LAPIN AMK
Lapland University of Applied Sciences

ADAPTIIVINEN PELIMAAILMA

Joonas Häkkinen

Opinnäytetyö
Tekniikan ja liiketalouden ala
Tieto- ja viestintätekniikka
Insinööri (AMK)

2016

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tekniikan ja liiketalouden ala
Tieto- ja viestintätekniikka
Insinööri

Tekijä Joonas Häkkinen
Ohjaaja Tauno Tepsa
Toimeksiantaja
Vuosi 2016
Työn nimi Adaptiivinen pelimaailma
Sivu- ja liitemäärä 40 + 0

Opinnäytetyön tavoitteena on tutkia tietokonepelien adaptiivisuuden logiikkaa. Lisäksi tavoitteena on toteuttaa toimiva adaptiivisuuden logiikka tietokonepeliin Unity3D-pelimoottorilla. Aiheen valinta pohjautuu kirjoittajan omaan kiinnostukseen pelien tekemisestä ja tietokonepeleistä. Viimeisien vuosien aikana peliala on kasvanut Suomessa nopeasti, mikä on lisännyt alan koulutuksen ja ammattilaisten tarvetta.

Opinnäytetyössä tutkitaan ja avataan erilaisia menetelmiä adaptiivisien ominaisuuksien toteuttamiseen. Tutkimusmateriaalina on käytetty kirjoittajan omia kokemuksia. Lisäksi materiaalina on käytetty eri tietokonepelien wikia-sivustoja, jotka koostuvat muiden pelaajien kokemuksista ja johtopäätöksistä. Suunniteltu ja toteutettu pelilogiikka pohjautuu kirjoittajan omaan peli-ideaan.

Koska aihe on suhteellisen laaja, eivätkä tavoitteet ole eksakteja, voidaan useat opinnäytetyössä esitetyt menetelmät toteuttaa usealla eri tavalla. Työn tarkoituksena on selventää pintapuolisesti, mitä asioita tulee ottaa huomioon pelin kehityksen alkuvaiheessa teknisestä näkökulmasta. Lisäksi opinnäytetyössä esitetään muutama tapa adaptiivisuuden luomiseen tietokonepeliin.

Asiasanat pelimaailma, tietokonepeli, Unity3D

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

School of Technology and business
Information Technology

Author	Joonas Häkkinen	Year	2016
Supervisor	Tauno Tepsa		
Commissioned by			
Subject of thesis	Adaptive Game World		
Number of pages	40 + 0		

The objective of this thesis was to research computer game logics for adaptivity and to create working logic for adaptivity with Unity3D game engine. The subject was chosen based on author's own interest in the game development and computer games. During the past few years game industry has grown in Finland quite rapidly, which has increased the demand of education and professionals in industry.

In this thesis different methods to create adaptive aspects for computer game were researched and explained. The research material is based on authors own experiences. In addition the game wikia websites that contain experiences and conclusions of other gamers were used as material. The game logics that were designed and created is based on author's own game idea.

Since the subject was rather wide and the objectives were not precise, many of the methods described in the thesis can be implemented in several ways. The thesis explained what to consider at the early stages of game development process superficially from a technical point of view. In addition the thesis explains few different ways to create adaptivity for computer games.

Key words Game, Game World, Unity3D

SISÄLLYS

ALKUSANAT

Haluan kiittää Rovaniemen Pelilabra -yhteisöä, joka on pitkälti mahdollistanut oman osaamiseni kehittämisen pelien tekemisessä. Yhteisö on mahdollistanut graafikoiden ja ohjelmoijien yhteistyön erilaisten peliprojektien toteuttamisessa.

Haluan myös kiittää opettajaamme Petri Hannulaa, joka on omalla ammattitaidollaan ja aktiivisuudellaan ohjannut sekä kannustanut pelilabra -yhteisöä kehittymään sen alkutaipaleella. Lisäksi olemme saaneet kokeneelta tekijältä arvokkaita neuvoja oman osaamisemme kehittämisessä. Hän on myös vapaa-ajallaan järjestänyt pelikehitykseen liittyviä tapahtumia Rovaniemellä, ja näin kasvattanut pelikehityksestä kiinnostuneiden henkilöiden yhteisöä.

KÄYTETYT MERKIT JA LYHENTEET

AI	Tekoäly (augmented intelligence)
NPC	Pelihahmo jota pelaaja ei ohjaa (non player chararacter)
Scripti	Pelin toimintaa suorittava kooditiedosto
Tag	Tunniste
UI	Käyttöliittymä (User Interface)

1 JOHDANTO

Opinnäytetyössäni käsitellään tietokonepelin adaptiivisuutta, eli pelimaailman muuntautumista pelin edetessä. Teoriaosuudessa käydään läpi adaptiivisuutta yleisesti sekä tutkitaan, miten se on toteutettu kahdessa eri tietokonepelissä. Käytännön osuudessa esitetään pelin suunnittelua sekä, mitä asioita suunnitteluvaiheessa tulee ottaa huomioon. Lisäksi toteutin adaptiivisuuden ydinlogiikan suunnitelman mukaisesti. Käytin työssäni Unity3D pelimoottoria sekä sen kehitysympäristöä Unity3D Editoria.

Aiheen valinta pohjautuu omaan kiinnostukseeni pelikehitystä kohtaan. Olen aina pitänyt tietokonepeleistä. Opiskeluaikana kiinnostus pelien tekemistä kohtaan kasvoi ohjelmointitaitojen kehittyessä. Opiskelun ohella olen ollut mukana usean harrastepeliprojektin toteutuksessa. Opinnäytetyön käytännön osuuden peli-idea pohjautuukin juuri yhden aloitetun harrasteprojektin peli-ideaan, jota on haudottu muutaman vuoden ajan.

Opinnäytetyön tavoitteena on selvittää eri menetelmät adaptiivisuuden toteutukselle tietokonepeliin. Lisäksi käytännön osuudessa logiikan tulisi toimia siten, että siihen ei tarvitse tehdä suurempia muutoksia mahdollisen jatkokehityksen aikana. Opinnäytetyön tarkoituksena puolestaan ei ole toteuttaa pelin toimintaan liittyviä ominaisuuksia, vaan painopiste on vahvasti pelimaailman muuntautumista ohjaavassa logiikassa.

2 ADAPTIIVINEN PELIMAAILMA

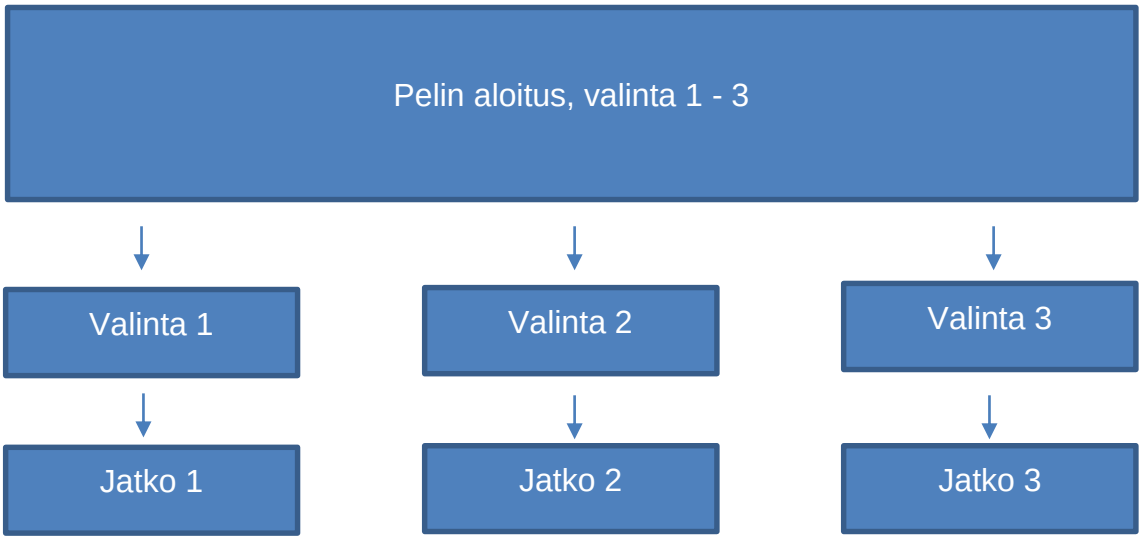
2.1 Yleisesti

Adaptiivisuus on nykypäivänä isossa roolissa tietokone- sekä pelikonsolipeleissä. Adaptiivisuudella peleissä tarkoitetaan pelin reagointia pelaajan tekemiin valintoihin tai pelimaailman mukautumista pelaajan pelitavan mukaan. Yksinkertaisin esimerkki adaptiivisesta pelistä lienee tekstiseikkailupelit, joissa peli kertoo pelaajalle tarinaa tekstin muodossa, ja pelaajan on tehtävä valintoja eri vaihtoehdoista tarinan jatkamiseksi. Pelin tarina jatkuu sen mukaan, minkä valinnan pelaaja valitsee. Adaptiivisuudella peleihin saadaan muuttuvia elementtejä, joiden ansiosta jokainen läpipeluukerta on erilainen.

Pelin adaptiivisuuden rakentamisessa voidaan käyttää erilaisia menetelmiä. Adaptiivisuus peliin voidaan rakentaa pelaajan tekemien valintojen pohjalle, jolloin pelaajan tulevat tehtävät pohjautuvat esimerkiksi NPC-hahmon kanssa käydyn dialogin aikana tehtyyn päätökseen. Nämä valinnat voivat sulkea pois muut mahdolliset tehtäväketjut samalla läpipeluukerralla. Joissakin peleissä pelimaailma muuttuu pelaajan pelitavan mukaan. Jos pelaaja esimerkiksi pelaa aggressiivisemmin, maailma muuttuu aggressiivisemmaksi pelaajaa kohtaan. Pelimaailman adaptiivisuus voidaan toteuttaa myös satunnaisuuden pohjalta, jolloin pelin muuttuvat elementit kyseisellä läpipeluukerralla eivät ole täysin pelin kehittäjänkään tiedossa. Kuviossa 1 on esitetty yksinkertainen malli adaptiivisuudesta.

Kuvion 1 mukaan valinta on absoluuttinen, eikä mahdollisuutta uuteen valintaan anneta pelaajalle. Kyseistä mallia voidaan käyttää, kun adaptiivisuus rakennetaan pelaajan tekemien valintojen pohjalle.

Adaptiivisissa peleissä pelaajan tekemät valinnat voivat vaikuttaa pelin loppuratkaisuun. Useammalla erilaisella läpipeluukerralla pelisuunnittelijat pyrkivät lisäämään pelaajien kiinnostusta peliä kohtaan. Samassa pelissä eri läpipeluukerroilla juoni, pelimekaniikat tai hahmokehitys voivat olla jopa täysin erilaisia.



Kuvio 1. Esimerkki yksinkertaisesta adaptiivisuudesta

2.2 Dishonored

Dishonored on Arkane Studiosin kehittämä ja Bethesda Softworksin® vuonna 2012 julkaisema pelikonsoli- ja tietokonepeli. Peli on kehitetty Unreal Engine 3 pelimoottorilla ja se on pelattavissa PlayStation® 3 ja Xbox 360 -pelikonsoleilla sekä tietokoneella. (Bethesda Softworks 2012.)

Pelin juoni keskittyy fiktiiviseen, rottien levittämän ruton riivaamaan, Dunwallin kaupunkiin. Pelin päähenkilö on kaupungin hallitsijattaren henkivartija Corvo Attano, joka lavastetaan syylliseksi hallitsijattaren murhaan sekä kruununperillisen Emilyn kidnappaukseen. Ennen teloitustaan Corvo pakenee vankilasta ja palaa Dunwallin kaduille. Naamion taakse piiloutuneen sekä yliluonnollisia kykyjä omaavan salamurhaaja Corvon tehtävänä on löytää Emily, ja palauttaa tyttö valtaistuimelle. Apunaan hänellä on pieni kruunulle uskollinen ryhmä sekä jumalhahmo, joka kantaa nimeä Outsider.

Pelin adaptiivisuutta ohjataan kaaossysteemillä (Chaos system), joka seuraa pelimaailman tilaa. Pelaaja saa yhteenvedon jokaisen läpi pelatun tason jälkeen omasta suorituksestaan, jossa kerrotaan kyseisen tason kaaos (Low, High) yhdessä muun statistiikan kanssa. Yhteenvedosta käy myös ilmi läpipeluukerran

kokonaistilanne kaaoksen suhteen (Low, High). Kaaossysteemi on toteutettu pelimaailman muuntautumisena pelaajan pelitavan mukaan. Esimerkiksi ympäristön yleinen vihamielisyys muuttuu pelitavan aggressiivisuudesta riippuen. Mitä enemmän pelaaja tappaa vastaan tulevia henkilöitä, sitä enemmän kaupungissa on ruttoa levittäviä rottia sekä ruton saaneita NPC hahmoja. Rotat sekä ruton saaneet NPC hahmot hyökkäävät ohikulkijoiden kimppuun pienestäkin ärsykkeestä lisäten näin kaaoksen määrää. Pelaaja voi myös suorittaa pelin varsinaisen tarinan lisäksi lisätehtäviä, jotka mahdollistavat erilaisia vaihtoehtoja varsinaisen tarinan läpäisemiseksi. Lisätehtävillä voi vaikuttaa tason lopputulokseen. Lisäksi peli on mahdollista läpäistä tappamatta yhtään vihollista. (Dishonored Wikia 2012a.)

Erilaisia laskelmia on esitetty kaaoksen kertymästä. Esimerkiksi, jos Corvo tappaa 20 prosenttia pelitason aikana vastaan tulevasta väestöstä, läpipelatun tason palaute kertoo kaaoksen olevan korkea. On myös esitetty väite, että jos Corvon tappamien ihmisten yhteenlaskettu määrä koko pelimaailman väestöstä ylittää 50 prosenttia, määräytyy läpipeluukerran kaaos korkeaksi. Tällöin siihen ei voi enää vaikuttaa. Väitteet perustuvat pelaajien omiin laskelmiin, eikä niihin ole saatu varmistusta pelin kehittäjiltä. (Dishonored Wikia 2012b.)

Tasojen yhteenvedon lisäksi pelaaja voi aistia kaaoksen ympäristössään NPC hahmojen käytöksestä. Matalan kaaoksen aikana NPC hahmot ovat huomattavasti ystävällisempiä Corvoa kohtaan. Kaaos vaikuttaa myös Emilyn käyttäytymiseen sen jälkeen, kun Corvo on onnistunut vapauttamaan tytön. Tehtävien välissä pelaaja voi nähdä Emilyn piirtävän paperille iloisia tai synkkiä kuvia riippuen kaaoksen tasosta.

Pelin viimeisen tehtävän alku muodostuu kaaoksen mukaan, jolle voi latautua kolme eri vaihtoehtoa: matala, keskitaso tai korkea kaaos (low, medium, high). Matalan kaaoksen aikana vartijat eivät ole viimeisen pelitason alkaessa hälytystilassa. Korkean kaaoksen aikana Corvon venemies, Samuel aiheuttaa hälytyksen, jonka jälkeen alue kuhisee vartijoita. Keskitason kaaoksen aikana, venemies

ilmaisee, ettei tahdo koskaan enää nähdä Corvoa, mutta ei aiheuta hälytystä. (Dishonored Wikia 2012c.)

Läpipeluukerran kokonaiskaaos määrittää pelin loppuratkaisun. Matala kaaos tuottaa onnellisen loppuratkaisun, jossa Emily nousee valtaistuimelle muun ylimystön seuratessa. Onnellisen loppuratkaisun kertomus näyttää muiden pelissä vastaan tulleiden henkilöiden onnellisen nykytilan. Korkean kaaoksen loppuratkaisu näyttää Emilyn istuvan valtaistuimella Corvon suojeluksessa, muun ylimystön väkivaltaisesti taistellessa valtaistuimen edessä. Pelillä on myös kolmas loppuratkaisu. Mikäli viimeisen tehtävän lopussa Emily kuolee, on loppuratkaisu erittäin kaoottinen, ja kertomus näyttää Samuelin pakenevan kaupungista veneellä myrskyssä. Kaoottisin loppuratkaisu näyttää myös Corvon naamion ja miekan Emilyn hautakivellä, jättäen Corvon nykytilan arvoitukseksi.

2.3 Middle Earth: Shadow of Mordor

Monolith Productionsin kehittämä ja Warner Bros. Entertainmentin julkaisema peli ilmestyi syyskuussa 2014. Peli on pelattavissa Tietokoneella Microsoftin Windows käyttöjärjestelmällä, Playstation® 3 ja 4 sekä Xbox 360 ja Xbox One -pelikonsoleilla. Peli nojaa vahvasti J.R.R Tolkienin luomaan keskimaan maailmaan sekä tuo mukaansa tuttuja hahmoja Taru Sormusten Herrasta -kirjasarjasta. Peli on tyypiltään seikkailuroolipeli. Pelityypille ominaisia piirteitä pelissä on hahmokehitys, vahva tarina ja toimintaa vapaasti liikuttavassa maailmassa.

Pelin päähenkikö on Thalion, Gondorin kuningaskunnan viimeisen raja-aseman vartija. Peli alkaa raja-asemalle kohdistuneesta hyökkäyksestä, jossa pahan armeijan komentaja murhaa Thalionin perheen sekä Thalionin. Kirouksen seurauksena Thalion ei kuole, vaan hänen elämänsä sitoutuu yhteen mystisen aaveen kanssa, joka ei muista nimeään tai menneisyyttään. Pelin tarkoituksena on tuhota pahan armeija ja sen komentaja. Kirous estää Thalionia kuolemasta, jonka vuoksi hän palaa takaisin henkiin vihollisen onnistuttua surmaamaan hänet. Tarinan edetessä Thalionia motivoi kosto, jonka avulla hän voi murtaa kirouksen ja kuolla

lopullisesti. Aave puolestaan haluaa muistaa menneisyytensä, ja tarinan edetessä erilaiset esineet avaavat sitä hänelle pala palalta.

Maailma pitää sisällään erilaisia tapahtumia, joiden seurauksena vihollisen armeija kasvaa voimakkaammaksi. Pelaaja voi estää tapahtumien läpiviennin, jolloin hänen tulee voittaa yksi tai useampi vihollisarmeijan komentaja. Jos pelaaja onnistuu tässä, vihollisarmeija heikkenee. Jos tapahtumien kulkua ei pysäytetä, vihollisarmeijan komentajat voimistuvat. Tapahtumat ovat voimassa vain rajallisen ajan. Jos pelaaja ei tuossa ajassa pysäytä tapahtumaa, vihollisarmeija voimistuu. Kuviossa 2 on esitetty pelimaailman kartta, jonne on sijoitettu erilaisia tapahtumia ja tehtäviä.

Kuvio 2. Kuvankaappaus pelin kartasta, jossa on näkyvillä eri tehtäviä

Pelimaailman karttaan on merkattu valkoisella haasteita ja punaiset merkit ovat vihollisarmeijan tapahtumia. Merkkien sisällä oleva symboli kertoo tapahtuman tai haasteen tyypistä. Haaste tulee aina suorittaa määrättyä mekaniikkaa käyttäen. Esimerkiksi valkoinen merkki, jonka sisällä on jousen symboli, on jousiaseella suoritettava haaste.

Suurin uhka Thalionille on vihollisen sotapäälliköt, jotka ovat kapteeneidensa ja sotilaidensa suojelemia. Sotapäälliköt ja kapteenit omaavat persoonallisuuden sekä heikkoudet ja vahvuudet. Thalion voi yksittäisiä vihollisia kuulustelemalla

saada tietoa kapteeneista ja sotapäälliköistä sekä näiden välisistä suhteista. Samalla kuulustelu paljastaa komentajien vahvuudet ja heikkoudet. Kuviossa 3 on esitetty vihollisKomentajan ominaisuuksien esittely.

Kuvio 3. Vihollishahmon ominaisuudet

Pelaaja voi vihollisen ominaisuuksien selvittyä suunnitella kohtaamistaan tämän kanssa. Esimerkiksi kuvan komentaja on haavoittuvainen, jos Thalion ratsastaa pedolla. Mikäli komentaja kohtaa pedon ilman ratsastajaa, on komentajan voittaminen puolestaan vaikeampaa.

Kuviossa 4 on esitetty vihollisarmeijan suhteita, joita hyödyntäen Thalion voi heikentää sotapäällikön joukkoja ennen sotapäällikön kohtaamista.

Kuvio 4. Vihollisten väliset suhteet ja hierarkia

Punainen viiva kuvassa kertoo kahden kapteenin olevan vihamielisiä toisilleen. Punainen merkki kapteenin päällä kertoo tämän liittyvän pelimaailmassa olevaan tapahtumaan. Vihreä viiva kapteenin ja sotapäällikön välissä kertoo kapteenin olevan uskollinen kyseiselle sotapäällikölle. (Shadow of Mordor Wikia 2014.)

Jokainen yksittäinen vihollinen muistaa aiemmat kohtaamisensa Thalionin kanssa. Tätä pelikehittäjät kutsuvat Nemesis -järjestelmäksi (Nemesis system). Mikäli vihollinen onnistui pakenemaan Thalionia aiemmin tai onnistui tappamaan Thalionin, vaikuttaa se seuraavan kohtaamisen alussa näkyvään dialogiin. Jos yksittäinen vihollinen onnistuu tappamaan Thalionin pelin aikana, ylennetään hänet välittömästä Kapteeniksi ja hänelle muodostuu heikkoudet sekä vahvuudet.

Pelin adaptiivisuus on rakennettu erittäin syvään henkilöiden väliseen hierarkiaan ja vihollisarmeijan muuttumiseen tapahtumien myötä. Kaikkien pelin tapahtumien pysäyttäminen on mahdotonta, joten peli muuttuu väistämättä ja jokainen läpipeluukerta on aina erilainen.

3 PELIN SUUNNITTELU JA MÄÄRITTELY

3.1 Pelisuunnittelun perusta

Pelin perustan muodostaa pelin idea. Idea pitää sisällään mitä pelissä on tarkoitus tehdä, sen keskeiset mekaniikat sekä mahdolliset taustatiedot pelin tarinaan liittyen. Pelin idea voi olla kehittäjän itse luoma tai pelistudiolle annettu toimeksianto. Toinen tärkeä osa pelin pohjaa on kuinka pelin tarkoituksenmukaiseen voitto-tilanteeseen päästään ja onko mahdollisuuksia yksi vai useampi. Pelin kehittäjällä on päätösvalta, minkälaiset työkalut pelaajalle annetaan pelin läpäisemiseksi. Pelin suunnitteluvaiheessa alkuperäinen idea harvoin pysyy täysin samana. Yleensä idea kehittyy logiikkaan, tarinaan tai mekaniikkoihin liittyviä ongelmia ratkottaessa. Pelin määrittelyvaiheessa suunnitellaan mekaniikkojen toimintaa ja logiikkaa, jolla peliä ohjataan. Määrittelyvaiheessa siis luodaan teoria pelaajalle annettavista työkaluista pelin idean mukaisesti.

Jo pelin suunnitteluvaiheessa tulee ottaa huomioon, mille alustalle peli tullaan kehittämään. Eri alustoja ovat esimerkiksi Microsoft Windows, Xbox- ja PlayStation® -pelikonsolit sekä eri mobiilialustat. Nykypäivänä peli voi olla pelattavissa kaikilla alustoilla. Tietokoneelle kehitetty peli voi tukea Windows käyttöjärjestelmän lisäksi myös Linux ja OSX käyttöjärjestelmiä. Myös mobiililaitteilla voidaan julkaisua rajata esimerkiksi vain iOS, Windows tai Android käyttöjärjestelmään. Mahdolliset alustat voivat muuttua vielä kesken kehityksen mahdollisen julkaisijan ehdoista johtuen. Kun alustat on päätetty, tulee valita pelikehityksessä käytettävät työkalut, tärkeimpänä tietysti pelimoottori. Kehittäjä voi tehdä oman pelimoottorin tai käyttää valmista pelimoottoria kuten Unreal Engine, Unity3D tai Cry Engine. Pelin alustat ja mekaniikat vaikuttavat kehittäjien oman kokemuksen ohella siihen, mitä pelimoottoria on järkevintä käyttää.

Työ pohjana on oma idea pelistä, jonka suunnittelu on jatkunut vuodesta 2012 saakka. Peliä kehitetään tässä vaiheessa vain Windows alustalle. Pelin tyyppi ja testausjärkevyys ovat päätöksen takana. Opinnäytetyössä peliin kehitetään

adaptiivisuuden logiikka, jonka testaamiseen ei vaadita kuin yksi käyttöjärjestelmä. Muiden alustojen tai järjestelmien käyttöönotto tässä vaiheessa aiheuttaisi vain ylimääräistä lisätyötä kehityksessä. Pelimoottoriksi valikoitui Unity3D sen tuttuuden ja valmiiden ominaisuuksien vuoksi. Unreal Engine olisi ollut hyvä vaihtoehto, mutta sen työkalujen opettelu olisi vienyt ylimääräistä aikaa itse kehitystyöltä.

Pelimoottoria voi mahdolliseen jatkokehitykseen lähdettäessä vaihtaa, jos tarve vaihdolle ilmenee. Unity3D editorilla prototyypin tekeminen ja testaus vain on helpompaa kuin Unreal Enginellä. Myös Linux alusta voisi olla järkevää tuoda mukaan myöhemmässä vaiheessa uuden Steam Machine konseptin vuoksi.

3.2 Pelin idea

Työssä toteutettavan pelin perusajatuksena on voittaa määrällisesti suurempi vihollinen, joka mukautuu alueeseen ja alueella tapahtuviin toiminteisiin. Pelin miljöö on kaupunki, jonka hallintoa ja vartiostoa vastaan pelaaja taistelee. Pelimaailma on jaettu kuuteen osaan, jotka toimivat itsenäisesti. Peli on tyypiltään vuoropohjainen strategiaroolipeli. Pelaaja ja tietokone suorittavat toiminteensa omilla vuoroillaan niin, että pelaaja aloittaa.

Pelaaja voi vuoronsa aikana liikkua, sekä suorittaa erilaisia toimintoja. Pelaajan mahdollisuuksia suorittaa toimintoja vuoronsa aikana mitataan energialla. Jokainen toiminne vie pelaajalta ennalta määrätyn määrän energiaa. Mahdollisia toimintoja ovat esimerkiksi varastaminen, puhuminen sekä muut taisteluun liittyvät toiminteet. Kun energia loppuu, ei pelaaja voi enää suorittaa toimintoja ja vuoro siirtyy tietokoneelle.

Taistelu pelissä toimii myös vuoropohjaisena. Pelaaja voi omalla vuorollaan hyökätä vihollista vastaan tai vaihtoehtoisesti jäädä odottamaan vihollisen lähestymistä valmiustilassa, jolloin pelaaja voi vahingoittaa vihollista vaikka vuoro ei olisikaan pelaajalla. Vihollinen voi omalla tai pelaajan vuorolla havaita tapahtumia

pelimaailmassa ja reagoida niihin, huolimatta siitä onko vuoro pelaajalla vai tietokoneella. Näin ollen jos vihollinen havaitsee pelaajan, pelaajan omalla vuorolla, voi taistelu alkaa jo havainnosta. Tällöin tietokone suorittaa rajallisia liikkeitä pelaajan vuoron aikana, joihin pelaaja voi reagoida jäljellä olevan energiansa mukaan.

Pelin adaptiivisuutta ohjaa pelimaailman levottomuus. Näin adaptiivisuus on vahvasti sidottuna maailmaan ja sen tapahtumiin. Maailman levottomuuteen vaikuttava tekijät voivat olla pelaajan, vihollisen tai tavallisten NPC-hahmojen suorittamia. NPC-hahmojen suorittamat tapahtumat ilmenevät sattumanvaraisesti. Vihollisen tekemät toiminteet ovat seurausta joko pelaajan tai NPC-hahmojen toiminteista. Jokainen alue käsittelee omaa levottomuuttaan. Koko pelimaailman levottomuus muuttuu yksittäisten alueiden levottomuuden muuttuessa.

Pelaaja saa pelin edetessä lisää mahdollisia toimintoja kun hahmon tunnettavuus maailmassa kasvaa. Samalla tunnettavuus lisää pelaajan riskiä tulla vihollisen havaitsemaksi. Pelaaja toimintoja NPC-hahmojen näkökulmasta rajoitetaan, jos he eivät luota pelaajaan tarpeeksi. Pelaajalle siis määritetään luottamus, jota hallinnoidaan myös alueittain.

Pelaajalle on tarkoitus antaa mahdollisimman laaja valikoima työkaluja, joilla hän voi pelissä edetä. Näin ollen pelaaja voi päästä pelin voittotavoitteeseen useammalla eri tavalla. Esimerkiksi pelaaja voi pyrkiä voittotavoitteeseen toimimalla agitaattorina likaamatta omia käsiään. Pelin haaste muodostuu annettujen työkalujen käytön tasapainottamisesta. Mikäli pelaaja pyrkii voittotavoitteeseen vain yhdellä menetelmällä, voi peli muuttua jopa mahdottomaksi läpäistä.

3.3 Pelimaailman mittarit

Pelimaailma mittaa adaptiivisuutta levottomuudella. Jokainen alue mittaa omaa levottomuuttaan yksittäin, mutta kasvattaa koko maailman levottomuutta sekä muiden alueiden levottomuutta. Mittarit on nimetty seuraavasti: disorder = koko pelimaailman levottomuus, disorderArea0 = alueen 0 levottomuus, disorderArea1

= alueen 1 levottomuus ja niin edelleen. Jokainen mittari on jaettu pykäliin, joilla ohjaillaan maailman toimintaa. Taulukossa 1. On esitetty levottomuuden vaikutus alueiden käyttäytymiseen.

Taulukko 1. Alueiden levottomuuden vaikutus käyttäytymiseen

Levottomuus	Vaikutus
0 – 24	Ei vaikutusta toimintaa
25 – 49	Alueella vihollisten toiminta tehostuu
50 – 74	NPC hahmot käyttäytyvät varautuneemmin
75 - 100	Vihollishahmot ovat erittäin aggressiivisia. NPC hahmot toimivat arvaamattomasti
yli 100	Vihollistoiminta maksimaalista. NPC hahmot erittäin aggressiivisia

Arvoasteikko 0 – 200, sekä raja-arvot 25. yksikön välein on suunnitteluvaiheessa määritetty vain helposti havainnollistavista luvuista. Asteikkoa ja raja-arvoja voi myöhemmin muuttaa tarpeen niin vaatiessa. Mikäli levottomuus nousee yli sadan, ei toiminta enää muutu. Alueen levottomuuden muutos muuttaa viereisten alueiden ja maailman levottomuutta taulukossa 2 esitettävällä tavalla.

Taulukko 2. Levottomuuden muutoksen vaikutus

Alue	Vaikutus
Alue 0	Alue 1 + muutos/2, Alue2 + muutos/2 + Maailma + muutos/3
Alue 1	Alue 0 + muutos/2, Alue 2 + muutos/2 + Maailma + muutos/3
Alue 2	Alue 1 + muutos/2, Alue 3 + muutos/2 + Maailma + muutos/3
Alue 3	Alue 2 + muutos/2, Alue 4 + muutos/2 + Maailma + muutos/3
Alue 4	Alue 3 + muutos/2, Alue 5 + muutos/2 + Maailma + muutos/3
Alue 5	Alue 4 + muutos/2, Alue 3 + muutos/2 + Maailma + muutos/3

Mikäli alueen 0, levottomuus kasvaa kymmenellä, alueen 1 ja 2 levottomuus kasvaa viidellä ja maailman levottomuus 3,33. Koko pelimaailman levottomuus vaikuttaa pelimaailman muutoksen suuruuteen. Taulukossa 3 esitetään maailman levottomuuden vaikutus levottomuuden muutokseen.

Taulukko 3. Maailman levottomuuden vaikutus levottomuuden muutokseen

Levottomuus	Vaikutus
0 – 24	Ei vaikutusta
25 – 49	Levottomuuden muutos 1,5x
50 – 74	Levottomuuden muutos 2x
Disorder > 75	Levottomuuden muutos 3x

Taulukon 3 mukaan alueiden levottomuuden muutoksen kerroin kasvaa maailman levottomuuden mukaan. Jos maailman levottomuus on 25 ja levottomuuteen määrätyllä alueella tulee muutos, on kokonaismuutos x * 1,5.

Levottomuus muuttuu toimintojen myötä joko heti tai viiveellä, riippuen toiminteesta. Esimerkiksi jos pelaaja varastaa NPC-hahmolta ja varkaus onnistuu, levottomuus muuttuu vasta pelaajan seuraavan vuoron päätyttyä. Jos pelaaja jää kiinni varkaudesta, ei kyseinen toiminto tällöin nosta levottomuutta alueella.

Alueiden levottomuutta ohjataan niiltä kerättävällä statistiikalla. Esimerkiksi tehdyt rikokset, vaikuttavat alueen levottomuuteen. Mikäli vartijat ratkaisevat rikoksia, levottomuus pienenee. Mikäli pelaaja pyrkii tarkoituksen mukaisesti kiihdyttämään kansaa, voi levottomuus muuttua alueella nopeasti.

3.4 Pelaajan mittarit

Ensimmäinen pelaajan mittari on pelaajan tunnettavuus. Mitä suurempi tunnettavuus pelaajalla on alueella, sitä helpommin vartijat havaitsevat pelaajan väkijoukosta. Pelaajan tunnettavuutta seurataan Notoriety-mittarilla. Samoin kuin levottomuutta, pelaajan tunnettavuutta seurataan koko pelimaailman tasolla sekä jokaisella alueella erikseen. Tunnettavuus vaikuttaa vihollisten havainnointiin pelaajan suhteen silloin kun pelaaja astuu vartijan näkökenttään. Tunnistaminen tehdään satunnaisgeneraattorilla, joka arpoo luvun väliltä 1 – 100. Jos arvottu luku on pienempi kuin pelaajan tunnettavuus vihollinen tunnistaa pelaajan. Jos luku on suurempi, vihollinen ei tunnista pelaajaa. Esimerkiksi pelaajan tunnettavuus alueella on 48 ja arvottu luku 43 vihollinen tunnistaa pelaajan.

Pelaajan tunnettavuus alkaa arvosta 0 ja lisääntyy pelaajan suorittamista toiminteista. Pelin alussa vihollinen ei siis tunnista pelaajaa lainkaan. Tunnettavuus maailman suhteen lisää pelaajan tunnettavuutta alueilla. Alueiden tunnettavuus lisää pelaajan tunnettavuutta muilla alueilla ja koko pelimaailmassa. Koko pelimaailman tunnettavuus on minimiarvo, joka välittyy kaikille alueille. Taulukossa 4 on selitetty alueellisen tunnettavuuden vaikutus koko maailman tunnettavuuteen.

Taulukko 4. Pelaajan tunnettavuuden vaikutus maailmaan

Raja-arvo	Vaikutus pelimaailman tunnettavuuteen
25	1 – 25, generoidaan satunnaisuudella
50	25 – 50, generoidaan satunnaisuudella
75	50 – 75, generoidaan satunnaisuudella

Pelaajan alueellisen tunnettavuuden vaikutus kokomaailman tunnettavuuteen kasvaa jäljessä alueelliseen tunnettavuuteen suhteutettuna. Kun yhden pelialueen tunnettavuus ylittää raja-arvon uusi arvo generoidaan satunnaisesti taulukon 4 esittämillä vaihteluväleillä. Mikäli pelaajan tunnettavuus yhdellä pelialueella nousee arvoon 50 ja koko maailman tunnettavuus on alle 25, arvotaan koko maailman tunnettavuudelle uusi arvo väliltä 25 – 50. Tämä arvo välitetään kullekin alueelle uudeksi tunnettavuuden arvoksi. Mikäli pelaajan tunnettavuus koko maailmassa on jo suurempi kuin uusi generoitu arvo, ei arvo voi muuttua pienemmäksi tässä vaiheessa. Esimerkiksi tunnettavuus maailmassa on toisen alueen johdosta noussut jo arvoon 50 ja satunnaisuus arpoo luvuksi 25, pysyy arvo 50 edelleen voimassa.

Pelaaja voi pienentää tunnettavuuttaan alueella ja koko pelimaailman tasolla suorittamalla erilaisia toimintoja. Mikäli tunnettavuus laskee yhdellä pelialueella alempaan raja-arvoon, arvotaan pelaajalle uusi tunnettavuus koko maailmalle. Generointi tapahtuu tässä vaiheessa koko maailman tunnettavuuden ja pienimmän raja-arvon väliltä. Esimerkiksi pelaajan tunnettavuus laskee alueella 1 raja-arvoon 25 ja alueen kaksi tunnettavuus on korkein, 75. Pelaajan tunnettavuus

koko maailmassa on arvossa 50. Generaattori arpoo tällöin luvun väliltä 25 – 50. Tunnettavuus ei siis voi tässä vaiheessa nousta.

Toinen pelaajan mittareista on luottamus. Luottamus toimii samalla tavalla kuin tunnettavuus tai levottomuus. Luottamus voi olla arvona myös negatiivinen. Mikäli luottamus on negatiivinen, NPC-hahmot eivät halua tehdä yhteistyötä pelaajan kanssa. Jos luottamus puolestaan on positiivinen, NPC-hahmot suostuvat auttamaan pelaajaa ja pelaaja voi suorittaa valuuttaa vaativia asioita myös velaksi. Luottamus toimii myös portaittain ja alueellinen luottamus vaikuttaa muihin alueisiin. Luottamus kasvaa, jos pelaaja auttaa NPC-hahmoja ja pienenee jos pelaaja tekee rikoksen josta jää kiinni.

3.5 Adaptiivisuuden esittäminen

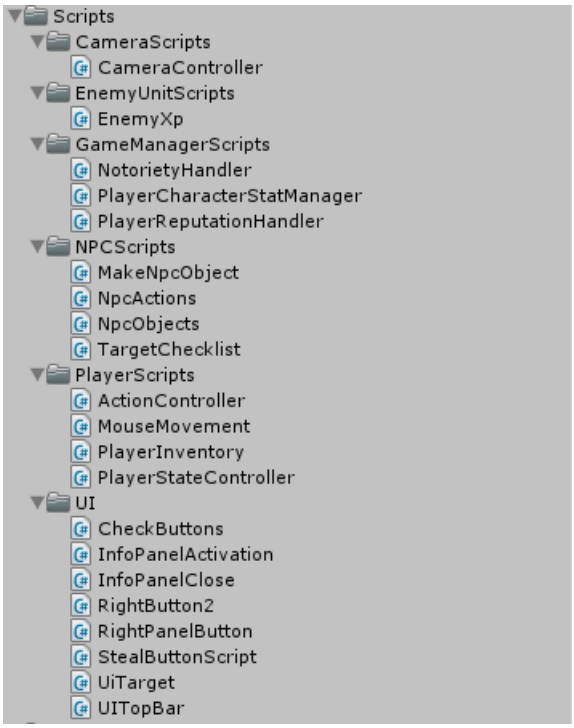
Adaptiivisuus esitetään pelaajalle pelin UI-elementtien kautta. Pelaaja pystyy tarkastelemaan kunkin alueen sekä koko maailman tilastoja. Pelaaja pystyy suunnittelemaan ja tasapainottamaan toimintaansa saamansa tilastopalautteen kautta. Pelaajalle myös esitetään yksittäisten toimintojen vaikutus mittareihin joko plus (+) tai miinus (-) merkkisenä, kuitenkaan antamatta tarkkaa vaikutuksen määrää ennen suoritusta. Tämä jättää pelaajalle mahdollisuuden päättää riskistä, tietämättä lopputuloksen koko vaikutusta.

Pelin UI-elementteihin toteutetaan tilastomuotoinen osuus, josta pelaaja voi käydä katsomassa nykytilan statistiikkaa. UI-elementti on lähtökohtaisesti poissa näkyvistä ja pelaaja voi avata sen pelin aikana. Samaan UI-elementtiin tullaan toteuttamaan myöhemmin myös muuta statistiikkaa pelaajan tarkasteltavaksi.

4 PELIN TOTEUTUS

4.1 Toteutuksen valmistelu

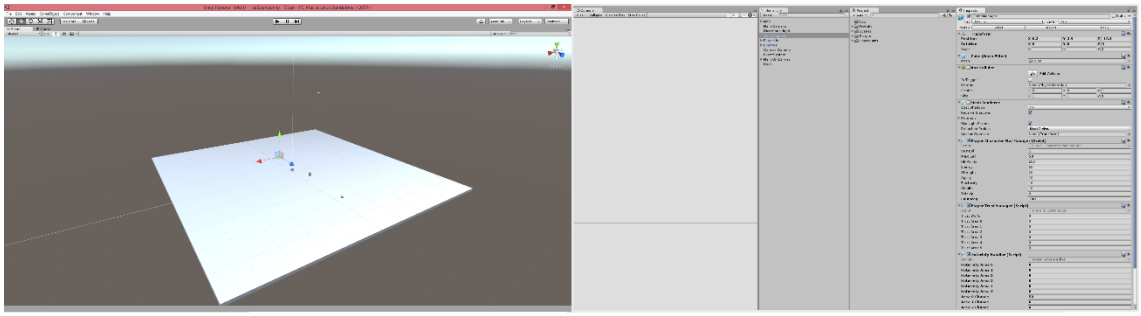
Adaptiivisuuden varsinainen toteutus tehdään edellä esitetyn idean ja määrittelyn pohjalta. Toteutuksen aikana idea kehittyy ja määrittelyä tullaan korjaamaan tarvittaessa. Toteutusta jäsennellään Unity3D:n editorissa niin, että toimintoihin vaadittavat Scriptit ovat projekti hierarkiassa omissa kansioissaan. Kansiot jaotellaan käyttötarkoituksen mukaisesti. Tiedostojen jäsentely editorissa on esitetty kuviossa 5.



Kuvio 5. Scriptien jäsentely Editorissa

Scriptaus tapahtuu C# -ohjelmointikieltä käyttäen Unity3D:n mukana tulevaa MonoDevelop -ohjelmaa. Pelimoottori tukee C#, Javascript ja Boo –scriptauskieliä. C# valikoitui käytetyksi kieleksi, sillä se on tutuin edellä mainituista. Pelin tasot rakennetaan Unity3D Editorissa. Tason tekeminen vaatii scenen johon sijoitetaan peliobjekteja (Game Object). Peliobjekteihin kiinnitetään kaikki komponentit kuten scriptit, fysiikkakomponentit tai äänitiedostot.

Unity3D:n Editorissa tapahtuu itse pelin rakentaminen. Editori pitää sisällään Scene, Game, Console, Hierarchy, Project ja Inspector -näkymät. Lisäksi editoriin saa halutessaan muitakin näkymiä esille. Kuviossa 6 on esitetty Editorin rakennetta.



Kuvio 6. Unity3D Editorin rakenne

Scene-näkymässä rakennetaan itse peliä. Scene-näkymään luodaan objekteista kokonaisia tasoja peliin. Game-näkymässä peliä voidaan suorittaa poistumatta editorista ja kääntämättä pelin kaikkia tiedostoja. Console-näkymään peli tulostaa virheviestit ja erilaiset log-viestit, joita kehittäjä haluaa nähdä. Hierarchy-näkymä kertoo mitä peliobjekteja avoinna olevaan sceneen on sijoitettu. Project-välilehti näyttää koko projektin kaikki tiedostot. Inspector-välilehdessä näytetään valitun peliobjektin ominaisuudet. Inspector-välilehdessä valitun objektin ominaisuuksia ja komponentteja voi muokata. Lisäksi objectiin kiinnitettyjen scriptien julkisia muuttuja voi hienosäätää inspector-näkymästä. (Unity3D Documentation 2015a.)

Jokaista pelin tasoa kohti luodaan oma scene Editorissa ja tallennetaan se projektikansioon Editorin kautta. Tallennetut scenet näkyvät Editorin project-välilehdellä. Jokaisen tason peliobjektit sijoitetaan tasoa vastaavaan sceneen.

Projektille tehdään myös Git-versionhallinta Bitbucket -verkkopalveluun. Versionhallintaa ylläpidetään SourceTree -ohjelmalla. Versionhallinnalla mahdollistetaan palaaminen projektissa aiempaan toimivaan versioon, mikäli uudet toteutetut ominaisuudet rikkovat aiemman toiminnan.

Scriptit tullaan kommentoimaan siten, että funktioiden toiminta selviää koodin lukijalle ilman syvällisempää perehtymistä koodiin. Myös erilaiset isommat toiminteet tullaan selittämään kommentein. Jokainen scripti pitää lisäksi sisällään versioinnin, eli tiedon siitä kuka scriptin on tehnyt, editoinut sitä ja milloin tämä on tapahtunut.

4.2 Perusominaisuuksien toteuttaminen

Logiikan toteutusta varten pelaaja tarvitsee ominaisuuksia, joilla toimintaa voidaan ohjata. Ensin luodaan peliobjekti sceneen, jolle annetaan nimi PlayerUnit. PlayerUnit -objekti on pelaajan hallittavissa. Pelaajan objekti tarvitsee ensimmäiseksi mahdollisuuden ottaa muita objekteja pelimaailmasta kohteekseen. Tätä varten luotiin ActionController niminen C# -scripti joka kiinnitetään PlayerUnit -peliobjektiin. Scripti pitää sisällään julkisen GameObject tyyppisen muuttujan nimeltään playerTarget. Tämä mahdollistaa pelaajan kohteen tallentamisen kyseiseen muuttujaan, sen tarkastelun editorista sekä kohteen tietojen välittämisen muille objekteille ja komponenteille. ActionController pitää myös sisällään toiminnan pelaajan kohteen valinnalle. Scripti on esitetty kuviossa 7.

```
//Player targetin by targetable or enemy tag
if (Input.GetMouseButtonUp (0))
{
    RaycastHit hit;
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

    //Select target for action
    if(Physics.Raycast(ray, out hit))
    {

        //Tag information to string
        string tempTag = hit.transform.gameObject.tag;
        CameraController cc = GameObject.Find ("Main Camera").GetComponent<CameraController> ();

        //Add player target
        if(tempTag == "Targetable" || tempTag == "Enemy")
        {
            playerTarget = hit.transform.gameObject;

            //Send target info to Ui
            UiTarget UIT = GameObject.Find("TargetText").GetComponent<UiTarget>();
            UIT.setUiTarget(playerTarget);
        }

        //Only add camera target
        else if(tempTag == "Player")
        {
            cc.SetCamTarget(hit.transform.gameObject);
            playerTarget = null;
        }
    }
}
```

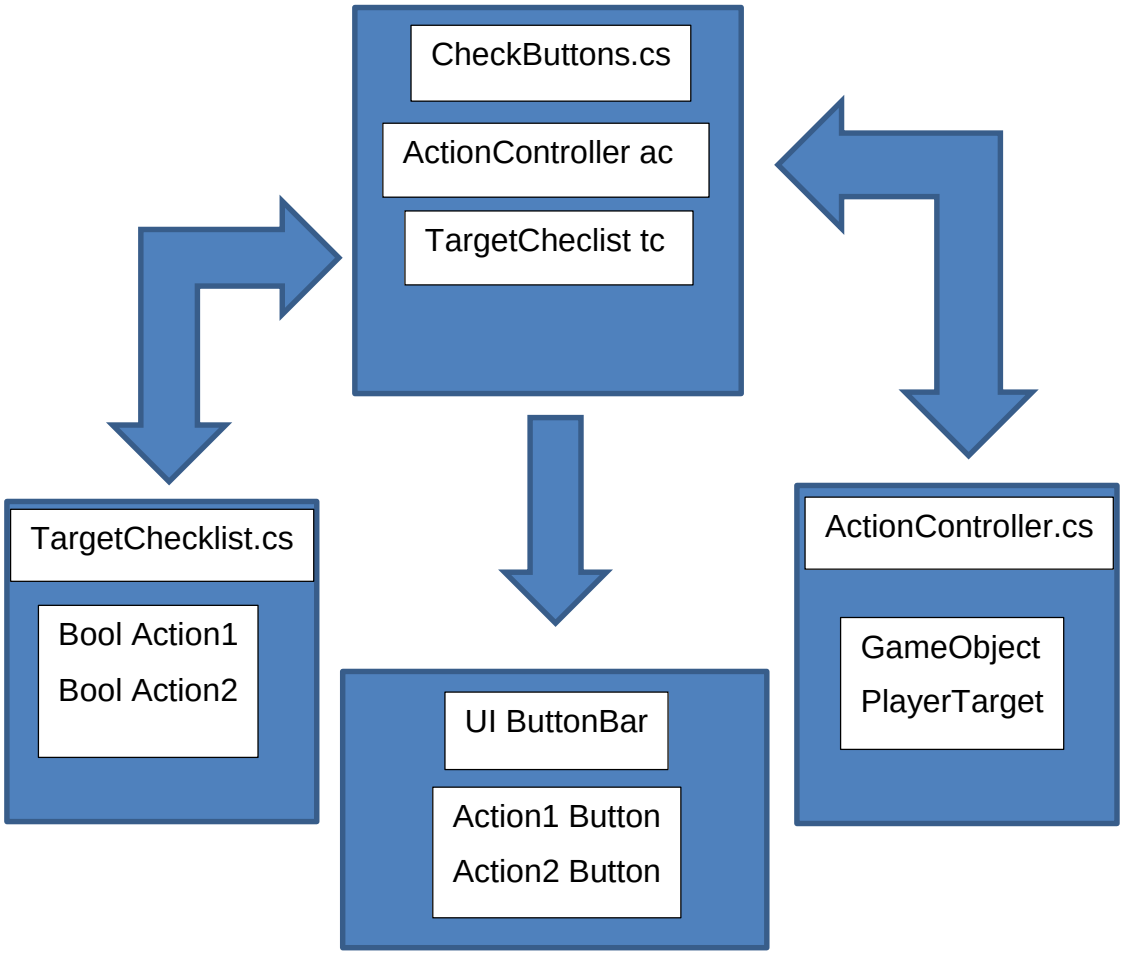
Kuvio 7. ActionController scriptin koodi kohteen valinnalle.

Pelaajan painaessa hiiren vasemman puoleista painiketta, pelin kameraobjekti lähettää kursorin kohdalle säteen. Säde palauttaa osumapisteen objektin tiedot. Mikäli säteen palauttaman objektin tiedoista tag on joko enemy tai targetable, objekti asetetaan playerTarget -muuttujaan.

NPC-objekteille on luotu TargetChecklist -scripti, joka pitää sisällään julkisen boolean tyyppisen muuttujan kaikista mahdollisista pelaajan toiminnoista. Scripti kiinnitetään jokaiseen peliobjektiin, jolle pelaaja voi suorittaa toimintoja. Näin jokaiselle objektille voidaan muuttaa mahdollisia toimintoja suoraan editorista boolean-muuttujan oletusarvoa vaihtamalla. Lisäksi NPC-objekteille lisätään jokaista toiminnetta varten erillinen scripti, joka pitää sisällään TargetChecklist -muuttujan. Kyseinen muuttuja välittää TargetChecklist -scriptin tiedot. Muuttujan kautta toiminnan suorittava scripti tarkastaa onko toiminnan suorittaminen kyseisellä objektilla lainkaan mahdollista. Lisäksi TargetChecklist -scripti omistaa tiedon siitä, mihin pelialueeseen kyseinen objekti kuuluu.

Toiminta suoritetaan UI-elementtiin kiinnitetyn painikkeen kautta. Mikäli toiminnan suorittaminen kyseiselle hahmolle ei ole mahdollista, painiketta ei pelaajalle näytetä. UI-elementtiin on kiinnitetty CheckButtons -scripti. Scriptin toiminta on selitetty kuviossa 8.

CheckButtons -scripti omistaa ActionController sekä TargetChecklist -muuttujat, joiden avulla scriptien julkiset arvot saadaan välitettyä. ActionControllerista saadaan välitettyä pelaajan tämän hetkinen kohde. TargetChecklist omistaa kyseisen kohteen mahdolliset toiminnot. CheckButtons -scriptin Update-funktio tarkastaa boolean-arvon tilaa jatkuvasti. Tämän perustella CheckButtons -scripti asettaa painikkeet näkyville UI-elementtiin.



Kuvio 8. CheckButtons -scriptin toimintalogiikka

Pelaan tunnistaminen vihollisen näkökulmasta toteutetaan luomalla vihollisobjektiin kiinnitetty peliobjekti DetectionBox. Kyseessä on erillinen peliobjekti, joka sijaitsee varsinaisen peliobjektin etupuolella. Tämä objekti toimii vihollisen näkökenttänä. Peliobjektilta poistetaan graafiset ominaisuudet, sillä objektin näkymisellä ei tässä vaiheessa ole merkitystä. Fysiikka ominaisuuksista objektille jätetään Collider, joka asetetaan IsTrigger -tilaan. Peliobjekti sisältää VisionDetection -scriptin, joka tunnistaa Colliderin perusteella peliobjektin, joka astuu DetectionBox -peliobjektin sisään. Tunnistamisessa käytetään OnTriggerEnter -funktiota (Unity3D documentation 2015b.)

Lisäksi pelimaailmaan luodaan peliobjekti GameManager, joka pitää sisällään scriptit pelaajan luottamuksen, tunnettavuuden ja sekä alueiden ja maailman le-

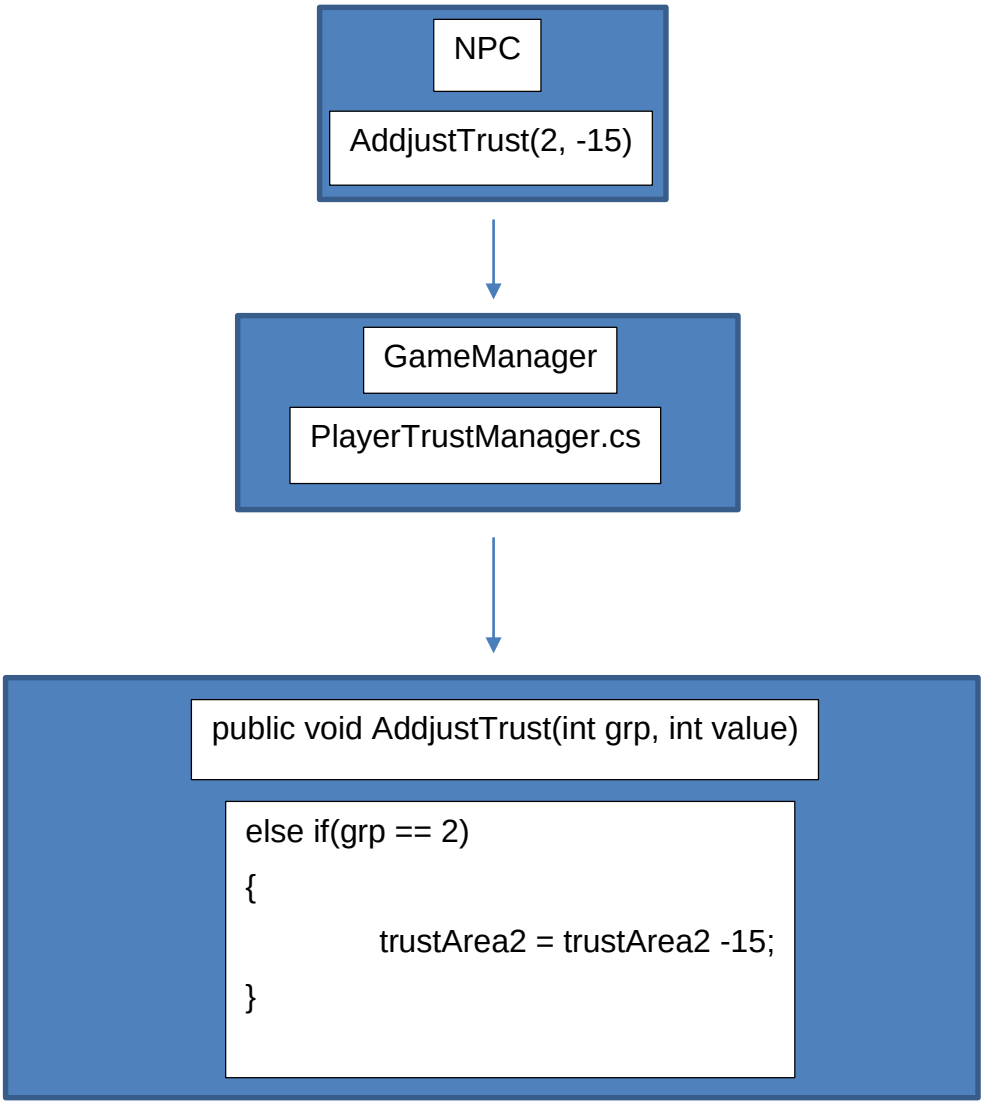
vottomuuden säätelyyn. Peliobjektilta poistetaan kaikki grafiikka ja fysiikka -ominaisuudet, koska objektin ei ole tarkoitus näkyä pelimaailmassa. GameManager -peliobjekti siis pitää sisällään kaiken adaptiivisuuteen liittyvän toiminnan. Peliobjektin on säilytettävä tietonsa myös scenen vaihtuessa. Peliobjekti siis haluttaan siirtää toiseen sceneen, jos pelaaja haluaa vaihtaa aluetta pelissä. Tämä saadaan toteutettua DontDestroyOnLoad -funktiolla. Peliobjektiin kiinnitetään PlayerTrustManager, NotorietyHandler sekä DisorderManager -scriptit. (Unity3D Documentation 2015c.)

4.3 Pelaajan mittareiden hallinta

GameManager -peliobjektiin kiinnitetyt scriptit hoitavat tiedon ylläpidon pelimaailman ja pelaajan nykyisestä tilasta. Scripteille on myös toteutettu omat funktionsa tiedon muokkaamiseen tarpeen tullen.

PlayerTrustManager -scriptin muuttujat koostuvat julkisista integer tyyppisistä muuttujista pelimaailmaa sekä jokaista aluetta kohden. Lisäksi raja-arvot luottamukselle luodaan kahdella muuttujalla joiden arvot ovat -100 ja 100. Luottamuksen muokkaus tapahtuu AddjustTrust -funktiolla. Funktio ottaa vastaan kaksi integer tyyppistä muuttujaa, tiedon alueesta sekä muutoksen suuruudesta. Kuviossa 9 on esitetty luottamuksen hallinnan logiikkaa.

Käsky pelaajan luottamuksen muutokseen tulee NPC-hahmoon kiinnitetystä pelaajan toiminallisuutta hoitavasta scriptistä. Komento pitää sisällään NPC-hahmon ryhmän PlayerTrustManager -skriptistä, joka välitetään grp-muuttujana funktiolle. Funktio saa myös komennon antavalta scriptiltä value-muuttujan, joka pitää sisällään muutoksen suuruuden. AddjustTrust -funktio tarkastaa grp-muuttujan perusteella minkä ryhmän luottamusta säädetään.



Kuvio 9. Pelaajan luottamuksen säätely

Funktio tarkastaa myös raja-arvojen ylityksen ja tarvittaessa kutsuu AddjustWorldTrust -funktiota. Kuviossa 10 on esitetty ryhmän 2 luottamusta säätelevä toiminta kokonaisuudessaan.

```
//GROUP 2 ADDJUSTMENT
else if (grp == 2)
{
    int tempTrustValue = trustArea2;
    trustArea2 = trustArea2 + value;

    if (tempTrustValue < 50 && trustArea2 >= 50)
    {
        AddjustWorldTrust (1);
    }

    else if (tempTrustValue < 75 && trustArea2 >= 75)
    {
        AddjustWorldTrust (2);
    }

    else if (trustArea2 == 100 && tempTrustValue < 100)
    {
        AddjustWorldTrust (3);
    }

    else if(tempTrustValue > 0 && trustArea2 < 0)
    {
        AddjustWorldTrust (-1);
    }

    else if(tempTrustValue > -25 && trustArea2 < -25)
    {
        AddjustWorldTrust (-2);
    }
}
//GROUP 2 ADDJUSTMENT
```

Kuvio 10. Ryhmän 2 luottamuksen säätäminen

Luottamuksen säätely aloitetaan tallentamalla vanha luottamuksen arvo väliaikaiseen muuttujaan, jonka jälkeen muutetaan luottamuksen arvo. Tämän jälkeen funktio tarkastaa oliko vanha luottamuksen arvo eri puolella jotakin raja-arvoa kuin uusi arvo. Mikäli tarkastuksen perusteella raja-arvo on ylittynyt, kutsutaan AddjustWorldTrust -funktiota ja välitetään funktiolle integer-muuttujan arvo. AddjustWorldTrust -funktio tarkastaa välitetyn muuttujan perusteella, mihin arvoon koko maailman luottamus asetetaan. Vastaavat tarkastukset tehdään jokaisen ryhmän kohdalla.

Pelaajan tunnettavuutta pelimaailmassa säädellään samalla logiikalla kuin luottamusta. Tunnettavuus vaikuttaa vihollisen mahdollisuuteen tunnistaa pelaaja.

Pelaajan astuessa vihollisen DetectionBox -peliobjektin sisälle, suoritetaan VisionDetection -scriptin OnTriggerEnter -funktio. Kuviossa 11 esitetään pelaajan tunnistaminen.

```
//Something enters detection box
void OnTriggerEnter(Collider collider)
{
    if (collider.name == "PlayerVisual")
    {
        //Get notoriety handler and target checklist information
        NotorietyHandler nh = GameObject.Find ("GameManager").GetComponent<NotorietyHandler> ();
        TargetChecklist tc = parent.GetComponent<TargetChecklist>();

        //Generate alert chance
        float alertChance = Random.Range (0f, 100f);

        //Find enemy AI
        BasicAi ba = parent.GetComponent<BasicAi> ();

        //Check if detected
        if (alertChance < nh.notorietyArea0)
        {
            ba.setTarget (GameObject.Find ("PlayerUnit"));
            Debug.Log ("Player Detected");
        }

    }
}
```

Kuvio 11. Pelaajan tunnistaminen

Minkä tahansa objektin astuessa vihollisen näkökenttään, tarkastaa VisionDetection -scripti objektin nimen. Mikäli objekti on pelaaja, haetaan tieto pelaajan tunnettavuudesta sekä vihollisobjektin ryhmä, jonka suhteen tunnettavuutta halutaan tarkastella. Tämän jälkeen arvotaan luku satunnaisesti väliltä 0 – 100 ja verrataan arvottua lukua pelaajan tunnettavuuden arvoon. Jos pelaaja tunnistetaan, asetetaan pelaajan objekti vihollisen AI-scriptissä olevan julkisen GameObject -muuttujan arvoksi.

4.4 Levottomuuden hallinta

Pelimaailman levottomuutta hallitaan DisorderManager -scriptillä. Scripti pitää sisällään float tyyppisen muuttujan jokaista aluetta kohden sekä float-muuttujan vielä koko pelimaailman levottomuuden hallintaan. Lisäksi scripti sisältää muuttujan levottomuuden kertoimelle. Jokaista aluetta kohden on oma funktio jolla säädellään levottomuutta. Luottamukseen ja tunnettavuuteen nähden toteutus on

erilainen, koska levottomuuden muutos vaikuttaa erilailla muiden alueiden levottomuuteen kuin muut mittarit. Funktio kutsutaan vastaavalla tavalla, kuin muidenkin mittareiden funktiot, pelaajan toimintaa suorittavasta scriptistä. Toimintaa suorittava scripti tarkastaa kyseessä olevan alueen, jonka mukaan oikea funktio kutsutaan. Kuviossa 12 on esitetty DisorderManager -scriptin toimintaa.

```
public void AddjustArea5Disorder(float value)
{
    //Addjust area5 disorder
    area5Disorder = area5Disorder + value * disorderMultiplier;
    //Addjust nearby areas disorder
    area3Disorder = area3Disorder + ((value / 2f) * disorderMultiplier);
    area4Disorder = area4Disorder + ((value / 2f) * disorderMultiplier);
    //Addjust world disover
    AddjustWorldDisorder (value / 3f);
}

public void AddjustWorldDisorder(float value)
{
    totalDisorder = totalDisorder + value;

    //Check if multiplier needs to change
    if (totalDisorder < 25f)
    {
        disorderMultiplier = 1;
    }

    else if(totalDisorder >= 25 && totalDisorder < 50)
    {
        disorderMultiplier = 1.5f;
    }

    else if(totalDisorder >= 50 && totalDisorder < 75)
    {
        disorderMultiplier = 2f;
    }

    else if(totalDisorder >= 75)
    {
        disorderMultiplier = 3f;
    }
}
```

Kuvio 12. Levottomuuden hallinta

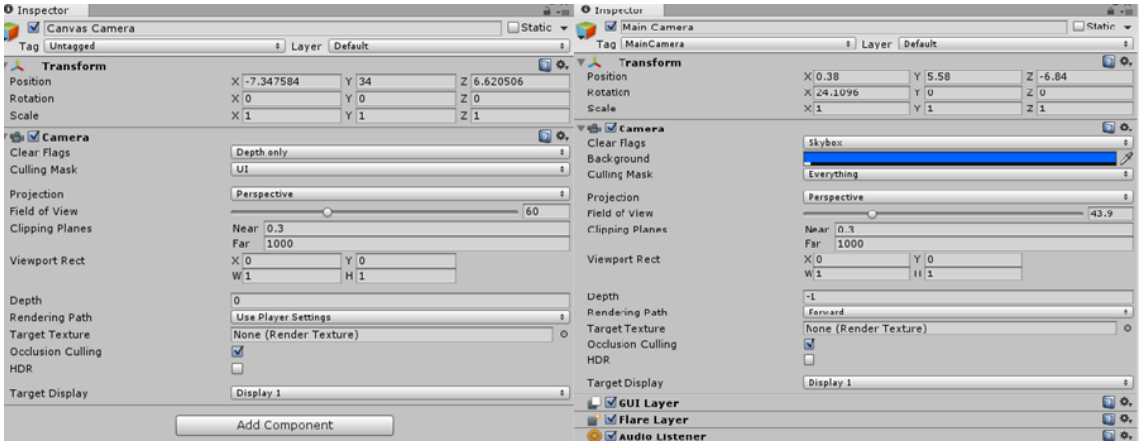
Toisin kuin pelaajan mittareita säädeltäessä, maailman levottomuutta muutetaan jokaisen levottomuuden muutoksen yhteydessä. Maailman levottomuuden muuttuessa, tarkastaan tarvitseeko muutoksen kerrointa muuttaa.

Levottomuuden hallinta itsessään on yksinkertainen toteuttaa. Levottomuuden vaikutus pelimaailmaan on monimutkaisempi rakentaa. Logiikka voidaan tehdä jokaiselle pelimaailman hahmolle erilaiseksi. Vaikutus voidaan myös rakentaa satunnaisuuden varaan, ennalta määrättyjä ehtoja noudattaen. Esimerkiksi NPC-hahmolle puhuttaessa voi vastaus olla tylympi, mikäli maailman levottomuus on korkea. Esimerkkitapauksia esitetään kappaleessa viisi.

4.5 UI-elementtien toteutus

UI-elementit peliin rakennetaan Unity3D:n canvas-systeemillä. Suurin osa UI-elementeistä saadaan toteutettua ja aseteltua paikoilleen suoraan Editorissa. Näytettäviä UI-elementtejä hallitaan canvasin sisälle toteutettujen painikkeiden avulla, jotka pitävät sisällään scriptin painikkeen toimintaa varten. Vaihtoehtoinen tapa toteutukselle olisi ollut vanha GUI-luokka, mutta uusi toteutustapa on lähes kokonaan korvannut GUI-luokan tarpeellisuuden. (Unity 3D Documentation 2015d. Unity 3D Documentation 2015e.)

Toteutusta varten luodaan canvas-peliobjekti Editorilla. Canvasin graafinen tulkinta säädetään Screen Space – Camera -asetukseen, jolloin canvas asettuu koko kameran näkökentän kokoiseksi. Koska UI-elementit ovat yksittäinen ja täysin erillinen osa graafisten elementtien tulkinnassa, luodaan pelimaailmaan niiden esittämistä varten täysin oma kamera. Canvasin näyttämistä varten luodulta kameraobjektilta poistetaan Audio Listener, Flare Layer ja GUI Layer -komponentit. Tällä saadaan varmistettua, että kamera ei yritä tulkita pelimaailmassa tapahtuvia partikkeliefektejä. Kameran ominaisuuksista Clear Flags asetetaan Depth only tilaan, sekä Culling Mask ominaisuuksiin jätetään vain UI-asetus voimaan. Näin varmistetaan että liikkuvat objektit eivät tule peittämään UI-elementtejä sekä poistetaan kahden kameran aiheuttamat ristiriidat grafiikan tulkinnasta. Kuviossa 13 esitetään kahden eri kameroiden asetukset Editorissa.



Kuvio 13. Kameroiden asetukset

UI-elementtien toteutuksen loppuosa koostuu UI-objektien sijoittelusta canvasin sisään. Haluttu objekti jätetään aktiiviseksi ja muut inaktiivisiksi. Canvasiin sijoitetut painikkeet vaihtavat määrättyjen objektien tilaa näiden kahden asetuksen välillä.

Vanhalla toteutustavalla koko toiminta pitäisi ohjelmoida OnGui -luokkaa käyttäen. OnGui -luokkaa käyttäen scriptit olisivat olleet satojen rivien mittaisia. Uudella tavalla koko UI-rakenteen toteutus onnistuu hyvin vähällä ohjelmoinnilla ja lyhyillä scripteillä, joiden tulkinta on helppoa.

5 TESTAUS JA JOHTOPÄÄTÖKSET

5.1 Testausmenetelmät

Toteutetun logiikan testaus tapahtuu Unity3D:n Editorissa. Suurin osa mittareista on toteutettu julkisia muuttujia käyttäen, jotka mahdollistavat niiden seuraamisen editorin kautta. Toteutetut ominaisuudet huomioon ottaen, tärkein testattava toiminta on tiedon välittyminen oikein. Toimivuus voidaan todeta editorin Inspector-näkymästä.

Testaus suoritettiin pääasiassa kehitystyön ohella. Toimintalogiikan toteutuksen jälkeen testausta varten toteutetaan scripti, joka hoitaa pelaajan toimintaa. Tämän jälkeen tulkitaan onnistuiko tiedon välittyminen editorissa.

5.2 Pelaajan mittareiden testaus

Pelaajan luottamuksen sekä tunnettavuuden säätelyn testausta varten luotiin NpcActions -scripti. Scripti pitää sisällään toiminnan pelaajan varastamiselle ja se on kiinnitettynä NPC hahmoon. Kuviossa 14 on Steal-funktion koodi.

```
public void Steal()
{
    //Randomization to steal success
    float success = Random.Range(0f, 4f);

    //Successfull steal, move money from npc to player and addjust xp
    if (success < 0f)
    {
        GameObject.Find("PlayerUnit").GetComponent<PlayerInventory>().AddjustPlayerMoney(moneyValue);
        AddjustMoneyValue(-moneyValue);
        Debug.Log("Steal Successfull"); //Remove debugs
        GameObject.Find("GameManager").GetComponent<PlayerCharacterStatManager>().addjustXp(20);

    }

    //Steal failed decrease npc group reputation, increase notoriety
    else
    {
        GameObject.Find("GameManager").GetComponent<PlayerTrustManager>().AddjustTrust(tc.workingGroup, -5);
        Debug.Log("Steal Failed");
        GameObject.Find("GameManager").GetComponent<NotorietyHandler> ().AddjustNotoriety(tc.workingGroup, 5);
    }

    //Set cd for steal and remove button
    tc.canSteal = false;
    tc.stealTimer = Time.time;
}
```

Kuvio 14. Steal-funktio

Pelaajan painaessa UI-elementissä olevaa Steal-painiketta, kutsutaan Steal-funktiota pelaajan kohteelta. Funktiossa luodaan väliaikainen float tyyppinen muuttuja, jonka arvo arvotaan esimerkissä väliltä 0 – 4. Mikäli arvottu luku on esimerkiksi pienempi kun 3, varkaus onnistuu ja NPC-hahmon valuutta siirtyy pelaajalle. Mikäli varkaus epäonnistuu, tarkastetaan NPC-objektin TargetChecklist -scriptistä alue. Tämän jälkeen kutsutaan GameManager -objektilta PlayerTrustManager -scriptin AddjustTrust -funkiota, jossa arvon muuttaminen tapahtuu. Sama toiminne suoritetaan NotorietyHandler -scriptin AddjustNotoriety -funktiolle. Kumpaakin funktiota kutsuttaessa välitetään tieto alueesta sekä määrä, jolla muuttujan arvoa muutetaan. Esimerkissä varkaus epäonnistuu aina, jotta arvojen välitys voidaan varmistaa.

Jatkokehitystä varten luottamuksen ja tunnettavuuden muutosta voisi ohjata muuntautuvilla arvoilla. Esimerkiksi luottamusta ja tunnettavuutta muuttava määrä voisi tulla arvalla väliltä 5 – 10. Kyseinen väli tulisi testata eri arvoilla, kunnes oikea arvoväli löytyy. Logiikan toiminnan testausta varten ei ole tarve toteuttaa satunnaisuuteen nojaavaa arvojen muuttumisia, sillä se on logiikan toiminnan kannalta merkityksetöntä.

5.3 Levottomuuden säätelyn testaus

Levottomuuden vaikutusta käyttäytymiseen testataan luomalla NPC hahmolle, NpcBehavior -scripti. Scriptin avulla voidaan testata raja-arvojen ylittyessä, muuttuuko NPC-hahmon käytös. NpcBehavior -scripti pitää sisällään toiminnan pelaajan kanssa puhumiselle. Scripti on esitetty kuviossa 15.

```
//Player Talks to NPC
public void Talk()
{
    DisorderManager dm = GameObject.Find ("GameManager").GetComponent<DisorderManager> ();
    TargetChecklist tc = this.GetComponent<TargetChecklist> ();

    if (tc.workingGroup == 1)
    {
        //Check disorder amount
        //Log correct phrase

        if (dm.area1Disorder < 50f)
        {
            Debug.Log (phrase1);
        }

        else if (dm.area1Disorder >= 50f && dm.area1Disorder < 75f )
        {
            Debug.Log (phrase2);
        }

        else if (dm.area1Disorder >= 75 && dm.area1Disorder <= 100f )
        {
            //Can be modified to work with trust
            float randomPhrase =  Random.Range (0f, 2f);

            if (randomPhrase < 1f)
            {
                Debug.Log (phrase3);
            }

            else
                Debug.Log (phrase4);

        }

        else if (dm.area1Disorder > 100f )
        {
            Debug.Log (phrase5);
        }
    }

}
```

Kuvio 15. NpcBehavior -scripti

Talk-funktiota kutsuttaessa ensimmäisenä tarkastetaan, minkä alueen levottomuuden mukaan fraasit toistetaan. Testauksessa NPC-hahmon alue on 1, joten scriptissä esitetään vain alueen yksi toiminta. Vastaavalla logiikalla toteutetaan toiminta kaikille alueille. Mikäli alueen levottomuus on alle 50, ei NPC-hahmon ole syytä olla varautunut tai vihainen. Mikäli arvo on yli 50 mutta alle 75, NPC-hahmo toistaa ennalta määrätyn fraasin string tyyppisestä muuttujasta. Mikäli arvo on 75 – 100, arvotaan toistettava fraasi kahdesta vaihtoehdosta ja jos levottomuus on yli 100, toistetaan fraasi 5.

Vastaavaa logiikkaa käyttäen voidaan toteuttaa vihollishahmojen toiminta pelaajan kohtaamisen yhteydessä. Esimerkiksi jos levottomuus on alle 25, voidaan arpoa hyökkääkö vihollinen pelaajaa vastaan. Arvon ollessa 25 – 50, vihollinen

hyökkää pelaajaa vastaan. Arvon ollessa yli 50, vihollinen lähtee suorittamaan hälytystä.

Levottomuutta ohjaava logiikka saadaan yksinkertaisena toimimaan suhteellisen helposti. Kokonaisen pelin toteutus vaatii huomattavaa työmäärää, jotta hahmoihin saadaan toteutettua yksilöllisyyttä. Tämä vaatii useampia erilaisia vaihtoehtoja toimintojen suorittamiseen. Levottomuuden vaikutusta NPC-hahmojen käytökseen voi yhdistää pelaajan luottamukseen. Esimerkissä esitetyn satunnaisuuden voisi korvata pelaajan luottamuksen arvon tarkastuksella, jonka perusteella haluttu fraasi saadaan toistettua.

6 JOHTOPÄÄTÖKSET

Opinnäytetyön aiheen valinnan jälkeen ei ollut epäselvyyttä siitä, että työmäärä tulee olemaan suunnittelu- ja toteutusvaiheessa suuri. Suunnitteluvaiheessa tuli nopeasti selväksi, että työn kasassa pitäminen tulee olemaan haasteellista. Suunnitteluvaihe koostuikin pienempien elementtien rakentamisesta tähdäten samalla suurempaa kokonaisuutta hallitsevaan toimintaan. Suunnitteluvaiheessa erityisen opettavaa oli mittareiden rakentaminen siten, että ne eivät ole irrallaan pelin konseptista. Adaptiivisen pelimaailman suunnittelu olisi ollut erityisen vaikeaa, ilman paneutumista muihin, jo valmiisiin peleihin. Pelin ideaan ja tarinaan on lähes mahdoton luoda dynaamista pelimaailman muuntautumista ilman, että kehittäjällä on käsitys erilaisista menetelmistä. Taustatyötä tehdessäni havaitsin, että pelimaailman muuntautumista voidaan ohjata lukuisilla tavoilla ja vain kehittäjän mielikuvitus toimii rajana. Työssä toteutetussa logiikassa onkin samoja piirteitä kuin Dishonored ja Middle Earth: Shadow of Mordor peleissä. Yksinkertaisemmat mittarit pohjautuvat Dishonored-pelin toteutukseen. Maailman monipuolinen muuntautuminen puolestaan Middle Earth: Shadow of Mordor -peliin.

Toteutusvaiheessa havaitsin pientenkin toiminteiden toteuttamisen paisuvan nopeasti. Pelin idean mukaisesti kaikki toiminta voi vaikuttaa useampaan mittariin, joilla maailman logiikkaa ohjataan. Selkeän toimintalogiikan toteuttaminen vei aikaa ja viimeisimmän version toteuduttua, muiden toiminteiden kehittäminen muuttui huomattavasti helpommaksi. Työssä toteutettua logiikkaa tehdessäni havaitsin myös sen, että aiempi suunnitelma voi muuttua hyvinkin nopeasti. Toimintalogiikkaa koskevat scriptit, tehtiin useampaan kertaan uudelleen työn edetessä. Suunnitteluvaiheessa en kyennyt huomaamaan mahdollisia suunnitelman ongelmakohtia, vaan ne tulivat vastaan toteutusvaiheessa. Tilanteen korjaaminen oli helpointa scriptien uudelleen kirjoittamisella.

Työtä tehdessäni opin paljon suunnittelun vaikeudesta, kun tuotteelle ei ole tarkkaa määritelmää taustalla. Suunnitelma kehittyi vielä pitkään toteutuksen aikana. Pelikehityksessä näenkin tärkeänä, että mitään ei lyödä lopullisesti lukkoon ennen kuin tuote on kokonaan valmis.

Opinnäytetyön tavoitteeksi asetin toimivan logiikan toteutuksen. Saavutin tavoitteeni, sillä tiedonvälitys kaikkien osien kohdalla toimii. Myös hahmojen käyttäytymistä voidaan ohjata muuttujatasolla toteutetun logiikan pohjalta. Eri mittareiden arvoasteikot kuitenkin tulee vielä testata tarkemmin. Luotettavaan testaustulokseen ei kuitenkaan päästä, ennen kuin pelin mekaniikat ovat pitemmällä ja peliä voi jo pelata.

Opinnäytetyötä voi käyttää erilaisten pelikehitys kurssien opetuksen tukena muun muassa suunnittelun osalta. Pelikehityksestä kiinnostuneet henkilöt voivat hyödyntää opinnäytetyötä pohjana omalle itseopiskelulleen.

LÄHTEET

Bethesda Softworks 2012a. Dishonored. Viitattu 4.11.2015 http://beth-soft.com/en-gb/games/dishonored.

Dishonored Wikia 2012b. Dishonored. Viitattu 4.11.2015 http://dishon-ored.wikia.com/wiki/Dishonored.

Dishonored Wikia 2013c. Chaos. Viitattu 4.11.2015 http://dishon-ored.wikia.com/wiki/Chaos

Shadow of Mordor Wikia 2014 The spirit of Mordor. Viitattu 6.1.2016 http://sha-dowofmordor.wikia.com/wiki/The_Spirit_of_Mordor

Unity3D Documentation 2015a. Learning the Interface. Viitattu 23.2.2016 http://docs.unity3d.com/Manual/LearningtheInterface.html

Unity3D Documentation 2015b. Object. Viitattu 21.2.2016 http://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html

Unity3D Documentation 2015c. Collider. Viitattu 21.2.2016 http://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html

Unity3D Documentation 2015d. UI. Viitattu 23.2.2016 http://docs.unity3d.com/Manual/UISystem.html

Unity3D Documentation 2015e. Canvas. Viitattu 23.2.2016 http://docs.unity3d.com/Manual/UICanvas.html