Samu Ahvenjärvi

TEKOÄLYN LUOMINEN TIETOKONEPELIIN KÄYTTÄEN UNITY 3D -OHJELMISTOA

Opinnäytetyö
KESKI-POHJANMAAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Syyskuu 2012

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö Ylivieska	**Aika** Syyskuu 2012	**Tekijä/tekijät** Samu Ahvenjärvi
Koulutusohjelma Tietotekniikka		
Työn nimi TEKOÄLYN LUOMINEN TIETOKONEPELIIN KÄYTTÄEN UNITY 3D - OHJELMISTOA		
Työn ohjaaja Hannu Puomio		**Sivumäärä** 42 + 49
Työelämäohjaaja Ville Autio		

Tämän opinnäytetyön tarkoituksena oli luoda toimiva tekoäly tietokonepeliin ja samalla selvittää Unity 3D-ohjelman soveltuvuutta kyseiseen tehtävään. Työssä tutkittiin myös Motion Capture-tekniikan hyödyntämistä Steamdroid tietokonepelissä. Se tehtiin osana Centria – tutkimus & kehityksen ”Pelipaja”-projektia.

Opinnäytetyön teoriaosuus käsittelee historiaa tekoälystä yleisesti ja tietokonepelien kehittymisen osalta, tilakoneen toimintaa ja sumeaa logiikkaa. Opinnäytetyön pääpaino oli tekoälyn ohjelmoinnissa.

Työ rajattiin pelkästään vihollispelaajien tekoälyn toteuttamiseen.

Asiasanat
Ohjelmointi, sumea logiikka, tekoäly, Unity 3D, äärellinen tilakone

CENTRAL OSTROBOTHNIA
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES	Date September 2012	Author Samu Ahvenjärvi
Degree programme Information Technology		
Name of thesis CREATING ARTIFICIAL INTELLIGENCE TO A COMPUTER GAME USING UNITY 3D SOFTWARE		
Instructor Hannu Puomio		Pages 42 + 49
Supervisor Ville Autio		

The objective of this thesis was to create workable artificial intelligence to a computer game and find out, if the Unity 3D software is suitable for this purpose. Using motion capture technology in Steamdroid computer game was also researched.The Game was made as a part “Gaming Workshop” project of centria research and development.

The Theory part of this thesis focuses on the history of artificial intelligence as well as on the development of computer games, the principles of state machines and fuzzy logic. The main focus of this thesis was on programming artificial intelligence.

The subject of the thesis was limited only to implementing the artificial intelligence of enemy players.

Key words
Artificial intelligence, finite state machine, fuzzy logic, programming, Unity 3D

TIIVISTELMÄ
ABSTRACT
SISÄLLYS

KUVIOT

TAULUKOT

1 JOHDANTO

Olen aina ollut kiinnostunut erilaisista tietokonepeleistä ja siinä sivussa useasti pohdiskellut vihollispelaajien toimintaa. Miten joku vihollinen voi käyttäytyä tietyllä tavalla ja mistä se tietää toimia juuri näin?

Sain tilaisuuden vuoden 2010 keväällä, sillä kuulin koulullamme järjestettävän Pelipaja-nimisen peliprojektin. Projektin järjestäjä ja tämän opinnäytetyön toimeksiantaja oli koulumme oma Centria - tutkimus ja kehitys. Hain projektiin mukaan ja pääsinkin peliohjelmoijaksi. Tämän jälkeen olen ollut jokaisena vuonna mukana projektissa. Jossain vaiheessa aloin tuumia, että olisiko mahdollista tehdä opinnäytetyötä tietokonepeliin liittyen. Vuoden 2012 keväällä järjestettävästä projektista sain lopulta opinnäytetyöni aiheen – vihollisten tekoälyn luomisen.

Pelipaja-projektitiimissämme oli mukana yhteensä 10 ihmistä. Heillä kaikilla oli omat erikoisosaamisalueensa ja sitä kautta kaikki panostivat omalla osaamisellaan projektiimme. Opinnäytetyössä käytetty ohjelmakoodi on kirjoitettu suurimmalta osin itse, mutta joitain pieniä osa-alueita oli tehty valmiiksi kyseistä projektia varten. Tässä käydään läpi vain oma osuuteni ohjelmakoodista. Työharjoittelusta saadulla kokemuksella valitsin käytettäväksi Unity 3D-ohjelmiston, sillä se palvelee tarkoitustaan mielestäni parhaiten. Pelillämme ei aluksi ollut nimeä, mutta lopulta nimesimme pelin SteamDroidiksi.

Tässä opinnäytetyössä tutkin Unity3D-ohjelmiston soveltuvuutta tekoälyn luomiseen, sekä Motion Capture-tekniikan hyödyntämistä tekoälyssä. Jotta aihe ei olisi paisunut liian suureksi, oli opinnäytetyö rajattava pelkästään vihollisten tekoälyn toteuttamiseen. Tarkoituksena oli saada aikaan toimiva viholispelaajien tekoäly tietokonepeliin. Saatavilla olisi ollut tekoälyä tukevia apuohjelmia Unity 3D:lle, mutta lopulta päätin toteuttaa tekoälyn itse ohjelmoimalla.

Luvussa kaksi kerron aluksi tekoälyn historiasta yleisesti ja samalla tietokonepelien näkökulmasta. Siinä käsitellään samalla tilakoneiden teoriaa, motion capture-tekniikka ja sumeaa logiikkaa. Kolmannessa luvussa käydään läpi Unity 3D-

ohjelman ominaisuuksia ja neljäs luku on varsinainen käytännön osio, jossa käydään läpi SteamDroid-pelin ohjelmointi.

2 TEKOÄLY PELIMAAILMASSA

Tekoäly (Artificial intelligence) voidaan käsittää yhtenä ohjelmointitekniikan haarana, jonka avulla pyritään matkimaan tietokoneen avulla ihmisen tekemiä päättely- ja ajattelumalleja (Järvinen 2003, 670).

Tekoälyn avulla tietokonepelien hahmoille pyritään luomaan inhimillistä käyttäytymistä erilaisissa tilanteissa. Se on rakennettava siten, että pelaajan tekemät valinnat vaikuttavat suoraviivaisesti tietokonepelin vihollisen käyttäytymiseen. Tekoälylle ei löydy yksiselitteistä määritelmää, vaan siitä löytyy lukuisia eri versioita.

2.1 Pelien ja tekoälyn historiaa

2.1.1 1950-luku

Nykyaikaisen tekoälyn pioneerina pidetään englantilaista matemaatikkoa Alan Turingia, sillä hän kehitti 1950-luvulla julkaisun nimeltä *Machinery and Intelligence.* Tässä julkaisussa Turing ilmoitti, että tietokoneet voitaisiin ohjelmoida toimimaan älykkäästi ihmisen tavoin (Brookshear 2003, 438.)

Kyseisessä julkaisussa Turing kuvaili tekoälytestin, joka tunnetaan nykyisin nimellä *Turingin testi*. Testissä kyselijä, eli ihminen, kommunikoi kirjoituskonejärjestelmän kautta toisen ”ihmisen” kanssa. Kyselijälle ei kerrota, onko kyseessä ihminen vai tietokone. Testin avulla pystytään arvioimaan koneen kykyä käyttäytyä ihmisen tavoin. Tietokoneen sanottiin käyttäytyvän älyllisesti, mikäli kyselijä ei huomannut eroa käytöksessä verrattaessa tavalliseen ihmiseen (Brookshear 2003, 439.)

Turingin kunniaksi on 90-luvun alusta lähtien järjestetty kilpailu nimeltä *Loebner Prize*. Se on kilpailu, jonka pääpalkintona on 100 000 dollaria. Pääpalkinto luovutetaan henkilölle, jonka luoma tietokoneohjelma läpäisee Turingin testin. Kukaan ei ole tähän mennessä pystynyt luomaan tietokoneohjelmaa, joka olisi läpäissyt Turingin testin (Kukkonen 2010, 12.)

Turing ei ollut ainoa 1950-luvulla toiminut merkkihenkilö tekoälyn saralla. Vähintään yhtä merkittävään asemaan pääsi Arthur L. Samuel. Hän kehitti niin viisaan tammipelin, että hävisi sille lopulta itse (Kukkonen, 12).

2.1.2 1960-luku

Turingin jälkeen Joseph Weizenbaum kehitti 1960-luvulla ohjelman nimeltä DOCTOR. Se oli interaktiivinen ohjelma, joka oli ohjelmoitu psykoanalyytikon haastattelun kaltaiseksi. Tietokone analysoi käyttäjän vastauksia varsin yksinkertaisesti. DOCTOR-ohjelma toimi yksinkertaisella logiikalla, sillä se järjesti käyttäjän kysymän kysymyksen ovelalla tavalla uuteen muotoon. Käyttäjä saattoi kirjoittaa esimerkiksi: ”Olen väsynyt” ja ohjelma vastasi käyttäjälleen takaisin ”Miksi luulet olevasi väsynyt?”. Mahdolliset virhetilanteet ohjelma korjasi tokaisemalla epämääräisesti ”ahaa” tai ”sepä mielenkiintoista” (Brookshear 2003, 439.)

Samana vuosikymmenenä Stanfordin yliopisto rakennutti itselleen tekoälylaboratorion. Kyseisessä laboratoriossa rakennettiin robotti nimeltä Shakey. Se kykeni liikkumaan maailmassa, joka oli sijoitettu ruudukkoon ja se ymmärsi yksinkertaisia ohjeita (Kukkonen 2010, 12).

2.1.3 1970-luku

1970-luvulla tapahtui varsinainen laskukausi tekoälyn saralla. Tekoälysovelluksiin ei panostettu taloudellisesti ja tämän takia tutkimus- ja kehittämistoiminta kärsi. Suurin edistys oli Backgammon-lautapeliin luotu tekoäly, joka voitti vastapelaajansa maailmanmestari Luigi Villan. Koskaan aikaisemmin tietokone ei ollut voittanut ihmistä näin vaikeassa lautapelissä (Jones 2003, 6.)

1970-luvun alussa peliyhtiö nimeltä Atari alkoi kehittämään seuraavaa hittipeliään edellisen Computer Space-pelin menestyksen jäätyä vaisuksi. Atari kehitti täysin uuden pelin nimeltään *Pong*. Pong-peli soveltui käytettäväksi Computer Space-pelin runkoon, eli siinä oli vain senaikainen televisio ja joystick (Lecky-Thompson 2007, 7.) Pong on yksinkertainen kolikkoautomaattipeli, jossa on kaksi mailaa. Toinen maila on pelaajan oma ja toinen vastustajan. Palloa lyödään puolelta toiselle, ja pallon toisen pelaajan ohi saanut pelaaja saa itselleen pisteen. Periaate

on täysin sama kuin pöytätenniksessä. Se on samalla yksi ensimmäisistä kaupalliseen levitykseen tehdyistä peleistä.

Mainittakoon myös peli nimeltä *Asteroids*. Se julkaistiin suurelle yleisölle vuonna 1979 (Lecky-Thompson, 7). Asteroidsissa pelaajalla oli tarkoitus ohjata avaruusalusta ja samalla ampua näytölle ilmestyviä asteroideja. Peli vaikeutui koko ajan, sillä asteroidit jakaantuivat kahtia ja pelaaja joutui varomaan lisääntyneitä kivenlohkareita. Ulkoasultaan Asteroids oli hyvin pelkistetty ja selkeä.

2.1.4 1980-luku

1980-luku näytti jo selvästi edellisvuosikymmentä lupaavammalta, sillä tekoälypohjaisten komponenttien ja ohjelmistojen myynti ylitti silloiset 400 miljoonaa dollaria. Suurin osa myyntivoitoista koostui LISP-tietokoneista tai asiantuntijajärjestelmistä, jotka muuttuivat koko ajan paremmiksi ja huokeammiksi. Neuroverkot tekivät myös tuloaan ja sen avulla pystyttiin parantamaan puheentunnistusta. Puheentunnistus kehittyi lopulta niin, että puhuja saattoi puhua normaalisti pitämättä turhia taukoja. (Jones 2003, 7).

1980-luvulla kolikkopelit mullistivat peliteollisuuden täysin. Yksi tunnetuimmista kolikkoautomaattipeleistä julkaistiin vuonna 1980, peli nimeltä *Pac-Man* (Lecky-Thompson, 7-8.) Pelissä pelaajan oli tarkoitus syödä Pac-Manilla kentällä lojuvia ”kolikoita” ja kerätä itselleen mahdollisimman suuri määrä pisteitä. Kenttä on tyypillään sokkeloinen ja siinä lojuu myös isompia kolikoita. Näiden kolikoiden avulla viholliset muuttuvat tuhoamiskelpoisiksi.

Toinen merkittävä julkaisu on peli nimeltä *Donkey Kong.* Se on Nintendon vuonna 1981 julkaisema peli (Lecky-Thompson, 8). Pelin tarkoituksena pelaajan on väistelтävä vihollisen heittämiä tynnyreitä, jotka valuvat kerroksittain alaspäin. Pelaajan on pelastettava prinsessa ilkeän Donkey Kongin kynsistä päähahmon, Super Marion, avulla.

2.1.5 1990-luku

IBM kehitti shakinpeluu ohjelmiston nimeltä ”Deep Blue” vuonna 1997. Tätä ohjelmaa pyöritettiin silloisella IBM:n omistamalla supertietokoneella ja ohjelmisto onnistui päihittämään shakin maailmanmestarin Gary Kasparovin. Tämän lisäksi tiedemiehet onnistuivat luomaan ohjelmiston, jonka avulla avaruusalusta pystyttiin ohjaamaan maapallolta käsin. Muita merkittäviä 90-luvun merkkipaaluja ovat mm. kasvojentunnistus ja ajanhallintajärjestelmät (Jones 2003, 7.)

1990-luvulla syntyi monia tekoälyllisesti merkittäviä pelejä. Mainittakoon näistä peleistä ainakin kaksi kappaletta: *Unreal Tournament* ja *Half-Life.*

Epic Games julkaisi vuonna 1999 Unreal Tournament – pelin. Se loi maineensa vihollisten muokattavuudella ja taktikoinnilla. Vastustajien sanotaan olevan ihmispelaajien tasoisia ja ennennäkemättömän realistisesta tekoälyn luonnista vastasi Epic Gamesin ohjelmoija Steve Polge (Planet Unreal, 2011). Unreal Tournament -sarjaa pelataan vieläkin esim. suomalaisissa LAN-party tapahtumissa ja sillä on oma kannattajakuntansa.

Valve Software julkaisi Half-Life-pelin vuonna 1998 ja nosti samalla ns. räiskintäpelien rimaa korkeammalle kehittyneen tekoälynsä ansiosta. Peli oli toteutettu äärellisten tilakoneiden avulla ja sen avulla pystyttiin tekemään monimutkaisia toimintoja. Tilakoneen avulla pystyttiin luomaan vihollisjoukkioita, jotka liikkuivat omassa muodostelmassaan. Pelin viholliset kykenivät myös tarvittaessa antamaan suojatulta, sekä lisävoimia pelaajan energian vähentyessä ja tekemään väijytyksiä (Middleton 2002, 1.)

KUVIO 1. Half-Life – pelin vihollinen (Steam 2012).

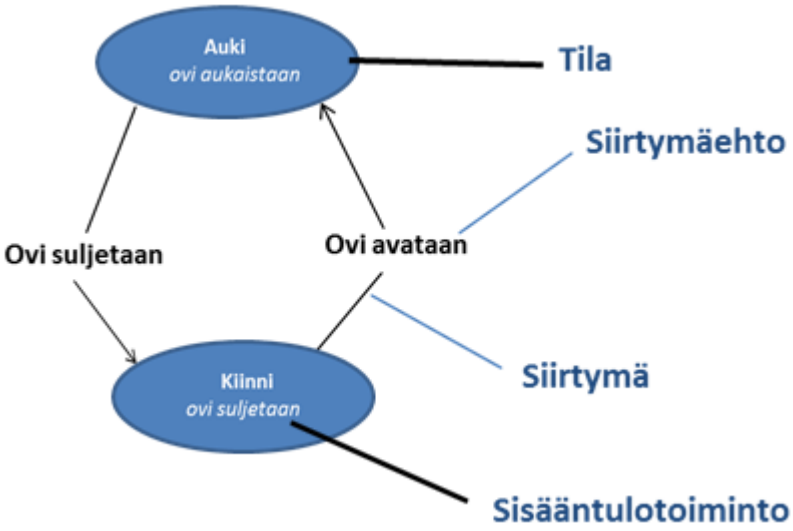
2.3 Äärellinen tilakone (FSM)

Äärellisellä tilakoneella (*Finite State Machine)* tarkoitetaan tila-automaattia, joka koostuu äärellisistä määristä erilaisia tiloja, sekä näiden välisistä tilasiirtymistä. Yksittäinen tilakoneen tila sisältää tiedot syötteen muutoksista. Tilasiirtymällä tarkoitetaan siirtymistä yksittäisestä tilasta toiseen tilaan (Holmström 2008, 1.)

Peliohjelmoinnissa käytetään äärellisiä tila-automaatteja, koska niiden avulla pystytään hallitsemaan pelinaikaista dynaamista toimintaa. Tila-automaatti suorittaa ohjelmakoodia, joka on liitettynä senhetkiseen tilaan tai tiettyyn siirtymään. Tilakoneiden avulla pystytään pilkkomaan suuria ja monimutkaisia ongelmia pienemmiksi osa-alueiksi. Sen toimintaa pystytään vaikuttamaan ulkopuolisten tapahtumien kautta, kuten esim. ajastimien, animaatioiden ja äänten avulla (Holmström 2008, 2.)

Tässä opinnäytetyössä käytetty tilakone on toiminnaltaan deterministinen. Deterministisessä tilakoneessa annetusta syötteestä ja senhetkisestä tilasta voidaan päätellä seuraava tilakoneen tila (Brownlee 2002, 1). Toisin sanoen, tilakoneen

käytöksen oppii seuraamalla sen toimintaa hetken aikaa, sillä se noudattaa aina tiettyä kaavaa tilasiirtymissä. Tämän huomaa erityisesti tietokonepelien loppuvastustajien käytöksessä, sillä ne on voitu ohjelmoida käyttäytymään toistuvasti samalla tavalla.



KUVIO 2: Tilakoneen toiminnan kuvaus (mukaillen Holmström 2008, 2).

Edeltävässä kuviossa on selitetty yksinkertaisesti äärellisen tilakoneen toimintaperiaate. Esimerkissä on kuvattu oven avaaminen tilakoneen toimintaperiaatteen mukaisesti. Ensin ollaan alkutilassa ”Auki”, jolloin ovi täysin auki. Kun ovi halutaan sulkea, siirrytään seuraavaan tilaan ”Kiinni” ja suoritetaan ”Kiinni”-tilan poistumistoiminnot. Kiinni-tilassa suoritetaan oven sulkemiseen tarvittavat toiminnot. Kun ovi halutaan taas aukaista, niin suoritetaan ”Kiinni”-tilan poistumistoiminnot ja vaihdetaan tilaksi ”Auki”. ”Auki”-tilassa suoritetaan avaamiseen tarvittavat toiminnot.

Itse tilat sisältävät erilaisia toimintoja, joita voidaan lisätä tilan ollessa aktiivisena. Tällaisia tiloja ovat esimerkiksi:

- Sisääntulotoiminto (entry)
- Poistumistoiminto (exit)
- Tilassa ollessa (do) (Laine 2008, 9.)

Sisääntulotoimintoja suoritetaan heti tilaan siirryttäessä, poistumistoiminto suoritetaan tilasta lähdettäessä ja tilassa ollessa suoritetaan koko ajan samaa haluttua tapahtumaa.

2.3.1 Mooren kone

Kuviossa 2 esitetty tilakoneen kuvaus on tyypiltään Mooren tila-automaatti. Mooren automaatissa tilakoneen toiminnot tapahtuvat tietyn tilan sisällä (Piipponen 2008, 18). Mooren tila-automaattia käytetään peliteollisuudessa paljon tekoälyn luomiseen, koska on luontevaa ajatella tekoälyn toimivan tietyllä tavalla tietyssä tilassa.

2.3.2 Agentti

Agentiksi kutsutaan tietokoneohjelman osaa, joka on tekoälyn ohjaama ja toimii itsenäisesti, sekä dynaamisesti omana ohjelmanaan. Sillä on sensoreita, jonka avulla se havaitsee toimintaympäristöään, toimilaitteita ja vaikuttaa ympäristöönsä (Domander 2012, 7.) Agenttia ei voida yksiselitteisesti määrittää yhdellä lauseella, vaan siitä vallitsee useita eri näkemyksiä. Tietokonepelien tekoälyssä agentti voi olla esimerkiksi autonominen, eli se toimii täysin itsenäisesti. Autonomisen agentin toimintaa voisi selkeästi kuvata seuraavalla lainauksella:

> Autonominen agentti on järjestelmä, on sijoitettu johonkin ympäristöön, havainnoi ja suorittaa toimintoja siinä, ajallisesti pitkäänkin, oman agendansa mukaan vaikuttaakseen siihen, mitä se havainnoi tulevaisuudessa (Franklin & Graesser 1996.)

Perinteisesti agentit ovat toimineet tietokonepelin vihollisen tehtävässä, mutta agentit voivat toimia myös pelaajaan sivullisina, kumppanina, yksikköinä, selostajina tai pelin ohjaajina (Domander 2012, 9). Yksinkertaistettuna agentti voi olla tietokonepelin vastustaja, joka on ohjelmoitu toimimaan tilakoneen avulla.

2.4 Sumea logiikka

Sumean logiikan periaatteena on, että sen totuusarvo vaihtelee nollan ja yhden välillä. Arvot voivat olla joko tosia, epätosia tai kummankin välimuotoja. Sumeaa logiikkaa voidaan käyttää tila-automaateissa siten, että eri tiloissa käytetty aika vaihtelee tilojen mukaisesti (Holmström 2008, 15.) Tällainen tilakoneen käyttäytyminen tuo tiettyä vaihtelevuutta ja oppivuutta tekoälyä käyttäville agenteille. Ne pystyvät oppimaan pelaajan käyttäytymistä ja mukauttamaan omaa toimintaansa sen mukaisesti. Peliteollisuudessa sumean logiikan käyttäminen on harvinaista, mutta sitä on käytetty esimerkiksi *S.W.A.T 2*- ja *Civilization: Call to Power*-peleissä.

S.W.A.T 2-pelissä sumeaa logiikkaa on hyödynnetty niin, että vihollispelaajat reagoivat tilanteen lisäksi myös oman vihollisyksikkönsä persoonan mukaisesti. Esimerkiksi hyökkäystilanteessa vihollisjoukkiolla on useita tiloja valittavanaan, kuten liikkuminen, suojaus ja erilaisia panttivankeja koskevia tiloja. Näiden vihollispelaajien toiminta ei siis ole sattumanvaraista, vaan ne ottavat huomioon pelaajan pelityylin ja päättävät sen perusteella sopivan reaktion (Johnson, D. & Wiles, J. 2012, 2.)

Civilization: Call to Power-pelissä pelaaja kohtaa runsaasti erilaisia vihollisyksiköitä, jotka ovat kulttuurillisesti erilaisia. Pelin tekijät loivat tätä varten ohjelmointikoodia säästääkseen tekoälymoottorin, jonka avulla vihollisten päätökset voitiin tehdä ryhmän persoonallisuuden perusteella. Erilaiset persoonallisuudet omaavat ryhmät erosivat toisistaan omilla päämäärillään (Johnson, D. & Wiles, J. 2012, 2.)

2.5 Motion Capture –tekniikka

Motion Capture (suom. liikkeenkaappaus) on tekniikka, jonka avulla voidaan kaapata ihmisen liikkeet digitaaliseen formaattiin. Tekniikan avulla kerättyä dataa voidaan käyttää digitaalisten hahmojen liikuttamiseen (Brotkin 2010.)

Tässä opinnäytetyössä liikkeenkaappausta käytettiin lähinnä vihollishahmojen hyökkäystä silmälläpitäen. Esimerkiksi vihollisten lyöminen on toteutettu liikkeenkaappausta hyödyntämällä kahden ryhmämme jäsenen toimesta. Liikkeenkaap-

pauksen jälkeen saatu data siirrettiin 3DsMax-ohjelmaan, jossa graafikot muuttivat saadun datan vihollisten kolmiulotteisten mallien liikkeiksi.

Tarkoituksena oli käyttää liikkeenkaappauksesta saatuja animaatioita pelissä, eikä perehtyä asiaan pintaa syvemmältä. Liikkeenkaappauksen avulla vihollisten hyökkäykset näyttävät tekoälyllisestä perspektiivistä katsottuna paljon aidommalta, kuin käsin työstetyt animaatiot. Tavoitteenani oli saada toimimaan nämä animaatiot myös Unity3D – ohjelmiston puolella.

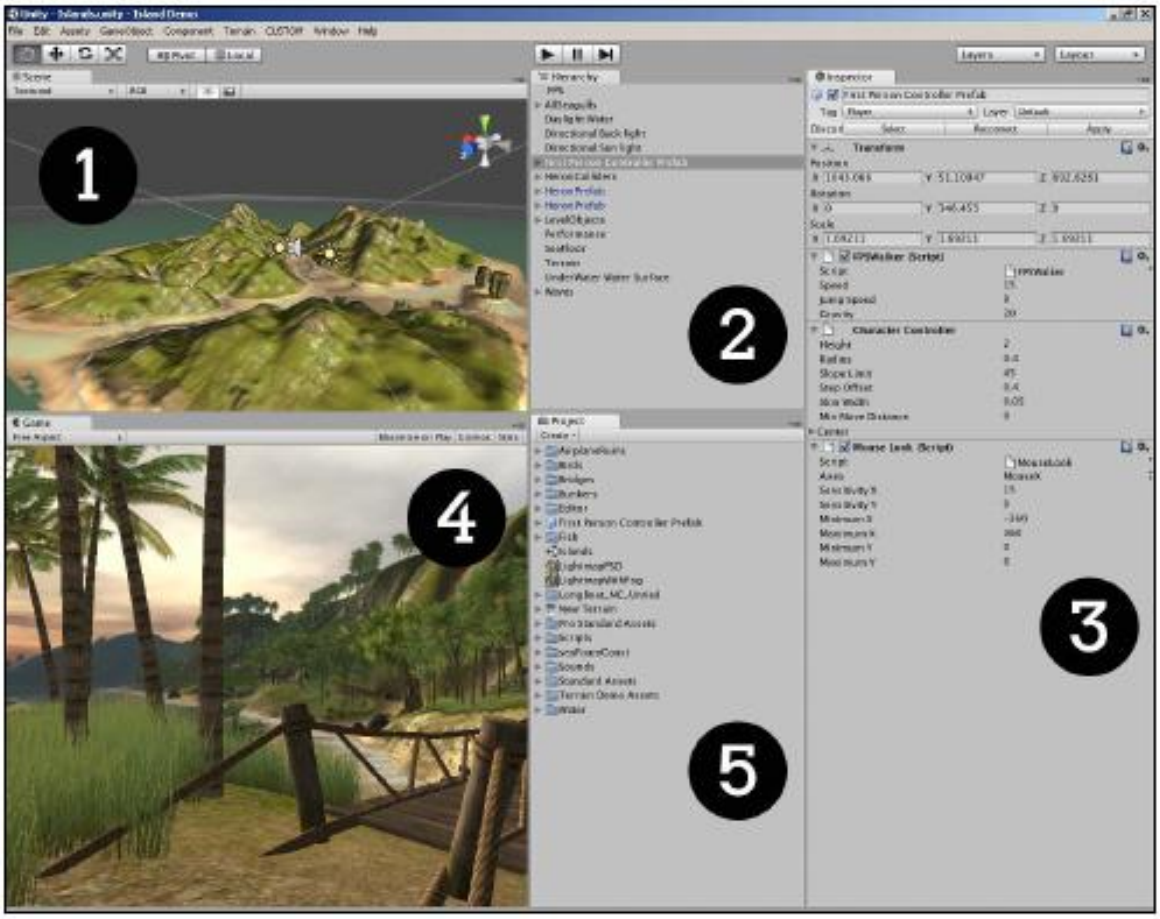
3 UNITY 3D –OHJELMISTO

Unity 3D on pelinkehittämiseen tarkoitettu laitteistoriippumaton alusta, jonka on kehittänyt Unity Technologies. Ohjelmiston avulla on mahdollista luoda monimutkaisiakin projekteja yllättävän näppärästi sen selkeän ja vaivattoman käyttöliittymän avulla. Unity:sta on saatavilla ilmainen versio ja maksullinen pro -versio. Unity:ssa luodun sovelluksen käyttäminen onnistuu myös Web Player-laajennuksen avulla. Unity tukee kolmea eri ohjelmointikieltä: C#, Javacsript ja Boo (Unity 3D 2012). Tässä opinnäytetyössä on käytetty Unity:n ilmaista versiota ja se on ohjelmoitu kokonaan käyttämällä C# -ohjelmointikieltä.

Unity:n näppäryys on huomattu laajalti muuallakin, koska se on saavuttanut jo miljoonan käyttäjän rajan ja ohjelmaa on ladattu maailmanlaajuisesti jopa 6 miljoonaa kertaa. Web Player – lisäosaa on asennettu 125 miljoonaa kertaa ja luku kasvaa koko ajan. Joka kuukausi latauksia tulee 5 miljoonalle uudelle tietokoneelle (Marketwire 2012.)

3.1 Käyttöliittymä

Unity:n peruskäyttöliittymä voidaan jakaa karkeasti viiteen osaan: *Scene*-ikkunaan, *Hierarchy*-ikkunaan, *Inspector*-ikkunaan, *Game*-ikkunaan ja *Project*-ikkunaan. Näiden ikkunoiden lisäksi yläreunassa on tarvittavat *Stop*-, *Play*- ja *Step* -painikkeet projektin kääntämistä varten.



KUVIO 3. Unity:n käyttöliittymä (Goldstone 2009, 17).

Scene-ikkuna

Unity:ssa luodut projektit koostuvat erilaisista sceneistä eli kohtauksista. Scene-ikkunassa näkyy aktiivisena oleva ”kenttä”. Scenet ovat tavallaan yksittäisiä tietokonepelin tasoja, joita yhdistelemällä peli saa rakenteensa. Ohjelmoija pystyy tämän ikkunan avulla siirtelemään kentällä olevia peliobjekteja (GameObject) haluamansa mukaan.

GameObjectit ovat tyhjiä objekteja, joihin käyttäjä saa lisätä haluamiaan ominaisuuksia. Tällaisia ominaisuuksia voivat olla esimerkiksi koodi, ääni, animaatio jne. Kaikilla GameObjecteilla on lähtökohtaisesti Unity:n oma *Transform*-komponentti, jonka avulla peliobjekteille voidaan asettaa pyörittämissuunta, skaalaus ja sijainti. Kaikkia näitä voidaan hallita x-, y- ja z-koordinaatistoon viitaten (Goldstone 2009, 15.)

Hierarchy-ikkuna

Tässä ikkunassa käyttäjä pystyy lisäämään kentälle esimerkiksi: kuutioita, kameroita, kolmiulotteista tekstiä jne.. Hierarchy-ikkunassa näkyvät kaikki senhetkisen scenen lisätyt objektit omana listauksenaan.

Inspector-ikkuna

Inspector-ikkunassa käyttäjä voi tarkastella eri objektien ominaisuuksia ja muokata niitä tarpeen mukaisesti. Esimerkiksi GameObjectiin lisätty koodi näkyy Inspector-ikkunassa omana rivinään ja koodin ominaisuudet on lueteltu allekkain.

Game-ikkuna

Game-ikkunassa käyttäjä näkee senhetkisen kentän (scenen) sisällön todenmukaisena, eli miltä peli näyttäisi käännettynä jollekin alustalle. Tässä ikkunassa käyttäjä ei pysty muokkaamaan omaa tuotostaan millään tavalla, vaan se toimii enemmän esikatselutilana.

Project-ikkuna

Project-ikkunassa on listattuna kaikki projektissa olevat tiedostot. Tiedostoja pystytään järjestelmään omiin alikansioihinsa, jolloin tiedonkäsittely helpottuu. Tässä ikkunassa pystytään myös luomaan mm. uusia skriptejä, animaatioita ja prefabeja.

3.2 Käytettävät ominaisuudet

3.2.1 Prefab

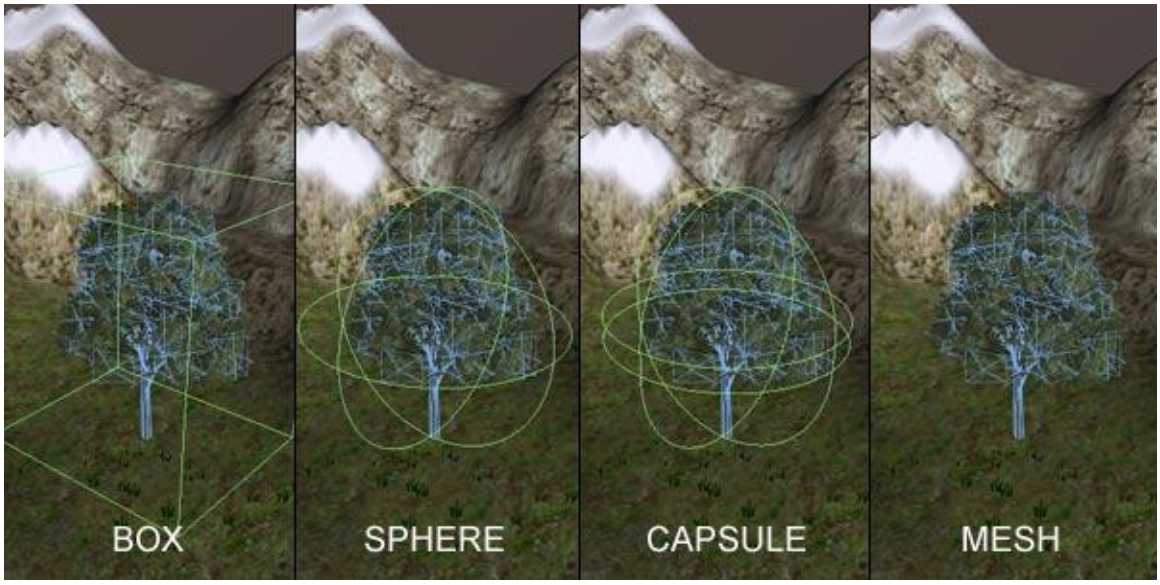
Prefabit ovat uudelleenkäytettäviä GameObjecteja, joita voidaan asettaa haluttu määrä Unity:n Sceneihin. Kun Prefab luodaan kentälle, niin samalla siitä tulee instanssi. Tämä tarkoittaa sitä, että kaikki Prefabin ominaisuudet linkittyvät toisiinsa ja ovat toistensa kopioita. Kun yhtä Prefabin ominaisuutta muutetaan, niin muutos näkyy kaikissa instansseissa. (Unity 3D 2012.)

Prefabit ovat erityisen hyödyllisiä vihollisten tekoälyä luotaessa, sillä niiden avulla pystytään tarkoin määrittelemään vihollisten lukumääriä ja samalla yksinkertaistamaan ohjelmointiprosessia.

3.2.2 Collider

Collider-komponenttia käytetään Unity:ssa kahden kappaleen välisten törmäysten tunnistukseen. Collidereita on käytettävissä viisi erilaista:

- Sphere Collider (pallo)
- Box Collider (laatikko)
- Capsule Collider (kapseli)
- Mesh Collider (verkko)
- Physic Collider



KUVIO 4. Colliderin tyyppejä (Stuttard Parker 2011).

Otetaan esimerkiksi pallo, joka vyöryy lattiaa pitkin ja törmää lopuksi seinään. Tässä tilanteessa molemmilla objektilla, sekä pallolla että seinällä on oma collide-rinsa. Tällaisissa kahden objektin törmäyksissä lähetetään kolme viestiä eteen-päin, joita voidaan hyödyntää kirjoitettavassa ohjelmakoodissa. (Unity 2012). Näitä viestejä on kolme kappaletta: InCollisionEnter, OnCollisionStay ja OnCollisionExit. Ensimmäisessä tapauksessa tapahtumat suoritetaan törmäyksen sattuessa, toi-sessa, kun törmäyksessä pysytään ja viimeisessä tapauksessa törmäyksen lop-puessa. Esimerkissä mainittu pallo voidaan laittaa tuhoutumaan tai toimimaan NVIDIA PhysX -fysiikkamoottorin mukaisesti.

Fysiikkamoottorin käyttäminen vaatii Rigidbody -komponentin käyttämistä. Tässä opinnäytetyössä ei käytetä fysiikkamoottorin toimintoja, mutta Colliderit yleensä vaativat vähintään toiseen objektiin Rigidbodyn käyttämistä varman toiminnan takaamiseksi. Colliderien pääasiallinen tarkoitus tässä työssä oli tunnistaa törmäyksiä, eikä luoda realistiselta näyttävää fysiikkaa.

3.2.4 Trigger

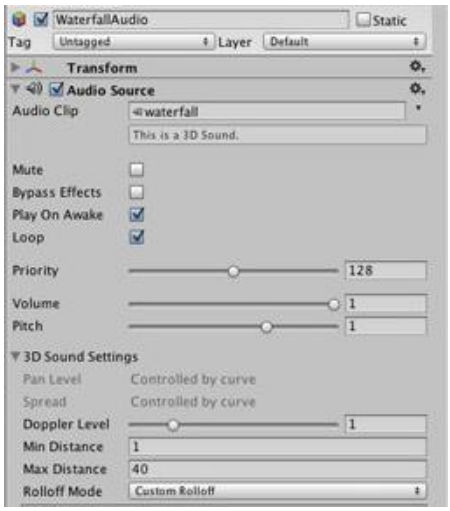
Collidereissa on olemassa *IsTrigger* -ominaisuus, jota käyttämällä valittu peliobjekti voidaan muuntaa toimimaan triggerinä. Triggereitä käytetään yleensä, kun halutaan suorittaa tiettyä toiminnallisuutta tarkasti määritellyssä kohdassa. Esimerkiksi kentän läpäiseminen voidaan toteuttaa triggerin avulla kätevästi. Pelaaja pääsee kentän loppuun, osuu triggeriin ja trigger lähettää viestinä eteenpäin kentän vaihdoksen. Trigger on siis nimensä mukaisesti liipaisin, joka lähettää haluttaessa kolmea viestiä eteenpäin törmäyksen sattuessa. Näitä viestejä ovat *OnTriggerEnter, OnTriggerStay* ja *OnTriggerExit.*

OnTriggerEnter -tapahtumaa käytetään, kun pelaaja osuu liipaisimeen ja sen jälkeen suoritetaan tietyt toiminnot. *OnTriggerStay* -tapahtumassa suoritetaan toiminnallisuutta vain silloin, kun pelaaja pysyy liipaisimen rajoittamalla alueella. *OnTriggerExit* -tapahtumassa toiminnallisuutta suoritetaan vain silloin, kun pelaaja poistuu liipaisimen alueelta.

Tässä työssä on hyödynnetty liipaisimen kätevyyttä yhdistämällä ne edellä mainittuun prefabiin. Triggerin ja Prefabin yhteistoiminnalla voidaan varmistaa pelissä olevan toiminnallisuuden tapahtuminen halutuissa kohdissa.

3.2.5 Äänet

Unity:ssa äänien lisääminen on tehty varsin helpoksi, lisätään vain GameObjectiin *Audio Source* eli *äänilähde*. Sen jälkeen ääniä voidaan vapaasti käyttää haluamallansa tavalla valitsemalla haluttu ääni Audio Clip -kohdasta tai ohjelmoinnin avustuksella.



KUVIO 5. Äänten asetukset (Unity 3D 2012.)

Unity:ssa ääniä pystytään jonkin verran käsittelemään Audio Source -näkymän kohdalta, mutta suuremmat editoinnit tulee jättää jonkin toisen ohjelman hoidettavaksi. Unity kykenee toistamaan äänet myös kolmiulotteisesti eli 3D-ääninä. Listataan seuraavaksi tärkeimmät käytettävät äänten ominaisuudet.

Ominaisuus	Tehtävä
AudioClip	Soitettava ääniklippi
Mute	Ääni mykistetään
Bypass Effects	ByPass-suodatin
Play On Awake	Ääni toistetaan heti scenen alussa
Loop	Toistaa ääniklippiä koko ajan
Volume	Äänenvoimakkuus
Pitch	Äänenkorkeus
3D Sound Settings	3D-äänten asetuksia

TAULUKKO 1. Äänilähteen ominaisuuksia (Unity 3D 2012.)

Koodillisesti äänen toistaminen on varsin yksinkertaista.

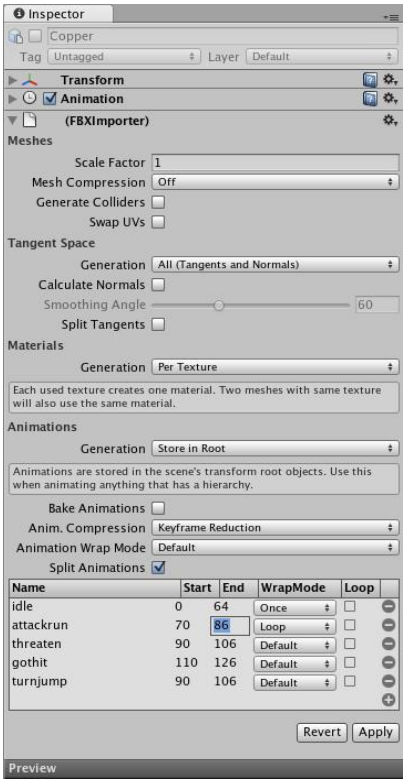
```
audioClip aani;

  void Start() {
        aani.Play();
   }
```

Ensin määritellään toistettavan äänen nimi, *aani.* Sitten toistetaan haluttu ääni komennolla *Play.* Ja äänen voi halutessaaan pysäyttää perinteisillä *Stop*- ja *Pause*-komennoilla.

3.2.6 Animaatiot

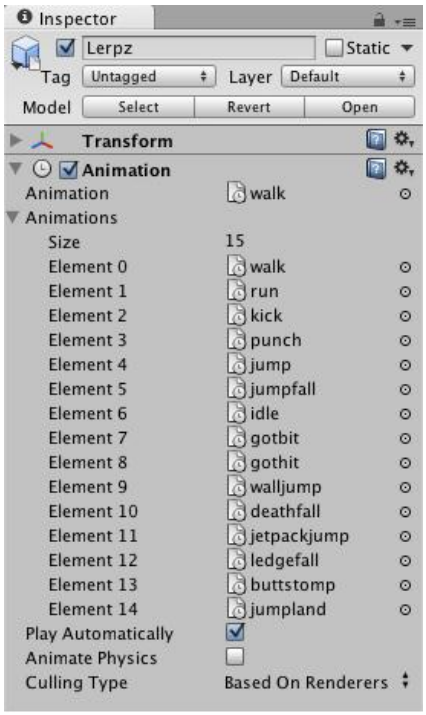
Unity kykene pilkkomaan animaatiot osiin luoduista fbx -tiedostoista. Fbx- tiedostot sisältävät 3D-mallin ja animaatiot. Yksi 3D-malli sisältää yhden pitkän animaation, joka voidaan jaotella toiminnan mukaan pienempiin osiin. Esimerkiksi kävelylle, juoksemiselle ja hyppimisille voi olla oma animaationsa. Kävely voi tapahtua kohdassa 0 – 42, juokseminen kohdassa 43- 47 ja hyppiminen 48 - 52.



KUVIO 6. Animaation jakaminen

Kuviossa näkyy selkeästi eri animaatioiden nimet, aloitus- ja lopetuspisteet, animaation toistamistapa ja ikuinen toisto. Esimerkiksi kävelylle on syytä asettaa kuviossa näkyvstä WrapMode -valikosta toistotavaksi Loop. Tällöin kävelyanimaatiota toistetaan ikuisesti, eikä pelaajahahmon kävely näytä epäluonnolliselta.

Jotta animaation muutokset tulevat voimaan, on animaatio-objekti lisättävä myös GameObjectiin ja Animation-valikkoon juuri määritetyt animaatioklipit. Tämän jälkeen jaettuja animaatiota pystytään kutsumaan suoraan koodista.



KUVIO 7. Animaation asetukset (Unity 2012.)

Animation-valikossa ensimmäisenä valitaan oletusanimaatio, jota toistetaan heti ohjelman käynnistyessä. Seuraavaksi valitaan Animations-taulukon koko, esim. 15, ja tähän taulukkoon valitaan kaikki käytettävät animaatiot. Sitten valitaan Play Automatically-painike, joka varmistaa animaation automaattisen toistamisen pelin ollessa käynnissä. Fysiikkaa voidaan myös mallintaa Animate Physics- painikkeella, mutta tässä opinnäytetyössä kyseistä ominaisuutta ei käytetty ollenkaan.

Ominaisuus	Tehtävä
Animation	Oletusanimaatio
Animations	Lista animaatiosta, joihin pääsee käsiksi koodin kautta.
Play Automatically	Toistaa animaation automaattisesti
Animate Physics	Fysiikan toistaminen
Culling Type	Määrittää toistoajankohdan

TAULUKKO 2. Animaation ominaisuudet (Unity 2012.)

Animaatioita voidaan käyttää ohjelmoinnin avulla esimerkiksi näin:

```
if(!this.animation.IsPlaying("kavely"))
        {
            this.animation["kavely"].speed = 0.9f;
            this.animation.Play(”kavely”);
        }
```

Eli jos animaatiota ei toisteta tällä hetkellä, niin käynnistetään kävelyanimaatio. Kävelyanimaation tilalle voidaan sijoittaa mikä tahansa jaetuista animaatiosta. Niiden toimintaa voidaan myös muokata, esimerkiksi muuttamalla toistonopeutta nopeammaksi tai hitaammaksi käyttämällä speed -ominaisuutta ja syöttämällä jonkin suhdeluvun. Normaalinopeus on luku 1.

4 STEAMDROID-PELI

4.1 Pelin juoni

Eletään vuotta 1983 ja kylmä sota on muuttunut lämpimämmän sodan kaltaiseksi. Ihmiskunta on selviytynyt juuri ja juuri sukupuuton partaalta ydinaseiden aiheuttamien tuhojen tuhkissa. Ydinaseiskusta selvinneet ihmiset taantuvat takaisin kivikauden aikaiselle tasolle, paitsi Iso-Britannia, johon tähdätty ydinase osuikin harhaan. Tämän johdosta brittiläiset lähtevät valloittamaan maailmaa täysin ylivoimaisella ja edistyksellisellä teknologiallaan. Vuosikymmeniä tämän kaiken jälkeen aletaan huhuilla kapinasta erämaaksi muuttuneella alueella Texasissa, Yhdysvalloissa. Britannian salainen palvelu lähettää yhden parhaimmista agenteistaan, Britbot VI:n, hankkiutumaan eroon kyseisestä kapinasta. Hän suunnistaa kohti paikkaa, joka nykyisin tunnetaan nimellä ”Uusi Britannia”.

Britbot matkustaa laivalla meren yli kohti entistä Texasin osavaltiota ja suunnistaa siellä tuiki olemattoman pieneen kaupunkiin – keskellä ei mitään. Hän etsii sieltä yhteyshenkilöään, baarimikkoa, joka majailee omassa ”Vierivä ankka”- nimisessä kantapaikassaan. Baarimikko neuvoo Britbotia aloittamaan tehtävänsä tuhoamalla erään Henry Maydullboardin, tunnetun kapinallisjohtajan. Tiedosta viisastuneena Britbot suuntaa empimättä etsimään tätä ”hevihemmoa” ja tuhoamaan hänet lopullisesti.

4.2 Ohjelmointi

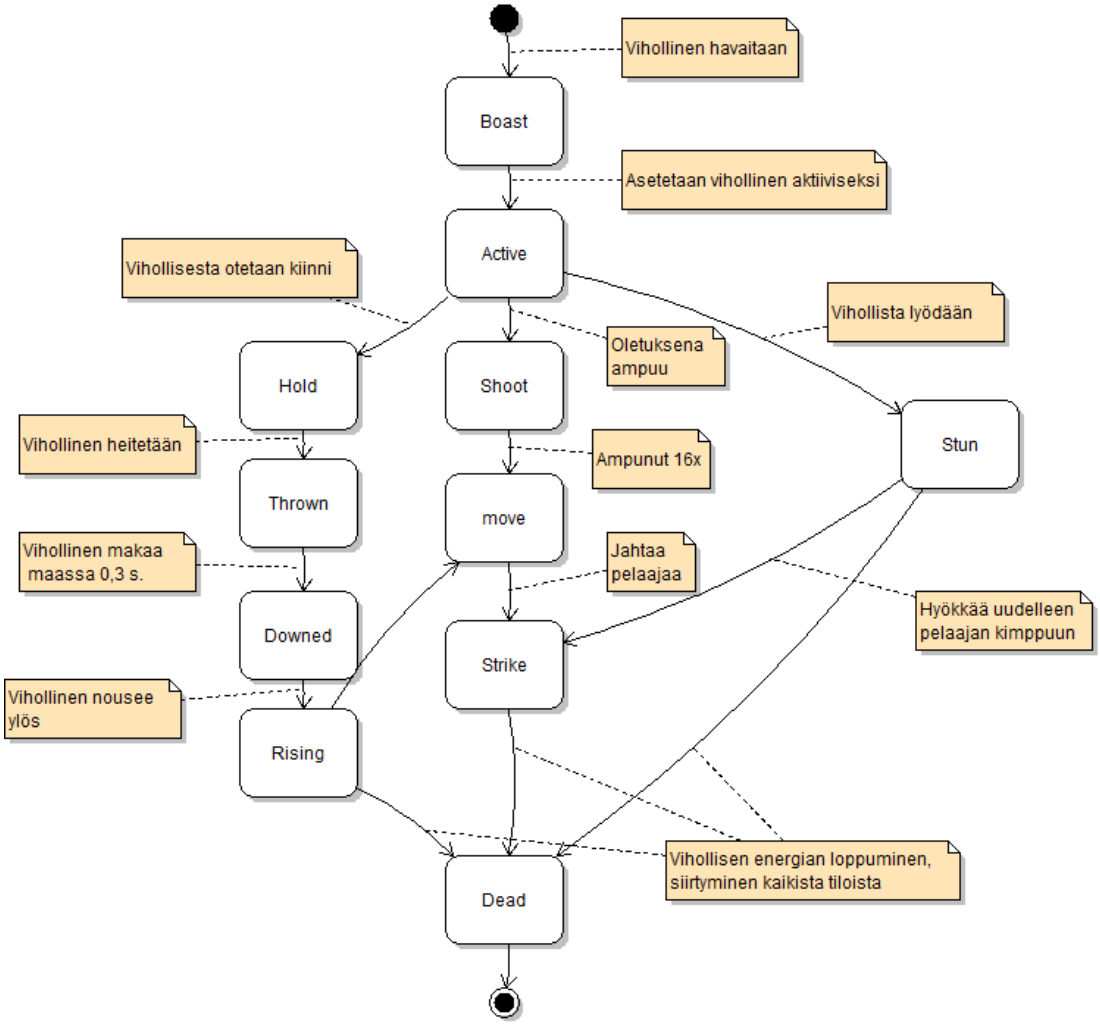
Päähenkilö Britbot VI:n matkaa hidastamassa on kolmea erilaista vihollista ja heidät tunnetaan paremmin nimillä: Natural-Twenty Paul, Länkkäri Bill ja Henry Maydullboard. Kaikilla vihollisilla on omat eritystaitonsa Britbotin eliminoimiseksi ja ne on ohjelmoitu toimimaan kunkin oman äärellisen tilakoneen mukaisesti, noudattaen tiettyjä tilasiirtymiä. Esittelen seuraavaksi viholliset em. järjestyksessä hahmoesittelyn, tilakaavion, koodin ja selitysten kanssa. Koodiesittely tehdään samassa järjestyksessä vihollisten tilakaavioiden tilasiirtymien kanssa, jotta kaaviota on helpompi seurata.

4.2.1 Natural-Twenty Paul

Natural-Twenty Paulia voi kuvailla vain yhdellä sanalla – rento. Hyökkäystilanteessa Paul kohottaa aseensa rauhallisesti pelaajaa kohti, ja tämän jälkeen ammuskelee. Jonkin ajan kuluttua hän siirtyy jahtaamaan pelaajaa ja aiheuttaa lähietäisyydellä tuhoa nyrkeillään.

Paulin tilakone koostuu *FSM_NTPaul*-luokasta, *FSM_NTPaulPrefab:*sta, *FSM_NTPaulGunPrefab*:sta, äänistä, sekä animaatioista.

KUVIO 8. Natural-Twenty Paul



KUVIO 9. Natural-Twenty Paulin tilakaavio

Paulilla on olemassa 11 erilaista tilaa, jotka ovat: *Boast, Active, Stun, Downed, Strike, move, Thrown, Hold, Rising ja Dead.* Aivan aluksi pelaajan havaittuaan Paul on Boast tilassa.

```
void boast()
    {
        this.animation.Play("idle");
        transform.LookAt(new Vector3(playeri.position.x, 0,
playeri.position.z));
        int spottausaani = Random.Range(1,4);

        switch(spottausaani)
        {
            case 1:
                if(vihuHuomattuAluksi== false)
                {
                    vihuHuomattuAluksi = true;
                    audio.clip = hey1;
                    audio.Play();
                }
                    break;
        }
```

```
            if(!audio.isPlaying)
            {
                _tila = Tila.Active;
            }
        }
```

Boast -tilassa ollessa toistetaan aluksi animaatio ”idle”. Idle -animaatiossa Paul hieman huojuu paikallaan ja näyttää tarkkailevan pelaajaa. Seuraavaksi määrätään Paul seuraamaan pelaajaa silmä tarkkana, mihin tahansa hän liikkuukin. Tämä hoidetaan tranform.LookAt:n avulla. Ilmoitetaan Unity:lle pelaajan sijainti x-, y- ja z-koordinaattien mukaisesti em. koodissa 4. rivillä näkyvässä lausekkeessa.

Tarvitaan myös ns. huomausääni, joka ilmoittaa pelajaalle Paulin huomanneen hänet. Sitä varten on luotu satunnaiskokonaisluku *spottausaani,* joka saa arvoja välillä 0 – 4. Tämän jälkeen käydään ääni läpi switch-case-rakenteessa, jossa valitaan ääni kahdesta erilaisesta huomausäänestä. Luvuilla 1 tai 2 toistetaan *hey1*-ääni. Kun taas luvuilla 3 tai 4 toistetaan ääni *hey2.* Tällä tavoin saadaan vaihtelevuutta Paulin sanomisiin huomaustilanteessa. Alkuperäisessä koodissa case -kohtia on 4 kappaletta. Lopuksi, kun mikään ääni ei enää soi, vaihdetaan tilaksi *Active.*

```
        void toactive()
        {
            transform.LookAt(new Vector3(playeri.position.x, 0,
playeri.position.z));
            _tila = Tila.Shoot;
        }
```

Active – tilassa asetetaan Paul taas seuraamaan pelaajan liikkeitä automaattisesti ja tässä tilassa valitaan vain seuraava aktiivinen tila. Tästä siirrytään seuraavaksi tilaan *Shoot*.

```
        void shoot()
        {
            transform.LookAt(new Vector3(playeri.position.x, 0,
playeri.position.z));
            if(Time.time > ( lastFireTime + burstDelay) )
            {
                this.animation.CrossFade("ase_nosto");
                this.animation.CrossFade("ampuminen");
                audio.clip = revolveri;
                audio.Play();
                vihuAmpuu = true;
                lastFireTime=Time.time;
                Instantiate(FSM_NTPaulGunPrefab, trans-
form.position, transform.rotation);
```

```
            PaulLuotiLaskuri++;
        }

        if(PaulLuotiLaskuri > 16)
            _tila = Tila.move;
    }
```

Shoot -tilassa seurataan jälleen aluksi pelaajan liikkeitä, jotta luodit lentäisivät oikeaan suuntaan. Seuraavaksi if -lauseen sisällä verrataan viimeksi ammutun luodin, viiveen ja ajastimen avulla sopiva tauko ammusten välille. Sitten luodaan animaation *Crossfade* -ominaisuuden avulla kahden animaation välinen sulava siirtyminen. Animaatiossa ”ase_nosto” Paul kohottaa aseen kerran ilmaan ja ”ampuminen”- animaatiossa se jää tekemään ampumisliikettään.

Valitaan toistettavaksi ääniklipiksi revolverin oma ääni ja asetetaan vihuAmpuutotuusarvo todeksi. Tämän totuusarvon avulla ampuminen pystytään varmasti lopettamaan oikeissa tilanteissa. Sitten luodaan *FSM_NTPaulGunPrefab*, jonka avulla luoti voidaan luoda uudestaan ja uudestaan. Seuraavaksi kasvatetaan *PaulLuotiLaskuria* jokaisen ammutun luodin jälkeen. Kun laskuri saavuttaa suuremman arvon kuin 16, niin tila vaihdetaan *move*-tilaan. Tarkastellaan seuraavaksi ennen move -tilan tarkastelua Paulin aseen ohjelmakoodia, sillä sen toiminta liittyy olennaisesti tilakoneeseen.

```
void Update () {
        if(Vector3.Distance(transform.position,Vihu2.position)
> 15)
        {
            Destroy(this.gameObject);
        }

        transform.Translate(Vector3.forward * Time.deltaTime *
luodinNopeus);
    }

    void OnTriggerStay(Collider otherObject)
    {
        if (otherObject.tag == "Player")
        {
            storedData.SendMessage("ApplyDamage", damage2,
SendMessageOptions.DontRequireReceiver);
        }
     }
```

Ensimmäiseksi if-lauseen sisään on laitettu välimatkavektori, joka laskee etäisyyttä pelaajaan. Kun välimatka kasvaa tarpeeksi suureksi, niin luoti tuhotaan. Tämä sen takia, ettei luoti lennä naurettavan kauas kentällä. Transform.Translate:lla luoti

pistetään liikkumaan halutulla nopeudella suhteessa forward-vektoriin ja käytettyyn aikaan ja luodinNopeus-kertoimeen. Luotiin on lisätty trigger, jonka avulla tunnistetaan luodin osuminen pelaajaan. Pelaajalle on annettu ”Player”-tunniste, jonka avulla luoti tunnistaa osumisen. Kun luoti osuu pelaajaa, niin SendMessage-komennolla lähetetään ”damage2”-arvo, joka on ennalta määrätyn suuruinen. Tällä tavalla pelaajan energiat vähenevät.

```
    void tomove()
    {
        if(Time.timeScale !=0)
        {
          this.animation.Play("kavely");
          transform.LookAt(new Vector3(playeri.position.x, 0,
playeri.position.z));

          if(valimatka > maksimiEtaisyys )
          {
             transform.Translate(Vector3.forward / vi-
hun_nopeus,Space.Self);
          }

          if(_tila != Tila.Rising)
          {
             vihuSelallaan_timer = 1.8f;
             RisingTimeri = 0.8f;
          }
        }
    }
```

Move-tilassa on määritetty timeScale:lle arvoksi nolla, koska hahmon on pysähdyttävä Pause-nappia painettaessa. Sitten toistetaan animaatio ”kavely”, jotta Paul saadaan kävelemään. Tämän jälkeen lisätään pelaajan seuraaminen. Paul on suunniteltu kävelemään tietyn välimatkan päästä, eli jos etäisyys on pienempi kuin maksimietäisyys. Vihun_nopeus määrittää Paulin nopeuden ja vektorilla määritetään suunta. Lopuksi nollataan *Rising*-tilassa tarvittavia laskureita, joista kerron myöhemmin.

```
    void strike()
    {
        if(Time.timeScale !=0)
        {
        transform.LookAt(new Vector3(playeri.position.x, 0,
playeri.position.z));

        if(!this.animation.IsPlaying("lyonti"))
        {
            this.animation.Stop();
        }
            this.animation.Play("lyonti");
```

```
                this.animation["lyonti"].speed = 2.5f
                storedData.SendMessage("ApplyDamage", damage,
SendMessageOptions.DontRequireReceiver);
            }
        }
```

Strike-tilassa varmistetaan aluksi toisen animaation lopettaminen, mikäli se on eri kuin lyönti-animaatio. Kun edellinen animaatio on pysäytetty, niin toistetaan lyontianimaatio 2,5-kertaisella nopeudella. Tällä tavoin lyöntiin saadaan kaivattua voiman tuntua. Muutoin tilassa olevat pelaajan seuraamiset, energian vähenemiset ja pelin pysäyttämiset on tehty samalla tavalla toisten tilojen kanssa.

Kaikki tämä suoritetaan silloin, kun Paulia ei keskeytetä mitenkään. Seuraavaksi käydään läpi tilasiirtymät, kun pelaaja on lyönyt sitä. Tämä aiheuttaa aluksi tilasiirtymän jähettymiseen, *Stun*-tilaan. Tällöin Paul ei voi puolustautua, eli lyödä takaisin.

```
void stun()
{
    this.animation["pataan"].speed = 2.5f;
    this.animation.Play("pataan");

    if(StunnissaTimer<=0)
    {
        this.animation.Stop();
        _tila = Tila.move;
    }
}
```

Stun-tilassa toistetaan ”pataan”-animaatio 2,5-kertaisella nopeudella, jossa Paul hieman heilahtaa taaksepäin pelaajan iskun voimasta. Paul on keskeytyneenä 0,3 sekuntia *StunnissaTimer*:n arvon mukaisesti. Kun *StunnissaTimer*:n arvon suurempi tai yhtä suuri kuin nolla, niin animaation toistaminen pysäytetään ja tilaksi muutetaan move-tila. Eli Paul ryhtyy jälleen jahtaamaan pelaajaa. Jos Paul on tarpeeksi lähellä pelaajaa, niin tilaksi muutetaan tilakaavion mukaisesti Strike-tila.

Mikäli pelaaja haluaa heittää Paulia, niin tilakoneen tila muuttuu mistä tahansa tilasta Hold-tilaan. Tässä tilassa Paulista pidetään kiinni rinnuksista, eli hieman koholla maasta. Paul katsoo koko ajan pelaajaa kohti – heittosuunnasta riippumatta.

```
    void hold()
    {
        this.animation.Play("kiinni");
        this.transform.position = new Vec-
tor3(dmgcollider.position.x, 0, dmgcollider.position.z);
        transform.LookAt(new Vector3(playeri.position.x, 0,
playeri.position.z));
        if (Attack.kiinniCheck <= 0) {
            _tila = Tila.move;
        }
      }
```

Hold-tilassa toistetaa aluksi ”kiinni”-animaatio, jossa Paul on hieman takakenossa. Sitten Paulin sijaintia muutetaan *transform.position*-komennon avulla hieman koholla maan pinnasta. Sen jälkeen laitetaan Paul tuijottamaan pelaajaa silmiin heittosuunnasta riippumatta. Lopuksi Paul lähtee pois tästä tilasta, mikäli pelaaja ei heitä Paulia mihinkään suuntaan tietyn ajan kuluessa. Tällöin tilaksi vaihdetaan move ja Paul lähtee jahtaamaan pelaajaa. Heiton onnistuessa siirrytään tilaan *Thrown.*

Thrown-tilassa toistetaan animaatio heittämisestä, eli Paul on käytännössä aivan vaakatasossa. Tämän tilan jälkeen Paul jää maahan makaan ja tilaksi muutetaan *Downed.*

```
void downed()
    {
        vihuSelallaan_bool = true;

        if(vihuSelallaan_timer <=0)
        {
            vihuSelallaan_bool = false;
            vihuSelallaan_timer = 0.0f;
            _tila = Tila.Rising;
        }
    }
```

Downed-tilassa *vihuSelallaan_bool*-totuusarvo asetetaan todeksi ja jos *vihuSellaan_timer*-ajastin menee umpeen (1,8 sekuntia), niin em. totuusarvo asetetaan epätodeksi. Myös em. ajastin asetetaan nollaksi ja tämän jälkeen Paul nousee ylös maasta. Tilaksi vaihdetaan *Rising.*

```
void rising()
    {
        vihuNouseeYlos = true;
        this.animation.Play("ylos");
```

```
            if(RisingTimeri <=0)
            {
                vihuNouseeYlos= false;
                RisingTimeri = 0.0f;
                _tila = Tila.move;
            }

        }
```

Rising-tilassa *vihuNouseeYlos*-totuusarvo asetetaan todeksi ja toistetaan nousemisanimaatio. *RisingTimeri*-ajastimen mennessä umpeen asetetaan totuusarvo epätodeksi ja nollataan ajastimen arvo. Lopuksi vaihdetaan tilaksi move-tila ja tilakone palautuu alkuasetelmaan.

Mikäli Paulin energiat loppuvat eli pelaaja saa tuhottua hänet, niin mistä tahansa tilasta siirrytään Dead-tilaan.

```
void dead()
{
  if(curHp <= 0)
   {
      this.animation.Play("kaatuu");

       if(vihuDeadTimer<=0)
       {
           this.animation.Stop();
           Destroy(this.gameObject);
        }
    }
}
```

Dead-tila on laitettu kokonaan if-lausekkeen sisään, joka sisältää Paulin erergiatilaa kuvaavan muuttujan *curHp.* Mikäli se on pienempi tai yhtä suuri kuin nolla, niin toistetaan ”kaatuu”-animaatio. Ja mikäli *vihuDeadTimer*-ajastimen arvo on nolla, niin lopetetaan animaation toistaminen ja tuhotaan Paul pois kentältä.

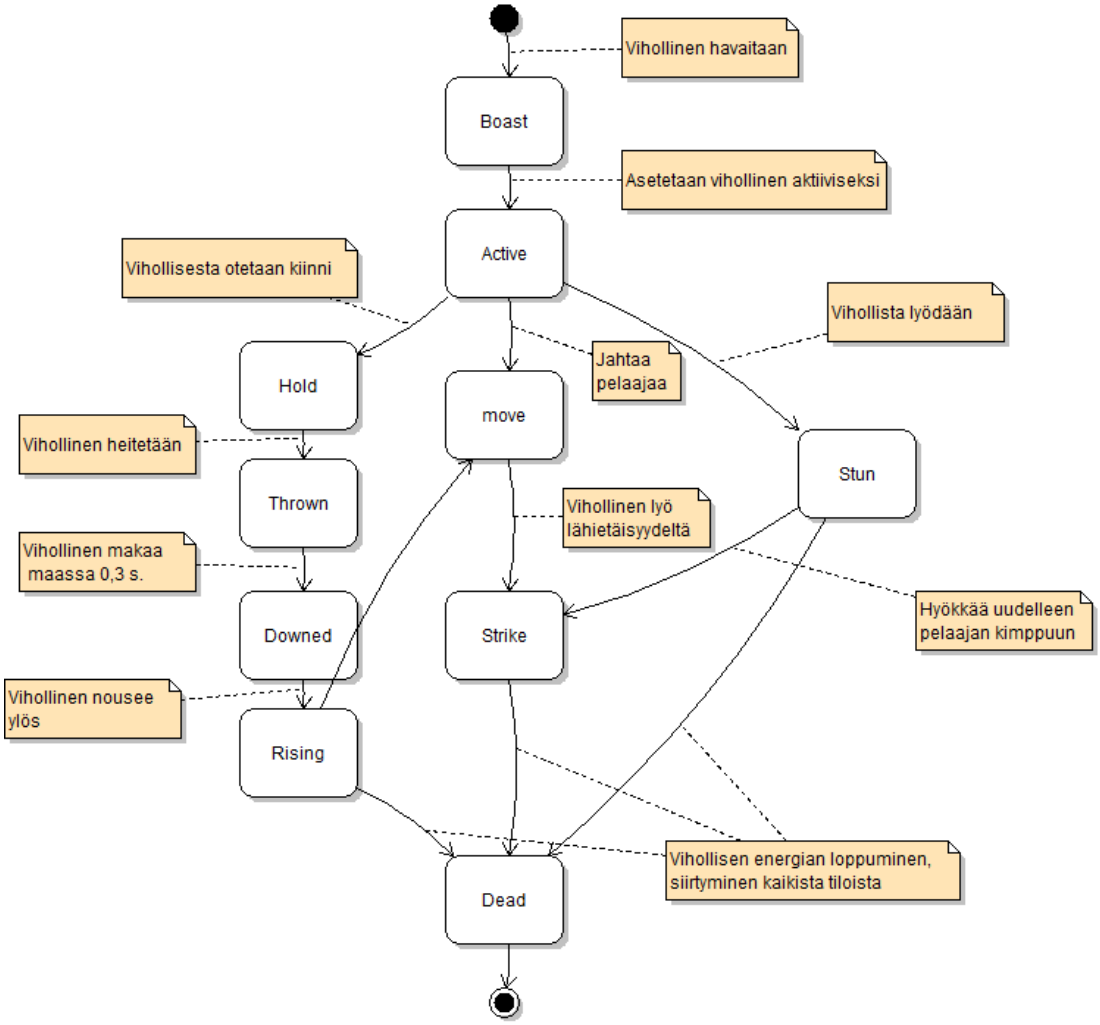
4.2.2 Länkkäri Bill

Länkkäri Bill on Paulia hieman brutaalimpi pelihahmo, sillä hänen ainoa aseensa on alkukantainen pamppu. Bill hyökkää pelaajaa kohti huitoen pampullaan sinnikkäästi, eikä luovuta ennen pelaajan tuhoutumista.

Billin tilakaaviosta voidaan havaita sen samankaltaisuus Paulin tilakoneen kanssa. Tämän vuoksi ei ole syytä käydä Billin toiminnallisuutta koodillisesti läpi, sillä ainoat eroavaisuudet ovat animaatioiden nimeämisissä ja Shoot-tilan puuttumisessa.

Billin tilakone koostuu *FSM_Bill-luokasta*, *FSM_BillPrefab:sta, animaatioista ja äänistä.*

KUVIO 10. Länkkäri Bill



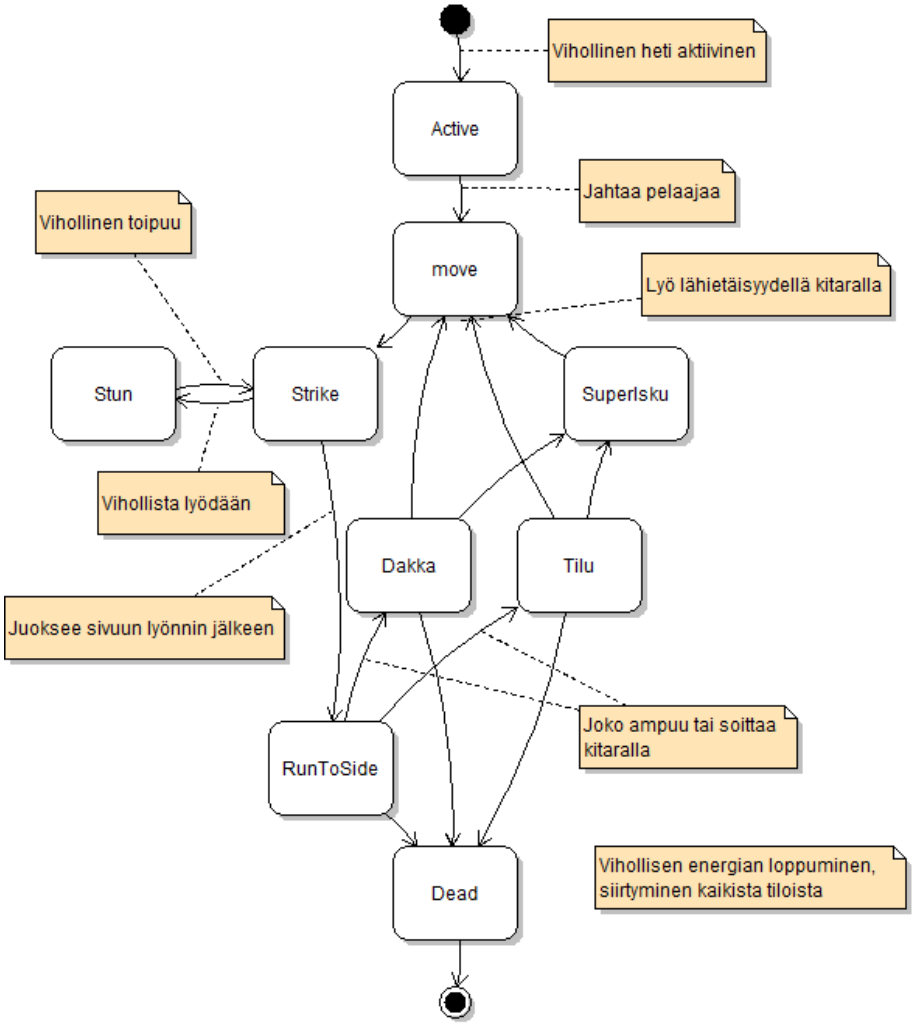
KUVIO 11. Länkkäri Billin tilakaavio

4.2.3 Henry Maydullboard

Henry “Hevihemmo” Maydullboard on pelätty kapinallisjohtaja, jonka pelkkä ulkonäkö aiheuttaa useimmissa pelaajissa vilunväristyksiä. Henry on pelin päävastustaja, joka on tuhottava pelin läpäisemiseksi. Henryn aseena toimii kitara, jonka avulla hän pystyy ampumaan sarjatulta, sekä soittamaan kohtalokkaan soolonsa. Soolon jälkeen hän tekee ns. superiskun, joka pudottaa roimasti pelaajan käytettävissä olevaa energiaa.

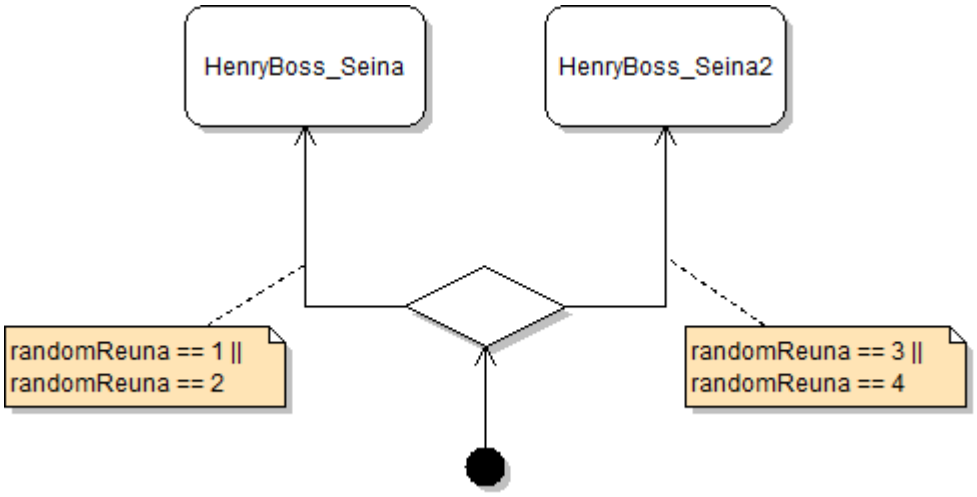
KUVIO 12. Henry ”Hevihemmo” Maydullboard

Henryn tilakone koostuu *FSM_HHBoss-luokasta, FSM_HHBossPrefab:sta, FSM_HHBossGunPrefab:sta, FSM_HHBossGunPrefabScript-luokasta,* animaatioista ja äänistä.



KUVIO 13. Henryn tilakaavio

Henrylla on yhteensä 9 erilaista tilaa: *Active, move, SuperIsku, Tilu, Dakka, RunToSide, Strike, Stun ja Dead.* Koska kentällä on vain yksi loppuvastustaja, niin Henry ei välttämättä tarvitse Boast-tilaa havaitsemiseen. Se on aivan aluksi Active-tilassa. Active-tilasta se siirtyy oletuksena move-tilaan jahtaamaan pelaajaa samalla tavalla kuin muutkin viholliset. Kun Henry saapuu tarpeeksi lähelle pelaajaa, niin se alkaa lyödä. Tällöin siirrytään Strike-tilaan. Mikäli Henrya lyödään takaisin, se heilahtaa hieman taaksepäin eli menee Stun-tilaan. Tähän asti Henryn tilakoneen toiminta on samanlaista kuin aiemmin, mutta eroavaisuudet alkavat, kun se juoksee sivuun tulittamaan vihollista kitarakonekiväärillään. Juostessa mennään RunToSide-tilaan.



KUVIO 14. Henry juoksee reunaan

RunToSide-tilassa Henry juoksee jompaankumpaan reunaan kentällä, riippuen randomReuna kokonaislukumuuttujan arvosta. Se saa arvoja väliltä 1 – 4 satunnaisessa järjestyksessä, jotta Henryn toiminta ei olisi ennalta arvattavaa. Yllä olevasta kuviosta havaitaan, että Henry juoksee ylempään reunaan arvoilla 1 ja 2. Ja alempaan reunaan arvoilla 3 ja 4.

```
void runtoside()
    {
        if(Time.timeScale!=0)
        {
            VihuReunassa = true;

            this.animation.Play("juoksu");

            if(VihuReunassa)
            {
                if (_tila == Tila.RunToSide &&
transform.position == kohde.position && randomReuna == 1)
                {
                    kohde =
GameObject.Find("HenryBoss_Seina").transform;
                    transform.LookAt(new
Vector3(kohde.position.x, 0, kohde.position.z));
                }

                else if (_tila == Tila.RunToSide &&
transform.position == kohde.position && randomReuna == 2)
                {
                    kohde =
GameObject.Find("HenryBoss_Seina").transform;
                    transform.LookAt(new
Vector3(kohde.position.x, 0, kohde.position.z));
                }

                else if (_tila == Tila.RunToSide &&
transform.position == kohde.position && randomReuna == 3)
                {
```

```
                    kohde =
GameObject.Find("HenryBoss_Seina2").transform;
                    transform.LookAt(new
Vector3(kohde.position.x, 0, kohde.position.z));
                }

                else if (_tila == Tila.RunToSide &&
transform.position == kohde.position && randomReuna == 4)
                {
                    kohde =
GameObject.Find("HenryBoss_Seina2").transform;
                    transform.LookAt(new
Vector3(kohde.position.x, 0, kohde.position.z));
                }
            }
        }
    }
```

Aluksi *vihuReunassa*-totuusarvo asetetaan todeksi ja toistetaan Henryn juoksu-animaatio. Sitten muodostetaan if-lause, jonka sisältö vaihtelee valittavan reunan mukaisesti. Valittavana on neljä erilaista kohdetta, jonka luokse se ohjataan aluksi GameObject.Find().transform -komennon avulla. Tämän avulla kerrotaan, mihin kohteeseen Henryn tulee liikkua. Transform.LookAt-komennon avulla kerrotaan, mihin suuntaan Henryn pitää katsoa liikkuessaan. Se on ohjelmoitu katsomaan siis aina suoraan edessä olevaan reunaan.

```
void Start () {

    enemy = GameObject.FindWithTag("Boss");
    tiluta = 2;
    ammu = 1;
}

void OnTriggerEnter(Collider otherObject)
{
    if(otherObject.tag == "Boss")
    {
        FSM_HHBoss.vaihdaTila(ammu);
    }
}
```

Edeltävä koodi on lähes sama Seinä:ssä ja Seinä 2:ssa. Ainoana erotuksena objektien välillä on tilan vaihtumisen nimi. Aluksi määritetään vihollisen nimi, joka on merkattu ”Boss”-tunnisteella. Sitten määritetään kokonaislukumuuttujat ”tiluta” ja ”ammu”, jotka saavat arvot 2 ja 1.

OnTriggerEnter-kohdassa määritetään Henryn osuminen seinään. Osumistilanteessa viitataan FSM_HHBoss-luokkaan, joka sisältää vaihdaTila-funktion. Tämän avulla saadaan tila kätevästi vaihdettua, ilman monimutkaisia ohjelmakoodeja.

Kun Henry on juossut jompaankumpaan seinään, niin osumistilanteessa Henryn tilanvaihdos riippuu osuttavasta seinästä. Seinässä tilaksi vaihtuu ampuminen, eli Dakka-tila ja Seinä 2:ssa tilaksi vaihtuu kitaran soittaminen eli Tilu-tila.

```
void dakka()
    {
        this.animation.Play("ampuminen");

        int panostenLkm = 17;
        transform.LookAt(new Vector3(playeri.position.x, 0,
playeri.position.z));

            if(Time.time > ( lastFireTime + burstDelay) &&
BossLuotiLaskuri< panostenLkm)
            {
                vihuAmpuu = true;
                lastFireTime= Time.time;
                Instantiate(FSM_HHBossGunPrefab,
transform.position, transform.rotation);
                BossLuotiLaskuri++;
            }

            if(BossLuotiLaskuri == 1)
            {
                audio.clip = sarjatuli;
                audio.Play();
            }

        if(BossLuotiLaskuri == panostenLkm && vihuAmpuuViive
<=0)
        {
            vihuAmpuu = false;
            BossLuotiLaskuri = 0;
            vihuAmpuuViive = 2;
            vihuAmpuu = true;
        }

        if(VihuReunassaTimeri <=0)
        {
            VihuReunassa = false;
            _tila = Tila.move;
        }

    }
```

Dakka-tilassa toistetaan aluksi ”ampuminen”-animaatio, jossa kitara heilahtelee lievästi edestakaisin. Tämän jälkeen esitellään kokonaislukumuuttuja *panostenLkm*, jonka avulla voidaan määritellä Henryn ampumien panosten lukumäärää.

Transform.LookAt:n avulla seurataan taas pelaajan liikkeitä ja ampumislaskurit on toteutettu samalla tavalla kuin Paulin ampuminen. Lopuksi, kun luodit on ammuttu ja ampumisen viive on saanut arvon 0, niin *vihuAmpuu*-totuusarvo asetetaan epätodeksi. Henryn ampumien luotien lukumäärä nollataan, ampumisen viive asetetaan alkuperäiseksi. Ennen tilanvaihdosta asetetaan *vihuAmpuu* totuusarvo alkuperäiseksi. Lopussa on *VihuReunassaTimeri,* jonka avulla voidaan määritellä aika, jonka Henry viettää reunassa. Tämän ajan umpeutuessa Henry lähtee liikkeelle jahtaamaan pelaajaa, ja tilaksi vaihdetaan move-tila.

```
void tilu()
    {
        if(vihuTiluttaa == false
        {
            vihuTiluttaa = true;

            if(!audio.isPlaying)
            {
                audio.clip = tilulilu;
                audio.Play();
            }

            this.animation["tilulilu"].speed = 0.9f;
            this.animation.Play("tilulilu");

            transform.LookAt(new Vector3(playeri.position.x,
0, playeri.position.z));

        }

        if(!this.animation.IsPlaying("tilulilu"))
        {
            vihuTiluttaa = false;
            _tila = Tila.SuperIsku;
        }
    }
```

Tilu-tilassa tarkistetaan aluksi *vihuTiluttaa*-totuusarvon tilanne. Jos totuusarvon on epätosi, se asetetaan todeksi. Seuraavaksi toistetaan Henryn kitarasoolon ääniraita ja samalla toistetaan soittamisen animaatio. Pelaajan seuraaminen ei ole välttämätöntä, mutta tässä tapauksessa se tuo eloa vastustajan pelihahmoon. Lopuksi *vihuTiluttaa*-totuuarvo asetetaan epätodeksi ja tilaksi vaihdetaan SuperIsku.

```
void superisku()
    {
        transform.LookAt(new Vector3(playeri.position.x, 0,
playeri.position.z));
        if(valimatka > maksimiEtaisyys )
        {
            maksimiEtaisyys = 0.5f;
```

```
                transform.Translate(Vector3.forward /
vihun_nopeus, Space.Self);
            }
        }
```

SupeIisku-tilaan joudutaan, mikäli Henryä ei saada keskeytettyä kitarasoolon soittamisen aikana. Tässä tilassa Henry seuraa jälleen pelaajaa silmä tarkkana ja maksimiEtaisyys-muuttujan arvon ollessa 0,5. Jotta SuperIsku-tila toimisi oikein, oli lisättävä Update-funktion sisälle päivitettäviä arvoja.

```
        if(_tila == Tila.SuperIsku)
            {
                vihun_nopeus = 5.0f;
                 if(valimatka > 2f){
                    this.animation.Play("juoksu");
                 }
                else if (valimatka < 2f)
                 {
                    if(this.animation.IsPlaying("juoksu"))
                        this.animation.Stop();
                    if(!this.animation.IsPlaying("juoksu"))
                    {
                        SuperIskuTimer -= Time.deltaTime;
                        if(SuperIskuTimer>0)
                        {
                            this.animation.Play("voimakas_isku");
                            storedData.SendMessage("ApplyDamage",
SuperDamage, SendMessageOptions.DontRequireReceiver);
                        }
                        else if(SuperIskuTimer <=0)
                        {
                            _tila = Tila.move;
                        }

                    }
                 }
            }
```

Aluksi määritettiin Henryn nopeus, joka olisi huomattavasti nopeampi tavalliseen verrattuna. Sitten määritettiin if-lauseen sisälle, mikäli välimatka kasvaa tarpeeksi suureksi, niin toistetaan ”juoksu”-animaatio. Muussa tapauksessa animaation toistaminen lopetetaan ja SuperIskuTimer-ajastin lähtee käyntiin. Toistetaan ”voimakas_isku” animaatio ajastimen arvon ollessa suurempi kuin 0. Henry huitoo tässä vaiheessa voimakkaasti kitaralllaan. SendMessage-toiminnon avulla lähetetään SuperDamage-muuttujan avulla tieto pelaajan energian massiivisesta vähenemisestä. *SuperIskuTimer*-ajastimen nollautuessa vaihdetaan tilaksi move-tila. Mikäli Henry tuhotaan, se menee tietenkin Dead-tilaan. Muutoin tilakone lähtee suorittamaan kaikkia toimintoja alusta lähtien samalla kaavalla.

5 TULOKSET JA POHDINTA

Unity 3D-ohjelmiston avulla on mahdollista luoda tekoälyä ja käyttää hyväksi Motion Capture-tekniikkaa. Tosin ilman minkäänlaisia apuohjelmia se vaatii kovasti työtä. Tilakoneen ohjelmoiminen tyhjästä vei oman aikansa, mutta ensimmäisen tilakoneen ohjelmoimisen jälkeen muiden tilakoneiden toteuttaminen oli helpompaa. Helpompaa siksi, että oli jo tietynlainen ”runko” ja ymmärrys tilakoneen koodin toiminnasta. Tilakoneen toteuttamista helpotti myös Unity:n oma käyttöliittymä, jonka Inspector-ikkunan avulla näin käynnissä olevien tilojen nimet. Se helpotti ohjelmointityötä suuresti, sillä virheiden korjaus oli helpompaa tilasiirtymien näkyvyyden avulla.

Motion Capture -tekniikkaa voitiin käyttää hyväksi vihollispelaajien ylävartalon liikkeissä, kuten esimerkiksi Natural-Twenty Paulin lyöntiliikkeissä. Ongelmaksi muodostui tekniikan käyttäminen alaraajoihin, sillä askelten tunnistaminen oli järjestelmälle hankalaa. Kävelyanimaatioihin tuli ylimääräisiä liikkeitä ja niiden korjaaminen oli käytännössä erittäin vaikeaa. Tämän takia graafikkomme tekivät hahmojen kävelyanimaatiot manuaalisesti käsin toisella ohjelmalla. Lopulta kävelyanimaatio saatiin toimimaan toivotulla tavalla.

Animaatioiden toteuttaminen oli myös oma haasteensa, sillä niitä oli välillä haastava saada kohdilleen pelirytmin kanssa. Steamdroid peli on kohtuullisen nopeatempoinen ja siinä käytetään paljon lyöntejä. Tämän vuoksi vihollispelaajien oli reagoitava nopeasti pelaajan tekemiin valintoihin eli lyönteihin. Animaatioiden toteuttamista suunnittellin vanhanaikaisesti paperilla ja kävin mielessäni läpi tilasiirtymien aiheuttamia animaatiomuutoksia. Sain animaatiot lopulta toimimaan erilaisten ajastimien ja laskureiden avulla.

Steamdroid-pelissä viholliset käyttäytyvät samanlaisesti ja vaikeustaso ei muutu pelin edetessä. Tässä olisi oiva jatkokehittämismahdollisuus sumean logiikan avulla. Sen avulla viholliset oppisivat pelaajan pelityylin ja osaisivat muokata omaa käytöstään sen mukaisesti. En sano nykyistäkään peliä missään tapauksessa liian

helpoksi, mutta tällä tavoin peliin saataisiin lisää vaihtelevuutta. Pelaaja pystyy päättelemään tilakoneen siirtymät ja näin ennakoimaan tulevia iskuja.

Opinnäytetyön ansiosta opin paljon lisää Unity3D-ohjelmistosta ja C#-ohjelmointikielestä. Opin myös ymmärtämään tilakoneiden toimintaa sekä teoriassa, että käytännössä, ja uskon siitä olevan hyötyä tulevaisuudessa ohjelmoinnin parissa. En voi myöskään olla korostamatta suunnitellun tärkeyttä kaikenlaisessa ohjelmointityössä, koska ilman sitä ollaan äkkiä hukassa. Suuremmilta vastoinkäymisiltä vältyttiin juuri suunnittelun takia ja aion käyttää samantapaista toimintatapaa jatkossakin.

LÄHTEET

Brookshear, J. 2003. Tietotekniikka. Helsinki: Edita Prima.

Brotkin, E. 2010. Optisen liikkeenkaappauksen käyttö hahmoanimoinnissa. Opinnäytetyö. Lahden ammattikorkeakoulu. Mediatekniikan koulutusohjelma.

Brownlee, J. 2002. Finite State Machines (FSM). Www-dokumentti. Saatavissa: http://www.scribd.com/doc/23724065/Finite-State-Machines-FSM. Luettu 11.7.2012.

Domander, R. 2012. Agenttien uskottavuus ja tekotyperyys digitaalisissa peleissä. Tietotekniikan kandidaatintutkielma. Jyväskylän yliopisto. Tietotekniikan koulutusohjelma.

Franklin, S & Graesser, A. 1996. Is It an agent, or just a program?: A taxonomy for autonomous agents. Lontoo: Springer-Verlag.

Goldstone, W. 2009. Unity Game Development Essentials : Build Fully Functional, Professional 3D Games with Realistic Environments, Sound, Dynamic Effects, and More!. Olton Birmingham: Packt Publishing Ltd.

Holmström, M. 2008. Tila-automaatit tietokonepeleissä. Www-dokumentti. Saatavissa: http://www.cs.helsinki.fi/u/vihavain/k08/sem/korjatut/holmstrom_tilakoneet.pdf. Luettu 10.7.2012.

Johnson, D. & Wiles, J. 2012. Computer Games with Intelligence. Www-dokumentti. Saatavissa: http://www.google.fi/url?sa=t&rct=j&q=fuzzy%20logic%20in%20game%20industry&source=web&cd=3&ved=0CFcQFjAC&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.84.1848%26rep%3Drep1%26type%3Dpdf&ei=OBYeUI_pDIHk4QT9iIC4Aw&usg=AFQjCNE0QOXMF7glTnv0fyQjMccXnH6d-A. Luettu 5.8.2012.

Jones, M. 2003. AI Application Programming. Herndon : Charles River Media.

Järvinen, P. 2003. IT-tietosanakirja. Jyväskylä: Docendo Finland.

Kukkonen, A. 2010. Reitinhaku ja tekoälyn päätöksenteko kaksiulotteisessa videopelissä. Opinnäytetyö. Rovaniemen ammattikorkeakoulu. Tietotekniikan koulutusohjelma.

Laine, H. 2008. Ohjelmiston mallintaminen olioiden elinkaaret - tilakaavio. Www-dokumentti. Saatavissa: http://www.cs.helsinki.fi/u/laine/malli/s08/pdf/tilakaavio_c.pdf. Luettu 10.7.2012.

Lecky-Thompson, G. 2007. AI and Artificial Life in Video Games. Boston: Course Tecnology.

Marketwire. 2012. Unity Reaches One Million Registered Developers Engine. Www-dokumentti. Saatavissa: http://www.marketwire.com/press-release/unity-reaches-one-million-registered-developers-1641486.htm. Luettu 27.7.2012.

Middleton, Z. 2002. Case History: The Evolution of Artificial Intelligence in Computer Games. Www-dokumentti. Saatavissa:http://www.stanford.edu/group/htgg/cgi-bin/drupal/sites/default/files2/zmiddleton_2002_1.pdf. Luettu 9.7.2012.

Piipponen, M. 2008. Samanaikaisuuden mallintaminen oliokeskeisessä ohjelmistokehitysmenetelmässä. Pro gradu-tutkielma. Tampereen yliopisto. Tietojenkäsittelyopin koulutusohjelma.

Planet Unreal. 2012. Unreal Tournament Game Guide. Www-dokumentti. Saatavissa: http://planetunreal.gamespy.com/View.php?view=UTGameInfo.Detail&id=7&game=6. Luettu 9.7.2012

Stuttard Parker, M. 2011. Getting Started with Unity – Colliders & UnityScript. Www-dokumentti. Saatavissa http://active.tutsplus.com/tutorials/unity/getting-started-with-unity-colliders-unityscript/. Luettu 30.7.2012.

Unity 3D. 2012. Engine. Www-dokumentti. Saatavissa: http://unity3d.com/unity/engine/programming. Luettu 27.7.2012.

Unity 3D. 2012. Sphere Collider. Www-dokumentti. Saatavissa: http://docs.unity3d.com/Documentation/Components/class-SphereCollider.html. Luettu 30.7.2012.

Unity 3D. 2012. Audio Source. Www-dokumentti. Saatavissa: http://docs.unity3d.com/Documentation/Components/class-AudioSource.html. Luettu 31.7.2012.

Unity 3D. 2012. Prefabs. Www-dokumentti. Saatavissa: http://docs.unity3d.com/Documentation/Manual/Prefabs.html. Luettu 1.8.2012.

Unity 3D. 2012. Animation. Www-dokumentti. Saatavissa: http://docs.unity3d.com/Documentation/Components/class-Animation.html. Luettu 1.8.2012.

FSM_NTPaul.cs

```
using UnityEngine;
using System.Collections;

public class FSM_NTPaul: MonoBehaviour {

    // NATURAL-TWENTY PAULIN TILAKONE
      // ASEEN PREFAB: FSM_NTPaulGunPrefab

    // eri tilat
    public enum Tila {
        Boast,
        Active,
        Stun,
        Downed,
        Strike,
        Shoot,
        move,
        Thrown,
        Hold,
        Rising,
        Dead
    }

    public GameObject hitParticle;

    Transform lhand;
    Transform rhand;
    Transform lleg;

    public AudioClip hit1;
    public AudioClip hit2;
    public AudioClip hit3;

    //paikallinen muuttuja taistelu
    public Tila _tila;

    public Transform playeri;
    public Transform enemy;

    public GameObject FSM_NTPaulGunPrefab;
    public GameObject coin; // kolikko
    GameObject storedData;

    //paulin aseen Ã¤Ã¤ni
    public AudioClip revolveri;

    // paul huomaa pelaajan-Ã¤Ã¤net
    public AudioClip hey1;
    public AudioClip hey2;
```

```
public bool kuolema_aani_toista = false;
public AudioClip death1;
public AudioClip death2;

//vÃ¤limatkan mittaamiseen tarvittavat muuttujat
float valimatka;
float maksimiEtaisyys;

bool vihuHuomattuAluksi = false;

// Vihollisen move-tilaan liittyviÃ¤
public float vihun_nopeus; // vihollisen liikkumisnopeus
private float damage;   // paulin tekemÃ¤ damage

// vihollisen ampusin timeri, ettei ammu koko ajan
public float vihuAmpuuViive;
public float burstDelay;
public float lastFireTime;
public bool vihuAmpuu = false;
public float AnimaatioTimer;
public bool AnimaatioPaalla = false;
public static int PaulLuotiLaskuri;

public float aika;
public float klipinaika;

// Downed-tilan timerin
public float vihuSelallaan_timer;
public bool vihuSelallaan_bool = false;

//Dead-tila
public float vihuDeadTimer;

//Stun-tila
public float StunnissaTimer;

// Rising-tilan timeri, vihu nousee ylos
public float RisingTimeri;
public bool vihuNouseeYlos = false;

// Paulin health
public float curHp;
//Vector3 healthBarPos;

// knocbackissa kÃ¤ytetyt muuttujat
float cTime = 0;
int knockbackX;
int knockbackZ;
Vector3 startPos;
Vector3 endPos;

//vihollisten tÃ¶rmÃ¤ys/suunnanvaihto
Vector3 start;
```

```
Vector3 end;
bool vihutTormaa = false;
int tonaisyKerroinX;
int tonaisyKerroinZ;
float lerpAjastin;

//heitossa kÃ¤ytetyt muuttujat
int heittoX;
int heittoZ;
float trajectoryHeight = 2;

//holdissa kÃ¤ytetyt muuttujat
Transform dmgcollider;
//float kiinniCheck = 0f;
bool heittoon = false;
bool pitoon = false;

bool colliderissa = false;

public bool deadEnemy = false;

void Start () {

    lhand = GameObject.FindWithTag("vasen").transform;
    rhand = GameObject.FindWithTag("oikea").transform;
    lleg = GameObject.FindWithTag("varvas").transform;

    //animaatioiden toistotilat
    this.animation["idle"].wrapMode = WrapMode.Loop;
    this.animation["lyonti"].wrapMode = WrapMode.Once;
    this.animation["kaatuu"].wrapMode = WrapMode.Once;
    this.animation["ylos"].wrapMode = WrapMode.Once;
    this.animation["kavely"].wrapMode = WrapMode.Loop;
    this.animation["ase_nosto"].wrapMode = WrapMode.Once;
    this.animation["ase_lasku"].wrapMode = WrapMode.Once;
    this.animation["ampuminen"].wrapMode = WrapMode.Loop;

    //Vihollisten tÃ¶rmÃ¤ys/suunnanvaihto
    lerpAjastin = 0.7f;

    // asetetaan vihollisen tilaksi aluksi idle ja muutetaan sitÃ¤ myÃ¶hemmin
    // TÃ„SSÃ„ ASETETAAN VIHOLLISEN ENSIMMÃ„INEN TILA
    _tila = Tila.Boast;

    //etsitÃ¤Ã¤n pelaaja tagin perusteella
    dmgcollider = GameObject.Find("dmgCollider").transform;
    playeri = GameObject.FindWithTag("Player").transform;
    enemy = GameObject.FindWithTag("Enemy").transform;
    storedData = GameObject.Find("gui_kehys").gameObject;

    // Paulin health
    curHp = 30.0f;

    // mÃ¤Ã¤ritetÃ¤Ã¤n paulin tekemÃ¤ damage nyrkeillÃ¤
```

```
    damage = 0.05f;

    // alustetaan muuttujat
    vihun_nopeus = 22.0f;   //sÃ¤Ã¤dÃ¤ Paulin nopeutta

    // vihollisen ampuma timer
    burstDelay = 2.0f;  // eli ampuu 1,25 sekunnin vÃ¤lein pelaajaa
    lastFireTime = 0.0f; // nollaus
    AnimaatioTimer = 2.0f;
    PaulLuotiLaskuri = 0;

    // downed-tilassa vihollinen kaksi sekuntia maassa
    vihuSelallaan_timer = 1.8f;

    //dead-tilan timeri eli animaatio toistetaan sen verran
    vihuDeadTimer = 1.0f;

    //Stun-tilan animaation timer
    StunnissaTimer = 0.3f;

    // Rising-tilassa oleva nousemistimer
    RisingTimeri = 0.8f;

    // move-tilan muuttujien alustus
    maksimiEtaisyys = 2.0f; // vihollinen jÃ¤Ã¤ 0.7:n pÃ¤Ã¤hÃ¤n pelaajasta, eikÃ¤ tule kiinni

    // vihollinen tulee kentÃ¤lle oikein pÃ¤in
     transform.rotation = Quaternion.Euler(0,0,0);

}

/* UPDATE*/

void Update ()
{
        // muutetaan eri taistelutiloja kÃ¶tevÃ¤sti switchillÃ¤
        switch(_tila) {
            case Tila.Boast:
                boast();
                    break;

            case Tila.Active:
                toactive();
                break;

            case Tila.Stun :
                stun();
                break;

            case Tila.Downed :
                downed();
                break;

            case Tila.Strike :
```

```
                strike();
                break;

            case Tila.Shoot :
                shoot();
                break;

            case Tila.move :
                tomove();
                break;

            case Tila.Thrown :
                thrown();
                break;

            case Tila.Hold:
                hold();
                break;

            case Tila.Rising :
                rising();
                break;

            case Tila.Dead:
                dead();
                break;

        }

        //jos tÃ¶rmÃ¤Ã¤ viholliseen, siirretÃ¤Ã¤n toiseen suuntaan
        start = new Vec-
tor3(transform.position.x,transform.position.y,transform.position.z);
        end = new Vector3(start.x + tonaisyKerroinX, start.y,start.z + to-
naisyKerroinZ);


        if (_tila != Tila.Hold)
        {

            if(vihutTormaa == true)
            {
                if(lerpAjastin>0)
                lerpAjastin -= Time.deltaTime;

                Vector3 nykyinenSijainti = Vector3.Lerp(start, end, Time.deltaTime * 2);
                transform.position = nykyinenSijainti;

                if(lerpAjastin <=0)
                {
                    lerpAjastin = 0;
                    vihutTormaa = false;
                }

            }
```

```
    }

        // luodaan vektori eulerAngles, ottaa paulin sen hetkisen rotation ja asettaa samaan vektoriin y-komponentin.
        // kÃ¤Ã¤ntÃ¤Ã¤ y-komponentin mukaan, muut ovat nollia.

        Vector3 eulerAngles = transform.rotation.eulerAngles;
        eulerAngles = new Vector3(0, eulerAngles.y, 0);
        transform.rotation = Quaternion.Euler(eulerAngles);

        if (pitoon == true) {
            _tila = Tila.Hold;
        }

        if(_tila == Tila.Stun)
        {
            StunnissaTimer -= Time.deltaTime;
        }

        if(_tila != Tila.Stun)
        {
            StunnissaTimer = 0.3f;
        }

        if (curHp <= 0){
            _tila = Tila.Dead;
        }

            // lasketaan vÃ¤limatka vihollisen ja pelaajan vÃ¤lillÃ¤
        valimatka = Vector3.Distance(playeri.position, transform.position);

        if (valimatka < 2f && _tila == Tila.move)
            _tila = Tila.Strike;
        else if (valimatka > 2.5f && _tila == Tila.Strike)
            _tila = Tila.move;

        // vihollisen ampusen timerin viive pitÃ¤Ã¤ olla updatessa, ettÃ¤ pÃ¤ivittyy

        // downed-tilan laskuria pÃ¤ivitetÃ¤Ã¤n

        if(vihuSelallaan_bool == true)
        {
            vihuSelallaan_timer -= Time.deltaTime;
        }

        if(vihuNouseeYlos == true)
        {
            RisingTimeri -= Time.deltaTime;
        }

        if(AnimaatioPaalla == true)
        {
            AnimaatioTimer -= Time.deltaTime;
        }
```

```
    if(_tila == Tila.Dead)
    {
        vihuDeadTimer -= Time.deltaTime;

    }

    if(_tila == Tila.Dead)
    {
        //Attack.enemyIsDead = true;
        deadEnemy = true;
    }
    else {
        //Attack.enemyIsDead = false;
        deadEnemy = false;
    }

}

// KUTSUTTAVAT TILAKONEEN FUNKTIOT JÃ„RJESTYKSESSÃ„
//-----------------------------------------------------------------------------------------------
// //taistelun funktiot, joita kutsutaan. Eli mitÃ¤ halutaan tehdÃ¤?
// nÃ¤itÃ¤ kutsutaan switch-case rakenteesta.

void boast()
{
    this.animation.Play("idle");
    // Vihollinen havaitsee pelaajaan ensimÃ¤isen kerran ja asettaa tilan Active
    transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));
    int spottausaani = Random.Range(1,4);

    switch(spottausaani)
    {
        case 1:
            if(vihuHuomattuAluksi== false)
            {
                vihuHuomattuAluksi = true;
                audio.clip = hey1;
                audio.Play();
            }
                break;
        case 2:
            if(vihuHuomattuAluksi== false)
            {
                vihuHuomattuAluksi = true;
                audio.clip = hey2;
                audio.Play();
            }
                break;

        case 3:
            if(vihuHuomattuAluksi== false)
            {
                vihuHuomattuAluksi = true;
                audio.clip = hey2;
                audio.Play();
```

```
}
                break;
            case 4:
                if(vihuHuomattuAluksi== false)
                {
                    vihuHuomattuAluksi = true;
                    audio.clip = hey2;
                    audio.Play();
                }
                    break;
        }

        if(!audio.isPlaying)
        {
            _tila = Tila.Active;
        }
    }

    void toactive()
    {
        Debug.Log("Active!");
        // Vihollinen alkaa tekemÃ¤Ã¤n toimintoja
        transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z)); //
kÃ¤Ã¤nnetÃ¤n paul valmiiksi pelaajaa kohti
        _tila = Tila.Shoot; // asetetaan paul ampumaan ensin, koska pÃ¤Ã¤asiallinen tehtÃ¤vÃ¤
    }

    void stun()
    {
        this.animation["pataan"].speed = 2.5f;
        this.animation.Play("pataan");

        if(StunnissaTimer<=0)
        {
            this.animation.Stop();
            _tila = Tila.move;
        }

        Debug.Log("Stun!");
    }

    void downed()
    {
        vihuSelallaan_bool = true;

        Debug.Log("Downed!");

        if(vihuSelallaan_timer <=0)
        {
            vihuSelallaan_bool = false;
            vihuSelallaan_timer = 0.0f;
            _tila = Tila.Rising;
        }
    }
```

```
void strike()
   {
      if(Time.timeScale !=0)
      {
      transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));   //
vihollinen lyÃ¶ pelaajaan suuntaan

      if(!this.animation.IsPlaying("lyonti")) // stopataan edellisen tilan animaatio
      {
         this.animation.Stop();
      }
      // Vihu lyÃ¶ pelaajaa nyrkeillÃ¤, kun vihollinen (Paul) on tarpeeksi lÃ¤hellÃ¤ pelaajaa
      // valimatka-funktio on mÃ¤Ã¤ritelty aikaisemmin updatessa.

      // Paulin tekemÃ¤Ã¤ damagea voidaan muuttaa ylhÃ¤Ã¤llÃ¤ olevalla damage-muuttujalla
kÃ¤tevÃ¤sti.
         Debug.Log("Strike!");
         this.animation.Play("lyonti");
         this.animation["lyonti"].speed = 2.5f;  // nopeutetaan animaatiota 2,5-kertaiseksi,
jotta isku nÃ¤yttÃ¤Ã¤ vauhdikkaalta
         storedData.SendMessage("ApplyDamage", damage, SendMessage-
Options.DontRequireReceiver);
      }
   }

   void shoot()
   {
      // vihu nostaa aseen ampumista varten
      transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z)); // am-
puu pelaajaa kohti

      if(Time.time > ( lastFireTime + burstDelay) )
      {
         this.animation.CrossFade("ase_nosto");
         this.animation.CrossFade("ampuminen");
         audio.clip = revolveri;
         audio.Play();   // toistetaan revolverin Ã¤Ã¤ni
         vihuAmpuu = true;
         lastFireTime=Time.time;
         Instantiate(FSM_NTPaulGunPrefab, transform.position, transform.rotation);
         PaulLuotiLaskuri++;
      }

      // Kuinka monta luotia ammutaan, ennen siirtymistÃ¤ move-tilaan
      if(PaulLuotiLaskuri > 16)
         _tila = Tila.move;
   }


   void tomove()   // Move-tila
   {

      if(Time.timeScale !=0)
      {
```

```
        this.animation.Play("kavely");  // soitetaan kavely

        Debug.Log("move!");
        // Vihollinen liikkuu suhteessa viholliseen
        transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));

        // jos on kauempana kohteesta kuin max_etaisyys, liikkuu pelaajaa kohti
        if(valimatka > maksimiEtaisyys )
        {
            transform.Translate(Vector3.forward / vihun_nopeus,Space.Self);
        }

        if(_tila != Tila.Rising)
        {
            // alustetaan aina uudestaan, jotta viholliset pystyy kaatumaan uudestaan, eikÃ¤ kaadu vain
yhden kerran
            vihuSelallaan_timer = 1.8f;

            // Rising-tilassa oleva nousemistimer
            RisingTimeri = 0.8f;
        }

        }

    }

    void hold()
    {
        this.animation.Play("kiinni");  // lopetetaan edellisen tilan animaatio, mikÃ¤li sellainen on.

        if (pitoon == true)
            pitoon = false;

        this.transform.position = new Vector3(dmgcollider.position.x, 0, dmgcollid-
er.position.z); // siirtÃ¤Ã¤ paulia oikeaan kohtaan heittÃ¤essÃ¤
        transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z)); // kat-
soo pelaajaa kohti kun otetaan rinnuksista kiinni
        Debug.Log("Hold");

        if (Attack.kiinniCheck <= 0) {
            _tila = Tila.move;
        }

        if (heittoon == true) {
            _tila = Tila.Thrown;
        }
    }

    void thrown()
    {
        this.animation.Play("kaatuu");

        Debug.Log("Thrown!");
        // Vihollista ollaan heittÃ¤mÃ¤ssÃ¤
```

```
    if (heittoon == true)
        heittoon = false;

      if (cTime == 0) {
        startPos = new Vector3(dmgcollider.position.x, dmgcollider.position.y-1,
dmgcollider.position.z);
        endPos = new Vector3(startPos.x + heittoX, startPos.y, startPos.z + heit-
toZ);
      }

      cTime += Time.deltaTime * 3;

      // calculate straight-line lerp position:
      Vector3 currentPos = Vector3.Lerp(startPos, endPos, cTime);
      // add a value to Y, using Sine to give a curved trajectory in the Y direction
      currentPos.y += trajectoryHeight * Mathf.Sin(Mathf.Clamp01(cTime) *
Mathf.PI);
      // finally assign the computed position to our gameObject:

      this.transform.position = currentPos;

      if (cTime > 2f) {
        cTime = 0;
        _tila = Tila.Downed;
      }

  }

  void rising()
  {
      Debug.Log("Rising!");
      // Vihu nousee pystyyn

      vihuNouseeYlos = true;
       // katsoo pelaajaa kohti kun otetaan rinnuksista kiinni
      this.animation.Play("ylos");    // vihollinen nousee ylÃ¶s animaation avulla

      if(RisingTimeri <=0)
      {
        vihuNouseeYlos= false;
        RisingTimeri = 0.0f;
        _tila = Tila.move;
      }

  }

  void dead()
  {
  Debug.Log("Dead!");
      // kun Paulin health loppuu niin aseta suoraan destroy(this.gameObject);
      if(curHp <= 0)
      {
        this.animation.Play("kaatuu");
```

```
    if(vihuDeadTimer<=0)
      {
         this.animation.Stop();

       if(curHp <= 0)
       {
          Debug.Log("Tuhosit vihun!");
          for (int i = 0; i < Random.Range(1, 10); i++)
          {
           Instantiate(coin, transform.position, transform.rotation);
          }

          if(colliderissa == true)
          Attack.enemiesOnTrigger--;
          Destroy(this.gameObject);
       }

       }

   }
}

float timer(float s) {
if (s > 0)
   s -= Time.deltaTime;
if (s <= 0)
   s = 0;
return s;
}

//NÃ¤itÃ¤ funktioita kutsutaan sendmessagella.
void toHold(int s) {
   if (s == 1){
      pitoon = true;
      Debug.Log("PAULISTA KIINNI");
   }
}

void toThrown(int t){
   if (t == 1){
      heittoon = true;
   }
}

void suunta(int s){
   if (s == 1){
      heittoZ = 0;
      heittoX = -4;
      knockbackZ = 0;
      knockbackX = -2;
   }

   if (s == 2) {
      heittoZ = 0;
      heittoX = 4;
```

```
            knockbackZ = 0;
            knockbackX = 2;
        }

        if (s == 3){
            heittoZ = 4;
            heittoX = 0;
            knockbackZ = 2;
            knockbackX = 0;
        }

        if (s == 4){
            heittoZ = -4;
            heittoX = 0;
            knockbackZ = -2;
            knockbackX = 0;
        }

        if (s == 5) {
            heittoZ = 3;
            heittoX = 3;
            knockbackZ = 1;
            knockbackX = 1;
        }
        if (s == 6){
            heittoZ = -3;
            heittoX = 3;
            knockbackZ = -1;
            knockbackX = 1;
        }
        if(s == 7){
            heittoZ = -3;
            heittoX = -3;
            knockbackZ = -1;
            knockbackX = -1;
        }
        if (s == 8){
            heittoZ = 3;
            heittoX = -3;
            knockbackZ = 1;
            knockbackX = -1;
        }
    }

    //Kombojen vastaanotto
    void iskuJ(int dmg)
    {
        if (vihuSelallaan_bool == false){
            if(this.animation.isPlaying)
            {
                this.animation.Stop();
            }
            _tila = Tila.Stun;
            curHp -= dmg;
            Instantiate(hitParticle, lhand.position, transform.rotation);
```

```
        audio.clip = hit1;
        audio.Play();
    }
}

void iskuJJ(int dmg)
{
    if (vihuSelallaan_bool == false){
        if(this.animation.isPlaying)
        {
            this.animation.Stop();
        }
        _tila = Tila.Stun;

        curHp -= dmg;
        Instantiate(hitParticle, rhand.position, transform.rotation);
        audio.clip = hit2;
        audio.Play();
    }
}

void iskuJJJ(int dmg)
{
    if (vihuSelallaan_bool == false){
        curHp -= dmg;
        //_tila = Tila.Stun;
        Instantiate(hitParticle, lhand.position, transform.rotation);
        audio.clip = hit3;
        audio.Play();
    }
}

void iskuJJK(int dmg)
{
    if (vihuSelallaan_bool == false){
        curHp -= dmg;
        Instantiate(hitParticle, lleg.position, transform.rotation);
        audio.clip = hit3;
        audio.Play();
    }
}

void saha(int dmg)
{
    curHp -= dmg;
}

void super(int s)
{
    if(s==1)
    {
        curHp = 0f;
    }
}
```

```
// kun vihollinen osuu pelaajaan
   void OnTriggerStay(Collider otherObject)
   {
      if(otherObject.tag == "Enemy") {
         Tormays();
      }

      if(otherObject.tag == "PlayerDmg") {
         colliderissa = true;
      }
   }

   void Tormays()
   {
      if(vihutTormaa != true)
      {
         vihutTormaa = true;
         lerpAjastin = 0.7f;

         tonaisyKerroinZ = Random.Range(-2, 2);
         tonaisyKerroinX = Random.Range(-2, 2);
      }
   }

}
```

FSM_NTPaulGunPrefabScript.cs

```
using UnityEngine;
using System.Collections;

public class FSM_NTPaulGunPrefabScript : MonoBehaviour {

    public float luodinNopeus;

    // Paulin aseenta tuleva damage
    private float damage2 = 0.50f;

    //public Transform Vihu;
    public GameObject Vihu; // gameobject ja transform erikseen, jotta distance toimii oikein.
    public Transform Vihu2;

    GameObject storedData;
    void Start () {

        Vihu = GameObject.Find("Player");
        Vihu2 = GameObject.Find("Player").transform;
        storedData = GameObject.Find("gui_kehys");

        luodinNopeus = 15.0f;   // määritellään luodin nopeus
        transform.Rotate(0,0,90);   // laitetaan luoti vaaka-asentoon

        transform.position = transform.position + new Vector3(0,1.5f,0);    // luodin läh-
töposition määritteleminen
    }

    void Update () {

        // Jos luoti menee tarpeeksi kauas Paulista, niin luodin prefab tuhotaan. Näin ei viedä turhaan
resursseja.....
        if(Vector3.Distance(transform.position,Vihu2.position) > 15)
        {
            Destroy(this.gameObject);
        }

        // liikutetaan luotia kovalla vauhdilla
        transform.Translate(Vector3.forward * Time.deltaTime * luodinNopeus);
    }
    // laita tähän vihollisen (Paulin) luodin vaikutus pelaajaan!!!
    void OnTriggerStay(Collider otherObject)
    {
        if (otherObject.tag == "Player")
        {
            Debug.Log("Paul osuu pelaajaa!");
            //lähettää viestin
            storedData.SendMessage("ApplyDamage", damage2, SendMessage-
Options.DontRequireReceiver);
        }
    }
}
```

FSM_Bill.cs

```
using UnityEngine;
using System.Collections;

public class FSM_Bill: MonoBehaviour {

  // BILLIN TILAKONE

  // taistelun eri tilat
  public enum Tila {
     Boast,
     Active,
     Stun,
     Downed,
     Strike,
     move,
     Thrown,
     Hold,
     Rising,
     Dead
  }

  public GameObject hitParticle;

  Transform lhand;
  Transform rhand;
  Transform lleg;

  public AudioClip hit1;
  public AudioClip hit2;
  public AudioClip hit3;

  //paikallinen muuttuja taistelu
  public Tila _tila;

  public Transform playeri;
  public Transform enemy;

  //  public GameObject FSM_NTPaulGunPrefab;
  public GameObject coin; // kolikko
  GameObject storedData;


  //vÃ¤limatkan mittaamiseen tarvittavat muuttujat
  float valimatka;
  float maksimiEtaisyys;

  // Vihollisen move-tilaan liittyviÃ¤
  public float vihun_nopeus; // vihollisen liikkumisnopeus
  private float damage;   // paulin tekemÃ¤ damage

  // vihollisen ampusin timeri, ettei ammu koko ajan
  public float vihuAmpuuViive;
```

```
public float burstDelay;
public float lastFireTime;
public bool vihuAmpuu = false;
public float AnimaatioTimer;
public bool AnimaatioPaalla = false;
public static int PaulLuotiLaskuri;

public float aika;
public float klipinaika;

// Downed-tilan timerin
public float vihuSelallaan_timer;
public bool vihuSelallaan_bool = false;

//Stun-tila
public float StunnissaTimer;

//Dead-tila
public float vihuDeadTimer;

// Rising-tilan timeri, vihu nousee ylos
public float RisingTimeri;
public bool vihuNouseeYlos = false;

// Paulin health
public float curHp;
//Vector3 healthBarPos;

// knocbackissa kÃ¤ytetyt muuttujat
float cTime = 0;
float cTime2 = 0;
int knockbackX;
int knockbackZ;
Vector3 startPos;
 Vector3 endPos;

 //vihollisten tÃ¶rmÃ¤ys/suunnanvaihto

 Vector3 start;
 Vector3 end;
 bool vihutTormaa = false;
 int tonaisyKerroinX;
 int tonaisyKerroinZ;
float lerpAjastin;

 //heitossa kÃ¤ytetyt muuttujat
 int heittoX;
int heittoZ;
float trajectoryHeight = 2;

    //holdissa kÃ¤ytetyt muuttujat
    Transform dmgcollider;
    //float kiinniCheck = 0f;
 bool heittoon = false;
```

```
    bool pitoon = false;

    bool colliderissa = false;

    public bool deadEnemy = false;

void Start () {

    //animaatioiden toistotilat
    this.animation["idle"].wrapMode = WrapMode.Loop;
    this.animation["lyonti"].wrapMode = WrapMode.Once;
    this.animation["kavely"].wrapMode = WrapMode.Loop;
    this.animation["heitto"].wrapMode = WrapMode.Once;

    lhand = GameObject.FindWithTag("vasen").transform;
    rhand = GameObject.FindWithTag("oikea").transform;
    lleg = GameObject.FindWithTag("varvas").transform;

    //Vihollisten tÃ¶rmÃ¤ys/suunnanvaihto
    lerpAjastin = 0.7f;

    // asetetaan vihollisen tilaksi aluksi idle ja muutetaan sitÃ¤ myÃ¶hemmin
    // TÃ„SSÃ„ ASETETAAN VIHOLLISEN ENSIMMÃ„INEN TILA
    _tila = Tila.move;

    //etsitÃ¤Ã¤n pelaaja tagin perusteella
    dmgcollider = GameObject.Find("dmgCollider").transform;
    playeri = GameObject.FindWithTag("Player").transform;
    enemy = GameObject.FindWithTag("Enemy").transform;

    curHp = 30.0f; //30

    // vihollisen liikkumiseen tarvittavia muuttujia
    transform.LookAt(playeri);

    // mÃ¤Ã¤ritetÃ¤Ã¤n paulin tekemÃ¤ damage nyrkeillÃ¤
    damage = 0.05f;

    vihun_nopeus = 22.0f;   //sÃ¤Ã¤dÃ¤ Paulin nopeutta

    // vihollisen ampuma timer
    burstDelay = 2.0f;  // eli ampuu 1,25 sekunnin vÃ¤lein pelaajaa
    lastFireTime = 0.0f; // nollaus
    AnimaatioTimer = 2.0f;
    PaulLuotiLaskuri = 0;

    // downed-tilassa vihollinen kaksi sekuntia maassa
    vihuSelallaan_timer = 1.8f;

    //kun vihollinen kuolee, niin toistetaan animaatio tietyn ajan
    vihuDeadTimer = 1.0f;

    // Rising-tilassa oleva nousemistimer
    RisingTimeri = 0.8f;
```

```
    // move-tilan muuttujien alustus
    maksimiEtaisyys = 2.0f; // vihollinen jÃ¤Ã¤ 0.7:n pÃ¤Ã¤hÃ¤n pelaajasta, eikÃ¤ tule kiinni

    // stun-animaation timeri
    StunnissaTimer = 0.2f;

     transform.rotation = Quaternion.Euler(0,270,0);
     storedData = GameObject.Find("gui_kehys");

}

/* UPDATE*/

void Update ()
{
        // muutetaan eri taistelutiloja kÃ¶tevÃ¤sti switchillÃ¤
        switch(_tila) {
            case Tila.Boast:
                boast();
                    break;

            case Tila.Active:
                toactive();
                break;

            case Tila.Stun :
                stun();
                break;

            case Tila.Downed :
                downed();
                break;

            case Tila.Strike :
                strike();
                break;

            case Tila.move :
                tomove();
                break;

            case Tila.Thrown :
                thrown();
                break;

            case Tila.Hold:
                hold();
                break;

            case Tila.Rising :
                rising();
                break;

            case Tila.Dead:
                dead();
```

```
break;

        }

    //jos tÃ¶rmÃ¤Ã¤ viholliseen, siirretÃ¤Ã¤n toiseen suuntaan
    start = new Vec-
tor3(transform.position.x,transform.position.y,transform.position.z);
    end = new Vector3(start.x + tonaisyKerroinX, start.y,start.z + tonaisyKerroinZ);


    if (_tila != Tila.Hold)
    {
    // tÃ¤hÃ¤n tulee lerp-ajastin
        if(vihutTormaa == true)
        {
            if(lerpAjastin>0)
            lerpAjastin -= Time.deltaTime;

            Vector3 nykyinenSijainti = Vector3.Lerp(start, end, Time.deltaTime * 2);
            transform.position = nykyinenSijainti;

            if(lerpAjastin <=0)
            {
                lerpAjastin = 0;
                vihutTormaa = false;
            }

        }

    }


    // luodaan vektori eulerAngles, ottaa paulin sen hetkisen rotation ja asettaa samaan vektoriin y-
komponentin.
    // kÃ¤Ã¤ntÃ¤Ã¤ y-komponentin mukaan, muut ovat nollia.

    Vector3 eulerAngles = transform.rotation.eulerAngles;
    eulerAngles = new Vector3(0, eulerAngles.y, 0);
    transform.rotation = Quaternion.Euler(eulerAngles);


    if (pitoon == true) {
        _tila = Tila.Hold;
    }

    if(_tila == Tila.Stun)
    {
        StunnissaTimer -= Time.deltaTime;
    }

    if(_tila != Tila.Stun)
    {
        StunnissaTimer = 0.2f;
    }
```

```
if (curHp <= 0){
        if(this.animation["heitto"].enabled == false && vihuDeadTimer == 1 &&
vihuSelallaan_bool == false)
            this.animation.Play("heitto");

        _tila = Tila.Dead;
    }

    // lasketaan vÃ¤limatka vihollisen ja pelaajan vÃ¤lillÃ¤
     valimatka = Vector3.Distance(playeri.position, transform.position);

    if (valimatka < 2f && _tila == Tila.move)
        _tila = Tila.Strike;
     else if (valimatka > 2.5f && _tila == Tila.Strike)
        _tila = Tila.move;

    // vihollisen ampusen timerin viive pitÃ¤Ã¤ olla updatessa, ettÃ¤ pÃ¤ivittyy

    // downed-tilan laskuria pÃ¤ivitetÃ¤Ã¤n

    if(vihuSelallaan_bool == true)
    {
        vihuSelallaan_timer -= Time.deltaTime;
    }

    if(vihuNouseeYlos == true)
    {
        RisingTimeri -= Time.deltaTime;
    }

    if(AnimaatioPaalla == true)
    {
        AnimaatioTimer -= Time.deltaTime;
    }

    if(_tila == Tila.Dead)
    {
        vihuDeadTimer -= Time.deltaTime;

    }

    if(_tila == Tila.Dead)
    {
        //Attack.enemyIsDead = true;
        deadEnemy = true;
    }
    else {
        //Attack.enemyIsDead = false;
        deadEnemy = false;
    }


}
```

```
// KUTSUTTAVAT TILAKONEEN FUNKTIOT JÃ„RJESTYKSESSÃ„
    //-------------------------------------------------------------------------------------------------------
    // //taistelun funktiot, joita kutsutaan. Eli mitÃ¤ halutaan tehdÃ¤?
    // nÃ¤itÃ¤ kutsutaan switch-case rakenteesta.

    void boast()
    {
        // Vihollinen havaitsee pelaajaan ensimÃ¤isen kerran ja asettaa tilan Active
        _tila = Tila.Active;

    }

    void toactive()
    {
        Debug.Log("Active!");
        // Vihollinen alkaa tekemÃ¤Ã¤n toimintoja
        _tila = Tila.move;
    }

    void stun()
    {
        this.animation["pataan"].speed = 2.5f;
        this.animation.Play("pataan");

        if(StunnissaTimer<=0)
        {
          this.animation.Stop();
          _tila = Tila.move;
        }
    }

    void downed()
    {
        //this.animation.Stop();

        vihuSelallaan_bool = true;

        Debug.Log("Downed!");

        if(vihuSelallaan_timer <=0)
        {
            vihuSelallaan_bool = false;
            vihuSelallaan_timer = 0.0f;
            _tila = Tila.Rising;
        }
    }

    void strike()
    {
        if(Time.timeScale !=0)
        {
            transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));   //
vihollinen lyÃ¶ pelaajaan suuntaan
```

```
    if(!this.animation.IsPlaying("lyonti")) // stopataan edellisen tilan animaatio
        {
            this.animation.Stop();
        }
        // Vihu lyÃ¶ pelaajaa nyrkeillÃ¤, kun vihollinen (Paul) on tarpeeksi lÃ¤hellÃ¤ pelaajaa
        // valimatka-funktio on mÃ¤Ã¤ritelty aikaisemmin updatessa.

        // Paulin tekemÃ¤Ã¤ damagea voidaan muuttaa ylhÃ¤Ã¤llÃ¤ olevalla damage-muuttujalla
kÃ¤tevÃ¤sti.
        Debug.Log("Strike!");
        this.animation.Play("lyonti");
        this.animation["lyonti"].speed = 1.5f;  // nopeutetaan animaatiota 2,5-kertaiseksi,
jotta isku nÃ¤yttÃ¤Ã¤ vauhdikkaalta
        storedData.SendMessage("ApplyDamage", damage, SendMessage-
Options.DontRequireReceiver);
    }
  }

  void tomove()   // Move-tila
  {
    if(Time.timeScale !=0)
    {
        this.animation["kavely"].speed = 1.5f;
        this.animation.Play("kavely");  // soitetaan kavely

        Debug.Log("move!");
        // Vihollinen liikkuu suhteessa viholliseen
        transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));

        // jos on kauempana kohteesta kuin max_etaisyys, liikkuu pelaajaa kohti
        if(valimatka > maksimiEtaisyys )
        {
            transform.Translate(Vector3.forward / vihun_nopeus, Space.Self);
        }

        if(_tila != Tila.Rising)
        {
            // alustetaan aina uudestaan, jotta viholliset pystyy kaatumaan uudestaan, eikÃ¤ kaadu vain
yhden kerran
            vihuSelallaan_timer = 1.8f;

            // Rising-tilassa oleva nousemistimer
            RisingTimeri = 0.8f;
        }
    }

  }

  void hold()
  {
    this.animation.Stop();  // lopetetaan edellisen tilan animaatio, mikÃ¤li sellainen on.

    if (pitoon == true)
        pitoon = false;
```

```
 this.transform.position = new Vector3(dmgcollider.position.x, 0, dmgcollid-
er.position.z); // siirtÃ¤Ã¤ paulia oikeaan kohtaan heittÃ¤essÃ¤
      transform.LookAt(new Vector3(playeri.position.x, 0 , playeri.position.z)); // kat-
soo pelaajaa kohti kun otetaan rinnuksista kiinni
      Debug.Log("Hold");

      if (Attack.kiinniCheck <= 0) {
         _tila = Tila.move;
      }

      if (heittoon == true) {
         _tila = Tila.Thrown;
      }
   }

   void thrown()
   {
      this.animation.Play("heitto");

      Debug.Log("Thrown!");
      // Vihollista ollaan heittÃ¤mÃ¤ssÃ¤
      if (heittoon == true)
         heittoon = false;

      if (cTime == 0) {
         startPos = new Vector3(dmgcollider.position.x, dmgcollider.position.y-1,
dmgcollider.position.z);
         endPos = new Vector3(startPos.x + heittoX, startPos.y, startPos.z + heittoZ);
      }

      cTime += Time.deltaTime * 3;

      // calculate straight-line lerp position:
      Vector3 currentPos = Vector3.Lerp(startPos, endPos, cTime);
      // add a value to Y, using Sine to give a curved trajectory in the Y direction
      currentPos.y += trajectoryHeight * Mathf.Sin(Mathf.Clamp01(cTime) *
Mathf.PI);
      // finally assign the computed position to our gameObject:

      this.transform.position = currentPos;

      if (cTime > 3f) {
         cTime = 0;
         _tila = Tila.Downed;
      }

   }

   void rising()
   {
      Debug.Log("Rising!");
      vihuNouseeYlos = true;
       // katsoo pelaajaa kohti kun otetaan rinnuksista kiinni
```

```
this.animation.Play("nousu");   // vihollinen nousee ylÃ¶s animaation avulla kohti, ennen nousemista

        if(RisingTimeri <=0)
        {
            vihuNouseeYlos= false;
            RisingTimeri = 0.0f;
            _tila = Tila.move;
        }

    }

    void dead()
    {
        Debug.Log("Dead!");
        // kun Paulin health loppuu niin aseta suoraan destroy(this.gameObject);
        if(vihuDeadTimer<=0)
        {
            vihuDeadTimer = 0;

            if(curHp <= 0)
            {
                this.animation.Stop();
                Debug.Log("Tuhosit vihun!");
                for (int i = 0; i < Random.Range(1, 10); i++) {
                    Instantiate(coin, transform.position, transform.rotation);
                }
                if(colliderissa == true)
                    Attack.enemiesOnTrigger--;
                Destroy(this.gameObject);
            }
        }
    }

    float timer(float s) {
    if (s > 0)
        s -= Time.deltaTime;
    if (s <= 0)
        s = 0;
    return s;
    }

    //NÃ¤itÃ¤ funktioita kutsutaan sendmessagella.
    void toHold(int s) {
        if (s == 1){
            pitoon = true;
            Debug.Log("PAULISTA KIINNI");
        }
    }

    void toThrown(int t){
        if (t == 1){
            heittoon = true;
        }
    }
```

```
void suunta(int s){
   if (s == 1){
      heittoZ = 0;
      heittoX = -4;
      knockbackZ = 0;
      knockbackX = -2;
   }

   if (s == 2) {
      heittoZ = 0;
      heittoX = 4;
      knockbackZ = 0;
      knockbackX = 2;
    }

    if (s == 3){
      heittoZ = 4;
      heittoX = 0;
      knockbackZ = 2;
      knockbackX = 0;
    }

    if (s == 4){
      heittoZ = -4;
      heittoX = 0;
      knockbackZ = -2;
      knockbackX = 0;
   }

   if (s == 5) {
      heittoZ = 3;
      heittoX = 3;
      knockbackZ = 1;
      knockbackX = 1;
   }
   if (s == 6){
      heittoZ = -3;
      heittoX = 3;
      knockbackZ = -1;
      knockbackX = 1;
    }
    if(s == 7){
      heittoZ = -3;
      heittoX = -3;
      knockbackZ = -1;
      knockbackX = -1;
    }
    if (s == 8){
      heittoZ = 3;
      heittoX = -3;
      knockbackZ = 1;
      knockbackX = -1;
    }
}
```

```
//Kombojen vastaanotto

    void iskuJ(int dmg)
    {
        if (vihuSelallaan_bool == false && _tila != Tila.Dead && curHp>0){
            if(this.animation.isPlaying)
            {
                this.animation.Stop();
            }
            _tila = Tila.Stun;
            curHp -= dmg;
            Instantiate(hitParticle, lhand.position, transform.rotation);
            audio.clip = hit1;
            audio.Play();
        }
    }

    void iskuJJ(int dmg)
    {
        if (vihuSelallaan_bool == false && _tila != Tila.Dead && curHp>0){
            if(this.animation.isPlaying)
            {
                this.animation.Stop();
            }
            _tila = Tila.Stun;
            curHp -= dmg;
            Instantiate(hitParticle, rhand.position, transform.rotation);
            audio.clip = hit2;
            audio.Play();
        }
    }

    void iskuJJJ(int dmg)
    {
        if (vihuSelallaan_bool == false && _tila != Tila.Dead && curHp>0){
            curHp -= dmg;
            Instantiate(hitParticle, lhand.position, transform.rotation);
            audio.clip = hit3;
            audio.Play();
        }
    }

    void iskuJJK(int dmg)
    {
        if (vihuSelallaan_bool == false && _tila != Tila.Dead && curHp>0){
            curHp -= dmg;
            Instantiate(hitParticle, lleg.position, transform.rotation);
            audio.clip = hit3;
            audio.Play();
        }
    }

    void saha(int dmg)
    {
        curHp -= dmg;
```

```
}

  void super(int s)
  {
     if (s == 1) {
        curHp = 0;
     }
  }

  // kun vihollinen osuu pelaajaan
  void OnTriggerStay(Collider otherObject) {

     if(otherObject.tag == "Enemy") {
        Tormays();
     }

     if(otherObject.tag == "PlayerDmg") {
        colliderissa = true;
     }

  }

  void Tormays()
  {
     if(vihutTormaa != true){
        vihutTormaa = true;
        lerpAjastin = 0.7f;

        tonaisyKerroinZ = Random.Range(-2, 2);
        tonaisyKerroinX = Random.Range(-2, 2);
     }
  }

}
```

FSM_HHBoss.cs

```
using UnityEngine;
using System.Collections;

public class FSM_HHBoss: MonoBehaviour {

    // BOSSIN TILAKONE

    // ASEEN PREFAB: FSM_BossGunPrefab
    public bool deadEnemy = false;
    // taistelun eri tilat
    public enum Tila {
        Active,
        Stun,
        Strike,
        move,
        Rising,
        RunToSide,
        Tilu,
        Dakka,
        SuperIsku,
        Dead
    }

    public GameObject hitParticle;

    Transform lhand;
    Transform rhand;
    Transform lleg;

    public AudioClip hit1;
    public AudioClip hit2;
    public AudioClip hit3;

    //paikallinen muuttuja taistelu
    public static Tila _tila;

    public Transform playeri;
    public Transform enemy;

    public GameObject FSM_HHBossGunPrefab;  // bossin ase
    public GameObject coin; // kolikko
    public GameObject BossDefeated;
    public Transform seina;

    GameObject storedData;

    //vÃ¤limatkan mittaamiseen tarvittavat muuttujat
    float liianLikella;
    float valimatka;
    float maksimiEtaisyys;
    float alkuValimatka;    // kun prefab laukaistaa ja lasketaan silloin pelaajan ja vihollisen
vÃ¤limatka
```

```
bool vihuHuomattuAluksi = false;

// Vihollisen move-tilaan liittyviÃ¤
public float vihun_nopeus; // vihollisen liikkumisnopeus
private float damage;   // bossin tekemÃ¤ damage
private float SuperDamage;

// vihollisen ampusin timeri, ettei ammu koko ajan
public float vihuAmpuuViive;
public float burstDelay;
public float lastFireTime;
public static bool vihuAmpuu = false;
public float AnimaatioTimer;
public bool AnimaatioPaalla = false;
public static int BossLuotiLaskuri;

//Ã¤Ã¤net
public AudioClip sarjatuli;
public AudioClip tilulilu;
public GameObject Musiikki;

//tilutus
public bool vihuTiluttaa;
public float SuperIskuTimer;

public float aika;
public float klipinaika;

// Downed-tilan timerin
public float vihuSelallaan_timer;
public bool vihuSelallaan_bool = false;

//Dead-tila
public float vihuDeadTimer;

// Rising-tilan timeri, vihu nousee ylos
public float RisingTimeri;
public bool vihuNouseeYlos = false;

// Bossin health
public static float curHp;
//Vector3 healthBarPos;

// vihollinen (pelaaja) makaa maassa ja sille nauretaan
private float vihuNauruTimer;
public bool vihuNauraa = false;

public bool NaytaBossDefeated;
public int BossDefeatedCounter;

// knocbackissa kÃ¤ytetyt muuttujat
float cTime = 0;
float cTime2 = 0;
int knockbackX;
```

```
int knockbackZ;
Vector3 startPos;
Vector3 endPos;

//vihollisten tÃ¶rmÃ¤ys/suunnanvaihto

Vector3 start;
Vector3 end;
bool vihutTormaa = false;
int tonaisyKerroinX;
int tonaisyKerroinZ;
float lerpAjastin;

//heitossa kÃ¤ytetyt muuttujat
int heittoX;
int heittoZ;
float trajectoryHeight = 2;

//holdissa kÃ¤ytetyt muuttujat
Transform dmgcollider;
//float kiinniCheck = 0f;
bool heittoon = false;
bool pitoon = false;

//Stunnissa kÃ¤ytetyt muuttujat
bool stunCheck = false;

bool colliderissa = false;

// Vihollisen siirtyminen reunaan BOSS
//public static int tilavaihdos;
Transform kohde;
float speed = 10f;
int worldcounter = 1;

// RunToSide-tila
public int randomReuna;
public bool VihuReunassa = false;
public float VihuReunassaTimeri;

// KÃ¤vely-tilan timeri
public float kavelyTimer;

// Creditsi
public float creditsTimer;
public bool aloitaCreditsTimer = false;

void Start () {

    Musiikki = GameObject.Find("Musiikki");

    lhand = GameObject.FindWithTag("vasen").transform;
    rhand = GameObject.FindWithTag("oikea").transform;
    lleg = GameObject.FindWithTag("varvas").transform;
```

```
    //randomReuna = Random.RandomRange(1,4);

    //animaatioiden toistotilat
    this.animation["isku"].wrapMode = WrapMode.Loop;
    this.animation["kavely"].wrapMode = WrapMode.Loop;
    this.animation["tilulilu"].wrapMode = WrapMode.Once;
    this.animation["juoksu"].wrapMode = WrapMode.Loop;
    this.animation["voimakas_isku"].wrapMode = WrapMode.Once;
    this.animation["ampuminen"].wrapMode = WrapMode.Loop;
    this.animation["kuolema"].wrapMode = WrapMode.Once;

    kavelyTimer = 5.0f; // vihollisen kÃ¤velyn kesto sekunteina

    //Vihollisten tÃ¶rmÃ¤ys/suunnanvaihto
    lerpAjastin = 0.7f;

    // asetetaan vihollisen tilaksi aluksi idle ja muutetaan sitÃ¤ myÃ¶hemmin
    // TÃ„SSÃ„ ASETETAAN VIHOLLISEN ENSIMMÃ„INEN TILA
    _tila = Tila.move;

    //etsitÃ¤Ã¤n pelaaja tagin perusteella
    dmgcollider = GameObject.Find("dmgCollider").transform;
    playeri = GameObject.FindWithTag("Player").transform;
    //enemy = GameObject.FindWithTag("Enemy").transform;


//  FSM_NTPaulGunPrefab = GameObject.FindWithTag("PaulinLuoti");

    // Bossin health aluksi
    curHp = 800.0f; // 800

    // vihollisen liikkumiseen tarvittavia muuttujia
    transform.LookAt(playeri);

    // mÃ¤Ã¤ritetÃ¤Ã¤n bossin tekemÃ¤ damage nyrkeillÃ¤
    damage = 0.15f;
    SuperDamage = 0.5f;

    // alustetaan muuttujat
    liianLikella = 7.0f;
    vihun_nopeus = 22.0f;   //sÃ¤Ã¤dÃ¤ Paulin nopeutta

    // vihollisen ampuma timer
    burstDelay = 0.1f;  // eli ampuu 0,5 sekunnin vÃ¤lein pelaajaa eli sarjatulta
    lastFireTime = 0.0f; // nollaus
    AnimaatioTimer = 2.0f;
    BossLuotiLaskuri = 0;
    vihuAmpuuViive = 2.0f;

    // downed-tilassa vihollinen kaksi sekuntia maassa
    vihuSelallaan_timer = 1.8f;

    // Rising-tilassa oleva nousemistimer
    RisingTimeri = 0.8f;
```

```
        //kun vihollinen kuolee, niin toistetaan animaatio tietyn ajan
        vihuDeadTimer = 1.8f;

        // laugh-tilassa viholliselle asetetaan parin sekunnin timer
        vihuNauruTimer = 2.0f;

        // move-tilan muuttujien alustus
        maksimiEtaisyys = 2.0f; // vihollinen jÃ¤Ã¤ 0.7:n pÃ¤Ã¤hÃ¤n pelaajasta, eikÃ¤ tule kiinni

        // creditsien timer, kuinka kauan odotetaan ennen siirtymistÃ¤?
        creditsTimer = 10.0f;

        //heathbarin sijainti
        //healthBarPos = new Vector2(Screen.width - Screen.width / 1.7f, Screen.height - Screen.height
/ 3);

        //healthBarPos = enemy.transform.position;

        // Vihujen erotusTimer ja erotuskerroin
        //erotusTimer = 2.0f;
        //erotuskerroin = 3.0f;

         transform.rotation = Quaternion.Euler(0,0,0);

         VihuReunassaTimeri = 7;    // vihollinen sekuntteina reunassa

        kohde = transform;

        vihuTiluttaa = false;
        SuperIskuTimer = 1.0f;

        randomReuna = 0;

        storedData = GameObject.Find("gui_kehys");

    }

    /* UPDATE*/

    void Update ()
    {
          // muutetaan eri taistelutiloja kÃ¶tevÃ¤sti switchillÃ¤
          switch(_tila) {

              case Tila.Active:
                  toactive();
                  break;

              case Tila.Stun:
                  stun();
                  break;

              case Tila.Strike:
                  strike();
```

```
break;

            case Tila.move:
                tomove();
                break;

            case Tila.Rising:
                rising();
                break;

            case Tila.RunToSide:
                runtoside();
                break;

            case Tila.Tilu:
                tilu();
                break;

            case Tila.Dakka:
                dakka();
                break;

            case Tila.SuperIsku:
                superisku();
                break;

            case Tila.Dead:
                dead();
                break;

        }


    // KÃ¤velytimerin asetukset
    if(_tila == Tila.move || _tila == Tila.Strike)
    {
        kavelyTimer -= Time.deltaTime;

        if(kavelyTimer<=0)
        {
            kavelyTimer = 0;
            _tila = Tila.RunToSide;
        }
    }


    // luodaan vektori eulerAngles, ottaa paulin sen hetkisen rotation ja asettaa samaan vektoriin y-komponentin.
    // kÃ¤Ã¤ntÃ¤Ã¤ y-komponentin mukaan, muut ovat nollia.

    Vector3 eulerAngles = transform.rotation.eulerAngles;
    eulerAngles = new Vector3(0, eulerAngles.y, 0);
    transform.rotation = Quaternion.Euler(eulerAngles);
```

```
if (curHp <= 0){
        deadEnemy = true;
        _tila = Tila.Dead;
    }

        // lasketaan vÃ¤limatka vihollisen ja pelaajan vÃ¤lillÃ¤
     valimatka = Vector3.Distance(playeri.position, transform.position);

     if (valimatka < 2f && _tila == Tila.move)
        _tila = Tila.Strike;
     else if (valimatka > 2.5f && _tila == Tila.Strike)
        _tila = Tila.move;


    // vihollisen ampusen timerin viive pitÃ¤Ã¤ olla updatessa, ettÃ¤ pÃ¤ivittyy

    // downed-tilan laskuria pÃ¤ivitetÃ¤Ã¤n

    if(AnimaatioPaalla == true)
    {
        AnimaatioTimer -= Time.deltaTime;
    }


    if(_tila == Tila.Dead)
    {
        vihuDeadTimer -= Time.deltaTime;
    }


    // BOSSI JUOKSEE REUNAAN

    // kun ei olla runtosidetilassa
    if(_tila != Tila.RunToSide && _tila !=Tila.Dakka && _tila != Tila.Tilu)
    {
        RandomiReuna();
        VihuReunassaTimeri = 7; // alustetaan vihuReunassatimeri alkuperÃ¤iseen arvoonsa
        VihuReunassa = false;    // vihollinen ei ole reunassa, kun tila on joku muu kuin runtoside

    }

    if(VihuReunassa == true)
    {
        kavelyTimer = 5.0f; // kun vihollinen on reunassa, niin asetetaan kÃ¤velytimer alkuperÃ¤iseen arvoonsa.

        float distance = Vector3.Distance(kohde.position, transform.position);

        Vector3 delta = kohde.transform.position - transform.position;
        delta.Normalize();
        float moveSpeed = speed * Time.deltaTime;

        transform.position = transform.position + (delta * moveSpeed);
```

```
        if (distance < 0.1)
          {
             transform.position = kohde.position;
          }


          if(VihuReunassaTimeri>0)
             VihuReunassaTimeri -= Time.deltaTime;

          if(VihuReunassaTimeri<=0)
          {
             VihuReunassaTimeri = 0;
             _tila = Tila.move; // kun timeri menee nollaan laitetaan seuraavaksi tilaksi
          }

       }

       // kun tila vaihdetaan dakkaksi, niin vihuAmpuu viive ajastin lÃ¤htee menemÃ¤Ã¤n alaspÃ¤in
       if(_tila == Tila.Dakka)
       {
          vihuAmpuuViive -= Time.deltaTime;

          if(vihuAmpuuViive<=0)
             vihuAmpuuViive = 0;

       }

       if(_tila != Tila.Dakka)
       vihuAmpuuViive = 0;


       // jos tila on RunToSide, vihollinen katsoo seinÃ¤Ã¤ pÃ¤in, kunnes saavuttaa seinÃ¤n. TÃ¤mÃ¤n
jÃ¤lkeen vihollinen kÃ¤Ã¤nnetÃ¤Ã¤n eri tilassa
       // katsomaan pelaajaa
       if(_tila == Tila.RunToSide && _tila != Tila.Dakka)
       {

          transform.LookAt(new Vector3(kohde.position.x, 0, kohde.position.z)); //
ampuu pelaajaa kohti


       }

       if(_tila != Tila.Dakka && audio.clip == sarjatuli)
       {
          if(audio.isPlaying)
          {
             audio.Stop();
          }
       }else if(_tila == Tila.Dakka && audio.clip == sarjatuli)
       {
          if(!audio.isPlaying)
          {
             audio.Play();
          }
```

```
}

        // varmistetaan Ã¤Ã¤nien lopetus tilasiirtymien vÃ¤lillÃ¤
        if(_tila != Tila.Tilu && audio.clip == tilulilu)
        {
            if(audio.isPlaying)
            {
                audio.Stop();
            }
        }else if(_tila == Tila.Tilu && audio.clip == tilulilu)
        {
            if(!audio.isPlaying)
            {
                audio.Play();
            }
        }


        // vaihdetaan kohdetta suoraan updatesta, koska kohde ei pÃ¤ivity uudestaan funktion sisÃ¤llÃ¤
        if(randomReuna == 1)
        {
            kohde = GameObject.Find("HenryBoss_Seina").transform;
        }else if(randomReuna == 2 || randomReuna == 3 || randomReuna == 4)
        {
            kohde = GameObject.Find("HenryBoss_Seina2").transform;
        }

        // Jos ollaan superisku-tilassa ja riittÃ¤vÃ¤n lÃ¤hellÃ¤ pelaajaa
        if(_tila == Tila.SuperIsku)
        {

            vihun_nopeus = 5.0f;

             if(valimatka > 2f) // jos vihollinen yli 2 pÃ¤Ã¤ssÃ¤ pelaajasta niin juoksee
             {
                this.animation.Play("juoksu");
             }

            else if (valimatka < 2f) // muutoin jos alle kahden pÃ¤Ã¤ssÃ¤
             {

                if(this.animation.IsPlaying("juoksu"))  // lopetetaan juoksu
                    this.animation.Stop();

                if(!this.animation.IsPlaying("juoksu")) // aloitetaan isku // tÃ¤hÃ¤n superisku
timer
                {

                    SuperIskuTimer -= Time.deltaTime;

                    if(SuperIskuTimer>0)
                    {
                        this.animation.Play("voimakas_isku");
```

```
 storedData.SendMessage("ApplyDamage", SuperDamage, SendMessage-
Options.DontRequireReceiver);
                }
                else if(SuperIskuTimer <=0)
                {

                    _tila = Tila.move;
                }

            //if(!this.animation.Play("voimakas_isku")) // tehdÃ¤Ã¤n isku kerran ja vaihdetaan tilaa

            }

        }
    }

    if(_tila != Tila.SuperIsku)
    {
        SuperIskuTimer = 1.0f;  // alustetaan superiskutimer uudestaan seuraava iskua varten
        vihun_nopeus = 22.0f;
    }


    if(BossDefeatedCounter >= 1) // luodaan vain yski bossdefeated teksti tÃ¤llÃ¤ tavaoin
    {
        NaytaBossDefeated = false; // ei enÃ¤Ã¤ instantioida prefabia
        aloitaCreditsTimer = true;
        // vaihda biisiÃ¤, kun bossi tapetaan
    }

    if(aloitaCreditsTimer)
    {
        creditsTimer -= Time.deltaTime;

        if(creditsTimer<=0)
            Application.LoadLevel("credits");

    }


}

// KUTSUTTAVAT TILAKONEEN FUNKTIOT JÃ„RJESTYKSESSÃ„
//-------------------------------------------------------------------------------------------------
// //taistelun funktiot, joita kutsutaan. Eli mitÃ¤ halutaan tehdÃ¤?
// nÃ¤itÃ¤ kutsutaan switch-case rakenteesta.


void toactive()
{
    Debug.Log("Active!");
    // Vihollinen alkaa tekemÃ¤Ã¤n toimintoja
    _tila = Tila.move;
}
```

```
    void stun()
    {

        this.animation.Stop();


        Debug.Log("Stun!");

        if (stunCheck == true)
            stunCheck = false;

        cTime2 += Time.deltaTime;

        if (cTime2 > 1.5f){
            cTime2 = 0;
            _tila = Tila.move;
        }
        // Vihollinen ottanut lyÃ¶nnin, estÃ¤mÃ¤ssÃ¤ vihollista lyÃ¶mÃ¤stÃ¤ vÃ¤littÃ¶mÃ¤sti takai-
sin.
    }

    void strike()
    {
        if(Time.timeScale!=0)
        {

            if(!this.animation.IsPlaying("isku"))   // stopataan edellisen tilan animaatio
            {
                this.animation.Stop();
            }

            transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));   //
vihollinen lyÃ¶ pelaajaan suuntaan

            this.animation.Play("isku");
            this.animation["isku"].speed = 2.5f;
            // Vihu lyÃ¶ pelaajaa nyrkeillÃ¤, kun vihollinen (Paul) on tarpeeksi lÃ¤hellÃ¤ pelaajaa
            // valimatka-funktio on mÃ¤Ã¤ritelty aikaisemmin updatessa.

            // Paulin tekemÃ¤Ã¤ damagea voidaan muuttaa ylhÃ¤Ã¤llÃ¤ olevalla damage-muuttujalla
kÃ¤tevÃ¤sti.
            Debug.Log("Strike!");
        //  this.animation.Play("lyonti");
        //  this.animation["lyonti"].speed = 2.5f;  // nopeutetaan animaatiota 2,5-kertaiseksi, jotta isku
nÃ¤yttÃ¤Ã¤ vauhdikkaalta
            storedData.SendMessage("ApplyDamage", damage, SendMessage-
Options.DontRequireReceiver);

        }

    }


    void tomove()   // Move-tila
```

```
{
    if(Time.timeScale!=0)
    {

        this.animation.Play("kavely");  // soitetaan kavely

        Debug.Log("move!");
        // Vihollinen liikkuu suhteessa viholliseen
        transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));

        // jos on kauempana kohteesta kuin max_etaisyys, liikkuu pelaajaa kohti
        if(valimatka > maksimiEtaisyys )
        {
            transform.Translate(Vector3.forward / vihun_nopeus, Space.Self);
        }

    }

}

void rising()
{
    Debug.Log("Rising!");
    // Vihu nousee pystyyn

    vihuNouseeYlos = true;
     // katsoo pelaajaa kohti kun otetaan rinnuksista kiinni
//  this.animation.Play("ylos");    // vihollinen nousee ylÃ¶s animaation avulla
// transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z)); // suuntautuu vihollista
kohti, ennen nousemista

    if(RisingTimeri <=0)
    {
        vihuNouseeYlos= false;
        RisingTimeri = 0.0f;
        _tila = Tila.move;
    }

}


void runtoside()
{
    if(Time.timeScale!=0)
    {
        // vihu juoksee sivulle

        VihuReunassa = true;

        this.animation.Play("juoksu");

        if(VihuReunassa)
        {
```

```
 if (_tila == Tila.RunToSide && transform.position == kohde.position && random-
Reuna == 1)
            {
               //randomReuna = 1;
               kohde = GameObject.Find("HenryBoss_Seina").transform;
               transform.LookAt(new Vector3(kohde.position.x, 0, kohde.position.z));
            }

            else if (_tila == Tila.RunToSide && transform.position == kohde.position
&& randomReuna == 2)
            {
               //randomReuna = 2;
               kohde = GameObject.Find("HenryBoss_Seina").transform;
               transform.LookAt(new Vector3(kohde.position.x, 0, kohde.position.z));
            }

            else if (_tila == Tila.RunToSide && transform.position == kohde.position
&& randomReuna == 3)
            {
               //randomReuna = 3;
               kohde = GameObject.Find("HenryBoss_Seina2").transform;
               transform.LookAt(new Vector3(kohde.position.x, 0, kohde.position.z));
            }

            else if (_tila == Tila.RunToSide && transform.position == kohde.position
&& randomReuna == 4)
            {
               //randomReuna = 4;
               kohde = GameObject.Find("HenryBoss_Seina2").transform;
               transform.LookAt(new Vector3(kohde.position.x, 0, kohde.position.z));
            }

         }

      }


   }


   void tilu()
   {
         // BOSSIN KITARAN TILUTUS
      // Bossi tiluttaakitaraansa. Jos ei keskeytetÃ¤ lyÃ¶mÃ¤llÃ¤, tilutetaan loppuu asti ja
hyÃ¶kÃ¤tÃ¤Ã¤n pelaajan kimppuun.

      if(vihuTiluttaa == false)   // asetetaan vihutiluttaa trueksi, ettei lÃ¤hetÃ¤ koko ajan
falsea, muuten kitara ei soi oikein vaan tÃ¶kkii :)
      {
         //transform.LookAt(new Vector3(kohde.position.x, 1, kohde.position.z));
         vihuTiluttaa = true;

         if(!audio.isPlaying)
         {
```

```
            audio.clip = tilulilu;
            audio.Play();
        }

        this.animation["tilulilu"].speed = 0.9f;
        this.animation.Play("tilulilu"); // toistaan tilutuksen animaatio

        transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));

    }

    if(!this.animation.IsPlaying("tilulilu"))
    {
        vihuTiluttaa = false; // resetoidaan tilutus, jotta voidaan kÃ¤yttÃ¤Ã¤ myÃ¶s myÃ¶hemmin
        _tila = Tila.SuperIsku; // seuraava tila on superisku
    }

    // tilutuksen jÃ¤lkeinen lyÃ¶nti on massiivinen 100hp.

}

void dakka()    // ampuu sivusta vihollista
{
    //TÃ„HÃ„N MYÃ-S ANIMAATIO AMPUMISESTA!
    //this.animation.Stop();

    this.animation.Play("ampuminen");

    int panostenLkm = 17;   // tÃ¤stÃ¤ pystyy sÃ¤Ã¤tÃ¤Ã¤mÃ¤Ã¤n kerralla ammuttavien panosten lukumÃ¤Ã¤rÃ¤Ã¤

    // vihu nostaa aseen ampumista varten
    transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z)); // ampuu pelaajaa kohti

        //this.animation.CrossFade("ase_nosto", 0.8f);
        //this.animation.CrossFade("ampuminen", 1.0f);

        if(Time.time > ( lastFireTime + burstDelay) && BossLuotiLaskuri< panostenLkm)
        {
            vihuAmpuu = true;
            lastFireTime= Time.time;
            Instantiate(FSM_HHBossGunPrefab, transform.position, transform.rotation);
            BossLuotiLaskuri++;
        }

        // kun ensimmÃ¤inen luoti ammutaan, asetetaan samalla sarjatuli-Ã¤Ã¤niklippi pyÃ¶rimÃ¤Ã¤n
        if(BossLuotiLaskuri == 1)
        {
            audio.clip = sarjatuli;
            audio.Play();
```

```
}


        if(BossLuotiLaskuri == panostenLkm && vihuAmpuuViive <=0)   // jos luotilaskuri ja vihuampuu viive ovat nollia, niin asetetaan ampuminen falseksi ja vihu ampuuu viive 2. Lopuksi true.
        {
            vihuAmpuu = false;
            BossLuotiLaskuri = 0;   // nollataan laskuri
            vihuAmpuuViive = 2; // viive asetetaan alkuperÃ¤iseksi
            vihuAmpuu = true;

        }

        if(VihuReunassaTimeri <=0)
        {
            VihuReunassa = false;
            _tila = Tila.move;
        }

    }

    void superisku()
    {
        //TÃ„TÃ„ ISKUA KÃ„YTETÃ„Ã„N TILUTUKSEN JÃ„LKEEN, MIKÃ„LI BOSSIN TILUTTELUA EI PYSÃ„YTETÃ„!

        transform.LookAt(new Vector3(playeri.position.x, 0, playeri.position.z));

        // jos on kauempana kohteesta kuin max_etaisyys, liikkuu pelaajaa kohti
        if(valimatka > maksimiEtaisyys )
        {
            maksimiEtaisyys = 0.5f;
            transform.Translate(Vector3.forward / vihun_nopeus, Space.Self);
        }
    }

    void dead()
    {
        Debug.Log("Dead!");

        this.animation.Play("kuolema");
        if(vihuDeadTimer<=0)
        {
            this.animation.Stop();

            if(curHp <= 0 && this.animation.IsPlaying("juoksu") == false)
            {
                NaytaBossDefeated = true;
                Debug.Log("Tuhosit vihun!");
            /*  for (int i = 0; i < Random.Range(1, 10); i++) {
                Instantiate(coin, transform.position, transform.rotation);
            } */
                if(colliderissa == true)
                Attack.enemiesOnTrigger--;
```

```
            if(NaytaBossDefeated == true && BossDefeatedCounter <1)
            {
                Instantiate(BossDefeated, new Vector3(580, 4, 5), new Quaterni-
on(0,0,0,0));
                Music.music = 4;
                BossDefeatedCounter++;
            }
        }
    }
}

// MUUT FUNKTIOT

// bossin tilan vaihto muualta
public static void vaihdaTila(int tilavaihdos)
{
    if(tilavaihdos == 1)
    {
        _tila = Tila.Dakka;  //ampuminen
    }

    if(tilavaihdos == 2)
    {
        _tila = Tila.Tilu;  //tilutus
    }
}

// arvotaan satunnainen luku ja kutsutaan tÃ¤tÃ¤ funktiota updatessa
void RandomiReuna()
{
    randomReuna = Random.RandomRange(1,4);
}

float timer(float s) {
if (s > 0)
    s -= Time.deltaTime;
if (s <= 0)
    s = 0;
return s;
}

//NÃ¤itÃ¤ funktioita kutsutaan sendmessagella.
void suunta(int s){
    if (s == 1){
        heittoZ = 0;
        heittoX = -4;
        knockbackZ = 0;
```

```
            knockbackX = -2;
        }

        if (s == 2) {
            heittoZ = 0;
            heittoX = 4;
            knockbackZ = 0;
            knockbackX = 2;
         }

         if (s == 3){
            heittoZ = 4;
            heittoX = 0;
            knockbackZ = 2;
            knockbackX = 0;
         }

         if (s == 4){
            heittoZ = -4;
            heittoX = 0;
            knockbackZ = -2;
            knockbackX = 0;
        }

        if (s == 5) {
            heittoZ = 3;
            heittoX = 3;
            knockbackZ = 1;
            knockbackX = 1;
        }
        if (s == 6){
            heittoZ = -3;
            heittoX = 3;
            knockbackZ = -1;
            knockbackX = 1;
         }
         if(s == 7){
            heittoZ = -3;
            heittoX = -3;
            knockbackZ = -1;
            knockbackX = -1;
         }
         if (s == 8){
            heittoZ = 3;
            heittoX = -3;
            knockbackZ = 1;
            knockbackX = -1;
         }
    }


    //Kombojen vastaanotto
```

```
void iskuJ(int dmg)
  {
     if(_tila == Tila.Tilu)
     {
        curHp -= 50.0f;
        _tila = Tila.Stun;
        VihuReunassa = false;
        vihuTiluttaa = false;
     }else{
        curHp -= dmg;
        }
        Instantiate(hitParticle, lhand.position, transform.rotation);
        audio.clip = hit1;
        audio.Play();
    }

  void iskuJJ(int dmg)
  {
     if(_tila == Tila.Tilu)
     {
        curHp -= 50.0f;
        _tila = Tila.Stun;
        VihuReunassa = false;
        vihuTiluttaa = false;

     }else{
        curHp -= dmg;
     }
        Instantiate(hitParticle, rhand.position, transform.rotation);
        audio.clip = hit2;
        audio.Play();

  }

  void iskuJJJ(int dmg)
  {
     Debug.Log("Bossi JJJ");
  if(_tila == Tila.Tilu)
     {
        curHp -= 50.0f;
        _tila = Tila.Stun;
        VihuReunassa = false;
        vihuTiluttaa = false;

     }else{
        curHp -= dmg;
     }
        Instantiate(hitParticle, lhand.position, transform.rotation);
        audio.clip = hit3;
```

```
            audio.Play();
            _tila = Tila.Stun;

    }

    void iskuJJK(int dmg)
    {
       Debug.Log("Bossi JJK");
    if(_tila == Tila.Tilu)
       {
            curHp -= 50.0f;
            _tila = Tila.Stun;
            VihuReunassa = false;
            vihuTiluttaa = false;

       }else{

            curHp -= dmg;
       }
            Instantiate(hitParticle, lleg.position, transform.rotation);
            audio.clip = hit3;
            audio.Play();
            _tila = Tila.Stun;

    }


    void saha(int dmg)
    {
       curHp -= dmg;
    }


}
```

FSM_HHBossGunPrefabScript.cs

```
using UnityEngine;
using System.Collections;

public class FSM_HHBossGunPrefabScript : MonoBehaviour {

    public float luodinNopeus;

    // Bossin aseesta tuleva damage
    private float damage2 = 0.3f;

    //public Transform Vihu;
    public GameObject Vihu; // gameobject ja transform erikseen, jotta distance toimii oikein.
    public Transform Vihu2;
    GameObject storedData;

    void Start () {

        Vihu = GameObject.Find("Player");
        Vihu2 = GameObject.Find("Player").transform;
        storedData = GameObject.Find("gui_kehys");

        luodinNopeus = 15.0f;   // mÃ¤Ã¤ritellÃ¤Ã¤n luodin nopeus
        transform.Rotate(0,0,90);   // laitetaan luoti vaaka-asentoon

        transform.position = transform.position + new Vector3(0,1.0f,0);    // luodin
lÃ¤htÃ¶position mÃ¤Ã¤ritteleminen // 0.5f
    }

    void Update () {

        // Jos luoti menee tarpeeksi kauas Paulista, niin luodin prefab tuhotaan. NÃ¤in ei viedÃ¤ turhaan
resursseja.....
        if(Vector3.Distance(transform.position,Vihu2.position) > 20)
        {
            Destroy(this.gameObject);
        }
         // liikutetaan luotia kovalla vauhdilla
        transform.Translate(Vector3.forward * Time.deltaTime * luodinNopeus);
     }

    // laita tÃ¤hÃ¤n vihollisen (Paulin) luodin vaikutus pelaajaan!!!
    void OnTriggerStay(Collider otherObject)
    {
        if (otherObject.tag == "Player")
        {
            Debug.Log("Paul osuu pelaajaa!");
            //lÃ¤hettÃ¤Ã¤ viestin
            storedData.SendMessage("ApplyDamage", damage2, SendMessage-
Options.DontRequireReceiver);
        }
     }
}
```