OULUN AMMATTIKORKEAKOULU

Hannu Manninen

O.R.B OFFENSIVE ROLLING BOT -PELIN OHJELMOINTI

O.R.B OFFENSIVE ROLLING BOT -PELIN OHJELMOINTI

Hannu Manninen
Opinnäytetyö
Syksy 2020
Tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely, Web-sovelluskehitys

Tekijä(t): Hannu Manninen
Opinnäytetyön nimi: O.R.B Offensive Rolling Bot -pelin ohjelmointi
Työn ohjaaja: Matti Viitala
Työn valmistumislukukausi ja -vuosi: Syksy 2020 Sivumäärä: 61

Päiväkirjamuotoisessa opinnäytetyössä tarkastellaan juniortason peliohjelmoijan työpäiviä kymmenen viikon ajalta. Kirjaan päivittäin suunnitellut työtehtävät, kohtaamani ongelmat ja ratkaisut ongelmiin. Jokaisen viikon päätteeksi analysoin kuluneen viikon tapahtumat esimerkiksi oman oppimisen näkökulmasta.

Päiväkirjaraportoinnin jälkeen tarkastelen kymmenen viikon aikana saavutettuja tuloksia ja omaa kehitystäni pelinkehittäjänä. Pohdin oppimiani asioita esimerkiksi ohjelmoinnissa, työskentelymetodeissa, projektinhallinnassa sekä tiimin sisäisissä kommunikointitaidoissa.

Toimin peliohjelmoijana kuuden hengen tiimissä, joka tekee Unity-pelimoottorilla kaksiulotteista tasohyppelypeliä nimeltään O.R.B Offensive Rolling Bot. Projekti on aloitettu syksyllä 2018 Oulu Game Lab -opinnoissa. Opinnäytetyössä seurataan projektin loppuvaiheita, ja projektin on tarkoitus valmistua pian opinnäytetyön valmistumisen jälkeen.

Opinnäytetyön tarkoituksena on oman ammatillisen osaamisen kehitys ja realistinen juniortason peliohjelmoijan arjen ja työtehtävien kuvaus, josta lukijan pitäisi saada selvä kuva siitä, mitä taitoja ja osaamistasoa tämän tyyppinen projekti vaatii peliohjelmoijalta pienessä tiimissä.

Päiväkirjan seurantajakso on 14.10.2019 – 24.01.2020. Seurantajakson päätyttyä peli edistyi odotetusti, mutta siihen jäi vielä paljon tehtävää, ennen kuin se on valmis.

Seurantajakson aikana kehityin todella paljon yleisesti Unityn käytössä, peliohjelmoinnissa, projektinhallinnassa ja yleisesti monissa projektiin liittyvissä tehtävissä. Tiimin pienen koon vuoksi oma työni oli monesti laajempaa kuin pelkkää peliohjelmointia, joten opin työtehtäviä laajemminkin.

Asiasanat: peliala, ohjelmistokehitys, peliohjelmointi, päiväkirja

ABSTRACT

Oulu University of Applied Sciences
Business Information Systems, Web-Application Development

Author(s): Hannu Manninen
Title of thesis: O.R.B Offensive Rolling Bot -pelin ohjelmointi
Supervisor(s): Matti Viitala
Term and year when the thesis was submitted: Autumn 2020 Number of pages: 61

This thesis that is written in diary form depicts a junior level game programmer's working days for the duration of ten weeks. I report my planned tasks, problems, and solutions each day. At the end of each week, I analyze what I have accomplished in the past week and analyze the events of the past week from my own learning's point of view.

After the working diary is complete, I analyze what I have accomplished in the past ten weeks and my own progress as a game developer. I reflect the things that I have learned from programming, working methods, project management and internal communication skills in the team.

I work as a game developer in a team of six people, that is making a 2D-platformer game called O.R.B Offensive Rolling Bot using Unity game engine. The project was started in autumn 2018 in Oulu Game Lab -studies. The thesis follows the projects final phases, the project is supposed to be finished soon after this thesis is complete.

The purpose of the thesis is the improvement my own professional skills and realistically depicting the ordinary workdays of a junior level game programmer, from which the reader should get a clear picture, that what kind of skills and level of expertise does this type of project require from a game programmer in such a small team.

The diary's monitoring period is 14.10.2019 – 24.01.2020. After the monitoring period was over the game had progressed as expected, but there was still a lot to do before it is finished.

During the monitoring period I improved a lot in using Unity game engine, game programming, project management and in many things relating to the project. Due to the small size of the team, my own job was often more extensive, than just game programming, so I naturally learned things on a bit wider scale.

Keywords: unity, game industry, software development, game programming, diary

Projektin keskeiset käsitteet

Array – Taulukko, jota käytetään koodissa.

Asset – Tarkoittaa valmiita komponentteja, grafiikoita, ääniä, koodia tai vastaavia. Unity-pelimoottorin Asset Storesta pystyy lataamaan muiden käyttäjien tekemiä assetteja.

Build – Pelistä tehty itsenäinen sovellusversio, jota pystyy pelaamaan. Tässä versiossa ei voi muokata peliä mitenkään.

Collider – Peliobjektissa oleva komponentti, jolla voi tarkistaa törmäykset muihin peliobjekteihin, joilla on myös Collider-komponentti.

Commit – Versiohallinnan työkalu, jossa käyttäjä lukitsee ja kommentoi tekemänsä muutokset.

Coroutine – Eräänlainen metodi, joka voidaan käynnistää tekemään jotain tiettyä toimintoa rutiininomaisesti Update-funktion tapaisesti, mutta se voidaan myös keskeyttää halutessa ja niitä voi olla monta samanaikaisesti päällä.

Editor – Unity-pelimoottorin päänäkymä, jossa peliä rakennetaan.

Kuvataajuus – Kuvataajuus määrittää, kuinka usein peli päivittyy sekunnissa.

Git – Opinnäytetyön pelissä käytetty versionhallintajärjestelmä.

HUD – Heads Up Display on näyttö/ikkuna, josta pelaaja näkee kesken pelin tietoja, kuten omat elämäpisteensä.

Index – Listalla oleva luku, joka kertoo, missä kohdassa listaa jokin tietty asia on.

Instantiate – Kesken pelin luodaan koodin sisällä peliin jokin asia, kuten uusi vihollinen.

Killbox – Näkymätön alue, joka tappaa pelaajan, jos pelaaja ja tämä alue osuvat toisiinsa.

Luokka – Määrittelee tietyn oliojoukon yhteiset piirteet. Peliin voi tehdä esimerkiksi Vihollinen-nimisen luokan, jossa on piirteitä, joita kaikilla pelin vihollisilla on, kuten elämäpisteitä. Sen jälkeen tästä luokasta voi luoda erilaisia vihollisia.

Merge – Versionhallinnan työkalu, jolla voidaan yhdistää kaksi eri versiota toisiinsa.

Metodi – Tarkoittaa jotakin ohjelman sisäistä toimintoa, jolla on oma nimi ja jolle voidaan antaa jotain arvoja, joilla se tekee jotakin sitä kutsuttaessa.

Optimointi – Pelin toimivuuden parantaminen esimerkiksi säätämällä pelin graafisia asetuksia.

Ohjelmointivirhe – Jokin virhe koodissa, joka estää ohjelmaa toimimasta oikein. Virhe voi olla kirjoitusvirhe tai koodissa oleva looginen virhe.

Peliobjekti – Jokin objekti pelissä, jolla on yleensä useita eri ominaisuuksia ja komponentteja.

Pivot – Jonkin objektin tai grafiikan kiertopiste.

Placeholder – Jokin keskeneräinen kohta, jonka kuitenkin täytyy olla mukana esimerkiksi testaamisen vuoksi. Yleensä kyse on keskeneräisestä grafiikasta.

Plugin – Koodi-asset.

Prefab – Jostain peliobjektista tehty tallennettu versio, jolloin sama peliobjekti on helppo lisätä uudelleen peliin myös koodin kautta.

Projektinhallinta – Projektin suunnittelua, jossa jokaiselle projektin jäsenelle aikataulutetaan heidän tehtävänsä.

Pull – Versionhallinnan työkalu, jossa käyttäjä lataa tietyn version koneelleen.

Push – Versionhallinnan työkalu, jossa käyttäjä lähettää tekemänsä muutokset versiohallintaan muiden nähtäväksi.

Raycast – Näkymätön viiva tai muoto, joka kulkee tietystä pisteestä tiettyyn suuntaan ja kertoo tietoja osumistaan.

Rigidbody – Peliobjektissa oleva fysiikan komponentti, jolla pystytään käsittelemään peliobjektin fysiikkaan liittyviä tietoja, kuten esimerkiksi objektin massaa.

Sprintti – Eräänlainen projektin sisäinen miniprojekti, jossa päätetään jokin projektin tavoite ja jolle asetetaan aikaraja, joka yleensä on yhdestä neljään viikkoa.

Skene – Yksi pelin näkymä, tyypillisesti jokin valikko tai kenttä pelissä.

Scriptable object – Objekteja, jotka ovat pelkkiä arvoja, joita muut objektit tai koodit voivat käyttää.

Shader – Ohjelma, joka laskee kuvassa olevat efektit, kuten varjostuksen.

Sprite – 2D-peleissä käytettävä 2D-grafiikka.

Tag – Yhdistävä nimeäminen samankaltaisille objekteille. Eräs tag voisi olla vaikka ”Vihollinen”, jolloin kaikki viholliset merkitään sillä. Koodissa pystyy silloin käsittelemään kaikkia niitä kerralla.

Transform – Jonkin peliobjektin sisäinen sijaintitieto.

Trigger – Kun jokin asia koskee johonkin toiseen asiaan tai alueeseen pelissä, kutsutaan trigger tekemään jotain.

UI - User Interface eli pelin käyttöliittymä.

Update – Ohjelman sisäinen funktio, joka päivittyy jokaisella kuvataajuuden päivityksellä.

Unity - Opinnäytetyössä käytetty erittäin suosittu pelimoottori.

Versionhallinta – Projektin säännöllinen tallentaminen eri versioihin, joihin kaikki projektin jäsenet pääsevät käsiksi ja tallentavat tekemiään muutoksia.

2D – Tarkoittaa kaksiulotteisuutta, jolloin pelin grafiikat piirretään tasoille vain pituus- ja leveyssuunnassa. Syvyyttä ei siis ole.

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä tutkitaan peliohjelmointia työnä ja sen haasteita juniortason peliohjelmoijan näkökulmasta pienessä ryhmässä. Työssä perehdytään puolivälissä olevaan peliprojektiin, joka on opinnäytetyön aikana tarkoitus saada lähes julkaisuvalmiiksi.

Opinnäytetyö suoritetaan päiväkirjamuotoisena tutkielmana, jossa seurataan peliohjelmoijan työpäiviä kymmenen viikon ajalta: laadin päivittäin suunnitelman päivän työtehtävistä projektihallintatyökalua hyväksikäyttäen ja päivän lopuksi tarkastelen päivän kuluessa suorittamiani työtehtäviä ja mahdollisesti esille tulleita ongelmia. Jokaisen työviikon päätteeksi teen viikkoanalyysin, jossa pohditaan, mitä kuluneella viikolla on saatu aikaan, ja mitä uutta olen oppinut esimerkiksi ohjelmoinnista, työmetodeista, erilaisten työkalujen käytöstä tai tiimityöskentelystä.

Opinnäytetyössä kehitettävänä projektina on tasohyppelypeli O.R.B (Offensive Rolling Bot). Peliä kehitetään PC-alustalle pienessä kuuden hengen tiimissä, jossa toimin peliohjelmoijana. Pelin kehitys on aloitettu syksyllä 2018 Oulu Game Lab -opinnoissa. Tavoitteenani tämän opinnäytetyön aikana on lisätä peliin viimeiset puuttuvat ominaisuudet ja mekaniikat sekä lopuksi korjata viat sekä optimoida ja viimeistellä peli julkaisua varten keväällä 2020.

2 NYKYTILANTEEN KUVAUS

2.1 Oman nykyisen työn analyysi

Nykyinen työtehtäväni on toimia peliohjelmoijana kuuden hengen pelinkehitystiimissä. Tiimissä on eräs toinen ohjelmoija minun lisäkseni, kolme artistia sekä tasorakentaja. Eräs artisteistamme toimii myös pelin pääsuunnittelijana ja muut, kuten minä, toimimme avustavina suunnittelijoina. Oma osuuteni pelisuunnittelussa on enemmänkin tasapainotella pelin eri ominaisuuksia numeroiden suhteen sekä rajoittaa realistisesti pelisuunnitelmia niiden teknisen toteutuksen kannalta eli ottaa kantaa siihen, mitä voi ja mitä ei voi tehdä. Suunnittelen toki myös peliin uusia ominaisuuksia ja jaan uusia ideoita tiimin kesken. Ohjelmoinnin puolella minun osuuteni on oikeastaan kaikki muu paitsi pelin viimeinen päävihollinen eli kaikki uudet ominaisuudet ja ohjelmointivirheiden korjaukset. Pelin viimeistely tehdään todennäköisesti yhdessä toisen peliohjelmoijan kanssa. Autan myös tiimin muita jäseniä Unity-pelimoottorin ja versionhallinnan käytössä, aina kun tarvitsee.

Työtehtävissäni vaaditaan hyvää Unity-pelimoottorin osaamista, C#-ohjelmointikielen osaamista, pelisuunnittelutaitoja, versionhallintatyökalun osaamista, projektinhallintatyökalun osaamista sekä tiimin sisäisiä kommunikointitaitoja. Unity-pelimoottorin osaamistaidot ovat erittäin tärkeät minulle, sillä koko projekti tehdään Unity-pelimoottorilla, ja siihen kuluisi todella paljon ylimääräistä aikaa, jos en osaisi käyttää sitä sulavasti ja se täytyisi opetella. C#-ohjelmointikielen käyttö on tärkeää osata hyvin, koska peli tietysti ohjelmoidaan sillä. Minun täytyy myös osata suunnitella arkkitehtuurisesti ohjelmoinnin kannalta ominaisuudet, ennen kuin teen ne, sillä jos alkaa vain ohjelmoimaan jotakin ilman kunnon suunnittelua, niin sen saattaa tehdä huonosti toimivalla tai epäoptimoidulla tavalla, minkä jälkeen sen joutuu tekemään uudelleen. Täytyy myös ajatella koodin laajennettavuutta, koska jos vaikka paljon myöhemmin siihen halutaan lisätä jotain uusia ominaisuuksia, niin siihen kuluisi vähemmän aikaa. Koodi täytyisi myös kommentoida hyvin ja selkeästi sitä tehdessä, sillä jos sen pariin palaa myöhemmin, niin ei kuluisi ylimääräistä aikaa koodin toiminnan tutkimiseen. Täytyy myös tietää ja ymmärtää, mitä pystyy tekemään ja mitä ei tai mihin kuluu liikaa aikaa, jotta osaa tuoda sen esille pelisuunnitteluvaiheissa. Esimerkiksi jos jokin ominaisuus on liian vaikea tai aikaa vievä, niin voidaan pohtia, voisiko sitä yksinkertaistaa tai korvata jollain muulla ominaisuudella. Pelisuunnittelutaidot ovat tärkeät, koska pitää tietää, mikä pelissä on hauskaa ja mikä ei.

Pitää myös osata tasapainottaa pelin eri ominaisuudet keskenään, jotta ne toimisivat sulavasti yhdessä ja jotta ne ovat pelikokemuksen kannalta hyviä. Versionhallinta on tärkeää osata hyvin, sillä se varmistaa sujuvan yhteistyön tiimin välillä ja antaa niin sanotusti turvaverkon projektille, sillä jos jokin menee vikaan, niin aina voidaan palata takaisin aiempaan versioon todella nopeasti. Kommunikaatiotaidot sekä projektinhallintatyökalun käyttötaidot ovat minulle tärkeät, koska niiden avulla voin priorisoida tehtäväni. Puhun tiimiläisteni kanssa siitä, mitkä ominaisuudet olisivat nyt tärkeitä saada nopeasti peliin, ja onko joitakin ohjelmointivirheitä, jotka olisivat tärkeitä korjata nopeasti. Lisäksi kommunikointitaitoja vaaditaan pelitestaustapahtumissa, kun seuraan uusien pelaajien pelaamista ja keskustelen heidän kanssaan siitä, mistä he pitävät ja mistä eivät ja onko jokin liian vaikeaa tai helppoa tai vaikkapa epäselvää. Lisäksi pelitestaajilla on useasti myös hyviä ehdotuksia pelisuunnitteluun liittyen. nykyisellään koen osaavani nämä taidot vaadittavan hyvin selviytyäkseni tehtävistäni projektissa.

Pystyn kaikissa näissä osa-alueissa varmasti kehittymään paljon, eniten kuitenkin C#-ohjelmoinnissa, varsinkin koodin ulkoasun, kommentoinnin ja optimoinnin suhteen, koska sitähän työni suurimmaksi osaksi on. Minun täytyisi oppia tekemään koodistani selvempää ja kommentoimaan sitä paremmin. Lisäksi minulla on vielä paljon opittavaa koodin suunnittelusta ja toteutuksesta, joissa uskon kehittyväni tasaista tahtia ohjelmoidessani. Versionhallinnasta minulla on vielä paljon opittavaa, mutta sen päivittäiseen käyttöön taitoni riittävät aivan hyvin, koska versionhallinnan normaalikäyttö yleisesti on Commit-, Pull- ja Push-ominaisuuksien käyttöä ja joskus Merge-ominaisuudella huonosti yhteen sopivien versioiden yhdistämistä ja ääritapauksissa aiempiin versioihin palaamista, jotka kaikki osaan jo hyvin.

2.2 Sidosryhmät työpaikalla

Projektia tehdään pienessä kuuden hengen tiimissä nimeltään Karva Games, joka tekee yhteistyötä Riimu Games -nimisen peliyhtiön kanssa samoissa toimitiloissa. Karva Games siis tekee projektia, ja Riimu Games valvoo projektin aikataulua, neuvoo Karva Gamesia ja etsii pelille julkaisijaa ja rahoitusta ja hankkii tarvittaessa uusia työntekijöitä ja toimistotarvikkeita kyseiseen projektiin. Karva Games ja Riimu Games ovat kummatkin syntyneet Oulu Game Lab -opinnoissa, joissa tämäkin projekti sai alkunsa. Oulu Game Lab vaikuttaa vielä projektiin järjestämällä suosittuja pelitestaustapahtumia, joissa saamme näkyvyyttä ja paljon uusia testaajia pelillemme.

Karva Games -tiimissä työskentelee kuusi jäsentä. Yksi on pääsuunnittelija sekä artisti, toinen on pääartisti, kolmas on artisti ja sosiaalisen median vastaava, neljäs on pelisuunnittelija sekä tasorakentaja, viides on peliohjelmoija sekä pelisuunnittelija ja kuudes on peliohjelmoija. Meitä avustaa myös ulkopuolinen audio-henkilö. Tiimimme on pieni, mutta tarpeeksi suuri ja kyvykäs pienen projektin tekemiseen.

2.3 Vuorovaikutustaidot työpaikalla

Karva Gamesin kaltaisessa pienessä tiimissä on tärkeää, että pystyy kommunikoimaan hyvin ja tulee toimeen kaikkien kanssa. Tämä on tärkeää työilmapiirin vuoksi, joka vaikuttaa kaikkien työmoraaliin ja tehokkuuteen. Päivittäin tulee useasti keskusteltua pelin eri ominaisuuksista ja siitä, miten asioiden pitäisi olla tai miten ne pitäisi tehdä, ja useasti tulee tiimin välillä erimielisyyksiä, jotka täytyy osata ratkaista jonkinlaisella ratkaisulla tai kompromissilla. Useimmiten tällaiset erimielisyydet liittyvät johonkin pelin sisäiseen mekaniikkaan, josta ei olla samaa mieltä, millainen sen pitäisi olla. Kommunikointitaitoja tarvitaan tietysti myös aina, kun pidetään palavereita, kuten pelisuunnittelupalavereita, jotka voivat venyä hyvinkin pitkiksi, kun ideoidaan jotain uutta tai mietitään, millaisen jonkin olemassa olevan asian pitäisi olla.

Lisäksi kommunikointitaitoja vaaditaan hyvinkin paljon pelitestaustapahtumissa, joita Oulu Game Lab järjestää. Pelitestaustapahtumissa meillä on yleensä valmiiksi tehdyt kyselylomakkeet testaajille, mutta seuraamme kuitenkin aktiivisesti vierestä pelaajia ja kyselemme ja kirjaamme ylös heidän kokemustaan ja mielipiteitä pelistä. Monet mahtavat ideat ovatkin peräisin pelitestaajiltamme. Pelitestaustapahtumien jälkeen käymme sitten läpi heidän palautettaan ja meidän muistiinpanojamme tapahtumasta ja mietimme, että mitä uutta sen perusteella täytyy tehdä tai täytyykö jotakin pelimekaniikkaa säätää vaikeustason suhteen ja onko pelissä joitain vikoja, joita täytyy korjata.

2.4 Projekti

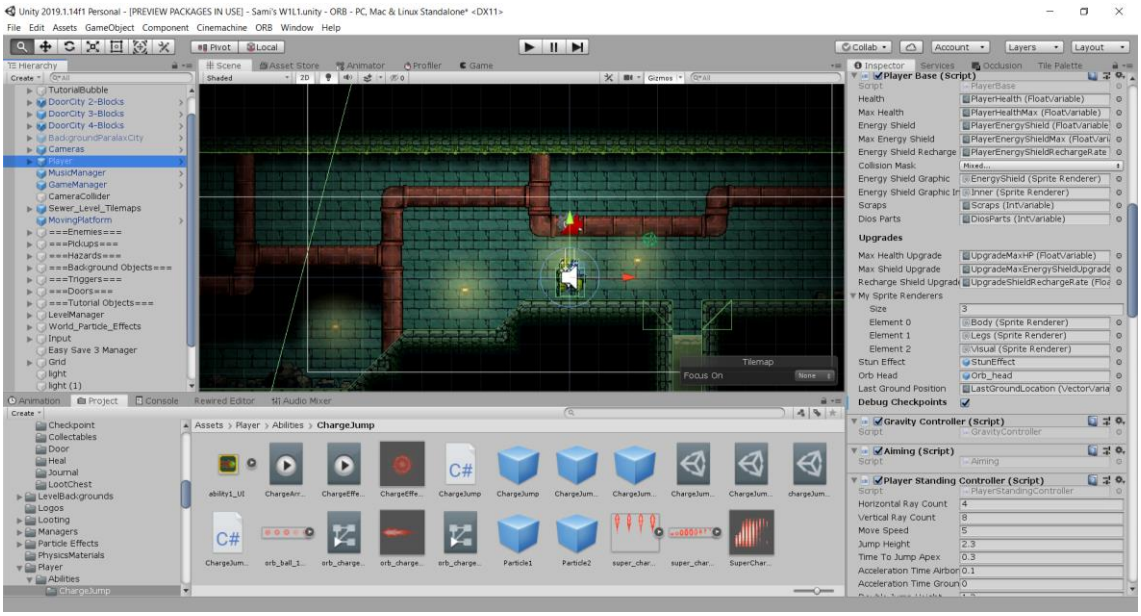
Projektina toimii O.R.B. Offensive Rolling Bot niminen retrotyylinen 2D-tasohyppelypeli, jota on tehty syksystä 2018 lähtien. Pelissä pelaaja ohjaa robottia, jolla on kaksi eri muotoa, jotka ovat kävelevä taistelumuoto, jossa pelaaja pystyy ampumaan erilaisilla aseilla, kävelemään ja hyppimään, sekä paljon mobiilimpi mutta heikompi pallomuoto, jossa pelaaja pystyy hyppimään ja kim-

poilemaan nopeasti tasojen läpi ja väistelemään vihollisten hyökkäyksiä. Pelaaja voi sulavasti vaihdella näiden muotojen välillä lähes missä tilanteessa tahansa. Pelaajalla on myös käytettävissään kykyjä kummallekin muodolle. Tässä kuva pelaajahahmosta kummassakin muodossa (kuvio 1.).

KUVIO 1. Kuva, jossa näkyy pelaajahahmon molemmat muodot.

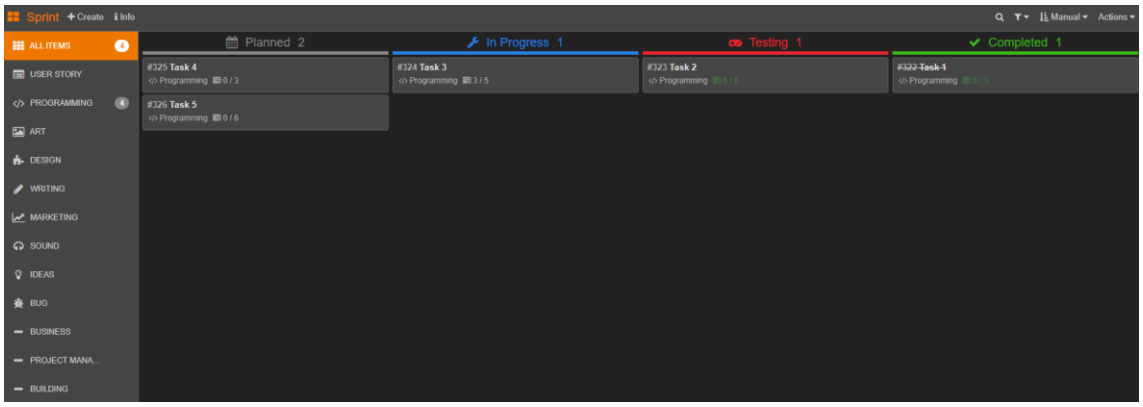
Projektia tehdään Unity-pelimoottorilla C#-ohjelmointikielellä. Projektin työssä käytän päivittäin apunani Hack n' Plan -projektinhallintatyökalua sekä Sourcetree-nimistä versionhallintatyökalua.

Unity-pelimoottori on pelimoottori, jolla projektia toteutetaan. Unity-pelimoottori on yksi suosituimmista pelimoottoreista maailmalla, ja sitä käytetään hyvin laajasti. Unity-pelimoottorissa käytämme C#-ohjelmointikieltä pelin ohjelmointiin. C#-ohjelmointikieli on suhteellisen yksinkertainen, moderni, tehokas sekä monikäyttöinen olio-ohjelmointikieli, joka sopii erittäin hyvin videopelien ohjelmointiin. Unity-pelimoottorissa on hyvin monipuolinen käyttöliittymä, jossa voi käsitellä eri skene-näkymiä, animaatioita, skriptejä, objekteja, ääniä ja monia muita asioita. Unity-pelimoottoriin on myös todella helppo ladata Unity-pelimoottorin omasta Asset storesta uusia Asset-lisäosia, jotka ovat joko ilmaisia tai maksullisia. Yksi esimerkki lataamastamme lisäosasta on EasySave3-lisäosa, jota käytämme pelin tallentamiseen. Tällä tavalla säästimme paljon aikaa ja vaivaa, kun emme joutuneet itse tekemään omaa pelintallennus koodiamme. Unity-pelimoottorissa on myös todella helppo testata peliä milloin tahansa, sillä siinä on pelitila sitä varten, että pystyy Editor-näkymässä käynnistämään pelin ja pelaamaan ja testaamaan sitä. Tässä on kuva Unity-pelimoottorin käyttöliittymästä ja samalla kuva projektistamme siinä (kuvio 2.).



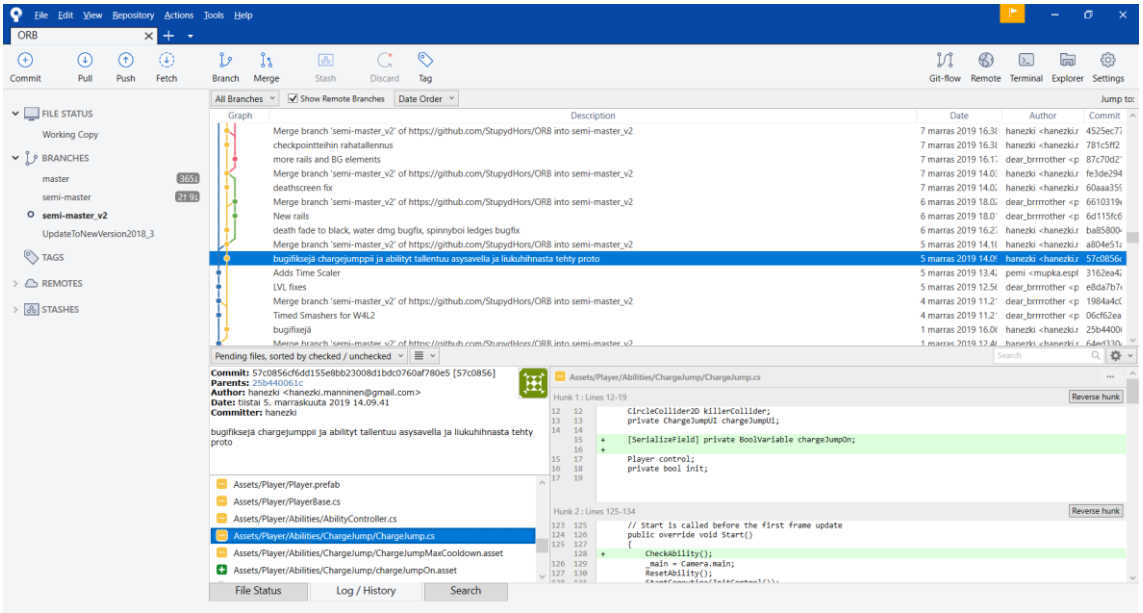
KUVIO 2. Kuva, Unity-pelimoottorin käyttöliittymä sekä projektimme Unity-pelimoottorin sisällä

Hack n' Plan (kuvio 3) on projektinhallintatyökalu, jolla projektin jäsenet voivat yhdessä tehdä tehtäviä ja seurata niiden etenemistä todella helposti. Tämä auttaa koko tiimiä hahmottamaan, missä vaiheessa projekti on ja mitä pitäisi priorisoida, ja se myös helpottaa projektin aikatauluttamista. Yksilöllisellä tasolla se helpottaa työntekijää katsomaan, mitä tehtäviä hänellä on, missä vaiheessa ne ovat, milloin niiden pitäisi olla valmiita, ja monia muitakin tietoja. Projektinhallintatyökalu toimii käytännössä suurena muistilistana, jolle voi käydä lisäämässä uusia tehtäviä sitä mukaa, kun niitä huomaa, ja jos pelistä löydetään jokin vika, se voidaan lisätä sinne saman tien, jotta ohjelmoija sitten myöhemmin löytää ja voi korjata sen. Tehtäviin voi lisätä todella paljon erilaisia ominaisuuksia, kuten kenelle tehtävä kuuluu, tehtävän kategoria, alitehtäviä, tehtävän tärkeys, tehtävän aikaraja ja paljon muuta.



KUVIO 3. Hack n' Plan -projektinhallintatyökalun käyttöliittymä.

Sourcetree (kuvio 4) on versionhallintatyökalu, joka on graafinen käyttöliittymä Git-versionhallintajärjestelmälle, jonka avulla voimme työskennellä saman projektin parissa usealta eri koneelta. Tämä tapahtuu niin, että kun joku tekee muutoksia projektiin, niin hän voi lisätä ne muutokset kaikille muillekin versionhallintatyökalua käyttäville. Ongelmia voi silti tulla, jos monet muokkaavat samoja tiedostoja yhtä aikaa, mutta tämän voi kuitenkin välttää hyvällä projektinhallinnalla ja kommunikoinnilla. Sourcetree-versionhallintatyökalun avulla pystyy myös palaamaan vanhempaan versioon, jos nykyisessä menee jotain pahasti rikki. Siinä pystyy myös käyttämään eri kehityshaaroja. Jos esimerkiksi peli on valmis ja siihen halutaan tehdä lisäsisältöä tai korjauksia, ne pystytään tekemään erillisessä kehityshaarassa pääkehityshaarasta, jonka jälkeen ne pystytään yhdistämään päähaaraan.



KUVIO 4. Sourcetree-versionhallintatyökalun käyttöliittymä.

3 PÄIVÄKIRJARAPORTOINTI

3.1 Lähtökohdat raportoinnille

Päiväkirjaraportoinnissa keskityin ohjelmointiin ja siinä vastaan tuleviin ongelmiin ja siihen, kuinka ratkaisin ne ongelmat ja miten toteutin kaikki ominaisuudet. Asetin päiväkohtaisesti tavoitteet, jotka kävin katsomassa Hack n' Plan -projektinhallintatyökalusta, johon lisäilin itselleni tehtäviä päivittäin, kun ne tulivat esille. Päivittäiset tehtäväni olivat yleensä joidenkin uusien ominaisuuksien tekemistä, ohjelmointivirheiden korjauksia tai pelin testaamista. Kirjasin päivien päätteeksi, mitä käytännössä tein ja miten. Viikkojen päätteeksi pohdin aina kulunutta viikkoa ja sitä, mitä olen saanut aikaan, mitä ongelmia tuli vastaan ja miten kehityin ohjelmoijana tai muuten. Päiväkirjaa kirjoitettiin aikavälillä 14.10.2019 – 24.01.2020.

3.2 Seurantaviikko 1

Maanantai 14.10.

Tämän päivän suunnitelmanani oli ensiksi päivittää Unity-pelimoottori uudempaan versioon, korjata viime pelitestauksessa ilmennyt ohjelmointivirhe, jossa pelaajan rahat eivät tallennu oikein silloin, kun pelaaja etenee uuteen kenttään, sekä sen jälkeen suunnitella ja tehdä sellainen systeemi, jossa peliviholliset pudottavat satunnaisesti lisäpanoksia sekä elämäpisteitä antavia pakkauksia pelaajalle poimittavaksi. Pudotussysteemi vaatii kolme vaihetta: Ensiksi täytyi tehdä pudotettavat objektit valmiiksi prefab-peliobjekteiksi, joihin lisätään koodi, jossa on metodi, joka antaa pelaajalle panoksia ja elämäpisteitä haluamamme verran. Toiseksi täytyi kehittää ratkaisu näiden poimimiseen. Tein tämän lisäämällä joko pelaajaan näiden poimimisesta vastaavan koodin, tai sitten lisäsin itse näihin objekteihin koodin, joka huolehtii, että ne tulevat poimituksi. Kolmantena minun täytyi lisätä vihollisten pudotettavien tavaroiden luetteloon nämä kaksi objektia ja lisätä niihin koodi, että ne tippuvat maahan, kun vihollinen kuolee.

Päivä alkoi heti ongelmilla, kun asensin Unity-pelimoottorin version 2019.2.7f2, koska siinä oli ohjelmointivirhe, joka esti projektiamme toimimasta. Sen jälkeen asensin toiseksi uusimman version 2019.1.14f1, joka toimikin moitteettomasti. Sain korjattua rahantallennukseen liittyvän ohjelmointivirheen, joka ohjelmointivirheen korjattua, joka johtui yksinkertaisesti siitä, että kenttää vaihdettaessa rahoja ei tallennettu, vaan rahat tallennettiin vain pelin tallennuspisteillä. Korjasin ohjelmointivirheen yksinkertaisesti lisäämällä kentän vaihdokseen liittyvään koodiin rahantallennuksen. Tämän jälkeen siirryin tekemään panoksien ja elämäpistepakkausten pudotus- ja keräyssysteemiä. Olin ennestään tehnyt peliin pudotussysteemin, jonka pohjaa käytin tähänkin. Vanha systeemi kuitenkin osoittautui riittämättömäksi, joten aloin tekemään sitä uusiksi, että se voisi hoitaa kaiken pudotetun tavaran keräyksen, eikä vain rahan.

Tiistai 15.10.

Tämän päivän suunnitelmana oli jatkaa pudotussysteemin uusimista ja parantelua. Tämä vaatii kolme asiaa. Vihollisten peliobjektien pudotuskoodia täytyi uusia siten, että se ottaa vastaan eri pudotuksia peliobjekteina, mikä mahdollistaa sen, että se ottaa vastaan kaikkia erilaisia pudotuksia, kunhan tein niistä peliobjekteja. Toiseksi minun piti tehdä sitten entisestä rahasta peliobjekteja, jotta ne kelpaisivat tähän uuteen systeemiin. Kolmanneksi minun täytyi parannella systeemiä, jolla pelaaja kerää pudotettuja objekteja maasta.

Päivän aikana sain tehtyä uusiksi koko tavaroiden pudotussysteemin. Tein rahasta neljä eri peliobjektia eri rahamäärille ja muokkasin vihollisten tavaroiden pudotuskoodia siten, että ne ottavat vastaan peliobjekteja ja niille pystyy antamaan satunnaisen mahdollisuuden ja määrän, mitä ne pudottavat ja paljonko. Uusi systeemi toimii siten, että siihen voi Editor-näkymässä lisätä pudotettavia peliobjekteja ja niille prosenttimahdollisuuden ja mahdollisen minimi- ja maksimimäärän, jota mitä voi pudottaa. Kun vihollinen kuolee, niin kutsutaan metodia, joka arpoo pudotettavan rahamäärän, jonka jälkeen käydään läpi, mitä rahayksiköitä pelaajalle pudotetaan. Mahdollisia rahamääriä ovat yksi, viisi, kymmenen ja viisikymmentä. Vihollisen pudottaessa esimerkiksi 113 rahaa, halutaan sen pudottavan sopivasti kaikkia eri rahatyyppejä, eikä vain 2 x 50 + 1 x 10 + 3 x 1, koska halusimme, että rahaa tippuisi fyysisesti enemmän, koska se näyttäisi paremmalta. Ratkaisuna oli tehdä koodi, joka vaatii aina tiettyä rahamäärää yhden pienempää rahayksikköä olevan ainakin kaksi enemmän kuin isompaa, ennen kuin se pudottaa isomman rahan. Tämä toimii käytännössä siten, että se pudottaa ensiksi kaksi yhden arvoista rahaa, sitten yhden viiden arvoisen rahan, sitten taas kaksi

yhden arvoista, sitten jälleen yhden viiden arvoisen ja sen jälkeen yhden kymmenen arvoisen rahan, kunnes haluttu rahamäärä on pudotettu. Tämä johtaa tulokseen, jossa pelaajalle putoaa 13 x 1 + 6 x 5 + 2 x 10 + 1 x 50 rahaa. Alla on kuva kyseisestä koodista (kuvio 5). Tämän jälkeen koodi käyttää Instantiate-metodia rahaobjektien lisäämiseen peliin ja pudottaa ne hienolla räjähdyksellä.

```
//scraps drop
scrapsToDrop = Random.Range(minScraps, maxScraps);

while(scrapsToDrop > 0)
{
    //calculate drops
    if(tenScraps >= fiftyScraps + 2 && scrapsToDrop >= 50)
    {
        // drop fifty scraps
        fiftyScraps++;
        scrapsToDrop -= 50;
    }
    else if (fiveScraps >= tenScraps + 2 && scrapsToDrop >= 10)
    {
        // drop ten scraps
        tenScraps++;
        scrapsToDrop -= 10;
    }
    else if (oneScrap >= fiveScraps + 2 && scrapsToDrop >= 5)
    {
        // drop five scraps
        fiveScraps++;
        scrapsToDrop -= 5;
    }
    else if (scrapsToDrop >= 1)
    {
        // drop one scrap
        oneScrap++;
        scrapsToDrop--;
    }
}
```

KUVIO 5. Koodi, joka laskee, paljonko eri rahoja pudotetaan.

Keskiviikko 16.10.

Tämän päivän suunnitelmana oli lisätä elämäpistepakkausten ja lisäpanosten pudottaminen pudotussysteemiin ja pitää äänisuunnittelupalaveri, jossa käsiteltiin pelimusiikkia ja ääniefektejä. Elämäpistepakkausten ja lisäpanosten lisäämisen pitäisi sujua helposti, koska nyt niille on hyvä systeemi valmiina.

Päivän aikana sain lisättyä elämäpistepakkauksen ja lisäpanokset kerättäviksi objekteiksi. Tarvitsin siinä eräässä kohdassa yksinkertaista koodia, joka arpoo tietyllä prosenttimahdollisuudella, että

tippuuko tavara vai ei. Sain tehtyä tämän helposti Random-luokan Random.value-metodia käyttäen (Unity Dokumentaatio 2019a. Random.value). Kyseisessä koodissa arvon Random.valuella numeron 0.0 ja 1.0 väliltä, ja vertaan sitä mahdollisuuteen pudottaa kyseinen objekti. Jos arvottu numero on sama tai pienempi, niin objekti putoaa. Alla on kuva koodista (kuvio 6).

```
if (Random.value <= healthPackChance)
{
    GameObject loot = Instantiate(healthPack, transform.position, Quaternion.Euler(new Vector3(0, 0, 0))) as GameObject;
    loot.GetComponent<HealthPickup>().DropHealth();
}
```

KUVIO 6. Koodi, joka arpoo, pudotetaanko objekti, ja pudottaa sen.

Tämän jälkeen pidimme pitkän äänisuunnittelupalaverin. Palaverin tavoitteena oli lähinnä hakea oikeanlaista suuntaa musiikille niin kappalevalintojen kuin musiikkiefektien suhteen. Kappalevalinnoilla olisi tarkoitus tukea pelin visuaalista kuvaa sekä tunnelmaa mukaillen eri maailmojen teemoja. Tietyissä kohdissa peliä, kuten loppuvastuksissa, voidaan luoda musiikilla esimerkiksi vaaran tunnetta pelaajalle. Musiikkiefekteillä taas luodaan hetkellisiä muutoksia soivaan musiikkiin. Esimerkiksi pelaajan ollessa lähellä kuolemaa musiikki kuuluu etäisemmin kuin tajunnan rajamailta. Keskustelimme myös siitä miten tärkeitä jotkin ääniefektit pelissä ovat. Esimerkiksi kävelylle emme kokeneet tarpeelliseksi luoda omaa ääniefektiä, sillä pelissä on niin paljon muita aggressiivisempia ääniefektejä, joiden alle se kuitenkin jäisi. Me myös selailimme ääniefektikirjastoamme läpi ja keskustelimme siihen mahdollisesti tulevista lisäyksistä ja muutoksista. Keskustelimme myös ääniefektien teknisestä toteutuksesta siinä määrin, miten ratkaisemme ongelman, joka syntyy usean ääniefektin soidessa yhtä aikaa. Tällöin ääniefektit sekoittuvat toisiinsa saaden ne kuulostamaan huonoilta. Emme vielä löytäneet ratkaisua ongelmaan.

Torstai 17.10.

Tämän päivän tavoitteena oli tehdä muutoksia pelin toisen maailman lopputaisteluun pelitestauksesta saadun palautteen perusteella. Ensimmäinen tehtävä asia oli, että kun loppuvastukselta hajoaa osia, niin täytyisi viestiä paremmin pelaajalle, että ne osat ovat rikki. Monet pelaajat yrittivät vielä ampua osia, kun ne olivat jo hajonneet. Toinen asia oli, että kentässä oleva sähköinen hissi aiheutti liikaa vaikeuksia pelaajille. Tätä täytyisi helpottaa.

Päivän aikana sain tehtyä nämä muokkaukset lopputaisteluun ja siinä samalla huomasin useita ohjelmointivirheitä, jotka korjasin. Aloitin lopputaistelun muokkauksilla. Ratkaisuna ongelmaan,

jossa pelaajat eivät huomanneet osien olevan rikki, oli tummentaa niitä huomattavasti, kun ne hajoavat. Tein tämän yksinkertaisesti muokkaamalla koodista käsin osien SpriteRenderer-komponentissa olevaa väriarvoa osan hajotessa. Sähkölattiaa helpotin lyhentämällä sen käynnissä oloaikaa. Huomasin päivän aikana myös muutaman ohjelmointivirheen. Ensimmäinen ohjelmointivirhe oli, että lopputaistelukentässä oleva hissi välillä hyppäsi pitkästi alaspäin, vaikka sen pitäisi mennä tasaisesti ylös. Löysin ratkaisun tähän ohjelmointivirheeseen kaikissa liikkuvissa alustoissa olevasta MovingPlatform-luokasta. Koodissa on monta vaihtoehtoa liikkumistavalle, jolla alusta liikkuu, esimerkiksi edestakainen liike. Kyseisellä alustalla oli käytössä yhdensuuntainen liike, jossa oli sellainen vika, että kun se asettaa itselleen uuden pisteen, joka kertoo sille, minne mennä, se resetoi vanhan pisteen. Jostain syystä siinä oli koodipätkä, että se samalla hyppäsi takaisin puoliväliin vanhaa pistettä kohti. Poistin vain sen koodipätkän, ja ohjelmointivirhe korjaantui näin. Toinen ohjelmointivirhe oli sellainen, että sähkölattiahissistä tipahti läpi, jos pelaajahahmo jäi sen ja jonkin kentällä olevan kovan objektin väliin. Korjasin tämän laittamalla hissin sisälle KillBox-peliobjektin, joka tappaa pelaajahahmon, jos se on litistymässä hissin läpi. Tämä oli hyvä ratkaisu, koska niin loogisesti tapahtuisi. Kolmas ohjelmointivirhe oli sellaisessa ominaisuudessa, että kun pelaajahahmo kuolee, osa pelaajan rahoista jää pelaajahahmon kuolemispaikkaan, josta pelaajahahmo voi ne kerätä. Kyseessä ei oikeastaan ole ohjelmointivirhe, mutta kuitenkin epähuomiossa tekemättä jäänyt ominaisuus, nimittäin rahoilla, joita pelaajahahmosta jää, ei ollut fysiikoita, joten jos pelaajahahmo kuoli esimerkiksi hissiin, rahat jäivät leijumaan ilmaan, vaikka hissi olisikin alhaalla. Lisäsin niihin fysiikat ja laitoin ne samalle fysiikkatasolle kuin kaikki pudotuksetkin, jotta vain pelaajahahmo, hissit, pelikentän rajat ja muutama muu asia voivat koskea niihin. Tämän jälkeen tutkin äänimiehemme kanssa ratkaisuja siihen, kun joissain tapauksissa samat ääniefektit kuuluvat monesti päällekkäin ja kuulostavat silloin huonolta. Tämä tapahtui esimerkiksi haulikolla ammuttaessa, jolloin monet haulit osuivat viholliseen yhtäaikaisesti. Jonkinlaisena ratkaisuna tähän oli pakottaa nämä äänet kuulumaan samalla äänikanavalla. Aikaisemmin ne oli laitettu monikanavaiselle AudioSource-komponentille, jolloin ne menivät eri kanaville ja soivat yhtä aikaa. Pakotimme siis tällaiset äänet yksikanavaiselle AudioSource-komponentille ja tarkistimme, soiko kyseisellä AudioSource-komponentilla jo ääni. Jos soi, emme soittaneet uutta ääntä.

Perjantai 18.10.

Tämän päivän suunnitelmana oli korjata musiikinhallintapeliobjektin ohjelmointivirhe, jossa musiikkia ei soi ja peli menee rikki, jos pelaa skene-näkymässä, jota ei ole asetettu pelin build-asetuksissa. Lisäksi suunnitelmana oli korjata yksi epähuomiossa tehty asia elämäpistepakkauksista ja

lisäpanoksista. Lisäksi oli tarkoituksena tutkia vielä lisää edellisenä päivänä tekemääni korjausta samanaikaisesti kuuluville äänille.

Päivän aikana sain korjattua musiikinhallintapeliobjektissa olleen ohjelmointivirheen lisäämällä siihen tarkistuksen, että jos kyseinen pelattava skene-näkymä ei ole asetettuna pelin build-asetuksissa, niin soitetaan pelin vakiomusiikkia. Elämäpistepakkauksissa ja lisäpanospudotuksissa oli sellainen ongelma, jossa kenttään itse aseteltavissa, kerättävissä peliobjekteissa ei saisi olla fysiikkaa, mutta vihollisten pudottamissa olisi fysiikka, joten erittelin ne vain kahdeksi eri objektiksi, fysiikalliseksi ja ei-fysikaaliseksi. Huomasin myös päivän aikana ohjelmointivirheen kamerassa eräässä skene-näkymässä, jossa kamera oli väärän kokoinen. Korjasin tämän tutkimalla käyttämämme CineMachine-lisäosan koodia, ja löysin sieltä, että siellä oli Lens.size-arvo, joka pakotti kameran koon tietyksi. Editor-näkymässä pystyi muokkaamaan tätä Lens.size-arvoa oikeaksi. Päivän lopuksi huomasin vielä ongelmia tekemässäni korjauksessa samanaikaisesti kuuluville äänille, joten yritin siihen erilaisia ratkaisuja. Äänet oli pakotettu samalle kanavalle ja ne eivät voineet soida, jos kanavalla soi jo ääni. Tämä aiheutti sellaisen ongelman, että jos ampui nopeasti sarjatulta ampuvalla aseella, niin ehkä vain joka kolmannesta panoksesta kuului ääni, eikä tämä ollut toivottua. Yritin asettaa ajastimen, jossa jokainen panos vertaa osumisaikaansa edelliseen panokseen: jos ajoilla on yli 0,1 sekunnin väli, niin ainoastaan silloin uusi ääniefekti saa soida. Tämäkään ratkaisu ei kuitenkaan toiminut, ja siinä oli lukuisia ongelmia, joten loppujen lopuksi päädyin palauttamaan äänet kuulumaan monikanavaisella AudioSource-komponentilla niin kuin ennenkin ja ajattelin yrittäväni joskus myöhemmin tutkia ja korjata ongelmaa uudelleen, kun siihen on enemmän aikaa.

Viikkoanalyysi

Viikon aikana sain pelin pudotussysteemin oikeastaan valmiiksi, mikä parantaa peliä todella paljon ja vaikka en audiopuolelta saanutkaan koodin suhteen paljon tehtyä, niin opin todella paljon siitä, miten pelin äänet käytännössä toimivat.

Viikon aikana oivalsin, että kun tekee isoja systeemejä, kuten pudotussysteemin, ne pitäisi alun perin suunnitella kunnolla ja miettiä pidemmän päälle, mitä niiden pitää tehdä myös tulevaisuudessa. Ne pitää suunnitella alun perin sillä tavalla, että niistä on helppo jatkaa tai muokata uusia ominaisuuksia eikä tarvitsisi paljon tehdä asioita uusiksi ja eri lailla.

Kun jatkossa teen isoja uusia systeemejä, pyrin suunnittelemaan ja tekemään ne sillä tavalla, että ne ovat helposti laajennettavissa ja kommentoin niitä mahdollisimman hyvin. Se helpottaa vanhan koodin pariin palaamista.

Viikon aikana tunnen kehittyneeni ohjelmoijana, sillä viikolla oli paljon sopivan haasteellisia ohjelmointitehtäviä. Tunnen myös kehittyneeni ohjelmoinnin suunnittelussa paremmaksi, koska tunnistin tekemiäni virheitä ja opin niistä. Lisäksi ymmärrykseni peliäänistä kasvoi todella paljon.

3.3 Seurantaviikko 2

Maanantai 28.10.

Tämän päivän suunnitelmana oli suunnitella ja kehittää ratkaisu sellaiseen ongelmaan, jossa eräs pelin ansa, joka murskaa pelaajahahmon osuessaan siihen, ei toiminut oikein. Ansoja voi yhdellä tasolla olla todella paljon, ja niiden tarkoitus oli toimia tietyssä rytmissä, mutta ne menevät pois synkroniasta aina pikkuhiljaa.

Päivän aikana yritin monia ratkaisuja kyseiseen ongelmaan, kuten tehdä uusiksi koodipätkän, joka liikuttaa ansoja. Se ei korjannut ongelmaa, mutta koodista tuli selvempi ja optimoidumpi, mikä on kuitenkin hyvä. Toinen ratkaisu, jota yritin, oli mitata aikaa, joka ansoilla kesti toteuttaa niiden liikerata. Jos ansat laukesivat vaikkapa 0,01 sekuntia liian nopeasti tai hitaasti, niin seuraavalla kierroksella niiden aikaa säädettiin sen mukaisesti. Tämä paransi ongelmaa todella paljon, mutta ei siltikään korjannut sitä täysin. Ansat eivät enää aina menneet ollenkaan pois synkroniasta, mutta kuitenkin joskus menivät. Lopulta päädyin ratkaisuun, jossa teen näille ansoille oman hallintakoodin, joka ohjaa niitä, jolloin niiden pitäisi pysyä synkroniassa ja niille voi antaa minkä tahansa rytmin. Minulla ei kuitenkaan ollut aikaa aloittaa tätä ratkaisua tänä päivänä.

Tiistai 29.10.

Tänä päivänä meillä oli tarkoituksena pitää iso pelisuunnittelupalaveri, jonka jälkeen aloitin ansojen hallintakoodin tekemisen. Ansojen hallintakoodin tarkoituksena oli saada ansat toimimaan synkronoidusti keskenään.

Pelisuunnittelupalaveri kesti kauan, ja mietimme paljon aseiden tasapainottelua keskenään ja panos- ja rahaekonomiaa sekä sitä, millainen viimeinen vihollinen pelissä olisi. Täyttelimme myös pelisuunnitteludokumentteja ja taulukoita. Tämän jälkeen ehdin vielä jonkin verran suunnitella ansojen hallintakoodia.

Keskiviikko 30.10.

Tämän päivän tarkoituksena oli tehdä pelaajahahmon murskaaville ansoille hallintakoodi, joka varmistaa, että ne pysyvät synkroniassa ja että niille voi antaa minkälaisen rytmin tahansa. Hallintakoodi oli paras ratkaisu tähän ongelmaan, koska se on varmin ja yksinkertaisin tapa ratkaista ongelma.

Päivän aikana sain tehtyä ansojen hallintakoodin perusasiat sekä ensimmäisen järjestysvaihtoehdon, joka laukaisee ansoja satunnaisessa järjestyksessä. Hallintakoodissa on lista ansoista, jotka voi asettaa siihen Editor-näkymässä, sekä nopeusvaihtoehdot ansan eri vaiheisiin. Satunnaisessa järjestyksessä laukeavissa ansoissa täytyy varmistaa, ettei jokin ansa ole jo päällä, kun se arvotaan uudelleen. Aina täytyy siis valita jokin vireillä olevista ansoista. Ansat palautuvat takaisin vireeseen lauettuaan. Ratkaisin tämän tekemällä bool-muuttujalistan ansoista, jotka ovat vireessä ja jotka ovat lauenneet. Sitten vertaan arpomaani ansaa tähän listaan, jos se on vireessä, koodi hyväksyy sen ja ansa laukeaa, jos ei niin arvotaan uudelleen (kuvio 7.).

```
while(currentlySmashing[randomSmasher] == true)
{
    randomSmasher = Random.Range(0, smashers.Count);
    yield return new WaitForSeconds(0.1f);
}
```

KUVIO 7. Koodi, joka tarkistaa, onko kyseinen ansa käytössä.

Torstai 31.10.

Tämän päivän tarkoituksena oli viimeistellä ansojen hallintakoodi lisäämällä siihen vaihtoehto tietylle järjestykselle, jossa ansat laukeavat, sekä vaihtoehdot, että laukeavatko ne ajastetusti vai sitten, kun edellinen on palautunut takaisin vireeseen. Ansojen järjestysvaihtoehto on tärkeä tehdä, että tasorakentaja voi helposti ja nopeasti tehdä hyviä ansoja kenttiin.

Päivän aikana sain hallintakoodin täysin valmiiksi. Lisäsin koodiin järjestysvaihtoehdon tekemällä array-taulukon, johon voi Editor-näkymässä syöttää järjestyksen asettamalla ansoja taulukkoon. Koodi käy sitten läpi tämän taulukon ja laukaisee ansoja järjestyksessä (kuvio 8.).

```
smashers[smasherOrder[orderSmashInt].SmasherIndex].StartSmash(smashSpeed, returnSpeed, stayDownTime);
```

KUVIO 8. Koodi, joka laukaisee seuraavan ansan järjestyksen mukaan.

Perjantai 1.11.

Tämän päivän tarkoituksena oli korjata yksi ohjelmointivirhe ansojen hallintakoodista, joka tuli silloin kun järjestyksessä laukeavissa ansoissa alkaa uusi kierros ennen kuin ensimmäinen oli ehtinyt palata vireeseen. Lisäksi minun täytyy myös korjata ohjelmointivirhe, jossa pelaajahahmon pudottamat rahat jäävät vääriin paikkoihin pelaajahahmon kuollessa, joka johtui siitä, että pudotetut objektit pystyivät jäämään hisseihin.

Päivän aikana tein ensiksi sen, että pelaajahahmon pudottamat rahat jäävät oikein maahan tämän kuollessa. Ratkaisuna oli vaihtaa se, että miten se tunnistaa viimeisen maakontaktin ennen kuolemaa. Ennen se tunnisti sen BoxCollider-komponentilla, joka etsi osumia maasta. Tämä ei kuitenkaan toiminut hyvin sillä, jos pelaaja pyöri, niin tämä tunnistin osui myös seiniin ja kattoon. Käytin siis RayCast-ominaisuutta osumien etsimiseen suoraan pelaajan alapuolelta, ja se toimi hyvin. Seuraavaksi korjasin ohjelmointivirheen järjestyksessä laukeavissa ansoissa ja lisäsin siihen lisäominaisuuden, joka tuli puheeksi päivän aikana tasosuunnittelijamme kanssa. Tämä ominaisuus oli ansojen järjestyksen muuttaminen käänteiseksi aina kierroksen loputtua. Tässä koodipätkässä näkyy ohjelmointivirheen korjaus, jossa ansat odottavat, jos eivät ole vielä vireillä sekä ansojen suunnan muuttaminen aina kierroksen loputtua (kuvio 9.).

```
if(orderSmashInt > smasherOrder.Length - 1)
{
    orderSmashInt = 0;
    if (pingPong)
    {
        System.Array.Reverse(smasherOrder);
    }
    while (currentlySmashing[smasherOrder[orderSmashInt].SmasherIndex] == true)
    {
        yield return new WaitForSeconds(0.1f);
    }
}
```

KUVIO 9. Koodi, joka huolehtii ansojen järjestyksestä.

Viikkoanalyysi

Tällä viikolla sain tehtyä hyvän ja toimivan hallintakoodin pelin tietyille ansoille, mikä mahdollistaa paljon monipuolisemmat ansojen käyttäytymistavat kuin ennen. Lisäksi saimme suunniteltua hyvin asioita palaverissa.

Jälleen ongelmana oli se, että vanha systeemi oli tehty huonosti ja vaati isoja muutoksia toimiakseen oikein. Kyseiset ansat tehtiin ennen kuin monia pelin tasoja oli suunniteltu, joten niissä ei ollut varmistettu toimivuutta tiettyjen tilanteiden suhteen ja ne oli jätetty pääosin testaamatta.

Viikon aikana opin myös todella ison asian siitä, että pelin Coroutine-metodit eivät toimi täysin synkronoidusti keskenään (StackExchange 2014. Timing of coroutines in Unity is never precise), joten niillä ei ikinä kannata tehdä erillisten peliobjektien toimivuutta keskenään, jos niiden on tarkoitus toimia millään tavalla synkronoidusti. Tämä johtuu siitä, että Coroutine-metodit ovat täysin Update-funktion armoilla, jolloin niihin vaikuttaa kuvataajuus. Jos minun täytyy vaikka Coroutine-metodissa kutsua WaitForSeconds-metodia 0,1 sekunnin välein, koska minun täytyy vaikkapa liikuttaa jotain objektia 0,1 sekunnin välein, niin itse asiassa se ei kestä 0,1 sekuntia vaan 0,1 sekuntia ja lisäksi edellisen kuvataajuuden päivityksen vaatiman ajan, joka vaihtelee, koska kuvataajuus vaihtelee. Vaihtoehtoja tälle on tehdä hallintakoodi kyseisille objekteille, jolloin voi käyttää Corou-

tine-metodia vapaasti, koska hallintakoodi huolehtii automaattisesti synkroniassa pysymisen. Toinen vaihtoehto on varmaankin laittaa kaikkien koodi FixedUpdate-funktioon, joka ajetaan aina 60 kertaa sekunnissa. Tällöin niiden pitäisi käytännössä pysyä synkroniassa keskenään.

Viikon aikana koen kehittyneeni ohjelmoijana, vaikka jotkin viikon ohjelmointitehtävät olivatkin minulle erittäin haasteellisia. Opin ymmärtämään paljon paremmin, miten Coroutine-metodit toimivat, ja muutenkin kehityin minulle jo rutiininomaisissa ohjelmointiasioissa paremmaksi.

3.4 Seurantaviikko 3

Maanantai 4.11.

Päivän tavoitteena oli päivittää pelin tallennussysteemiä sekä tehdä pelaajahahmolle kuolemaruutu, jossa peli hiipuu tummaksi pelaajan kuoltua sekä parannella ja korjailla pelaajahahmon erikoiskykyä sekä tutkia, että miksi liikkuvat hissit menevät pois synkroniasta pikkuhiljaa.

Päivän aikana sain päivitettyä tallennussysteemin niin, että se tallentaa myös pelaajahahmon erikoiskyvyt. Käytämme EasySave3-lisäosaa tallentamiseen. Tämän jälkeen aloitin pelaajan erikoiskyvyn parantelun. Erikoiskyky on sellainen, että pelaaja hidastaa aikaa samalla kun hyppää moninkertaisella vauhdilla normaalihyppyyn verrattuna kohteeseensa. Pelaaja voi kimpoilla seinillä sekä aiheuttaa vihollisille vahinkoa kyseisellä ominaisuudella. Parantelin tämän ominaisuuden toimivuutta tekemällä siitä hieman hitaamman ja säätämällä sen törmäyksen ennustus arvoja. Korjasin siitä myös joitain ohjelmointivirheitä, joita olivat se, että pelaaja pystyi ottamaan vahinkoa kesken kyvyn käytön, vaikka pelaajan ei pitäisi silloin ottaa vahinkoa ja kyky ei toiminut kunnolla pelin lukuisten ansojen kanssa. Tämän jälkeen tutkin pois synkroniasta meneviä hissejä. En keksinyt ratkaisua ainakaan heti niiden korjaamiseen ja se vaikuttaa niin aikaa vievältä, että päätimme tehdä niille vaihtoehtoisia ratkaisuja peliin ja vähennämme sillä tavalla niiden määrää sekä hidastamme niitä. Tämä helpottaa ongelmaa, että ne menevät pois synkroniasta. Yksi vaihtoehtoinen ratkaisu oli tehdä liukuhihna, jonka lisäsinkin työlistalleni. En päivän aikana saanut aloitettua ollenkaan uutta kuolemaruutua ja päätin siirtää sitä myöhemmäksi.

Tiistai 5.11.

Päivän tarkoituksena oli tehdä peliin liukuhihna vaikeuttamaan tiettyjä esteitä pelissä. Liukuhihna on osittain korvike sivuttain liikkuville hisseille ja se mahdollistaa uudenlaisia ansa-alueita, joka ylläpitää pelaajan mielenkiintoa.

Päivän aikana sain tehtyä ihan hyvin toimivan version liukuhihnasta, ongelmina oli pääasiassa se, että pelaajahahmon fysiikat toimivat erillä tavalla pallomuodossa ja kävelymuodossa. Kokeilin päivän aikana monia ratkaisuja pelaajahahmon liikuttamiseen ja loppujen lopulta päädyin käyttämään RigidBody-komponentin sisäistä AddForce-metodia pallon liikuttamiseen ja kävelymuotoon liikuttamalla hahmon Transform-komponenttia sen sisäisellä Translate-metodilla. Kaikkiin muihin objekteihin pelissä sovelsin aina jompaakumpaa noista tavoista ja sain kaiken toimimaan lopulta hyvin.

Keskiviikko 6.11.

Tämän päivän tarkoituksena oli vähän testailla liukuhihnaa sekä tehdä tummaksi hiipuvaa kuolemaruutu animaatiota ja sen jälkeen vielä vähän etsiä ja korjata ohjelmointivirheitä. Tämän kaltainen ohjelmointivirheiden etsiminen oli tärkeää aina silloin tällöin, että peli olisi mahdollisimman stabiili.

Päivän aikana testasin liukuhihnaa ja siitä löytyi ohjelmointivirhe, että jos hihna on vinossa niin se ei oikein toimi, se nimittäin aina työntää asioita suoraan vasemmalle tai suoraan oikealle. Keskustelin asiasta kuitenkin muiden kanssa ja oltiin sitä mieltä, että hihnan ei koskaan ole tarkoituskaan olla vinossa, joten ohjelmointivirhe ei haittaa. Tämän jälkeen tein tummaksi hiipuvan kuolemaruutu animaation. Tein sen yksinkertaisesti koko ruudun kokoisella mustalla kuvalla, jonka alpha-arvoa muokkaan animaatiossa, jossa se menee pelaajan kuollessa tyhjästä täyteen ja pelaajan syntyessä henkiin täydestä tyhjään tietyssä ajassa. Se toimii oikein hyvin. Löysin muutaman ohjelmointivirheen samalla kun testasin tummaksi hiipuvaa kuolemaruudun animaatiota, vesi ei enää tappanut pelaajaa, jos pelaaja meni sinne kyvyllä, jolloin pelaaja ei voi ottaa vahinkoa, vesi kuitenkin kuuluu asioihin, joiden täytyy tappaa pelaaja milloin tahansa, joten korjasin sen tappamaan pelaajan kaikissa tilanteissa. Toinen ohjelmointivirhe oli, että eräs vihollinen ei pystynyt kävelemään rinteitä alas, koska se luuli tippuvansa alas, joten korjasin sen.

Torstai 7.11.

Päivän tarkoituksena oli tehdä pelin tallennuspisteisiin mekaniikka, jossa pelaajan keräämät rahat tallentuvat pankkiin. Pelissä on vain rahat pelaajalla koko ajan ja jos pelaaja kuolee, hän menettää rahansa. Tämä mekaniikka helpotti peliä ja rahan keräämistä, sillä monet pelaajat kuolevat niin useasti, että he vain menettävät koko ajan kaikki rahansa eivätkä etene.

Päivän aikana sain tehtyä tämän rahantallennussysteemin tallennuspisteisiin lähes valmiiksi. Systeemi toimii siten, että kun pelaaja osuu tallennuspisteeseen ensimmäistä kertaa, pelaaja tiputtaa kaikki rahansa maahan samanlaisella logiikalla kuin viholliset pudottavat rahaa, jonka jälkeen ne imeytyvät tallennuspisteeseen satunnaisella viiveellä ja tallentuvat pankkiin.

Perjantai 8.11.

Päivän tarkoituksena oli viimeistellä rahantallennussysteemi ja sitten päivittää pelaajahahmon ChargeJump-kykyä niin, että sillä voi kimpoilla seinistä useita kertoja. Rahan piti tallentua oikein ja säilyä pelisessioiden välillä.

Päivän aikana sain viimeisteltyä rahantallennuksen kokonaan ja testailin sitä paljon ja se toimi hyvin. Rahat tallentuivat oikein ja säilyivät pelisessioiden välillä oikein. Tämän jälkeen aloin tutkimaan ChargeJump-kyvyn koodia, jonka olen tehnyt yli puoli vuotta sitten ja se osoittautui aika haasteelliseksi, koska koodissa oli yli 1400 riviä koodia ja olin kommentoinut sen huonosti. Koodissa oli myös paljon mielestäni epäselkeitä ja huonoja ratkaisuja tiettyihin asioihin. Sain suunnilleen selvitettyä, että miten aion toteuttaa säädeltävän kimpoilu mahdollisuuden mutten ehtinyt tehdä sitä vielä. Huomasin koodia tutkiessani kehittyneeni ohjelmoijana todella paljon puolessa vuodessa.

Viikkoanalyysi

Tällä viikolla sain tehtyä paljon parannuksia pelin tallennussysteemille ja liukuhihnan. Rahantallennussysteemin muutokset tein, koska pelitestauksessa huomattiin, että peli on monille liian vaikea ja he eivät saa kerättyä rahaa eivätkä siten myöskään voi parannella aseitaan tai pelihahmoaan, joka vaikeuttaa peliä entisestään. Liukuhihna tehtiin koska oli erinäisistä syistä tarve uudelle pelaajaa liikuttavalle objektille, ennen oli vain hissit.

Tällä viikolla törmäsin uudelleen ongelmaan pois synkroniasta menevien peliobjektien kanssa. Ne menivät pois synkroniasta tismalleen samasta syystä kuin ansat, mutta en tehnyt niille hallintakoodia, koska en keksinyt, että miten toteuttaisin sen, koska ansat, joiden oli tarkoitus toimia synkroniassa, olivat aina yksi yhtä kokonaisuutta suhteellisen pienellä alueella. Hissit puolestaan ovat hajanaisesti ympäri karttaa ja nykyään niillä kestää kuitenkin ainakin kymmenen minuuttia mennä pois synkroniasta ja yleensä pelaaja joko läpäisee kentän tai kuolee sitä ennen, joten sillä ei ole niin paljon väliä. Ansat menivät erittäin nopeasti pois synkroniasta ja olivat paljon huomattavampia, joten ne vaativat korjaamista.

Huomasin myös viikon aikana ChargeJump-kyvystä, että olin mielestäni tehnyt sen todella huonosti. Siinä on paljon enemmän koodia, mikä on tarpeen, koska olin tehnyt siitä alun perin todella rikkinäisen ja sitten korjaillut kaikkia mahdollisia vikoja nopeasti ja huonosti. Nyt sitten kun siihen pitäisi tehdä parannuksia niin alkoi selviämään, että olen vain ampunut itseäni nilkkaan tässä asiassa, kun olen tehnyt sen huonosti. Toisaalta jos nykyään tekisin sen uudelleen, niin osaisin tehdä sen paljon paremmin kuin ennen, huomaan siis kehittyneeni todella paljon ohjelmoijana.

Jatkossa pyrin korjaamaan ohjelmointivirheet oikein enkä hutiloiden. Käytännössä tämä useasti tarkoittaa sitä, että teen jonkin ominaisuuden ihan uudella tavalla alusta, kunhan se on tarpeeksi pieni eikä aikaa vievä, enkä väkisin korjaa sitä jollain huonolla tavalla. ChargeJump-kyvyn koodi oli myös kommentoitu todella huonosti ja meni todella paljon aikaa tutkia taas se koko kokonaisuus, että miten se toimi.

Viikolla koin taas kehittyneeni ohjelmoijana, sillä viikon tehtävät olivat sopivan haastavia, että niiden kanssa ei jäänyt jumiin, mutta vaati kovasti miettimistä ja yrittämistä, että sain ne tehtyä.

3.5 Seurantaviikko 4

Maanantai 11.11.

Tämän päivän hommana oli sitten tehdä ne ChargeJump-kyvyn parannukset sekä korjata eräs liukuhihnassa ilmennyt ohjelmointivirhe. Liukuhihnan ohjelmointivirhe oli sellainen, että kaikki viholliset eivät tunnista sitä, jolloin ne kääntyvät poispäin siitä, eivätkä kävele sen päälle.

Liukuhihnan ohjelmointivirhe oli aika yksinkertainen, kaikki viholliset eivät vain tunnistaneet sitä maaksi, eivätkä osanneet mennä kävelemään siihen, korjasin tämän asettamalla liukuhihnan Ground-fysiikkatasolle. Samalla huomasin, että yhteen viholliseen ei vaikuttanut painovoima ollenkaan. Viholliseen ei tarkoituksella vaikuttanut Unity-pelimoottorin oma fysiikkamoottori, että se toimisi paremmin, joten lisäsin sen koodiin, että jos se ei tunnista maata allaan, koodi liikuttaa sitä alaspäin. Tämän jälkeen aloin tekemään ChargeJump-kykyyn parannuksia. Annoin sille muuttujan, johon saa säätää, että monestiko haluaa, että kyky kimpoaa seinistä, ja sitten loin kokonaan uuden ja paremman metodin kimpoilulle, niin että minulla on eri metodi kimmotuksille ja ihan viimeisellä kimmotukselle. Sitten vain miinustin kimmotusten määrää jokaisella kimmotuksella, ja kun se on 0, kyseessä on viimeinen. Vaikea osuus oli kuitenkin tehdä tämä uusi metodi kimmotuksille sillä olin käyttänyt koodissa 20 eri bool-muuttujaa, jotka määrittelevät monimutkaisissa If-lausekkeissa, että mitä pitäisi milloinkin tehdä, ja huonon kommentoinnin vuoksi oli melkoista testailua saada metodi toimimaan halutulla tavalla. Opin tästä kyllä, että täytyy kommentoida paremmin ja miettiä fiksumpia ratkaisuja vaikeisiin pulmiin, kuin loputon määrä bool-muuttujia ja If-lauseita. Olen kuitenkin kehittynyt todella paljon ohjelmoijana siitä, kun tein tämän kyseisen koodin.

Tiistai 12.11.

Tämän päivän teemana oli harjoitella partikkeliefektien käyttöä tekemällä ChargeJump-kyvylle, sekä loppuvastukselle, joka kyseisen kyvyn pelaajalle antaa, omat hienot partikkeliefektit ChargeJump-kykyyn liittyen. En ollut kovin paljoa harjoitellut partikkeliefektien käyttöä tätä ennen.

Päivä oli erittäin mielenkiintoinen minulle ja partikkeliefektit osoittautuivat todella hauskoiksi tehdä. Testailin todella paljon erilaisia vaihtoehtoja ja opin todella paljon ihan vain testaamalla. Aloitin kuitenkin katsomalla nopean kymmenen minuutin oppaan, että kuinka tehdään Trail-efektejä (Abhinav a.k.a Demkeys 2017. Particle System Trails | Unity Particle Effects | Visual FX Tärkein asia Trail-

efektiin liittyen oli, että partikkeli efekti täytyi laittaa tuottamaan partikkeleita matkan perusteella, ei ajan. Sen jälkeen kyse oli vain siitä, että testailin kaikkia eri asetuksia ja värejä löytääkseni hienon yhdistelmän. Lempiasetukseni oli Noise, joka lisää ikään kuin turbulenssia partikkeliin ja vääntelee sitä eri suuntiin, tämä oli todella hieno efekti. Halusin myös, että pelaajan ympärillä olisi myös pelaajan kokoinen partikkeli ja lisäksi pelaajan perään jäisi pitkä viiva, joten lisäsin kaksi eri partikkelia, ja lopputulos oli mielestäni todella hieno (kuvio 10.).

KUVIO 10. Kuva partikkeliefektistä ChargeJump-kyvylle.

Tämän jälkeen lisäsin samantyylisen partikkeliefektin loppuvastukselle, joka antaa kyseisen kyvyn pelaajalle. Tämä muutos parantaa loppuvastuksen pelikokemusta paljon.

Keskiviikko 13.11.

Tänä päivänä tavoitteena oli tehdä tallennussysteemi myös pelaajan ostamille parannuksille ja aseille sekä lisätä päävalikkoon vaihtoehto nollata peli halutessa, koska olimme suunnittelemassa julkaista päivitetyn demoversion tällä viikolla.

Päivän aikana sain lisättyä tallennussysteemin aseille, joka olikin vähän monimutkaisempi, kuin muut tallennussysteemit, koska eri aseiden parannus vaihtoehtoja on niin paljon. Tein lopulta aseille oman hallintakoodin, joka huolehtii tallentamisesta, aina kun pelaaja ostaa uuden parannuksen, lähetetään käsky hallintakoodille tallentaa kyseinen tallennus ja aina kun pelaaja käynnistää pelin, hallintakoodi lataa tallennuksista tallennetut parannukset ja lisää ne pelaajalle. Lisäksi tein resetointi napin pelin päävalikkoon, joka nollaa pelaajan kuolemat, rahat, aseiden parannukset

sekä pelaajan saamat kyvyt. Tämä vaati lisätä sellaisen reset koodin jokaiselle näistä asioista ja kutsua niitä kaikkia sitten yhdestä napista.

Torstai 14.11.

Tarkoituksena oli tänään julkaista uusi demoversio ja minulla oli paljon pieniä muutoksia vähän kaikkialta siihen liittyen, mitä minua pyydettiin tekemään. Tehtäviä oli helpottaa demoversiossa olevaa loppuvastusta, tuplata pelissä saatavan rahan määrä, korjata ohjelmointivirhe lentäviltä vihollisilta, missä ne eivät tunnista pelaajaa, korjata eri tönäisy mekaniikkoja pelissä, korjata joissain kentissä soiva väärä musiikki ja lisätä kameran tärisytys efekti lukuisiin eri paikkoihin.

Päivän aikana sain tehtyä aika lailla nämä kaikki asiat. Aloitin helpottamalla loppuvastusta hidastamalla sen tekemää rynnäkköhyökkäystä, tein loppuvastuksen käyttäytymisestä kuitenkin paljon satunnaisemman, joten loppujen lopuksi siitä ei välttämättä tullut edes yhtään helpompi, mutta todella paljon hauskempi. Tämän jälkeen korjasin ohjelmointivirheen lentäviltä vihollisilta, että jos ne syntyvät pelaajan vieressä, ne eivät tunnista pelaajaa. Tämä korjaantui vaihtamalla niiden alustus koodit Awake-metodiin Start-metodista, joka ratkaisi ongelman. Tämän jälkeen keskityin korjaamaan pelissä olevia eri tönäisy mekaniikkoja, jotka tönäisevät pelaajaa johonkin suuntaan, monet niistä pystyivät tönäisemään pelaajan seinien läpi ja olivat kaikin puolin liian voimakkaita, joten vaimensin niitä kaikkia, ne tarvitsivat myös eri asetukset riippuen siitä, onko pelaaja pallo vai kävelymuodossa. Pallomuodon tönäisy vaati myös erilaisen koodin kokonaan. Tämän jälkeen laitoin kaikissa kentissä soimaan oikeat musiikit ja loppupäivän tein kameran tärisytys efektin Cine-Machine-lisäosalla, johon löysin YouTubesta loistavan lyhyen ohjevideon (Lumidi Developer 2018. Camera Shake In Cinemachine - Unity Mini Tutorial), jonka lisäsimme kenttään, jossa vesi nousee nopeaa ja jahtaa pelaajaa, kameran tärisytys luo kiireen tunnelmaa ja vaikutelman, että paikka olisi romahtamassa, joka sopii kyseiseen kenttään todella hyvin.

Perjantai 15.11.

Päivän tarkoituksena oli viimeistellä pelin demoversio ja testata sitä. Minun täytyi enää korjata yhdeltä viholliselta ohjelmointivirhe, jossa se ei osannut kävellä rinteitä alas, lisätä kenttäsuunnittelijamme tekemät lopputekstit peliin sekä korjata päävalikon napeissa olevia virheitä.

Päivän aikana sain ensimmäisenä korjattua viholliselta ohjelmointivirheen, jossa se ei pystynyt kävelemään rinteitä alas, ohjelmointivirhe johtui taas siitä, että vihollinen tarkisti RayCast-ominaisuudella, tippuuko se alas ja RayCast-ominaisuus oli vähän väärin aseteltu, jolloin se luuli rinteitä pudotuksiksi. Tämän jälkeen lisäsin lopputekstit peliin. Tein tämän tekemällä ihan uuden skene-näkymän lopputeksteille, jossa animaatio menee automaattisesti päälle. Animaation päätyttyä pelaaja ohjataan päävalikkoon. Minulla oli hieman ongelmia tämän kanssa, koska tasojen välillä liikkuminen oli rakennettu UI-peliobjektiin kiinni, jolloin minun täytyi lisätä meidän UI-peliobjektimme skene-näkymään, tämä kuitenkin vaikeutti kaikkea, sillä UI-peliobjekti vaatii skene-näkymässä olevan useita eri referenssejä, kuten pelaajahahmo, joten jouduin lisäämään kaikki nämä skene-näkymään, että sain sen toimimaan oikein, lopulta se kuitenkin toimi ihan oikein. Tämän jälkeen korjasin UI-peliobjektissa olevia virheitä, jotka olivat lähinnä sitä, että monet napit toimivat väärin, ne esim. valikoituivat, vaikka niiden vierestä klikkasi. Nämä korjatakseni minun piti käydä niitä vain yksitellen läpi ja asetella niiden sisäisiä komponentteja oikein. Viimeisenä testasin sitten peliä, että toimiiko se hyvin. Demoversion täytyi tietysti olla stabiili ja ainakin vakavista ohjelmointivirheistä vapaa. En testaamisen aikana löytänyt kuin muutaman ohjelmointivirheen, jotka olivat liian harvinaisia, vaikeita korjata tai ei tarpeeksi vakavia, että niitä täytyisi ainakaan tähän demoversioon korjata. Demoversio siis toimi hyvin ja se julkaistiin.

Viikkoanalyysi

Tällä viikolla kokonaisteemana oli selvästi demoversion tekeminen, joka vaatii aivan erilaista tekemistä kuin normaalit työviikot. Kaikki täytyy tarkistaa useaan kertaan, että ne toimivat oikein ja paljon pitää testata kaikkea ja miettiä että mitä vielä puuttuu ja mitä vielä pitää tehdä. Tällä viikolla tuli paljon ylityötunteja juuri siksi, että kerkesi kaiken laittaa demoversioon. Saimmekin kuitenkin demoversion valmiiksi ja julkaistua suunnitelmien mukaisesti, joka oli hyvä.

Viikon aikana sain tehtyä ChargeJump-kyvyn parannukset, lukuisia pieniä parannuksia vähän kaikkialle, lukuisia ohjelmointivirheiden korjauksia, aseiden tallennussysteemin sekä vähän visuaalista parantelua kuten partikkeliefektin ChargeJump-kyvylle ja sen antavalle viholliselle ja kameran tärisytys efektin, tällaiset visuaaliset parannukset saivat pelin näyttämään ja tuntumaan paljon paremmalta.

Viikon aikana opin todella paljon partikkeliefektien tekemisestä ja käyttämästämme EasySave3-lisäosasta. Partikkeliefektit varsinkin osoittautuivat todella mielenkiintoisiksi ja hauskoiksi tehdä.

Koen kehittyneeni viikon aikana ohjelmoinnissa taas aika paljon, mutta erityisesti kehityin Unity-pelimoottorin partikkelisysteemin käytössä, nyt tunnen oikeasti ymmärtävän, että miten se toimii ja osaisin ilman ohjeita aika hyvin jo tehdä monenlaisia partikkeleita, kunhan ne eivät ole liian edistyneitä.

3.6 Seurantaviikko 5

Maanantai 18.11.

Päivän tarkoituksena oli lisätä vasta tekemäni kameran tärisytys efekti myös ChargeJump-kykyyn, koska se sopi siihen oikein hyvin. Tämän jälkeen aloin tutkimaan asesysteemiämme tarkemmin, koska seuraaviin tehtäviini kuului tehdä muutama uusi ase.

Päivän aikana sain lisättyä kameran tärisytys efektin ChargeJump-kykyyn, joka teki siitä todella paljon voimakkaamman tuntoisen ja mukavamman käyttää. Tämän jälkeen ryhdyin heti tutkimaan asesysteemiämme ja aloitin sen katsomalla tutoriaali videon mihin asesysteemimme pohjautuu (Webcam 2017. Unity Tutorial - Create a Customizable Weapons System with Modular Code: Part 1). Tämän jälkeen aloin työstämään parannuksia kranaatinheitin aseelle. Tein sille valmiit parannusvaihtoehdot sen tekemälle vahingolle, ampumisvoimalle sekä räjähdyksen laajuudelle. Lisään nämä ominaisuudet ja aseen myöhemmin kauppaan ostettavaksi. Aloin myös tekemään sen viimeistä päivitystä, joka on oikeastaan eri ase, nimittäin rypälepommia. Sen olisi tarkoitus räjähtäessään ampua ympärilleen x määrä pienempiä pommeja. Tämä vaatii lisäksi uusien partikkeliefektien tekemistä eri räjähdyksille.

Tiistai 19.11.

Päivän tarkoituksena oli saada rypälepommi parannus kranaatinheittimeen valmiiksi sekä lisätä sopiva laser-efekti laser-ansoihin, tällä hetkellä ne olivat vain punaisia viivoja. En ollut vielä varma, miten toteuttaisin laserille uuden grafiikan.

Päivän aikana aloin ensiksi tekemään laser-efektiä laser-ansoihin. Aluksi minun täytyi paljon miettiä ja googlettaa, että kuinka teen kyseisen efektin. Ensiksi löysin paljon ohjeita, että kuinka sen voisi tehdä Shader-ominaisuudella, mutta ne vaativat aina jonkin maksullisen plugin-lisäosan Unity-pelimoottorin Asset Storesta. Lisää tutkittuani keksin, että voisin tehdä ne vain partikkeleilla, ensiksi mietin, että minun täytyy jotenkin luoda partikkeli pienen välimatkan välein koko laserin matkalle, mutta sitten keksin kokeilla yksinkertaisempaa keinoa eli ampua vain laserin lähtöpisteestä nopeaa vauhtia partikkeleita laserin osoittamaan suuntaan. Kovalla säätämisellä sain siitä todella hienon näköisen ja hyvin toimivan efektin (kuvio 11.).

KUVIO 11. Kuva laserin partikkeliefektistä.

Tämän jälkeen aloin tekemään rypälepommia kranaatinheittimelle. Päivän aikana sain tehtyä sen mekaanisesti aika lailla valmiiksi. Se toimii siten, että kun se räjähtää niin se synnyttää X määrän pienempiä pommeja, jotka lentävät satunnaisiin suuntiin satunnaisella nopeudella ja räjähtävät 0.1–1 sekunnin kuluttua tai osuessaan vihollisiin. Seuraavaksi minun täytyy tehdä sillekkin uudet partikkeliefektit, jotta se näyttäisi paremmalle. Suunnittelimme myös päivän aikana myös muita

aseita kuten haulikon paranneltua muotoa, ja meillä oli paljon ideoita siihen, kuten että se ampuisi vihollisiin automaattisesti ohjautuvia luodin kokoisia miniohjuksia.

Keskiviikko 20.11.

Päivän suunnitelmana oli tehdä kranaatinheittimen eri räjähdyksille omat partikkeliefektit, parannella eilen tekemääni laser-ansan efektiä niin että siinä ei olisi rakoja sekä aloittaa uuden laser-aseen tekeminen. Uuden laser-aseen täytyy pystyä ampumaan vihollisten läpi ja kilpien läpi todella nopeasti ja sen on jätettävä pitkä partikkeliefekti peräänsä saaden sen näyttämään laserimaiselta. Sille täytyi tehdä myös paranneltu versio, jossa se pystyy kimpoilemaan seinistä.

Päivän aikana ensiksi tein kranaatinheittimen eri räjähdyksille omat partikkeliefektit, tein niistä lähinnä erivärisiä ja kokoisia saaden ne näyttämään paljon paremmalta, kun ruudulla on paljon räjähdyksiä. Tämän jälkeen muokkasin laser-ansan partikkeliefektiä niin, että siihen ei jää rakoja ja tein siitä vähän punaisemman, nyt se näyttää paljon enemmän laserilta. Viimeisenä aloin tekemään kokonaan uutta laser-asetta pelaajalle, jolla pystyy ampumaan useiden vihollisten läpi. Ase muistuttaa aika paljon kiikarikivääriä. Sain tehtyä aseen perusmekaniikat ja vihollisten läpimenemisen valmiiksi sekä sen mahdolliset kaupasta ostettavat parantelut, eli lisävahinkoa, nopeampi lataus ja parempi vihollisten lävistys, näitä parannuksia ei kuitenkaan kaupasta vielä löydy, vaan lisään lopuksi kaikki puuttuvat parannukset kerralla, kunhan saan kaikki aseet valmiiksi. Seuraavaksi minun täytyy tehdä laserille paranneltu asevaihtoehto, joka saa sen kimpoilemaan seinistä.

Torstai 21.11.

Päivän suunnitelmana oli tehdä kimpoileva luoti eilen tekemälleni laser-aseelle. Yritin toteuttaa luotia ensiksi Unity-pelimoottorin fysiikkamoottorilla.

Päivän aikana sain tehtyä kimpoilevaa luotia aika pitkälle. Ensimmäisenä minun täytyi muuttaa luoti lentämään Unity-pelimoottorin fysiikalla toisinkuin toiset luodit pelissä. Se lensi ennen Transform.Translate-metodilla, jota ajettiin Update-funktiossa. Tämä ei kuitenkaan toimi, jos luodin täytyy pystyä kimpoamaan, joten annoin luodille fysiikat ja AddForce-metodilla laitoin sen lentämään ampuma suuntaan. Tässä vaiheessa minun täytyi myös muokata luodin grafiikkaa ympyräksi ja estämään sen pyöriminen ja lisätä sille uusi ympyrän muotoinen CircleCollider-komponentti. Seuraa-

vaksi minun täytyi luoda koodi, joka saa sen kimpoilemaan seinistä. Ongelmana tämän tekemisessä oli, että luodin täytyi toimia fysiikalla toisinkuin ennen, joka sai sen käyttäytymään eri tavalla vihollisosumien kanssa. Ratkaisin tämän tekemällä luodille lapsiobjektin, joka huolehtii viholliskontakteista TriggerCollider-komponentilla ja omalla koodillaan, luodin omaa koodia muokkasin tutkimaan vain seinäosumia fysiikalla. Tällöin pystyin kimmottamaan Vector2.Reflect-metodilla (Unity Dokumentaatio 2019b. Vector2.Reflect) luodin uuteen suuntaan seinäosumissa. Vector2.Reflect-metodi tarvitsee vain kaksi muuttujaa, jotka ovat kimmotettavan objektin meno suunta, inDirection, sekä objektin pinnan suunta 90 asteen kulmassa eli inNormal, johon kyseinen kimmotettava objekti osuu. Sain loppujen lopuksi kimpoilevan luodin melkein täysin tehtyä, siihen jäi vain yksi ongelma, jota en saanut ratkaistua, joka oli sellainen, että kun luoti osuu nurkkaan niin koodi ei laske sen kimpoamiskulmaa oikein ja se lähtee liukumaan seinää pitkin johonkin suuntaan. Tämän aiheutti ilmeisesti se, että luoti osuu samalla kuvataajuuden päivityksellä kahteen eri seinään, jolloin se menee sekaisin siitä, että mihin sen pitäisi kimmota. Yritin googlata asiaa, ja löysin monelta saman ongelman, mutta en ratkaisua. Muuten luoti kuitenkin tuntuu toimivan oikein hyvin seinien ja vihollisten kanssa.

Perjantai 22.11.

Päivän tarkoituksena oli korjata laser-aseen kimpoaminen nurkista ja lisätä uudet aseet kauppaan päivityksineen ostettavaksi. Laserin kimpoilu nurkista oli todella haastavaa korjata, sillä en ole joutunut korjaamaan tämän tapaisia ohjelmointivirheitä kovin paljoa aikaisemmin.

Yritin aika kauan aikaa korjata kulmakimpoilua laserista monin eri konstein, mutta mikään ei auttanut. Yritin esimerkiksi tutkia, että osuuko se moneen seinään yhtä aikaa, ja jos osuu, niin kimpoa suoraan takaisin. Yritin myös eri aika asetuksia, että laserin täytyy odottaa esimerkiksi 0.1 sekunnin ennen uutta kimpoamista sekä yritin FixedUpdate-funktiolla korjata sen velocity-arvoa, mutta mikään ei auttanut, joten luovutin ainakin toistaiseksi ongelman suhteen ja aloin lisäämään uusia aseita kauppaan päivityksineen. Sain lisättyä uudet aseet kauppaan ja niiden päivitysten ostaminen toimi hyvin. Minun pitää kuitenkin vielä seuraavaksi muuttaa kauppaa niin, että kaikki aseet paitsi aloitus ase täytyy ostaa kaupasta, ennen kuin niitä voi käyttää. Tällä hetkellä pelaajalla on heti kaikki aseet ja voi ostaa kaupasta niihin päivityksiä.

Viikkoanalyysi

Viikon aikana sain lisättyä kameran tärisytys efektin ChargeJump-kykyyn, laser-ansalle uuden partikkeliefektin, kaksi uutta asetta, kranaatinheitin ja sen rypälepommi muoto ja niihin uusia partikkeliefektejä, sekä laser-ase, vaikka siihen jäikin vielä ohjelmointivirhe ja sain lisättyä ne päivityksineen kauppaan.

Viikon isoin ongelma oli laser-aseen parannellun muodon kimpoilu, jota en saanut korjattua. Luoti ei kimpoa nurkista ollenkaan, vaan lähtee liukumaan jompaakumpaa seinää pitkin hitaasti, kunnes se jää jumiin seuraavaan nurkkaan.

Viikon aikana opin kunnolla, että miten asesysteemimme toimii, kun tein kaksi uutta asetta ja niiden päivitykset ja lisäsin ne kauppaan. Opin myös lisää partikkeliefekteistä tehdessäni niitä laser-ansalle ja uusille aseille. Opin myös todella paljon Unity-pelimoottorin fysiikkamoottorista ja siitä, että kuinka huono se oikeastaan on.

Viikon aikana koen kehittyneeni ohjelmoijana, vaikka jotkin ongelmat vaikuttivatkin nyt ylivoimaisen vaikeilta, sain kuitenkin sopivan haastavia tehtäviä tehtyä monta. Kehityin myös Unity-pelimoottorin fysiikkamoottorin käytössä.

3.7 Seurantaviikko 6

Maanantai 25.11.

Päivän suunnitelmana oli muokata kauppaa niin, että sieltä pitää ostaa uudet aseet ennen kuin niitä voi käyttää sekä lisätä kranaatteihin kameran tärisytys efekti ja korjata ohjelmointivirhe, jossa kranaatit räjähtävät osuessaan automaattisten valvontatykkien valvonta alueelle.

Päivän aikana ensiksi lisäsin kameran tärisytys efektin kranaatteihin, joka teki niistä kyllä paljon paremmat. Sen jälkeen korjasin ohjelmointivirheen kranaattien törmäyksessä tykkien alueelle, tämä korjaantui helposti muokkaamalla vain tykkien valvonta alueiden tag-merkkiä. Kranaatit nimittäin tunnistivat näkymättömän alueen niiden tag-merkin perusteella viholliseksi ja räjähtivät. Tämän

jälkeen aloin muokkaamaan kauppaa niin, että sieltä täytyy ostaa uudet aseet ennen kuin niitä voi käyttää. Sain sen tehtyä melkein kokonaan päivän aikana, mutta siihen jäi joitain ohjelmointivirheitä huomiselle ratkottavaksi. Joskus kun aseen ostaa kaupasta, niin pelaaja voi ampua kahdella aseella yhtä aikaa ja toinen ohjelmointivirhe on sellainen että, aseiden ikonit näkyvät ruudun yläkulmassa väärin. Tämä johtuu siitä, että se systeemi on tehty niin, että aseiden on tarkoitus olla tietyssä järjestyksessä, mutta jos pelaaja ostaa ne eri järjestyksessä niin ikonit menevät väärin, tämä täytyy korjata tekemällä systeemi vain erillä lailla. Joko niin että aseet menevät automaattisesti järjestykseen tai sitten ikonit eivät katso aseiden järjestystä vaan jotain muuta.

Tiistai 26.11.

Päivän tarkoituksena oli korjata nyt kaupassa ja aseiden vaihtelussa olevat ohjelmointivirheet. Aseiden vaihtosysteemi vaatii todennäköisesti isoja muutoksia siihen mekaniikkaan.

Päivän aikana sain kaupan toimimaan hyvin. Minun täytyi tehdä aseen vaihtosysteemi kokonaan uusiksi ja tehdä kauppa toimimaan sen kanssa hyvin. Uudessa systeemissä kaikilla aseilla on oma ase numero koodissaan, ja ne ovat kaikki järjestyksessä Editor-näkymässä. Niillä on myös boolmuuttuja, että omistaako pelaaja kyseisen aseen. Sitten kun pelaaja vaihtelee aseita, niin koodi katsoo kyseisen aseen index-luvusta, että onko pelaajalla asetta, jos ei ole niin systeemi katsoo seuraavaa asetta jne. (kuvio 12.). Aseita vaihdellaan hiiren rullalla ja numeronäppäimillä.

```
if (control.GetButtonDown("NextWeapon"))
{
    weapons[currentWeapon.Value].isActive = false;
    currentWeapon.Increment();
    if (currentWeapon.Value >= weapons.Length)
    {
        currentWeapon.SetValue(0);
    }

    while (!weapons[currentWeapon.Value].hasGun)
    {
        currentWeapon.Increment();
        if (currentWeapon.Value >= weapons.Length)
        {
            currentWeapon.SetValue(0);
        }
    }
```

KUVIO 12. Kuva koodista, kun asetta vaihdetaan seuraavaan hiiren rullalla.

Keskiviikko 27.11.

Päivän tarkoituksena oli tehdä uusi panossysteemi. Tämä vaati sitä, että tein uudet panokset eri aseille, maksimi panos määrän kaikille panoksille, HUD-ominaisuuden muutoksia, että panokset näkyvät x määrä panoksia / x määrä maksimi panoksia muodossa, viholliset pudottavat kaikkia panoksia ja panokset tallentuvat tallennuspisteillä ja tasojen lopussa.

Päivän aikana sain tehtyä uuden panossysteemin kokonaan ja huomasin ohjelmointivirheen laser-aseen laserin kanssa, se nimittäin pystyy joskus menemään seinien ja vihollisten läpi tekemättä mitään, se kulkee tällä hetkelle ihan Unity-pelimoottorin omalla fysiikalla aika kovaa vauhtia niin se taitaa vain mennä osuma kuvataajuuden päivityksellä yli kohteen Collider-komponentista. Minun täytyy huomenna selvittää mistä se johtuu ja korjata se. Ensiksi päivän aikana tein jokaiselle aseelle omat panoksensa ja sitten tein niille maksimimäärän mitä pelaaja voi kantaa. Panoksia ja niiden maksimimäärää säädellään Scriptable object-objektilla. Tämän jälkeen tein HUD-ominaisuuden muutoksia, niin että siinä näkyy kyseisen kädessä olevan aseen panokset muodossa x määrä panoksia / x määrä maksimi panoksia. Tämän jälkeen lisäsin panokset vihollisten pudotus koodiin ja

jouduin tekemään siihen vähän muutoksia, koska nyt pudotettavia panostyyppejä on kolme eikä yksi. Sen jälkeen tein koodin, joka aina pelin alkaessa tarkistaa tallennetut panokset ja jos niitä on olemassa, lisää ne pelaajalle, jonka jälkeen tein panosten tallentamisen tallennuspisteisiin ja kenttien loppuun.

Torstai 28.11.

Päivän suunnitelmana oli lisätä uusien aseiden ostoon kaupassa ominaisuus, että kun ostaa uuden aseen niin siihen on valmiiksi täydet panokset ja lisäksi sellainen ominaisuus, että viholliset eivät pysty pudottamaan panoksia aseille, jota pelaajalla ei ole. Tämän jälkeen tarkoituksena oli korjata laser-aseen luoti, sillä se menee joskus seinien läpi ja kimpoaa väärin nurkista.

Päivän aikana sain lisättyä kauppaan ominaisuuden, että pelaaja saa täydet panokset ostamaansa aseeseen sekä että viholliset eivät pudota panoksia aseille, jota pelaajalla ei ole. Tämän jälkeen aloin korjaamaan laser-aseen toimivuutta. Yritin korjata sitä muokkaamalla sen fysiikka asetuksia ja kaikkea siihen liittyvää, mutta ainoa, joka auttoi ongelmaan, että luodit menivät seinien läpi, oli se, että luodista teki hitaamman, ja siinä silti oli se ongelma, että se kimpoaa nurkista väärin. Aloin tutkimaan ongelmaa netistä ja löysin tämän artikkelin (Unity answers 2017. Collision in fast speed). Lukemalla tämän opin, että Unity-pelimoottorin fysiikkamoottori ei pysty kunnolla hoitamaan nopeasti liikkuvia objekteja RigidBody-komponentilla. Vaihtoehtoisina ratkaisuina olisi liikuttaa luotia koodissa ja tarkistaa osumia Raycast-ominaisuudella tai tehdä ns. Pyyhkäisytesti, jossa objekti liikkuu fysiikalla, mutta koodissa tarkistetaan jokaisella kuvataajuuden päivityksellä Raycast-ominaisuudella taaksepäin, että onko tullut osumia edellisen ja nykyisen kuvataajuuden päivityksen välillä. Käytin tätä pyyhkäisytestiä jo tehdessäni ChargeJump-kykyä, kun se oli aluksi paljon nopeampi, nykyään kyseinen koodi on turha, koska se kyky ei enää ole läheskään niin nopea ja toimii todella hyvin. Kuitenkin, koska luodeissa on lisäksi ongelma nurkkien kanssa, niin päätin alkaa tekemään luodin uusiksi Raycast-ominaisuudella. Ensiksi katsoin tutoriaali videon asiasta (Brackeys 2017. Shooting with Raycasts - Unity Tutorial). Videosta sain hyvän pohjatiedon, että miten lähden huomenna toteuttamaan tätä.

Perjantai 29.11.

Päivän tarkoituksena oli nyt korjata se laser-ammus. Yritin toteuttaa ominaisuuden RayCast-ominaisuutta hyväksikäyttäen, sillä sen pitäisi soveltua hyvin tällaiseen.

Normaalin laserin tekeminen oli suhteellisen helppoa, koska se ei mene vihollisten läpi. Kimpoileva laser oli paljon vaikeampi tehdä ja olin kauan jumissa kimmotus suunnan kanssa. Normaali laseriin verrattuna siihen täytyi lisätä vihollisten läpi meneminen ja kimpoilu. Vihollisten läpi meneminen oli suhteellisen helppo tehdä. Laser tarkistaa koko ajan koodissa oman vauhtinsa verran eteenpäin Raycast-ominaisuudella, että tuleeko osumia, jos se osuu viholliseen, se tekee siihen vahinkoa, tarkistaa voiko vihollisesta mennä läpi ja sitten lisää sen ”Uniikit osumat” listalle, että se tekee vahinkoa yhteen viholliseen vain kerran, ja sitten se jatkaa menemistä vihollisen läpi normaalisti, jos sen läpi pystyy menemään. Huomatessaan seinäosuman koodi muuttaa suunnan Vector2.Reflect metodilla. Tässä kohtaa minulle kuitenkin tuli ongelmia ja olin kauan jumissa sen kanssa. Laser kimpoili oikein vain, jos sen ampui alun perin oikealle, en tajunnut ongelmaa aluksi yhtään. Kuitenkin pitkään selviteltyäni huomasin, että luodin rotaatio vaikuttaa sen suuntaan, jossa ei ollut mielestäni mitään järkeä, koska sitä liikutetaan vain tiettyyn suuntaan riippumatta sen rotaatiosta, kuitenkin sitten keksin, että vika johtuu siitä, että miten luotia liikutetaan (kuvio 13.). Ja tein sen uudella tavalla.

```
else
{
    //transform.Translate(_speed * direction * Time.deltaTime);
    transform.position += direction * _speed * Time.deltaTime;
}
```

KUVIO 13. Kuvassa vanha ja uusi tapa liikuttaa objektia.

Viikkoanalyysi

Viikon aikana sain tehtyä paljon uudistuksia pelin kauppaan lisäämällä uudet aseet sinne sekä uutta eloa pelin asesysteemiin lisäämällä omat luodit jokaiselle aseelle, joka rohkaisee pelaajaa käyttämään ja parantelemaan kaikkia aseita yhden sijaan ja tekee pelistä paljon monipuolisemman. Sain myös tehtyä laser-aseen uusiksi ja nyt se toimii juuri niin kuin sen pitää. Laser-aseen luodit korjaantuivat käyttämällä niissä Raycast-ominaisuutta törmäyksien tutkimiseen. Raycast-ominaisuus osoittautui erittäin tarkaksi ja hyväksi tähän hommaan. Visuaalisesti parantelin peliä myös lisää kameran tärisytys efektillä sekä korjailin lukuisia ohjelmointivirheitä.

Viikon aikana opin todella paljon Unity-pelimoottorin fysiikkamoottorista, sen eri asetuksista ja siitä, että mihin se pystyy ja mihin se ei pysty, esimerkiksi liian nopeat objektit tuottavat fysiikkamoottorille vaikeuksia, jolloin parempi ratkaisu on manuaalisesti liikuttaa objektia, ja ennustaa RayCast-ominaisuudella x määrä kuvataajuuden päivityksiä eteenpäin ja etsiä siten osumia. Jatkossa osaan jo suunnitteluvaiheessa päättää, että mikä on järkevin tapa liikuttaa pelin objekteja ja tarvitsevatko ne oman optimoidun koodin itsensä liikuttamiseen vai riittääkö Unity-pelimoottorin fysiikat.

Viikon aikana koen kehittyneeni todella paljon ohjelmoijana, koska sain ratkaistua minulle erityisen vaikeita ongelmia. Kehityin myös Unity-pelimoottorin fysiikkamoottorin käytössä.

3.8 Seurantaviikko 7

Maanantai 2.12.

Päivän tarkoituksena oli suunnitella uusi muoto haulikolle, että pystyin tekemään sen seuraavana päivänä sekä lisätä liukuhihnaan grafiikat ja korjata erään vihollisen liikkuminen ympäristössään, se nimittäin ei onnistunut liikkumaan muualla kovin hyvin kuin tasaisella maalla.

Päivän aikana suunnittelimme haulikkoa ensiksi. Ideoita oli muutamia kuten haulikon panokset hajoaisivat pienemmiksi ja useammiksi lentomatkallaan, panokset kimpoisivat tai räjähtäisivät vihollisista tai panokset vaikka korvattaisiin pienillä hakeutuvilla ohjuksilla tai panokset olisivat sähköisiä ja aiheuttaisivat muutaman sekunnin kestävän sähkötys efektin, joka vahingoittaa vihollista. Päädyimme sähkövaihtoehtoon ja pelin pääsuunnittelija suunnittelee minulle sen kunnolla huomiseksi. Tämän jälkeen aloin lisäämään liukuhihnaan grafiikat, tästä tulikin sitten paljon suurempi projekti kuin alun perin luulin, sillä grafiikat oli jaettu kolmeen osaan, keskipalanen, vasen palanen ja oikea palanen. Päätin helpottaa tasorakentajamme työtä ja yrittää tehdä ensimmäistä kertaa jonkinlaisen Editor-näkymässä toimivan koodin, eli siis tarkoituksena oli luoda koodi, jolla voi Editor-näkymässä määrittää, että kuinka pitkä liukuhihna on, ja sitten koodi lisää keskipalikoita liukuhihnaan ja siirtää sen mukaan sivupalikoita sivummaksi. Löysin tämän Unity-pelimoottorin dokumentaatiosta, jolla toteutin koodin (Unity Dokumentaatio 2019c. ExecuteInEditMode). Tämä oli siis ensimmäinen

kerta, kun tein Editor-näkymässä toimivan koodin ja opin siitä paljon, ensinnäkään Editor-näkymässä koodi ei pysty käyttämään ollenkaan Start tai Awake-metodeja, ainoastaan Update-funktiota ja itse tekemiäni metodeja. Tein siis niin, että tarkistan Update-funktiossa, että vaihtuuko liukuhihnan pituus, jos vaihtuu, niin ajetaan itse tekemäni metodi, joka käytännössä hoitaa muutoksen. Se käytännössä ensiksi resetoi liukuhihnan pienimpään kokoonsa, jonka jälkeen lisää vuorotellen oikealle ja vasemmalle palikoita, kunnes se on oikean kokoinen. Sain kyseisen koodin toimimaan mielestäni yllättävän hyvin ja opin kyllä paljon siitä, että kuinka voin tehdä tasorakentajien työstä helpompaa tekemällä heille tällaisia työkaluja Editor-näkymään.

Tiistai 3.12.

Päivän tarkoituksena oli tehdä haulikolle nyt uusi muoto, jossa se ampuu DNA-kuviossa leijuvia sähköpanoksia kahdessa syklissä, jotka jäävät sähköttämään vihollista vähäksi aikaa aiheuttaen niihin lisävahinkoa osuessaan.

Päivän aikana sain melkein tehtyä haulikon uuden muodon valmiiksi. Isoin ongelma oli saada panokset leijumaan symmetrisesti DNA-muodossa ampumasuuntaa kohti, ratkaisin tämän muokkaamalla niiden rotaatiota x astetta jokaisella lukitulla kuvataajuuden päivityksellä, ja tietyn ajan jälkeen ne vaihtavat suuntaa. Tämä ratkaisu toimi hyvin ja sain ne näyttämään hyvältä. Seuraavaksi tein niille sähköltä näyttävän partikkeliefektin, jonka ne jättävät jälkeensä lentäessään. Vielä pitäisi kuitenkin tehdä vihollisille sähkötysefekti, mutta en ole vielä varma, kuinka sen toteuttaisin, joten jätän sen myöhemmälle. Tämän jälkeen tein jatkuvan vahingon aiheuttamisen. Tein sen muokkaamalla meidän IDamageble-rajapintaamme niin, että lisäsin siihen vaihtoehdon jatkuvalle vahingon ottamiselle, seuraavaksi minun täytyy toteuttaa itse vahingon ottaminen yksitellen jokaisen vihollisen koodissa, mutta se jää huomiselle.

Keskiviikko 4.12.

Päivän suunnitelmana oli tehdä haulikon uusi muoto loppuun ja saada se toimimaan jokaisella vihollisella. Tämän jälkeen tarkoituksena oli korjata kilpivihollisen ympäristössä liikkuminen, kyseisellä hetkellä se jäi jumiin vähän kaikkialle ja ei toiminut kunnolla.

Päivän aikana sain ensiksi tehtyä haulikon uuden muodon loppuun muutamaa graafista ja audio asiaa lukuun ottamatta. Minun täytyi muokata pelin jokaisen vihollisen vahingon ottamiseen tarkoitettua koodia siten, että ne pystyvät ottamaan vastaan jatkuvan vahinko efektin. Varsinkin loppuvastuksille piti tehdä vähän isompia muutoksia koodiin, jotta haulikko toimi niiden kanssa oikein. Tämän jälkeen aloin korjaamaan kilpivihollisen ympäristössään liikkumista, tein oikeastaan sen koko maan ja seinien tunnistus koodin uusiksi ja lisäsin sille keinotekoisen painovoiman, joka ei siis käytä Unity-pelimoottorin fysiikkaa, vaan liikutan sitä koodissa vain alaspäin, jos se ei tunnista maata jalkojensa alla. Sain vihollisen lopulta toimimaan hyvin ja se ei enää jää jumiin mihinkään. Tehdessäni sitä sain myös idean, että kilpivihollisen aseena pitäisi olla haulikko ja kilven pitäisi olla hajotettava. Nykyään se käyttää kilpeä lyömäaseena eikä mitään muuta. Juttelin tästä pääsuunnittelijamme kanssa ja sovittiin, että teen nämä muutokset viholliseen seuraavaksi, sillä ne tekisivät siitä paljon vaarallisemman pelaajalle ja mielenkiintoisemman. Pääpointtina tässä kai on, että kilpivihollinen on yksi ensimmäisiä pelin vihollisia mitä on tehty, ja se on nyt todella vanhentunut ja ei toimi enää kunnolla niin kokonaisuutena korjaan sen toimimaan oikein ja lisätään sille uusia ja mielenkiintoisempia mekaniikkoja.

Torstai 5.12.

Päivän suunnitelmana oli tehdä kilpiviholliselle rikkoutuva kilpi ja haulikko. Lisäksi minun täytyi etsiä siitä mahdollisia ohjelmointivirheitä, että se toimisi paremmin.

Päivän aikana sain lisättyä kilpiviholliselle uuden kilven väliaikaisilla rikkoutuvilla grafiikoilla.
Minun täytyi muokata projektin fysiikka asetuksia vähän, että sain sen toimimaan. Lisäksi täytyi muokata kaikkia aseita vähän, että ne toimivat kunnolla kilven kanssa. Tämän jälkeen huomasin joitain vikoja vihollisen liikkumisessa ympäristössään, jos se näki pelaajan, minun täytyi muokata sen ohjelmoitua tekoälyä niin, että se toimii eri lailla eri tilanteissa riippuen, näkeekö se pelaajan vai ei. Minun täytyy vielä muokata sitä lisää, kun lisään haulikon sille, ja lisäksi sen täytyy lyödä kilvellään, kun pelaaja tulee liian lähelle, ja ampua haulikolla, jos pelaaja on kauempana, ja jos se näkee pelaajan, mutta ei yllä ampumaan, niin kävellä lähemmäs, muussa tapauksessa vaellella satunnaisesti.

Perjantai 6.12.

Itsenäisyyspäivä

Viikkoanalyysi

Viikon aikana lisäsin liukuhihnalle Editor-näkymässä toimivan koodin, jolla tasorakentaja voi helposti muokata liukuhihnan pituutta. Tein myös haulikolle uuden muodon, joka ampuu sähköpanoksia DNA-kuviossa leijuen ja sille uudet partikkeliefektit. Sitten tein vielä kilpiviholliselle uuden rikkoutuvan kilven ja lukuisia ohjelmointivirheiden korjauksia.

Viikon aikana opin paljon Editor-näkymässä toimivasta koodista ja jos olisin osannut tehdä niitä aiemmin, niin olisin voinut tehdä monille muillekin pelin objekteille koodin, joka auttaisi tasorakentajaa. Jatkossa kyllä teen sen kaikelle mitä teen, jos sille vain on tarvetta.

Viikon aikana koen kehittyneeni ohjelmoijana, sillä viikon tehtävät olivat taas sopivan haastavia ja tein ensimmäisen Unity-pelimoottorin Editor-näkymässä toimivan koodini. Kehityin myös tavallaan projektinhallinnassa, koska keksin, että voin helpottaa muiden työtä tekemällä ominaisuuksista sellaisia, että niitä on mahdollisimman helppo käsitellä Editor-näkymässä.

3.9 Seurantaviikko 8

Maanantai 9.12.

Päivän suunnitelmana oli lisätä pelin kilpiviholliselle haulikko ase ja sen lisäksi lisätä kaikkiin pelaajan aseiden panoksiin RayCast-ominaisuudella tunnistus, jonka avulla ne eivät menisi seinien läpi niin useasti, huomasin tarpeen tälle testatessani kilpivihollisen kilpeä, koska joskus luodit menivät siitä suoraan läpi.

Päivän aikana lisäsin kaikkiin pelaajan aseisiin ensiksi RayCast-ominaisuudella tunnistuksen, jonka avulla ne toimivat paljon paremmin ja eivät enää mene läpi objekteista. Tämän jälkeen lisäsin kilpiviholliselle oman haulikon, jolla se ampuu tietyltä etäisyydeltä. Tällä hetkellä se siis yrittää ampua pelaajaa haulikolla, jos pelaaja on tarpeeksi lähellä ja jos pelaaja tulee liian lähelle niin se lyö pelaajan kauemmas kilvellään ja sitten ampuu taas haulikolla.

Tiistai 10.12.

Tämän päivän suunnitelmana oli tehdä kilpiviholliseen sellainen muutos pääsuunnittelijan pyynnöstä, että se käyttäytyy vähän eri lailla, kun sillä on ehjä kilpi, ja kun sillä ei ole kilpeä. Tarkoituksena oli, että se ehjällä kilvellä kävelee normaalisti pelaajaa kohti ja yrittää lyödä kilvellään ja vasta kilven rikkouduttua se vaihtaa haulikkoon ja ei enää yritä lyödä. Toisena asiana tänään oli lisätä tekemiini liukuhihnoihin oikeat grafiikat ja animaatiot.

Päivän aikana sain tehtyä kilpiviholliselle uuden käyttäytymisen mallin kilven kanssa ja ilman, toteutin sen vain yhdellä bool-muuttujalla, joka katsoo, onko kilpi rikki ja sitten tätä bool-muuttujaa hyödyksi käyttäen järjestelin uusiksi vihollisen käyttäytymisen koodin. Tämän jälkeen lisäsin liukuhihnoihin oikeat grafiikat ja animaatiot. Siinä tuli esille sellainen ongelma, että en millään saanut yhdellä animaattorilla liukuhihnan isäntäobjektissa hoidettua animaatioita, koska kun animaatiota luo, niin kaikki sen liikuttamat asiat pitäisi jo olla olemassa niin vaihtoehtona oli luoda lennosta animaatiot niille koodissa, kun liukuhihna luodaan Editor-näkymässä toimivalla koodillani, tai lisätä jokaiselle osalle oma animaattori, johon päädyin vaikka se on selvästi huonompi vaihtoehto, mutta oli kiire ja se oli paljon helpompi toteuttaa, minun täytyi myös tehdä animaattori ja koodi siten, että liukuhihnat pystyvät rullaamaan molempiin suuntiin. Loppujen lopuksi liukuhihnojen animaatiot kuitenkin toimivat ihan hyvin. Testaillessani liukuhihnojen animaatioita huomasin, että pelaajan EMP (Electro Magnetic Pulse) -kyky, jonka pitäisi pysäyttää kaikki sähköllä liikkuvat asiat pelissä (paitsi pelaajahahmo itse) ei toiminut kuitenkaan liukuhihnoihin, joten lisäsin liukuhihnaan vielä sen pysäyttämiseen vaadittavan koodin.

Keskiviikko 11.12.

Tämän päivän tarkoituksena oli lisätä peliin uusi vihollinen, joka on käytännössä uusi ja voimakkaampi versio pelin perusvihollisesta, joka yksinkertaisesti vain kävelee ympäriinsä ja ampuu pelaajaa. Tämän uuden vihollisen on tarkoitus ampua todella aggressiivisesti sarjoissa useita luoteja kerralla, ja se oli tähän mennessä vaarallisin vastus pelaajalle.

Päivän aikana sain tehtyä tämän uuden vihollisen valmiiksi. Pääosin sen pystyi suoraan kopioimaan vanhasta heikommasta viholliseSta ja muuttelemaan vaan muutamia numeroita kuten sen

nopeutta ja kestävyyttä. Ainoa uusi mekaniikka sillä oli sen uusi sarjatuli ase, joka olikin ihan mielenkiintoinen tehdä (kuvio 14.). Tehdessäni tätä uutta vihollista huomasin taas monia vikoja vihollisten liikkumisessa, ne eivät taas toimineet ihan oikein rinteissä, sillä nykyisessä koodissa sillä on aika suuri merkitys, että minkä kokoinen vihollinen on, lisäksi oli ongelmia tietyissä tilanteissa, esimerkiksi, jos vihollinen näkee pelaajan, mutta heidän välissään on rotko. Korjailin loppupäivän sitten kaikkia näitä eri tilanteita ja sain ne suurimmaksi osaksi korjattua.

```
private IEnumerator ShootingBurst()
{
    if (IsDead) yield return null;

    isWalking = false;
    MyAnimator.SetTrigger("Shoot");
    //How many times to shoot per burst, /2 because shoots 2 bullets at a time.
    float sequences = shotsPerBurst / 2;
    //for cooldown
    _timeToShoot = Time.time + 1 / burstsPerSecond;
    //For loop for going through the shooting sequences, shoots 2 bullets per sequence.
    for (int i = 0; i < sequences; i++)
    {
        //Calculate angle for shooting direction
        float angle = transform.CalculateAngle(_player.transform);
        float angle2 = angle;
        //randomize angle a bit
        float randomAngle = Random.Range(-5, 5);
        angle += randomAngle;
        float randomAngle2 = Random.Range(-5, 5);
        angle2 += randomAngle2;

        //Instantiate a bullet from both guns
        Instantiate(bulletPrefab, _weaponHolder.transform.position, Quaternion.Euler(new Vector3(0, 0, angle)));
        Instantiate(bulletPrefab, _weaponHolder2.transform.position, Quaternion.Euler(new Vector3(0, 0, angle2)));
        //If shooting sound is not null, play it
        if (shootSound != null)
        {
            multiChannelAudioSource.PlayOneShotAtSequence(shootSound);
        }
        //Wait for shooting speed
        yield return new WaitForSeconds(1 / ShotsPerSecond);
    }
}
```

KUVIO 14. Kuvassa koodi, jolla vihollinen ampuu sarjoissa.

Torstai 12.12.

Tämän päivän tarkoituksena oli tehdä paljon ohjelmointivirheiden korjauksia sekä testata ja tasapainottaa peliä. Pelin tasapainottaminen on tärkeää, koska silloin pelikokemus on pelaajalle parempi.

Päivän aikana korjailin useita pieniä ohjelmointivirheitä ja testasimme peliä paljon, päädyimme myös muokkaamaan pelin aseiden eri arvoja ja lähinnä tekemään niistä heikompia ja vähentämään pelissä olevien lisä ammusten määrää. Tarkoituksena oli siis vähän vaikeuttaa peliä.

Perjantai 13.12.

Päivän tarkoituksena oli saada uusi toimiva demoversio pelistä valmiiksi illan demoversion testausta varten. Päivän hommana oli siis paljon testausta ja ohjelmointivirheiden korjailua.

Päivän aikana testattiin peliä paljon ja korjailtiin kaikkia pieniä ohjelmointivirheitä mitä ilmeni. Yksi iso ohjelmointivirhe ilmestyi jostain ja kilpivihollinen meni aina aivan sekaisin nähdessään pelaajan rotkon takana, ja korjasin sen, kyseessä oli vain looginen virhe sen koodissa. Korjasin myös päivän aikana monen vihollisen animaatioita, sillä useissa niiden Sprite-grafiikoissa oli Pivot-pisteet väärässä kohtaa. Lopuksi saimme kuudennella yrityksellä tehtyä pelistä demoversion, johon olimme tyytyväisiä, tämän jälkeen lähdimme pelitestaustapahtumaan. Pelitestaustapahtuma oli huomattavasti lyhyempi kuin kaikki edelliset, missä olemme käyneet, emmekä saaneet testattua paljon mitään.

Viikkoanalyysi

Viikon aikana lisäsin kilpivihollliselle haulikon ja parantelin sen ohjelmoitua tekoälyä paljon. Lisäsin myös oikeat grafiikat ja animaatiot liukuhihnoihin ja tein uuden erittäin voimakkaan vihollisen, sitten korjasin useilta vihollisilta niiden animaation, että ne näyttivät mahdollisimman hyvältä ja sitten oli vielä paljon pelitestausta, ohjelmointivirheiden korjailua ja pelin balansointia. Lopuksi oli pelitestaustapahtuma.

Viikon aikana opin paljon pelimme ohjelmoidusta tekoälyistä ja animaatioista. Viikon aikana oli todella kova kiire ja tein taas varsinkin loppuviikosta paljon ylityötunteja ja stressasin paljon, että saanko pelin vaadittuun kuntoon perjantaiksi. Lopuksi sain sen kuitenkin tarpeeksi hyväksi ja sitten pelitestaus kuitenkin olikin todella lyhkäinen ja turha. Yleensä pelitestaus on ollut monituntinen tapahtuma, missä meillä on paljon eri testaajia, joiden kanssa keskustelemme pelistä ja seuraamme heidän pelaamistaan ja katsomme varsinkin että, missä kohtaa heillä on hauskaa ja missä ei, mitkä kohdat tai asiat ovat liian vaikeita ja mitkä liian helppoja sekä UI-peliobjektin ymmärrettävyyttä ja helppokäyttöisyyttä ja vastaavaa. Lopuksi he yleensä vielä täyttävät meidän antamamme lomakkeen, jossa kyselemme kaikkia näitä asioita pelistä. Kyselemme heiltä myös paljon uusia ehdotuksia ja lisäyksiä peliin, monet uudet ja hyvät ominaisuudet ja muutokset ovatkin lähtöisin pelitestaajan suusta.

Viikon aikana koen kehittyneeni ohjelmoijana, varsinkin loppuviikosta oli kovat paineet ja kiire saada tehtyä kaikki tarvittava valmiiksi, mutta sain kuitenkin tehtyä kaiken. Ylityötuntien tekeminen ei kuitenkaan ollut kovin mukavaa ja väsymys vaikeutti oppimista.

3.10 Seurantaviikko 9

Maanantai 13.1.

Tämän päivän tarkoituksena oli suunnitella tiimin kanssa yhdessä tulevia sprinttejä seuraavilta kuukausilta. Jokainen lisäsi sprintteihin omat tehtävänsä ja etenemistahdin oman arvionsa mukaan. Suunnittelun jälkeen minulla oli tarkoituksena tehdä energiakilpi pelaajahahmolle. Olen tehnyt jo ennestään pelaajahahmolle energiakilven, joka näkyy HUD-ominaisuudessa sinisenä palkkina ja suojelee pelaajaa tietyltä määrältä vahinkoa ja kun pelaaja ei ota vahinkoa vähään aikaan, energiakilpi palautuu takaisin täyteen hiljalleen. Nyt kuitenkin täytyi tehdä sille placeholder-grafiikka, joka välähtää pelaajan ympärillä aina kun pelaaja ottaa vahinkoa ja hänellä on energiakilpeä jäljellä.

Päivän aikana pidimme ensiksi suunnittelupalaverin ja saimme suunniteltua projektin jatkon hyvin, että mitä kukakin tekee milläkin sprintillä. Tämän jälkeen tein energiakilven pelaajalle ja sain sen toimimaan oikein hyvin. Seuraavaksi täytyy vain odotella, että graafikko tekee sille oikeat grafiikat. Tein kilven sillä tavalla, että lisäsin pelaajahahmolle kilven, jossa on kaksi eriä sprite-grafiikkaa, ja kun pelaajahahmo ottaa vahinkoa, niin tarkistetaan, että onko kilpeä jäljellä, ja jos on, niin kilvestä vähennetään pisteitä ja kutsutaan FlashShield nimistä coroutine-metodia, joka nopeasti kasvattaa sprite-grafiikkojen alpha arvoa ja sitten vähentää sen takaisin nollaan (kuvio 15.). Tässä kuva energiakilven placeholder-grafiikasta, jonka tein (kuvio 16.).

```
for (int i = Mathf.RoundToInt(originalColor.a * 5); i < 5; i++)
{
    tempColor.a = i / 5f;
    tempColor2.a = i / 5f / 3f;

    energyShieldGraphic.color = tempColor;
    energyShieldGraphicInner.color = tempColor2;

    yield return new WaitForSeconds(0.005f);
}

yield return new WaitForSeconds(0.25f);

for(int i = 5; i > 1; i--)
{
    tempColor.a = i / 5f;
    tempColor2.a = i / 5f / 3f;

    energyShieldGraphic.color = tempColor;
    energyShieldGraphicInner.color = tempColor2;

    yield return new WaitForSeconds(0.005f);
}
```

KUVIO 15. Kuva FlashShield coroutine-rutiinista, jossa ensiksi katsotaan kilven alphan nykyinen arvo ja sitten siitä nostetaan se nopeasti täyteen ja sitten takaisin nollaan.

KUVIO 16. Kuva energiakilvestä pelissä. Kilpi on näkymätön silloin kun siihen ei osu mitään.

Tiistai 14.1.

Tänä päivänä minun täytyi ensiksi kuvata pelistä videomateriaalia sosiaalista mediaa varten. Tämän jälkeen testailin eri vihollisten ohjelmoitua tekoälyä kaikissa eri tilanteissa ja etsin virheitä, joita täytyy korjata. Tein tämän nyt koska pian minun täytyy tehdä muutamasta vihollisesta vielä paranneltu versio.

Päivän aikana kuvasin ensiksi videomateriaalin sosiaalista mediaa varten, jonka jälkeen aloin tutkimaan ja testailemaan eri vihollisten ohjelmoitua tekoälyä. Huomasin useita vikoja ja puutteita ja mietimme myös päivän aikana, että niiden käyttäytymisestä pitäisi tehdä ennalta arvaamattomampaa. Päätin siis, että lisään niiden ohjelmoituun tekoälyyn paljon satunnaisuutta siihen, että mitä ne tekevät, ne toimivat vain tietyllä tavalla tapellessa pelaajan kanssa. Kaikilla vihollisilla on myös tällä hetkellä oma uniikki koodi koko niiden ohjelmoituun tekoälyyn, mutta ne ovat erittäin samanlaisia. Päätin, että teen yhden ohjelmoidun tekoälykoodin, joka toimii kaikille vihollisille, siinä on vaan sitten paljon asetuksia, joilla voi customisoida kaikki viholliset käyttäytymään erillä tavalla. Vihollisille jäisi vain niille erittäin uniikit käyttäytymismallit ja mekaniikat uniikiksi omaan koodiin. Kaikki viholliset perisivät tämän pääkoodin. Tämä päivä menikin vain vihollisten ohjelmoidun tekoälyn testailuun ja tämän uuden systeemin suunnitteluun.

Keskiviikko 15.1.

Tämän päivän tarkoituksena oli aloittaa pelin vihollisten ohjelmoitujen tekoälyjen pääkoodin tekeminen sekä ensiksi tehdä itselleni vihollisten testaus skene-näkymä, jossa on kaikki mahdolliset tilanteet, joihin ne voivat joutua ja sitten voi laittaa aina vihollisia sinne ja katsoa yhtä aikaa niiden käyttäytymistä kaikissa eri tilanteissa ja nähdä että toimiiko ne.

Päivällä ensiksi tein tämän testaus skene-näkymän itselleni, jossa voin testata tiettyä vihollista kaikissa eri tilanteissa yhtä aikaa ja sainkin nopeasti testattua pelin perusvihollista ja löysin siitä paljon vikoja. Tämän jälkeen aloin miettimään ja tutkimaan sitä, että tekisin vihollisille yhteisen ohjelmoidun tekoälyn, mutta tutkiessani asiaa ja vihollisten omia uniikkeja koodeja, huomasin että ne olivat monet jo niin erilaisia, että olisi todella kova työ alkaa tekemään niille enää mitään yhteistä perittävää koodia, joten päätin alkaa kuitenkin korjaamaan jokaisen koodia yksitellen, ajattelin, että se olisi kuitenkin pienempi ja helpompi homma tässä vaiheessa. Olen kuitenkin vahvasti sitä mieltä,

että viholliset olisi täytynyt alun perin tehdä yhteisellä koodilla, josta olisi vain tehnyt mahdollisimman monipuolisen, että se soveltuu kaikille vihollisille tietyillä asetuksilla ja ainoastaan eri vihollisten uniikeimmat mekaniikat ovat niillä omassa koodissa, tämä nyt kuitenkin on tässä vaiheessa jo aivan liian suuri ja vaikea urakka, että siinä olisi mitään järkeä tehdä. Päivän aikana sain kuitenkin jo korjattua pelin perusviholliselta todella paljon ohjelmointivirheitä ja nyt se toimii aika moitteettomasti yksinään, vielä se tietysti käyttäytyy erilain nähdessään pelaajan ja siinä on varmasti vielä ohjelmointivirheitä, mutta tulen tekemään jatkossa kuitenkin kaikille kaksi eri käyttäytymismallia, ensimmäinen on se, kun ne eivät näe pelaajaa ja kuljeskelevat satunnaisesti vain ympäriinsä ja toinen on niiden ”taistelumoodi”, joka menee päälle kun ne näkevät pelaajan, jolloin ne jonkin aikaa automaattisesti yrittävät etsiä pelaajaa ja hyökkäävät. Eli siis ensiksi korjaan kaikkien vihollisten käyttäytymisen tekemälläni testi skene-näkymällä, jonka jälkeen teen nämä lisäykset niiden ohjelmoituun tekoälyyn.

Torstai 16.1.

Päivän tarkoituksena oli korjata ja testata pelin perusvihollisen ohjelmoitu tekoäly loppuun ja aloittaa sitten kilpivihollisen korjailu. Perusvihollisella oli vielä ongelmia nähdessään pelaajan ja kilpivihollisella oli paljon vikoja.

Päivän aikana testailin ensiksi perusvihollista ja löytyi muutamia ohjelmointivirheitä, jotka sitten korjasin. Tämän jälkeen vaihdoin testi radalle kilpiviholliset, jotka olivatkin sitten rikki useassa paikassa. Huomasin lisäksi ohjelmointivirheitä myös joissain pelin ansoista, jotka korjasin, esimerkiksi sähköansa yritti ikuisesti sähköttää vihollista, vaikka se olisi jo kuollut. Monet ansat eivät myöskään toimineet kilpivihollista vastaan ja korjasin ne ja säätelin kilpivihollisen Collider-komponenttia sekä monia sen RayCast-ominaisuuden asetuksia, joita se käyttää tunnistaessaan maata ja seiniä ja pelaajan, että se toimisi paremmin. Sainkin korjattua päivän aikana kilpivihollisen niin, että se toimii yksinään todella hyvin ja kaikki ansat toimivat siihen, mutta se ei vieläkään toimi monelta osin oikein nähdessään pelaajan ja siinä on vielä joitain ohjelmointivirheitä, jotka jäävät huomiselle korjattavaksi.

Perjantai 17.1.

Päivän tarkoituksena oli korjata kilpivihollisen käytös pelaajahahmon kanssa. Sen täytyisi reagoida, kun sitä ammutaan, ampua tai lyödä kilvellä tilanteen mukaan ja liikkua oikein maastossa sekä seurata pelaajaa oikein.

Päivän aikana tein ensiksi sille koodin, että se kääntyy pelaajaa kohti, kun sitä ammutaan, jos se katsoo pelaajasta poispäin. Tätä varten minun täytyi tehdä metodi, joka tarkistaa, että katsooko se pelaajaa kohti, sitten vain kutsun kyseistä metodia ja käännän hahmon, jos se katsoo poispäin pelaajasta. Tämän jälkeen aloin tutkimaan, että miksi vihollinen joskus lyö kilvellään, vaikka sen kilpi on rikki ja sen täytyisi ampua haulikolla. Tätä täytyi tutkia aika kauan, mutta lopulta löytyi, että jos pelaaja on yhtään alempana (esim. rinteessä) kuin vihollinen, niin se luulee, että heidän välissänsä on seinä, jolloin se kutsuu koodia, jossa se vain kävelee jne. mutta myös lyö kilvellään, jos on tarpeeksi lähellä. Korjasin tämän tekemällä sen erittä tavalla. Tämän jälkeen tutkin sen liikkumista maastossa ja korjasin siitä vielä viimeiset virheet ja nyt se tuntuu pystyvän kävelemään kaikissa maastoissa oikein ja seuraa pelaajaa, jos näkee tämän.

Viikkoanalyysi

Viikon aikana aluksi koko ryhmällä suunnittelimme tulevia sprinttejä ja suunnittelin omat tehtäväni ainakin muutaman kuukauden päähän aika tarkasti viikkotasolla. Tämän jälkeen sain tehtyä pelaajalle energiakilven ja kuvasin videomateriaalia sosiaalista mediaa varten. Tämän jälkeen aloitin loppuviikon ajaksi kovan vihollisten testaus urakan ja tein niille testiradan, jossa voin testata vihollisia monessa eri tilanteessa ja maastossa yhtä aikaa ja seurata, onko missään vikoja ja sitten korjailla vikoja yksitellen, kunnes kaikki toimii.

Viikon aikana huomasin, että vihollisissa lähes aina on jotain vikaa monissa eri tilanteissa. Tämä johtuu siitä, että en ole ikinä kunnolla testannut tekemiäni vihollisia kuin muutamassa tilanteessa, jolloin niihin on jäänyt useita piileviä ohjelmointivirheitä. Tästä lähtien testaan kaiken kunnolla mitä teen kaikki eri tilanteet huomioon ottaen.

Viikon aikana kehityin taas normaalin verran ohjelmoijana, mutta kehityin testaajana paljon enemmän, sillä keksin tehokkaan tavan testata nopeasti kaikki viholliset kaikissa eri tilanteissa. Pelin tehokas testaaminen on tärkeää, sillä se säästää aikaa muita tehtäviä varten.

3.11 Seurantaviikko 10

Maanantai 20.1.

Päivän tarkoituksena oli testata vasta tekemäni perusvihollisen voimakkaamman version ohjelmoitua tekoälyä ja korjata siinä olevia ohjelmointivirheitä ja lisätä puuttuvia asioita. Pystyin käyttämään tekemääni testirataa tätä varten.

Päivän alussa laitoin vihollisia testiradalle ja tutkin niitä ja löysin useita ohjelmointivirheitä. Ne mm. tipahtelivat rotkoihin ja sekoilivat rinteissä. Ne eivät osanneet kääntyä oikein, jos niitä ammuttiin ja niiden kävelyanimaatio jäi pyörimään, vaikka ne alkaisivat ampumaan. Päivän aikana sitten korjailin tämän kaiken. Aloitin korjaamalla niiden liikkumisen ympäristössään, vikana oli niiden kääntymisen koodi, siinä oli ohjelmointivirhe, joka monesti esti niitä kääntymästä. Tämän jälkeen tein niille uuden koodin, joka hoitaa täydellisesti niiden kääntymisen, kun pelaaja ampuu niitä ja lisäsin kyseisen koodin muillekin jo korjaamilleni vihollisille. Tulen kuitenkin vielä kaikki viholliset korjattuani lisäämään niille yhteisen koodin, jolla ne käyttäytyvät eri lailla taistelutilanteissa ja partioidessaan. Taistelutilanne käynnistyy, kun ne ottavat vahinkoa pelaajalta tai huomaavat pelaajan, jolloin ne automaattisesti kääntyvät koko ajan pelaajaa kohti, yrittävät kävellä lähemmäs ja ampuvat, jos ne kadottavat pelaajan, niin ne yrittävät etsiä sitä siitä suunnasta, missä näkivät pelaajan viimeksi ehkä noin viiden sekunnin ajan, jolloin ne palaavat partiointi käyttäytymiseen. Viimeiseksi korjasin vihollisen animaatiot, jolloin sillä jäi kävelyanimaatio muiden animaatioiden päälle pyörimään.

Tiistai 21.1.

Tämän päivän tarkoituksena oli testata ja korjata seuraavaksi pelin erästä vihollista, joka rynnistää pelaajaa kohti lyöden käsillään, kun se huomaa pelaajan. Toinen homma tänään oli tehdä mainitsemani taistelumoodi pelin tietyille vihollisille, joille se sopii.

Päivän aikana ensiksi tutkin rynnistävää vihollista ja löysin siitäkin vikoja tietyissä tilanteissa ja maastoissa liikkumisessa, jotka korjasin. Seuraavaksi aloin tekemään taistelumoodia vihollisille, sillä tällä hetkellä ne käyttäytyvät tyhmästi, että jos ne taistelevat pelaajan kanssa ja pelaaja vaikka hyppää vihollisen yli, niin vihollinen unohtaa pelaajan täysin ja jatkaa kävelyä eteenpäin. Suunnittelin toteuttavani taistelumoodin niin, että jos vihollinen ottaa vahinkoa tai näkee pelaajan, niin se

menee päälle pariksi sekunniksi ja tietysti ajastin alkaa koko ajan alusta, jos vihollinen vain näkee pelaajan tai ottaa vahinkoa. Taistelumoodin ollessa päällä vihollinen yksinkertaisesti koko ajan kääntyisi pelaajan suuntaan. Päivän aikana tein tämän taistelumoodin kaikille vihollisille, joille se sopii ja lisäsin niille kaikille myös omat kääntymisajat, joilla kaikki viholliset kääntyvät eri nopeutta niiden koosta riippuen. Korjailin myös animaatiossa esiintyviä ohjelmointivirheitä kääntymiseen liittyen, koska kävelyanimaatio oli päällä myös kääntyessä kaikilla vihollisilla. Nyt vihollisten kanssa taisteleminen on paljon haastavampaa, koska niiden ympärillä hyppiminen ei enää ole ylivoimainen taktiikka niitä vastaan. Seuraavaksi minun täytyy varmaankin alkaa tekemään satunnaista partiointia vihollisille, jossa ne kävelevät arvottuun suuntaan x määrän sekunteja, kunnes ne pysähtyvät ja arpovat uudelleen suunnan ja x määrän sekunteja kävellä siihen suuntaan. Tämä on varmasti hyvä homma huomiselle.

Keskiviikko 22.1.

Tämän päivän tarkoituksena oli tehdä partiointimoodi vihollisille. Partiointimoodi toimii käytännössä toimisi niin, että kun peli käynnistyy niin ne arpovat suunnan mihin mennä ja ajan, että kuinka kauan ne kävelevät siihen suuntaan, pysähtyessään ne arpovat, että milloin pitää lähteä uudelleen liikkeelle.

Päivän aikana sain tehtyä tämän partiointiominaisuuden yhdelle viholliselle, josta se on sitten helppo kopioida lopuille. En tehnyt sitä yhteisenä periytettävän koodina kaikille, koska kaikki on tosiaan tehty jo aivan erikseen ja niillä on osittain jopa aika erilaiset koodit, eli siis olisi täytynyt tehdä sillä tavalla jo alusta alkaen, mutta nyt vihollisten ohjelmoitu tekoäly on jo melkein valmis, niin samahan se on tehdä näin loppuun. Tein partiointikoodin juuri suunnitelmieni mukaan, mutta minun täytyi ottaa huomioon ja korjata jotenkin monet tilanteet, jotka voivat keskeyttää partioinnin, että ne toimisivat oikein, kuten seinään törmääminen, taisteluun joutuminen tai laidalle kävelemінen.

Torstai 23.1.

Tämän päivän tarkoituksena oli nyt lisätä partiointiominaisuus pelin lopuillekin vihollisille sekä kunnolla testata pelin vihollisia. Ongelmia kuitenkin tuotti se, että monet viholliset toimivat eri tavalla.

Päivän aikana lisäsin sen monelle muulle viholliselle, se ei kuitenkaan ollut ihan yksinkertaista, koska jotkin niistä toimivat ja käyttäytyvät todella eri tavalla kuin muut, joten sitä piti vähän muokata kyseisien vihollisten tarpeiden mukaan. Lisäksi lisäilin taistelumoodin vielä joillekin vihollisille ja jouduin sitä myös muokkaamaan niille sopivaksi, koska se ei suoraan sopinut niille. Sitten testailin vielä paljon kaikkia vihollisia tekemässäni testaus ympäristössä aika paljon ja nyt näyttää siltä, että ne toimivat hyvin. Tämän jälkeen muokkasin vähän pelaajan aseiden kantamaa ja lopuksi aloin tekemään valvontakameraa, joka selailee hitaasti vasemmalle ja oikealle vuorotellen, kunnes se näkee pelaajan ja tuijottaa suoraan kohti. Muuta se ei tee, sen tarkoitus on vaan luoda pelaajalle sellainen tunnelma, että häntä tarkkaillaan. Ehdin tehdä valvontakameralle ominaisuuden, että se kääntyilee oikealle ja vasemmalle, mutta suurimmaksi osaksi se jää huomiselle.

Perjantai 24.1.

Päivän tarkoituksena oli tehdä valvontakamera valmiiksi. Sen täytyisi siis kokonaisuudessaan kääntyillä vasemmalle ja oikealle johonkin tiettyyn asteeseen ja huomatessaan pelaajan sen täytyisi katsoa pelaajaa päin, kunnes pelaaja poistuu sen alueelta, jolloin se palaa katselemaan vasemmalle ja oikealle. Valvontakameran pitää myös pystyä hajottamaan ja sen täytyy pudottaa vähäsen rahaa pelaajalle.

Päivän aikana ensimmäiseksi tein uusiksi paremmalla tavalla valvontakameran peruskääntyilyn. Olin tehnyt sen vain Update-funktioon muokkaamalla sen rotaatiota z akselilla ensiksi -45 asteeseen asti ja sitten +45 asteeseen. Päätin kuitenkin siirtää kääntyilyn Coroutine-metodilla pois Update-funktiosta, että sitä on helpompi hallita, esimerkiksi laitettaessa se päälle ja pois. Tein myös sen kääntymisen erillä tavalla täältä löytyneitä ohjeita soveltamalla (Unity answers 2011. Quaternion.Slerp in a coroutine). Nyt valvontakamera kääntyilee vasemmalle ja oikealle tasaista vauhtia tiettyyn kulmaan ja tietyssä ajassa. Tämän jälkeen aloin tekemään sitä, että valvontakamera tunnistaisi pelaajan ja alkaisi katsomaan sitä kohti. Tein tämän lisäämällä sille kolmion muotoisen Trigger collider-komponentin alueen, joka edustaa sen näkökenttää. Nyt voin koodissa kutsua OnTriggerExit, OnTriggerEnter ja OnTriggerStay -funktioita, joiden avulla voin ohjelmoida valvontakameraan toimimaan eri tavalla, kun pelaaja tulee alueen sisälle, pysyy sen sisällä tai lähtee alueelta pois. Tämän jälkeen tein TrackPlayer nimisen Coroutine-rutiinin, jonka laitan päälle pelaajan mennessä sisään alueelle, ja pois päältä, kun pelaaja poistuu alueelta. Minun täytyi tehdä kameran kääntyminen eri tavalla pelaajaa seuratessa, koska pelaajan paikka voi muuttua joka ikisellä kuvataajuuden päivityksellä, toisinkuin vasemmalle ja oikealle kääntyilyssä kameralla on tietyt kulmat

mihin se kääntyy tietyssä ajassa. Löysin ratkaisun tähän googlaamalla vähän ja minun täytyi käyttää Quaternion.RotateTowards-metodia (Unity Dokumentaatio 2019d. Quaternion.RotateTowards). Ensiksi minun täytyy kuitenkin laskea kulma, että missä pelaaja on valvontakameraan nähden, teen tämän miinustamalla valvontakameran paikka pelaajan paikasta, jonka jälkeen voin Mathf.Atan2-metodilla laskea niiden välisen kulman. Tämän jälkeen voin kääntää kameraa joka kuvataajuuden päivityksellä pelaajaa kohti tietyllä kääntymisvauhdilla (kuvio 17.).

```
Quaternion targetRot = Quaternion.Euler(new Vector3(0.0f, 0.0f, angle));

var step = speed * Time.deltaTime;
transform.rotation = Quaternion.RotateTowards(transform.rotation, targetRot, step);
```

KUVIO 17. Kuva koodista, joka kääntää valvontakameraa tiettyä kulmaa kohti tietyllä nopeudella joka kuvataajuuden päivityksellä.

Tämän jälkeen täytyi enää korjata muutamat ohjelmointivirheet kameran kääntyilyyn liittyen, esimerkiksi kun pelaaja menee yli kulmien mihin kamera ei saisi enää kääntyä, sen kääntymistä täytyy rajoittaa ja se jää sitten odottamaan hetkeksi. Kun kaikki ohjelmointivirheet oli korjattu niin minun täytyi enää tehdä kamerasta tuhottava ja lisätä sille pudotus koodi, jotta se pudottaisi rahaa. Siitä oli helppo tehdä tuhottava, kun pystyin vain kopioimaan tarvittavat koodit pelin vihollisilta. Päivän päätteeksi sain kameran täysin valmiiksi.

Viikkoanalyysi

Viikon aikana testasin vihollisia tekemälläni testiradalla ja korjasin niistä lukuisia ympäristössä liikkumiseen liittyviä ohjelmointivirheitä, kunnes ne toimivat testiradalla täydellisesti. Lisäksi minä suunnittelin ja tein pelin vihollisille taistelu- ja partiointimoodit, jotka monipuolistavat peliä yllättävänkin paljon. Taistelumoodi toimii käytännössä siten, että kun vihollinen näkee pelaajan, tai ottaa pelaajalta vahinkoa, niin käynnistyy ajastin, jonka aikana vihollinen koko ajan pyrkii katsomaan pelaajaa kohti ja yrittää aktiivisesti tulla taisteluetäisyydelle ja hyökätä. Tämä tekee pelistä paljon vaikeamman, sillä ennen pelaaja on voinut hyppiä vihollisen yli puolelta toiselle, ja vihollinen on aina unohtanut pelaajan täysin, kun pelaaja on kadonnut sen näkökentästä. Partiointimoodi toimii käytännössä niin, että vihollisille arvotaan suunta ja kävelyaika pelin käynnistyessä, viholliset kävelevät arvotun ajan verran arvottuun suuntaan, kunnes ne pysähtyvät arvotuksi ajaksi ja sitten taas sama uudestaan. Tämä tekee pelistä paljon monipuolisemman, sillä ennen viholliset van kävelivät tiettyyn suuntaan, kunnes osuvat seinään ja sitten vaihtoivat suuntaa. Kun olin tehnyt nämä asiat niin viimeiseksi tein valvontakameran, jonka on tarkoitus aiheuttaa pelaajalle sellainen olo,

että häntä tarkkaillaan. Valvontakamerat eivät tee mitään muuta kuin kääntyilevät hitaasti ja huomatessaan pelaajan, tuijottavat tätä kohti.

Viikon aikana ei ollut erityisiä ongelmia, mutta kameran kääntyily oli yllättävän vaikea tehdä, koska sen piti kääntyillä tasaista vauhtia ja osoittaa oikeaan suuntaan sekä sillä täytyi olla kääntymisrajat. Googlettamalla selvisin kuitenkin näistä ongelmista ihan hyvin.

Viikon aikana tunnen kehittyneeni taas paljon ohjelmoijana, alkuviikko oli täynnä testaamista ja korjaamista, joka on nyt aika rutiininomaista, mutta puolivälissä viikkoa sain sopivan vaikeita tehtäviä, jotka haastoivat minua juuri sopivasti ja loppuviikosta oli sitten vaikeampi tehtävä, joka vaati paljon enemmän googlausta ja tutkimista kuin koodausta. Loppujen lopuksi tällainen viikko on kehittymisen kannalta juuri tasapainoinen ja hyvä mielestäni.

4 POHDINTA

Tein opinnäytetyön päiväkirjamallisena, koska se sopi parhaiten tilanteeseeni, jossa teen samaa projektia päivittäin. Päiväkirjamallisen opinnäytetyön tekeminen auttoi minua seuraamaan omaa kehitystäni ja oppimistani kriittisesti joka viikko ja kehittymään normaalia enemmän.

Opinnäytetyö tulee olemaan hyödyllinen lukijoille, jotka eivät ole koskaan olleet pelialalla töissä, mutta heitä kiinnostaisi se, koska tässä opinnäytetyössä näkee, millaista on olla töissä pienessä pelifirmassa, joka on realistisin työpaikka aloittelevalle peliohjelmoijalle.

Viimeisen kymmenen viikon aikana olen kehittynyt todella paljon peliohjelmoijana kaikissa siihen liittyvissä taidoissa. Unity-pelimoottorin käytöstä on tullut rutiininomaisempaa, ja olen oppinut myös paljon Unity-pelimoottorin sisäisten komponenttien käytöstä, kuten Unity-pelimoottorin partikkeli- tai animaatiosysteemin käytöstä. C#-ohjelmointikielestä olen oppinut todella paljon uusia asioita, joiden oppiminen yleensä tapahtui jonkin ongelman kautta, joka oli niin vaikea, että minun täytyi googlata apua kyseiseen ongelmaan. Yleensä hain apua Stackoverflow-sivustolta, jossa ihmiset kysyvät apua kaikenlaisiin ohjelmointiongelmiin. Unity-pelimoottorin sisäiset C#-ohjelmoinnin ongelmat ovat hyvin yleisiä siellä, ja niistä löytyy todella paljon keskustelua, joten on suhteellisen helppoa löytää yleensä jokin keskustelulanka, jossa on sama tai samantapainen ongelma kuin minulla ja johon sitten monet ihmiset ovat vastanneet omilla ratkaisuillaan. Sen jälkeen kyse on enää siitä, että soveltaa heidän antamiaan ratkaisuja omaan koodiinsa, jotta saa sen toimimaan halutulla tavalla. Opin myös paljon versionhallinnan käytöstä, koodin kommentoimisesta sekä koodin suunnittelusta. Loppupuolella päiväkirjaa aloin joskus suunnittelemaan koodin kommentoimalla koko koodin ennen kuin aloin koodaamaan. Kommentointi toimi tavallaan pohjapiirustuksena koko ominaisuudelle. Tein tämän silloin, kun koodia oli niin vaikea suunnitella, että en saanut keksittyä, miten se pitäisi toteuttaa. Kun koodin rakenteli ensiksi kommenteilla, niin että se on loogisesti ja rakenteellisesti järkevä, oli paljon helpompaa koodata päälle tietty toiminto. Kehityin myös todella paljon tiimityöskentelyssä ja projektinhallinnassa. Opin kommunikoimaan paremmin tiimijäsenteni ja varsinkin pääsuunnittelijan kanssa. Projektinhallinnassa opin aikatauluttamaan tehtäviä aikaisempaa realistisemmin. Opin myös pelisuunnittelusta todella paljon ja varsinkin siitä, että pelin tasapainottaminen on hyvin vaikeaa ja vaatii todella paljon pelin laajaa testaamista.

Päiväkirjamallisen opinnäytetyön aikana projekti edistyi todella paljon, mutta siihen jäi vielä paljon tehtävää, ennen kuin peli voidaan julkaista. Peliprojekteissa monesti ilmestyy uusia tehtäviä ja edellisten kriteerit ja vaatimukset muuttuvat ja vanhoja täytyy uusia ja korjailla, mikä pitkittää projektin kestoa yllättävänkin paljon. Tällainen johtuu siitä, että tiimimme oli kokematon projektinsuunnittelussa. Opin myös näiden kymmenen viikon aikana sen, että pienessäkin peliprojektissa voi olla yllättävän paljon tekemistä, varsinkin koodin kannalta. Monissa pienissäkin peliprojekteissa kestää yhdestä kolmeen vuotta tehdä ne valmiiksi.

LÄHTEET

Abhinav a.k.a Demkeys 2017. Particle System Trails | Unity Particle Effects | Visual FX. Hakupäivä 12.11.2019. https://www.youtube.com/watch?v=agr-QEsYwD0.

Brackeys 2017. Shooting with Raycasts - Unity Tutorial. Hakupäivä 28.11.2019. Saatavilla: https://www.youtube.com/watch?v=THnivyG0Mvo

Lumidi Developer 2018. Camera Shake In Cinemachine - Unity Mini Tutorial. Hakupäivä 14.11.2019. Saatavilla: https://www.youtube.com/watch?v=O2Pg8e2xwzg

StackExchange 2014. Timing of coroutines in Unity is never precise. Hakupäivä 1.11.2019. Saatavilla: https://gamedev.stackexchange.com/questions/80560/timing-of-coroutines-in-unity-is-never-precise

Unity answers 2011. Quaternion.Slerp in a coroutine. Hakupäivä 24.1.2020. Saatavilla: https://answers.unity.com/questions/195895/quaternionslerp-in-_a-co routine.html

Unity answers 2017. Collision in fast speed. Hakupäivä 28.11.2019. Saatavilla: https://answers.unity.com/questions/1372430/collision-in-fast-speed.html

Unity Dokumentaatio 2019a. ExecuteInEditMode. Hakupäivä 2.12.2019. Saatavilla: https://docs.unity3d.com/ScriptReference/ExecuteInEditMode.html

Unity Dokumentaatio 2019b. Quaternion.RotateTowards. Hakupäivä 24.1.2020. Saatavilla: https://docs.unity3d.com/ScriptReference/Quaternion.RotateTowards.html

Unity Dokumentaatio 2019c. Random.value. Hakupäivä 16.10.2019. Saatavilla: https://docs.unity3d.com/ScriptReference/Random-value.html?_ga=2.121987970.880795514.1571136899-1054107281.1548325174

Unity Dokumentaatio 2019d. Vector2.Reflect. Hakupäivä 21.11.2019. Saatavilla: https://docs.unity3d.com/ScriptReference/Vector2.Reflect.html

Webcam 2017. Unity Tutorial - Create a Customizable Weapons System with Modular Code: Part 1. Hakupäivä 18.11.2019. Saatavilla: https://www.youtube.com/watch?v=35BmB7s3OfY