OULUN AMMATTIKORKEAKOULU

Jarno Kouvalainen, Teemu Kontio

PELI-IDEAN TOTEUTUS PROTOTYYPIKSI

Oman peli-idean toteutusprosessi

PELI-IDEAN TOTEUTUS PROTOTYYPIKSI

Oman peli-idean toteutusprosessi

Jarno Kouvalainen, Teemu Kontio
Opinnäytetyö
Syksy 2017
Tietojenkäsittelyn koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma, Web-sovelluskehitys

Tekijä(t): Jarno Kouvalainen, Teemu Kontio
Opinnäytetyön nimi: Peli-idean toteutus prototyypiksi
Työn ohjaaja: Teppo Räisänen
Työn valmistumislukukausi ja -vuosi: Syksy 2017 Sivumäärä: 37 + 0

Opinnäytetyö on tehty oman peli-idean suunnittelemisesta ja sen pelattavan prototyypin toteuttamisesta. Sillä ei täten ole toimeksiantajaa. Tavoitteisiin kuului luoda peli, jossa yhdistetään sellaisia pelimekaniikkoja ja peligenrejä yhteen, mitä ei yleensä ole peleissä yhdistetty.

Työssä käytettiin Unity 3D -pelimoottoria, ja pelin ohjelmointi on toteutettu C#-ohjelmointikielellä. Näihin päädyttiin siksi, koska nämä ovat kummallekin työn tekijälle tuttuja asioita, joiden käytössä kummallakin on kokemusta.

Prototyyppiin ei saatu täysin suunnitelmien mukaan siihen tavoiteltuja ominaisuuksia. Pelin keskeisiä systeemejä jäi puutteellisiksi, joten prototyyppi ei ole vielä testattavalla tasolla.

Asiasanat: Pelikehitys, Drafting, Bullet hell, Unity, Taitojen ohjelmointi

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems, Web Application Development

Author(s): Jarno Kouvalainen, Teemu Kontio
Title of thesis: Developing a game idea to a prototype
Supervisor(s): Teppo Räisänen
Term and year when the thesis was submitted: Autumn 2017 Number of pages: 37 + 0

This thesis is about designing our own game idea and then developing it to a playable prototype. The goal of the project was to create a game, that combines the kind of game mechanics and genres that are not usually seen together in a single game.

The development was done with Unity game engine using C# programming language. They were selected due to their familiarity and because we have a lot of experience and knowledge about them and their usage.

The resulting prototype did not get all the features we were planning to implement in it. Some of the game's core systems were left incomplete, so the prototype is not yet in a testable state.

Keywords: Game Development, Drafting, Bullet hell, Unity, Skill Development

TERMISTÖ

MMORPG - Lyhenne sanoista Massive Multiplayer Online Role Playing Game, joka tarkoittaa verkkoroolipeliä, jota pelaavat tuhannet ihmiset.

Prototyyppi - Ensimmäinen versio pelistä, jossa on perusmekaniikat implementoitu ja peliä voi testata. Pelissä ei ole isoa määrää sisältöä eikä ulkoasua ole viilattu tässä versiossa.

Pool - Kokoelma kyseistä asiaa. Poolissa voi olla mitä tahansa, mitä käytetään samassa käyttötarkoituksessa esimerkiksi samantyyppiset kortit, eli toisin sanoen pakka kortteja on eräänlainen pool.

Core loop - Käsite millä selitetään mitä pelaaja tekee milloinkin ja siitä näkee nopeasti miten pelin kuuluisi edetä.

Skripti - Tiedosto tai teksti, joka on koodia

Luokka - Ohjelmoinnissa käytetty termi, joka tarkoittaa eräänlaista kokoelmaa, jossa voi olla muuttujia, metodeja ja muuta informaatiota.

Objekti - Objektilla tarkoitetaan jotain pelissä olevaa asiaa, jossa on tietoa ja jota voi muokata. Esimerkiksi luokka voi olla objekti.

ScriptableObject - Unity-pelimoottorissa käytettäviä objekteja, joiden yleisin tarkoitus on pitää dataa tallessa ilman, että niitä tarvii luoda oikeiksi pelimaailman asioiksi.

Asset - Tiedosto projektissa, joka voi olla vaikka kuvatiedosto, äänileike tai skripti tiedosto.

Framework - Ohjelmistokehityksessä käytetty termi koodikirjastolle, jota voi käyttää runkona tai pohjana omassa ohjelmistossa. Frameworkit yleensä tarjoavat tietynlaisia toiminnallisuuksia valmiina käytettäväksi.

PvP - Lyhenne sanoista player versus player, ja tarkoittaa peliä, jossa pelaajat taistelevat toisiaan vastaan

UI - Lyhenne sanoista User Interface, joka tarkoittaa käyttöliittymää

SISÄLLYS

1 JOHDANTO

Tämä opinnäytetyö kertoo oman peli-idean toteuttamisesta pelattavaksi prototyypiksi. Pelin perusidea oli keksittynä jo ennen opinnäytetyön aloittamista, ja tässä työssä käydään läpi sen jatkosuunnittelua sekä luomisprosessia. Pääpaino keskittyy kuitenkin pelin draftaus- ja taitosysteemin suunnitteluun ja toteutukseen. Opinnäytetyöllä ei siis ole toimeksiantajaa, vaan aihe on itse keksitty.

Tätä työtä ei ole tarkoitus lukea ohjeena sille, miten pelejä tehdään, vaan se on eräänlainen kerronta oman peli-idean toteuttamisesta. Valitsimme tämän aiheen, koska olemme kauan olleet innokkaita pelaajia ja meillä on ollut haaveita pelien kehittämisestä nuoresta lähtien. Haluamme toteuttaa omia ideoitamme, sekä oppia luomaan viihdyttäviä ja merkityksellisiä pelejä.

Projektin tavoitteena oli alkuperäisen idean perusteella yrittää yhdistää bullet hell -genre draftauksen, verkkopelaamisen ja muokattavien taitojen kanssa. Pelistä oli tarkoituksena tulla strateginen ja nopeatempoinen adrenaliinia nostattava pelaajien välinen taistelupeli, joka tekee jokaisesta erästä mieleenpainuvan kokemuksen.

Opinnäytetyössä kerrotaan aluksi hieman pelinkehityksen historiasta, sekä Unity3D-pelimoottorista ja sen roolista tämän päivän pelinkehittäjän keskuudessa. Sen jälkeen keskitytään itse peliprojektiin ja käydään läpi sen kehitysvaiheita aloittaen suunnittelusta ja jatkaen siitä toteutukseen.

2 PELIKEHITYS

Pelinkehityksen prosessi on pitkä ja prosessiin yleensä tarvitaan monen alan ammattilaisia. Pelin elinkaareen kuuluu konseptin luominen, idean myyminen sijoittajille, prototyypin teko, lopullisen suunnitelman luominen sekä kehitys, markkinointi, testaus ja myynti. Prosessi voi kestää puolesta vuodesta puoleentoista vuoteen tai jopa pidempään, ja budjetti voi olla jopa kymmeniä miljoonia euroja (Wikimedia Foundation, Inc., 2017. Viitattu 18.12.2017).

Historia pelikehityksestä lähtee liikkeelle vuodesta 1952 brittiläisen professorin A.S. Douglasin luomasta OXO-pelistä, joka tunnetaan myös ristinollana, jonka Douglas loi osana väitöskirjaansa. (History, 2017. Viitattu 18.12.2017.) Pelikehitys sai siis alkunsa tutkimuslaboratorioissa, mutta ensimmäiset tietokoneet olivat todella kalliita ja isoja, joten ei ole yllätys että kuluttajakäyttöön ei ensimmäisiä pelejä kehitettykään.

1900-luvulla on pelinkehityksen osalta saavutettu muutamia merkittäviä merkkipaaluja, joita ovat mm. Space Invaders -arcadepelin julkaisu vuonna 1978 ja Activision-pelifirman perustaminen vuonna 1979, joka oli ensimmäinen peliyritys, joka loi itsenäisiä pelejä ilman laitteita (History, 2017. Viitattu 18.12.2017).

Nykypäivänä pelikehitys on erittäin nopeaa kasvava ala. Uusia pelejä tulee maailmalla kehitetyksi koko ajan. Pelinkehityksen suosio perustuu pelien suosioon. Pelien maailmanlaajuiset myyntilukemat ovatkin kasvaneet vuosi vuodelta suuremmaksi, ja tänä vuonna pelien yhteenlasketuksi myyntituloksi on arvioitu tulevan jopa 108,9 miljardia dollaria (Newzoo, 2017. Viitattu 19.12.2017).

3 UNITY3D

Unity3D on järjestelmäriippumaton pelimoottori, jolla voidaan luoda 2- tai 3-ulotteisia pelejä ja simulaatioita eri alustoilla. Unityn tärkeimpiä ominaisuuksia kehittäjille ovat sen helposti lähestyttävä "drag and drop" toiminnallisuus, joka mahdollistaa Unityn editorin käytön kokemattomillekin, kaksikymmentäseitsemän (27) erilaista alustaa, mille voi kehittää ohjelman sekä Unityn community ja asset store, joista saa apua kehityksen eri vaiheissa (Wikimedia Foundation, Inc., 2017. Viitattu 18.12.2017).

Vuonna 2005 Unity Technologies julkaisi Unity3D-pelimoottorin Applen OS X -käyttöjärjestelmälle. Tiimi lähti nopeasti laajentamaan muille alustoille ja vuonna 2008 pelimoottori oli tullut entistä kehittyneemmäksi ja tuottavammaksi, mikä mahdollisti Unityn laajentaa työvoimansa määrää (Dice, 2013. Viitattu 18.12.2017).

Unity oli ensimmäinen pelimoottori, joka tuki vuoden 2008 puolessavälissä julkaistua Apple iPhonea, mikä nosti kysyntää paljon. Unity teki myös ison kaupan Cartoon Networkin kanssa, joka kehitti Unity3D:llä FusionFall nimisen MMORPG-pelin lapsille, jota on pelannut kahdeksan miljoonaa ihmistä. Vuonna 2009 pelimoottorilla kehitettiin Tiger Woods PGA Tour Online -peli ja Unity sai lisää asiakkaita isoista firmoista kuten Microsoftista ja Ubisoftista (Dice, 2013. Viitattu 18.12.2017).

Unity3D on nykypäivänä (18.12.2017) johtava pelikehitysohjelmisto. Unity-pelejä tehdään ja pelataan enemmän ja yhä useampi kehittäjä käyttää Unity3D:tä ja sen palveluita pelien kehityksessä. (Unity3D, 2017. Viitattu 18.12.2017.)

4 PELI-IDEA

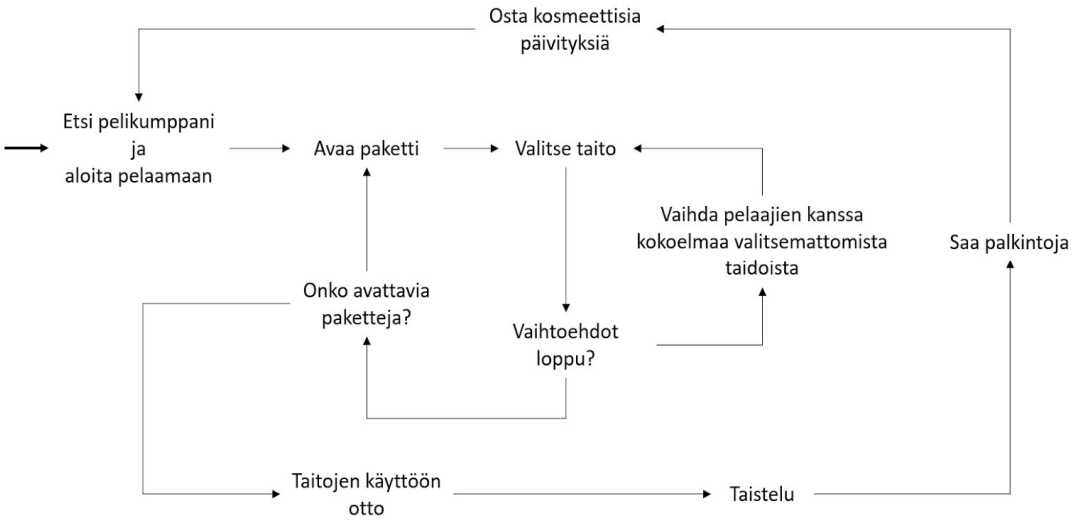
Pelin ydin oli alussa perustua jonkinlaiseen taitojen yhdistelyyn ja draftaukseen.
Ideaksi tuli sitten se, että pelaajille draftataan taitokomponentteja, joita he voivat yhdistellä luodakseen heille hyödyllisiä taitoja eriin. Erien suunniteltiin olevan PvP-taisteluita. Ideaan tuli myös lisäksi keskeiseksi asiaksi bullet hell -genre.

4.1 EpicDraft

“EpicDraft” on peli-ideasta tehty prototyyppi, jonka tarkoituksena oli testata eri pelimekaniikkojen yhdistämistä toimivaksi kokonaisuudeksi. Pelissä pelaajan on tarkoitus aluksi valita taitoja ja rakentaa niistä synerginen kokoelma, jolla pelaaja sitten taisteluvaiheessa yrittää voittaa vastustajansa nokkeluudellaan.

4.2 Gameplay loop

Pelin core gameplay loop koostuu kahdesta loopista, jotka ovat esiteltynä kuviossa 1. Draftausvaiheessa pelaajat avaavat määrätyn määrän paketteja yhden kerrallaan ja valitsevat yhden taidon. Valinnan jälkeen pelaajat vaihtavat pakkaukset keskenään ja tätä jatketaan kunnes kaikki taidot on jaettu. Tämän jälkeen pelaajat valmistautuvat tulevaan taisteluvaiheeseen ja varustavat aluksensa taidoilla mitä he ovat valinneet. Taisteluvaiheessa pelaajat taistelevat kokoamillaan taidoilla kunnes toinen pelaajista voittaa.



KUVIO 1 EpicDraftin core gameplay loop

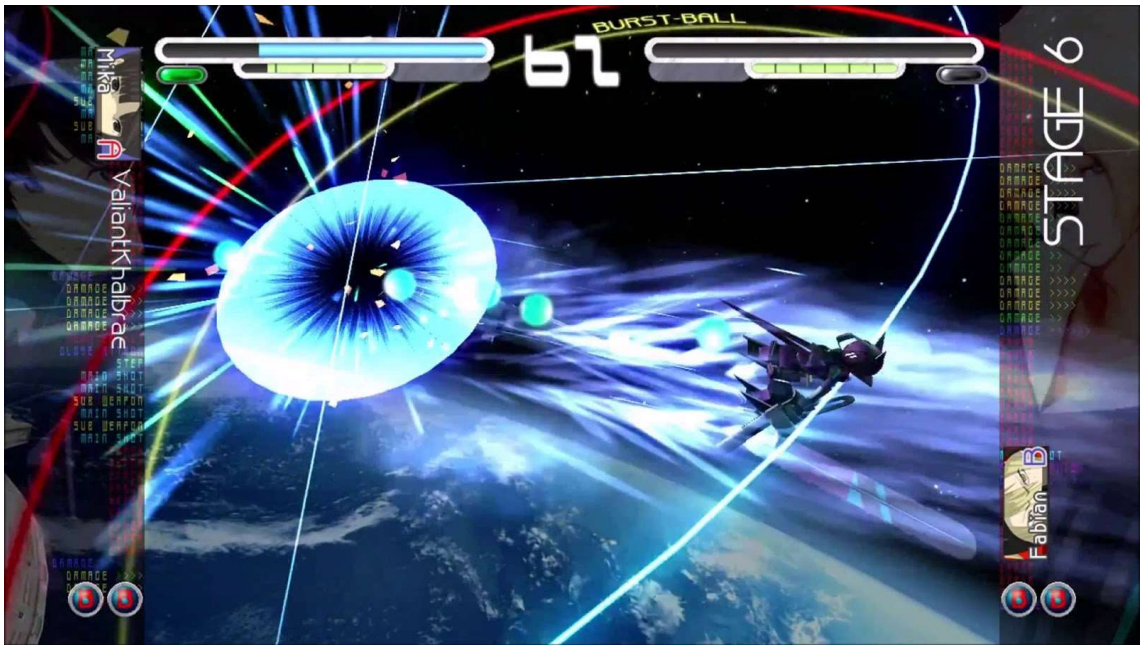
4.3 Progressio

Taisteluista palkinnoksi saatava valuutta antaa pelaajalle mahdollisuuden ostaa kosmeettisia tavaroita, joilla voi muokata aluksen ulkonäköä. Valuutalla ei voi ostaa pelillistä etua, koska pelin alkaessa kaikki pelaajat ovat yhtä vahvoja ja vain pelitieto ja tuuri tuovat pelaajalle etua taistelussa.

5 SUUNNITTELU

Yksi pelinkehityksen osa-alue on pelin suunnittelu ja siinä yhdistyvät mm. visuaalinen suunnittelu, luova kirjoittaminen ja tekninen taito. Suunnittelija ottaa luovan roolin ja luo suunnitelman pelin maailmasta ja kaikesta peliin kuuluvasta (International student, 2017. Viitattu 18.12.2017).

EpicDraftin idea on tullut yhdistelemällä erilaisia mekaniikkoja, joita on aikaisemmin haluttu toteuttaa oppimismielessä ja samalla kokeilla miten ne toimivat yhdessä. Peli on saanut suurta inspiraatiota erilaisista draftaus-, bullet hell-, rpg- ja PvP-peleistä. Yksi inspiraation lähteistä on WarTech: Senko No Ronde Mika -peli, joka esitetään kuviossa 2.



KUVIO 2 Kuvankaappaus WarTech: Senko No Ronde Mika -pelistä

5.1 Pelikonsepti

Pienestä ideasta lähtevä pelikonsepti on dokumentti, jossa kerrotaan lukijalle helposti ymmärrettävässä muodossa se, millainen peli olisi. Pelikonseptin tehtävä on kertoa osallisille tärkeät tiedot pelistä. Siitä voi paljastua esimerkiksi minkälaista grafiikkaa peliin olisi tulossa, millä tavalla pelillä on tarkoitus tienata ja kannattaako peliin investoida. Kuten kuvion 3 osoittamassa esimerkissä näytetään, voi pelikonseptissa vaikka näyttää kuvia suunnitelluista efekteistä, sekä kertoa niistä ja niiden tarkoituksista pelissä.

KUVIO 3 Esimerkki konseptin sisältämästä peligrafiikasta

Peli-idea on tärkeä osa konseptia ja se kannattaa olla dokumentin alussa. Se voi olla vain muutaman kappaleen pituinen teksti, jossa kerrotaan pelistä, sekä sen maailmasta ja hahmoista. Siihen voi myös sopia lyhyt kuvaus pelin tarinasta. Tärkeää on saada lukija kiinnostumaan peli-ideasta (Pluralsight, 2014. Viitattu 18.12.2017).

5.2 Draftaus

Draftaus on peleissä pohjimmiltaan resurssien jakomekaniikka, jossa resurssit jaetaan pelaajien kesken mahdollisimman tasaisesti. Se on laajasti levinnyt eri pelilajityypeille ja formaateille. Draftaus on vahva pelimekaniikka, joka tuo peliin syvyyttä ja monimuotoisuutta, ja joissain tapauksissa draftaus voi olla pelin ydin. (Make A Game Of that, 2013, viitattu 14.12.2017.)

5.2.1 Säännöt

Draftauksesta on monia variaatioita, jotka muuttavat miten draftaus suoritetaan. Vaihtelevuutta voi olla mm. näissä säännöissä:

- ovatko resurssit näkyvissä kaikille pelaajille vai ei

- tietävätkö pelaajat mitä toiset ovat valinneet
- jaetaanko koko pool vai vain osa siitä
- rajoitetaanko pelaajia valitsemasta koko poolista vai ei
- ovatko pelaajilla omat henkilökohtaiset poolit, josta valita vai onko kaikilla pelaajilla yhteinen (Make A Game Of That, 2013, viitattu 14.12.2017)?

KUVIO 4 Hearthstonen areena-draftaus

Kuviossa 4 on kuvankaappaus korttipakan draftaustilanteesta Blizzardin julkaisemassa Hearthstone-pelin Arena-pelimuodossa. Hearthstonen areena-pelimuodossa pelaajille draftataan kolmekymmentä (30) korttia pakkaan ja kaikki julkaistut omistettavat kortit ovat pelaajan henkilökohtaisessa poolissa, mistä saa kolme sattumanvaraista korttia valittavaksi. Pelaajat eivät voi tietää toistensa pakoista mitään etukäteen, eivätkä voi myöskään vaikuttaa toistensa draftausprosessiin, mikä tekee draftauksesta yksinkertaisempaa.

Erilaiset sääntöyhdistelmät luovat erilaisia kokemuksia ja EpicDraftissa valitut säännöt antavat mahdollisuuden pelaajille vaikuttaa poolin sisältöön, tehdä strategisia päätöksiä ja reagoida vastustajan tekemiin valintoihin mukauttamalla omaa strategiaansa draftauksen aikana. Pelaajien draftaamat taidot ovat myös monikäyttöisiä, mikä vähentää ei-toivottujen taitojen määrää, sillä niitä voi käyttää toisten taitojen päivittämiseen.

5.2.2 Dynaaminen draftaus

Draftauksen dynaamisuus on mielenkiintoista, sillä se voi tehdä pelistä hyvin erilaisen jokaiselle pelaajalle. Säännöt, draftauksen vaihe, pelaajien määrä ja valittavien resurssien monimutkaisuus vaikuttavat paljon pelaajan tekemään valintaan ja mm. valittavien resurssien näkeminen ensimmäisenä pelaajana antaa paljon strategista etua, koska muut pelaajat eivät voi tietää mitä poolista on jo otettu.

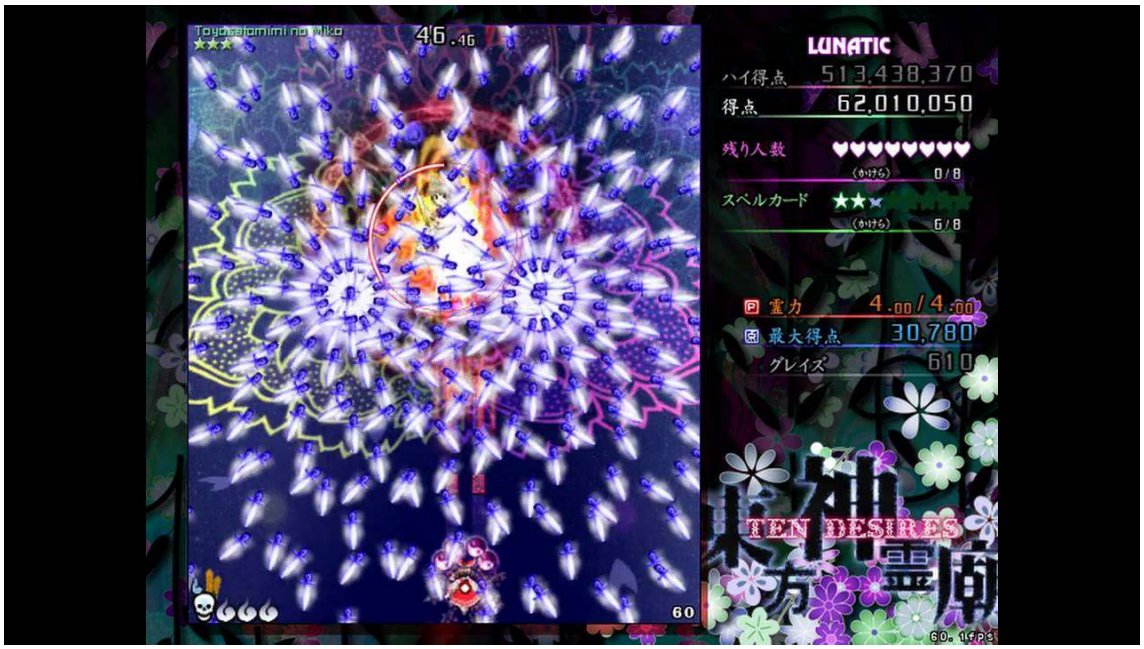
Tieto muiden pelaajien valinnoista ja omat valinnat vaikuttavat tuleviin valintoihin monella tapaa, koska jos pelaaja tietää jonkin resurssin olevan uhka, mutta ei itse tarvitse kyseistä resurssia, hän saattaa valita kortin tästä huolimatta estääkseen vastapelaajaa pääsemästä käsiksi kyseiseen korttiin. Resurssien arvo myös muuttuu draftauksen edetessä riippuen pelaajien tarpeesta.

5.3 Shoot ‘em up

Räiskintäpelien alalaji Shoot ‘em up, eli lyhennettynä shmup, on peligenre, jossa pelaaja ohjaa yhtä hahmoa tai ajoneuvoa, jonka tehtävä on tuhota aalloittain vihollisia. Pelit ovat joko ylhäältä tai sivulta päin kuvattuja ja ovat nopeatempoisia räiskintöjä, joita pidetään peleistä vaikeimpina (Fandom, 2016. Viitattu 19.12.2017).

5.3.1 Bullet hell

Bullet hell on Shoot 'em up -genren lajityyppi, jossa pelialue yleensä tulee täyttymään useilla ammuksilla, joita pelaajan pitää väistää. Se sai alkunsa 90-luvulla, kun 3D-pelien suosio oli nousemassa nopeasti ja 2D-pelien tekijät joutuivat keksimään keinoja kilpailla niitä vastaan. Bullet hell -tyyppisten pelien samaan aikaan ruudulla näkyvien suurten ohjusmäärien oli tarkoitus tehdä pelaajaan vaikutus. (Wikimedia Foundation, Inc., 2017, viitattu 14.12.2017.) Kuviossa 5 on kuvankaappaus shmup-tyylisestä Touhou Project -pelistä, joka näyttää hyvän esimerkin bullet hell -peleistä tunnetusta ruutuja täyttävästä ammusmäärästä.



KUVIO 5 Kuvankaappaus Touhou Project -pelistä

5.3.2 Bullet hell EpicDraftissa

Pelin prototyypin tavoitteisiin kuului kokeilla yhdistää draftaamis- ja bullet hell -mekaniikat ja luoda omanlainen bullet hell -kokemus pelaajille yhdistämällä mukaan muita pelaajille tutumpia mekaniikkoja. Tavoitteena on luoda samaan aikaan helpommin lähestyttävän, mutta myös kilpailullisemman ja korkean taitokaton omaavan pelin pelaajille, jotka eivät välttämättä ole pelanneet tämän genren pelejä.

Genrestä tuttu ominaisuus, ruudun täyttäminen ammuksilla, on yksi suurimmista tavoitteista EpicDraftia varten. Haluamme antaa pelaajille mahdollisuuden luoda uniikkeja ammuskuvioita väisteltäväksi, mutta myös niin, että pelaajat pystyvät yrittämään saada vastustajansa strategisesti kiinni.

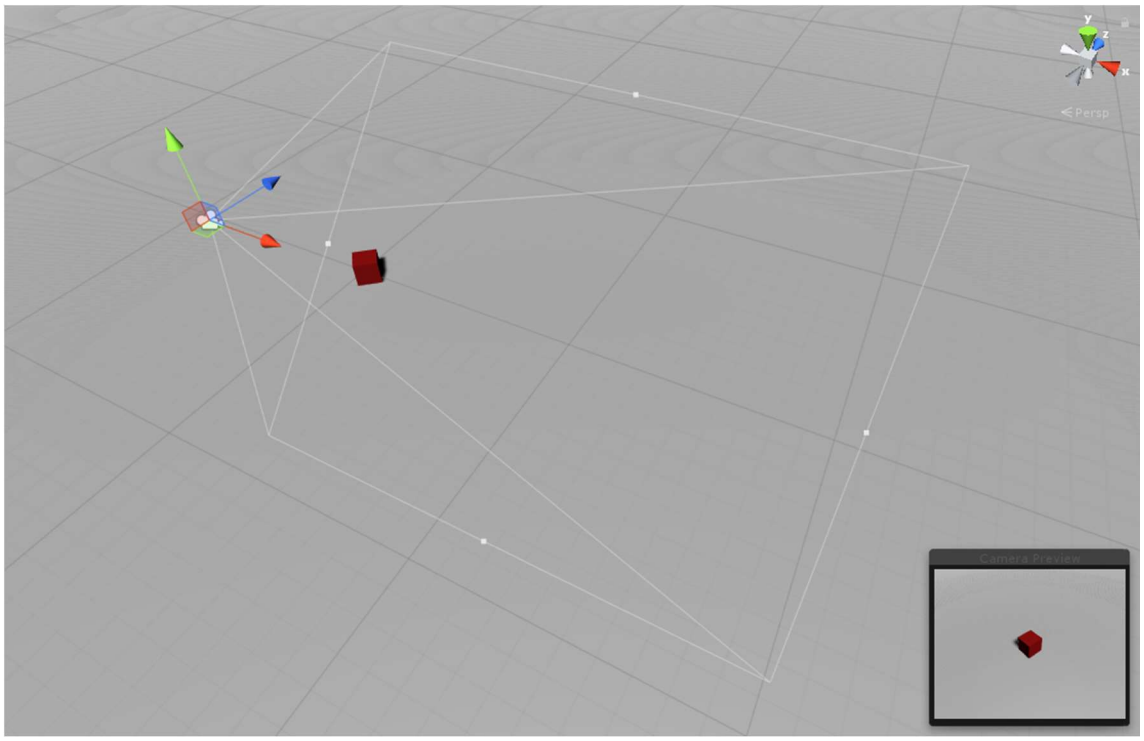
6 TOTEUTUS

Pelikehityksessä pelin toteutus voi tarkoittaa koko peliprojektin kehityskaarta, mutta tässä työssä sillä tarkoitetaan enemmänkin pelin teknistä toteutusta. Tässä kappaleessa käydään läpi pelin toteutuksen keskeisimpiä ominaisuuksia ja ratkaisuja.

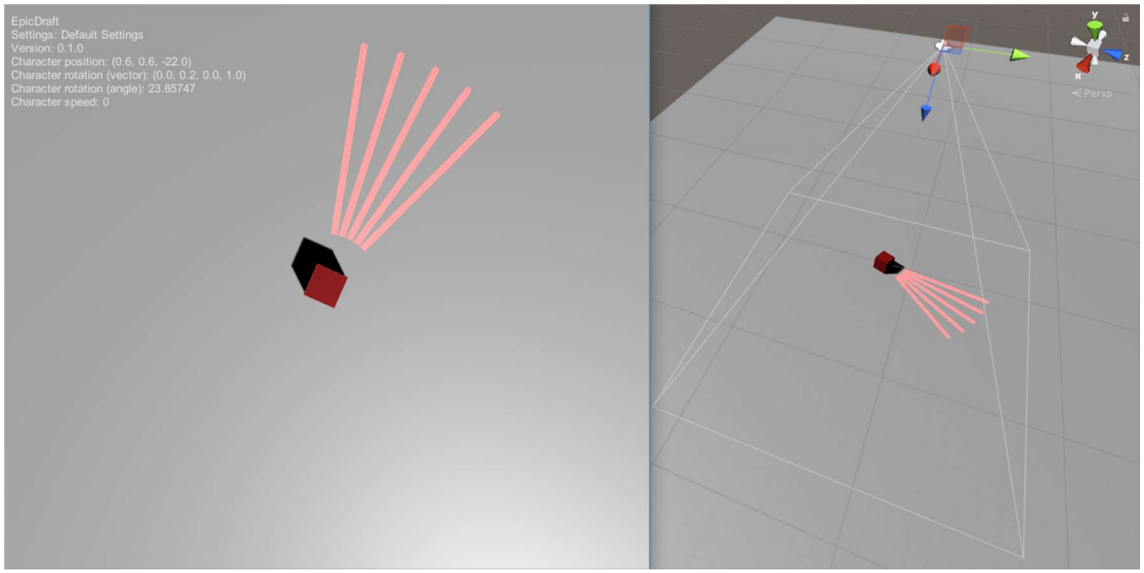
6.1 Yleiset ominaisuudet

Prototyypissä ei ole tällä hetkellä mitään malleja käytössä, vaan kaikki pelimaailmassa olevat objektit ovat kuutioita tai palloja. Pelihahmot ovat kuutioita ja väriltään punaisia, kun taas muut objektit ovat muun värisiä.

Kamerana toimii kolmannen persoonan kamera, joka kuvaa pelaajaa noin 45 asteen kulmassa takaapäin ja seuraa pelaajaa pitäen kuvakulman aina samana. Kuviossa 6 esitetään tämän kameran sijainti ja kuvakulma 3D-maailmassa, sekä sen kuvaama näkymä oikeassa alalaidassa. Tätä kameraa on käytetty väliaikaisesti erilaisten testausten vuoksi. Tarkoituksena on kuitenkin myöhemmin muuttaa kamera kuvaamaan pelaajaa suoraan yläpuolelta, mikä sopii paremmin bullet hell -tyyppiselle pelille. Lopulliseen versioon suunniteltu, ylhäältä päin pelaajaa kuvaava kamera on esitetty kuviossa 7, jossa vasemmalla näkyy pelaajan näkymä ja oikealla kameran sijainti ja kuvakulma pelimaailmassa.



KUVIO 6 Pelaajaa takaapäin kuvaava kamera



KUVIO 7 Pelaajaa ylhäältä päin kuvaava kamera

Koska EpicDraft on genreltään bullet hell, pelimaailman objektien luomista ja tuhoamista tapahtuu todella paljon lyhyessä ajassa. Tämä voi aiheuttaa suorituskykyongelmia, joten peliin otettiin käyttöön object pool -suunnittelumalli. Tämän mallin tarkoituksena on parantaa suorituskykyä ja muistinkäyttöä käyttämällä uudelleen kerran luotuja peliobjekteja sen sijaan, että niitä luotaisiin ja tuhottaisiin yksitellen joka kerta, kun niitä käytetään (Nystrom 2014, viitattu 18.12.2017).

6.2 Peruspelimekaniikat

Pelin perusmekaniikkoja ovat liikkuminen, ampuminen, taitojen valinta ja taitojen linkittäminen toisiinsa. Ampuminen tapahtuu yksinkertaisesti vain aktivoimalla taidon, joka on yhdistetty johonkin näppäimistön näppäinpainallukseen. Taito voi kuitenkin myös olla automaattisesti aktivoituva tai vaikka on/off-taito, jos se on sellaiseksi muokkautunut.

6.2.1 Liikkuminen

Pelihahmon liikuttaminen tapahtuu muokkaamalla hahmon Transform-komponentin sijaintiarvoja pelaajan näppäinpainallusten perusteella. Peli seuraa standardi WASD-näppäinasetelmaa, jossa pelaajahahmoa liikutetaan näppäimistön WASD-napeilla ja juoksun voi aktivoida vasemmasta Shift-napista. Pelissä ei ole hyppymahdollisuutta, joten kaikenlainen väistäminen tapahtuu liikkumalla, piiloutumalla tai suojautumalla jollain muulla tavalla. Liikkumista ohjataan CharacterMovementController-komponentilla, joka on itse tehty ohjausskripti pelissä oleville hahmoille.

Hahmon ohjauskomponentissa kutsutaan jokaisen fysiikkapäivityksen aikana HandleMovement-funktiota, joka esitetään kuviossa 8. Funktiossa katsotaan horisontaaliset (A, D) ja vertikaaliset (W, S) liikkumiskomennot pelaajalta. Samalla tarkistetaan, pitääkö pelaajan juosta, ja se tapahtuu vasemman Shift-näppäimen painamista seuraamalla. Juoksu- ja liikkumisdatat otetaan talteen muistiin, sillä niitä tarvitaan toisinaan eri tilanteissa. Transform-komponentin sijainti muokataan lopuksi liikkumistietojen perusteella.

```
protected virtual void HandleMovement()
{
    Vector3 moveInput = new Vector3(
        Input.GetAxisRaw("Horizontal"),
        0f,
        Input.GetAxisRaw("Vertical")
    );

    IsRunning = Input.GetKey(KeyCode.LeftShift);
    IsMoving = Movement.magnitude != 0f;

    Movement = moveInput.normalized * MoveSpeed * Time.deltaTime *
               (IsRunning ? RunningMultiplier : 1f);

    Transform.position += Movement;
}
```

KUVIO 8 Liikkumisen käsittelymetodi hahmon ohjauskomponentissa

Ohjauskomponentti hoitaa myös hahmolla tähtäämisen, mikä tapahtuu kutsumalla HandleRotation-funktiota, joka on esitetty kuviossa 9. Koska peli kuvaa pelaajan hahmoa kolmannesta persoonasta, tähtääminen tapahtuu seuraamalla hiiren osoittimen sijaintia pelihahmon ympärillä. Tähän käytetään raycastingia, jossa kamerasta “ammutaan” näkymätön säde hiiren osoittamaan suuntaan. Pelaajahahmon kohdalla on teoreettinen, horisontaalinen taso, jolla seurataan säteen osumakohtaa tasoon. Osumakohdan avulla saadaan tieto siitä, mihin kohtaan hiirellä tähdätään pelimaailmassa. Tämän avulla pelaajahahmo saadaan pyöritettyä katsomaan hiiren suuntaan.

```
protected virtual void HandleRotation()
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    Plane horizontalPlane = new Plane(Vector3.up, transform.position);
    float distance = 0;
    if (horizontalPlane.Raycast(ray, out distance))
    {
        LookPosition = ray.GetPoint(distance);
        transform.LookAt(LookPosition);
    }
}
```

KUVIO 9 Tähtäämisen käsittelymetodi hahmon ohjauskomponentissa

6.2.2 Taitojen valinta

Draftauksen toteuttaminen jäi projektissa kesken verkkopelin keskeneräisyyden vuoksi, mutta on tehtävälistalla jatkossa. Toteutuksen implementoinnin vaikeus riippuu paljon pelin sääntöjen monimutkaisuudesta. Yksinkertaisimmillaan toteutus voi olla vain se, että annetaan pelaajalle täysin sattumanvaraiset vaihtoehdot mistä valita, eikä valinnat vaikuta toisten pelaajien vaihtoehtoihin. Draftausprosessi on kuitenkin tarkoitus toteuttaa niin, että pelaajat valitsevat kortteja samanaikaisesti omasta poolistaan ja valinnan jälkeen pelaajat vaihtavat poolejaan.

Tämänhetkisessä versiossa draftaus on toteutettu staattisella DraftingManager-luokalla, joka hallitsee draftausprosessia. Luokan instanssille annetaan korttipaketit, joista jaetaan kortit pelaajille. Draftauksen alkaessa DraftingManager lähettää pelaajille vaihtoehdot mistä valita ja callback-funktion, jonka tarkoituksena on pelaajan valittaessa ilmoittaa takaisin DraftingManagerille valitsemansa kortin. Valintojen jälkeen DraftingManager lähettää pelaajille uudet kortit mistä valita, mutta se vain vaihtaa korttipoolit pelaajien kesken. Tätä jatketaan kunnes kortit ovat loppu ja draftausprosessi on käyty loppuun.

6.2.3 Taitojen linkitys

Taitojen linkityksen käyttöliittymäpuolen toteutus jäi kesken, mutta käytännössä jokaisen taidon itsessään käyttämisen sijaan voi taidon käyttää vaihtoehtoisesti toisien taitojen päivittämiseen. Pelaajalla on SkillInventory-luokka, jossa kaikki valitut taidot ovat ja pelaaja voi tulevan SkillInventory-käyttöliittymän kautta joko ottaa taidon käyttöön tai linkittää taidon päivittämään toista taitoa. UI huolehtii, että linkitys on sallittu ja sen jälkeen lisää referenssin päivitettyyn taitoon. Lopuksi pelin alkaessa taitoja koottaessa linkitykset otetaan huomioon ja lopulliset taidot luodaan.

6.3 Taidot

Taitojen toteuttaminen vaati runsaasti etukäteen suunnittelua. Tarkoituksena oli kehittää hyvin joustava systeemi, joka antaa taitojen tekijöille mahdollisuuden luoda taitokomponentteja, joilla voi tehdä melkein mitä vain. Nämä komponentit pitäisi voida myös yhdistää ilman rajoituksia yhdeksi taitokokonaisuudeksi. Tämä oli hieman haastavaa, sillä tällaista erittäin joustavaa, kaiken mahdollistavaa systeemiä on yleensäkin vaikea luoda, sekä se piti saada sopivasti toimimaan Unityssä mahdollisimman hyvällä suorituskyvyllä.

6.3.1 Eventit

Taidoissa päädyttiin käyttämään runsaasti C#-kielen delegaatteja ja eventtejä, sillä ne sopivat dynaamisesti käyttäytyvään toimintaan hyvin. Delegaatti tarkoittaa tyyppiä, johon voi tallentaa viittauksen metodiin, jolla on delegaattia vastaavat palautustyyppi ja parametrit (Microsoft Corporation 2015, viitattu 18.12.2017). Eventit mahdollistavat luokkien ja objektien huomauttaa toisia luokkia tai objekteja jonkin tilanteen tapahtumisesta (Microsoft Corporation 2015, viitattu 18.12.2017). Eventit käyttävät tyyppinään delegaatteja, kuten kuvion 10 esimerkki osoittaa.

```
public delegate void EventHandler(object sender, SkillEventArgs args);

public event EventHandler CoreBehaviour;
public event EventHandler OnActivate;
public event EventHandler AfterActivate;
public event EventHandler OnTick;
public event EventHandler OnHit;
public event EventHandler OnCritical;
public event EventHandler OnKill;
public event EventHandler OnDestroy;
```

KUVIO 10 Esimerkki delegaateista ja eventeistä, kuvassa EpicDraftin Skill-nimisen luokan eventtejä

Luokkia, joissa on eventtejä, kutsutaan julkaisijoiksi. Toiset luokat voivat tulla näiden eventtien tilaajiksi "lisäämällä" kyseisiin eventteihin metodin, jota kutsutaan handleriksi, ja jonka palautusarvo ja parametrit vastaavat eventtien delegaattityypin palautusarvoa ja parametrejä. Kuvio 11 näyttää esimerkin eventin tilausprosessista, jossa OnActivate-eventtiin lisätään Behaviour-niminen handler-metodi. Kun julkaisija-luokka laukaisee eventtejä kuvion 12 osoittamalla tavalla, niiden tilaajien handler-metodit tulevat automaattisesti kutsutuksi. Eventtien avulla objektit saadaan helposti reagoimaan eri tilanteisiin ja tapahtumiin, mitä on hyödynnetty EpicDraftissa hyvin paljon.

```
    Skill.OnActivate += Behaviour;
}

public void Behaviour(object sender, SkillEventArgs args)
{

}
```

KUVIO 11 Esimerkki Skill-luokan OnActivate-eventin tilaamisesta. OnActivate-event tulee aktivoimaan Behaviour-nimisen metodin

```
if (OnActivate ≠ null)
    OnActivate(this, args);
```

KUVIO 12 Esimerkki OnActivate-eventin laukaisemisesta

6.3.2 Skill

Taidon käynnistäjänä toimii Skill-tyyppinen scriptable object, joka vastaa taidon valmistamisesta ja sen kokoamisesta. Se myös sisältää useita eventtejä, sekä perusdataa taidosta. Taidon pääaktivointi tapahtuu tämän luokan kautta.

Kun puhutaan taidoista itse pelissä sitä pelatessa, ei tarkoiteta niillä Skill-objekteja. Skill-objektit toimivat käytännössä vain luokkina, jotka kokoavat tarvitsevat tiedot pitäen ne tallessa ja käynnistävät taidon aktivoitumisprosessin. Pelin taidot ovat loppujen lopuksi kokoelma taustalla toimivia objekteja ja tapahtumia.

Pelaajalla voi olla useita Skill-objekteja taitokokonaisuuksina, ja niissä voi olla kiinni useita taitoa muokkaavia komponentteja, SkillModifiereja. Jos taitokokonaisuudessa ei ole mitään kiinni, joka muokkaa sen käyttäytymistä, ottaa se oman perustoiminnallisuutensa käyttöön. Tällainen perustaito luo vain yhden luodin pelimaailmaan, joka liikkuu sille määrättyä nopeutta eteenpäin.

6.3.3 SkillEventArgs

SkillEventArgs-luokka toimii tavallaan taidon keskustana ja kaikkea yhdistävänä linkkinä, ja se luodaan jokaisen taidon aktivoitumisen alussa. Se sisältää monenlaista dataa kyseisestä aktivoitumistapahtumasta sekä hyödyllisiä eventtejä. Tämä luokka on yksi muuttujista, joita eventit lähettävät niiden tilaajilleen eventtien aktivoitumistilanteessa. Sen takia tätä luokkaa syötetään useiden objektien välillä, minkä avulla kaikki saavat yhteyden aktivoidun taidon tietoihin.

6.3.4 SkillInstance

Taidon pelimaailmaan luomat objektit saavat SkillInstance-tyyppisen komponentin, eli jokainen taidon luoma asia on SkillInstance. Tämä komponentti pitää tallessa useanlaista taidon pelimaailman versioon liittyvää tietoa, jota taitoa muokkaavat skriptit, SkillModifierit, voivat muokata. Tässä komponentissa on myös suuri määrä eventtejä, joiden avulla eri tapahtumia voi aktivoitua erilaisissa tilanteissa. Aikaisemmin mainittua object pool -mallia seuraamalla kaikki SkillInstancet ovat lisäksi poolattuja.

Useimmissa tilanteissa taitojen luomat SkillInstancet ovat jonkinlaisia eteenpäin liikkuvia ammuksia, joilla yritetään osua vastustajiin. Tämä ei kuitenkaan aina ole totta, sillä muun muassa niiden liikkumisen, osumiskäyttäytymisen, muodon sekä koko käyttötarkoituksen voi muokata tai korvata täysin SkillModifier-objekteilla.

6.3.5 SkillModifier

Taito saa toiminnallisuutensa ja käyttäytymisensä SkillModifier-tyyppisistä scriptable objekteista. Nämä objektit sisältävät koodia, jotka muokkaavat taitoa eri tavoilla, ja näitä voi draftauksessa "linkata" taitoon kiinni, mikä yhdistää näiden toiminnot taitoon. EpicDraft kutsuu näitä objekteja itse pelin sisällä taidoiksi, mutta taustalla ja käytännössä nämä kuitenkin toimivat vain taidon muokkaajina.

SkillModifier-objektien sisältämä koodi pääsee käsiksi melkein kaikkeen mahdolliseen dataan, jolla on jotain tekemistä sen taidon kanssa, mihin objekti on yhteydessä. SkillModifier-objektin koodi voi esimerkiksi muokata taidon arvoja ja peruskäyttäytymistä, luoda esineitä pelimaailmaan, asettua reagoimaan automaattisesti tilanteisiin, liikuttaa pelaajaa tai tehdä melkein mitä tahansa muutakin. Tämän lisäksi taitojen alkuperäisen suunnittelun perusteella nämä kaikki voivat vielä tapahtua samanaikaisesti yhdelle taidolle.

6.3.6 Taidon aktivointiprosessi

Tässä kappaleessa käydään läpi esimerkkiprosessi taidon aktivoitumisesta, jolla pyritään selventämään, mitä ja missä järjestyksessä mitäkin tapahtuu. Koska taidoissa on yleensä linkitettynä taitoa muokkaavia komponentteja eli SkillModifiereja, taidot tulevat peleissä olemaan hyvin vaihtelevia ja usein erilaisia, joten tässä ei kerrota mitä taito käytännössä pelimaailmassa tekee.

Taidon aktivoituminen tapahtuu, kun pelaajahahmo kutsuu Skill-luokan aktivointimetodia. Hahmo syöttää argumentiksi oman Character-luokkansa kuvion 13 näyttämään metodiin. Tämän avulla

taito pystyy muistamaan, kuka oli taidon aktivoija ja omistaja.

```
public bool Activate(Character activator)
{
    return Activate(CreateNewArgs(activator), null);
}
```

KUVIO 13 Taidon aktivointimetodi

Aktivoinnissa luodaan uusi SkillEventArgs-luokka, johon tallennetaan alussa taidon aktivoitumistilanteeseen liittyvää tietoa. Tätä luokkaa tullaan käyttämään useassa paikassa tietojen hakemiseen ja muokkaamiseen sekä asioiden linkitykseen. Kuviossa 14 esitetään uuden SkillEventArgs-luokan luomiseen käytetty metodi.

```
SkillEventArgs CreateNewArgs(Character owner)
{
    SkillEventArgs args = new SkillEventArgs()
    {
        SkillBase = this,
        Owner = owner,
        ActivationPosition = owner.Transform.position,
        ActivationRotation = owner.Transform.rotation,
        LookPosition = owner.MovementController.LookPosition,
    };
    args.UpdateTime();
    return args;
}
```

KUVIO 14 Uuden SkillEventArgs-luokan luontimetodi

Tämän jälkeen siirrytään pääaktivointi-funktioon, joka on esitetty kuviossa 15. Ensiksi varmistetaan, että taito on valmiustilassa. Sitten tapahtuu tarkistus, jossa katsotaan taidon cooldown, joka päättää, voiko taitoa vielä aktivoida. Jos tarkistus menee läpi, siirrytään itse taidon käyttäytymisen aktivointiin. TryActivateBehaviour-metodi yrittää kutsua taidon ydinkäyttäytymisfunktion, ja sen onnistumisen perusteella laukaistaan aktivoitumiseventit ja taitoon linkatuiden SkillModifier-objektien koodi, jotka ovat asetettu aktivoitumaan tässä tilanteessa. Koska taidon ydinkäyttäytyminen voi vaihdella SkillModifier-scriptien vaikutuksesta, tässä tilanteessa voi jo pelimaailmassa tapahtua mitä vain. Yleensä tässä kohdassa luodaan kuitenkin ainakin yksi objekti pelimaailmaan, jossa on SkillInstance-komponentti. Tämä ei kuitenkaan ole pakollista. Ydinkäyttäytymisen aktivoinnin jälkeen siirrytään vielä jälkiaktivointiin, jonka avulla SkillModifierit voivat esimerkiksi muokata juuri pelimaailmaan luotuja asioita. Jälkiaktivointi on kuvattu kuviossa 16.

```
public bool Activate(SkillEventArgs args, params Type[] ignoreModifiers)
{
    bool success = false;

    if (!Initialized)
        Initialize();

    if (!CanActivate())
        return false;

    args.UpdateTime();
    args.ActivationOccurring = true;

    if (TryActivateBehaviour(args))
    {
        if (OnActivate ≠ null)
            OnActivate(this, args);

        TriggerModifiers(SkillModifier.Trigger.TriggerOnActivate,
                         this,
                         args,
                         ignoreModifiers);

        success = true;
    }

    args.ActivationOccurring = false;
```

KUVIO 15 Pääaktivoinnin koodi

```
    args.ActivationOccurring = false;

    if (success)
    {
        if (AfterActivate ≠ null)
            AfterActivate(this, args);

        TriggerModifiers(SkillModifier.Trigger.TriggerAfterActivate,
                         this,
                         args,
                         ignoreModifiers);

        SubscribeToEvents(args);
    }

    return success;
}
```

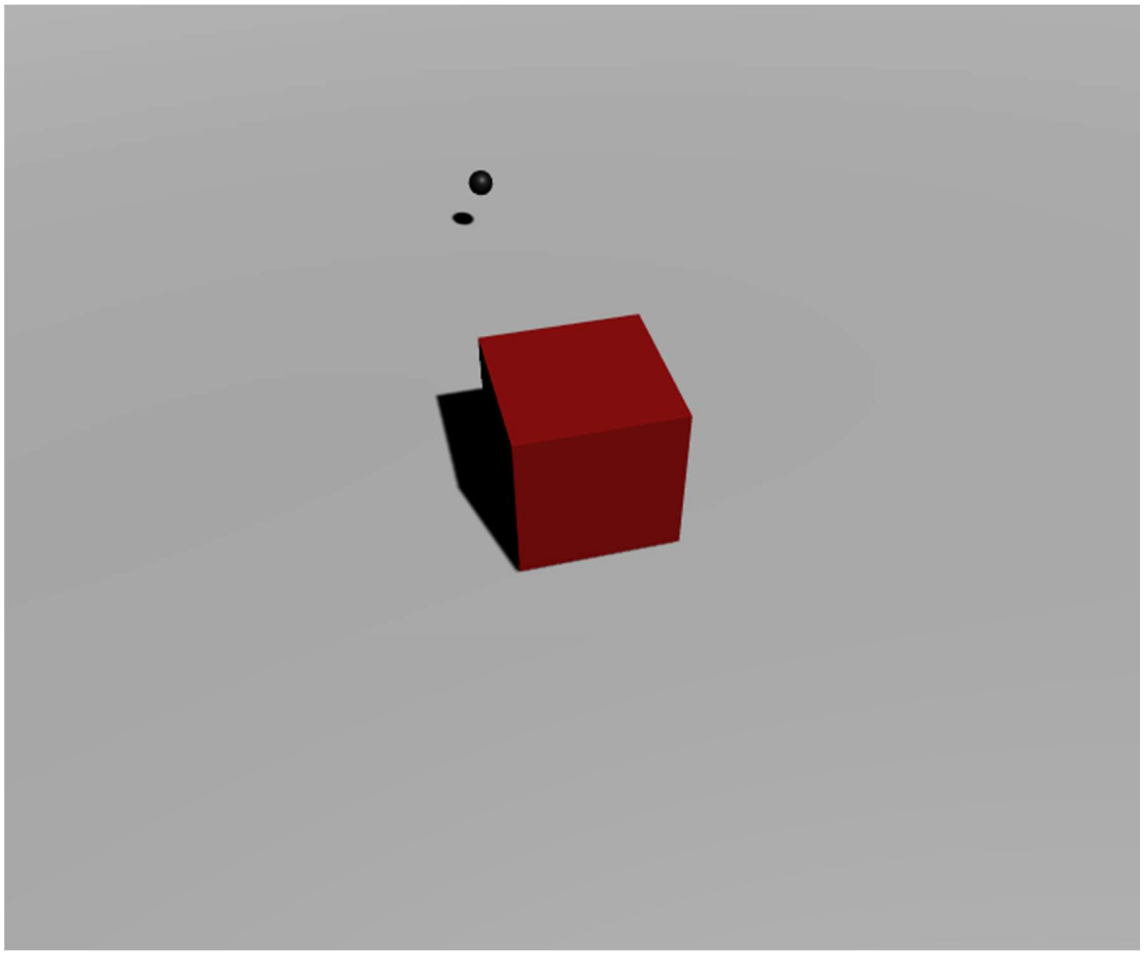
KUVIO 16 Aktivoitumis-tilanteen jälkeen laukaistaan vielä aktivoitumisen jälkeiset eventit

Tässä vaiheessa taidon tapahtumien kontrolli menee pääosin aktivoituneiden SkillModifier-objektien sekä pelimaailmaan luotujen SkillInstance-objektien käsiin. SkillModifier-objektit ovat joko suoraan yhteydessä SkillInstance-objekteihin tai sitten ne toimivat SkillEventArgs-luokan kautta. SkillInstancet suorittavat niille määriteltyä toimintoaan pelimaailmassa, esimerkiksi liikkumalla jotain

kohti, ja aktivoivat eri eventtejä tilanteiden mukaan. Esimerkiksi yksi aktivoitava eventti on nimeltään OnHit, joka laukaistaan silloin kun SkillInstance osuu johonkin pelimaailmassa olevaan asiaan. Tämä eventti voi aktivoida jonkun lisäkäyttäytymisen jostain SkillModifier-objektista.

6.3.7 Esimerkkejä

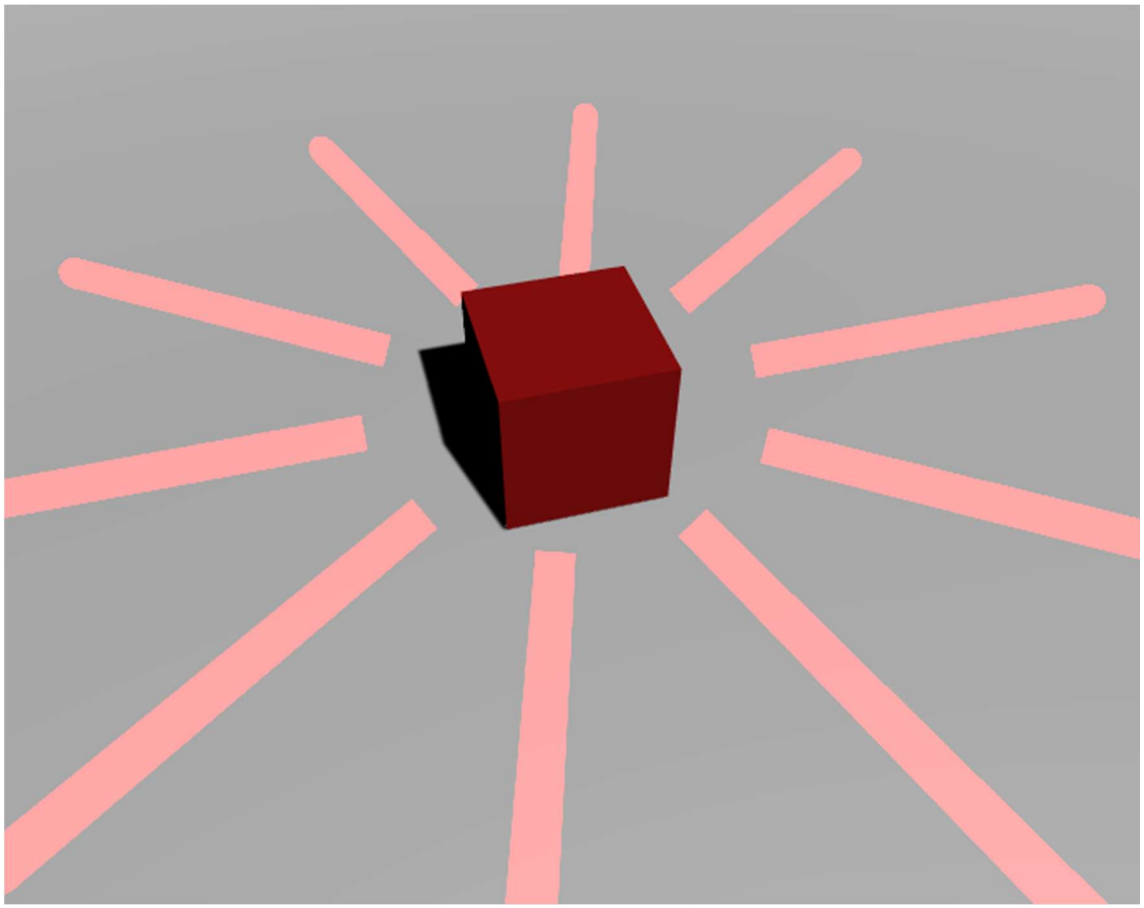
Tässä kappaleessa kerrotaan esimerkkejä SkillModifier-objektien mahdollisista muutoksista taitoihin. Taito ilman mitään sitä muokkaavaa komponenttia luo yksinkertaisen luodin, joka kulkee tasaista nopeutta eteenpäin ja tekee vahinkoa asioihin, joihin se osuu. Kuviossa 17 näytetään esimerkki tällaisesta perustaidosta. Luodissa on SkillInstance-komponentti, joka seuraa tapahtumia ja aktivoi eventtejä.



KUVIO 17 Taito, jossa on vain peruskäyttäytyminen

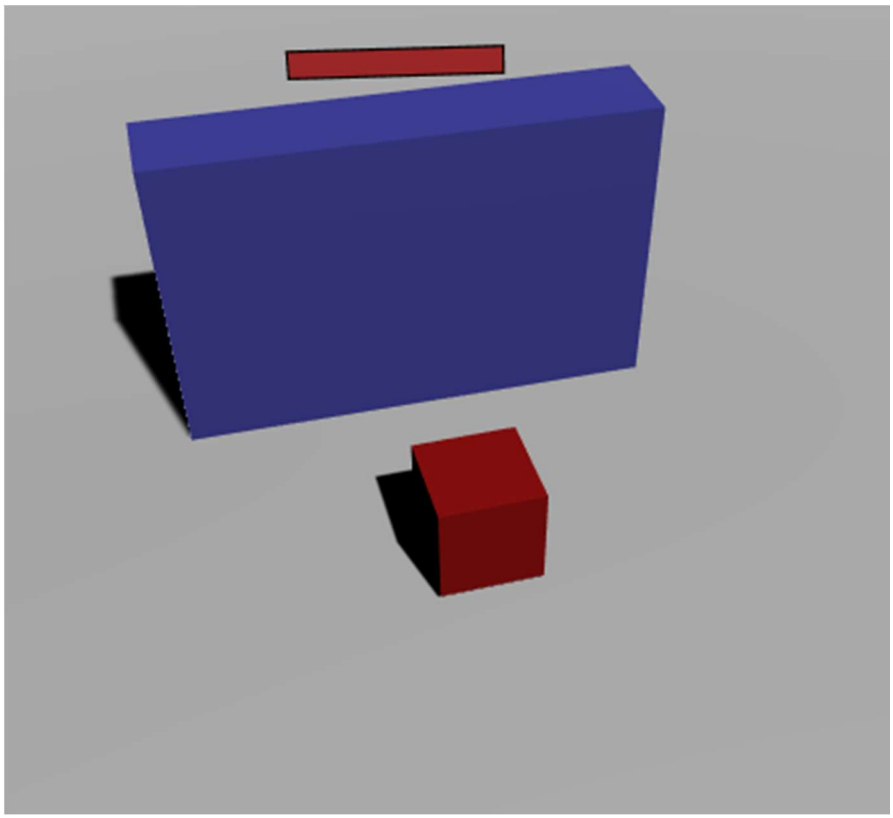
SkillModifierit, jotka ovat asetettu aktivoitumaan AfterActivate-tilanteessa, eli jälkiaktivoinnissa, pystyvät muokkaamaan aktivoinnissa luotuja objekteja. Ne voivat esimerkiksi muuttaa ilmestyneen

ammuksen sähkövirraksi, tulipalloksi tai lasersäteeksi. Ilmestyneiden ammuksien määrää, nopeutta, vahinkoa ja muuta dataakin voi vaihtaa. Kuvion 18 kuvakaappauksessa on esimerkkinä taito, joka ampuu yhden luodin sijaan 10 lasersädettä ympärilleen.

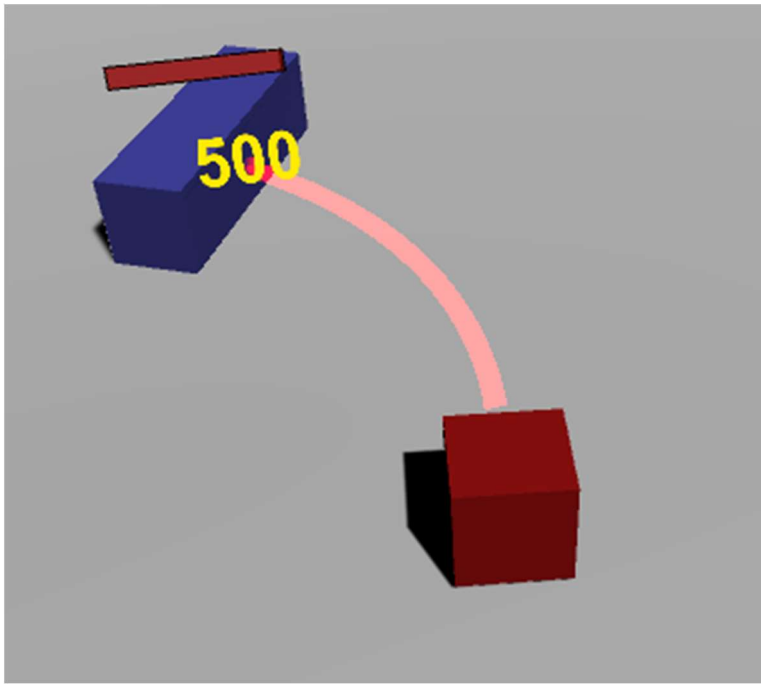


KUVIO 18 Taito, joka ampuu ympärille useita lasersäteitä

SkillModifier-objektit pystyvät myös vaihtamaan taidon ydinkäyttäytymisiä. Ne voivat esimerkiksi vaihtaa ammuksen liikkumisen kohdetta seuraavaksi, muuttua hahmoa liikuttavaksi taidoksi tai vaikka korvata luodin spawnaamisen seinäksi, kuten kuviossa 19 esitetään. SkillModifier-objekteja voi vapaasti yhdistellä haluamallaan tavalla, ja niiden muokkaukset toimivat toistensa kanssa, jos ne eivät satu täysin korvaamaan toistensa tekemiä muutoksia. Kuten kuviosta 20 näkyy, SkillModifierit ovat onnistuneesti muokanneet taidon luomaan kohdetta seuraavan lasersäteen.



KUVIO 19 Taito, joka luo seinän



KUVIO 20 Taito, joka luo kohdetta seuraavan lasersäteen

SkillModifierit pystyvät näiden lisäksi lukemattomiin muihin toimintoihin, kuten muuttamaan taidon numeroarvoja, määrittelemään ehtoja tapahtumille, aktivoimaan muita taitoja tilanteissa ja niin edelleen. EpicDraftin taitosysteemin tavoitteena ja tarkoituksena onkin alusta lähtien ollut rajaton vapaus.

6.4 Visuaalisuus

Prototyypin visuaalinen puoli on yksinkertainen, sillä kumpikaan työn tekijöistä ei ole niin keskittynyt grafiikoiden ja mallien tekemiseen. Pelissä tällä hetkellä olevat mallit ovat perusmuotoja, kuten kuutioita ja palloja. Myös taitojen ulkomuodot ovat vielä karkeita, mutta myöhemmin niistä tullaan muokkaamaan näyttävämpiä. Grafiikat ovat joko itse tehtyjä, tai ilmaisassetteja. Pelin lopullisen version ulkomuodosta ei ole vielä täysin päätetty, mutta ideana on ollut jättää se yksinkertaisen näköiseksi.

Pelin taisteluvaiheen aikana oleva käyttöliittymä tulee mahdollisesti jäämään hyvin minimaaliseksi, sillä bullet hell -tyyppisessä pelissä ei yleensä ole aikaa eikä tarvetta keskittyä mihinkään muuhun kuin asioiden väistelyyn. Tärkein käyttöliittymä pelissä tulee olemaan draftauksen käyttöliittymä. Sen suunnittelu ei ole vielä valmis, mutta sen avulla pelaajien tulee kyetä valitsemaan draftauksen aikana korttinsa ja asettamaan ne paikoilleen luodakseen taitonsa.

6.5 Online-ominaisuus

Pelin online-ominaisuutta toteutettiin käyttäen Photon Unity Networking Free -frameworkkia. Photon on Exit Games -nimisen yhtiön alustariippumaton multiplayermoottori, joka tarjoaa reaaliaikaisia verkkopeliominaisuuksia sekä chattimahdollisuudet sitä käyttäville peleille (Wikimedia Foundation, Inc., 2017, viitattu 12.12.2017). Photon Unity Networking on Unityyn integroitu framework, joka käyttää Photonia tuodakseen sen multiplayer-ominaisuudet Unity-moottorilla tehdyille peleille (Exit Games 2017, viitattu 18.12.2017).

Peliin ei ehditty saada valmiiksi täysin toimivaa online-ominaisuutta, joten se jätettiin tästä prototyypistä pois. Photonin käyttöönotto pelissä ei vie paljoa aikaa, mutta koska projektin systeemit saattavat vielä muuttua huomattavasti nykyisestään, ei multiplayer-muutoksia vielä nähty viisaaksi tehdä.

EpicDraft on kuitenkin suunniteltu ainakin kahden pelaajan multiplayer-peliksi, joten online-pelattavuus tullaan jossain vaiheessa lisäämään peliin. Koska peli on PvP bullet hell, tulee tapahtumien online-synkronointi toteuttaa huolella. Pelissä liikkuvien luotien, pelihahmojen ja muidenkin asioiden täytyy päivittyä jokaiselle pelaajalle oikein ja ajallaan. Pelin luonteen vuoksi lyhyessä ajassa ehtii tapahtua todella paljon, joten virheellisesti toteutettu multiplayer-synkronointi voi aiheuttaa epäreiluja tilanteita. Näiden asioiden pohtiminen tulee kuitenkin jäämään myöhemmäksi prosessiksi.

7 JATKOKEHITYS

Peli on opinnäytetyön valmistuessa prototyyppivaiheessa, ja sitä tullaan hyvin mahdollisesti jatkamaan ainakin siihen asti, että peliä voisi kutsua jotakuinkin valmiiksi projektiksi. Siitä on mahdollista myös jatkaa pelin oikeaan valmistumiseen saakka.

Projektin valmistuessa peliä on tarkoitus testata ihmisillä, joita kiinnostaa kyseisen genren pelit, jotta saataisiin palautetta ja kehitysehdotuksia pelistä. Peliä voi palautteen avulla jatkokehittää, ja jos pelillä on potentiaalia, niin sitten on mahdollista miettiä lisäavun etsimistä ja peliprojektin muuttamiseksi kokoaikaiseksi työksi, eikä vain pieneksi harrasteprojektiksi.

Myöhemmin valmiimmalle versiolle pelistä voisi olla hyvä myös saada pelitestaajia, mikä hyvin mahdollisesti auttaisi löytämään tärkeimmät tekniset ongelmat pelistä.

8 YHTEENVETO

Opinnäytetyön tarkoituksena oli luoda prototyyppi omasta peli-ideasta, jossa yhdistyy pelimekaniikkoja ja -genrejä, joita ei yleensä esiinny yhtä aikaa yhdessä pelissä.

Pelin tekemiseen valittiin Unity-pelimoottori, koska sen käyttämisestä on kummallakin meillä tietoa ja kokemusta. Ohjelmointikieleksi valittiin C# samoista syistä, ja myös koska C# on yleisempi ja suositumpi Unityn pelinkehittäjien keskuudessa kuin JavaScript.

Työn tavoitteisiin kuului tekniseltä näkökannalta luoda dynaaminen, melkein kaiken mahdollistava taitojen yhdistelysysteemi. Vaikka tällaisen systeemin suunnittelu ja toteuttaminen päätyi olemaan mukavaa, vaikkakin haastavaa, olisi mahdollisesti enemmän rajattu järjestelmä ollut sopivampi juuri tätä ideaa varten.

Työn aihe oli kummallekin meistä kiinnostava, sillä toteutimme omaa ideaa. Pelin idea kuitenkin päätyi muuttumaan laajemmaksi kuin mitä sen olisi kannattanut tätä työtä varten tehdä. Pelissä oli useita isoja kokonaisuuksia toteuttamishaasteineen, ja niiden kaikkien saaminen toimiviksi prototyyppiin muuttui hankalaksi.

Opinnäytetyön aikataulutus ei ollut hyvin toteutettu. Työ aloitettiin hyvin ajallaan, mutta edistyminen päätyi hyvin kauan ajan olemaan hidasta. Lopussa työn raportointivaiheeseen ei jäänyt paljoa aikaa, ja pelin prototyyppiinkin jäi tekemättä idealle keskeisiä ominaisuuksia.

LÄHTEET

A+E Networks, 2017. Video Game History. Viitattu 18.12.2017
http://www.history.com/topics/history-of-video-games

Exit Games, 2017. Introduction | Photon Engine. Viitattu 18.12.2017
https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro

Dice. How Unity3D Became a Game-Development Beast, 2013. Viitattu 18.12.2017
https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

Fandom, 2016. Shoot 'em up video games. Viitattu 19.12.2017
http://gaming.wikia.com/wiki/Shoot_%27em_up_video_games

International Student. What is Game Design, 2017. Viitattu 18.12.2017
https://www.internationalstudent.com/study-game-design/what-is-game-design/

Make A Game Of That, 2013. Mechanics: Drafting. Viitattu 14.12.2017
https://makeagameofthat.wordpress.com/2013/03/22/mechanics-drafting/

Microsoft Corporation, 2015. Delegates (C# Programming Guide). Viitattu 18.12.2017
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/

Microsoft Corporation, 2015. Events (C# Programming Guide). Viitattu 18.12.2017
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/

Newzoo, 2017. THE GLOBAL GAMES MARKET WILL REACH $108.9 BILLION IN 2017 WITH MOBILE TAKING 42%. Viitattu 19.12.2017
https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/

Nystrom, R, 2014. Object Pool. Viitattu 18.12.2017
http://gameprogrammingpatterns.com/object-pool.html

Pluralsight, 2014. Creating a Game Concept: The First Step in Getting your Game off the Ground. Viitattu 18.12.2017
https://www.pluralsight.com/blog/film-games/creating-game-concept-first-step-getting-game-ground

Unity3D. The Leading global game industry software, 2017. Viitattu 18.12.2017
https://unity3d.com/public-relations

Wikimedia Foundation, Inc., 2017. Exit Games. Viitattu 12.12.2017
https://en.wikipedia.org/wiki/Exit_Games

Wikimedia Foundation, Inc., 2017. Shoot 'em up. Viitattu 14.12.2017
https://en.wikipedia.org/wiki/Shoot_%27em_up

Wikimedia Foundation, Inc., 2017. Unity (game engine). Viitattu 18.12.2017
https://en.wikipedia.org/wiki/Unity_(game_engine)

Wikimedia Foundation, Inc., 2017. Video game development. Viitattu 18.12.2017
https://en.wikipedia.org/wiki/Video_game_development