Topi Kiiskilä

Tekoälyn hallintajärjestelmien soveltaminen pelinkehityksessä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

26.11.2023

Tiivistelmä

Tekijä:	Topi Kiiskilä
Otsikko:	Tekoälyn hallintajärjestelmien soveltaminen pelinkehityksessä
Sivumäärä:	37 sivua
Aika:	26.11.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaaja:	Lehtori Antti Laiho

Insinöörityön tarkoituksena oli kehittää pelisovellukseen tekoälyn hallintajärjestelmä Unity-pelimoottorilla ja tutkia, oliko se mahdollista toteuttaa käyttäen Square Enixin tekoälymallia järjestelmän pohjana. Tavoitteena oli luoda pelistä toimiva prototyyppiversio, jota pystyttäisiin myöhemmin laajentamaan valmiiksi täydeksi peliksi.

Insinöörityössä perehdyttiin tekoälyyn ja sen kehitysprosessiin. Tavoitteena oli saada käsitys, miten tekoäly, sen vaatimukset ja ominaisuudet peleissä ovat kehittyneet historian saatossa. Lisäksi tavoitteena oli tutkia eri tapoja tekoälyjärjestelmien kehittämiseen. Tekoäly käyttö ja sen tarve yleistyvät peleissä koko ajan, ja pelinkehittäjät pyrkivät tekemään aina vain monimutkaisempia tekoälyjä peleihin, jotta pelaajien mielenkiinto peliin säilyisi.

Projektissa käytettiin Square Enixin tekoälymallia, joka perustuu navigaation tekoälyn, pelijärjestelmien tekoälyn ja pelihahmojen tekoälyjen muodostamaan hallintajärjestelmään. Järjestelmä toteutettiin Unity-pelimoottorilla. Projektin tarkoituksena oli myös tutkia erilaisia tapoja toteuttaa hallintajärjestelmiä.

Insinöörityön lopputuloksena saatiin kehitettyä hallintajärjestelmä ja pelin prototyyppiversio valmiiksi. Hallintajärjestelmä saatiin toteutettua tekoälymallin mukaan ilman suurempia ongelmia. Järjestelmästä tuli yksinkertainen ja helposti laajennettava tulevaisuutta varten. Projektissa tehdyssä prototyyppiversiossa saatiin jaettua halutut tapahtumat ja kohtaukset pelissä omiin testitasoihinsa, minkä vuoksi peliä on helpompi testata. Pelistä saatiin tehtyä toimiva ja pelattava kokonaisuus.

Avainsanat: tekoäly, pelinkehitys

Abstract

Author:	Topi Kiiskilä
Title:	AI Game management systems in game development
Number of Pages:	37 pages
Date:	26 November 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Game Applications
Supervisor:	Antti Laiho, Senior Lecturer

The purpose of this final year project was to develop an artificial intelligence control system for a game using the Unity game engine. Another goal was to investigate whether it was possible to implement the system into a game when using the AI model of Square Enix as the basis for the system. The final goal was to create a functional prototype version of the game that could later be easily expanded into a full game.

This thesis delves into artificial intelligence and its development process. The aim was to gain an understanding of how Game AI has evolved over time. Another objective was to study different ways of developing AI systems. The use and demand of AI in games is constantly increasing. That is why game developers need to create more complex AI systems in order to keep players interested in their games.

The game project used the AI model of Square Enix, which is based on three different parts: Navigation AI, Meta AI and Character AI. The system was developed on the Unity game engine. Another purpose of this project was to explore various different ways of implementing AI and control systems.

As a result of this final year project, a control system and a prototype version of the game were successfully developed. The control system was implemented according to the Square Enix AI model without any major bugs or issues. The system is simple and it is easily expandable in the future. The prototype version was successfully developed. It had two important game levels separated into their own test scenes, making it easier to test the game and its parts in the future. The finished prototype is a functional and playable game.

Keywords: Game AI, game development

Sisällys

Lyhenteet

AI: *Artificial intelligence*. Tekoäly. Teknologia, jonka avulla koneet voivat oppia ja mukautua uusiin tilanteisiin.

FPS: *First-person shooter*. Ensimmäisen persoonan ammuntapeli. Peligenre, jossa pelaaja ohjaa pelihahmoaan ensimmäisessä persoonassa. Pelihahmolla on yleensä jokin ampuva ase käytössään, jolla hän ampuu vihollisia.

HP: *Health Points*. Elinvoimapisteet. Pelihahmon tilastollinen kestävyys ja kyky ottaa vastaan vahinkoa.

HUD: *Heads-Up Display*. Pelaajalle pelissä näytetty reaaliaikainen informaatio pelaajan pelitilanteesta. HUD ei yleensä ole yhteydessä pelimaailmaan, vaan on ikään kuin sen ja pelaajan välissä.

NavMesh: *Navigation Mesh*. Navigointiverkko. Ohjelmisto tiedon tehokkaan hakemisen, säilyttämisen ja päivittämisen toteuttamiseksi.

RPG: *Role-player game*. Roolipeli. Peligenre, jossa pelaaja ohjaa hahmoa tai hahmoja läpi vuorovaikutteisen tarinan. Pelaaja pystyy pelissä eläytymään hahmonsa rooliin tarinassa.

UI: *User Interface*. Käyttöliittymä. Kaikki pelaajan näkemä ruudulla. Pelaaja käyttää käyttöliittymää ollessaan pelin kanssa vuorovaikutuksessa.

UX: *User Experience*. Käyttäjäkokemus. Pelinkehitystiimin osa, joka vastaa pelaajan pelikokemuksen parantamisesta.

1 Johdanto

Sovelluksissa ja koneissa käytetään nykypäivänä yhä enemmän tekoälyä ratkaisuna erilaisiin tilanteisiin. Tekoäly (artificial intelligence, AI) kehittyy jatkuvasti, ja se on yhä isommassa roolissa kaikissa uusissa teknologioissa. Pelisovelluksissa tekoäly on todella tärkeässä asemassa, sillä lähes kaikki peleissä tapahtuvat tilanteet ovat ainakin jollain tasolla tietokoneen ohjaamia. Pelaaja tekee peleissä omia valintojaan, joihin peli reagoi eri tavoin.

Tekoälyllä tarkoitetaan koneille annettua älykkyyttä, jonka avulla ne pystyvät käyttämään ihmisten taitoja, kuten oppimista, ongelmanratkaisua ja suunnittelua. Tekoälylle annetaan informaatiota, jonka avulla tehdään päätöksiä ja valintoja. Informaatiota voidaan kerätä esimerkiksi tietokannoista tai tietokoneeseen yhdistetystä kamerasta.

Usein peleissä sana tekoäly yhdistetään pelin ohjaamiin vihollisiin tai muihin tietokoneohjattuihin pelihahmoihin, mutta todellisuudessa tekoäly ohjaa myös monia muita erilaisia järjestelmiä peleissä. Edellä mainittujen erilaisten tekoälyjärjestelmien luominen on tämän insinöörityön tarkoitus. Järjestelmien pohjana käytetään Square Enixin tekoälyjärjestelmissä käytettyä teoriaa, joka koostuu kolmesta eri osasta:

- hahmojen tekoäly (Character AI)
- pelijärjestelmien tekoäly (Meta AI)
- navigaatiotekoäly (Navigation AI).

Nämä kolme osaa eivät ole ainoat peleissä käytetyt järjestelmät, mutta ne luovat hyvän pohjan pelien tekoälyn tekemiseen. [1; 2.]

Pelinkehityksessä käytetään nykypäivänä monia eri työkaluja pelien luomiseen. Niistä yleisimmät ovat Unity ja Unreal Engine. Molemmissa sovelluksissa tarjotaan paljon työkaluja tekoälyn kehittämiseen. Tämän insinöörityön tekemiseen käytettiin Unity-pelimoottoria, jonka avulla tehtiin prototyyppi pelistä. Unityn edi-

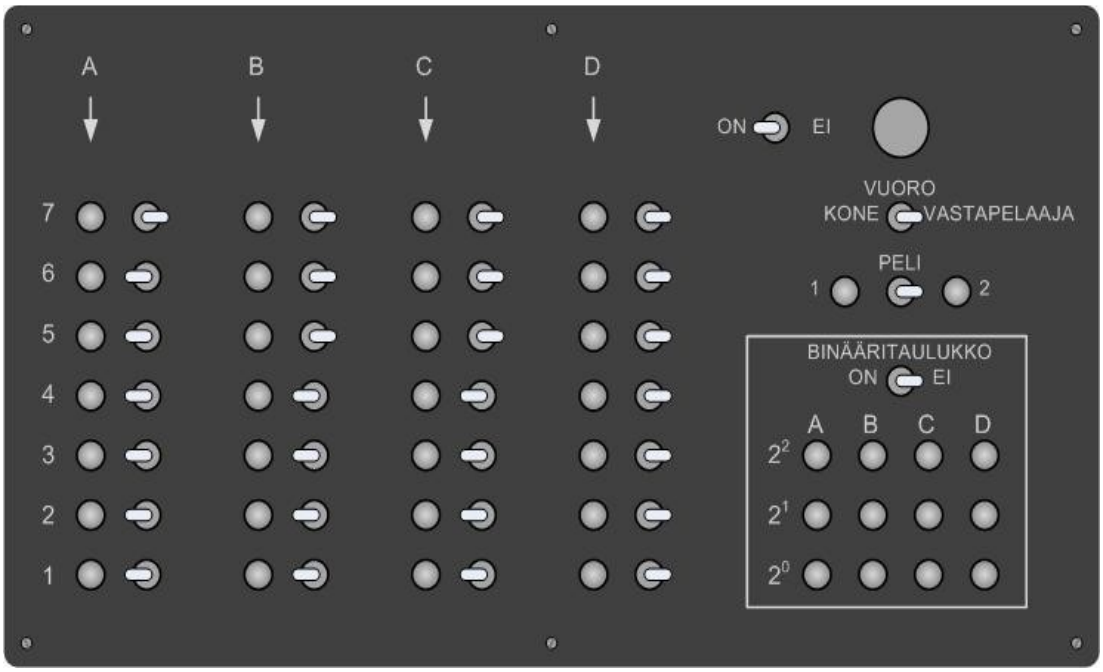
torisovelluksella pystytään tekemään monimutkaisia pelejä. Editorissa on tarjolla myös erilaisia valmiita tekoälyä hyödyntäviä ratkaisuja pelihahmojen ja pelin kulun hallintaan. Työssä tutkitaan myös erilaisia tapoja toteuttaa hallintajärjestelmiä peleihin, tekoälyn rajoitteita sekä tekoälyn käytön haasteita ja samalla otetaan selvää, onko Square Enixin tekoälymalli mahdollista toteuttaa Unityssä.

2 Tekoälyn hallintajärjestelmät

Tekoäly on tärkeä osa pelinkehitystä. Sitä käytetään ohjaamaan peliä ja pelihahmoja. Tekoälyn tehtävänä on luoda pelaajalle tunne siitä, että hän pystyy vaikuttamaan pelin etenemiseen ja että peli reagoi pelaajan valintoihin ja tekee valintoja niiden perusteella. Tekoäly voi olla myös hyvin yksinkertaista. Kahden pisteen välillä liikkuva vihollinen yleensä hyödyntää liikkumiseen ja päätöksentekoon liittyviä tekoälyjärjestelmiä.

2.1 Tietokoneohjattujen hahmojen historia

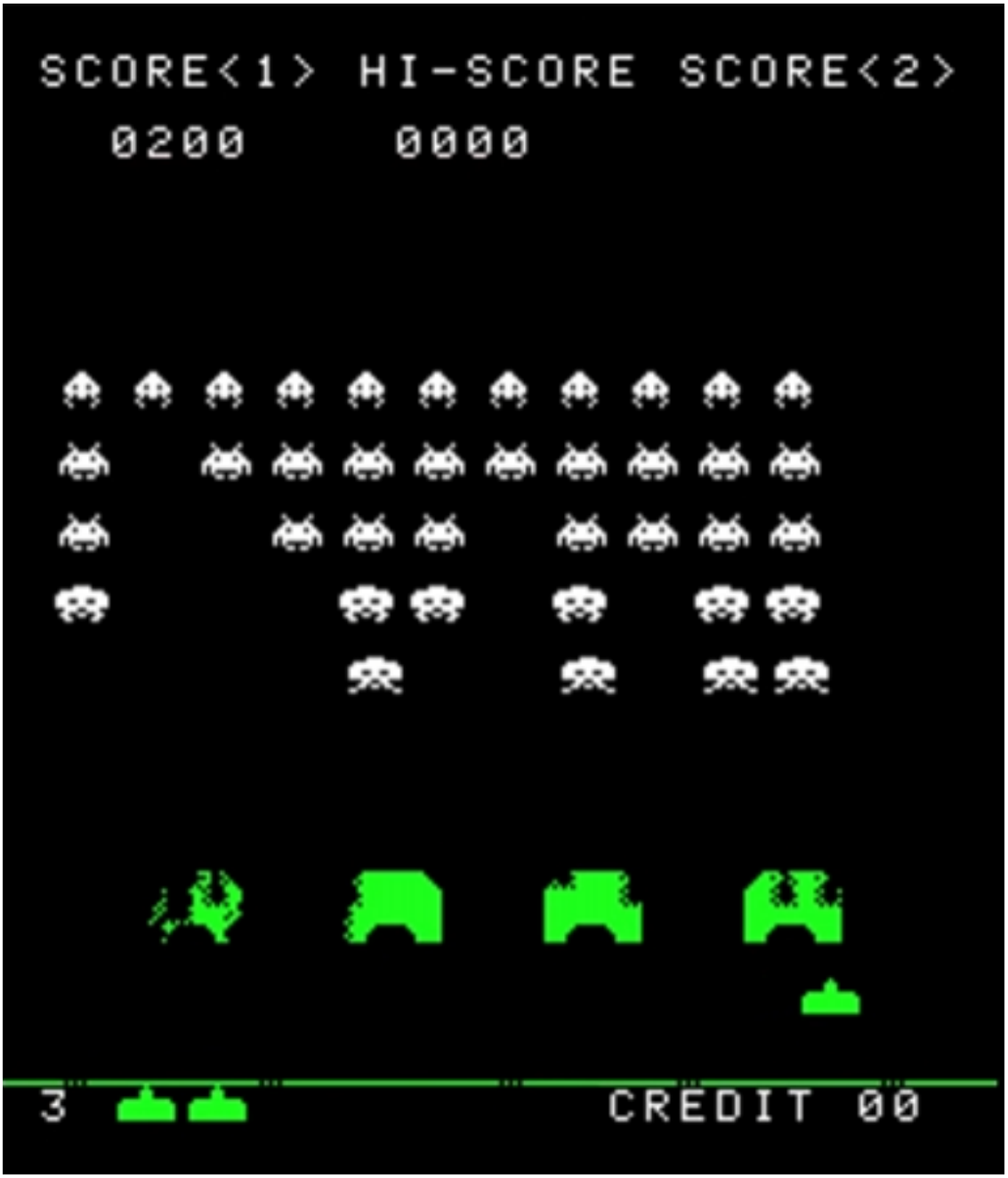
Pelinkehityksessä tekoäly on aina ollut isossa asemassa. Yksi ensimmäisistä tekoälyistä Suomessa oli 1950-luvulla kehitetty Aapeli, joka oli ensimmäinen suomalainen digitaalinen pelilaite. Se pystyi pelaamaan Nim-pelin digitaalista versiota. Nim-peli on kaksinpeli, jossa pelaajat poistavat kasasta objekteja vuorotellen. Pelin aina häviää kasasta viimeisen objektin poistava pelaaja. Peliä on mahdollista pelata täydellisesti, jolloin peli ei ole enää haastava tai mielenkiintoinen oikeat siirrot tietävälle pelaajalle. Aapelia kutsuttiin alun perin Nim-peliksi, jonka suunnitteli Hans Andersin. [3.] Laite on nähtävissä kuvassa 1.



Kuva 1: Nim-pelilaitteen käyttöpaneeli [4].

Ensimmäiset videopelit, kuten Pong, eivät käyttäneet tekoälyä. Näissä peleissä usein pelaajat laitettiin tosiaan vastaan, joten ei ollut tarvetta kehittää tekoäly-vastustajia pelaajille. 1970-luvulla yksinpelimahdollisuudet alkoivat yleistymään. Näihin peleihin kehitettiin tekoälyvihollisia. Viholliset liikkuivat tallennettujen liikesarjojen ja käyttäytymismallien avulla. Jossain peleissä oli myös mahdollista luoda hieman satunnaisuutta vihollisten liikkumiseen, sillä pelilaitteiden prosessorien suorituskyky parani 1970-luvun loppua kohden.

Vuonna 1978 julkaistu Space Invaders on yksi aikakautensa tunnetuimpia videopelejä. Pelissä pelaaja liikuttaa lasertykillä varustettua avaruusalusta ruudun alalaidassa. Tarkoituksena on ampua tykillä mahdollisimman monta yläpuolelta lähestyvää avaruusolioita kuvan 2 osoittamalla tavalla. Vihollisten liike perustuu pelin jatkuessa vaikeutuviin tallennettuihin liikesarjoihin ja kykyyn reagoida pelaajan syöttämiin ohjausliikkeisiin vaihtamalla liikkeen suuntaa satunnaisesti. [5.]



Kuva 2: Taito-pelistudion kehittämä Space Invaders [5].

Roolipelit, eli RPG:t, olivat ensimmäisiä videopelejä, jotka antoivat pelaajille mahdollisuuden hallita tietokoneohjattujen hahmojen käyttäytymistä, taktiikoita ja rutiinia pelin sisällä. 1990-luvulta alkaen pelinkehityksessä tarvittiin monimutkaisempia työkaluja tekoälyjärjestelmien luomiseen ja myös nykypäivänä käytetyt tilakoneet (state machine) alkoivat yleistymään. Tilakoneiden avulla voitiin pelihahmoille tehdä yksinkertaisia tekoälyjä, joiden avulla hahmoja pystyi hallit-

semaan. Dune 2 (1992) -pelissä tilakoneen logiikkaa käytettiin kaikissa pelihahmoissa. Sen avulla pelaajan oli mahdollista antaa pelihahmoille suoria käskyjä liikkua tai hyökätä tiettyjen vihollisten kimppuun. [6.]

Samoihin aikoihin alettiin myös havaitsemaan ongelmia ja tekoälyn käytön haasteita videopeleissä. Pelinkehittäjät antoivat tietokoneohjatuille hahmoille liian voimakkaita kykyjä, ja hahmot pystyivät huijaamaan pelaajalle asetettuja rajoituksia. Tekoälyn ohjaamilla hahmoilla saattoi olla myös muita haasteita, kuten esimerkiksi ongelmia navigoinnissa, ja ne saattoivat jäädä jumiin liikkuessaan pelimaailmassa.

2.2 Moderni tekoäly peleissä

Nykypäivän peleissä vaaditaan tekoälyltä paljon enemmän kuin ennen. Yksi tärkeimmistä tekoälyn sovelluksista nykypäivän peleissä on polunetsintä (pathfinding). Polunetsintään tarkoitettu tekoäly yrittää löytää mahdollisimman lyhyen reitin kahden eri pisteen välillä. Myös päätöksenteon tekoälyssä on käytössä uudempia menetelmiä. Tilakoneet ovat yhä toimiva ratkaisu, mutta yleisemmäksi tavaksi ohjata tekoälyä on noussut päätöspuu.

Hyvin usein modernien pelien kehitysvaiheessa havaitut ongelmat tekoälyn ohjaamissa hahmoissa korjataan mieluummin muokkaamalla pelimaailmaa kuin parantamalla tekoälyn ongelmanratkaisutaitoa. On huomattavasti helpompaa muokata estettä, joka hajottaa pelihahmon liikkumisen, kuin parantaa pelin tekoälyn polunetsintäjärjestelmiä. Valven vuonna 1998 julkaisemassa pelissä Half-Life (kuva 3) viholliset eivät edes yritä juosta karkuun pelaajan heittämiä kranaatteja, koska pelinkehittäjillä oli ongelmia saada kranaattiin reagoivien pelihahmojen reaktiot näyttämään hyviltä. Pelihahmojen epätoivoinen juokseminen näytti tyhmältä, joten pelinkehittäjät päättivät, että vihollisten on parempi kyykistyä, suojata päätä ja huutaa kranaatista, kuin epätoivoisesti yrittää päästä pois kranaatin alta. Tämä toimi pelissä hyvin. [7.]

Kuva 3: Taistelutilanne Half-Life-pelissä [8].

Moderni tekoäly pystyy myös keräämään enemmän tietoa pelimaailmasta. Nykypäivän teknologioilla on mahdollista antaa tekoälylle enemmän aisteja, mikä on tullut tärkeään käyttöön erityisesti ns. hiiviskelypeleissä (stealth games). Vanhoissa peleissä pelihahmot lähinnä havaitsivat pelaajan näkökentässä tai kosketuksella, mutta nykyään on mahdollista antaa tekoälylle myös kuuloaisti tai tehdä sille kyky havaita pelaaja vain, jos pelaajan käytös, asusteet tai sijainti poikkeavat pelaajalle sallituista säännöistä ja alueista. [9.]

Ensimmäisen persoonan ammuntapeleissä (first-person shooter, FPS) vihollisilla pitää olla liikkumiseen ja ampumiseen omat tekoälynsä. Ampumiseen liittyvän tekoälyn tekeminen ei kuitenkaan ole helppoa. Vihollisista saa yllättävän helposti tehtyä epäreiluja, sillä tietokone pystyy prosessoimaan saamaansa dataa nopeammin kuin ihminen ehtii reagoimaan näkemäänsä viholliseen ja painamaan ampumisnäppäintä. Tästä syystä FPS-genren videopeleissä on tyypillisesti pelihahmojen tekoälystä tehty tarkoituksella vähän tyhmiä ja niiden reaktionopeuksia ja tarkkuutta on huomattavasti alennettu.

Koska tekoälyllä on enemmän tietoa pelaajasta ja pelimaailmasta, peleistä on nykyään helppo tehdä liian vaikeita. Pelejä, joissa pelaajilla on tiedossaan kaikki peliin liittyvä, voidaan pelata täydellisesti. Näissä peleissä on mahdollista pelata siten, että aina voittaa tai pelaa vähintään tasapelin, eli voidaan sanoa, että peli on ”ratkaistu”. Tekoälyn on lähes mahdotonta hävitä tällaista ”ratkaistua” peliä. Tällaisista peleistä hyviä esimerkkejä ovat aikaisemmin mainittu Nim sekä ristinolla. [10.]

Uusimmissa peleissä saattaa olla käytössä myös koneoppivaa tekoälyä (machine learning AI). Koneoppiva tekoäly pystyy nimensä mukaan oppimaan kokemastaan ja parantamaan omaa toimivuuttaan seuraavaa kertaa varten. Koneoppivaa tekoälyä voidaan peleissä käyttää esimerkiksi säätämään pelin vaikeutta tai ohjaamaan pelihahmoja. Yksi esimerkki koneoppivasta tekoälystä on syväoppiva tekoäly (deep learning), jota voidaan käyttää esimerkiksi chatbotien kehityksessä. Syväoppivan tekoälyn tehtävä näissä sovelluksissa on ymmärtää käyttäjän lähettämä viesti ja tehdä jatkokysymys tai vastata viestiin botille annetun datan ja viestin sisällön perusteella. [11.]

2.3 Ratkaisuja hallintajärjestelmien luomiseen

Polunetsintään on monta eri toteutusvaihtoehtoa. Kuuluisimpia polunetsintäalgoritmeja ovat A*- ja Dijikstran-algoritmit. Polunetsintään voidaan myös käyttää täysin erilaista mallia, jossa kaikki polunetsintään liittyvä tieto kyvystä liikkua paikasta toiseen upotetaan pelimaailman maastoon. Tätä kutsutaan navigointiverkoksi (navigation mesh) tai lyhennyksenä NavMesh. Tällä mallilla on helpompi hallita pelihahmojen liikettä, kun kaikki navigointiin liittyvä laskenta ja polunetsintä tehdään pelimaailmaan eikä jokaiselle hahmolle tarvitse laskea erikseen uutta reittiä pelimaailman muuttuessa. [12.]

Käyttäytymismalleista tilakone ja päätöspuu ovat helppo ja hyvä ratkaisu pelinkehitykseen. Tilakoneissa tekoäly vaihtaa käyttäytymistään erilaisten toimintatilojen välillä. Tekoäly voi esimerkiksi olla ensin lepotilassa ja saa käskyn liikkua, jolloin se vaihtaa itsensä liikkumistilaan. Pelaajan hahmon nähdessään tekoäly

voi vaihtaa itsensä esimerkiksi hyökkäystilaan, jolloin sen ohjaama vihollinen hyökkää pelaajaa kohti. Tilakone sopii hyvin esimerkiksi todella yksinkertaisiin vihollisiin tai kaupungin vartiointiin tehtyihin pelihahmoihin. Päätöspuut sen sijaan soveltuvat paremmin peleihin, joissa tietokoneohjatun hahmon tarvitsee pystyä tekemään paljon erilaisia valintoja ja toimintoja, jotka yleensä riippuvat pelaajan valinnoista, pelin etenemisestä ja pelimaailman tilanteista. Päätöspuiden käyttöön liittyy myös omat pienet ongelmansa, sillä tietyt päätökset saattavat toistua monta kertaa, minkä takia pelihahmo saattaa toistaa samaa toimintoa useasti. [13.]

Vihollisen tekoäly tarvitsee myös tavan löytää pelaaja. Tilakoneella hallittua yksinkertaista vihollista voidaan käskeä etsimään pelaaja ja hyökkäämään. Kun halutaan, että viholliset etsivät ja metsästävät pelaajaa, niille annetaan mahdollisuus havaita pelaajan tekemät jalanjäljet ja askelien äänet. Pelin järjestelmät tai toiset viholliset voivat myös antaa yksittäiselle tilakoneella ohjatulle viholliselle pelaajan tarkan paikan. Näin on mahdollista hälyttää kaikki viholliset pelaajan kimppuun, kun yksikin vihollisista havaitsee pelaajan.

3 Tekoälyn hallinnan työkalut

Pelin tekemiseen tarvitaan hyvä suunnitelma. Suunnittelu on tärkeä osa pelinkehitystä. Aluksi päätetään pelin genre ja tyyli. Pelikehitys voidaan aloittaa paljon nopeammin, kun kaikilla kehitystiimin jäsenillä on samanlainen visio pelistä. Peliä suunnitellessa voidaan myös päättää, minkälaisia tekoälyjärjestelmiä peliin halutaan lisätä. Square Enixin tekoälymallia käytettiin insinöörityön peliprojektin suunnittelussa.

Square Enix on japanilainen yritys, joka keskittyy pelinkehitykseen ja julkaisuun. Se on tunnettu useista roolipelisarjoistaan, joista kuuluisimpia ovat Final Fantasy ja Dragon Quest. Nykyisin Square Enix kehittää ja julkaisee pelejä useista eri genreistä. Tekoälymallin järjestelmää Square Enix on käyttänyt esimerkiksi kriitikoiden ylistämässä Final Fantasy XIV: Online -pelissä. [14.]

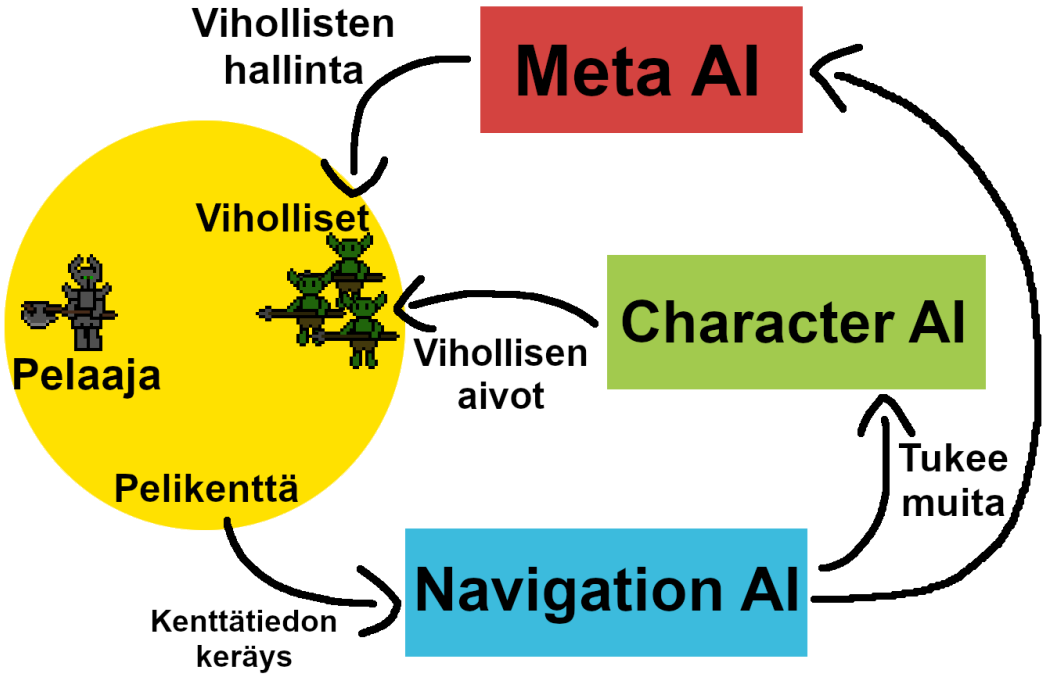
3.1 Pelin tekemisen lähtökohdat

Ennen tekoälyn hallinnan selittämistä on hyvä tarkastella enemmän videopelien rakennetta. Pelin rakenteen selittämiseen Square Enix käyttää neljävaiheista kerrosjärjestelmää. Kerroksien lisäksi pelissä on tekoälyn hallintajärjestelmä, joka vastaa kerroksien välisestä kommunikoinnista.

Pelimaailma koostuu erilaisista peliobjekteista (game objects), jotka yleensä ovat yksittäisiä esineitä kentässä, kuten kivet, peliesineet ja talot. Pelimaailman maata kutsutaan maastoksi (terrain), johon kuuluu esimerkiksi metsiä, seiniä ja siltoja. Maastoon kuuluvat peliobjektit eivät pysty liikkumaan. Pelitason suunnittelu on peliobjekteista ja maastosta koostuva kokonaisuus. Square Enixin nelikerroksisessa järjestelmässä käytetään kerroksien välistä tärkeysjärjestystä. Ensimmäinen kerros on graafinen kerros (graphics layer), jonka tehtävänä on näyttää käyttäjälle kaikki pelissä näkyvät mallit (Models), tekstuurit (textures), varjostukset (shaders) sekä muut pelin visuaaliset peliobjektit. Seuraavassa kerroksessa on törmäyskerros (collision layer), joka pitää sisällään pelissä tapahtuvat törmäykset ja yhteenotot. Tämä kerros vastaa esimerkiksi pelaajan ja peliobjektin välisistä vuorovaikutuksista (interaction). Tämän jälkeen on tekoälykerros (AI layer). Tekoälykerrokseen kuuluu kaikki pelissä käytetty tieto pelimaailman maastosta, peliobjekteista, tilanteista ja pelin säännöistä. Jokaisella peliobjektilla on omat tietonsa, joita tekoälyn hallintajärjestelmä tarvitsee erilaisten peliobjektiin liittyvien päätösten tekoon. Viimeisenä kerroksena on animaatiokerros (animation layer), joka huolehtii peliobjektin liikkeestä ja animaatioista. Pelihahmoilla voi olla luuranko, jonka avulla animaatiokerros voi liikuttaa peliobjektia käyttämällä tehtyjä animaatioita tai dynaamisesti muuttaa liikettä annetun animaatioinformaation avulla. Jokainen kerros laskee omat tehtävänsä ja kommunikoi tietoja pelihahmon hallintajärjestelmälle, joka puolestaan saadun tiedon avulla palauttaa kerroksille käskyjä, mitä tulee tehdä. [1.]

Insinöörityön peliprojektiin suunniteltiin tekoäly Square Enixin käyttämällä tekoälymallilla. Malli perustuu kolmeen osaan, jotka ovat nähtävissä kuvassa 4. Yksittäisiä pelihahmoja hallitseva Character AI on vastuussa kaikkien pelihahmo-

jen hallinnasta. Meta AI on pelin tekoälyn hallintajärjestelmä. Navigation AI puolestaan vastaa pelissä käytetystä navigaatiotekoälystä, joka auttaa hahmoja löytämään reitin paikasta toiseen pelimaailmassa. [2.]



Kuva 4: Tekoälyn hallintajärjestelmän toimintaperiaate [1].

Näillä kolmella tekoälyn osalla saadaan luotua tekoälyn hallintajärjestelmään hyvä pohja, joka sopii mihin tahansa peliprojektiin. Järjestelmää pystytään myös muokkaamaan tarpeen vaatiessa.

3.2 Navigaatio pelimaailmassa

Navigaatiosta vastuussa oleva tekoäly on usein yksinkertaistettu pelissä käytetyn editoriohjelman ominaisuuksiin. Unity tarjoaa monta eri vaihtoehtoa pelimaailmassa liikkumiseen. Pelihahmoja voidaan liikuttaa fysiikkamoottorilla, kun hahmoon kohdistetaan voimaa haluttuun liikkumissuuntaan. Pelihahmo ilmoittaa pelille, mihin hahmon on tarkoitus päästä, ja pelin hallintajärjestelmä antaa hahmolle halutun paikan ja suunnan. Navigaatiojärjestelmällä pystytään anta-

maan pelihahmoille tarkka liikkeen suunta, joka osaa välttää mahdollisia pelimaailman esteitä.

Unity tarjoaa käyttäjille valmiiksi tehdyn navigaatioverkkotekoälyn, joka osaa välttää pelimaailman esteitä ja pystyy liikuttamaan pelihahmoja paikasta toiseen ilman suurempia ongelmia. Navigaatioverkot sisältävät tietoa, jonka sisältöä tutkitaan ja käsitellään Navigation AI -koodissa. Sieltä käsitelty tieto viedään Meta AI- ja Character AI -tekoälyosiin kuvan 4 osoittamalla tavalla. Navigaatioverkossa kaikkiin peliobjekteihin ja maastoon on merkitty erilaista informaatiota. Esimerkiksi esteissä voi olla merkintä siitä, että niiden yli ei voi kävellä, joten navigaatiotekoälyn pitää osata ohjata pelihahmot kiertämään niiden ympäri. Verkkoon voidaan myös merkitä maaston tyyppi (terrain type), kuten luminen maa. Lumiseen maahan voi myös liittyä liikkeen nopeutta hidastavia tekijöitä, minkä navigaatiotekoäly ottaa huomioon laskiessaan reittejä pelihahmoille. [2.]

Navigaatioverkkoon voidaan myös tallentaa erilaisia pisteitä, joilla on eri tarkoituksia. Esimerkiksi pelimaailmaan voidaan tallentaa penkkiin piste, jossa pelihahmo pystyy istumaan. Koska piste on tallennettu pelimaailmaan, pelihahmon on helppo löytää peliobjektin ja olla vuorovaikutuksessa sen kanssa. Samalla tavalla voidaan tallentaa peliobjekteihin käskyjä. Näitä käskyjä voivat olla esimerkiksi paikallaan olevan kiven liikkumissuunta, jonka takia kiveen merkityn informaation mukaan kiveä voi liikuttaa vain yhteen suuntaan. Jokaisella pelihahmolla tulee myös olla oma navigaatioon liittyvä osuus, joka kertoo navigaatiotekoälylle esimerkiksi hahmon koon, nopeuden tai suurimman mahdollisen hypyn pituuden kahden eri navigaatioverkon pisteen välillä.

Esimerkkinä voivat olla myös pelin tapahtumat, joiden lopputulos voidaan merkitä navigaatiotekoälyn avulla. Pelaaja on vuorovaikutuksessa kaupungissa olevien pelihahmojen kanssa, mikä puolestaan vaikuttaa pelaajan maineeseen (reputation) koko kaupungin alueella. Kaikki kaupungin asukkaat voivat navigaatioon tallennetun mainearvon mukaan reagoida pelaajaan eri tavoin. [2.]

3.3 Tietokoneohjatun hahmon hallinta

Pelihahmon hallintaan tarvitaan useita osia. Tekoälyn tarkoitus on toimia hahmon aivoina, joten koodista saattaa helposti tulla monimutkaista. Hahmon Character AI koostuu aisteista, jotka havainnoivat ympäröivää pelimaailmaa, hahmon fyysisestä ruumiista sekä hahmon tietämästä tiedosta. Tekoälyn on pystyttävä reagoimaan pelin muuttuviin tilanteisiin ja antamaan käskyjä havaitun ja saadun tiedon perusteella muille tekoälyn osille.

Hahmon tekoälyn rakentaminen alkaa aistien luomisella. Yleisimmät moderneissa peleissä käytetyt tekoälyn aistit ovat näkö, kuulo ja tunto sekä kyky noudattaa muiden pelihahmojen antamia verbaalisia komentoja. Tekoälyn aisteja kutsutaan myös pelihahmon sensoreiksi (sensor). Sensorit yleisesti toimivat tekoälyn rajoitteina, jotta pelihahmoilla ei olisi pelaajasta tai muusta pelimaailmasta täydellistä tietoa. Pelin vaikeus voi kärsiä tilanteesta, jossa pelihahmot tietävät pelistä pelaajaa enemmän ilman hyvää syytä. Seuraavaksi pelihahmolle lisätään tekoälyn efektori (effector). Sen tehtävä on vastata hahmon saamaan tietoon. Efektori kuuluu törmäyskerrokseen, sillä hahmon vuorovaikutus muiden peliobjektien kanssa on tärkeä osa sen toimintaa. Sensorin ja efektorin välistä vuorovaikutusta voidaan kutsua myös ”koordinaatioksi”. [1; 2.]

Viimeinen iso osa pelihahmoa on tekoälyn älykkyys. Se puolestaan koostuu kolmesta pääosasta:

- tunnistamisesta (Recognition)
- päätöksenteosta (Decision-making)
- toiminnallisesta osasta (Action-making).

Tekoälyn on tärkeä osata tunnistaa, mitä sen on mahdollista tehdä pelimaailmassa. Kun tekoäly tunnistaa kaikki mahdolliset toiminnot, siirrytään päätöksentekoon. Päätöksenteosta vastuussa oleva osa päättää kaikista mahdollisista toiminnoista parhaan, minkä jälkeen tekoälyn toiminnallinen osa käskee pelihahmoa suorittamaan halutun toiminnon. Toiminto voi olla esimerkiksi pelihahmon liikkuminen. [2; 15.]

Pelihahmon päätöksenteko on älykkyyden tärkein osa. Se on myös samalla kaikkein monimutkaisin osa tekoälyä, joka voidaan toteuttaa monella eri periaatteella. Tilakoneet ja päätöspuut tekevät valintoja eri tavoilla. Päätöspuut voivat esimerkiksi noudattaa sääntöjä. Näistä säännöistä kaikkein yleisimpiä ovat *if*-lauseet. Niiden avulla voidaan valita tietty toiminto, jos haluttu ehto on täytetty. Tilakoneet toimivat lähes samalla periaatteella, mutta niissä on eri tilojen vaihtumisen välillä yleensä yksi if-lause, kun päätöspuussa saattaa olla monta samaan aikaan. Muita yleisiä valintamalleja ovat käytöspuut ja lopputulokseen, tehtävään, hyödyllisyyteen tai simulaatioon pohjautuvat valintamallit. Esimerkkinä käytöspuista voisi käyttää The Sims 3 -peliä, jossa on pelaajan tarkoituksena hallita pelihahmoista koostuvaa perhettä oikeaa elämää simuloivassa pelimaailmassa. Pelaaja hallitsee pelihahmojaan, mutta sen lisäksi jokaisella pelihahmolla on myös oma persoonallisuus. Tekoäly pystyy antamaan käytöspuustaan käskyjä pelaajan pelihahmolle, jos pelaaja ei itse ole antanut hahmolle käskyä. Pelihahmolle voi esimerkiksi pelissä tulla nälkä, minkä takia käytöspuun avulla hahmo voi itse päättää listatuista toiminnoista tilanteeseen ja hahmon persoonallisuuteen parhaiten sopivan vaihtoehdon. [16.]

Pelihahmot voivat myös keskustella toistensa kanssa. Tähän tarvitaan hahmojen väliseen viestitykseen tarkoitettu tekoäly, useammasta pelihahmosta koostuva tekoälytiimi tai jokin muu tapa tehdä yhteistyötä pelihahmojen välillä. Yksi helpoimmista tavoista tehdä yhteistyötä pelihahmojen välillä on toteuttaa vihollisten tekoälyyn tapa kertoa muille pelimaailmassa oleville vihollisille pelaajan paikka, jos pelaaja ja yksi vihollisista ovat taistelussa. Tällöin viholliset voivat tehdä yhteistyötä pelaajan kukistamiseksi. Myös pelaajaa auttavien hahmojen tekoäly voi tehdä pelaajan kanssa yhteistyötä. Nämä hahmot voivat esimerkiksi aina keskittyä hyökkäämään samaan kohteeseen, mihin pelaajan tiimi on hyökkäämässä. Hahmojen yhteistyössä tärkeässä osassa ovat myös pelijärjestelmät ja pelihahmojen hallintajärjestelmät, jotka sijaitsevat Meta AI:n puolella. [2.]

3.4 Pelijärjestelmien tekoäly

Pelijärjestelmiä hallitseva Meta AI tarkkailee pelin kulkua ja pyrkii dynaamisesti tekemään muutoksia pelissä. Vanhoissa peleissä Meta AI:n ainut tehtävä oli lähinnä vaikeustason säätäminen, minkä takia oli vaikeaa erottaa Meta AI ja Character AI toisistaan. Nykyään Meta AI keskittyy pelijärjestelmiin ja pelin kulkuun. Molemmat saavat tietoa pelin tilanteesta samalla tavalla. Navigation AI antaa kummallekin tietoa pelimaailmasta, ja molemmat tekoälyt käsittelevät sitä omalla tavallaan.

Meta AI pystyy tekemään monipuolisesti erilaisia asioita. Yksi hyvä esimerkki on vihollisten luominen pelimaailmaan. Meta AI pystyy seuraamaan pelaajan liikettä Navigation AI:n antaman tiedon avulla, ja tekemään sen avulla päätöksiä, mihin vihollisia voidaan laittaa. Tämän tiedon avulla Meta AI voi tarkistaa, että pelaajalla ei ole näköyhteyttä kohtaan, johon vihollisia luodaan. Tätä tekniikkaa on käytetty esimerkiksi Digital Extremesin vuonna 2013 julkaisemassa pelissä Warframe. Peli tekee pelaajan tekemästä vaikutuksesta eli läsnäolosta tai peliliikkeistä ns. lämpökartan, jonka avulla peli osaa luoda vihollisia alueille, joissa pelaajan vaikutus on lisääntymässä. Nämä alueet yleensä sattuvat osumaan pelitason tehtävän tavoitteen ja pelaajan väliin. Tämän varmistamiseksi pelissä on myös tieto tavoitteen ja pelaajan välisestä etäisyydestä, jonka avulla Meta AI voi luoda enemmän vihollisia tasoon, kun pelaaja lähestyy tehtävän päämäärää. [17.]

Warframe on hyvä esimerkki myös siksi, että siinä on toinen suosittu Meta AI:n ominaisuus. Kaikki pelin tasot ovat Meta AI:n generoimia. Tekoäly luo peliin tasot valmiiksi tehdyistä ”rakennuspalikoista”. Tätä kutsutaan tason proseduraaliseksi luomiseksi. Taso koostuu aloituspalikasta, joka yhdistetään muihin pelin rakennuspalikoihin niissä olevien ovien avulla. Ovet yhdistävät palikat toisiinsa, ja Meta AI luo tason satunnaisesti valituista rakennuspalikoista. Ylimääräiset ovet suljetaan, jotta pelaaja ei voi pudota tasolta pois avaruuteen. [17.]

Meta AI voi myös hallita pelin vaikeustasoa. Se voi esimerkiksi antaa uusille luoduille vihollisille enemmän tilastollista voimaa (statistics, stats), kuten lisätä hyökkäyksiin lisävahinkoa tai vähentää pelaajan vihollisille tekemää vahinkoa. Vaikeustasoa voi myös nostaa luomalla enemmän vihollisia tai tekemällä pelin tehtävistä monimutkaisempia. Vaikeustasoa lasketaan tekemällä vihollisista heikompia, vähentämällä vihollisten määrää tai tekemällä pelaajasta tilastollisesti voimakkaampi.

3.5 Käyttöliittymä

Pelinkehityksessä otetaan huomioon käyttäjien kokemukset tehdyn pelin pelaamisesta. Pelin käyttöliittymän (user interface, UI) on oltava hyvä, jotta pelaaja saisi pelaamisesta mahdollisimman hyvän kokemuksen. Huono käyttöliittymä ei tarjoa pelaajalle tarpeeksi informaatiota pelistä tai vaihtoehtoja tehdä valintoja. Pelaaja on vuorovaikutuksessa pelin kanssa yleensä ainoastaan käyttöliittymän läpi, joten käyttöliittymän ulkonäkö ja toimivuus vaikuttavat keskeisesti siihen, mitä mieltä pelaaja on pelistä. Pelin käyttöliittymän suunnittelu voidaan jakaa seuraaviin osiin:

- pelaajan ohjaus (controls)
- HUD (heads-up display)
- valikot (menus)
- apuohjelmat (utilities).

Pelaajan ohjaus on näistä yksinkertaisin, sillä se tarkoittaa pelaajan valitseman ohjaustyylin toteutusta. Peleissä voi olla ohjaukset useammalle peliohjaimelle, joista yleisimmät ovat PlayStation- tai Xbox-peliohjain tai näppäimistö ja hiiri. Jokaisella peliohjaimen painikkeella pelihahmo voi tehdä hyökkäyksen tai liikkeen pelimaailmassa.

HUD, eli Heads-Up Display, tarkoittaa peliruudulla olevia elementtejä, jotka ovat vain pelaajan nähtävissä. Nämä peliobjektit eivät siis ole pelimaailmassa, vaan ne ovat omalla tasollaan pelin ja pelaajan välissä. Käyttöliittymän HUD:n suun-

nittelussa mietitään, halutaanko siitä tehdä mahdollisimman minimalistinen vai todella yksityiskohtainen. Minimalistisessa HUD:ssa halutaan mahdollisimman paljon informaatiota niin pieneen tilaan ruudulla, että se ei häiritse pelaajaa. Näissä peleissä HUD-elementit voivat kadota ruudulta parin sekunnin jälkeen. Esimerkiksi pelaajan elinvoimapisteet (health points, HP) voidaan piilottaa ruudulta ja laittaa ne takaisin näkyviin vain silloin, kun pelaaja vahingoittuu tai parantaa itsensä taikajuomalla. Tässä tapauksessa HUD-elementti tulee aina näkyviin vain, kun elinvoimapisteissä tapahtuu muutos. Peliä suunnitellessa on myös hyvä miettiä, kuinka paljon peli haluaa antaa pelaajalle tietoa hänen pelitilanteestaan. Joskus ei ole järkeä antaa pelaajalle tarkkaa numeerista tietoa esimerkiksi pelaajan elinvoimapisteistä. Pelaaja ei tee numeroarvolla mitään, kun hän keskittyy pelissä vaikeaan taisteluun. Tämän takia kuvan 5 tapaiseen nopeatempoiseen peliin sopivat paremmin minimalistisesti pelaajan tilasta ilmoittavat HUD-elementit, jotka katoavat pois ruudulta, kun pelaajan parametrien arvoissa ei tapahdu muutoksia.

Kuva 5: Pelaaja juoksemassa ja taistelemassa Elden Ring -pelissä [18].

Valikot ovat pelaajalle tärkeä osa peliä. Pelin päävalikko on yleensä ensimmäinen asia, jonka pelaaja näkee pelissä. Pelin valikoiden on tärkeää reagoida nopeasti pelaajan valintoihin ja siirtyä seuraavaan valikkoon sulavasti. Hidas tai huonosti reagoiva valikko on pelaajalle tuskallinen kokemus. Valikossa käytetyn tekstin on myös tärkeää olla selkeää ja helposti luettavissa. Valitun fontin koko

on myös oltava aina luettavissa, koska pelaaja voi pelata peliä tietokoneen ruudulta tai sohvan pohjalta olohuoneessa. Kehittäjien tulisi harkita ruudun koon mukaan skaalautuvan fontin toteuttamista. [19.]

Apuohjelmilla tarkoitetaan pelissä esiintyviä helpotuksia ja asetuksia, joiden avulla saadaan parannettua pelin saavutettavuutta. Näitä asetuksia ovat esimerkiksi edellä mainittu fontin koko, mahdollisuus piilottaa HUD-elementtejä tai vaihtaa eri ohjauspainikkeiden toimintoja pelaajalle luonnollisempiin painikkeisiin. Apuohjelmien luomisesta pelinkehittäjät tekevät laajaa yhteistyötä käyttäjäkokemustiimin (user experience, UX) kanssa. Saavutettavuuden parantamiseksi voidaan pelin valikkoihin lisätä erilaisia värisokeus- ja vaikeusasteasetuksia sekä muita pelaamista helpottavia asetuksia, kuten asetus tehdä pelaajan taisteluista automaattisia tai automatisoida pelaajahahmon kyky kerätä ympäristöstä erilaisia materiaaleja. [19.]

4 Pelin ja tekoälyn toteutus

Peliprojektin suunnittelu aloitettiin genren valinnalla. Genreksi valittiin 3D-roolipeli, eli 3D RPG. Näissä peleissä on paljon mahdollisuuksia sulavaan pelaajan ja vihollisen hallintaan, joten se oli yksi parhaista vaihtoehdoista insinöörityöhön. Grafiikkaa peliin ei alusta lähtien tarkemmin mietitty, sillä se ei ollut tärkeä osa peliä. Projektin ideana oli tehdä toimiva ja hyvä tekoäly peliin ja luoda tekoälyllä toimiva pelin hallintajärjestelmä. Pelistä oli tarkoitus tulla selaimella toimiva, joten ainut järkevä pelaajahahmon ohjausvaihtoehto oli hiiri ja näppäimistö. Tällä tavalla pelistä voitaisiin myös mahdollisesti tehdä mobiiliversio, jos näppäimistön painikkeista pystyttäisiin tekemään ruudulle omat ohjauspainikkeet.

Projektin alussa pelaajalle tehtiin pelihahmo. Pelin kamera liitettiin pelaajaan, jotta se pystyy seuraamaan pelaajaa. Tällä tavalla pelaaja pysyy koko ajan kameran ruudussa. Tätä kamerakulmaa peleissä kutsutaan kolmannen persoonan näkymäksi (third person view). Kamera laitettiin kääntymään nuolinäppäimillä ja zoomaamaan pelaajaan hiiren rullalla. Samalla suunniteltiin pelaajahahmon liik-

kumiskykyä. Pelaajan tulisi pystyä liikuttamaan hahmoaan napsauttamalla haluttua kohtaa pelimaailman maastosta. Tähän tarvitaan navigaatiotekoälyä, koska pelaajahahmon täytyy pystyä löytämään reitti pelihahmon ja pelaajan valitseman maaston pisteen välillä.

4.1 Hallintajärjestelmä

Peliin haluttiin tehdä mielenkiintoinen järjestelmä kaikkien pelihahmojen hallitsemiseen. Square Enixin tekoälymallia käytettiin apuna hallintajärjestelmän luomiseen. Paras lähtökohta tekoälyjärjestelmän rakentamiseen oli Navigation AI. Sen jälkeen peliin lisättiin Character AI ja lopuksi pelin järjestelmän hallintaan suunniteltu Meta AI. [1.]

4.1.1 Navigaatiotekoäly

Navigaatioon pelissä haluttiin alkuun käyttää A*-polunetsintää. Polunetsinnässä A*-algoritmi etsii lyhyimmän reitin kahden pisteen välillä jakamalla pelimaailman pieniin osiin, joita kutsutaan solmuiksi. A* käsittelee solmujen välimatkaa maaliin sekä aloituspisteeseen, ja tämän avulla se pystyy aina laskemaan lyhyimmän mahdollisen reitin jokaiselle solmulle kaavalla 1.

$$f(n)=g(n)+h(n) \quad (1)$$

jossa

- n on kyseessä oleva solmu
- g(n) on kustannus aloitussolmusta solmuun n
- h(n) on kustannus maalista solmuun n
- f(n) on arvio solmun n kokonaiskustannuksesta.

Tämä lisättiin peliin ilman suurempia ongelmia. Solmujen tekemästä verkosta tehtiin tarpeeksi pieni, jotta pelaaja ei huomaa polunetsinnän kuutiomaista rakennetta. Tämä kuitenkin saattaa aiheuttaa ongelmia suorituskykyyn, jos pelis-

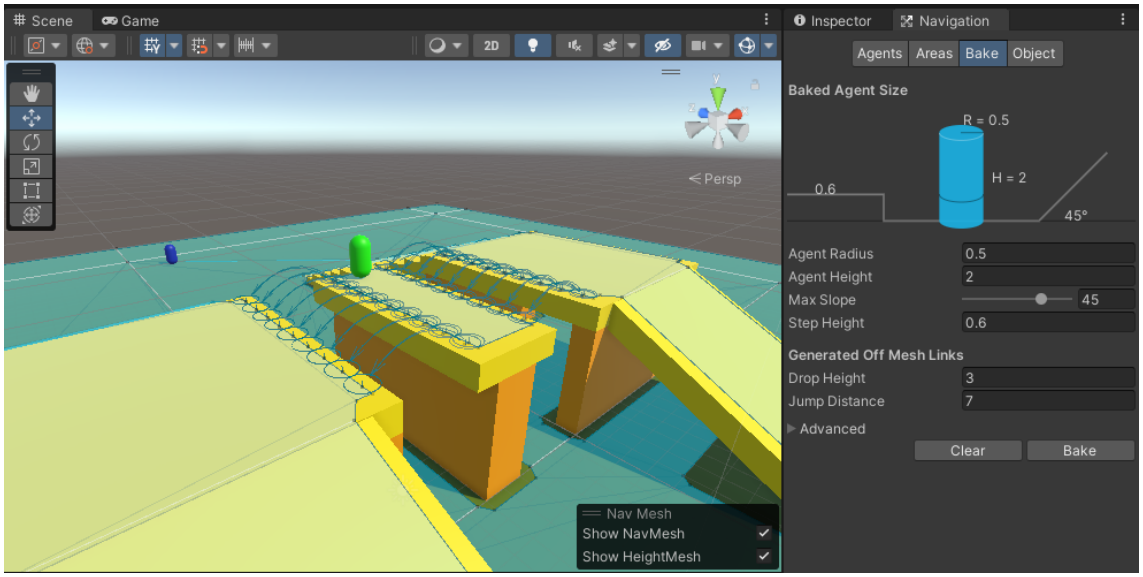
sä on liian paljon solmuja navigoitavaksi. Uuden polunetsintätekoälyn avulla pelaajahahmon liike saatiin toimimaan. Pelaaja liikkuu napauttamalla tiettyä kohtaa pelimaailman maastosta. Pelin kamera lähettää säteen (raycast) pelaajan hiiren osoittamaan suuntaan. Säde törmää solmuun, joka on pelaajan hiiren napsauttaman maaston kohdalla. Navigaatiotekoäly saa pelaajan painamasta solmusta tiedon, jonka avulla se laskee parhaan reitin pelaajan nykyisen solmun ja valitun solmun välillä. Sen jälkeen navigaatiotekoäly lähettää pelaajahahmolle käskyn liikkua sen laskemaa reittiä pitkin pelaajan valitsemaan kohteeseen. [20.]

Tässä vaiheessa huomattiin, että A*-polunetsintä ei ollut välttämättä paras mahdollinen navigaatiojärjestelmä peliin. Suureksi ongelmaksi osoittautui algoritmin huono suorituskyky, kun siihen yritettiin lisätä kolmatta ulottuvuutta. Solmuihin pohjautuva A*-polunetsintä päädyttiin korvaamaan Unityn tarjoamalla NavMesh-teknologialla. Unityn NavMesh toimii vokseleilla. Vokselit ovat pieniä kuutioita, jotka toimivat pikselien kolmiulotteisena vastineena. Nämä pienet kuutiot muodostavat matriiseja, jotka vastaavat jotain kohtaa navigaatioverkossa. Käytännössä A*-solmut toimivat samalla tavalla, mutta ne ovat huomattavasti isompia ja niiden paikat ovat sidottuna koordinaatistoon (x, y, z). Vokseleita sen sijaan voi olla enemmän pienellä alueella ja ne voidaan yhdistää peliobjekteihin, joten ne eivät ole yhteydessä pelin koordinaatistoon.

Unityn navigaatioverkko voidaan lisätä (bake) pelin maastoon ja peliobjekteihin editorissa, mutta Unityn navigaatiojärjestelmässä on myös mahdollista lisätä NavMesh peliin pelaamisen aikana. Tätä voi käyttää hyödyksi silloin, kun haluaa tehdä dynaamisesti muuttuvia pelitasoja. Unityn editorissa pelihahmoihin lisättiin NavMesh Agent (navigaatioverkkoagentti) -komponentit, joiden avulla pystyttiin liikuttamaan hahmoja NavMesh-alustoilla. Näihin NavMesh-komponentteihin pystytään myös merkitsemään OffMeshLink (verkon ulkopuolinen linkki). Nämä linkit ovat kohtia, joissa pelihahmo pystyy hyppäämään linkin toisesta päästä toiseen tai yhdistämään kaksi eri NavMesh- verkkoa toisiinsa. Pelimaailmassa oleviin peliobjekteihin voidaan laittaa NavMesh Obstacle (este) -

komponentteja, joiden avulla agentit osaavat kiertää ja välttää peliobjekteja. [21.]

Kuvasta 6 voidaan nähdä vaalean sinisellä agenteille sallitut alueet. Pilarien jalat ovat esteitä, joten niiden ympärille muodostuu pieni alue, johon agentit eivät voi liikkua. Samalla editorissa voidaan määrittää agenttien koko. Tämän koon ja korkeuden perusteella NavMesh osaa laskea tarpeeksi tilaa agentin ja seinien väliin, niin että agentti ei törmää niihin.



Kuva 6: Navigaatioverkon lisääminen kenttään Unityn editorissa [22].

Agentin korkeus vaikuttaa myös navigaatioverkkoon. Rampin alla olevasta tilasta agentti pääsee liikkumaan läpi silloin, kun maaston ja rampin välinen korkeus on vähintään yhtä korkea kuin itse agentti. Muuten agentti saattaisi yrittää mennä rampin läpi.

4.1.2 Pelihahmojen tekoäly

Seuraavaksi lisättiin pelihahmoille tekoälyä. Ensimmäisenä kehitettiin pelaajan liike toimivaksi. Aluksi pelihahmoja liikutettiin aina kohti A*-navigaation antamaa kohdesolmua. Navigaatio antoi hahmolle kohdesolmuja järjestyksessä jonossa, jotta pelihahmo ei pysähtyisi odottamaan seuraavaa solmua ja liike olisi sulavaa

solmusta toiseen. Pelissä päädyttiin kuitenkin vaihtamaan solmuista riippuvaisesta A*-navigointijärjestelmästä Unityn vokselipohjaiseen NavMesh-tekoälyjärjestelmään.

Uudella NavMesh-järjestelmällä pelihahmojen liikuttaminen helpottui huomattavasti. Aikaisemmassa järjestelmässä käytettiin hahmojen liikuttamiseen Unityn fysiikkamoottoria, mutta uusi järjestelmä osaa tehdä hahmojen liikuttamisen suoraan NavMesh-agenttien avulla. Agentit tarvitsevat liikkumiseen vain halutun pisteen paikan. Agentin asetuksia voidaan muokata editorissa tai koodissa. Jokaiselle hahmolle voidaan esimerkiksi antaa oma nopeus, pysähdysmatka ja kiihtyvyys. Liikkeeseen voidaan käyttää samaa koodia kaikissa pelihahmoissa esimerkkikoodin 1 osoittamalla tavalla.

```
void MoveCharacter(Vector3 _target)
{
     if (_target != null)
     {
          Agent.SetDestination(_target);
     }
}
```

Esimerkkikoodi 1: C#-funktio, joka asettaa NavMeshAgent-luokkaisen Agent-olion päämääräksi halutun pisteen _target. Unityn valmis NavMeshAgent-luokka osaa liikuttaa olion peliobjektin _target-pisteeseen navigaatioverkolla olevia esteitä väistellen.

Hahmojen liikkeen jälkeen seuraavaksi vuorossa olivat vuorovaikutusjärjestelmä ja taistelujärjestelmä sekä pelihahmojen attribuutit (stats). Jokaiselle pelihahmolle tehdään uusi tekoälyluokka, joka perii Interactable-luokan. Interactable-luokalle annettiin ylikirjoitettava funktio Interact, jonka avulla voidaan määrittää, mitä pelaaja tekee peliobjektille, jota pelaaja hiirellä ruudulta painaa. Jos pelaaja valitsee vihollishahmon, Interact-funktio käskee pelaajaa hyökkäämään valitun vihollisen kimppuun.

Taistelujärjestelmä laskee pelaajan tekemän vahingon suuruuden ja aiheuttaa sen perusteella vahinkoa viholliselle. Ennen taistelujärjestelmän tekemistä pelihahmot tarvitsevat attribuutteja, kuten niiden tekemän vahingon, vastaanotettua vahinkoa vähentävän panssarin ja hahmon elinvoimapisteet (HP). Näiden attri-

buuttien avulla luodaan perusluokka CharacterStats, jonka kaikki pelihahmot perivät omaksi luokakseen. Samalla luokkaan lisätään TakeDamage-, Heal- ja Die-funktiot, joiden avulla on helppoa tehdä pelissä parametreihin muutoksia. Kun HP-attribuutti on 0, Die-funktio tuhoaa pelihahmon pois pelistä.

Kunkin hahmon tekoälyluokan avulla saadaan helposti kaikki taistelussa tarvittavat arvot laskuja varten. Samalla taisteluun suunnitellaan yksinkertainen vahingon laskentakaava, jossa attacker.damage tarkoittaa hyökkääjän CharacterStats-luokasta löytyvää vahinkoparametria ja target.armor on hyökkäyksen kohteen armor-parametri (kaava 2).

$$damage = attacker.damage - target.armor \quad (2)$$

CharacterStats-luokan TakeDamage-funktion avulla voidaan laskettu vahinko tehdä kohteen HP-parametriin. Luokkaan voidaan myös määrittää muita parametrejä, kuten pelihahmon liikkumisnopeus, hahmon nimi tai hahmon hyökkäyksien välinen aika. Kaikille pelihahmoille merkittiin editorissa tägi, joka kertoo pelihahmosta. Hahmot merkittiin tunnisteilla Enemy (vihollinen), Player (pelaaja) ja Ally (liittolainen). Merkintöjä käytetään hahmon liittoutuman tunnistamiseen koodissa, jotta viholliset eivät hyökkää toisiaan vastaan ja liittolaiset tunnistavat pelaajan. Samalla annettiin vihollisille ja liittolaisille kohteen etsintään liittyvät aggressioetäisyys ja hyökkäysetäisyys. Pelihahmojen parametreihin lisättiin Unityn editorissa perusarvot, jotka sen jälkeen skaalattiin koodissa kasvamaan pelaajan tason perusteella.

Pelihahmojen päätöksentekotekoäly toteutettiin sääntöihin pohjautuvalla tekoälyllä. Säännöillä tarkoitetaan ehtolauseita (if-lause), joiden avulla saadaan tietty pätkä koodia toteutettua, kun ehtolauseessa haluttu totuusarvo on tosi tai epätosi. Hahmoille annettiin aluksi inGroup-totuusarvo, joka jakoi hahmon ryhmä- ja yksilötoiminnot ehtolauseella eri osiin. Hahmo käyttää yksilötoimintoja totuusarvon ollessa epätosi ja ryhmätoimintoja arvon ollessa tosi. Vihollisille ja liittolaisille luotiin säännöt, joiden avulla tekoäly pystyy päättelemään, onko pelihahmon aggressioetäisyydellä sille sopivaa kohdetta hyökkäykseen. Jos teko-

äly löytää kohteen, pelihahmo saa kohteesta paikkatiedon navigaatiotekoälyltä. Hahmoa liikutetaan kohdetta päin, kunnes kohde on pelihahmon hyökkäysetäisyydellä ja voi aloittaa hyökkäämisen kohteeseen. Ryhmässä olevan liittolaisen päätöksenteko tehtiin helposti tekoälyyn ehtolauseiden säännöillä:

- Jos pelaajalla ei ole kohdetta ja vihollisia ei ole aggressioetäisyydellä, seuraa pelaajaa.
- Jos pelaajalla on kohde ja kohde on hyökkäysetäisyydellä, hyökkää pelaajan kohteeseen.
- Jos pelaajalla on kohde ja kohde ei ole hyökkäysetäisyydellä, liiku pelaajan kohdetta päin hyökkäysetäisyydelle.
- Jos pelaajalla ei ole kohdetta ja hyökkäysetäisyydellä on kohde, hyökkää kohteeseen.
- Jos pelaajalla ei ole kohdetta ja aggressioetäisyydellä on kohde, liiku kohdetta päin hyökkäysetäisyydelle.

Sääntöjen lisäksi liittolaisille tehdään vielä ehtolauseet liittolaisten kykyjen käyttöä varten. Nämä ehtolauseet yhdistetään totuusarvoihin, joita pelaaja voi muuttaa pelin käyttöliittymän painikkeilla. Viholliset käyttävät päätöksentekoon samanlaista ehtolauseista muodostuvaa sääntörakennetta, mutta ryhmässä olevan hahmon pelaajan kohteet korvataan yhden vihollisryhmän jäsenen löytämällä kohteella, jos sellaisia on. Päätöksiin liittyvistä toiminnoista, kuten kohteen seuraamisesta, tehtiin omat funktiot, jotta ehtolauseiden koodista saadaan mahdollisimman siistiä.

Vihollisten hallintaan yritettiin kehittää koneoppivaa tekoälyä. Se onnistuu käyttämällä Unityn ML-Agents-pelinkehitystyökalupakettia. Koneoppivalle agentille annetaan koodi, jossa agentti säätää sille annettuja arvoja. Koodi antaa agentille vähän negatiivista palautetta, kun se vahingoittuu, ja paljon negatiivista palautetta sen kuollessa. Positiivista palautetta agentille annetaan, kun se tekee vahinkoa tai kukistaa pelaajan tai sen liittolaisen. Palautteen perusteella agentti yrittää maksimoida positiivisen palautteen määrää ja vähentää negatiivista palautetta. Agenttia pitää kouluttaa, mutta se kuitenkin oppi tehokkaasti löytämään pelaajan ja hyökkäämään pelaajan kimppuun. Kouluttaminen pelkällä pelaajalla ei ollut toimiva ratkaisu, joten kouluttamista varten luotiin taso, jossa oli pelkäs-

tään seiniä ja pelaajan liittolaisia. Tällä tavalla agentin koulutuksesta saatiin nopeampaa. [23.]

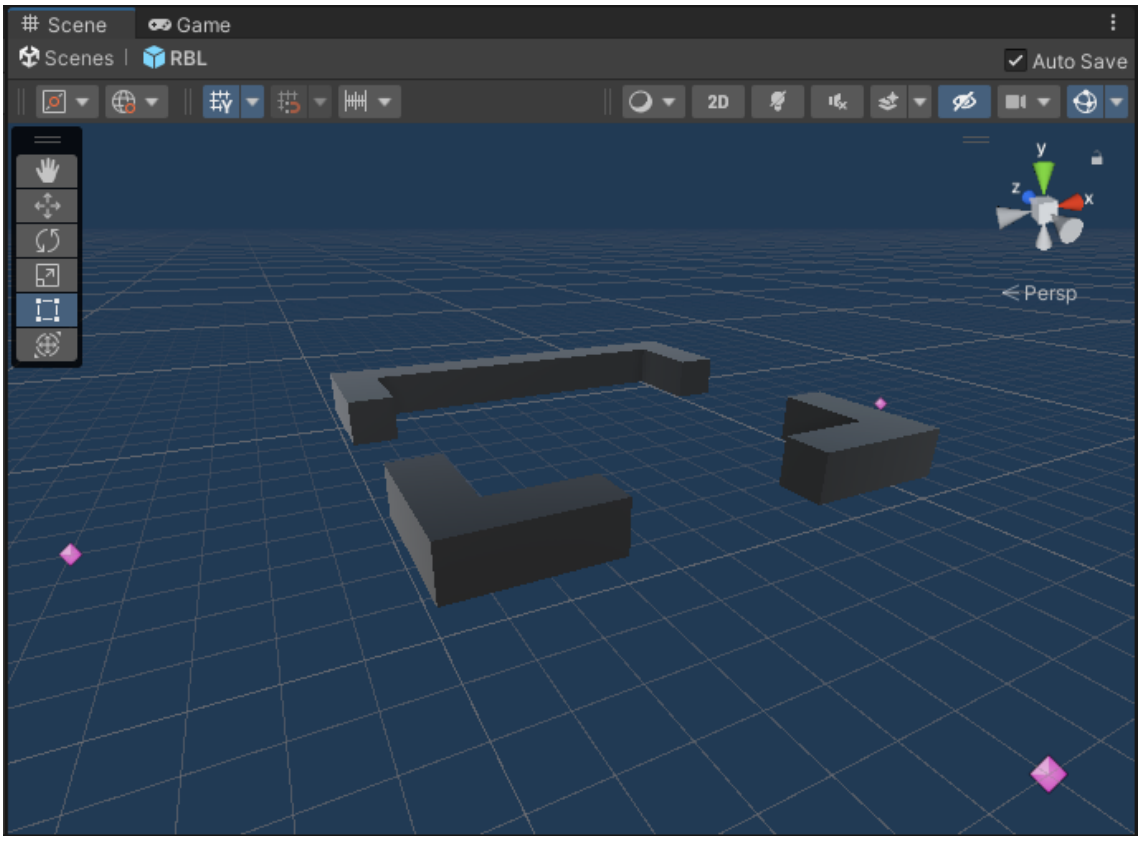
Ongelmat alkoivat, kun vihollisagentti keksi paeta pelaajalta yksittäisten hyökkäysten välissä. Tätä kutsutaan videopeleissä termillä *kiting*, joka tulee tekniikan tavasta johdattaa vihollista kuin vetämällä leijaa narusta. Yleisesti termi tarkoittaa vihollisen johdattamista ja samaan aikaan sitä kohti hyökkäämistä oman vahingon minimoimiseksi. Tämäntapaisesta vihollisesta on vaikea tehdä reilua. Vihollisen tekemää vahinkoa ja elinvoimapisteitä voidaan vähentää, mutta siitä huolimatta koneoppiva agentti ei muuttaisi omaa käyttäytymistään. Karkuun juoksevaa tekoälyä vastaan taisteleminen ei ole hauskaa. Joskus paras ratkaisu peliin on tyhmä yksinkertainen tekoäly, joka ei osaa monimutkaisia tekniikoita vahingoittumisen välttämiseen. Myös pelihahmojen voimakkuuksien tasapainon löytämiseksi on hyvä, että kaikki toimii yksinkertaisesti. ML-Agents ei tähän projektiin sopinut aktiivisen taistelujärjestelmän takia, mutta vuoropohjaiseen roolipeliin koneoppiva tekoäly voisi sopia paremmin.

4.1.3 Pelijärjestelmien tekoäly

Järjestelmien luominen aloitettiin tekemällä GameManagerin (pelin manageri), joka vastaa pelin kulusta ja auttaa tekoälyä löytämään järjestelmiä tai pelihahmoja koodissa. Sen jälkeen peliin lisättiin pelin tapahtumista vastaava EventManager (tapahtumamanageri). Managerin tehtävänä oli käynnistää pelissä tapahtuvat erikoistilanteet ja tapahtumat, kuten vihollisten yllätyshyökkäykset tai ystävällisten pelihahmojen antamat tehtävät. AllyTemplates-järjestelmä kehitettiin pelaajan liittolaisten lisäämisen helpottamiseksi ja aktiivisten liittolaisten hallitsemiseksi. Järjestelmässä on lista eri liittolaisten tyypeistä, joita pelissä ovat Rogue ja Paladin. Samalla järjestelmä pitää aktiivisten liittolaisten määrästä lukua, jotta järjestelmä ei anna pelin lisätä enempää liittolaisia peliin, kuin pelaajan seurueessa on tilaa.

Seuraavana peliin luotiin tyrmien kehittämiseen tarkoitettu DungeonGenerator. Jokaisen pelin tyrmän tulisi olla satunnaisesti luotu. Tätä varten luotiin satunnai-

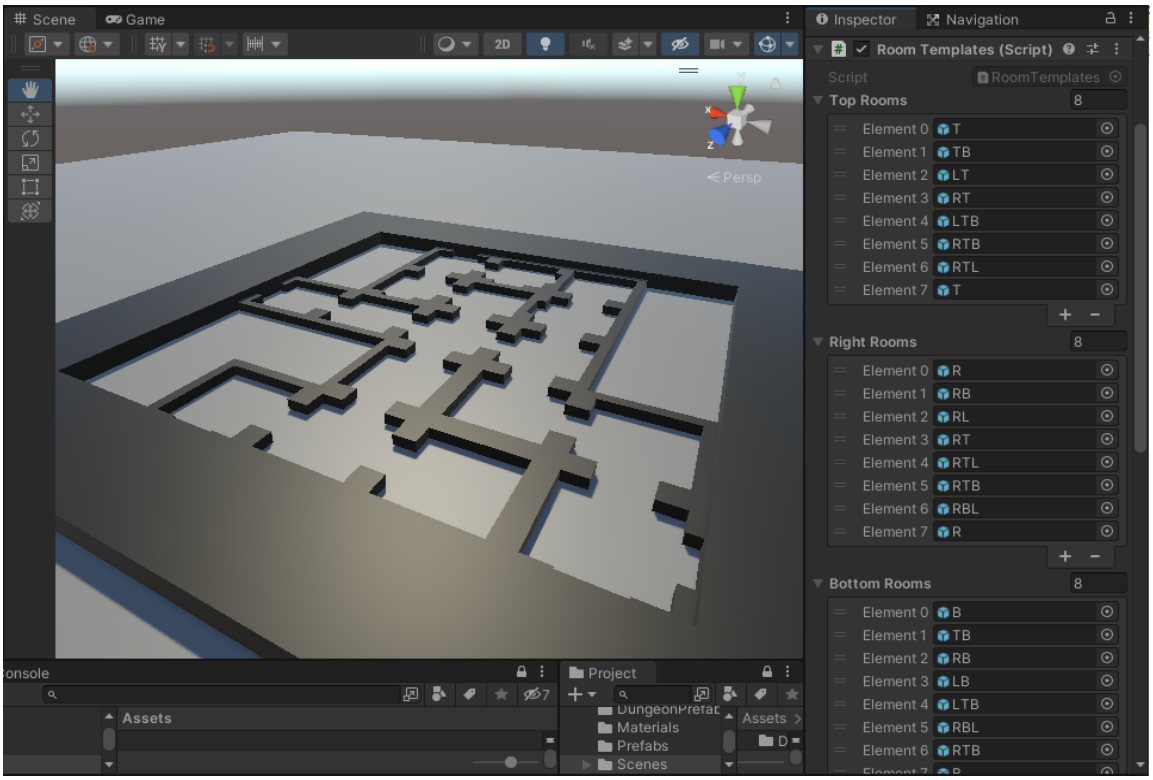
sista rakennuspalikoista koostuva tyrmä. Jokaisessa tyrmän palassa on satunnainen määrä ovia. Ovien kohtiin luodaan koodissa peliobjekteja, jotka valitsevat valmiiksi tehdyistä huonepalikoista siihen kohtaan sopivan palan. Kuvasta 7 nähdään, miten huoneessa on ovet vasemmalle, oikealle ja alaspäin. Pinkit pisteet kuvaavat näkymättömiä peliobjekteja, jotka ovat vastuussa uusien seuraavien palojen valinnasta.



Kuva 7: Esimerkki valmiiksi tehdystä tyrmän rakennuspalikasta [22].

Palasta saadaan varmasti oveen sopiva tekemällä RoomTemplates -järjestelmä (huonemallit), jossa jokainen valmiiksi tehty huonepalikka laitetaan palikan oviaukkojen suunnan perusteella listaan. Tämän jälkeen ovien kohdalla olevat peliobjektit valitsevat uuden palikan listasta, jossa sopivat palikat ovat listattu avoimen oven suunnan perusteella. Jos ovi on huoneen oikealla puolella, valitaan uusi huone RoomTemplates-järjestelmän listasta, jossa on kaikissa huoneissa ovi vasemmalle. Jokainen tyrmään lisätty uusi huone lisätään RoomTemplates-järjestelmästä löytyvään listaan. Listassa ovat kaikki tyrmän huoneet, joten sitä

voidaan käyttää tyrmän hallintaan. Kuvassa 8 on näkyvissä valmis kokonaan satunnaisesti luotu tyrmä. Kuvan oikeassa laidassa näkyvät RoomTemplates-järjestelmästä löytyvät ovien suunnan perusteella tehdyt listat. Top Rooms (yläosahuoneet) esimerkiksi on lista kaikista huoneista, joissa aukinaisen oven suunta on ylöspäin. Listoissa on kaksi umpikujaan johtavaa huonetta, jotta tyrmän koko pysyisi pienempänä satunnaisesta valinnasta riippumatta.



Kuva 8: Valmis tyrmä Unityn editorissa [22].

Tyrmän rakentamisen jälkeen peliin tarvitaan vihollisia. Peliin luotiin EnemyTemplates-järjestelmä, jossa listataan kaikki vihollistyypit. Erilaisia vihollistyyppejä pelissä ovat huoneesta toiseen matkustava vihollinen, yhdessä huoneessa satunnaisesti liikkuva vihollinen sekä edellisistä satunnaisesti liikkuvista vihollisista koostuva vihollisryhmä, joka hyökkää saman kohteen kimppuun. Järjestelmä käyttää aikaisemmin RoomTemplates-järjestelmässä tehtyä listaa tyrmän huoneista. EnemyTemplates-järjestelmä tekee huoneisiin vihollistyyppien listasta satunnaisesti valiten vihollisia, kunnes vihollisia on tyrmässä niiden määrää rajoittavan arvon verran. Järjestelmä myös lisää kaikki tyrmässä olevat viholliset

yhdeksi vihollislistaksi. Vihollisia ei kuitenkaan ole tarkoitus aina olla jokaisessa huoneessa, joten huoneisiin lisättiin satunnainen mahdollisuus, ettei peli tee kyseiseen huoneeseen vihollista. Vihollisten määrää rajoittava muuttuja lisättiin pelitilannetta hallitsevaan GameManager-järjestelmään, ja siitä tehtiin pelaajan tason mukaan skaalautuva. Vihollisten luomisesta tehtiin oma funktio, jota voidaan kutsua EventManager-järjestelmästä. Tapahtumien hallintajärjestelmä saa navigaatiotekoälyltä tyrmässä olevien vihollisten määrän. Kun vihollisia ei tyrmästä enää löydy, järjestelmä voi aloittaa uusien vihollisten luomisen. Kun kaikki viholliset on kukistettu toista kertaa, pelissä siirrytään tyrmän seuraavaan kerrokseen ruudulle ilmestyvällä painikkeella. Painike aloittaa tyrmän seuraavan kerroksen satunnaisen luomisen uudelleen alusta, minkä takia tyrmän kaikissa kerroksissa on aina erilainen pohjapiirros.

Peliin luotiin pelaajan ja pelimanagerin tietojen tallentamiseen SavedStats-järjestelmä. Järjestelmä säilyttää pelaajan ja pelimanagerin parametrit tyrmien kerroksista toiseen. Se tallentaa kaiken tarvittavat tiedon omiin parametreihinsa ja antaa ne uusille pelaaja- ja pelimanageri-peliobjekteille tyrmän seuraavassa kerroksessa.

4.2 Pelin käyttöliittymä

Peliin tarvitaan käyttöliittymä pelin ohjaamista varten sekä HUD-elementtejä antamaan pelaajalle informaatiota pelin tilanteesta. Pelin käyttöliittymää varten luotiin pelinhallintajärjestelmiin dialogijärjestelmä DialogueManager. Hahmojen repliikit tallennetaan Dialogue-luokkaan, joka sisältää listan yksittäisistä repliikkilinjoista (string). Hahmot saadaan puhumaan DialogueTrigger-luokan avulla. Luokka sisältää yhden Dialogue-luokan olion, puhujan nimen ja funktion, joka etsii ja käynnistää DialogueManager-järjestelmän StartDialogue-funktion. Dialogimanagerin funktio puolestaan paljastaa piilotetun dialogilaatikon ruudulle ja alkaa kirjoittamaan DialogueTrigger-luokan antamia repliikkejä laatikon tekstiosaan. Pelihahmoille voidaan antaa monta eri DialogueTrigger-luokkaa, jotta

hahmon repliikit voidaan helposti vaihtaa jonkin pelin tapahtuman tai tehtävän jälkeen.

Seuraavaksi projektissa tehtiin pelin HUD-elementit. Jokaiselle pelihahmolle tehtiin pieni elinvoimapisteiden määrää kuvaava palkki. Palkki on vihreä ja täynnä, kun hahmolla on 100 % elinvoimapisteistä jäljellä. Palkki täyttyy oikealta päin punaisella, kun pelihahmo menettää elinvoimapisteitä. Palkki laitetaan myös katoamaan ruudulta lyhyen ajan jälkeen, jotta ruutu ei täyttyisi pelihahmojen elinvoimapistepalkeilla. Pelihahmojen palkkien lisäksi pelaajan seurueen tiedoista tehtiin HUD-elementti. Elementissä on pelihahmojen nimet ja uusi HP-palkki, joka ei katoa ruudulta. Tämä on pelissä sen takia, että pelaajan olisi helpompi seurata omaa ja liittolaistensa tilannetta. Seurueesta kertova elementti on paljon isompi, joten sitä on helpompi lukea ja tulkita.

Pelaaja pystyy hallitsemaan omaa hahmoaan hiiren painikkeilla, mutta pelistä puuttui tapa ohjata pelaajan liittolaisia. Pelin käyttöliittymään lisättiin painikkeita pelaajan kykyjen käyttämiseen ja pelihahmojen hallitsemiseen. Pelaajan liittolaisten tekoälyyn lisättiin säännöt painikkeiden käskyistä, jotka olivat seuraavat:

- Hyökkää vihollisia päin.
- Käytä pelihahmon kykyä.
- Seuraa pelaajaa.
- Peru aiempi käsky.

Toimintojen lisääminen oli helppoa, sillä sääntöihin pohjautuvaan hahmon tekoälyyn on yksinkertaista lisätä uusia sääntöjä. Alkuun painikkeet käskivät kaikkia pelaajan liittolaisia tekemään painikkeen antaman komennon, mutta se ei ollut tarpeeksi tarkka tapa liittolaisten hallintaan, sillä projektiin haluttiin tehdä mahdollisuus lisätä uusia käskyjä myöhemmin. Peliin tarvittiin uusi pelijärjestelmä, jonka tehtävänä oli antaa käskyt vain valitulle liittolaiselle. Projektissa päädyin käyttämään hiiren vasenta painiketta liikkumiseen ja pelaajan liittolaisen valintaan. Hiiren oikea painike on silti varattu pelkästään vuorovaikutukseen eli puhumiseen ja hyökkäämiseen.

Pelissä olevat käskyille tehdyt HUD-elementit ovat epäaktiivisia, mutta ne muuttuvat aktiivisiksi pelaajan painaessa yhtä liittolaisista. Tämä osoittaa pelaajalle, että peli on valinnut pelaajan valitseman kohteen valmiiksi käskyjen vastaanottamiselle. Kun pelaaja painaa vasemmalla painikkeella jotain kohtaa maastosta, pelaajahahmo liikkuu valittuun kohtaan ja peli nollaa liittolaisen valinnan. Käskyjen elementit muuttuvat epäaktiivisiksi ja odottavat uuden liittolaisen valintaa.

Pelaajalle annettiin myös kykyjä vihollisten hallitsemiseen. Kykyjen painikkeet menevät epäaktiivisiksi, kun kykyä käytetään ja se menee jäähtymistilaan (cool-down) odottamaan kyvylle asetetun käyttökertojen välisen jäähtymisajan verran. Pelaajan kykyjen tehtävät pelissä ovat seuraavat:

- pakottaa kaikki viholliset hyökkäämään kohti pelaajaa (Taunt All)
- pelottaa kaikki viholliset juoksemaan karkuun (Fear All)
- palauttaa elinvoimapisteitä tehdyn vahingon verran (Steal Health)
- siirtää pelaajan takaisin pelitason aloituskohtaan (Return).

Kykyjen funktioiden koodit lisättiin pelaajan tekoälyyn, ja ne kytkettiin käyttöliittymän painikkeisiin. Vihollisille lisättiin pakotettu hyökkäys- ja pelkotila. Pelaajan painaessa painikkeita vihollisten tekoäly menee kyvyn keston ajaksi tilaan, jossa se juoksee karkuun tai ryntää hyökkäämään pelaajaa päin. Kyvyn keston loputtua viholliset palautuvat takaisin normaaliin sääntöjä noudattavaan tekoälymalliin.

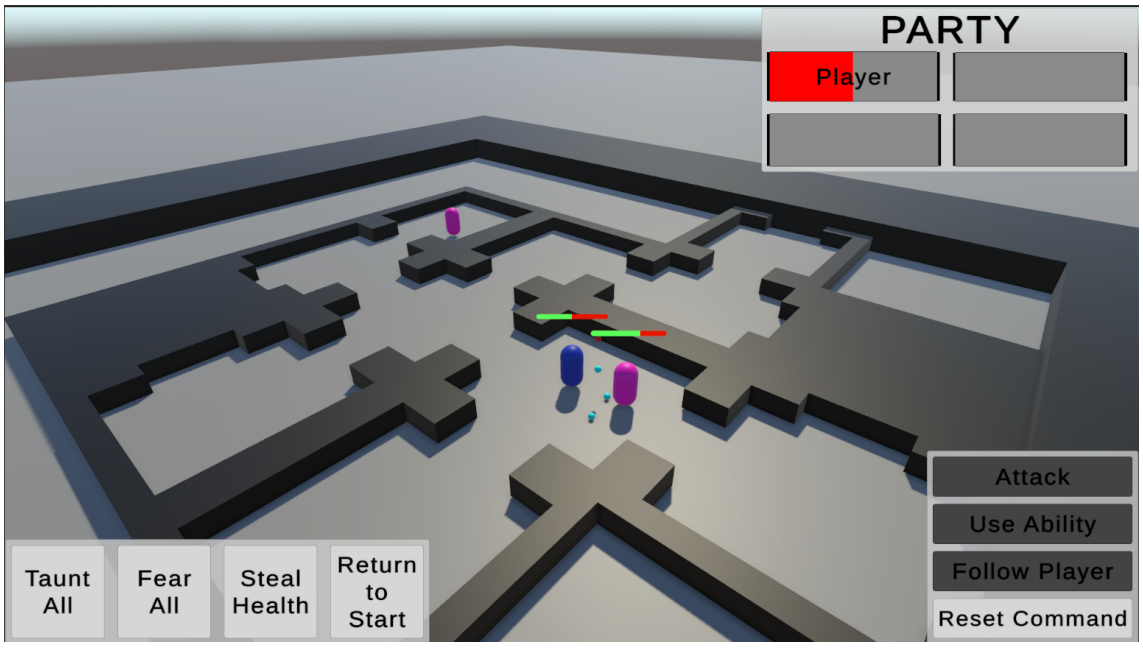
4.3 Pelinkehityksen haasteita

Insinöörityöprojektin alkupuolella ensimmäisiä haasteita olivat pelitasossa navigointi ja pelihahmojen tekoälyn suunnittelu. Solmupohjaisessa A*-järjestelmässä ja navigointiverkossa oli molemmissa sekä hyviä että huonoja puolia. A*-navigaatio oli yksinkertainen, mutta se muuttui todella raskaaksi suoritusteholle, kun solmujen kokoa pienennettiin, jotta navigoinnista ja liikkeestä saatiin tarkempaa. A*:n suurin ongelma oli korkeussuuntainen liike, johon alkuperäinen algoritmi ei edes pystynyt. A* on selkeästi tarkoitettu lähinnä 2D-peleihin.

Unityn navigaatioverkko sen sijaan toimii paremmin insinöörityön peliprojektin kaltaisissa 3D-peleissä. Verkoilla pystyttiin liikkumaan korkeussuunnassa ilman ongelmia. Unityn navigaatioverkkojen heikkous on niiden jatkuva kehitys, koska komponentit eivät ole vielä täysin valmiita ja virheettömiä. Teknologia on vielä melko uusi, joten siinä on vielä vikoja ja ohjelmointivirheitä. Yksi havaittu ongelma on, että ylämäkeen liikkuvat objektit saattavat yrittää mutkitella suoraa ramppia ylös, jotta nousun kulma olisi pienempi ja nousun nopeus pysyisi mahdollisimman samana. Tämä ei kuitenkaan aina toimi, joten agentti saattaa hidastella ja satunnaisesti vähän kääntyä mäkeä suoraan ylös noustessa. Projektin loppupuolella suurimmaksi haasteeksi osoittautui ideoiden ja uusien järjestelmien luominen peliin. Aikaa prototyyppiversion tekemiseen ei ollut niin paljon, että olisi ehditty lisätä kaikkia ideoita peliin.

5 Valmis hallintajärjestelmä

Insinöörityöprojektin tavoitteena oli tehdä peli, jossa käytetään Square Enixin tekoälymalleja pohjana järjestelmien kehittämiseen. Pelin prototyyppiversio (kuva 9) saatiin valmiiksi, ja tekoälyn hallintajärjestelmä saatiin toimimaan.



Kuva 9: Valmiin peliprototyypin satunnaisluotu tyrmätaso [22].

Valmis peliprototyyppi koostuu päävalikosta, josta pelaaja voi valita kahden eri tason välillä. Dungeon Generation vie pelaajan tasoon, jossa satunnaisesti luodaan tyrmä ja siihen viholliset. Pelaajan täytyy kukistaa kaikki viholliset, jotta seuraavat viholliset luodaan peliin. Kun pelaaja voittaa kaksi vihollisaaltoa, peli antaa pelaajan edetä seuraavaan kerrokseen, jossa pelaaja saa ystäväkseen yhden satunnaisen liittolaisen. Peli jatkuu, kunnes pelaajahahmo kuolee.

Toinen taso on RPG Sample Scene, jossa pelaaja voi kokeilla pelin dialogi- ja tapahtumajärjestelmiä. Pelaaja voi jutella vihreälle pelihahmolle, joka antaa pelaajalle tehtäväksi kukistaa kaikki tason viholliset. Vihollisten kukistamisen jälkeen vihreän pelihahmon seuraava dialogi käynnistää vihollisten vastahyökkäyksen, jonka jälkeen pelaajan on uudestaan voitettava uudet peliin luodut viholliset. Tämä testitaso loppuu toisten vihollisten kukistamiseen tai pelaajahahmon kuolemaan.

5.1 Pelin tekoälyjärjestelmien arviointi

Insinöörityön tarkoituksena oli tehdä tietokoneohjattujen pelihahmojen hallintajärjestelmä, jonka ympärille pystyttäisiin rakentamaan peli. Pelistä tuli prototyyppi, jota pystytään tulevaisuudessa laajentamaan RPG-peliksi. Pelin tekoälyjärjestelmät tehtiin Square Enixin malleja seuraten. Mallit perustuvat Navigation AI-, Meta AI- ja Character AI -osiin.

Prototyypissä navigoinnista vastaa Unityn navigaatioverkkoihin perustuva NavMesh-menetelmä. Tekoälymallista peli kuitenkin poikkeaa vähän, koska kaikki pelikentästä saatava tieto ei kulje välttämättä navigaatiotekoälyn kautta, vaan menee suoraan pelihahmolta toiselle. Unityn pelimoottorissa olevat valmiit perusjärjestelmät eivät varsinaisesti ole osa navigaation tekoälyä, mutta ne käytännössä toimivat samalla tavalla kuin Square Enixin malleissa. Esimerkiksi fysiikkamoottorin avulla pelihahmot pystyvät löytämään lähellä olevat viholliset. Tässä tapauksessa Unityn fysiikkamoottori toimii viestittäjänä pelimaailman ja pelaajahahmon välillä.

Järjestelmistä vastaava GameManager ja sen alajärjestelmät toimivat projektissa hyvin. Pelin managerissa on määritelty paljon eri mahdollisia muuttujia, joiden avulla voidaan helposti säätää esimerkiksi tyrmän määritettyä maksimikokoa tai pelin tyrmään tekemien vihollisten lukumäärää. Uusia liittolais- ja vihollistyyppejä on järjestelmään helppo lisätä Template-luokkien avulla. Tarvitsee vain tehdä uusi vihollistyyppi ja vetää se editorissa EnemyTemplates-järjestelmän vihollistyyppilistaan. Myös uusia ja parannettuja huoneita voi järjestelmään lisätä helposti. Uuteen huoneeseen täytyy laittaa ovien kohdalle uuden huoneen luova peliobjekti ja vetää editorissa huone kaikkiin oikeisiin listoihin, joita huoneen oviaukkojen suunnat vastaavat. Tyrmän luonti ja vihollisten lisäysjärjestelmät osaavat käyttää uusia vihollisia ja huoneita ilman muutoksia niiden koodiin. Square Enixin tekoälymallien mukaan Meta AI saa tietoa pelimaailmasta Navigation AI -järjestelmältä. Tätä ei pelissä paljon tarvita, koska pelijärjestelmät osaavat itse tehdä peliobjekteista viittaukset, koska pelijärjestelmät luovat pelin tason DungeonGenerator-järjestelmällä sekä viholliset siihen. Vihollisia luodessa järjestelmä myös tallentaa jokaisesta luodusta vihollisesta hallintajärjestelmään viitteen, jonka avulla niitä voidaan pelissä hallita tekoälyn päättämällä tavalla.

Pelihahmoja hallitsevat tekoälyt toteutettiin peliin yksinkertaisesti, jotta niihin olisi helppo myöhemmin lisätä uusia ominaisuuksia tai muokata niiden toimintaa kokonaan. Yksinkertaisen sääntöihin perustuvan tekoälyn takia uusien kykyjen ja sääntöjen lisääminen on helppoa. Pelihahmot osaavat toimia itsenäisesti pelimaailmassa, mutta tarvittaessa pelaaja pystyy antamaan käskyjä liittolaisilleen tai herättämään vihollisten huomion ja pakottamaan niitä hyökkäykseen. Hahmojen tekoälyn lopputulos on hyvä, koska se toimii sujuvasti ilman ongelmia ja siihen on helppo lisätä uusia osia.

5.2 Pelin tulevaisuus

Suurin insinöörityössä tehdyssä pelissä oleva ongelma ovat grafiikat. Pelin prototyyppiversio käyttää Unityn valmiita kapseleita pelihahmoille ja kuutioita pelitason suunnittelussa. Jos pelinkehityksessä olisi ollut enemmän aikaa ja resurs-

seja käytettävissä, peliin voitaisiin lisätä vaikka ilmaista valmiiksi tehtyä grafiikkaa Unityn Asset Store -verkkokaupasta parantamaan pelin ulkoasua. Samalla voitaisiin myös vaihtaa HUD-elementteihin modernimmat tyylitellyt ulkoasut.

Hallintajärjestelmissä ei ole suuria ongelmia, mutta yksi idea olisi parantaa pelin tylsää taistelujärjestelmää. Pelistä poistettaisiin nykyinen yksinkertaisilla perushyökkäyksillä ja kyvyillä toimiva aktiivinen järjestelmä. Suunnitelmana on luoda roolipeleissä yleisesti käytetty vuoropohjainen taistelujärjestelmä. Toinen muutos voisi olla pelaajahahmon ja liittolaisten ohjausten muuttaminen. Peli ei keskittyisi enää pelaajaan, vaan kameraa voisi liikuttaa hiirellä ympäri pelitasoa. Kaikista liittolaisista tehtäisiin pelaajahahmoja, ja pelaajalle annettaisiin kyky hallita pelihahmojen ryhmää. Tämä muuttaisi peliä muistuttamaan enemmän huippusuosiota saaneen Baldur’s Gate 3 -pelin järjestelmiä. [24.]

Projektin tekeminen oli opettavainen prosessi, mutta insinöörityön jälkeen tämän projektin tekeminen jää hetkeksi tauolle, jotta uusiin ideoihin saadaan mietittyä järkevät toteutustavat. Projektin tekemiseen voidaan palata myöhemmin, kun Unityn omat järjestelmät ovat kehittyneet ja suurimmat ongelmat navigointiverkoista on korjattu.

6 Yhteenveto

Insinöörityössä tehtiin tietokoneohjattujen pelihahmojen hallintajärjestelmä, jonka ympärille rakennettiin peli Unityn pelimoottoria käyttäen. Työn tarkoitus oli käyttää Square Enixin tekoälymallia ja samalla ottaa selvää, onko malli mahdollista toteuttaa Unityssä.

Square Enixin tekoälyjärjestelmä perustuu kolmesta eri osasta koostuvaan järjestelmään. Osia ovat navigointitekoäly, järjestelmätekoäly ja hahmotekoäly. Tekoälymallin avulla tehty hallintajärjestelmä toimii pelissä hyvin.

Projektissa tehdyssä peliprototyypissä navigoinnista vastaa Unityn navigaatioverkkoihin perustuva NavMesh-menetelmä. Tekoälymallista peli kuitenkin poikkeaa vähän. Unityn navigaatioverkko toimii hyvin insinöörityön peliprojektin kaltaisissa 3D-peleissä. Unityn valmiiden navigaatioverkkojen heikkous on niiden jatkuva kehitys, koska komponentit eivät ole vielä täysin valmiita ja virheettömiä. Teknologia on vielä melko uusi, joten siinä on vielä vikoja ja virheitä.

Järjestelmistä vastaava GameManager ja sen alajärjestelmät toimivat projektissa hyvin. Pelin managerissa on määritelty paljon eri mahdollisia muuttujia asioiden lennosta muokkaamiseen ja hienosäätämiseen. Järjestelmät toimivat ongelmitta, ja niitä on tulevaisuudessa helppo laajentaa.

Pelihahmot osaavat toimia itsenäisesti pelimaailmassa, mutta ne toteutettiin peliin yksinkertaisesti, jotta niihin olisi myöhemmin helppo lisätä toiminnallisuuksia.

Insinöörityön lopputavoitteena oli saada pelistä toimiva prototyyppiversio, jossa tekoälyn hallintajärjestelmä toimii ilman suurempia peliä rikkovia ongelmia. Tavoite toteutui: pelin prototyyppiversio ja hallintajärjestelmä saatiin valmiiksi ja toimimaan.

Lähteet

1 Miyake, Youichiro. 2015. Square Enix. Current Status of Applying Artifial Intelligence in Digital Games. Journal of the Japanese Society for Artificial Intelligence. Vol. 30, s. 45-64.

2 SQUARE ENIX AI ACADEMY: A Seminar Series for the Introduction of digital game AI. 2015. Yrityksen sisäinen aineisto. Square Enix.

3 Paju, Petri. 2003. Huvia hyödyn avuksi jo 1950-luvulla. Verkkoaineisto. Wider Screen. <http://widerscreen.fi/2003/2-3/huvia_hyodyn_vuoksi_jo_1950-luvulla.htm>. Luettu 10.7.2023.

4 Aapeli (peli). Verkkoaineisto. Wikipedia. <https://fi.wikipedia.org/wiki/Aapeli_(peli)> Luettu 31.10.2023.

5 Space Invaders 1978 – Arcade Gameplay. Verkkoaineisto. Game Archive – No Commentary Gameplay. <https://www.youtube.com/watch?v=MU4psw3ccUI>. Katsottu 20.9.2023.

6 Winstanley, Cam. The Making of Dune II. The birth of the real-time strategy game. Verkkoaineisto. Read-Only Memory. <https://readonlymemory.vg/the-making-of-dune-ii/>. Luettu 20.9.2023.

7 Lidén, Lars. 2017. Artificial stupidity: The art of intentional mistakes. Verkkoaineisto. <https://sudonull.com/post/68442-Artificial-Stupidity-The-Art-of-Intentional-Mistakes>. 29.7.2017. Luettu 10.7.2023

8 Half-Life. Versio 1.0. 1998. Valve.

9 Servat, Ludovic. 2020. Key features of Stealth games. Verkkoaineisto. Game Developer. <https://www.gamedeveloper.com/design/key-features-of-stealth-games>. 26.8.2020. Luettu 2.8.2023.

10 Kutchera, Ant. 2018. The best opening move in a game of tic-tac-toe. Verkkoaineisto. Maxant. <https://www.maxant.ch/2018/04/07/1523086680000/>. 4.7.2018. Luettu 9.10.2023.

11 Bhashkar, Kunal. 2019. Build your own Chat bot (text Pre-processing). Verkkoaineisto. Medium. <https://bhashkarkunal.medium.com/conversational-ai-chatbot-using-deep-learning-how-bi-directional-lstm-machine-reading-38dc5cf5a5a3>. 13.1.2019. Luettu 31.10.2023

12 Rubio, Fatima. 2023. The 5 Most Powerful Pathfinding Algorithms. Verkkoaineisto. Graphable. <https://www.graphable.ai/blog/pathfinding-algorithms/>. 6.6.2023. Luettu 10.9.2023.

13 Gregory, Jason, Q. 2009. State-Based Scripting in Uncharted: Drake's Fortune and Uncharted 2: Among Thieves. Verkkoaineisto. GDC. <https://www.gdcvault.com/play/1730/State-Based-Scripting-in-UNCHARTED>. Luettu 16.10.2023.

14 About Square Enix Group. Verkkoaineisto. Square Enix. <https://www.hd.square-enix.com/eng/company/>. Luettu 30.10.2023

15 Miyake, Youichiro. 2013. The Fundamental Theory of Digital Game AI. Journal of Virtual Reality Society of Japan. Vol. 18, No.3, s28-33.

16 Evans, Richard. 2010. Modeling Individual Personalities in The Sims 3. Verkkoaineisto. GDC. <https://www.gdcvault.com/play/1012450/Modeling-Individual-Personalities-in-The>. Luettu 20.9.2023.

17 Brewer, Daniel. 2013. AI Postmortems: Assassin's Creed III, XCOM: Enemy Unknown, and Warframe. Verkkoaineisto. GDC. <https://www.gdcvault.com/play/1018058/AI-Postmortems-Assassin-s-Creedm>. Katsottu 1.10.2023.

18 Elden Ring. Versio 1.10. 2022. From Software.

19 Bohn, Zach. 2023. God of War Ragnarok: Building the UI for AAA Sequel. Verkkoaineisto. GDC. <https://www.gdcvault.com/play/1029143/-God-of-War-Ragnarok>. Katsottu 16.10.2023.

20 Belwariar, Rachit. A* Search Algorithm. Verkkoaineisto. GeeksforGeeks. <https://www.geeksforgeeks.org/a-search-algorithm/>. Luettu 20.10.2023.

21 Inner Workings of the Navigation System. Verkkoaineisto. Unity. <https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/NavInnerWorkings.html>. Luettu 10.7.2023.

22 Unity. Versio 2021.3.22f1. 2023. Unity Technologies.

23 Unity ML-Agents Toolkit Overview. Verkkoaineisto. Unity. <https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/>. Luettu 16.7.2023.

24 Pikkarainen, Joonas. Baldur's Gate 3 on viiden tähden merkkiteos – Lähes 200 pelitunnin jälkeen asiasta ei ole epäilystäkään. Verkkoaineisto. Muropa-

ketti. <https://muropaketti.com/pelit/peliarvostelut/arvostelu-baldurs-gate-3-on-viiden-tahden-merkkiteos-lahes-200-pelitunnin-jalkeen-asiasta-ei-ole-epailystakaan/>. Luettu 25.10.2023.