Tampereen ammattikorkeakoulu

# Uudelleenkäytettävän taistelupelihahmon toteutus Unityssä

Petteri Koivuniemi

OPINNÄYTETYÖ
Huhtikuu 2019

Tietojenkäsittely
Web-palvelut

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Web-palvelut

KOIVUNIEMI, PETTERI:
Uudelleenkäytettävän taistelupelihahmon toteutus Unityssä

Opinnäytetyö 37 sivua
Toukokuu 2019

---

Tässä opinnäytetyössä selvitettiin taistelupelien yleisiä ominaisuuksia ja perehdyttiin niiden toteuttamiseen Unity-pelimoottorissa sekä syvennettiin tekijän henkilökohtaista osaamista Unity-kehittäjänä. Työn tarkoituksena oli esitellä taistelupelihahmojen yleisiä toimintoja ja niiden variaatioita, käydä läpi keinoja miellyttävän pelikokemuksen rakentamiseen ja lopulta toteuttaa yksinkertainen taistelupelihahmo Unityssä. Työhön kerättiin tietoa muun muassa Unityn dokumentaatiosta, pelialan ammattilaisten blogeista ja videoista sekä aihepiiriä käsittelevistä artikkeleista ja kirjoista.

Työn osana toteutettiin yksinkertainen taistelupelihahmo Unityssä. Pelimoottorin Mecanim-animaatiotilakonetta hyödynnettiin hahmon ohjaamiseen yhdessä hahmon pohjaluokan ja usean muun C#-skriptikomponentin kanssa. Työn olennaisimmat piirteet kuvaillaan opinnäytetyössä vaiheittain. Työtä varten mallinnettiin ja animoitiin yksinkertainen humanoidihahmo Blender-ohjelmalla.

Opinnäytetyössä havaittiin, että monet taistelupelihahmojen toiminnallisuudet kytkeytyvät tiiviisti hahmoanimaatioihin. Unityn animation event -funktiokutsujen ja Mecanimin avulla hahmon toiminnallisuuksien rakentamiseen vaadittu ohjelmointityö on suhteellisen yksinkertaista. Huomioita pidettiin yllättävinä.

Unityn kattavilta dokumentaatiosivuilta löytyi vastaus lähes kaikkiin työssä kohdattuihin ohjelmointiongelmiin. Taistelupelien terminologian pelaajalähtöisyyden vuoksi hahmojen toiminnallisuutta käsittelevän, tieteellisesti validin lähdeaineiston hakeminen koettiin haasteelliseksi.

---

Asiasanat: taistelupelihahmo, unity, mecanim, C#, animaatio

## ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Web Services

KOIVUNIEMI, PETTERI:
Developing a reusable fighting game character with Unity

Bachelor's thesis 37 pages
May 2019

---

The objective of this thesis was to gather information about the common features of fighting game characters, and to describe the implementation of such features in the Unity game engine. These topics were examined in part through the construction of an original, simplified fighting game character.

The Unity engine's integrated animation state machine, Mecanim, was used in controlling the general state and actions available to the character in tandem with several C# script components. A simple humanoid representation was also modeled and animated for the character using Blender. A step by step description of building the characters' functionality is included in this thesis.

The author was surprised by the close-knit relationship between animations and combat functionalities in fighting game characters, as well as the relative simplicity of the programming required for creating a working fighter. Solutions for almost all encountered problems in creating the original character were found in the official Unity documentation. Finding scientifically valid sources proved challenging, due to much of the fighting game related terminology being conceived and spread informally among anonymous forum posters.

---

Key words: fighting game character, unity, mecanim, C#, animation

**SISÄLLYS**

**ERITYISSANASTO**

| | |
|---|---|
| Blockbox | Torjunnassa käytetty törmäystarkastelukomponentti |
| Frame | Animaatiokehys, yksi ruudunpäivityskierros |
| Frame data | Hyökkäysten ominaisuuksien ja animaatioiden suhde toimintojen tasapainottamisessa |
| Hitbox | Hyökkäyksen aktiivinen, vahinkoa aiheuttava törmäystarkastelukomponentti |
| Hurtbox | Hahmon vahingoittuva alue, Hitboxin vastakappale |
| Mecanim | Unityn sisäänrakennettu animaatiotilakone |
| Prefab | Muovattava peliobjektien ja komponenttien kokonaisuus |
| Scene | Pelin tasonäkymä Unity-pelimoottorissa |

# 1 JOHDANTO

Pelihallipeleinä 80- ja 90-luvuilla läpimurtonsa tehneet taistelupelit nauttivat yhä peliharrastajien suosiosta (Gilyadov 2015). Kilpahenkinen pelityyppi on muodostunut sekä pelaajien että katsojien suosimaksi e-urheilumuodoksi. Tuoreena esimerkkinä tästä toimi vuoden 2018 Evolution Championship Series -taistelupeliturnaus Yhdysvalloissa, joka keräsi yhteensä 5.2 miljoonaa katsontatuntia Twitch-suoratoistopalvelussa sekä Youtubessa (Elliot 2018).

Tämän opinnäytetyön tavoitteena on selvittää taistelupelien yleisiä ominaisuuksia ja perehtyä niiden toteuttamiseen Unity-pelimoottorissa, sekä syventää tekijän henkilökohtaista ymmärrystä ja osaamista Unity-kehittäjänä. Työn tarkoituksena on esitellä taistelupelihahmojen yleisiä toimintoja ja niiden variaatioita, sekä käydä läpi keinoja miellyttävän pelikokemuksen rakentamiseen. Opinnäytetyön osana toteutettiin yksinkertainen taistelupelihahmo Unityssä. Toteutetun hahmon ominaisuudet yhdistelevät piirteitä taistelu- ja tasoloikkapeleistä. Unityn Mecanim-animaatiotilakonetta hyödynnetään hahmon ohjauksessa yhdessä useiden C#-skriptikomponenttien kanssa. Ohjelmointityön lisäksi toteutukseen sisältyy hahmon 3d-mallin rakentaminen ja animoiminen Blender-ohjelmalla.

Projektin painotus on hahmon ominaisuuksissa. Toteutuksessa keskitytään toimivaan liikkumiseen ja lähitaistelumekaniikkoihin. Taistelupelihahmolla on käytettävissään useita hyökkäyksiä, torjunta ja väistö. Tämän lisäksi hahmolla on tuplahyppy tasoloikkapelien hengessä. Tuotosta varten mallinnettiin ja animoitiin yksinkertainen ihmishahmo. Koko työn toteuttamiseen käytettiin Unityä ja sen Mecanim-animaatiotilakonetta, Blenderiä ja C#-ohjelmointikieltä. Työn tavoitteena ei ollut kokonainen taistelupeli, minkä vuoksi monet valmiin pelin kannalta olennaiset piirteet, kuten valikot ja vihollistekoäly, eivät sisälly toteutukseen.

# 2 TIETOA TAISTELUPELEISTÄ

## 2.1 Taistelupelien nousu

Vuonna 1991 julkaistua Street Fighter II:ta pidetään maailmanlaajuisen taistelupelibuumin aloittajana sekä suunnannäyttäjänä myöhemmille taistelupeleille (Gilyadov 2015). Pelin idea on yksinkertainen: kaksi taistelijaa ottaa toisistaan mittaa nyrkein, potkuin sekä taikuuden tai muiden erikoisvoimien avulla (kuva 1). Pelaaja, joka saa ensimmäisenä kulutettua toisen pelaajan elinpisteet loppuun tai jolla on enemmän elinpisteitä jäljellä ajan loppuessa, voittaa erän ja ensin kaksi erää voittanut pelaaja voittaa pelin. (Street Fighter II Turbo Instruction Manual n.d. s.10.)


KUVA 1. Street Fighter 2 (Maxim 2017)

Kaksiulotteisuus Street Fighter II:ssa pätee grafiikan lisäksi myös pelimekaniikkaan, eli toiminta tapahtuu vain pysty- ja sivuttaissuunnissa. Street Fighter sekä monet muut taistelupelisarjat ovat myöhemmissä iteraatioissaan siirtyneet 3D-grafiikkaan, mutta pelimekaanisesti ne toimivat yhä kaksiulotteisesti. Tästä syystä niitä nimitetään edelleen 2D-taistelupeleiksi. (Core-A Gaming 2017.)

90-luvulla pelit, kuten Virtua Fighter, Battle Arena Toshinden ja sittemmin SoulCalibur-nimellä tunnetun sarjan Soul Edge, uudistivat tuttua kaavaa lisäämällä yhtälöön kolmannen ulottuvuuden. Kolmiulotteisten hahmomallien ohella pelimekaniikkaa laajennettiin lisäämällä pelaajille kyky liikkua syvyysakselilla. Tämä näennäisen pieni uudistus johti uuden taistelupelien alalajin, 3D-taistelupelien, syntyyn. (Jensen 2016.)

2D-taistelupelien kirjoa laajensi ja uudisti 1999 julkaistu Super Smash Bros., joka toi suosittuja pelihahmoja useista Nintendon pelisarjoista yhteen taistelupelimuodossa (Gilyadov 2015). Super Smash Bros. erosi pelattavuudeltaan huomattavasti tyypillisistä taistelupeleistä ja sen esittelemää tasoloikan ja taistelun yhdistelmää jäljitteleviä pelejä on myöhemmin alettu kutsua osuvasti platform fighter -alagenren edustajiksi.

## 2.2 Yleisiä eroavaisuuksia

Eräs taistelupelin tärkeimmistä piirteistä on liikkumiseen käytössä olevien akseleiden määrä. Päällisin puolin samalta näyttävät taistelupelit voivat olla akseleiden määrästä riippuen toiminnallisesti hyvinkin erilaiset. Esimerkiksi 2D-taistelupeleissä hypyn tai torjunnan pakottavat ammushyökkäykset ovat pienemmässä roolissa kolmannen ulottuvuuden salliessa niiden reitiltä sivuun astumisen. (Core-A Gaming 2017.)

Hahmojen lisäksi myös taistelukentissä on merkittäviä eroja 2D- ja 3D-taistelupelien välillä. Perinteisille kaksiulotteisille taistelupeleille on Street Fighteria mukaillen ominaista rajata pelialue molemmilta puolilta loputtoman korkeilla seinillä. Kolmiulotteisissa taistelupeleissä seinillä voi olla vaihtelevia ominaisuuksia: jotkut seinät hajoavat ja voivat hajotessaan johtaa siirtymään eri osaan kenttää. Toisinaan kentissä ei ole seiniä ollenkaan, vaan taistelu tapahtuu loputtomalla tasangolla. Virtua Fighter esitteli uutena ominaisuutena rajalliset pelialueet ilman seiniä, mikä mahdollisti erän voittamisen vihollisen elinpisteiden ehdyttämisen lisäksi heittämällä tämän kentän reunan yli (1UP 2012).

Super Smash Bros. -sarjan pelit poikkesivat taistelupelien perinteisistä kaavoista tehden vihollisen kentältä ulos heittämisestä pelin pääasiallisen voittoehdon. Hahmon elämäpisteitä kuvataan palkin sijasta prosenttilukemalla. Mitä enemmän iskuja ottaa vastaan, sitä korkeammaksi lukema muuttuu ja sitä enemmän hahmo lentää iskuista, mikä tekee kentällä pysymisestä jatkuvasti vaikeampaa. (Burgun 2012, 120.)

## 2.3 Hahmon yleiset ominaisuudet

### 2.3.1 Liikkuminen ja kamera

Kameralla ja pelihahmojen liikkeellä on olennainen suhde taistelupeleissä, sillä kaikkien pelihahmojen tulee näkyä ruudulla jatkuvasti. Taistelupeleille tyypillisin tapa toteuttaa kamera on sijoittaa se toiselle sivulle kasvotusten oleviin pelaajiin nähden niin, että molemmat näkyvät sivuprofiilista kuvaruudulla. Kamera seuraa pelaajien liikettä ja voi siirtyä lähemmäs tai kauemmas pitääkseen kaikki hahmot kuvassa jos näiden etäisyys toisistaan kasvaa tai pienenee.

Poikkeavat kameraratkaisut ovat harvinaisempia, mutta niitäkin esiintyy. Esimerkiksi Nintendon WiiU- ja Switch -konsoleille julkaistussa Pokkén Tournamentissa kamera on hieman toisen, kamerasta poispäin katsovan hahmon takana. Toinen hahmo sijoittuu etäämmälle kamerasta kasvot vastustajaa ja samalla jokseenkin kameraa kohti (kuva 2).

KUVA 2: Pokkén Tournament DX (Nintendo 2017)

Hahmon ohjaaminen Tekkenissä (Tekken 7: Guides n.d.) ja Street Fighterissa (Street Fighter 5: How to Play n.d.) on suhteellisen samankaltaista: kameraan nähden sivusuuntainen liike on hidasta, mutta sallii nopeat syöksähdykset eteen- ja taaksepäin. Poispäin vihollisesta liikkuminen ja kyykkyyn meneminen aktivoivat ylä- ja alatorjunnan. Hahmoilla on käytettävissään yksittäinen hyppy. Hahmot myös pyrkivät katsomaan toisiaan kohti automaattisesti. Suurin osa taistelupeleistä toteuttaa pelaajan liikkumisen tätä kaavaa iteroiden.

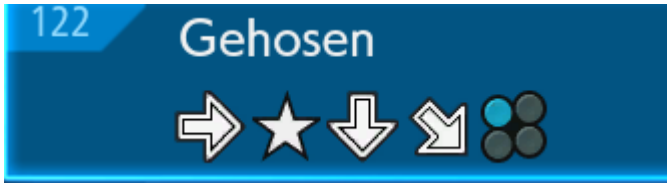
Monet taistelupelit sallivat liikkeiden keskeyttämisen tarkasti ajoitetuilla syötteillä tai jotain pelihahmon resurssia käyttämällä. Esimerkiksi useissa Tekken-sarjan peleissä taaksepäin suuntautuvan syöksähdyksen pystyy keskeyttämään, minkä seurauksena hahmo voi välittömästi suorittaa uuden toiminnon, vaikkapa uuden syöksyn (Shoryuken 2017). Taitava pelaaja voi pystyä liikkumaan hyvinkin nopeasti, aloittelija ei välttämättä niinkään.

Platform fighter -tyyppisissä taistelupeleissä liikkuminen on lähempänä tasohyppelypelejä kuin perinteisten taistelupelien mallia. Vihollisen suuntaan lukittumisen sijaan pelihahmon kasvot kääntyvät annetun liikesyötteen suuntaan ja liikkeen nopeus riippuu siitä, kuinka paljon ohjaustikkua kallistaa.

### 2.3.2 Hyökkääminen

Monesti taistelupeleissä nähdään sekä yksinkertaisilla syötteillä tapahtuvia heikkoja iskuja, että vaihtelevan kompleksisten syöteyhdistelmien taakse lukittuja tehokkaampia hyökkäyksiä. Lyönneille ja potkuille voi olla omat nappinsa. Mikäli pelissä on heittoja, tapahtuvat ne tyypillisesti painamalla joitakin kahta hyökkäysnappia samanaikaisesti, toisinaan yhdistelmänä liikesyötteen kanssa.

Paljon vauriota tekevät tai muuten erityisen hyödylliset liikkeet vaativat tyypillisesti sarjan syötteitä ohjaustikulla sekä jonkun hyökkäyssyötteen (kuva 3). Monimutkaisiin syöteyhdistelmiin hyödynnetään syötepuskuria (input buffer), eli vastaanotetut syötteet pidetään tallessa lyhyen aikaa ja tarkastellaan, täyttyvätkö jonkun hyökkäyksen vaatimukset (Dieterich 2011). Tämä mahdollistaa monimutkaisen syötteen suorittamisen jonkin toisen hyökkäyksen aikana, johtaen oikein suoritettuna sulavaan vaihdokseen hyökkäyksestä toiseen.




KUVA 3. Iskun suorittamiseen vaaditut syötteet, Tekken 7. Kuvakaappaus.

### 2.3.3 Hahmoarkkityypit

Taistelupelien hahmot mukailevat useimmiten yleisiä pelityylejä edustavia arkkityyppejä. Jotkut arkkityypit saattavat olla jonkin tietyn pelin tai pelisarjan sisäisiä, toiset taas voivat esiintyä missä tahansa taistelupelissä.

Arkkityyppehin liittyvä terminologia on suurelta osin pelaajalähtöistä eikä erityisen hyvin standardisoitua. Toisinaan samalla arkkityypillä esiintyy eri nimiä, esimerkiksi kahdesta hahmosta koostuvia taistelijoita voidaan nimittää duoiksi tai nukkemestareiksi (puppet master) riippuen pelistä ja puhujasta. Esimerkkejä yleisistä taistelupeleissä esiintyvistä hahmoarkkityypeistä ovat heittoliikepainotteiset painijahahmot, ammushyökkäyksillä vahinkoa etäisyyden

päästä tekevät hahmot, aggressiiviseen ja hyökkäyspainotteiseen pelityyliin sopivat mutta heikosti puolustautuvat hahmot sekä tasapainoiset hahmot, joilla on kaikkien edeltävien tyyppien ominaisuuksia mutta eivät ole minkään pelityylin voimakkaimpia edustajia. (Reddit 2018.)

### 2.3.4 Torjunta

Brownin (2018) mukaan hyvästä taistelusysteemistä löytyy tasapainoisesti sekä hyökkäämistä että puolustautumista, kummankaan olematta kiistattomasti parempi vaihtoehto joka tilanteessa. Heitot, iskut ja torjunta muodostavat taistelupelien perinteisen, lähes kivi-paperi-sakset-tyyppisen kokonaisuuden: torjunta päihittää iskut mutta häviää heitoille, vihollisen heitot taas voi tyypillisesti purkaa painamalla jotain hyökkäysnappia oikealla hetkellä.

Torjunnasta on taistelupeleissä usein kaksi variaatiota, torjunta seisaaltaan joka estää iskut päähän ja keskivartaloon mutta jättää jalat suojaamatta, sekä päinvastoin toimiva alatorjunta. Joissain taistelupeleissä pystyy torjumaan myös hypyn aikana. Äärimmäisyyksiin yksinkertaistetussa Divekickissä puolestaan ei poikkeuksellisesti ole torjuntaa tai erillistä liikkumista ollenkaan; peliä pelataan nimensä mukaisesti vain hyppy- ja potkunappia käyttäen.

Jotkut taistelupelit sallivat torjunnan hajoittamisen toistuvilla iskuilla, minkä vuoksi iskuja on mahdollista torjua vain rajallinen määrä kerrallaan. Torjunnan särkyessä hahmo on useimmiten hetken puolustuskyvytön palkiten aggressiivisen hyökkääjän. Vastapainoksi hyvin ajoitettu torjunta ei heikkene vastaanotetusta iskusta ja voi jopa luoda puolustajalle tilaisuuden siirtyä hyökkäyskannalle. Guilty Gear -sarjan peleissä torjunnan hajoaminen toimii hieman eri tavalla: hahmo vastaanottaa toistuvien torjuntojen läpi yhä enemmän vahinkoa, mutta torjuva pelaaja ei menetä täysin kontrollia hahmostaan. Taulukossa 1 esitellään muutamien taistelupelien ratkaisuja torjuntaan.

TAULUKKO 1. Eroavaisuuksia taistelupelien torjuntaominaisuuksissa

| Peli | Torjunta | Erillinen alatorjunta | Torjunta ilmassa | Torjunnan hajoittaminen |
|---|---|---|---|---|
| Tekken 7 | Kyllä | Kyllä | Ei | Ei |
| Street Fighter 5 | Kyllä | Kyllä | Ei | Kyllä (tietyt liikkeet) |
| SoulCalibur 6 | Kyllä | Kyllä | Ei | Kyllä |
| Super Smash Bros. Ultimate | Kyllä | Ei | Ei | Kyllä |
| Divekick | Ei | Ei | Ei | Ei |
| Guilty Gear Xrd | Kyllä | Kyllä | Kyllä | Kyllä |

Super Smash Bros. -sarjan peleissä torjunta on visualisoitu läpikuultavana kuplana hahmon ympärillä (kuva 3). Kupla pienenee vastaanotetuista iskuista ja hajoaa liian monen torjutun iskun tai liian pitkän aktiivisen jakson jälkeen, palautuen kohti alkuperäistä kokoaan, kun torjunta ei ole aktiivinen. Ajoitetulla ”täydellisellä torjunnalla” kilpi ei vaurioidu ja torjuja pystyy suorittamaan uusia toimintoja ennen hyökkääjää. Torjuntakuplaa on mahdollista liikuttaa hieman eri suuntiin, mutta erillistä ylä- ja alatorjuntaa ei Smash Bros. -sarjan peleistä löydy. Sivusuuntaisella liikesyötteellä torjunnan aikana hahmo tekee annettuun suuntaan kuperkeikan, minkä aikana tätä ei voi vahingoittaa. Vastaava liikesyöte alaspäin taas johtaa paikallaan tehtävään väistöliikkeeseen.


KUVA 3. Torjunta Super Smash Bros. Ultimatessa (AOTF 2018)

Syksyllä 2018 julkaistu SoulCalibur VI tarjoaa pelaajalle laajan valikoiman puolustusvaihtoehtoja. Torjunnalle on Smash Brosin tapaan oma nappinsa ja samaan tyyliin torjunta voi hajota toistuvista osumista. SoulCalibur kuitenkin yhdistää puolustukseen myös perinteisempiä ominaisuuksia kuten erillisen ylä- ja alatorjunnan. Lisäksi pelaajalla on käytettävissään kaksi pelisarjalle uniikkia ajoitetun torjunnan muotoa, ”Guard Impact” ja ”Reversal Edge,” joilla taistelun rytmin voi kääntää puolustautujalle edulliseksi (kuva 4).


KUVA 4. Vihreä välähdys kertoo onnistuneesta Guard Impact -torjunnasta, SoulCalibur VI. Kuvakaappaus.

### 2.3.5 Mittariliikkeet ja paluumekaniikat

Monet taistelupelit sitovat hahmojen toimintoja erilliseen resurssiin tai mittariin. Mittari täyttyy esimerkiksi tekemällä tai vastaanottamalla vahinkoa ja eri liikkeiden käyttö voi maksaa vaihtelevan määrän resurssia. Näin voidaan helposti varmistaa, että mikään yksittäinen liike ei tee muita liikkeitä turhiksi.

Hieman harvinaisemmin koko hahmokaartin sijasta yksittäisillä hahmoilla voi esiintyä mittariin sidottuja liikkeitä. Useimmiten kyseessä on jostakin muusta pelistä tuttu vieraileva hahmo, millä on omassa pelissään mittaririippuvaisia ominaisuuksia. Tällaisia hahmoja ovat esimerkiksi Street Fighterin Akuma Tekken 7:ssä ja Final Fantasy VII:n Cloud Super Smash Bros. Ultimatessa.

Taistelupeleissä on alkanut yleistyä häviöllä olevan pelaajan tehostaminen tavalla tai toisella eli niin kutsutut paluumekaniikat (kuva 5). Hahmo saattaa mekaniikan ehtojen täyttyessä saada käyttöönsä uusia liikkeitä tai tehdä hyökkäyksillään enemmän vahinkoa. Paluumekaniikkojen perimmäinen tarkoitus on tehdä taistelupeleistä aloittelijaystävällisempiä. Aiheesta on käyty paljon keskustelua muun muassa Redditissä ja Shoryukenin foorumeilla, hyödyttävätkö tällaiset ominaisuudet kuitenkin enemmän kokeneempia pelaajia ja onko pelaajan palkitseminen häviöllä olemisesta mielekästä.


KUVA 5. Hahmon punainen hehku viestii tehokkaiden paluuliikkeiden olevan käytettävissä, Tekken 7. Kuvakaappaus.

# 3 POSITIIVISEN PELIKOKEMUKSEN RAKENTAMINEN

## 3.1 Responsiivisuus

Hahmon toimiva ohjattavuus on kriittisen tärkeää pelin nautinnollisuudelle. Brownin (2018) mukaan pelaajan huomio kiinnittyy peliä pelatessa hahmon ohjattavuuteen vain silloin, kun siinä on jotain vikaa. Toiminnan välittömyys on olennaista responsiivisuuden tunteen luomisessa, eli pelihahmon tulee vastata viiveettä pelaajan syötteisiin ja hahmon suorittaman toiminnon tulee myös johdonmukaisesti vastata pelaajan odotuksia.

Pignole (2014) käy blogikirjoituksessaan läpi erilaisia keinoja miellyttävän ohjaussysteemin rakentamiseksi tasohyppelypelihahmolle: esimerkiksi maata kohti putoavan pelaajan lievästi ennakoitu hyppysyöte voidaan pitää muistissa ja rekisteröidä jälkikäteen, jotta tälle ei synny tunnetta hyppynapin puutteellisesta toiminnasta. Pignolen mukaan pelaajan täytyy ohjaimeen koskemattakin pystyä tiedostamaan, mikäli hahmon käytettävissä olevat kyvyt ovat rajallisemmat tai laajemmat hypyn aikana kuin maan pinnalla.

Animaatiot ja ohjattavuus kulkevat monesti käsi kädessä. Hahmon ohjauksen osittainen tai täysvaltainen lukittaminen käynnissä olevan animaation aikana on toimiva ratkaisu, jos taisteluun haetaan taktisuuden tuntua. Pelaajan syötteellä tapahtuva animaation keskeyttäminen ja siitä seuraava liikkeen vapaus puolestaan mahdollistaa suuremmat määrät mahdollisesti aggressiivisempia vihollisia ilman, että peli tuntuu epäreilulta. (Brown 2018.)

## 3.2 Audiovisuaalinen palaute

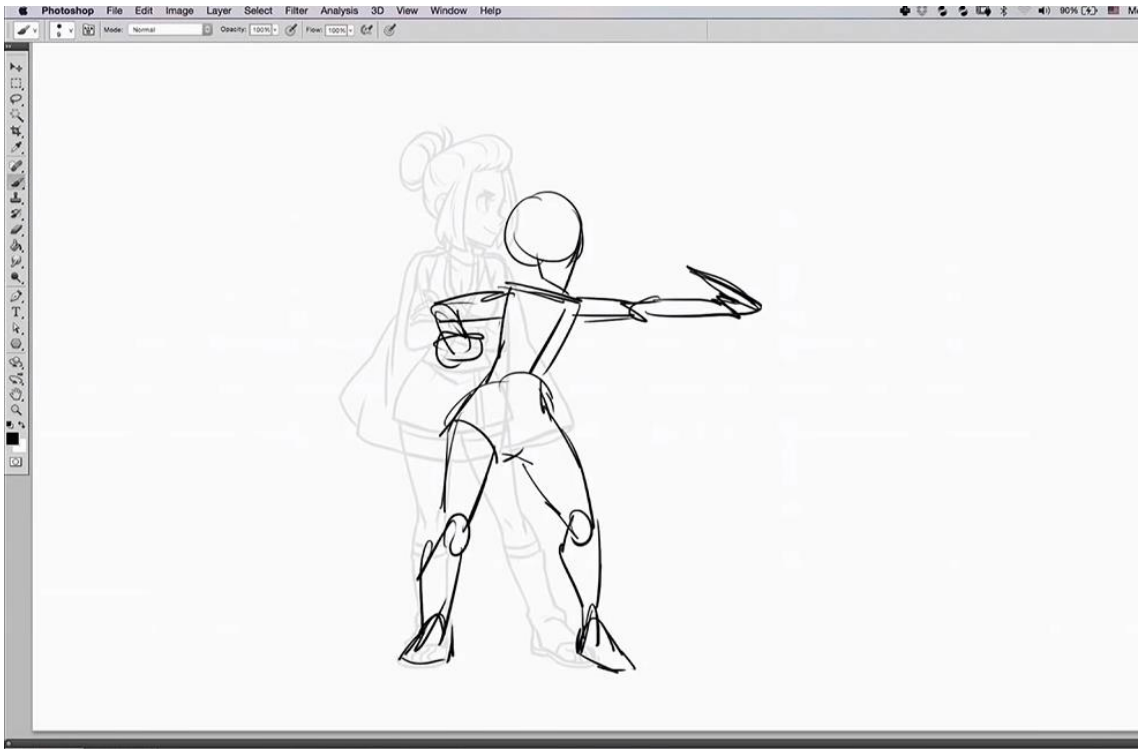
Turhautuneisuuden välttämiseksi pelaajan tulee pystyä luontevasti ymmärtämään syy-seuraus-suhteet pelin tapahtumien välillä. Pelin toimintaa pystyy selkeyttämään esimerkiksi maltillisella ääniefektien ja graafisten käyttöliittymäelementtien tai muiden visuaalisten vihjeiden avulla.

Hahmoanimaatiot ovat olennainen osa pelien visuaalista palautetta. Cartwright (2014) painottaa GDC-puheessaan taistelupelihahmojen animoinnista hahmon siluetin merkitystä: animaatioiden tulee olla helposti erotettavissa ympäristöstä ja viestiä selkeästi, mitä hahmo milloinkin tekee. Useiden Devil May Cry -toimintapelien ohjaajana toiminut Hideaki Itsuno puhuu siluetin käyttämisestä myös hahmon kykyjen viestimisessä pelaajalle, esimerkkinään useat kyyryselkäiset hahmot Devil May Cry 5:stä joilla on yhteisiä liikkeitä ja ominaisuuksia (Itsuno 2019).

Ammattilaisanimoija Daniel Floyd puhuu Youtube-videossaan ”How To Animate a Smash Bros Attack // LINK – New Frame Plus” (2018) animoinnin perusteista ja hyvän hyökkäysanimaation rakenteesta. Floydin mukaan hyökkäysanimaatiot tulisi jakaa kolmeen vaiheeseen: ”anticipation”, ”impact” ja ”recovery”, vapaasti kääntäen valmistautuminen, osuma ja palautus.

Valmistautumisen tarkoitus on viestiä hyökkääjälle pelin vastaanottaneen hänen syötteensä, samalla vihjaten vihollispelaajalle hyökkääjän aikeista. Lyhytkin odotusjakso animaation alussa parantaa hyökkäysanimaation luettavuutta merkittävästi (Cartwright 2014).

Osuma on hyökkäyksen aktiivinen, vahinkoa tekevä osa. Aktiivisuus kestää tyypillisesti vain lyhyen hetken, minkä aikana animaatiossa tapahtuu siirros valmiusasennosta suoritettuun iskuun. Iskun voimakkuuden vaikutelmaa osuman aikana voidaan lisätä animaation keinoin esimerkiksi sen liikerataa seuraavilla tahratehosteilla ja hahmon anatomian rikkomisella, vaikkapa tämän raajoja venyttämällä yksittäisissä animaatiokehyksissä (kuva 6).




KUVA 6. Kuvakaappaus hahmoanimoinnin live-esitelmästä (Cartwright 2015, muokattu)

Hyökkäyksen päättävä palautus on olennainen osa hyökkäyksen voiman viestimistä ja tyypillisesti animaation pisin vaihe. Hyökkääjä on palautuksen aikana puolustuskyvytön. (Floyd 2018.)

Vastapainoisesti hahmon tulee myös reagoida vastaanottamiinsa iskuihin mielekkäällä tavalla. Esimerkiksi heikko isku voi aiheuttaa pienen horjahduksen vahvemman heittäessä vastaanottajaa taaksepäin avaten tämän jatkohyökkäyksille. Osumien yhteydessä puolin ja toisin, hahmon hypätessä ja laskeutuessa sekä esimerkiksi ajoitetun täydellisen torjunnan seurauksena voi esiintyä partikkelitehosteita ja ääniefektejä toiminnan dramatisoimiseksi.

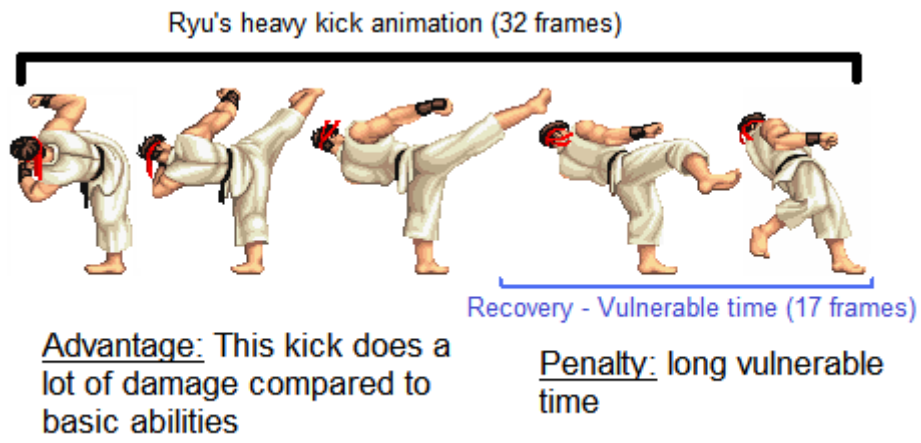
### 3.3 Haptinen palaute

Audiovisuaalisen palautteen lisäksi peleissä hyödynnetään usein haptista eli kosketusaistillista palautetta. Peliohjaimen tai puhelimen värisyttäminen jonkin sopivan pelimaailman tapahtuman seurauksena auttaa immersiivisen eli todellisuuden tuntuisen pelikokemuksen luomisessa.

Yleisiä käyttötarkoituksia ovat esimerkiksi räjähdysefektien tai pelaajan vastaanottamien iskujen voimakkuuden viestiminen. Ohjaimen värinää voidaan kuitenkin hyödyntää muihinkin tarkoituksiin. Esimerkiksi VR-jousiammuntapeleissä haptista palautetta käytetään usein viestimään pelaajalle nuolen kiinnittyminen jousen jänteeseen. Näin pelaaja tietää värinätehosteen perusteella, että jousi on ampumavalmis. Haptisia tehosteita tulisi kuitenkin käyttää maltillisesti, sillä liian voimakas tai jatkuva värinä voi olla pelaajalle epämiellyttävää. (Tullis & Rihn 2017.)

### 3.4 Tasapaino

Eri iskujen sisältämien riskien ja tarjoamien palkintojen tulee olla tasapainossa. Helppona esimerkkinä paljon vauriota tekevä isku jättää käyttäjänsä puolustuksen avoimeksi iskun mennessä ohi (kuva 7).




KUVA 7: Hyökkäyksen edut ja haitat, Street Fighter II (Lambottin 2012)

Iskujen vastaanottaminen heikentää pelaajan kontrollia hahmostaan hetkellisesti. Tätä nimitetään englannin kielisellä termillä ”hitstun”. Mikäli isku torjuttiin, kohdistetaan torjujaan lyhyempi viive, jota kutsutaan ”block stuniksi.” Yhdessä hyökkäysten palautumisnopeuden kanssa nämä viivetyypit muodostavat olennaisen osan taistelupelien tasapainottamisen menetelmistä. Koska viivetyypit ovat animaatiosidonnaisia ja taistelupelit tyypillisesti rakennetaan toimimaan kuudenkymmenen ruudunpäivityksen sekuntitahdilla, puhutaan taistelupelien tasapainottamisessa ”frame datasta”. (Core-A Gaming 2016.)

Frame datan avulla pystytään määrittelemään muun muassa torjuntaa vastaan turvalliset hyökkäykset tai toiminnaltaan varmat hyökkäysyhdistelmät sekä näiden vastakohdat. Hyökkäysanimaatioiden eri osioiden pituus suhteutettuna hyökkäyksen aiheuttamaan hitstuniin tai blockstuniin määrittää, mitkä iskut pelaajan on mahdollista linkittää toisiinsa. Taulokossa 2 on aiheesta yksinkertaistettu esimerkki.

TAULUKKO 2. Kuvitteellisen iskujoukon frame data ja keskenäiset suhteet

| Isku | Aktiivinen framella | Palautuminen | Hitstun / blockstun | Hyökkääjän tila torjunnan seurauksena | Varmat vaihtoehdot osuman seurauksena |
|---|---|---|---|---|---|
| Torjunta | 1 | (Torjunnasta aiheutunut blockstun) | - | - | - |
| Lyönti 1 | 4 | 6 | 10 / 8 | Etu, 2 framea (8 – 6 = 2) | Lyönti 2, torjunta |
| Lyönti 2 | 2 | 6 | 7 / 6 | Neutraali, 0 framea (6 – 6 = 0) | Torjunta |
| Potku | 8 | 16 | 20 / 10 | Tappio, 6 framea (10 – 16 = -6) | Lyönti 1, lyönti 2, torjunta |

Esimerkin mukainen lyönti yksi lukitsee torjujan torjuntatilaan useammaksi frameksi kuin mitä hyökkääjällä menee aloittaa lyönti kaksi. Potku sen sijaan on torjuntaa vastaan ”turvaton” liike ja mahdollistaa torjujalle lyönnit yksi ja kaksi. Esimerkin mukaisella potkulla on mahdollista osua varmasti ainoastaan, jos vastapuoli on keskellä ohi menneen hyökkäyksen animaatiota. Esimerkistä poiketen iskun aktiivinen osa kestää useimmiten pidempään kuin yhden framen ja pelistä ja liikkeestä riippuen palautumisen voi pystyä keskeyttämään.

# 4 OMAN TAISTELIJAN LUOMINEN

## 4.1 Toteutustyyli

Tässä raportissa kuvaillaan ominaisuuksiltaan karsitun, platform fighter -tyylisen taistelijan rakentamista Unity-pelimoottorilla. Toteutustyyppi valikoitui erityisesti yksinkertaisuutensa vuoksi, sillä kehitykseen käytettävissä ollut ajanjakso oli lyhyt. Yksinkertainen ohjattavuus Super Smash Bros.:in tyyliin mahdollisti muunmuassa erillisen syötepuskurin karsimisen toteutuksesta. Tuotoksen tulosten soveltaminen myös monimutkaisemman taistelijan kehitykseen lienee kuitenkin mahdollista.

Platform fighter -peleissä on tyypillisesti kaksi hyökkäysnappia, yksi perushyökkäyksille ja toinen niin kutsutuille spesiaalihyökkäyksille. Hahmon suorittama hyökkäys riippuu pelaajan liikesyötteistä, painetusta hyökkäysnapista sekä siitä, onko hahmo ilmassa vai maassa. Torjunnalle ja heitolle on omat nappinsa, mutta heiton voi suorittaa myös painamalla hyökkäysnappia torjunnan aikana.

## 4.2 Työn rajaus ja prioriteetit

Tuotoksen ei ole tarkoitus olla valmis taistelupeli. Tästä syystä työssä ei toteutettu kunnollista pelisilmukkaa tai muita valmiin pelin kannalta olennaisia ominaisuuksia, kuten valikkoja tai tekoälyä. Uusien hyökkäysten lisääminen Unityssä tarpeen mukaan pyrittiin tekemään mahdollisimman vaivattomaksi. Pelinäkymässä on testausta helpottava nappi kentän palauttamiseksi alkutilaansa.

Toteutettu hahmo osaa liikkua sivusuuntaisesti ja hypätä. Hahmolla on käytettävissään useita lähitaisteluhyökkäyksiä sekä torjunta hahmon ollessa maan pinnalla, yksi hyökkäys ja väistöliike hypyn aikana. Kaikki hyökkäykset on sidottu yhteen hyökkäysnappiin ja osa liikkeistä vaatii hyökkäyssyötteen lisäksi jotain ohjaustikun asentoa tai vastaavaa syötettä näppäimistöllä. Heittoliikkeitä,

aseita ja ammushyökkäyksiä ei sisällytetty työn skaalaan kompleksisten ominaisuuksien karsimiseksi.

Iskuista on kolme eri tyyppiä: ”heikko”, ”normaali” ja ”vahva”. Erityyppiset iskut aiheuttavat vastaanottajaansa erilaisen animaation. Iskujen aiheuttaman vaurion määrää ja positionaalisen vaikutuksen voimakkuutta ja suuntaa pystyy säätämään hyökkäyskohtaisesti Unityn editorissa.

Torjunta on toteutettu Super Smash Bros. -tyylisesti läpikuultavana kuplana, joka pienenee ottaessaan vastaan iskuja jättäen osia hahmosta suojaamatta. Kuplan sijaintiin suhteessa hahmoon ei pysty itse vaikuttamaan. Torjutun iskun positionaalinen vaikutus torjujaan on heikennetty ja hyökkääjään kohdistuu viivettä iskukohtaisesti määritetty määrä. Torjunnan aktivoituminen tapahtuu välittömästi nappia painettaessa mutta sen pudottamisessa on aina viiden framen viive.

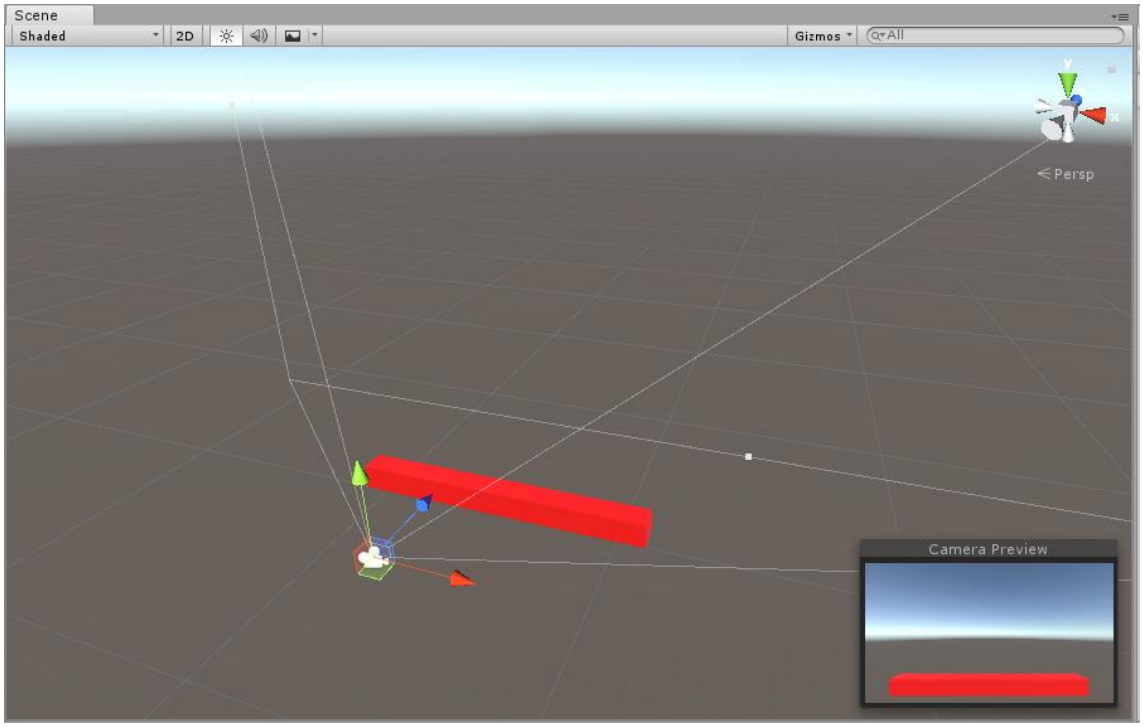
Hahmo pyrittiin toteuttamaan mahdollisimman modulaarisessa ja uudelleenkäytettävässä muodossa eräänlaiseksi hahmosapluunaksi. Perustoiminnot, kuten liikkuminen ja hyökkäykset, toteutettiin helposti säädettävinä ja monistettavina, että uusia, erilaisia hahmoja pystyttäisiin rakentamaan pohjahahmon klooneista vähäisin muutoksin.

Toteutuksen graafisesta asusta vastaa Blenderillä toteutettu yksinkertainen humanoidihahmo sekä Unityssä luodut primitiiviset kappaleet, kuten kuutiot ja kapselit. Käytetyt ohjelmistoversiot ovat Unity 2018.3 ja Blender 2.79b. Toteutukseen ei sisälly ääni- tai partikkeliefektejä; hahmon tilaa sekä iskujen tyyppejä visualisoidaan tekstien ja omatekoisten hahmoanimaatioiden ja mallin avulla.

## 4.3 Toteutus vaiheittain

### 4.3.1 Unity: uusi projekti, prefabit ja komponentit

Toteutus lähti liikkeelle halutun Unity-version lataamisella ja uuden projektin luomisella. Tyhjään vakiosceneen lisättiin kuutio-objekti editorin ylälaidan valikoista toimimaan maan pintana, minkä sijainti asetettiin 3d-avaruuden (0,0,0) -koordinaatteihin. Kuutio venytettiin x-akselilla järkevämpään kokoon ja sille annettiin punainen väri (kuva 7).




KUVA 7. Projektin aloitustila, kuvankaappaus

Unityssä on editorin vakiokappaleiden lisäksi mahdollista luoda käyttäjän määrittämiä, monistettavissa olevia peliobjekteja tai peliobjektikokonaisuuksia, joita kutsutaan prefabeiksi. Uuden prefabin voi luoda helposti raahaamalla editorissa minkä tahansa objektin vakioasetuksilla editorin vasemmasta laidasta löytyvästä Hierarchy-näkymästä alalaidan Assets-kansioon. Nyrkkisääntönä kaiken, mitä pelissä odotetaan esiintyvän useammin kuin yhden kerran, kannattaa olla prefab-muodossa. (Prefabs n.d.)

Unityn versio 2018.3 uudisti prefabien toiminnallisuutta ja muun muassa lisäsi editoriin uuden näkymän niiden editoimiselle. Prefab-moodissa kappaletta on mahdollista muokata omassa 3d-tilassaan itse peliscenen sijaan. Kappaleelle voidaan esimerkiksi lisätä Unityn vakiokomponentteja, luotuja skriptikomponentteja sekä lapsiobjekteja. Prefab-moodissa tehtävät muutokset päivittävät kaikkia missä tahansa peliscenessä olevia klooneja kappaleesta. Uudistusten myötä prefab voi sisältää osanaan myös toisia prefabeja. (Nested Prefabs n.d.)

Testisceneen lisättiin uusi kuutio toimimaan hahmon objektihierarkian korkeimpana tasona ja sille annettiin editorissa tunnistettava nimi ”Character”. Hahmon toimintojen toteutus aloitettiin liikkumisen implementoinnista, joten kuutio oli alkuvaiheessa riittävä representaatio hahmosta. Hyökkäyksiä lisättäessä se korvattiin animoidulla humanoidihahmolla.

Hahmosta tehtiin välittömästi uusi prefab ja prefab-moodissa sille lisättiin Character Controller -komponentti. Tämä komponentti sisältää oman törmäystarkastelukomponenttinsa (Collider) sekä hahmon liikuttamiseen hyödyllisiä toiminnallisuuksia kuten Move-funktion, mutta ei aseta peliobjektia fysiikkalaskennan piiriin (Character Controller n.d.). Ilmassa olevien hahmojen putoaminen toteutettiin osana hahmon liikkumista Move-funktiota käyttäen.

Hahmolle lisättiin myös fysiikasta vastaava Rigidbody-komponentti minkä asetus ”IsKinematic” asetettiin päälle. Tämän seurauksena hahmo ei putoa fysiikkamallinnuksen määräämällä tavalla mutta pystyy vaikuttamaan muihin fysiikkaobjekteihin (Rigidbody n.d.). Rigidbodystä oli hyötyä erityisesti hyökkäysten implementoinnissa, sillä Unityn törmäystarkastelufunktiot eivät toimi mikäli edes toisen törmäävistä kappaleista komponentteihin ei kuulu Rigidbody-komponenttia (MonoBehaviour.OnTriggerEnter n.d.).

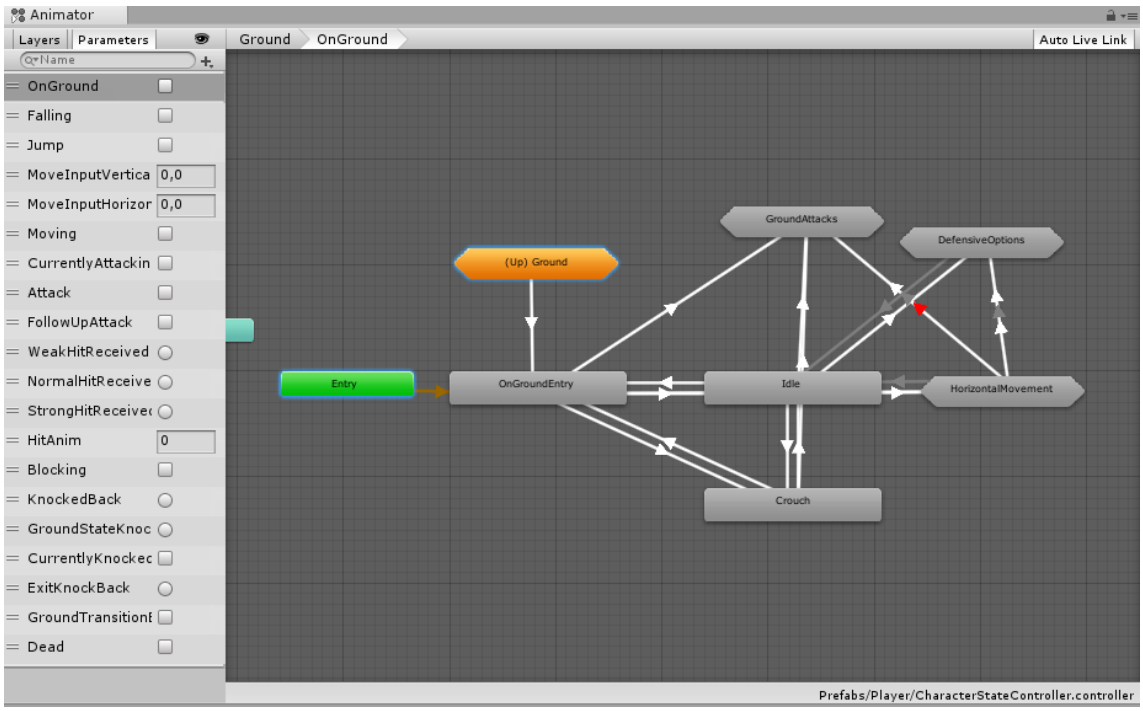
Rigidbody on myös eduksi optimoinnin kannalta, sillä kaikilla ajon aikana liikkuvilla, törmäystarkastelukomponentin omaavilla peliobjekteilla tai jollain näiden vanhemmista tulisi olla Rigidbody-komponentti. Muutoin Unityn fysiikkamoottori käsittelee näitä staattisina törmäyskappaleina, joiden

liikuttaminen johtaa moottorin suorittamaan tarpeettomaan fysiikkalaskentaan. (Physics best practises n.d.)

### 4.3.2 Hahmon tilan hallinta

Pelihahmolla on lukuisia eri tiloja, esimerkiksi yksinkertaisen hahmon toiminnot voisivat koostua seisonta-, juoksu- ja hyppytiloista. Taistelupeleissä hahmon tilat ovat huomattavasti tätä moninaisempia ja tiloissa tulee myös olla rajoitteita, että tilanmuutoksista ei seuraa epäloogista käytöstä. Lisäksi myös toisten hahmojen tulee pystyä vaikuttamaan hahmon tilaan.

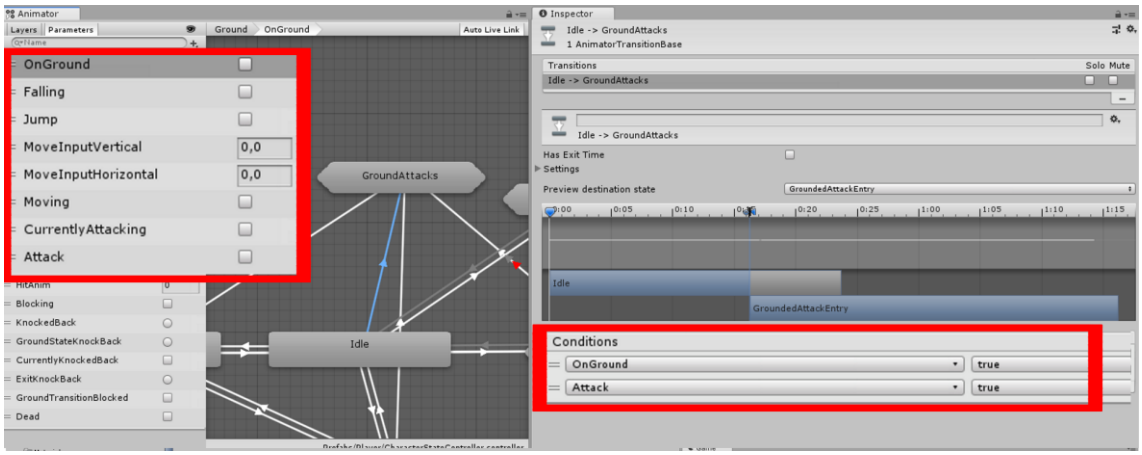
Oman tilakoneen ohjelmoiminen olisi yksi vaihtoehto, mutta myös Unityn sisäänrakennettu Mecanim-animaatiotilakone toimii tähän tarkoitukseen. Mecanimilla tilat ja niiden väliset suhteet rakennetaan hieman visuaalista ohjelmointia muistuttavalla editorilla. Näin pystytään esittämään hahmon toimintaa kaaviomaisesti ja samalla vähentämään tarvitun ohjelmoinnin määrää (kuva 8).




KUVA 8. Hahmon maassa tapahtuvien toimintojen ylin taso, kuvankaappaus

Tilat näkyvät editorissa suorakaiteen mallisina kappaleina, kuusikulmaiset kappaleet ovat alitilakoneita jotka sisältävät tilakokonaisuuksia kuten toisiinsa linkittyviä hyökkäyksiä. Alitilakoneet ovat käteviä tilaeditorin siistinä pitämisessä, sillä useita tiloja vaativien ominaisuuksien lisäämisen myötä näkymä muuttuu nopeasti vaikeaselkoiseksi hämähäkinverkoksi.

Tilojen välillä liikkuminen tapahtuu hyödyntäen nuolina visualisoituja siirtymiä ja Animator-ikkunan vasemmasta laidasta löytyviä parametreja. Siirtymät laukaistaan if-lauseen tapaan liitännäisten parametrien arvoista riippuen (kuva 9). Parametreihin voi asettaa arvoja hakemalla skriptissä viittaus saman peliobjektin sisältä löytyvään Animator-komponenttiin ja kutsumalla sen kautta sopivia funktioita kuten Animator.SetBool tai Animator.SetFloat.




KUVA 9. Korostetun siirtymän ehdot Inspector-näkymässä, kuvakaappaus

Tiloihin voi lisätä toiminnallisuutta StateMachineBehaviour-luokan periviä skriptejä hyödyntämällä. Kyseinen luokka tarjoaa muun muassa funktiot koodin suorittamiseen tilaan siirryttäessä ja tilasta poistuttaessa, sekä ruudunpäivityksen tahtia toistuvasti suoritettavaan OnStateUpdate-funktioon. (StateMachineBehaviour n.d.) Unityn vastaussivuilla (Five tips for keeping animator controllers nice n' tidy n.d.) neuvotaan kuitenkin olemaan suorittamasta monimutkaista koodia StateMachineBehavior-luokan perivissä skripteissä ja suosimaan mieluummin viestien lähettämistä muihin komponentteihin haluttujen tulosten saavuttamiseksi.

Ratkaisuna hahmolle luotiin pohjaluokka joka ohjaa hahmon kaikkea toimintaa muiden komponenttien ja animaatiotilakoneen avulla. Pohjaluokkaan asetettiin

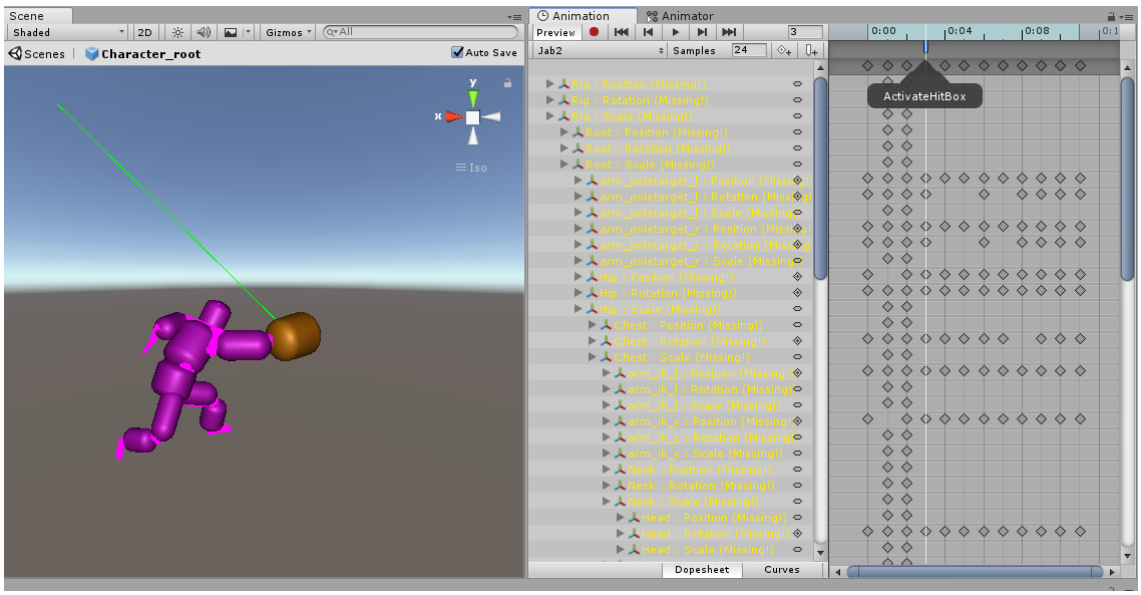
useita muuttujia, joista tehtiin julkisia Mecanim-tiloista tehtävien muutosten mahdollistamiseksi. Arvojen muutoksilla laukaistaan funktiokutsuja pohjaluokassa.

### 4.3.3 Hyökkääminen, vaurion vastaanottaminen ja torjunta

Unityn 2D- ja 3D-fysiikkamoottorit tarjoavat törmäystarkasteluun tarkoitetut Collider2D ja Collider -komponentit, joista on molemmista useita erimuotoisia vaihtoehtoja. Törmäystarkastelukomponenteilla ei itsessään kuitenkaan ole pelin ajossa näkyvää graafista asua. (Colliders n.d.) Hyökkäyksenä voisi yksinkertaisimmillaan toimia napin painalluksella aktivoituva näkymätön laatikko, missä tarkastellaan, jääkö jokin muu kappale kyseisen laatikon sisälle ja suoritetaan jatkotoimenpiteitä tarpeen vaatiessa. Tällaista laatikkoa kutsutaan yleensä termillä hitbox.

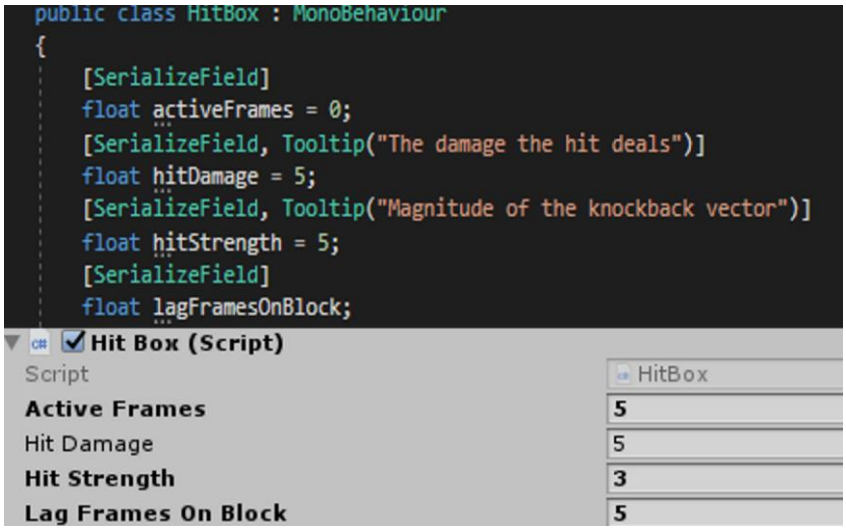
Tähän tarkoitukseen luotiin uusi prefab joka sisältää halutut toiminnot toteuttavan hitbox-skriptikomponentin ja törmäystarkastelukomponentin. Hitbox-luokka on vastuussa ainoastaan törmäystarkastelun suorittamisesta ja iskun parametrien välittämisestä osuman vastaanottaneelle objektille. Unityn versiossa 2018.3 lisätty tuki prefabien sisäkkäisyydelle (Nested Prefabs n.d.) mahdollistaa hyökkäysprefabin kloonien asettamisen hahmoprefabin objektien lapsiobjekteiksi, joita sitten aktivoidaan ja deaktivoidaan käytetyn hyökkäyksen mukaan.

Hahmon pohjaluokkaan lisättiin lista kaikista hahmon hitboxeista, julkinen kokonaislukutyyppinen indeksimuuttuja listan selaamiseen sekä julkinen ActivateHitbox-funktio, joka aktivoi listasta indeksimuuttujan osoittaman hitboxin. Mecanimissa hyökkäystiloihin lisätään uusi AttackBehaviour -skripti, minkä kautta syötetään pohjaluokkaan halutun hitboxin indeksi. Lopulta ActivateHitbox-funktiota kutsutaan animaatioon sidotun funktiokutsun, animation eventin, kautta (kuva 10). Hitbox deaktivoituu itsenäisesti komponentin activeFrames-muuttujassa määritetyn viiveen jälkeen.




KUVA 10. Oranssi hitbox aktivoidaan animation eventillä jab2-animaation edetessä framelle 4, kuvakaappaus

Eri hyökkäyksillä voidaan haluta olevan vaihtelevia ominaisuuksia, kuten aiheutettavan vaurion määrä sekä mahdollisen positionaalisen vaikutuksen voimakkuus ja suunta. Nämä voitaisiin asettaa julkisiksi muuttujiksi hitbox-luokkaan, jolloin ne näkyisivät vakiona editorissa. Muuttujista voidaan myös tehdä yksityisiä ja asettaa ne näkymään editorissa [SerializeField] -määreellä. (kuva 11).




KUVA 11. Hyökkäyksen yksityiset muuttujat saadaan näkyviin editorissa, kuvankaappaus

Unityssä on mahdollista asettaa peliobjekteja toisten peliobjektien lapsiobjekteiksi, jolloin ne perivät vanhemmiltaan ominaisuuksia, kuten sijainnin

ja rotaation (Parent Constraints n.d.). Demetrio valaisee blogissaan (2017), että taistelupelit hyödyntävät tätä usein asettamalla hitboxit hahmomallin animointiin käytettyjen ”luiden” lapsiobjekteiksi. Näin hitboxit saadaan liikkumaan animaatioiden mukana luontevasti.

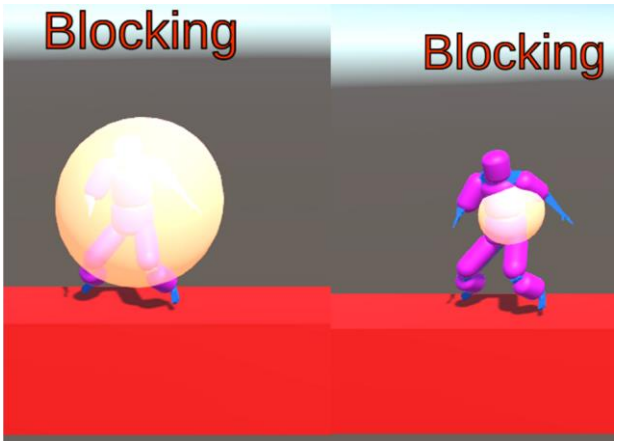
Hyökkäyksen osuessa toiseen pelihahmoon tulisi tämän reagoida iskuun sopivalla tavalla. Tähän voidaan vastaavasti käyttää peliobjektia, mikä sisältää törmäystarkastelukomponentin ja osuman prosessointiin käytettävän skriptikomponentin. Tällaista kappaletta kutsutaan yleensä hurtboxiksi.

Kappaleelle tehtiin oma prefabinsa, mikä sisältää hurtbox-skriptikomponentin. Hitbox ja hurtbox toimivat yhdessä seuraavasti: hitboxin törmäystarkastelun löytämät peliobjektit lisätään listaan, mikä sitten selataan läpi siltä varalta että jollakin kappaleista on hurtbox-komponentti. Sopivan kohteen löytyessä kutsutaan hurtboxin julkiseksi asetettua prosessointifunktiota, mihin syötetään parametreiksi hyökkäyksen ominaisuudet kuten sen tekemä vahinko, työntävä voima ja työnnön suunta.

Hurtboxit voidaan hitboxien tavoin asettaa hahmon animaatioluiden lapsiobjekteiksi. Näin osumatarkastelusta voidaan saada suhteellisen tarkasti hahmon muotoa ja liikkeitä mukaileva useita hurtboxeja käyttämällä (kuva 12).


KUVA 12. Vaaleanpunaiset hurtboxit seuraavat hahmon liikkeiden mukana, kuvakaappaus

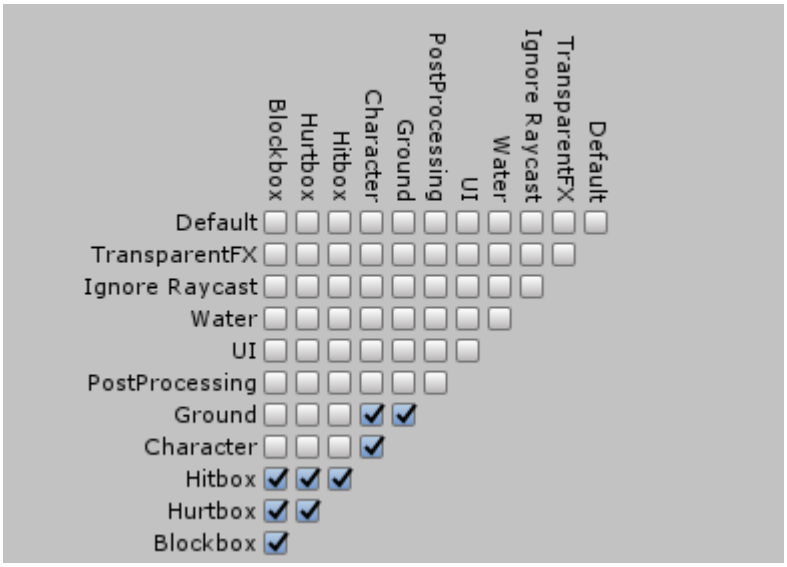
Torjunnalle on oma blockbox-komponenttinsa, mikä toimii samalla periaatteella kuin hurtbox, mutta ohjaa vastaanotetun vaurion omaan elämäpistevarantoonsa hahmon elämäpisteiden sijaan. Blockbox on tässä toteutuksessa Smash Bros. -tyylinen kupla joka pienenee menettäessään elämäpisteitään (kuva 13).




KUVA 13. Terve ja vaurioitunut blockbox rinnakkain, kuvakaappaus

Hurtbox ja Blockbox -luokkien toiminnan samankaltaisuudesta johtuen molemmat luokat periytyvät abstraktista AbstractHurtbox-luokasta, missä määritellään molempien toiminnan kannalta olennainen ProcessHit-funktio. Koska Unityn törmäystarkastelu ei palauta löytämiään objekteja missään mielekkäässä järjestyksessä, etsitään hitboxissa osuman tarkastelussa ensimmäisenä mahdollisena vaihtoehtona blockboxia. Näin voidaan varmistaa että iskut osuvat blockboxiin kuplan peittämien hurtboxien sijasta.

Peliobjektit tarkastetaan AbstractHitbox-komponentin varalta sen sijaan että hurtboxille ja blockboxille olisi erilliset tarkastelut ja erotetaan toisistaan Unityn kerrosten (layers) avulla. Kerroksilla voidaan myös esimerkiksi varmistaa, että törmäyksiä tarkastellaan ainoastaan iskujen ja niiden vastaanottamiseen tarkoitettujen objektien välillä (Layer-based collision detection n.d.). Tämä tapahtui säätämällä Unityn fysiikkamatriisia kuvan 14 mukaisesti.




KUVA 14. Mahdolliset törmäyssuhteet kerrosten välillä, kuvakaappaus

# 5 POHDINTA

Tämän opinnäytetyön tavoitteena oli perehtyä taistelupelien yleisiin ominaisuuksiin ja perehtyä niiden toteuttamiseen Unity-pelimoottorissa, sekä syventää tekijän henkilökohtaista ymmärrystä ja osaamista Unity-kehittäjänä. Työn tarkoituksena oli esitellä taistelupelihahmojen yleisiä toimintoja ja niiden variaatioita sekä käydä läpi keinoja miellyttävän pelikokemuksen rakentamiseen. Koen opinnäytetyön olleen hyödyllinen henkilökohtaisen kehittymiseni kannalta ja näkisin työn myös kokonaisuutena varsin onnistuneeksi.

Aiempi kokemus tasoloikkapelien kehityksestä Unityssä oli hyödyksi oman hahmon toteutuksessa valitulla tyylillä. Entuudestaan vieraampi lähitaistelumekaniikkojen implementoiminen oli ohjelmoinnin osalta yllättävänkin yksinkertaista ja hahmoanimaatioiden rooli hahmon toimivuudessa oli merkittävästi suurempi, kuin alunperin kuvittelin.

Kontrollien ohjelmointi Unityn sisäänrakennetuilla syötteenlukuominaisuuksilla sen sijaan osoittautui haastavaksi. Esimerkiksi syötteiden tarkasteluun käytettävä merkkijono ”joystick button 0” palauttaa eri napin Playstation 4 -ohjaimesta kuin Xbox 360 -ohjaimesta. Projektista tehdyssä WebGL-buildissa havaittiin myös eroja ohjainten toiminnassa riippuen käytetystä selaimesta ja käyttöjärjestelmästä; PS4-ohjaimelle rakennetut kontrollit toimivat oikein Firefoxilla Windows 10:ssä mutta eivät Linuxissa (Manjaro), Chrome puolestaan käsittelee PS4-ohjainta Xbox 360 -ohjaimena minkä vuoksi toiminnot tapahtuvat eri napeista kuin projektia editorissa testattaessa. Ongelmallisuuden ja aiheen kannalta epäolennaisuuden seurauksena projektiin toteutettiin vain testaukseen soveltuva ohjaus kunnollisen ohjaintuen sijaan.

Hieman yllättäen tiedonhaku työhön osoittautui suurimmaksi haasteeksi. Taistelupelien kehityksestä ei vaikuta olevan tuotettu erityisen kattavasti validia tieteellistä aineistoa. Taistelupeleille yleinen terminologia on jokseenkin epämääräinen yhdistelmä pelinkehittäjien ja pelaajien keksimää sanastoa, mistä lähellekään kaikki ei päde jokaiseen taistelupeliin ja mihin liittyvää tietoa löytyi

pääasiassa Wiki-mallisilta, avoimesti muokattavissa olevilta sivustoilta sekä keskustelupalstoilta.

Oman taistelijan toteutus onnistui mielestäni pääosin hyvin. Unityn Mecanim-animaatiotilakone oli käyttökelpoinen mutta ei täysin ongelmaton ratkaisu hahmon tilan hallintaan. Projektin jäljiltä sanoisin, että tosi/epätosi -tyyppisistä parametreista kannattaa tilasiirroksissa suosia automaattisesti vakioarvoonsa palautuvaa liipaisinta (trigger) booleanin sijaan aina kun mahdollista, että parametrejä ei unohdu väärille arvoille estämään hahmon tarkoitettua toimintaa.

Koen Mecanimin käytön olleen edullinen erityisesti hahmon toimintojen uudelleenkäytettävyyden kannalta. Uusien hyökkäyksien lisääminen hahmolle muuttui nopeaksi ja vaivattomaksi, kun sain yhden toimivan hyökkäystilan toteutettua. Alitilakoneiden käytöllä voidaan mahdollistaa suhteellisen monimutkaistenkin tilarakenteiden kierrättäminen. Jos esimerkiksi haluaisin uuden kolmen iskun yhdistelmän hahmolle, voisin vain kopioida olemassa olevan alitilakoneen, vaihtaa iskujen animaatiot ja säätää hitboxien ominaisuuksia sopimaan uusiin liikkeisiin. Uskon työn perusteella tämän lähestymistavan olevan pätevä menetelmä suuremman hahmovalikoiman toteuttamisessa.

**LÄHTEET**

1UP. 2012. The Essential 50 Part 35: Virtua Fighter. Luettu 17.4.2019. https://archive.is/20120719110526/http://www.1up.com/features/essential-50-virtua-fighter

AOTF. 2018. Super Smash Bros. Ultimate How To Block. Torjunta Super Smash Bros. Ultimatessa. Luettu 16.4.2019. https://attackofthefanboy.com/guides/super-smash-bros-ultimate-how-to-block/

Bandai Namco Entertainment. N.d. Tekken 7: Guides. Luettu 4.2.2019. https://tk7.tekken.com/guides

Brown, M. 2018. What Makes a Good Combat System? | Game Maker's Toolkit. Youtube-video. Katsottu 24.3.2019. https://www.youtube.com/watch?v=8X4fx-YncqA

Burgun, K. 2012. Game Design Theory: A New Philosophy for Understanding Games. Boca Raton, Florida: CRC Press.

Capcom. N.d. Street Fighter II Turbo Instruction Manual. Luettu 18.1.2019. https://www.nintendo.co.jp/clvs/manuals/common/pdf/CLV-P-SABHE.pdf

Capcom. N.d. Street Fighter 5: How to Play. Luettu 4.2.2019. https://streetfighter.com/how-to-play/

Cartwright, M. 2014. Making Fluid and Powerful Animations For Skullgirls. Game Developers Conference. Youtube-video. Katsottu 4.2.2019. https://www.youtube.com/watch?v=Mw0h9WmBlsw

Cartwright, M. 2015. The GDC 2015 Live Animation Demo. Kuvakaappaus hahmoanimoinnin live-esitelmästä. Game Developers Conference. Youtube-video. Katsottu 20.4.2019. https://www.youtube.com/watch?v=z-5djm1pRpU

Core-A Gaming. 2016. Analysis: What Makes a Move Overpowered? Youtube-video. Katsottu 10.4.2019. https://www.youtube.com/watch?v=uQnfm911Xoc

Core-A Gaming. 2017. Analysis: Tekken – The Difference Between 2D and 3D. Youtube-video. Katsottu 15.2.2019. https://www.youtube.com/watch?v=YGSkWVXE2mo

Demetrio, A. 2017. I Wanna Make a Fighting Game! Part IV – Hitboxes and Hurtboxes. Luettu 20.1.2019. https://indiewatch.net/2017/12/02/wanna-make-fighting-game-part-iv-hitboxes-hurtboxes/

Dieterich, R. 2011. QCF+Punch: Implementing Street Fighter-style Input. Game Dev Without a Cause. Luettu 13.3.2019. http://gamedevwith-outacause.com/?p=266

Elliot, R. 2018. Evo 2018 Was a Big Hit, Generating 5.2 Million Hours Watched on Twitch and YouTube Gaming. Newzoo. Luettu 16.1.2019.

https://newzoo.com/insights/articles/evo-2018-was-a-big-hit-generating-5-2-million-hours-watched-on-twitch-and-youtube-gaming/

Floyd, D. 2018. How to Animate a Smash Bros Attack // LINK – New Frame Plus. Youtube-video. Katsottu 10.4.2019. https://www.youtube.com/watch?v=3kTYazhO3fs

Gilyadov, A. 2015. KO: The History Of Fighting Games. CGMagazine. Luettu 10.2.2019. https://www.cgmagonline.com/2015/04/15/ko-the-history-of-fighting-games/

Itsuno, H. 2019. Devil May Cry 5: Creating a Standout Action Game. Game Developers Conference. Youtube-video. Katsottu 11.4.2019. https://www.youtube.com/watch?v=CWi5gjxdh4o

Jensen, K.T. 2016. Street Fighter, Mortal Kombat, Smash Bros. and Beyond: The Ultimate History of Fighting Games. Geek.com. Luettu 21.1.2019. https://www.geek.com/games/street-fighter-mortal-kombat-smash-bros-and-beyond-the-ultimate-history-of-fighting-games-1646671

Lambottin, S. 2012. The Fundamental Pillars of a Combat System. Hyökkäyksen edut ja haitat, Street Fighter II. Gamasutra. Luettu 2.4.2019. https://www.gamasutra.com/view/feature/175950/the_fundamental_pillars_of_a_.php

Maxim. 2017. 'Street Fighter II' Is Getting a 30th Anniversary Re-release That Should Thrill Old-School Gamers. Street Fighter 2. Luettu 16.4.2019. https://www.maxim.com/entertainment/street-fighter-ii-30th-anniversary-re-release-2017-9

Nintendo. 2017. Pokkén Tournament DX For Nintendo Switch – Nintendo Game Details. Pokkén Tournament DX. Luettu 16.4.2019. https://www.nintendo.com/games/detail/pokken-tournament-dx-switch/

Pignole, Y. 2014. Platformer controls: how to avoid limpness and rigidity feelings. Gamasutra. Luettu 13.3.2019. http://www.gamasutra.com/blogs/YoannPignole/20140103/207987/Platformer_controls_how_to_avoid_limpness_and_rigidity_feelings.php

Reddit. 2018. What Are The Fighting Game Playstyle Archetypes? Luettu 16.5.2019. https://www.reddit.com/r/Fighters/comments/8cei7c/what_are_the_fighting_game_playstyle_archetypes/

Shoryuken. 2017. This Is the Year You Will Learn How to Perform Tekken's Korean Backdash! Luettu 17.4.2019. http://shoryuken.com/2017/02/17/this-is-the-year-you-will-learn-how-to-perform-tekkens-korean-backdash/

Tullis, M. & Rihn, W. 2017. Unite Austin 2017 – Game Design with Haptics: The Next Level of Immersion. Youtube-video. Katsottu 16.5.2019. https://www.youtube.com/watch?v=QFzh8yeFR4k

Unity. N.d. Character Controller. Luettu 17.1.2019. https://docs.unity3d.com/Manual/class-CharacterController.html

Unity. N.d. Colliders. Luettu 17.1.2019. https://docs.unity3d.com/Manual/CollidersOverview.html

Unity. N.d. Five tips for keeping animator controllers nice n' tidy. Luettu 18.1.2019. http://response.unity3d.com/animator-controllers

Unity. N.d. Layer-based collision detection. Luettu 19.1.2019. https://docs.unity3d.com/Manual/LayerBasedCollision.html

Unity. N.d. MonoBehaviour.OnTriggerEnter. Luettu 17.1.2019. https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter.html

Unity. N.d. Nested Prefabs. Luettu 17.1.2019. https://docs.unity3d.com/Manual/NestedPrefabs.html

Unity. N.d. Parent Constraints. Luettu 17.1.2019. https://docs.unity3d.com/Manual/class-ParentConstraint.html

Unity. N.d. Physics Best Practises. Luettu 18.1.2019. https://unity3d.com/learn/tutorials/topics/physics/physics-best-practices

Unity. N.d. Prefabs. Luettu 17.1.2019. https://docs.unity3d.com/Manual/Prefabs.html

Unity. N.d. Rigidbody. Luettu 17.1.2019. https://docs.unity3d.com/Manual/class-Rigidbody.html

Unity. N.d. StateMachineBehaviour. Luettu 17.1.2019. https://docs.unity3d.com/ScriptReference/StateMachineBehaviour.html