostosta toiseen siirtymisen. Törmäystunnistus toteutettiin pelaajahahmolle liitetyn Tag-ryhmän kautta. Collider tunnistaa pelihahmon ja ohjelmointiskriptiin liitetty koodin osa siirtää pelaajan palvelutilanteeseen.

5 PÄÄTÄNTÖ

Opinnäytetyöni tavoitteena oli suunnitella ja toteuttaa toimiva prototyyppi Digiosaajaksi työelämään -hankkeen työelämäpelin palvelutilanteesta. Palvelutilanteesta piti luoda sellainen, että sen avulla voidaan testata pelin ydinmekaniikkojen toimivuutta ja luoda pohja pelin jatkokehitykselle. Dialogien luomisessa käytetyn Cradle-lisäosan so-

veltuvuudesta projektiin ei ollut varmuutta, joten siihen liittyvät tutkimusongelmat olivat samaan aikaa mielenkiintoisia ja haastavia. Projektin tekeminen aloitettiin tyhjästä, joten vastuu käytännön projektista oli suuri. Se minkälainen prototyypistä tuli vaikuttaa vahvasti seuraavien jatkokehityksen parissa työskentelevien ideoihin ja toimintatapoihin. Toisaalta se, että pelistä ei ollut valmista mielikuvaa, antoi suunnittelulle enemmän tilaa. Erilaisia ideoita ja toteutustapoja sai kehitellä rauhassa ja lopullisen pelin tyyli oli vahvasti tekijöidensä käsissä.

Opinnäytetyö oli prosessina positiivinen kokemus. Pelit ovat olleet merkittävässä roolissa läpi elämäni, joten kiinnostus myös pelien kehitystä kohtaan oli suuri. Unity3D oli pelimoottorina minulle tuttu, mutta sen tarkasteleminen tästä näkökulmasta oli itselleni uutta. Opinnäytetyöni kautta pääsin perehtymään tarkemmin pelisuunnittelun eri vaiheisiin ja se toi mukanaan paljon erilaisia näkökulmia. Mielestäni prototyyppi oli onnistunut ja vastasi toimeksiantoa. Ydinmekaniikat ovat hyvin esillä ja palautetta pelaajilta saadaan eri muodoissa.

Digiosaajaksi työelämään -hankkeen työelämäpelille on tässä vaiheessa rakennettu sen ydinmekaniikat ja runko, jonka päälle tullaan rakentamaan paljon lisäominaisuuksia. Pelillä on hyvät mahdollisuudet kehittyä opetusvälineenä moneen eri suuntaan. Palvelutilanteita voidaan kehitellä tarpeen mukaan lisää, ideaa voidaan viedä muihin kaupunkeihin ja pelin rakennetta voidaan soveltaa muihin käyttötarkoituksiin. Pelin kehitys jatkuu oman osallistumiseni jälkeen ja tulevaisuudessa esimerkiksi moninpelituki tai VR-laitteiden hyödyntäminen vahvistaisi pelin immersiota entisestään. Pelin kokonaisuuden ympärille voidaan myös kehittää suurempaa juonta, joka hyödyntäisi useita palvelutilanteita.

LÄHTEET

Digiosaajaksi työelämään. 2016. Digiosaajaksi työelämään ESR- hanke. WWW-dokumentti. http://www.digityo.fi. Luettu 10.10.2016.

Rogers, Scott 2014. Level Up! The Guide To Great Video Game Design.West Sussex: John Wiley and Sons.

Schell, Jesse 2015. The Art Of Game Design: A Book Of Lenses. Florida: Taylor & Francis Group.

Ries, Eric 2011. The Lean Startup. New York: Crown Publishing Group.

Making Your First Game: Minimum Viable Product. 2016. Extra Credits. WWW-dokumentti. https://www.youtube.com/watch?v=UvCri1tqIxQ. Luettu 26.10.2016.

Setälä, Risto. 2012. Nopeammin, ketterämmin, edullisemmin. WWW-dokumentti. http://www.tekes.fi/nyt/blogit_2012/nopeammin-ketterammin-edullisemmin. Päivitetty 5.9.2012. Luettu 25.10.2016.

Minumum Viable Product. 2016. SyncDev. WWW-dokumentti. http://www.syncdev.com/minimum-viable-product. Luettu 25.10.2016.

Zaninotto, Francois. 2016. Lean Startup. WWW- dokumentti. http://marmelab.com/blog/2016/02/12/build-measure-learn. Päivitetty 12.2.2016. Luettu 14.11.2016.

Makabee, Hayim. 2015. Effective Software Design. WWW- dokumentti. https://effectivesoftwaredesign.com/2015/12/10/invention-is-not-enough-for-innovation. Päivitetty 10.12.2015. Luettu 14.11.2016.

Davis, Ash. 2016. Version Control: Effective use, issues and thoughts, from a gamedev perspective. WWW-dokumentti. http://www.gamasutra.com/blogs/AshDavis/20161011/283058/Version_control_Effective_use_issues_and_thoughts_from_a_gamedev_perspective. Päivitetty 11.10.2016. Luettu 7.11.2016.

A Visual Guide to Version Control. 2007. Better Explained. WWW-dokumentti. https://betterexplained.com/articles/a-visual-guide-to-version-control. Päivitetty 27.9.2007. Luettu 7.11.2016.

Fast Facts. 2016. Unity Technologies. WWW-dokumentti. https://unity3d.com/public-relations. Luettu 26.10.2016

Unity Personal. 2016. Unity Technologies. WWW-dokumentti. https://store.unity.com/products/unity-personal. Luettu 26.10.2016.

Made With Unity. 2016. Unity Technologies. WWW-dokumentti. https://madewith.unity.com. Luettu 26.10.2016.

Scenes. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/CreatingScenes. Luettu 26.10.2016.

Asset Workflow. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/AssetWorkflow. Luettu 26.10.2016.

Primitive and Placeholder Objects. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/PrimitiveObjects. Luettu 26.10.2016.

Component. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/ScriptReference/Component. Luettu 26.10.2016.

Asset Packages. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/AssetPackages. Luettu 26.10.2016.

Prefabs. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/Prefabs. Luettu 26.10.2016.

Creating and Using Scripts. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. Luettu 1.11.2016.

Variable and the Inspector. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/VariablesAndTheInspector. Luettu 1.11.2016.

Rigidbody Overview. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/RigidbodiesOverview. Luettu 1.11.2016.

Colliders Overview. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/CollidersOverview. Luettu 1.11.2016.

Unity 4.6. 2015. Unity Technologies. WWW-dokumentti. https://unity3d.com/unity/whats-new/unity-4.6. Luettu 2.11.2016.

Canvas. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/UICanvas. Luettu 2.11.2016.

User Interfaces for VR. 2016. Unity Technologies. WWW-dokumentti. https://unity3d.com/learn/tutorials/topics/virtual-reality/user-interfaces-vr. Luettu 2.11.2016.

Creating immersive experiences with diagetic interfaces. 2010. Cooper. WWW-dokumentti. http://www.cooper.com/journal/2010/08/diagetic_interfaces. Luettu 2.11.2016.

Visual Components. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/UIVisualComponents. Luettu 2.11.2016.

Interaction Components. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/UIInteractionComponents. Luettu 2.11.2016.

Console Window. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/Console. Luettu 2.11.2016.

Debug.LogError. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/ScriptReference/Debug.LogError. Luettu 2.11.2016.

Debug.LogWarning. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/ScriptReference/Debug.LogWarning. Luettu 2.11.2016.

Debug.Log. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/ScriptReference/Debug.Log. Luettu 2.11.2016.

MonoDevelop's Debugger. 2016. Unity Technologies. WWW-dokumentti. https://unity3d.com/learn/tutorials/topics/scripting/monodevelops-debugger. Luettu 2.11.2016.

Twine. 2016. Twine. WWW-dokumentti. http://twinery.org. Luettu 3.11.2016.

Cradle. 2016. Github. WWW-dokumentti. https://github.com/daterre/Cradle. Luettu 3.11.2016.

Top 10 Unity Best Practices: Naming Conventions. 2015. Xeushack. WWW-dokumentti. http://dev.xeushack.com/top-10-unity-best-practices-naming-conventions-part-1. Päivitetty 3.3.2015. Luettu 3.11.2016.

Tulleken, Herman. 2012. 50 Tips for Working in Unity. WWW-dokumentti. http://devmag.org.za/2012/07/12/50-tips-for-working-with-unity-best-practices. Päivitetty 12.7.2012. Luettu 10.11.2016.

Tags. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/Tags. Luettu 10.11.2016.

Layers. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/Manual/Layers. Luettu 10.11.2016.

Tooltip Attribute. 2016. Unity Documentation. WWW-dokumentti. https://docs.unity3d.com/ScriptReference/TooltipAttribute. Luettu 10.11.2016.

3D Warehouse Terms of Use FAQ. 2016. SketchUp. WWW- dokumentti. https://help.sketchup.com/en/article/3000049. Luettu 15.11.2016.

SketchUp. 2016. Unify Community Wiki. WWW- dokumentti. http://wiki.unity3d.com/index.php?title=SketchUp. Luettu 15.11.2016.