Kolmiulotteisen pelin suunnittelu ja prototypointi Unity-pelimoottorilla

Jeremias Kontinen

Haaga-Helia ammattikorkeakoulu

Tradenomi

Opinnäytetyö

2022

Tiivistelmä

Tekijä(t)
Jeremias Kontinen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Kolmiulotteisen pelin suunnittelu ja prototypointi Unity-pelimoottorilla

Sivu- ja liitesivumäärä
22

Tässä opinnäytetyössä käydään läpi pelisuunnittelun teoriaa ja miten tämän teorian päälle saadaan rakennettua prototyyppi pelille. Tekijällä ei ole aikaisempaa kokemusta pelisuunnittelusta, joten sitä käsittelevä teoria on suunnattu henkilöille, joilla on sama kokemustaso. Työn tavoitteena on ymmärtää, kuinka suunnitella peli-idea, ja miten tästä peli-ideasta rakennetaan tarkoituksenmukainen prototyyppi. Empiirisessä osiossa kuvataan itsenäisen peliprojektin kehitys peli-ideasta prototyypiksi.

Teoria suurimmaksi osaksi keskittyy pelimekaniikkoihin, mutta sisältää myös asiaa siitä, miksi pelaajat pelaavat pelejä. Teoria alkaa yleishyödyllisillä pelisuunnittelun työkalulla, MDA-viitekehyksellä, ja jatkaa tästä pelisysteemeihin ja niiden ymmärtämiseen. Tämän jälkeen käydään läpi erilaisten pelien antia ja erilaisia pelaajatyyppejä. Teoriassa käydään läpi myös, miten lähteä suunnittelemaan uutta peli-ideaa ja miten ideoida pelimekaniikkoja tähän. Teorian lopussa käydään myös läpi, mitä virtuaaliselta prototyypiltä odotetaan ja mitä sen tavoitteet ovat.

Toiminnallisessa osassa tekijä rakentaa prototyypin peli-idealleen. Peli-idea on kolmiulotteinen ensimmäisen persoonan seikkailupeli, joka päätettiin toteuttaa Unity-pelimoottorilla. Tekijä käy läpi koodiaan, jolla sai prototyypissä olevat toiminnallisuudet toimimaan. Kun prototyyppi on osoittanut peli-idean joko toimivaksi tai puutteelliseksi, on se projektin kannalta valmis. Tässä opinnäytetyössä ei kehitetä peli-ideaa prototyypin jälkeen eikä suunnitella kokonaista peliä prototyypistä.

Opinnäytetyössä valmistettava tuotos on ensimmäisen persoonan seikkailupelin prototyyppi. Prototyypin tekeminen sujui ilman suurempia hankaluuksia. Prototyyppi päätyy osoittamaan ongelmakohtia peli-ideassa, jotka vaatisivat peli-idean uudelleen suunnittelua. Tekijällä ei ole suunnitelmia jatkaa peli-idean jalostamista tämän opinnäytetyön jälkeen.

Asiasanat
Pelisuunnittelu, Unity, Prototyyppi, Pelimekaniikat

Sisällys

1 Johdanto 1
2 Pelisuunnittelun teoriaa 2
2.1 MDA-viitekehys 2
2.2 Pelien systeemit 3
2.3 Pelien anti 4
2.4 Eri pelaajatyyppejä 4
2.5 Uuden pelin suunnittelu tyhjästä 6
2.6 Olennaiset pelimekaniikat 7
2.7 Prototypointi 7
3 Pelin rakentaminen 9
3.1 Peli-idea ja suunnittelu 9
3.2 Prototyypin aloittaminen Unityllä 10
3.3 Pelihahmon liikkeen luonti 12
3.4 Lintuperspektiivin lisääminen 14
3.5 Salamakyvyn tekeminen 15
3.6 Prototyypin päättäminen 18
4 Pohdinta 20
Lähteet 21

1 Johdanto

Videopelejä, joissa pelihahmoa ohjataan ensimmäisestä persoonasta, on ollut jo vuosikymmeniä. Itseäni ne ovat aina kiehtoneet sillä ne ovat immersioisempia kuin kolmannesta persoonasta kuvatut pelit, jossa pelaaja näkee pelihahmon takaviistosta. Ensimmäisen persoonan kuvakulma on erittäin käytetty erityisesti ammuntapeleissä, kuten esimerkiksi Call of Duty -pelisarjassa, mutta on myös yleinen kolmiulotteisissa Indie-peleissä. Ensimmäisen persoonan kuvakulma Indie-peleissä on tarkoittanut vähemmän työtä pelihahmon mallintamisessa ja animoinnissa.

Tässä opinnäytetyössä keskitytään ensimmäisen persoonan seikkailupelin prototyypin tekemiseen käyttämällä Unity-pelimoottoria. Keskityn erityisesti, miten pelisuunnittelijan kannattaa lähteä suunnittelemaan peli-ideaa ja miten tästä peli-ideasta kehitetään valmis prototyyppi. Prototyypin tarkoituksena on selvittää peli-idean kelpoisuus ja hahmottaa, onko pelin jatkokehittäminen mahdollista. Projektista ei ole tarkoitus tulla muuta kuin prototyyppi, joka vastaa asettamiini kysymyksiin peli-ideasta, joita on vaikea, ellei mahdoton, tietää ilman prototyyppiä.

Pelin ajatuksena on olla metroidvania-tyylinen peli. Metroidvania-peleissä pelihahmo seikkailee pelimaailmassa ja kerää taitoja ja tavaroita, joilla pelaaja pystyy avaamaan uusia alueita pelimaailmassa. (Stegner 19.10.2021) Kaksiulotteiset metroidvania-pelit ovat olleet erittäin suosittuja indie-pelikehittäjien keskuudessa jo yli 10 vuotta (Yu 9.2.2021), mutta nykypäivän pelimoottorien tehokkuus ja helppokäyttöisyys pitäisi saada vastaavien kolmiulotteisten pelin kehitys mahdolliseksi ilman suuria pelistudioita.

Toivon projektin parantavan omaa käsitystäni videopelien tekemisestä ja suunnittelusta. Lisäksi projektin tulisi täsmentää ajatuksiani siitä, olenko vielä innokas tekemään pelejä tulevaisuudessa työnä, vai sopisiko jokin muu IT-alan osa-alue minulle ammatillisesti paremmin.

Käsitteet

Indie-peli	Peli, joka on tuotettu ilman julkaisijan taloudellista tukea (Vito Oddo 2.8.2021)
MDA	Mekaniikat-Dynamiikat-Estetiikat viitekehys (Sellers 2018, 92-93)
Metroidvania	Toimintaseikkailupelien alalaji, jossa pelaaja kerää kykyjä edetäkseen epälineaarisessa pelimaailmassa (Stegner 19.10.2021)
Pelimekaniikka	Elementtejä, joiden kanssa pelaaja ja peli ovat vuorovaikutuksessa (Sicart 2008)
Pelisuunnittelu	Peli-ideaan liittyvä suunnittelu
Prototyyppi	Ensimmäinen versio jostakin, jonka tarkoitus on usein selvittää konseptin toimivuus
Unity	Yksi suosituimmista pelimoottoreista (Dealessandri 16.1.2020)

2 Pelisuunnittelun teoriaa

2.1 MDA-viitekehys

Ensimmäinen ja mahdollisesti parhaiten tunnettu pelien viitekehys on Mekaniikat-Dynamiikat-Estetiikat (engl. Mechanics-Dynamics-Aesthetics, lyhyesti MDA), jossa näille termeille on omat määritelmänsä. Mekaniikoilla tarkoitetaan pelin komponentteja algoritmien ja datan kuvauksen tasolla. Dynamiikalla tarkoitetaan pelin systeemejä, mekaniikkojen ajonaikaista käyttäytymistä pelaajan syötteeseen ja mekaniikkojen keskinäistä vuorovaikutusta. Estetiikalla yleensä tarkoitetaan visuaalisia estetiikkoja, mutta tässä kontekstissa sillä tarkoitetaan haluttua tunnereaktiota, jonka peli herättää pelaajassa ja koko kokemusta pelistä. (Sellers 2018, 92-93)

MDA:n mukaan pelisuunnittelija ja pelaaja näkevät pelin eri tavalla. Suunnittelija aloittaa pelin suunnittelun estetiikasta, ja miettii mitä tunteita hän haluaa pelin herättävän pelaajassa, ja työskentelee siitä alaspäin mekaniikkoihin. Pelaaja puolestaan ensin näkee pelin mekaniikat, joiden kanssa hän on vuorovaikutuksessa, ja näiden kautta löytää pelin dynamiikkaa ja lopulta saavuttaa jonkinlaisen tunnepohjaisen reaktion. (Gibson 2014, 20-21)

Pelisuunnittelija ei voi luoda pelin dynamiikkoja eikä estetiikkaa kuin välillisesti muokkaamalla pelin mekaniikkoja. Pelin mekaniikkoja muokkaamalla muuttuu myös pelin dynamiikka ja tätä kautta estetiikka. Pelisuunnittelija joutuu miettimään tarkkaan, miten muuttaa mekaniikkoja saadakseen halutun lopputuloksen. Kaikille peleille ja pelisuunnittelijoille samat tavat eivät kuitenkaan toimi. Joillekin pelisuunnittelijoille voi sopia paremmin suunnitella peli aloittaen mekaniikoista ja päätymällä johonkin estetiikkaan, kun taas toisille miettiä, mitä estetiikkaa peliltään haluavat ja rakentaa alla olevat mekaniikat niin, että haluttu estetiikka syntyy. (Sellers 2018, 92-93)

MDA-viitekehystä on kritisoitu termeistä, joita se käyttää. Näille termeille on jo aikaisempia merkityksiä ja keskustellessa on helppo jäädä kiinni semantiikkaan, kun eri ihmiset tarkoittavat eri asioita samoilla sanoilla. Lisäksi termit itsessään on määritelty aika löysästi MDA-viitekehyksessä, esimerkiksi jotkut näkisivät isompiakin kokonaisuuksia yhtensä mekaniikkana, ja toiset voisivat pilkkoa sen paloiksi ja kutsua näitä paloja mekaniikoiksi. Näistä huolimatta MDA-viitekehys on hyödyllinen työkalu pelisuunnittelussa ja oikein käytettynä auttaa meitä ymmärtämään pelejä paremmin ja miten ihmiset kokevat niitä. (Sellers 2018, 92-93)

2.2 Pelien systeemit

Systeemit ovat kokonaisuuksia, jotka koostuvat pienemmistä vuorovaikutteisista osista. Kokonaisuudella on omat ominaisuutensa ja toimintonsa, jotka syntyvät koko systeemin vuorovaikutuksesta (Sellers 2018, 50). Esimerkkinä systeemistä pelissä voisi olla pelin talous. Hyvillä systeemeillä peleissä on muutamia ominaisia piirteitä, joita on hyvä tiedostaa. Systeemien pitää olla ymmärrettävissä sekä pelisuunnittelijalle ja pelaajalle, ja pelaajan olisi hyvä pystyä rakentamaan päässään malli, miten pelin systeemit toimivat. Pelien sääntöjen ja sisällön tulee olla johdonmukaisia. Vaikka olisikin houkuttelevaa välillä lisätä yksittäisiä poikkeuksia peliin, yleensä se vain heikentää systeemien joustavuutta. Pelin systeemien pitäisi antaa ennakoitavissa olevia lopputuloksia pelaajan syötteelle. Lisäksi pelin systeemien pitäisi olla monikäyttöisiä ja elegantteja. (Sellers 2018, 175-176)

Lautapelit ovat hyviä harjoituksia hyvien pelisysteemien suunnitteluun, sillä kaikki laskentateho käytettävissä ovat vain pelaajat itse. Lisäksi pelin vuorovaikutusten pitää toimia tavoilla, jota pelaajat voivat fyysisesti muokata ja manipuloida. Siinä missä videopelit voivat piilottaa paljon asioita visuaalisten ja tarinallisten esitysten taakse, lautapelit eivät voi tehdä samaa ja kaikki pelin mekaniikat ovat pelaajien näkyvillä. (Sellers 2018, 176-177)

Raph Kosterin mielestä kaikki vuorovaikutteiset systeemit perustuvat oppimiseen. Kun teemme yhteyden, jota emme tajunneet aikaisemmin, aivot vapauttavat kemikaaleja, jotka auttavat meitä muistamaan tämän, ja saavat meidät tuntemaan hyvältä. Tämä oppiminen on Kosterin mukaan pääsyy miksi pelit ovat hauskoja. Vuorovaikutteiset systeemit tarjoavat turvallisen ympäristön oppimiselle. Keith Burgunin mukaan on yksi syy, miksi pelaaja kyllästyy systeemeihin. Se, että pelaajasta tuntuu, että jäljelle jäävän ymmärryksen saavuttaminen on liian vaikeaa. Tämä voi olla joko tilanne missä pelaaja on ratkaissut systeemit jo, jolloin pelissä ei ole mitään opittavaa enää, tai missä uuden oppiminen on niin vaikeaa enää, että se ei vaikuta vaivan arvoiselta. Keith Burgun käyttää lautapeli Go:ta esimerkkinä pelistä missä pelaajat eivät ole lähelläkään ratkaista peliä, mutta uuden oppiminen tuntuu turhalta, koska pelin eri mahdollisuudet ovat ylivoimaiset. (Burgun 2015, luku 1)

2.3 Pelien anti

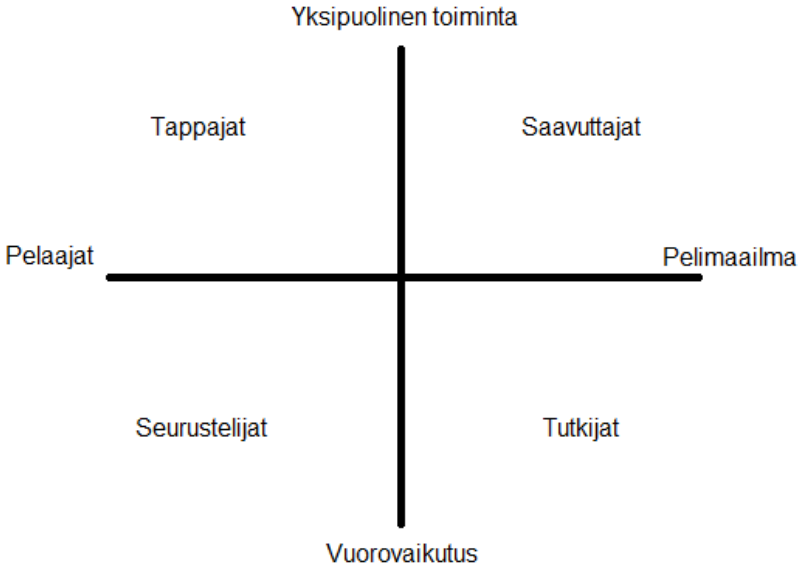
Pelien pelaamisen on yleensä tarkoitus olla kivaa tai hauskaa. Pelaajien näkemys siitä, mikä on kivaa tai tyydyttävää vaihtelee. Pelissä, joka tähtää pelaajalle helppoon nautintoon, olisi esimerkiksi mahdollista tutkia alueita tai toteuttaa itseään luovasti. Koska pelin ei ole tarkoitus vaatia pelaajalta paljoa, pelin voi olla vaikea ylläpitää pelaajan mielenkiintoa. Toisenlaisia tunteita herättävät sen sijaan pelit, jotka yrittävät haastaa pelaajaa tekemään jotain, joka voi aluksi tuntua mahdottomalta esteeltä. Mutta pelaajan oppiessa pelin systeemejä haaste, joka tuntui aluksi ylivoimaiselta, voikin olla tavoitettavissa. Vakavampia pelejä voivat olla simulaatio- tai oppimispelit, joissa tarkoituksena on saada jotain muutakin irti pelistä kuin vain hetkellistä ajanvietettä. (Lazzaro 2009, 20-22)

Nicolle Lazzaron (2009, 30-32) mukaan pelit kuten Myst, vuonna 1991 julkaistu seikkailu- ja pulmapeli, luovat helppoa hupia antamalla pelaajan selvittää asioita. Sen sijaan, että peli laskisi pisteitä se antaa pelaajan tutkia alueita ja pähkäillä itse mitä on meneillään. Tällaisissa peleissä usein matka itsessään on palkinto. Lisäksi helppoa hupia voi saada esimerkiksi täyttämällä pelaajan fantasioita, joita ei pysty täyttämään oikeassa elämässä.

Sen sijaan, että jakaisi huvin helppoon tai vaikeaan, Chris Sniezak (2016) jakaisi huvit vähän eri tavalla esimerkiksi fantasiaan, tarinalliseen ja ilmaisulliseen. Sniezak kertoo, että hänen kaverinsa Walt Ciechanowski tykkää sanoa, että ei ole olemassa huonoa, väärää hupia. Mistä ihmiset pitävät on kuitenkin subjektiivista, joten on hyvä olla jotain ajatuksia siitä, mihin osa-alueeseen on suunnittelemassa peliä.

2.4 Eri pelaajatyyppejä

Richard A. Bartlen pelaajatyyppien malli vuodelta 1996 on tänäkin päivänä yleinen malli, jota käytetään ymmärtämään mitä pelaajatyyppejä virtuaalisissa maailmoissa on ja miksi eri pelaajatyypit tekevät asioita pelimaailmoissa. Vaikka varsinaisesti malli keskittyy massiivisiin moninpeleihin, on muidenkin pelien tärkeä tiedostaa mallin esille tuomia asioita. Richardin mukaan on 2 akselia, jotka leikkaavat pelaajat 4 eri osa-alueeseen sen mukaan, miksi he pitävät niistä asioista, joita he peleissä tekevät (Kuvio 1). Ensimmäinen akseli koskee muita pelaajia ja maailmaa; onko pelaaja enemmän kiinnostunut muista pelaajista pelimaailmassa vai pelimaailmasta itsestään. Toinen akseli on kuinka he toimivat pelimaailmassa, joko enemmän painottaen omaa toimintaansa tai vuorovaikutusta pelimaailman ja/tai muiden pelaajien kanssa. (Bartle 2009, 121-124)



Kuvio 1. Richard A. Bartlen pelaajatyyppien mallin pelaajien kiinnostuskaavio

Kun pelaajat pelaavat peliä ilman ulkopuolisia syitä (esim. rahallinen korvaus), Richardin mukaan ei ole niinkään olennaista mitä pelaajat tekevät pelimaailmassa. Tärkeämpää on seurata miksi he tekevät niitä asioita mitä he tekevät. Tästä voidaan saada selville heidän pelaajatyyppiään, ja hyvä virtuaalimaailma Richardin mukaan sisältäisi kaikkia 4 eri pelaajatyyppiä ja pitäisi huolen, että kaikille löytyy mielekästä tekemistä. (Bartle 2009, 125-130)

Dixon (2011, 2) kritisoi mallia muutamasta syystä. Ensinnäkin pelaajat eivät välttämättä täytä kaikkia tietyn pelaajatyypin piirteitä. He voivat myös täyttää useita yhden pelaajatyypin piirteitä mutta olla silti sopimatta yläkäsitteeseen. Lisäksi pelaajat voivat olla osittain yhtä ja osittain toista pelaajatyyppiä, mutta Bartlen malli ei kata tällaisia henkilöitä.

Pelimaailma, josta puuttuu sosiaalinen kanssakäynti muiden pelaajien kanssa ei tietenkään täytä perusvaatimusta tällaisen mallin soveltamiseen täysin, mutta mallin muita huomioita on silti hyvä havaita. Tässä opinnäytetyössä luodussa pelin prototyypissä ei ole moninpeliä ja koska pelin laajuus on huomattavasti pienempi kuin massiivisessa moninpelissä, voi peli helposti keskittyä huolehtimaan vain yhdestä pelaajatyypistä, mikä tässä peliprojektissa olisi seikkailija-pelaajatyyppi.

Jeremy Gibson listaa useita yleisiä tavoitteita mitä hän on nähnyt pelisuunnittelijoilla tehdessä peliä. Nämä tavoitteet voi jakaa kahteen puoleen osaan: suunnittelijakeskeiset tavoitteet ja pelaajakeskeiset tavoitteet. Suunnittelijakeskeisiä tavoitteita on, jossa pelisuunnittelija itse saa jotain pelin tekemisestä, kuten rahaa, mainetta, ja kokemusta, kun taas pelaaja keskeisiä tavoitteita on missä pelin pelaaja saisi jotain pelistä, kuten hupia, haasteita tai voimaantumista. (Gibson 2014, 106-108)

2.5 Uuden pelin suunnittelu tyhjästä

Ernest Adams ja Joris Dormans (2012, luku 1) suosittelee aloittamaan pelin suunnittelun mekaniikoista, sillä pelin säännöistä on vaikea sanoa, kuinka sen pelattavuus on. Ainoa tapa tietää toimivatko pelin mekaniikat on pystyä kokeilemaan niitä. Tämän takia aikaiset prototyypit ovat tärkeitä. Lisäksi pienikin muutos pelin mekaniikkoihin voi vaikuttaa peliin huomattavasti ja pilata pelin systeemien herkän tasapainon, joten pelin keskeisimpiin mekaniikkoihin ei toivottavasti tarvitsisi tehdä muutoksia myöhään pelin kehityksessä.

Pelisuunnittelija Kyle Gabler piti puheenvuoron Global Game Jam:issa 2009, jonka ajatuksena oli, että kun alkaa tehdä uutta peliä, tekisi pienen lelun tai leikkikentän, jossa voi kokeilla pelin keskeisiä mekaniikkoja. Tämä tarkoittaa, että aloittaisi pelinkehityksen pienestä prototyypistä, jossa on vain pelin keskeiset mekaniikat ilman erityisempää grafiikkaa, tavoitteita tai nokkelaa kenttäsuunnittelua. Tämän prototyypin kanssa leikkiminen pitäisi itsessään olla hauskaa ja mielenkiintoista, jonka jälkeen voi kehittää pelin tämän ympärille. (Adams & Dormans 2012, luku 1)

Esimerkkinä lelu tyyppisestä prototyypistä Keith Burgun ottaa jojo lelun. Jojo on vuorovaikutteinen ja sitä voi ponnahdella ylös ja alas, sekä tehdä useita eri temppuja sen kanssa, ja kaikkien näiden mahdollisten temppujen ja käyttötapojen etsiminen on kiinnostavaa. Lelulla leikkimisellä löytää mahdollisuudet mitä sillä voi tehdä, ja mitä ei voi tehdä, ja näin myös tällaisen pelin prototyypistä voi saada kuvan onko pelin mekaniikat itsessään hyvät ja mielenkiintoiset, ja mihin suuntaan lähteä peliä toteuttamaan. (Burgun 2015, luku 1)

Keith Burgunin (2015, luku 1) mukaan seuraava askel tällaisesta lelusta olisi pulma; lelu, jolla olisi ratkaisu. Binäärisen tavoitteen tai ongelman esittäminen lelusta tuottaa tällaisen pulman ja pelaaja joko onnistuu ratkaisemaan sen, tai ei. Koska pulman tavoitteena on jokin ratkaisu, tämän ratkaisun löytämisen jälkeen pulmalla ei ole kauheasti jälleenpelaamisarvoa. Pelisuunnittelijan on tärkeä huomioida tämä piirre suunnitellessa pelejä tai pulmapelejä. Päätökset, jotka ovat hyviä pulmapeliä tehdessä ovat usein huonoja muille peleille, ja toisinpäin.

Vaikka jokainen peli lähtee jostain ideasta, on harvinaista, että alkuperäinen idea päätyy itse peliin. Varsinkin kun aloittaa pelisuunnittelun, ei pitäisi olla huolissaan, vaikka ei olisi täydellistä peli-ideaa. Hyvän peli-idean voi löytää matkan varrelta. Vaikka mielestään olisi keksinyt täydellisen peli-idean, ei sitä kannata heti alkaa takomaan peliä, vaan antaa sen kypsyä. Sen voi kirjoittaa ylös ja palata siihen myöhemmin. Jälkeenpäin peli-idea voi tuntua tyhmältä, tai vaihtoehtoisesti siihen voi saada uusia ideoita, jolloin sitä voi kehittää eteenpäin. (Catalin Marcu 9.12.2015)

2.6 Olennaiset pelimekaniikat

Pelistä on hyvä tunnistaa pelin olennaiset pelimekaniikat, ja niiden tarkoitukset. On vaikea sanoa suoraan mikä tekee hyvän olennaisen pelimekaniikan pelille. Keith Burgunin (2015, luku 2) mukaan on muutamia eri asioita, joita kannattaa kuitenkin miettiä. Hyvä pelimekaniikka olisi helppo selittää muille. Pelimekaniikka olisi hyvä olla myös selvä mekaaninen toiminto, joka tapahtuu pelissä. Hyvä pelimekaniikka olisi myös yksittäinen toiminto, eikä yhdistelmä useita toimintoja. Viimeisenä hän listaa moniulotteisuuden, jossa pelimekaniikka loisi syvyyttä pelille. Keith Burgunin (2015, luku 2) mukaan on kellokoneisto pelejä, joissa olennaiset pelimekaniikat selvästi ohjaavat pelaajaa kohti jotain tavoitetta. Tällöin pelaajan on helppo nähdä yhteys pelimekaniikkojen ja pelin tavoitteen välillä. Toinen laita on tilkkutäkkipelit, joissa useista olennaisista pelimekaniikoista on vaikea luoda johtopäätöksiä siitä, mikä pelaajan tavoite on pelissä. Tilkkutäkki suunnittelussa on vaarana, että pelin tärkeimmät pelimekaniikat eivät ole yhteydessä pelaajan tavoitteeseen.

Peleissä on myös auttavia pelimekaniikkoja, jotka avustavat näitä olennaisimpia pelimekaniikkoja toteuttamaan pelaajan tavoitteen pelissä, mutta myös luomaan lisää syvyyttä pelin strategialle ja taktiikalle. Esimerkkinä tällaisesta avustavasta pelimekaniikasta Keith Burgun tuo esille Shakissa 8x8 ruudun pelialueen, tai vuoropohjaisuuden, jossa pelaajat vuorottelevat kumman vuoro on tehdä siirto. Nämä avustavat pelimekaniikat tekevät Shakin olennaisimmista pelimekaniikoista toimivia ja mahdollistavat pelaajan tavoitteen; vihollisen kuninkaan matittamiseen. (Burgun 2015, luku 2)

Dustin Tyler (2021) esittelee muutamia yleisiä pelimekaniikkoja, joita useat pelit ovat käyttäneet. Ensimmäinen näistä on vuorot, jossa pelaajat pystyvät tehdä asioita vain omalla vuorollaan. Pelaajan vuorojen välissä voi olla joko muiden pelaajien vuoroja, tai esimerkiksi vihollisten pelivuoroja. Muita yleisiä pelimekaniikkoja on esimerkiksi pelaajan liike, toimintapisteet tai resurssienhallinta. Useat pelit myös antavat pelaajan itse sijoittaa pelimaailmaan asioita pelin edetessä, kuten erilaisia rakennuksia, hahmoja tai muita objekteja.

2.7 Prototypointi

Kaikkia pelisysteemejä ei pysty prototypoimaan fyysisesti, jolloin jäljelle jää virtuaalinen prototyyppi. Prototyypin ei välttämättä tarvitse olla tehty samalla pelimoottorilla tai tekniikalla kuin lopullinen peli, joten Adams ja Dormans (2012, luku 1) suosittelee avoimen lähdekoodin pelimoottoreita, kuten GameMaker ja Unity, prototyypin kehityksen nopeuttamiseksi. Virtuaalisissa prototyypeissä on etuna se, että pelin mekaniikoista saa hyvän näkemyksen, vaikka pelin grafiikka ja muut osat olisivat bugisia ja vaillinaisia. Virtuaalinen prototyyppi pystyy osoittamaan joitain pelin osia, kuten vuorovaikutteisia systeemejä, paremmin kuin sanat ja kuvaukset ikinä pystyisivät.

Jeremy Gibson (2014, 237-238) suosittelee Unityä itsenäisten pelin prototypointiin. Syynä tälle on se, että Unityn perusversio on ilmainen ja mahdollistaa pelien teon usealle eri alustalle. Lisäksi Unitylle on paljon tukea, kuten virallinen dokumentaatio ja paljon oppaita ja ohjeita käyttäjiltä. Joel Beardshaw (2020) sanoo heidän käyttäneen Unityä prototypoinnissa työskennellessään Ustwo Games:lla. He olivat myös käyttäneet ylimääräisiä ilmaisia työkaluja, jotka toimivat Unityn sisällä, nopeuttaen prototyypin rakentamista. Mitä työkaluja prototyyppiin tarvitsee, riippuu paljon peli-ideasta.

Adam Kramarzewskin ja Ennio De Nuccin (2018, alaluku What is a prototype?) mukaan prototyypin tarkoitus on antaa vastauksia kysymyksiin mitä suunnittelijalla on peli-ideasta. Mikäli pelisuunnittelijalla on jo vastaukset kysymyksiin muuta kautta, prototyyppi ei ole välttämättä välttämätön, mutta tämä on erittäin harvinaista. Kun prototyyppiä ei tehdä, on pelin lopullinen versio periaatteessa vain erittäin korkeabudjettinen prototyyppi, joka on tarkoitus myydä lopullisena versiona (Kramarzewski & De Nucci 2018, alaluku Digital prototyping). Prototyyppiin ei saa jäädä turhan kauaa myöskään kiinni. Kun prototyypin tarkoitus on täytetty, on tärkeä laittaa se sivuun ja aloittaa lopullisen pelin kehitys. Prototyypin valmistuttua sen koodia ei tulisi käyttää lopullisessa versiossa, vaan aloittaa se puhtaalta pöydältä hyvin suunniteltuna. Vaikka prototyyppi olisi toiminut hyvin, koska prototyyppien tarkoitus on eri kuin lopullisen pelin, ei sitä tulisi käyttää lähtöalustana koodille (Kramarzewski & De Nucci 2018, alaluku Using the prototype as a code-base for the production project). Kaikki aika mikä prototyypissä käytetään muuhun kuin suoraan vastauksien saantiin on hukattua aikaa, kuten animaatiot ja grafiikka. Lisäksi Adam huomauttaa suurimmasta sudenkuopasta prototypoinnissa, joka syntyy, kun pelisuunnittelija kokee, että hänellä on jo vastaukset ja haluaa rakentaa prototyypin osoittamaan tämän hinnalla millä hyvänsä. On tärkeä olla mieleltään avoin ja nähdä oman prototyypin ongelmat. Prototyyppi mikä näyttää, että jotain ei toimi on yhtä arvokas kuin prototyyppi, joka näyttää, että jokin toimii (Kramarzewski & De Nucci 2018, alaluku Seeking confirmations).

3 Pelin rakentaminen

En ole aikaisemmin käyttänyt Unity-pelimoottoria. Tästä johtuen suurin rajoittava tekijä prototyypille tulee olemaan omat taitoni Unityn käytöstä. Työllä ei ole toimeksiantajaa, joten pääsin itse päättämään käytettävistä kehitystyömenetelmistä ja aikataulusta. Projektin tuotoksen tulee olla peli prototyyppi, jossa peli-idea on esillä. Koska kyseessä on prototyyppi, jonka tarkoitus on selvittää peli-idean mahdollisia ongelmia, ei sen visuaalisella puolella ole juuri väliä. Prototyypistä ei itsessään tule olemaan hyötyä muille tahoille kuin minulle ilman, että sitä katsoo tämän opinnäytetyön kautta. Mikäli prototyyppi onnistuu näyttämään peli-idean joko toimivaksi, tai osoittamaan ongelmia, jotka tarvitsevat suurempaa muutosta sen suunnittelussa, on se täyttänyt tarkoituksensa. Näissä tilanteissa projektin lopputuotos voitaisiin katsoa onnistuneeksi.

3.1 Peli-idea ja suunnittelu

Peli-ideani on, että pelaaja seikkailee kolmiulotteisessa pelimaailmassa ensimmäisessä persoonassa. Pelaajalla on taikavoima, jolla hän pystyy pysäyttämään ajan ja näkemään ympärilleen lintuperspektiivissä. Tässä lintuperspektiivissä pelaaja pystyy merkata, minne haluaa käyttää kykyjä. Pelaaja tämän jälkeen pystyy ensimmäisessä persoonassa aktivoida näitä merkkejä eri vaikutuksilla. Ideana olisi, että pelaaja joutuu strategisesti miettimään etukäteen, mihin alueisiin haluaa iskeä taikavoimillaan, ja sitten ensimmäisessä persoonassa toteuttaa tämä suunniteltu strategia.

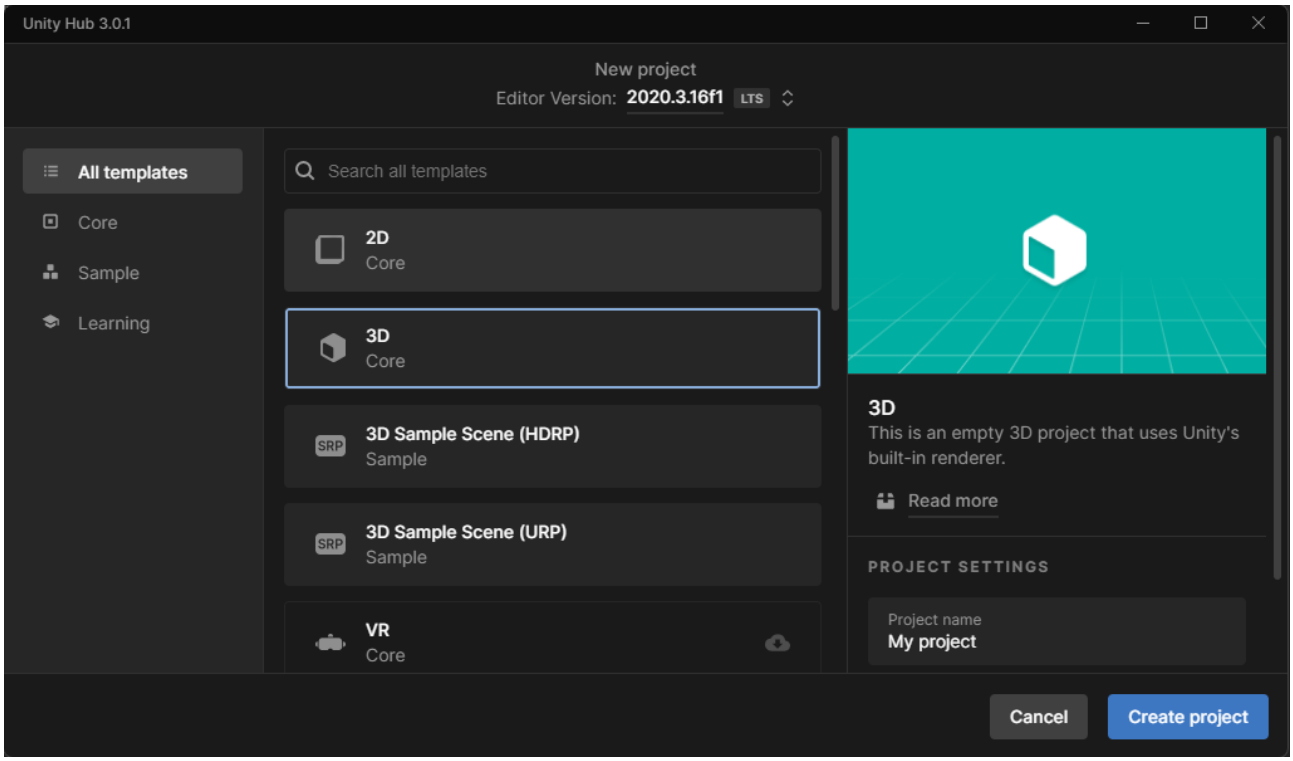
Pelin olennaisin osa olisi tämä ensimmäisen persoonan ja kolmannen persoonan välillä vaihtelu ja taikojen sijainnin merkkaaminen ja aktivointi. Peli olisi suunnattu pelaajatyypeille, jotka haluavat toteuttaa itseään peleissä ja tehdä asiat omalla tavallaan, ja myös heille, jotka haluavat seikkailla ja tutkia pelimaailmoja.

Tarvitsen prototyypiltäni vastauksia muutamaan kysymykseen, joihin en voi sitä kokeilematta vastata. Ensimmäinen on, että toimiiko tämä perspektiivin vaihto hyvin olennaisena osana peliä, jolloin pelaaja tekisi sitä jatkuvasti, ja suurin osa pelaajan toiminnallisuuksista ja kyvyistä on lukittu tämän taakse. Toinen on, että tuoko lintuperspektiivin käyttö mitään uusia mahdollisuuksia mitä en ole ajatellut. Kolmas on, että onko peli-idea itsessään tarpeeksi mielenkiintoinen ja täynnä mahdollisuuksia, vai tarvitaanko jotain muita mekaniikkoja lisäksi, kuten esimerkiksi taistelua vihollisia vastaan.

3.2 Prototyypin aloittaminen Unityllä

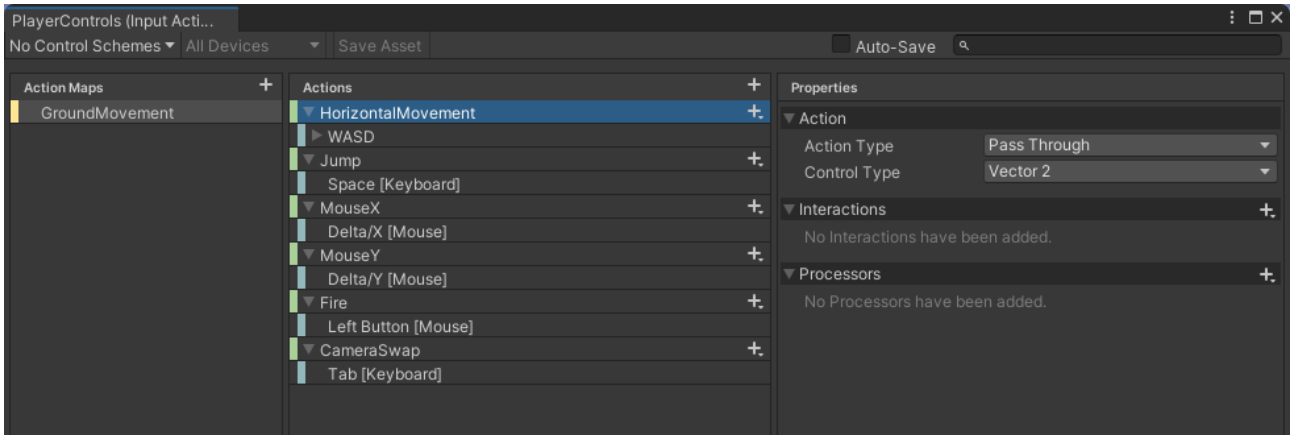
Valitsin tehdä pelin Unity pelimoottorilla, koska olen kuullut siitä hyvää pienissä peliprojekteissa. Unity on erittäin yleinen varsinkin indie-peleissä ja hyvä pelimoottori aloittelijoille (Dealessandri 16.1.2020). Unityllä on myös mahdollista tehdä 3D-pelejä, mitä kaikilla muilla indie-peleihin suunnatuilla pelimoottoreilla ei pysty tehdä. Lisäksi Unitystä on ilmainen perusversio, ja paljon materiaalia internetissä, kun ongelmia ilmenee (Jeremy Gibson 2014).

Unityssä pitää ensin luoda uusi projekti pelille. Koska kyseessä on 3D-peli valitsin tyhjän 3D-pelin pohjan (Kuva 1). Unityssä olisi ollut vaihtoehtoina myös saada valmis ensimmäisen persoonan 3D pelin malli, jossa olisi ollut valmiiksi tehty kamera ja pelaajan liike, mutta halusin aloittaa pelin teon tyhjästä perus 3D-pohjasta ja tehdä itse ensimmäisen persoonan kameran ja liikkeen.



Kuva 1. Unity Hubissa uuden projektin luonti

Kun Unity oli alustanut projektin ja edessä oli tyhjä 3D Unity projekti, piti alkaa rakentamaan pelaajahahmon liikettä. Jotta pelihahmo liikkuu, on pelaajan painettava jotain nappia näppäimistöllä tai hiirellä, joten ensin on saatava Unity tunnistamaan, kun pelaaja painaa tiettyjä nappeja. Tämä onnistui Unityn Input Managerilla (Kuva 2).

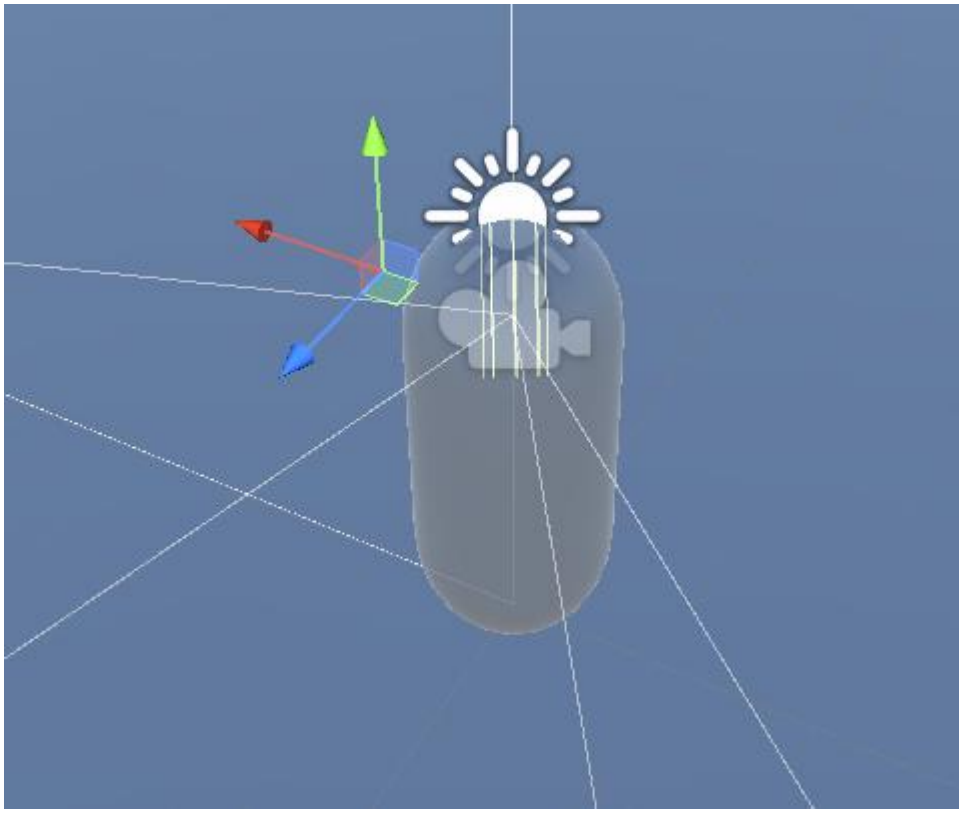


Kuva 2. Unityn input manager valikko

Unitylle kerrotaan, mitä pelaajan syötteitä pitää silmällä, tässä tapauksessa W, A, S ja D näppäimiä pelaajahahmon liikuttamiseen eteen, taakse, ja sivuille. Lisäksi tarvitaan hiiren liikuttamista ylös, alas ja sivuille ensimmäisen persoonan kameran kontrollointiin. Kun hiirtä liikuttaa ylös, pelihahmo katsoo ylös ja näin pelaaja myös näkee ruudullaan ylöspäin.

Eri syötteet pitää merkitä erityyppisiksi Unityssä. Joissain syötteissä riittää, kun Unity huomaa koska nappia painetaan. Toisissa tilanteissa on tärkeä tietää jatkuvasti, pitääkö pelaaja nappia alhaalla. Liikkumiseen tarkoitetuissa näppäimissä on pidettävä huoli, että Unity jatkuvasti seuraa onko nappi painettuna. Liikkeen pitää jatkua niin kauan, kun pelaaja pitää nappia alhaalla. Hyppäämisessä sen sijaan voi riittää, kun Unity seuraa, koska nappia painetaan. Hyppääminen tapahtuisi, kun nappia painetaan ja näppäimen pohjassa pitäminen ei tätä muuttaisi. Hiiren liikuttelussa taas tarvitaan tietää, kuinka paljon hiiri liikkuu. Tähän tarkoitukseen hiiren liikkeen kontrolli tyypiksi laitetaan akseli, jolloin Unity ei seuraa vain, että liikkuuko hiiri, vaan kuinka paljon hiiri liikkuu mihinkin suuntaan.

Tämän jälkeen luodaan pelaajalle objekti ja kamera Unityn graaffisessa käyttöliittymässä. Käytin pelaajan fyysisenä objektina Unityn valmista geometristä objektia ”capsule”, joka on päistä pyöristetty sylinteri. Se toimii pelaajan fyysisenä törmäyslaatikkona. Itse kapseli voi olla näkymätön koska kamera sijoitetaan kapselin yläosaan, jolloin pelaajahahmo itsessään ei ole näkyvissä (Kuva 3).



Kuva 3. Kapseli-pelaajahahmo ja kamera

3.3 Pelihahmon liikkeen luonti

Seuraavaksi pitää koodata Unityn input managerin tunnistama pelaajan syöte liikuttamaan pelihahmoa.

```
groundMovement.HorizontalMovement.performed += ctx =>
            horizontalInput = ctx.ReadValue<Vector2>();
```

Tämän jälkeen pitää saada Unity tunnistama pelaajan syöte lisättyä kaksiulotteiseen vektoriin. Tämä vektori tuodaan movement luokkaan ja lisätään tämä syöte pelaajahahmon tämänhetkiseen liikkeeseen kolmessa ulottuvuudessa.

```
public void ReceiveInput (Vector2 _horizontalInput) {
    horizontalInput = _horizontalInput;
}

Vector3 horizontalVelocity = (transform.right * horizontalInput.x
                           + transform.forward * horizontalInput.y) * speed;
```

Pelaaja-kontrolleri laitetaan tämän jälkeen liikkumaan tämän horizontalVelocity vektorin mukaan.

```
controller.Move(horizontalVelocity * Time.deltaTime);
```

Samalla lisätään pelaajahahmolle painovoima ja estetään pelaajahahmoa putoamasta maan tai muiden kiinteiden esineiden läpi.

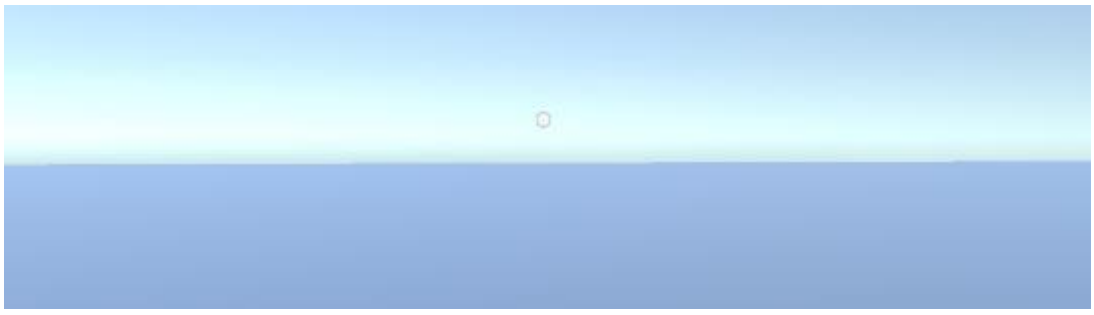
```
isGrounded = Physics.CheckSphere(transform.position, 0.1f, groundMask);
    if (isGrounded) {
       verticalVelocity.y = 0;
    }

verticalVelocity.y += gravity * Time.deltaTime;
```

Pelaajahahmo voi nyt liikkua kolmiulotteisessa ympäristössä, mutta ei voi katoa muihin suuntiin. Samalla tavalla kuin pelaajan liikkeen syöte tuotiin movement luokkaan, hiiren liike tuodaan MouseLook luokkaan.

```
private void Update () {
    transform.Rotate(Vector3.up, mouseX * Time.deltaTime);
    xRotation -= mouseY;
    xRotation = Mathf.Clamp(xRotation, -xClamp, xClamp);
    Vector3 targetRotation = transform.eulerAngles;
    targetRotation.x = xRotation;
    playerCamera.eulerAngles = targetRotation;
}

public void ReceiveInput (Vector2 mouseInput) {
    mouseX = mouseInput.x * sensitivityX;
    mouseY = mouseInput.y * sensitivityY;
}
```

Päivitys silmukassa pelihahmo laitetaan kiertämään sivuille hiiren liikkeen mukaan vapaasti. Mitä enemmän hiirtä liikuttaa oikealle ja vasemmalle, sitä enemmän pelihahmo ja kamera kiertää oikealle ja vasemmalle. Pystysuora akseli tarvitsee lukita joihinkin arvoihin, jotta kameraa ei saa käännettyä vahingossa ylösalaisin.

Pelaajakameraan luodaan käyttöliittymäelementtinä piste keskelle, jonka avulla pelaaja pystyy helpommin määrittelemään, mihin tarkkaan on katsomassa (Kuva 4).



Kuva 4. Taivaanranta pelimaailmassa ensimmäisestä persoonasta

Movement-luokkaan lisätään vielä koodia hyppäämiselle ja otetaan vastaan pelaajan syöte välilyönnin painamisesta.

```
if (jump) {
    if (isGrounded) {
        verticalVelocity.y = Mathf.Sqrt(-2f * jumpHeight * gravity);
    }
    jump = false;
}
controller.Move(verticalVelocity * Time.deltaTime);
```

Pelaajahahmoa voi nyt vapaasti liikuttaa eteen, taakse, ja sivuille, hypätä, ja liikuttaa kameraa, joka kuvaa pelaajan näkökenttää.

3.4 Lintuperspektiivin lisääminen

Seuraavaksi lisätään pelaajalle toinen kamera, joka on 45 asteen kulmassa taivaalla osoittamassa pelaajaa kohti. Pelaaja pystyy tab nappia painamalla vaihtamaan tämän kameran toiminnalliseksi. Lintuperspektiivin ollessa painettuna, pelaajan liikkeen tulisi pysähtyä paikoilleen ja pelaajan pitäisi pystyä liikuttamaan lintuperspektiivin kameraa ympäristössä asettaakseen merkkejä paikkoihin, joihin hän haluaa pystyä käyttämään kykyjään.

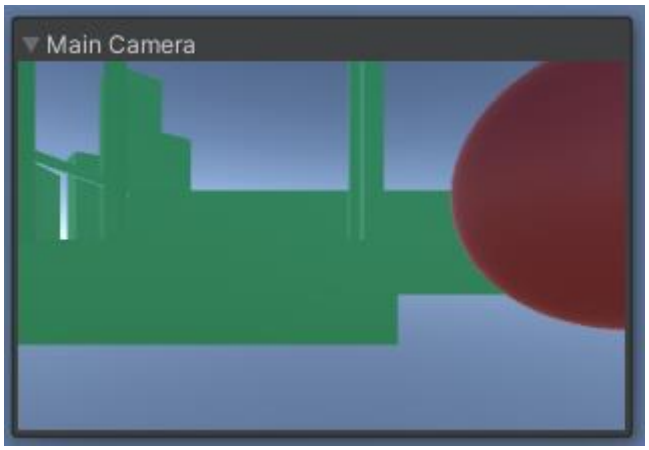
```
if (swap == true) {
    cam1.enabled = !cam1.enabled;
    cam2.enabled = !cam2.enabled;
    Cursor.visible = !Cursor.visible;
    swapLinger = true;

    if (cam1.enabled) {
        controller2.transform.position = controller.transform.position;
    }
    swap = false;
}
```

Kun pelaaja painaa tab nappia, kamera 1 ja kamera 2 vaihtavat toinen päälle ja toinen pois. Lisäksi kun pelaaja palaa ensimmäisen persoonan tilaan, lintuperspektiivin kamera palaa seuraamaan pelaajaa ja odottamaan uudelleen aktivointia.

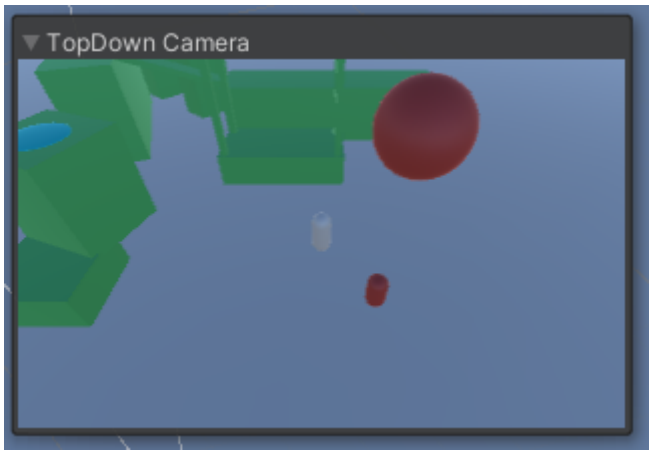
```
if (cam1.enabled) {
    *Koodia ensimmäiselle perspektiiville*
} else {
    controller2.Move(horizontalVelocity * Time.deltaTime);
}
```

Näin kun kamera 1 ei ole käytössä, oletus on, että kamera 2 on käytössä ja pelaajan liike laitetaan vastaamaan kontrolleri 2:ta, jonka tarkoituksena on mahdollistaa pelaajan liike lintuperspektiivissä

ja kykyjen asettaminen pelimaailmaan. Seuraavaksi vielä ero ensimmäisen persoonan ja lintuperspektiivin välillä samassa tilanteessa (Kuva 5, Kuva 6).



Kuva 5. Ensimmäisen persoonan kameran näkymä



Kuva 6. Lintuperspektiivin kameran näkymä

3.5 Salamakyvyn tekeminen

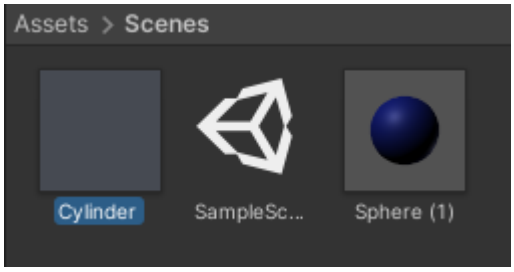
Pelin ideana on, että pelaaja pystyy merkkaamaan etukäteen lintuperspektiivissä, mihin haluaa käyttää eri kykyjä. Päädyin tekemään esimerkkinä näistä eri kyvyistä salamakyvyn. Pelaaja merkitsee paikan lintuperspektiivissä, ja osoittamalla tätä ensimmäisessä persoonassa aiheuttaa salamaniskun halutussa paikassa. Salamanisku voisi toimia aktivoijana napeille, jotka sitten esimerkiksi avaisivat oven pelaajalle. Aloitetaan kertomalla skriptissä, että olemassa peliobjekti ”prefab”.

```
public GameObject prefab;
```

Seuraavaksi lintuperspektiiviin tehdään ominaisuus laittaa luoda merkkejä, mitä pelaaja voi aktivoida kyvyillä myöhemmin. Koska lintuperspektiivissä ei hyppääminen ole mahdollista, voidaan samaa pelaajan syötettä käyttää merkkien luontiin.

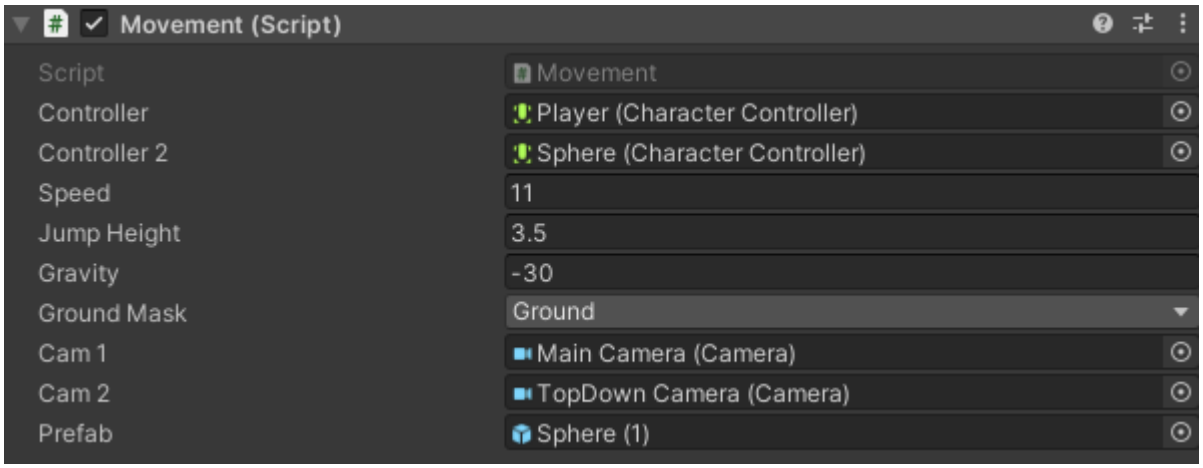
```
if (jump) {
    Instantiate(prefab,
    controller2.transform.position,
    controller2.transform.rotation);
    jump = false;
}
```

Pelin tiedostoihin lisätään objektit, jotka halutaan luoda pelin ajon aikana (Kuva 7).



Kuva 7. Sylinteri ja pallo objektit valmiina pelin tiedostoissa

Tämän jälkeen Unitylle pitää kertoa, että pallo objekti (Sphere (1)) pelin tiedostoissa halutaan käytettävän, kun skriptissä sanotaan ”prefab” (Kuva 8).



Kuva 8. Unitylle kerrottu mitkä asiat vastaavat mitäkin koodissa

Seuraavaksi pelaajan pitää pystyä tähtäämään näitä luotuja merkkejä ensimmäisessä persoonassa. Tässä käytetään Unityn valmiita raycast funktioita tunnistamaan, mikäli pelaaja tähtäsi merkkejä kohti. Mikäli pelaaja tähtää merkkiin ja painaa nappia 1, merkki tuhotaan ja tilalle luodaan toinen objekti salamalle. Salama luodaan sylinteristä, joka oli lisätty tiedostoihin kuvassa 7.

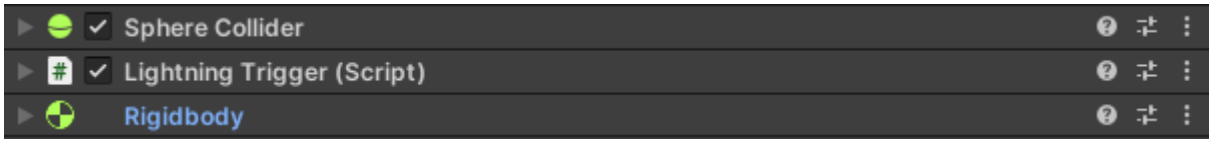
```
private void OnAction1() {
    RaycastHit hit;
    if (Physics.Raycast(fpsCam.transform.position,
```

```
        fpsCam.transform.forward, out hit, range)) {
            Target target = hit.transform.GetComponent<Target>();
            if (target != null) {
                target.TakeDamage(damage);
                Instantiate(prefab, target.transform.position, target.transform.rotation);
            }
    }
}
```

Salamalla itsessään on pieni skripti, joka saa salaman luontinsa jälkeen tuhoamaan itsensä.

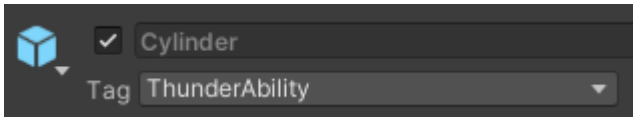
```
public class Thunder : MonoBehaviour {
    public float lifetime = 100f;

    void Update() {
        lifetime -= 1f;
        if (lifetime <= 0) {
            Destroy(gameObject);
        }
    }
}
```

Näin kun pelaaja aktivoi merkin salamalla, salama on hetken pelimaailmassa ja katoaa tämän jälkeen. Seuraava askel on luoda peliin objekti, joka tunnistaisi, kun salama osuu siihen. Tätä voidaan käyttää eri tavoin, mutta tässä prototyypissä laitoin salamaniskun aktivoimaan ison kuution liikkumaan ylös ja alas. Kun kuutio on alhaalla, se estää pelaajaa etenemästä, joten pelaajan on käytettävä salamaa edetäkseen. Aloitetaan luomalla peliobjekti, jolle lisätään kollisio, skripti ja rigidbody elementti (Kuva 9).



Kuva 9. Salamaniskun havaitsemiseen tarvittavat elementit objektissa

Salamalle tarvitsee lisätä leima, jolla skripti tunnistaa sen salamaksi (Kuva 10).



Kuva 10. Sylinterille lisätty leima

Tämän jälkeen skriptissä tarvitsee käyttää Unityn OnTrigger-funktiota huomatakseen, koska objekti osui toiseen objektiin, jolle on lisätty ”ThunderAbility” leima.

```
public void OnTriggerEnter (Collider other) {
    if (other.gameObject.tag == "ThunderAbility") {
        doorActivate = true;
    }
}
```

Tämän jälkeen tarvitaan vain hieman koodia, jotta kuutio saadaan liikkumaan, kun salama on osunut tähän nappiin.

```
void Update() {
    Vector3 p = door.transform.position;

    if (doorActivate) {
        if (p.y >= maxHeight) {
            movingDown = true;
        }
        if (p.y <= minHeight) {
            movingUp = true;
        }
        doorActivate = false;
    }

    if (movingUp == true) {
        p.y += speed;
    }
    if (movingDown == true) {
        p.y -= speed;
    }

    if (p.y > maxHeight) {
        p.y = maxHeight;
        movingUp = false;
    }
    if (p.y < minHeight) {
        p.y = minHeight;
        movingDown = false;
    }
    door.transform.position = p;
}
```

Napin omassa päivitysfunktiossa muutamalla ”if” lauseella saadaan kuutio liikkumaan ylös ja alas, riippuen siitä, kummassa tilassa se oli aikaisemmin. Mikäli kuutio oli ylhäällä, kun salama osuu nappiin, liikkuu kuutio alas. Tai toisinpäin.

3.6 Prototyypin päättäminen

Tässä vaiheessa prototyyppi mielestäni pystyi vastaamaan sille asettamiini kysymyksiin. Prototyyppiä olisi voinut jatkaa pidemmälle, mutta se olisi mielestäni ollut ajan tuhlausta, sillä tällä toiminnallisuudella pystyy jo sanomaan peli-idean toimivuudesta.

Ensimmäinen kysymykseni oli, että toimiiko tällainen perspektiivin vaihto hyvin olennaisena pelimekaniikkana. Mielestäni ei kovin hyvin, sillä sen käyttö on hidasta ja se keskeyttää sen hetkisen

toiminnan täysin. Mekaniikka toimisi paremmin vähemmän olennaisena osana peliä, jolloin sitä tarvitsisi käyttää harvemmin.

En myöskään ollut ollut varma, tuoko lintuperspektiivi mitään varsinaista hyötyä peliin. Implementaationi, jossa pelaaja joutuu nähdä merkit, joita on asetellut ympäriinsä, poistivat kaikki mahdollisuudet tästä. Mikäli pelaaja pystyisi aktivoimaan merkkejä seinien ja esteiden läpi, siinä voisi olla jotain ideaa, mutta silloin vaarana on, että pelaaja ei tunnista merkkejä toisistaan tai ei pysty helposti sanomaan, mitä merkkejä hän on aktivoimassa.

Viimeisenä olin yrittänyt miettiä, onko peli-idea tarpeeksi mielenkiintoinen itsessään, vai tarvitseeko se jotain lisää, kuten vihollisia. Mikäli peli-ideaa jalostaisi pidemmälle, pitäisi mielestäni pelin keskipistettä muuttaa hieman tästä perspektiivinvaihdosta johonkin muualle. Eräs tapa toteuttaa tämä olisi keskittyä esimerkiksi vihollisten kanssa taistelemiseen, ja pitää tämä perspektiivin muuttaminen avustavana osana peliä.

Tämä prototyyppi tällaisenaan sai minut näkemään ongelmia, joita en osannut nähdä aikaisemmin. Mikäli tästä haluaisi jatkaa, joutuisi peli-ideaa muokkaamaan, ja tämä tulisi todennäköisesti vaatimaan uuden prototyypin eri tavoitteilla.

4 Pohdinta

Valitsin tehdä opinnäytetyökseni pelin, koska olin alun perin hakenut alalleni ajatuksella, että haluaisin ohjelmoida pelejä. Mielestäni opinnäytetyöni olisi voinut sujua paremmin, mutta en ole miten-kään pettynyt lopputulokseenkaan. Projekti hoidettiin sprinttien muodossa, ja ensimmäiset sprintit olivat hieman haastavia, sillä minun oli vaikea käsittää työmäärän kokoa. Mielestäni projektin hallinta ja peli-idea olivat suurimmat haasteet minulle. Olen kiitollinen ohjaajalleni tuesta, joka auttoi pitämään projektin paremmin aikataulussa. Peli-idea olisi voinut olla myös helpompi toteuttaa ja näin ehkä saada enemmän itsessään viihdyttävä prototyyppi. Minulla oli tarkoituksena ottaa hieman erilaista kuvakulmaa pelin suunnitteluun, kuten laittaa pääpaino käyttöliittymälle. Projektin aikana päädyin löytämään itselleni paljon hyödyllistä materiaalia pelien suunnittelun perusajatuksista. Pelien käyttöliittymistä sen sijaan oli vaikea löytää paljoa mitään. Tämän lisäksi pelin prototyyppiä tehdessäni huomasin, että käyttöliittymä jäisi aika vähäiseksi prototyypissäni, joten jouduin jättämään käyttöliittymien tarkastelun pois.

Itse pelin suunnittelu on hankalaa. Peli-idean saaminen omasta päästä prototyypiksi on työlästä ja monimutkaisuutensa vuoksi aikaa vievää. Ensi kerralla varmaan yrittäisin suunnitella jotain, jonka voi prototypoida paperisena, koska se vähentäisi huomattavasti työtaakkaa. Jälkeenpäin ajateltuna olisin ehkä suunnitellut toisenlaisen pelin tähän opinnäytetyöhön, jonka idea olisi helpompi avata muille. Pelisuunnittelun teorian löytäminen ja lukeminen on vieläkin yhtä mielenkiintoista kuin se on aina ollut minulle. Itse pelin tekeminen puolestaan on mielestäni turhan raskas projekti tehdä yksin.

Opinnäytetyöni saavutti sen tavoitteet mielestäni. Tavoitteena oli ollut saada kokemusta peli-idean suunnittelusta ja sille prototyypin rakentamisesta. Lopputuloksena sain molemmat, vaikkakin prototyyppi osoitti peli-ideani ongelmat. Tämä itsessään ei tietenkään tarkoita, etteikö prototyyppi olisi ollut menestys, sillä prototyyppini vastasi juuri niihin kysymyksiin, joita minulla oli peli-ideasta. Tämän prototyypin jatkaminen eteenpäin olisi ollut turhaa, sillä seuraava vaihe olisi ollut muokata peli-ideaa tämän prototyypin vastausten mukaan. Minulle jäi tästä hyvä käsitys siitä, miten ensimmäisen oman pelin suunnittelu toimii ja miten sille kehitetään prototyyppi. Olen käsittänyt, että prototyypit usein näyttävät ongelmat peli-ideassa ja että harvasta prototyypistä rakennettaan kokonaista peliä. Ehkä tästä projektista oppineena seuraavalla kerralla onnistun peli-idean suunnittelussa ja prototyyppi osoittaa sen toimivaksi.

Peli prototyyppi, joka tehtiin tässä opinnäytetyössä, löytyy osoitteesta: https://github.com/Incomputable/oparipeli .

Lähteet

Adams, E & Dormans, J. 2012. Game Mechanics: Advanced Game Design. New Riders. San Francisco. E-kirja. Luettu 30.4.2022.

Bartle, R., Bateman, C., Falstein, N., Hinn, M., Isbister, K., Lazzaro, N., Grainer Ray, S., Saulter, J. 2009. Beyond Game Design: Nine Steps Toward Creating Better Videogames. Charles River Media. Boston.

Beardshaw, J. 13.4.2020. Three free tools to level up your prototyping. Games Industry. Luettavissa: https://www.gamesindustry.biz/articles/2020-04-08-three-free-tools-to-level-up-your-prototyping . Luettu 14.5.2022.

Burgun, K. 2015. Clockwork Game Design. Routledge. Lontoo. E-kirja. Luettu 30.4.2022.

Dealessandri, M. 16.1.2020. What is the best game engine: is Unity right for you?. Games Industry. Luettavissa: https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you . Luettu 17.5.2020.

Dixon, D. 2011. Player Types and Gamification. Vancouver. Luettavissa: http://gamification-research.org/wp-content/uploads/2011/04/11-Dixon.pdf . Luettu 14.5.2022.

Gibson, J. 2014. Introduction to Game Design, Prototyping and Development. Addison-Wesley. Boston.

Kramarzewski, A & De Nucci, E. 2018. Practical Game Design. Packt Publishing. Birmingham. E-kirja. Luettu 30.4.2022.

Marcu, C. 9.12.2015. An approach to game design. Game Developer. Luettavissa: https://www.gamedeveloper.com/design/an-approach-to-game-design . Luettu 12.5.2022.

Sellers, M. 2018. Advanced Game Design: A Systems Approach. Addison-Wesley. Boston.

Sicart, M. 2008. Defining Game Mechanics. Game Studies. Luettavissa: http://gamestudies.org/0802/articles/sicart . Luettu 18.5.2022.

Sniezak, C. 25.4.2016. The Eight Types of Fun. Gnomestew. Luettavissa: https://gnomestew.com/the-eight-types-of-fun/ . Luettu 14.5.2022

Stegner, B. 19.11.2021. What Are Metroidvania Video Games?. Make Use Of. Luettavissa: https://www.makeuseof.com/what-are-metroidvania-video-games/ . Luettu 12.5.2022.

Tyler, D. 24.12.2021. Video Game Mechanics (Core, Primary, Secondary): Everything You Need To Know. Game Designing. Luettavissa: https://www.gamedesigning.org/learn/basic-game-mechanics/ . Luettu 14.5.2022.

Vito Oddo, M. 2.8.2021. What's an Indie Game Anyway?. Collider. Luettavissa: https://collider.com/what-makes-an-indie-game/ . Luettu 13.5.2022.

Yu, J. 9.2.2021. History of Metroidvania Games. Game Rant. Luettavissa: https://gamerant.com/metroidvania-games-indie-history/ . Luettu 15.5.2022.