OPINNÄYTETYÖ
ANTTI KUKKONEN 2010

REITINHAKU JA TEKOÄLYN PÄÄTÖKSEN-TEKO KAKSIULOTTEISESSA VIDEOPELIS-SÄ

Rovaniemen ammattikorkeakoulu
University of Applied Sciences

OHJELMISTOTEKNIIKAN KOULUTUSOHJELMA

ROVANIEMEN AMMATTIKORKEAKOULU

TEKNIIKKA JA LIIKENNE

Ohjelmistotekniikan koulutusohjelma

Opinnäytetyö

REITINHAKU JA TEKOÄLYN PÄÄTÖKSENTEKO KAKSIULOTTEISESSA VIDEOPELISSÄ

Antti Kukkonen

2010

Ohjaaja Erkki Mattila

Hyväksytty _____ 2010 _____

Tekijä	Antti Kukkonen	Vuosi	2010
Työn nimi	Reitinhaku ja tekoälyn päätöksenteko kaksiulotteisessa videopelissä		
Sivu- ja liitemäärä	52		

Opinnäytetyössä käsitellään tekoälyä ja reitinhakua sekä teoreettiselta että käytännön kannalta. Opinnäytetyöllä ei ole toimeksiantajaa, vaan se toteutettiin itsenäisenä tutkimus- ja sovelluskehitysprojektina.

Työn tavoitteena oli antaa lukijalle käsitys tekoälyn historiasta ja tekoälyn osuudesta videopeleissä sekä esitellä tapoja soveltaa reitinhakualgoritmeja ja tekoälyn päätöksentekoon soveltuvia menetelmiä kaksiulotteisessa videopelissä. Tämän lisäksi työn tavoitteena oli luoda ihmispelaajan korvikkeena toimiva tekoäly kaksiulotteiseen monen yhtäaikaisen pelaajan videopeliin.

Videopeli ja tekoäly toteutettiin Game Maker 8 -ohjelmalla ja GML-skriptikielellä. Videopelin grafiikka toteutettiin GIMP 2 -ohjelmalla ja Game Maker 8 -ohjelman sisäisellä kuvanmuokkausohjelmalla.

Videopeliin toteutettu tekoäly onnistui toimimaan suhteellisen haastavana ja monipuolisena ihmispelaajien vastustajana. Tekoälyn ohjaaman pelihahmon käyttäytymisestä ei saatu luonnollista, ja tekoälyn käyttäytymiseen jäi pieniä, korjattavissa olevia virheitä.

Vaikka työssä keskitytään kaksiulotteisiin videopeleihin, siinä esitellyt algoritmit ja menetelmät sekä niiden vertailu ovat hyödyllisiä kenelle tahansa videopelien kehittämisestä kiinnostuneelle. Työssä toteutettua videopeliä on mahdollista kokeilla osoitteessa www.yoyogames.com/games/110832-demolisher.

Avainsana(t) **tekoäly, reitinhaku, videopeli**

Author	Antti Kukkonen	Year	2010
Subject of thesis	Pathfinding and AI Decision Making in a Two-Dimensional Video Game		
Number of pages	52		

This thesis is about the theory and the practical approach to AI (artificial intelligence) and pathfinding in two-dimensional games. The thesis is an independent research and an application development project and does not have a commissioner.

The first goal of the thesis was to describe the history of AI and to explain the role of AI in video games. The second goal was to present ways to apply the pathfinding algorithms and AI decision-making methods to two-dimensional video games. The third goal was to develop a functional AI controlled substitute for a human player in a two-dimensional video game.

The video game and the AI were developed using the Game Maker 8 software and the GML scripting language. The graphics of the video game were created using the GIMP 2 software and the Game Maker's integrated picture editor.

The artificial intelligence developed for this thesis managed to provide a relatively challenging and diverse opponent for human players, although the AI controlled character did not behave naturally compared to human players. In addition, some small, repairable errors were left in the AI behaviour.

This thesis focused on two-dimensional video games, but the algorithms and the methods described and compared in the thesis are useful to anyone interested in video game development. The video game developed for this thesis is available at www.yoyogames.com/games/110832-demolisher.

Key words **artificial intelligence, pathfinding, video games**

SISÄLTÖ

KUVIOLUETTELO

KÄSITTEET JA TERMIT

Algoritmi

Algoritmi on tarkasti määritelty sarja ohjeita, jotka pyrkivät ratkaisemaan tietyn ongelman.

Animaatio

Animaatio on esimerkiksi videopelissä esiintyvä kuvasarja, jonka osia vuoronperään näyttämällä saadaan aikaan illuusio liikkeestä.

Entiteetti

Entiteetti tarkoittaa jotain elävää tai elotonta, jonka oletetaan, tiedetään tai voidaan päätellä olevan selkeästi olemassa.

Graafi

Graafilla tarkoitetaan tiedon abstraktia esittämistä kuvion avulla.

Iteraatio

Iteraatiolla tarkoitetaan esimerkiksi tietokoneohjelmassa esiintyvää toistorakennetta, eli ohjelman osaa joka toistaa tiettyjä asioita useasti ohjelman ajon aikana.

Peli-idea

Peli-idea on se idea tai ajatus, jonka pohjalta videopeliä on alun perin alettu kehittämään.

Pelihahmo

Pelihahmolla tarkoitetaan videopelissä esiintyvää, yleensä pelaajan tai tekoälyn ohjaamaa tai hallitsemaa virtuaalista ihmishahmoa, olentoa, konetta tai muuta vastaavaa.

Pelisessio

Pelisessiolla tarkoitetaan videopelin sisäistä istuntoa, eli kokonaisuutta joka sisältää esimerkiksi yhden ottelun tai ottelusarjan.

Pseudo-koodi

Pseudo-koodi on tapa kuvata jonkin asian toteutusta tietokoneohjelmassa universaalilla tavalla, kiinnittämättä toteutusta mihinkään ohjelmointikieleen.

Skripti

Skripti eli komentosarja on tiedosto, joka sisältää jossakin ohjelmassa tai ympäristössä suoritettavaksi tarkoitettu komentoja.

Skriptikieli

Skriptikielen eli komentosarjakielen avulla kirjoitetaan komentosarjoja jollekin ohjelmalle tai ympäristölle. Skriptikieli eroaa ohjelmointikielestä siinä, että skriptikieli on yleensä tarkoitettu yhden tai useamman sovelluksen komentojen kirjoittamiseen, kun taas ohjelmointikieli on tarkoitettu sovellusten luomiseen.

Sovellus

Tässä opinnäytetyössä sovelluksella tarkoitetaan tietokoneohjelmaa.

Sprite

Spritellä tarkoitetaan yksittäistä, yleensä osittain läpinäkyvää kuvaa, joka edustaa esimerkiksi pelissä olevaa entiteettiä, kuten pelihahmoa tai pelin sisäistä objektia.

Videopeli

Videopeli on esimerkiksi tietokoneohjelma, jonka päämäärä on viihdyttää ihmistä antamalla muun muassa ajattelua, luovuutta, muistia tai refleksejä vaativia tehtäviä ihmisen suoritettavaksi. Videopelin päämääränä voi olla myös kertoa tarina, johon ihminen voi vaikuttaa toiminnoillaan. Videopelissä täytyy olla aina jokin päämäärä tai viihdyttävä elementti.

1 JOHDANTO

Sana tekoäly viittaa koneissa oleviin toimintoihin, jotka pyrkivät jäljittelemään ihmisille tyypillisiä, älykkyyttä vaativia toimintoja. Myös sanaa keinoäly (artificial intelligence) on käytetty synonyymina tekoälylle. Tekoälyllä on pyritty ratkaisemaan ainutlaatuisia ongelmia melkein vuosisadan ajan sekä sen avulla on jopa pyritty luomaan ihmisen kaltaisia, älykkäästi ajattelevia ja itsestään tietoisia koneita. Silti tekoälyn osuus viihteessä on tullut näkyväksi kuluttajille vasta viimeisimpien vuosikymmenien aikana. Yksi nykyaikaisen viihteen suurimmista osa-alueista, videopeliteollisuus, käyttää hyödykseen monia tekoälyllä saavutettuja tuloksia.

Siinä missä akateeminen tekoälytutkimus suuntautuu ihmisten elämää helpottavien tekoälyä hyödyntävien sovellusten suunnitteluun ja kehittämiseen, videopelikehittäjien tekoälysovellukset pyrkivät luomaan mahdollisimman viihdyttävän tekoälyn. Tekoälyn avulla peliin on mahdollista luoda voimakas illuusio todellisuudesta sekä sen avulla peliin saadaan lisättyä vaihtelevaa, pelaajan toimintoihin ja osaamistasoon mukautuvaa haastetta.

Tämän opinnäytetyön tavoitteet voidaan jakaa kolmeen osaan. Ensimmäinen tavoite on kertoa lyhyesti tekoälyn kehittymisen vaiheet ja kuvailla tekoälyn merkitys videopeleissä. Toinen tavoite on antaa lukijalle käsitys siitä, miten reitinhakualgoritmeja ja tekoälyn suorittamaa päätöksentekoa voidaan hyödyntää kaksiulotteisessa videopelissä. Kolmas tavoite on toteuttaa erilaisia reitinhakualgoritmeja ja tekoälyn päätöksentekoon soveltuvia menetelmiä käyttäen tekoälyn ohjaama ihmispelaajan korvike kaksiulotteiseen usean yhtäaikaisen pelaajan videopeliin.

Tekoälyn ja algoritmien toteutuksen osalta tässä työssä keskitytään vain tekoälyn reitinhakuun ja päätöksentekoon aiheen laajuuden vuoksi. Työssä ei käsitellä videopelin toteutusprosessista, vaan kuvataan toteutettavan pelin päämäärä, elementit ja käyttöliittymä sekä pelin kehittämisessä käytetyt työvälineet ja -menetelmät.

Työn ensimmäisessä ja toisessa luvussa määritellään tekoäly lyhyesti sekä käsitellään tekoälyn kehittymisen vaiheet ja tekoälyn osuus videopeleissä. Työn kolmannessa ja neljännessä luvussa esitellään reitinhakualgoritmeja

sekä tekoälyn päätöksentekoon soveltuvia menetelmiä. Työn viidennessä luvussa käsitellään toteutetun videopelin päämäärä, elementit ja käyttöliittymä, työssä käytetyt työvälineet ja -menetelmät sekä videopeliin toteutetun tekoälyn osa-alueet.

2 TEKOÄLYN HISTORIAA

2.1 Tekoälyn kehittymisen vaiheita

Tekoälyllä tarkoitetaan prosessia, jossa luodaan ihmisten mielestä älykkäästi toimivia koneita. Tämä voi tarkoittaa esimerkiksi ihmisten toimintojen matkimista tai yksinkertaisia toimintoja, kuten dynaamisessa ympäristössä selviämistä. (Jones 2003, 1.) Esimerkiksi itseohjautuva pölynimuri tarvitsee tekoälyn, joka osaa väistellä esineitä. Tätä voidaan kutsua dynaamisessa ympäristössä selviytymiseksi.

Tekoälyn modernin syntymän voidaan sanoa tapahtuneen 1950-luvulla. Tällöin brittiläinen matemaatikko Alan Turing ehdotti *Turingin koetta* tavaksi tunnistaa tekoäly. Testissä yksi tai useampi ihminen esittää kysymyksiä kahdelle salatulle entiteetille ja vastausten perusteella määritellään, kumpi entiteetti on ihminen. Jos tätä ei vastausten perusteella saada selville, tekoälyn todetaan olevan älykäs. (Jones 2003, 4.) Aiheeseen liittyen on pidetty vuodesta 1990 lähtien vuosittainen kilpailu *Loebner Prize*, jossa Turingin kokeen läpäisseen tietokoneohjelman kehittäjälle luvataan 100 000 dollarin palkinto ja kultamitali. Tähän mennessä vielä yksikään tietokoneohjelma ei ole läpäissyt testiä. (The Loebner Price 2007.) Tekoälyn modernin syntymän aikakautena todettiin, että tietokoneet pystyvät tunnistamaan numerotiedon lisäksi myös symboleita. Aikakauden suurimmaksi saavutukseksi tekoälyn saralla voidaan sanoa Arthur L. Samuelin kehittämä tammea pelaava ohjelma, jonka onnistui lopulta voittaa oma luojansa. (Jones 2003, 4.) Vuonna 1958 sai myös syntynsä hyvin tekoälyn kehittämiseen soveltuva ohjelmointikieli *Lisp*, joka on yhä tänäkin päivänä yleisessä käytössä (Russell–Norvig 2003, 18–19).

Ensimmäiseksi tekoälyn nousun aikakaudeksi voidaan kutsua 1960-lukua, jolloin tietokoneiden teknologian kehittyminen ja yhä useamman tutkijan keskittyminen tekoälyn tutkimiseen aiheutti tekoälyn näkyvyyden kasvua siinä määrin, että aiheesta julkaistiin kaksi aihetta kritisoivaa kirjaa – Mortimer Tauben *Computers and Common Sense: The Myth of Thinking Machines* ja Stuart Dreyfusin *Alchemy and AI*. Tuona aikakautena perustettiin Stanfordin yliopiston tekoälylaboratorio. Laboratorion tuloksena syntyi muun muassa liikkuva robotti *Shakey*, joka osasi liikkua ruudukkoon sijoitetussa maailmassa ja seurata yksinkertaisia ohjeita. (Jones 2003, 5.)

Tekoälyn historian ensimmäinen laskukausi tapahtui 1970-luvulla, jolloin tekoälysovellukset eivät vastanneet aikakaudelle luonteenomaisia kohtuuttomia odotuksia. Käytännölliset tekoälysovellukset olivat harvinaisia sekä rahoitus tekoälyn tutkimiseen ja kehittämiseen oli pientä. (Jones 2003, 5.) Yleinen syy tekoälysovellusten toimintakyvyttömyydelle oli niille annetun tehtävän vaikea hallittavuus (Russell–Norvig 2003, 21). Kaikesta huolimatta 1970-luvulla kehitettiin uusi ohjelmointikieli, *Prolog*, joka soveltui hyvin symboleita käsittelevien ohjelmistojen kehittämiseen. Yhtenä aikakauden suurena saavutuksena voidaan pitää Backgammon-lautapelin pelaamista varten kehitettyä tekoälyä, jonka onnistui voittaa Backgammonin maailmanmestarin, Luigi Villan. (Jones 2003, 6.)

Tekoälyn kannalta 1980-luku oli nousua ja laskua. Tekoälyyn pohjautuvien sovellusten ja laitteistojen myynti ylitti 400 miljoonan USA:n dollarin määrän vuonna 1986 sekä esimerkiksi puheentunnistussovellukset kehittyivät siihen pisteeseen, ettei puhujan tarvinnut olla aina sama henkilö tai aiheeseen liittyen koulutettu saadakseen sovelluksen toimimaan. Myös tarkasti rajatut, yksittäisiin ongelmiin vastaavat tekoälysovellukset yleistyivät. Näitä sovelluksia kutsutaan asiantuntijajärjestelmiksi, sillä ne pyrkivät mallintamaan tietyn sovellusalan asiantuntijan tietämystä ratkaistessaan jotain sovellusalaan liittyvää ongelmaa ja vastaamaan sille lähetettyihin kyselyihin automaattisesti. (Jones 2003, 6–7; Tozour 2002, 5.) Esimerkki tällaisesta sovelluksesta on rauta- ja terästeollisuudessa käytettävän masuunin valvontajärjestelmä, joka reagoi masuunin prosessihäiriöihin ja tekee prosessista vakaamman (Eteplan 2010).

Asiantuntijajärjestelmien laajuus, monimutkaisuus ja vaikea ylläpidettävyys aiheuttivat 1980-luvun nimeämisen myös tekoälyn laskun aikakaudeksi (Jones 2003, 6). Ensimmäinen kaupallinen asiantuntijajärjestelmä, nimeltään *R1*, otettiin käyttöön *Digital Equipment Corporation* -yrityksessä vuonna 1980. Järjestelmän tarkoitus oli auttaa uusien tietokonejärjestelmien tilausten kokoonpanossa. Vuoteen 1986 mennessä *R1* oli tuottanut arviolta 40 miljoonan USA:n dollarin säästöt yhtiölle. Tekoälyä alettiin pitää tieteenalana 1980-luvun loppuun mennessä. Tällöin yleistyi uusien tekoälyteorioiden luominen vanhojen teorioiden pohjalta. Tätä ennen uudet tekoälyteoriat oli yleensä luo-

tu ilman aikaisempia aiheeseen liittyviä teorioita. (Russell–Norvig 2003, 24–25.)

Niin kutsutut *heikon tekoälyn* sovellukset yleistyivät tekoälyn viimeisimpänä nousun aikakautena 1990-luvulla. *Vahvalla tekoälyllä* tarkoitetaan sovelluksia, joiden on tarkoitus ajatella samalla älykkyyden tasolla kuin ihmiset ja olla tietoisia itsestään, kun taas *heikolla tekoälyllä* tarkoitetaan sovelluksia, jotka pyrkivät toimimaan älykkäästi tietyissä toimintaympäristöissä. Esimerkkeinä *heikon tekoälyn* sovelluksista ovat muun muassa luottokorttihuijausten estämiseen kehitetyt järjestelmät, kasvontunnistusjärjestelmät ja automatisoidut aikataulutusjärjestelmät. (Jones 2003, 2, 7; Russel–Norvig 2003, 947.) Huomattava saavutus tekoälyn saralla oli shakkia pelaava ohjelma *Deep Blue*, joka voitti vuonna 1997 shakin maailmanmestarin Garri Kasparovin kuuden pelin ottelusarjassa luvuin 3,5–2,5. Vuonna 1996 *Deep Bluen* aikaisempi versio hävisi Kasparoville kuuden pelin ottelusarjassa luvuin 4–2. (Campbell 2001, 1–4.) Myöhään 1990-luvulla otettiin myös ensimmäistä kertaa käyttöön ohjelma *Remote Agent*, jonka tarkoituksena oli hallita avaruusaluksen ohjausta lyhyellä aikavälillä. Tuota ennen avaruusalusta oli yleensä ohjannut tiedemiesryhmä ohjauslaitteiston avulla maasta käsin. (Jones 2003, 7.)

”Viidenkymmenen vuoden aikana annetuista lupauksista huolimatta tekoäly ei ole päässyt oikeastaan yhtään lähemmäksi älykkyydeltään ja kyvyiltään ihmisen kaltaisten koneiden rakentamista. Tietokoneiden laskentateho on tänä aikana moninkertaistunut aivan uskomattomalla tavalla, mutta lisääntynyttä laskentatehoa ei ole päästy hyödyntämään, koska käyttökelpoinen teoria älykkyyden mekaanisesta toteuttamisesta puuttuu kokonaan.” (Kokkarinen 2003, 296.) Vaikka kukaan ei ole onnistunut vielä tähän mennessä kehittämään kaikin puolin ihmisen veroista tekoälyä, tai edes lähes ihmisen veroista, tekoälyn avulla on saavutettu hyviä tuloksia muilla sovellusalueilla.

Valtaosassa tekoälytutkimusta on ollut taustalla ajatus mahdollisuudesta kehittää esimerkiksi tietokoneohjelma tai robotti, jonka tarkoitus on auttaa jonkin yksittäisen ongelman ratkaisussa (Honkela 2009, 13). Tästä johtuen tekoälyn sovellusalue on nykyään erittäin laaja. Tunnetuimpina sovellusalueina voidaan mainita automaattiohjaus, lääketieteellinen diagnoosi, logistinen suunnittelu, robotiikka, luonnollisten kielten prosessointi ja videopelit. (Russell–

Norvig 2003, 27–28.) ”Käytännön tekoälytutkimuksen tavoitteet ovat nykyisin paljon vaatimattomampia kuin alan alkuaikoina. Tuolloin lupailtiin parissa vuodessa tai korkeintaan vuosikymmenessä rakennettavan laitteita, jotka alan parhaiden ihmisasiantuntijoiden tasoisesti kirjoittavat runoja, säveltävät musiikkia, kääntävät tekstejä kielestä toiseen ja toimivat eri alojen huippuasiantuntijoina.” (Kokkarinen 2003, 296.)

2.2 Tekoäly videopeleissä

Suurin osa eroavaisuuksista videopelien tekoälyssä ja akateemisessa, valtavirtaa edustavassa tekoälyssä johtuu niiden erilaisista päämääristä. Akateemisen tekoälyn päämäärä on ratkaista ainutlaatuisia ongelmia, kuten esimerkiksi ihmisen tajunnan simuloiminen tai luonnollisten kielten ymmärtäminen, kun taas videopelien tekoälyn päämäärä on viihdyttää. (Tozour 2002, 9.) Videopeli voi olla esimerkiksi kehittävällä tavalla haastava, jolloin ihmispelaaja sitä onnistuneesti pelaamalla tuntee oppineensa jotain (Rouse III 2001, 2). Koska videopelien tekoälyn on tarkoitus toimia vain pelin sisällöstä riippuvien käyttäytymismallien mukaisesti, ihmismäinen älykkyys tekoälyssä ei ole aina vaadittavaa tai edes haluttua (Tozour 2002, 10).

Ensimmäisten kaupallisten videopelien aikakautena tekoäly suunniteltiin toimimaan pääasiassa kolikkoautomaateissa olevissa peleissä. Tällaisia pelejä olivat muun muassa *Pong*, *Pac-Man*, *Space Invaders*, *Donkey Kong* ja *Joust*. (Tozour 2002, 3.) LaMothen (2000, 10) mukaan Nolan Busnellin suunnittelema *Pong* käynnisti koko videopeliteollisuuden yhdessä yössä ja sai aikaan videopeliyritys *Atarin* syntymän. Ylempänä mainitut pelit olivat tekoälyn osalta vain sarja yksinkertaisia listattuja toimintoja yhdistettynä satunnaiseen päätöksentekoon. Esimerkki satunnaisesta päätöksenteosta on ihminen, joka valitsee kolikkoa heittämällä suunnan saavuttuaan tien risteykseen. Satunnaisuus videopelin tekoälyssä tekee pelistä vähemmän ennustettavan. (Tozour 2002, 3.) Tällaiset pelit ovat pelaajan kannalta yleensä paljon viihdyttävämpiä kuin esimerkiksi pelit, joissa pelaaja osaa arvata yhden pelikerran jälkeen, mitä pelihahmot tulevat tekemään vastaisuudessa.

Ensimmäisinä varsinaisina pioneereina videopelien tekoälyn saralla voidaan pitää pelejä, jotka vaativat pelaajilta strategista ajattelua ja toimimista. Näitä pelejä kutsutaan yleisesti strategiapeleiksi. Ollakseen viihdyttävä pelaajan

silmissä, strategiapelin tekoälyn on osattava suorittaa monimutkaisia pelihahmokohtaisia toimintoja sekä korkeatasoisia strategisia ja taktisia toimintoja. Strategiapeleissä olevien pelihahmojen suuresta lukumäärästä johtuen pelissä esiintyvä tekoäly täytyy olla erityisen hyvin suunniteltu, jotta vältettäisiin pelin suorituskykyongelmat. Ensimmäisissä strategiapeleissä esiintyvä tekoäly on toiminut hyvänä esimerkkinä ja lähtökohtana joidenkin nykyaikaisten videopelien tekoälylle. Kuuluisia strategiapelejä ovat muun muassa *Civilization*, *Civilization 2*, *Warcraft II* ja *Age of Empires 2: The Age of Kings*. (Tozour 2002, 3.)

Suuri osa-alue nykyaikaisissa videopeleissä on ensimmäisen persoonan ammuskelupelit, niin kutsutut FPS-pelit (first-person shooter). Näissä peleissä pelaajan hallitseman pelihahmon toimintoja seurataan hahmon silmistä katsottuna. Koska FPS-pelien päämääränä on yleensä toimintapainotteisten tehtävien suorittaminen ja ihmisten kaltaisten tai ihmistä jäljittelevien vastustajien eliminoiminen pelissä esiintyviä aseita käyttämällä, tekoälyllä on suuri merkitys FPS-pelien todentuntuisuuden luomisessa.

Kun *Valve* julkaisi FPS-pelinsä *Half-Lifen* vuonna 1998, peliä kehuttiin sen edistyneen taktisen tekoälyn vuoksi. *Epic Gamen* vuonna 1999 julkaiseman *Unreal: Tournament* -pelin tekoälyn ohjaamat pelihahmot ovat tunnettuja mukautuvuudestaan ja taktisesta osaamisestaan. *Sierra Studiosin* vuonna 1999 julkaisema *SWAT 3: Close Quarters Battle* oli hyvä esimerkki ihmismäisestä toiminnasta tekoälyn osalta ja peli käytti sattumanvaraista tekoälyä onnistuneesti luodakseen erilaisen pelikokemuksen jokaiselle pelikerralle. (Tozour 2002, 4.)

Muista aikaisemmista peleistä hyvin paljon poikkeava *Maxisin* vuonna 2000 julkaisema *The Sims* oli erityisen tunnettu pelihahmojen persoonallisuuden syvyydestä. Tässä pelissä pelaajan päämäärä on rakentaa, kalustaa ja sisustaa pelihahmolle talo sekä huolehtia pelihahmon työssä käymisestä ja päivittäisistä rutiineista. Muista peleistä poiketen *The Sims* -pelin tekoäly tekee päätökset kuuntelemalla peliympäristön lähettämiä viestejä, kuten esimerkiksi pelihahmon ollessa nälkäinen, hahmo kuuntelee ympäristön lähettämiä viestejä liittyen mahdollisiin ruokapaikkoihin ja toimii näiden havaintojen perusteella. (Tozour 2002, 4.) Rouse III:n (2001, 161) mukaan *The Simsin* suo-

sio perustui yksinomaan sen tekoälyn vahvuuteen. Toinen muista videopeleistä huomattavasti poikkeava peli oli *Lionheadin* vuonna 2001 julkaisema *Black & White*, jonka peli-idea perustui pelissä olevan olennon (creature) kouluttamiseen ja kasvattamiseen sekä olennon kasvuympäristönä toimivan kylän rakentamiseen. Tämä sai aikaan pelaajan huomiokyvyn keskittymisen tekoälyn kehittymiseen sellaisella tavalla, joka on mahdotonta suurimmassa osassa muita videopelejä. (Tozour 2002, 4.) *The Sims* ja *Black & White* ovat hyviä esimerkkejä peleistä, joiden päämäärä ja viihdyttävyys perustuivat melkein yksinomaan pelissä esiintyvään tekoälyyn.

Syy tekoälyn menestykseen nykyaikaisissa peleissä johtuu yksinkertaisesti siitä, että videopelien kehittäjät ovat lopultakin alkaneet ottamaan tekoälyn kehittämisen vakavasti. Koska jokaisen videopelin päämäärä ja elementit ovat erilaisia, jokaisen videopelin tekoäly on hyvin riippuvainen itse videopelin toteutuksesta ja sisällöstä. Tästä johtuen viime vuosina on yleistynyt tapa nimetä yksinomaan tekoälyn kehittämiseen keskittyviä ohjelmoijia työskentelemään videopeliprojektissa koko pelin kehityksen ajan. Tämä on ollut suuri askel videopelien tekoälyn kehityksen osalta. Jopa ohjelmoijat ilman varsinaista kokemusta tekoälyn kehittämisestä ovat saaneet aikaiseksi hyviä tuloksia videopelien tekoälyyn liittyen. Tozourin mukaan tekoälyn kehittäminen ei vaadi ohjelmoijalta erityisosaamista tai loistavia ideoita, vaan todellisuudentajua, vähän luovuutta ja riittävästi aikaa työn loppuun saattamiseksi. (Tozour 2002, 4, 5, 11.)

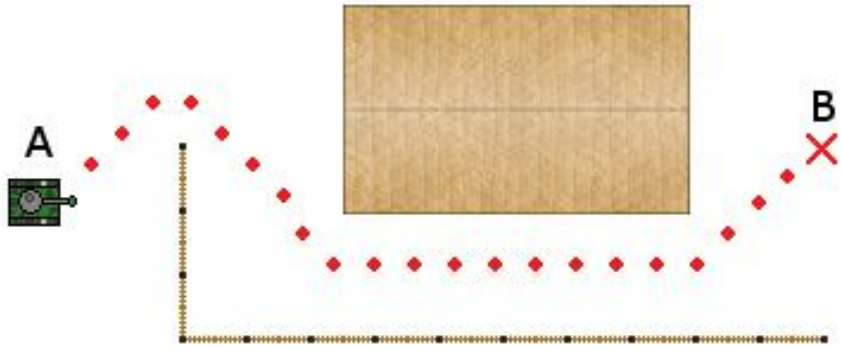
3 REITINHAKU OSANA VIDEOPELIÄ

3.1 Reitinhaun määritelmä

Reitinhaulla tarkoitetaan yleisesti ottaen kahden pisteen välisen reitin suunnittelua tietokoneohjelmissa. Videopeleissä reitinhaku tarkoittaa esimerkiksi liikkuvan pelihahmon kulkureitin suunnittelemista peliympäristössä. Reitinhaulla on suuri merkitys kun halutaan luoda videopelissä esiintyvästä tekoälystä uskottava. Jos tekoälyn ohjaama pelihahmo ei osaisi kävellä ovista tai törmäilisi puihin metsässä, pelin illuusio kärsisi huomattavasti.

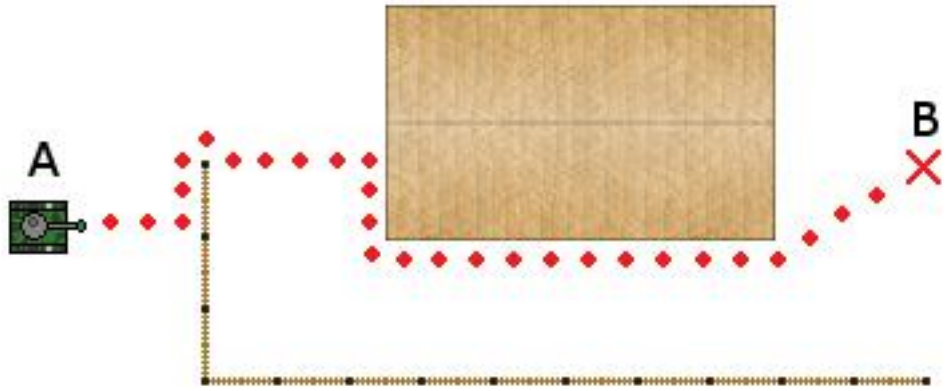
Monesti myös ihmisen ohjaamat pelihahmot tarvitsevat avukseen reitinhakua. Muun muassa RTS-peleissä (reaaliaikainen strategiapeli) ihmispelaajalla on yleensä ohjattavanaan monta pelihahmoa kerrallaan. Kun pelaaja haluaa liikuttaa pelihahmoja pelialueella, hän valitsee haluamansa hahmot ja asettaa niille päämäärän esimerkiksi osoittamalla pelialuetta hiiren osoittimella ja painamalla samaan aikaan hiiren nappia. Jos pelaajan hallussa olevat pelihahmot eivät osaisi esimerkiksi kiertää lähtöpisteen ja päämäärän välissä olevia rakennuksia, videopeli ei todennäköisesti olisi tarpeeksi pelattava pelaajan silmissä.

Tekoälyn ohjaama pelihahmo voi olla esimerkiksi panssarivaunu, jonka tarkoituksena on liikkua suorinta reittiä paikasta A paikkaan B. Jotta panssarivaunun liikkuminen peliympäristössä saataisiin näyttämään mahdollisimman luonnolliselta, sen täytyisi osata kiertää pelialueella sijaitsevat luonnolliset esteet.



Kuvio 1: Esimerkki reitinhaun käytöstä videopelissä.

Saavuttaakseen päämääränsä ohjelman täytyy laskea paras mahdollinen reitti paikasta A paikkaan B, ottaen huomioon alueet joihin ei voi päästä tai joihin ei ole järkevää mennä. Panssarivaunu voisi esimerkiksi mennä kuvion 1 osoittamalla tavalla kiertämällä kaikki esteet, tai ajaa kuvassa olevan aidan yli, jos aita on riittävän heikko. Johtuen mahdollisista ympäristössä tapahtuvista muutoksista, kuten esimerkiksi kuvassa olevan talon ja aidan välisen tilan tukkeutumisesta, polunlaskenta on tehtävä useasti pelin ajon aikana, jotta pelihahmo löytäisi aina parhaan mahdollisen reitin päämääräänsä.

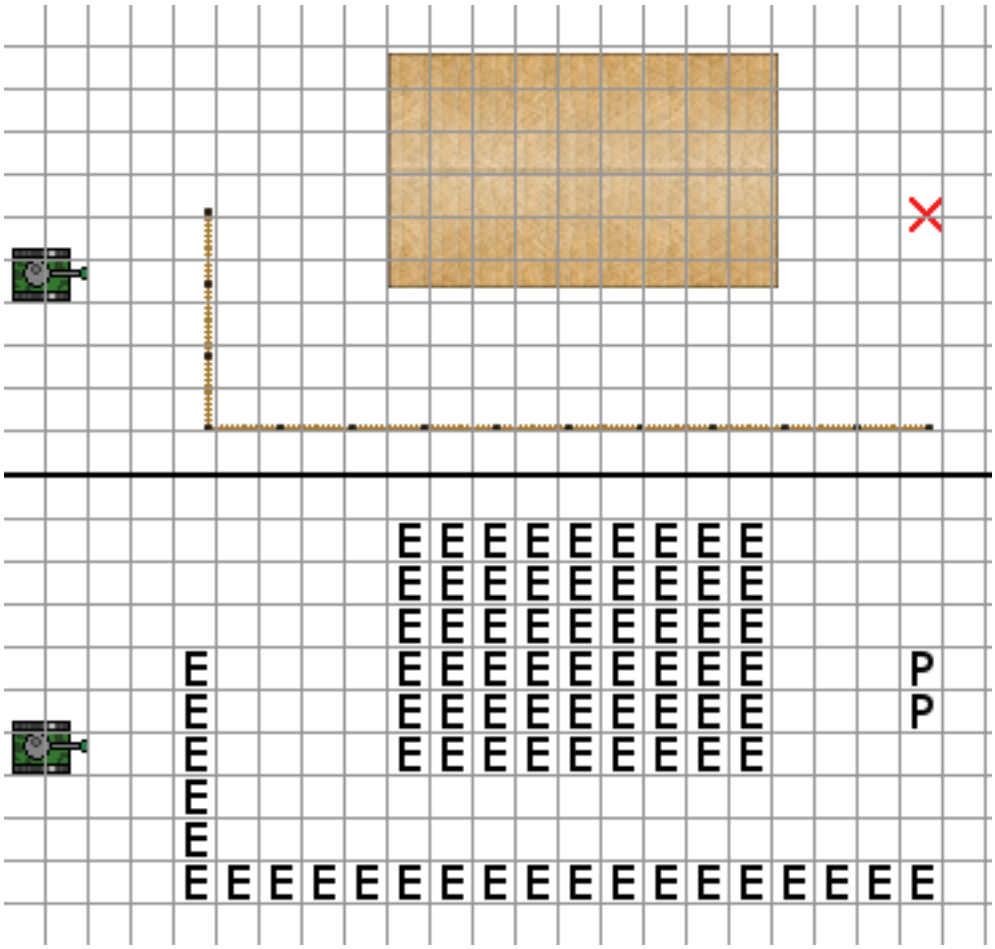


Kuvio 2: Esimerkki pelihahmon liikkumisesta ilman reitinhakua.

Jos videopelissä ei olisi käytössä reitinhakua, pelihahmo saattaisi liikkua kuvion 2 osoittamalla tavalla. Reitinhaun puuttuminen videopelistä saisi pelihahmon liikkumisen näyttämään epäluonnolliselta sekä kasvattaisi kahden pisteen välillä liikkumiseen tarvittavaa aikaa.

Koska polunetsintään liittyvät laskennat joudutaan monesti tekemään lähes reaaliajassa, yleensä rajoituksena on tietokoneen laskentateho. Huonosti suunniteltu polunlaskenta videopelissä voi nostaa pelin pelaamiseen tarvittavan laitteiston vaatimuksia huomattavasti. Tämän takia polunetsintää varten ja siihen liittyen on kehitetty monta erilaista algoritmia sekä aiheesta on kirjoitettu useita kirjoja, jotka kertovat kuinka algoritmeja voidaan soveltaa ja optimoida eri tilanteissa. Ennen algoritmien käyttöä videopelin pelialueesta luodaan yleensä yksinkertaistettu graafi, jonka tarkoitus on kuvata peliympäristön karttaa tai pelihahmon lähiympäristöä pelialueella. Tällaisten graafien käyttäminen mahdollistaa reitinhakualgoritmien suorittamisen yksinkertaisessa numero- tai tekstitietoa sisältävässä ympäristössä, mikä helpottaa arvojen

vertailua ja etäisyyksien laskemista. Graafin avulla saadaan myös rajattua peliympäristössä esiintyvät, algoritmien toiminnan kannalta merkityksettömät arvot pois ympäristöstä, jossa laskenta ja vertailu tulevat tapahtumaan.

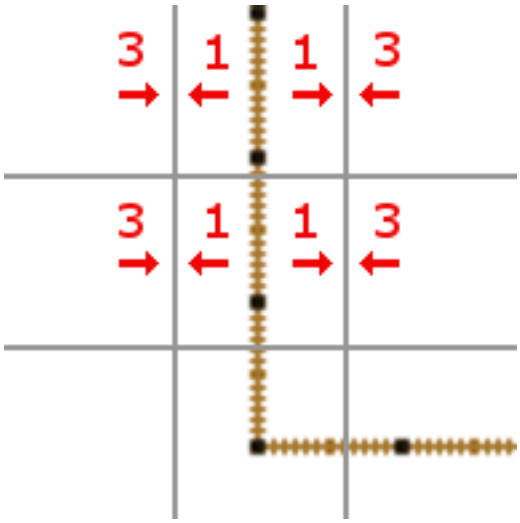


Kuvio 3: Esimerkki pelialuetta kuvaavasta graafista.

Jos kuvioissa 1 ja 2 kuvattu peliympäristö haluttaisiin yksinkertaistaa graafin avulla, pelimaailma voitaisiin mallintaa ruudukkoon esimerkiksi kuvion 3 osoittamalla tavalla. Kuviossa 3 jokaiseen esteen sisältävään ruutuun sijoitetaan teksti ”E” ja jokaiseen päämäärän sisältävään ruutuun sijoitetaan teksti ”P”. Kun algoritmi tarkkailee ruudukkograafia, sen on helppo suunnitella pelihahmon polku tyhjien ruutujen perusteella sekä selvittää missä päämäärä on. Ruudukoista voitaisiin myös tehdä suurempia, jolloin pelihahmon liikkumisen tarkkuus pienenisi, mutta peliympäristön mallintamiseen tarvittava graafi varaisi vähemmän tietokoneen keskusmuistia ja graafin prosessointi olisi nopeampaa.

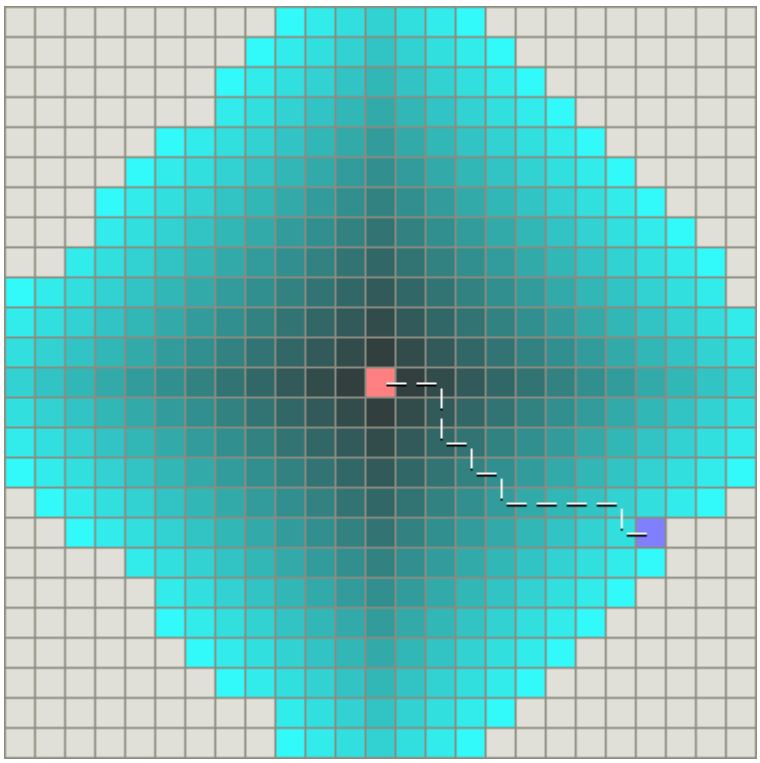
3.2 Dijkstran algoritmi

Dijkstran algoritmi on yksi kuuluisimmista algoritmeista lyhyimmän reitin etsimiseksi painotetusta graafista (weighted graph). Algoritmi on nimetty tietojenkäsittelyteoreetikko Edger Dijkstran mukaan, joka kehitti algoritmin 1950-luvun loppupuolella. *Dijkstran algoritmi* löytää lyhyimmän polun mistä tahansa painotetussa suunnatussa graafissa olevasta pisteestä mihin tahansa sekä kaikkiin graafissa oleviin pisteisiin, kun kaikki graafissa olevat painotusarvot ovat positiivisia. *Dijkstran algoritmi* saavuttaa tämän käyttämällä *ahnetta menetelmää* (greedy strategy, greedy algorithm). (McMillan 2005, 340.) *Ahne menetelmä* soveltuu sellaisiin ongelmanratkaisutilanteisiin, joissa täytyy mahdollisimman nopeasti löytää lähellä optimaalista ratkaisua oleva, tietyt kelvollisuusehdot täyttävä ratkaisu. Menetelmän jokaisessa vaiheessa valitaan paras tai sillä hetkellä ongelman ratkaisun kannalta parhaalta vaikuttava vaihtoehto. (Cormen–Leiserson–Rivest–Stein 2001, 370; Penttonen 1997, 54.)



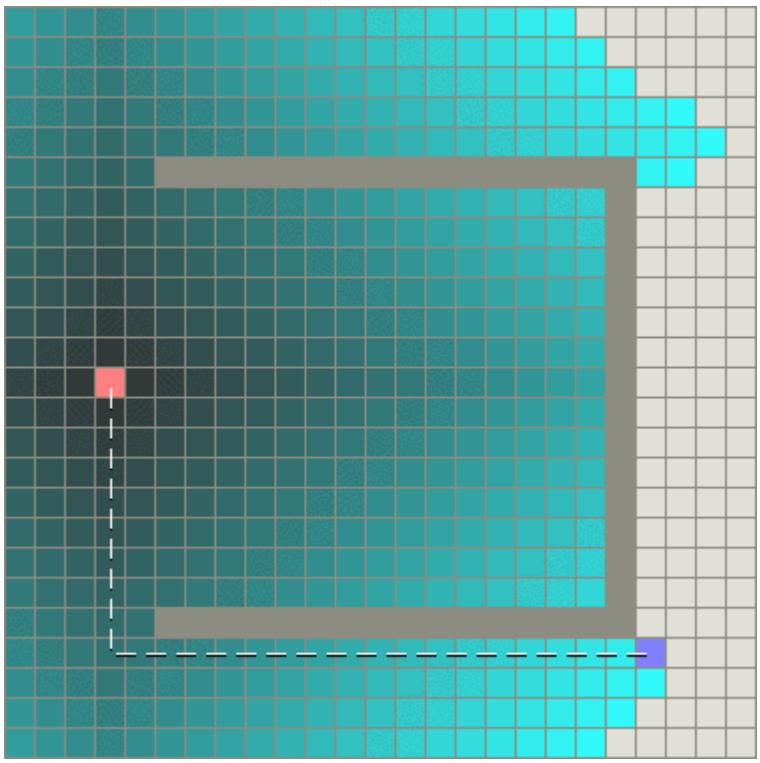
Kuvio 4: Esimerkki painotetusta suunnatusta graafista.

Kuvion 4 ruudukko on esimerkki pelissä esiintyvästä painotetusta suunnatusta graafista, jota algorimit voivat käyttää reitinhakuun. Ruutujen välillä olevat punaiset nuolet kertovat liikkumisen aiheuttaman kustannuksen kun siirrytään seuraavaan ruutuun. Ruskea viiva kuvaa peliympäristössä esiintyvää aitaa. Kuvion graafissa tyhjästä ruudusta aidan sisältävään ruutuun siirtyminen on kolme kertaa työläämpää pelihahmolle, kuin aidan sisältävästä ruudusta pois siirtyminen.



Kuvio 5: Dijkstran algorimin toiminta esteettömässä ympäristössä (Patel 2009).

Kuvio 5 kuvaa *Dijsktran algoritmin* toimintaa ympäristössä, jossa alkupisteen ja päämäärän välillä ei ole esteitä. Kuviossa reitinhakuympäristö ja algoritmin toiminta on mallinnettu yksinkertaisen ruudukon avulla, eikä ruutujen välisiä painotusarvoja ole esitetty kuvion selkeyden säilyttämiseksi. Vaaleanpunainen piste on alkupiste ja tummansininen ruutu oikealla on päämäärä. Loppuosa sinisestä kuvaa algoritmin tutkimaa aluetta siten, että tummin osa on ensimmäiseksi tutkittua aluetta. Algoritmin valitsemaa reittiä kuvataan katkoviivalla. (Patel 2009.)



Kuvio 6: Dijkstran algorimin toiminta esteiden lähellä (Patel 2009).

Kuvio 6 kuvaa, miten *Dijkstran algoritmi* käyttäytyy, kun alkupisteen ja päämäärän välillä on este. Vaaleanpunainen ruutu on alkupiste, tummanharmaat ruudut ovat esteitä, tummansininen ruutu oikealla on loppupiste ja loppuosa sinisestä kuvaa algoritmin tutkimaa aluetta samalla tavalla, kuin kuviossa 5. Algoritmin valitsemaa reittiä kuvataan katkoviivalla. Kuten kuvioista 5 ja 6 voidaan päätellä, *Dijkstran algoritmi* löytää aina reitin kahden pisteen välillä, jos se on mahdollista, mutta tutkii huomattavan määrän ympäristöä löytääkseen reitin päämäärään. Tämä voi vaikuttaa algoritmia käyttävän sovelluksen suoritusaikaan negatiivisesti, varsinkin jos kahden tutkittavan pisteen välinen etäisyys on suuri. (Patel 2009.)

Dijkstran algoritmin toteutus voidaan esittää pseudo-koodin avulla. Pseudo-koodissa käsitellään pelialuetta mallintavaa ruudukkograafia. Jokaisella ruudulla on arvot matka ja isäntä. Ruudun matka kuvaa matkaa alkupisteestä kyseiseen ruutuun. Ruudun isäntä tarkoittaa jotain toista ruutua, jonka kautta kyseiseen ruutuun on siirrytty, kun kuljetaan löydettyä reittiä pitkin. Näiden tietojen lisäksi algoritmi tarvitsee alkupisteenä toimivan ruudun, päämääränä toimivan ruudun, avoimen joukon tutkittaville ruuduille ja suljetun joukon jo tutkituille ruuduille.

Seuraava pseudo-koodi on esimerkki *Dijkstran algoritmin* toteutuksesta:

1 Aseta alkupisteen matkaksi 0 ja isännäksi määrittämätön.
2 Luo avoin joukko ja suljettu joukko sekä lisää alkupiste avoimeen joukkoon.
3 Valitse avoimesta joukosta se ruutu, jolla on pienin matka. Poista valittu ruutu avoimesta joukosta ja lisää suljettuun joukkoon.
4 Käy läpi jokainen kohdassa 3 valitun ruudun vieressä oleva sallittu ruutu, joka ei ole suljetussa joukossa ja tutki, onko niiden matka suurempi kuin valitun ruudun matka lisättynä näiden kahden ruudun välisellä matkalla.
5 Jos matka on suurempi tai matkaa ei löydy, aseta kohdassa 4 tutkittavan ruudun matkaksi valitun ruudun matka lisättynä näiden kahden ruudun välisellä matkalla ja aseta tutkittavan ruudun isännäksi kohdassa 3 valittu ruutu. Lisää tutkittu ruutu avoimeen joukkoon.
6 Jos avoin joukko on tyhjä tai päämäärä on lisätty suljettuun joukkoon, lopeta. Muussa tapauksessa palaa kohtaan 3.

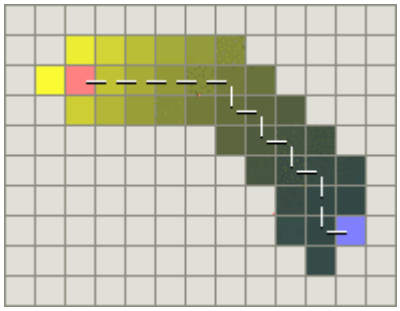
Algoritmi pitää yllä jokaisen tutkitun ruudun matkaa alkupisteeseen sekä isäntäruutua, jonka kautta kyseiseen ruutuun kuljetaan kuljettaessa optimaalista reittiä pitkin. Kun algoritmi on saavuttanut päämäärän, voidaan matka alkupisteen ja päämäärän välillä määrittää seuraamalla ruutujen isäntiä aloittamalla päämäärän isännästä. Kaikki algoritmin löytämän reitin ruudut ovat käsiteltyjen ruutujen joukossa. Pseudo-koodin kohdassa 4 mainittu sallittu ruutu tarkoittaa ruutua, jossa ei ole estettä liikkumiselle.

Dijkstran algoritmilla on myös muita käyttötarkoituksia, kuin kahden pisteen välisen reitin laskeminen. Jos videopelissä oleva pelihahmo haluaisi esimerkiksi tutkia onko sitä ympäröivässä pelialueessa mitään mielenkiintoista, *Dijkstran algoritmia* voitaisiin käyttää tutkimaan ympäröivä alue tasapuolisesti, painottamatta mitään suuntaa erityisesti (Lester 2005). Tässä tapauksessa algoritmin toteutus täytyy katkaista siinä vaiheessa, kun ympäristöä on tutkittu riittävän paljon tai kun vastaan tulee tietokoneen suorituskyvystä riippuvat rajat.

3.3 Paras ensin -haku

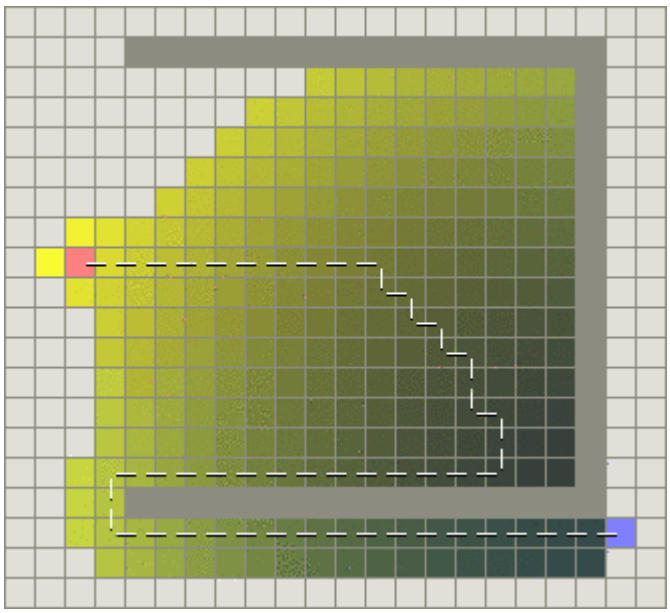
Paras ensin -haku (best-first search) toimii muuten samalla tavalla kuin *Dijkstran algoritmi*, mutta se valitsee jokaisessa vaiheessa tutkittavaksi sen pisteen, jonka oletetaan olevan lähimpänä päämäärää. Tämän takia *paras ensin -haun* suoritus tapahtuu paljon nopeammin kuin *Dijkstran algorimin* suoritus. Jos esimerkiksi päämäärä on alkupisteen eteläpuolella, haku keskit-

tyy etelään johtavaan alueeseen tutkiessaan ympäristöä. *Paras ensin -haun* nimi voi olla hämäävä, sillä se löytää aina esteettömässä ympäristössä parhaan reitin, mutta ei välttämättä löydä reittiä esteiden lähellä, vaikka reitti olisi löydettävissä muiden menetelmien tai algoritmien avulla. (Patel 2009; Russell–Norvig 2003, 94–95.)



Kuvio 7: Paras ensin -haun toiminta esteettömässä ympäristössä (Patel 2009).

Kuvio 7 kuvaa *paras ensin -haun* toimintaa ympäristössä, jossa alkupisteen ja päämäärän välillä ei ole esteitä. Kuviossa reitinhakuympäristö ja algoritmin toiminta on mallinnettu yksinkertaisen ruudukon avulla, eikä ruutujen välisiä painotusarvoja ole esitetty kuvion selkeyden säilyttämiseksi. Vaaleanpunainen piste on alkupiste ja tummansininen ruutu oikealla on päämäärä. Keltavihreä ruudukko kuvaa algoritmin tutkimaa aluetta siten, että vaalein osa on ensimmäiseksi tutkittua aluetta. Algoritmin valitsemaa reittiä kuvataan katkoviivalla. (Patel 2009.)



Kuvio 8: Paras ensin -haun toiminta esteiden lähellä (Patel 2009).

Kuvio 8 kuvaa, miten *paras ensin -haku* käyttäytyy, kun alkupisteen ja päämäärän välillä on este. Vaaleanpunainen ruutu on alkupiste, tummanharmaat ruudut ovat esteitä, tummansininen ruutu oikealla on loppupiste ja keltavihreät ruudut kuvaavat algoritmin tutkimaa aluetta samalla tavalla, kuin kuviossa 7. Algoritmin valitsemaa reittiä kuvataan katkoviivalla. Kuten kuvioista 7 ja 8 voidaan päätellä, paras ensin -haku tutkii pienemmän määrän ympäristöä löytääkseen reitin päämäärään, kuin *Dijkstran algoritmi*, mutta löydetty reitti ei ole aina yhtä hyvä, kuin *Dijkstran algoritmin* löytämä reitti. *Paras ensin -haun* käyttäminen voi joissain tapauksissa vaikuttaa algoritmia käyttävän sovelluksen suoritusaikaan positiivisesti, verrattuna *Dijkstran algoritmiin*. (Patel 2009.)

Paras ensin -haun toteutus voidaan esittää pseudo-koodin avulla. Pseudo-koodissa käsitellään pelialuetta mallintavaa ruudukkograafia. Jokaisella ruudulla on matka päämäärään ja isäntä. Ruudun matka päämäärään kuvaa arvioitua tai laskettua matkaa kyseisestä ruudusta päämäärään. Ruudun isäntä tarkoittaa jotain toista ruutua, jonka kautta kyseiseen ruutuun on siirrytty kun kuljetaan löydettyä reittiä pitkin. Näiden tietojen lisäksi algoritmi tarvitsee alkupisteenä toimivan ruudun, päämääränä toimivan ruudun, avoimen joukon tutkittaville ruuduille ja suljetun joukon jo tutkituille ruuduille.

Seuraava pseudo-koodi on esimerkki *paras ensin -haun* toteutuksesta:

1 Määritä alkupisteelle matka päämäärään ja aseta isännäksi määrittämätön.
2 Luo avoin joukko ja suljettu joukko sekä lisää alkupiste avoimeen joukkoon.
3 Valitse avoimesta joukosta se ruutu, jolla on pienin matka määränpäähän. Poista valittu ruutu avoimesta joukosta ja lisää suljettuun joukkoon.
4 Käy läpi jokainen kohdassa 3 valitun ruudun vieressä oleva sallittu ruutu, joka ei ole suljetussa joukossa.
5 Aseta kohdassa 4 tutkittavan ruudun matkaksi kohdassa 3 valitun ruudun matka lisättynä näiden kahden ruudun välisellä matkalla ja aseta tutkittavan ruudun isännäksi kohdassa valittu ruutu. Lisää tutkittu ruutu avoimeen joukkoon.
6 Jos avoin joukko on tyhjä tai määränpää on lisätty suljettuun joukkoon, lopeta. Muussa tapauksessa palaa kohtaan 3.

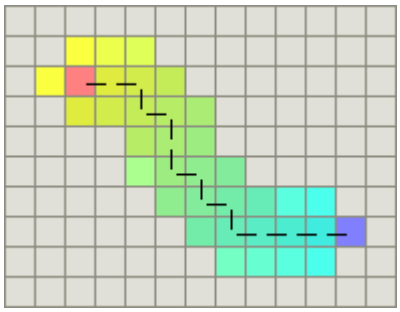
Algoritmi pitää yllä jokaisen tutkitun ruudun matkaa alkupisteeseen ja päämäärään sekä isäntäruutua, jonka kautta kyseiseen ruutuun kuljetaan men-

täessä optimaalista reittiä pitkin. Kun algoritmi on saavuttanut päämäärän, voidaan matka alkupisteen ja päämäärän välillä määrittää seuraamalla ruutujen isäntiä aloittamalla päämäärän isännästä. Kaikki algoritmin löytämän reitin ruudut ovat suljetussa, käsiteltyjen ruutujen joukossa. Pseudo-koodin kohdassa 4 mainittu sallittu ruutu tarkoittaa ruutua, jossa ei ole estettä.

Koska *paras ensin -haku* ei aina löydä reittiä kahden pisteen välillä, sitä ei kannata käyttää normaalitilanteissa reitinhakuun ympäristössä, jossa alkupisteen ja päämäärän välillä on mahdollisia esteitä. Algoritmilla voitaisiin simuloida esimerkiksi sokean pelihahmon liikkumista, koska algoritmi lähestyy määränpäätä sokeasti, tutkimatta ympäristöään kauempaa.

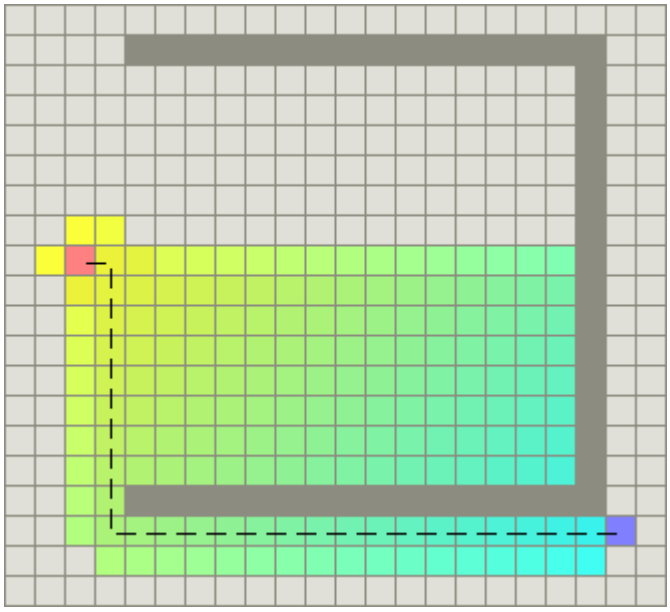
3.4 A*-algoritmi

*A** on tunnetuin muoto *paras ensin -haulle* sekä yleisesti peliteollisuudessa käytetty reitinhakualgoritmi. *A*-algoritmia* voidaan käyttää esimerkiksi löytämään reitti kahden kartalla olevan pisteen välillä. Vaikka on olemassa useita erilaisia reitinhakualgoritmeja, *A** löytää lyhyimmän reitin, jos reittejä on olemassa, ja tekee sen suhteellisen nopeasti – tämä erottaa sen muista reitinhakualgoritmeista. *A** on suunnattu algoritmi, mikä tarkoittaa sitä, ettei se hae sokeasti päämääräänsä (kuten rotta labyrintissä), vaan arvioi parhaan tutkittavan suunnan reitinhakuun ja palaa joskus takaisin päin tutkitulla reitillä kokeillakseen vaihtoehtoisia reittejä. (Tozour 2002, 8; Matthews 2002, 105; Russel–Norvig 2003, 97.) *A** saavuttaa tämän yhdistämällä *Dijkstran algoritmin* ja *paras ensin -haun* parhaat puolet. Tämä tarkoittaa sitä, että algoritmi valitsee ensimmäisenä tutkittaviksi pisteiksi ne, joilla on pienin yhteenlaskettu matka päämäärään ja alkupisteeseen. Matka päämäärään voidaan arvioida esimerkiksi laskemalla pisteiden välinen suora etäisyys, tai arvioimalla kuinka kauan pelissä esiintyvältä pelihahmolta menisi kulkea suorinta reittiä päämäärään, unohtaen mahdollisten reitillä olevien esteiden olemassaolon. (Patel 2009.)



Kuvio 9: A*-algoritmin toiminta esteettömässä ympäristössä (Patel 2009).

Kuvio 9 kuvaa *A*-algoritmin* toimintaa ympäristössä, jossa alkupisteen ja päämäärän välillä ei ole esteitä. Kuviossa reitinhakuympäristö ja algoritmin toiminta on mallinnettu yksinkertaisen ruudukon avulla, eikä ruutujen välisiä painotusarvoja ole esitetty kuvion selkeyden säilyttämiseksi. Vaaleanpunainen piste on alkupiste ja tummansininen ruutu oikealla on päämäärä. Kelta-vihreä-sininen ruudukko kuvaa algoritmin tutkimaa aluetta siten, että keltaisin osa on ensimmäiseksi tutkittua aluetta ja sininen osa viimeiseksi tutkittua aluetta. Algoritmin valitsemaa reittiä kuvataan katkoviivalla. (Patel 2009.).



Kuvio 10: A*-algoritmin toiminta esteiden lähellä (Patel 2009).

Kuvio 10 kuvaa, miten *A*-algoritmi* käyttäytyy, kun alkupisteen ja päämäärän välillä on este. Vaaleanpunainen ruutu on alkupiste, tummanharmaat ruudut ovat esteitä, tummansininen ruutu oikealla on loppupiste ja kelta-vihreä-siniset ruudut kuvaavat algoritmin tutkimaa aluetta samalla tavalla, kuin kuviossa 9. Algoritmin valitsemaa reittiä kuvataan katkoviivalla. Kuten kuvioista 9 ja 10 voidaan päätellä, *A*-algoritmi* tutkii pienemmän määrän ympäristöä löy-

tääkseen reitin päämäärään, kuin *Dijkstran algoritmi*, mutta löydetty reitti on aina yhtä hyvä kuin *Dijkstran algoritmissa*. Kun alkupisteen ja päämäärän välillä on esteitä, *A*-algoritmin* löytämä reitti on yleensä parempi kuin paras ensin -haun löytämä reitti. *A*-algoritmin* käyttäminen voi vaikuttaa algoritmia käyttävän sovelluksen suoritusaikaan positiivisesti, verrattuna *Dijkstran algoritmiin* ja *paras ensin -hakuun*. (Patel 2009.)

A*-algoritmin toteutus voidaan esittää pseudo-koodin avulla. Pseudo-koodissa käsitellään pelialuetta mallintavaa ruudukkograafia. Jokaisella ruudulla on arvot matka alkupisteeseen, matka päämäärään, näiden matkojen summa ja isäntä. Ruudun matka alkupisteeseen kuvaa matkaa alkupisteestä kyseiseen ruutuun. Ruudun matka päämäärään kuvaa arvioitua tai laskettua matkaa kyseisestä ruudusta päämäärään. Ruudun isäntä tarkoittaa jotain toista ruutua, jonka kautta kyseiseen ruutuun on siirrytty kun kuljetaan löydettyä reittiä pitkin. Näiden tietojen lisäksi algoritmi tarvitsee alkupisteenä toimivan ruudun, päämääränä toimivan ruudun, avoimen joukon tutkittaville ruuduille ja suljetun joukon jo tutkituille ruuduille.

Seuraava pseudo-koodi on esimerkki *A*-algoritmin* toteutuksesta:

1. Aseta alkupisteelle matka alkupisteeseen (0), arvioitu matka päämäärään, näiden matkojen summa ja isäntä (määrittämätön).
2. Luo avoin ja suljettu joukko sekä lisää alkupiste avoimeen joukkoon.
3. Valitse avoimesta joukosta se ruutu, jolla on pienin matkojen summa, lisää se suljettuun joukkoon ja poista se avoimesta joukosta.
4. Käy läpi jokainen kohdassa 3 valitun ruudun vieressä oleva sallittu ruutu, joka ei ole suljetussa joukossa.
5. Jos tutkittava ruutu ei ole avoimessa joukossa, lisää se siihen. Aseta tutkittavalle ruudulle matka alkupisteeseen, matka määränpäähän ja näiden matkojen summa. Aseta tutkittavan ruudun isännäksi kohdassa 3 valittu ruutu.
6. Jos kohdassa 4 tutkittava ruutu on avoimessa joukossa, tarkista onko sen matka alkupisteeseen suurempi kuin kohdassa 3 valitun ruudun matka alkupisteeseen lisättynä näiden kahden ruudun välisellä etäisyydellä. Jos on, aseta tutkittavan ruudun isännäksi kohdassa 3 valittu ruutu, aseta uusi arvo matkalle alkupisteeseen ja päivitä matkojen summa.
7. Jos määränpää lisättiin suljettuun joukkoon tai avoin joukko on tyhjä, lopeta. Muussa tapauksessa palaa kohtaan 3.

Algoritmi pitää yllä jokaisen tutkitun ruudun matkaa alkupisteeseen ja päämäärään sekä isäntäruutua, jonka kautta kyseiseen ruutuun kuljetaan mentäessä optimaalista reittiä pitkin. Kun algoritmi on saavuttanut päämäärän,

voidaan matka alkupisteen ja päämäärän välillä määrittää seuraamalla ruutujen isäntiä aloittamalla päämäärän isännästä. Kaikki algoritmin löytämän reitin ruudut ovat suljetussa, käsiteltyjen ruutujen joukossa. Pseudo-koodin kohdassa 4 mainittu sallittu ruutu tarkoittaa ruutua, jossa ei ole estettä liikkumiselle.

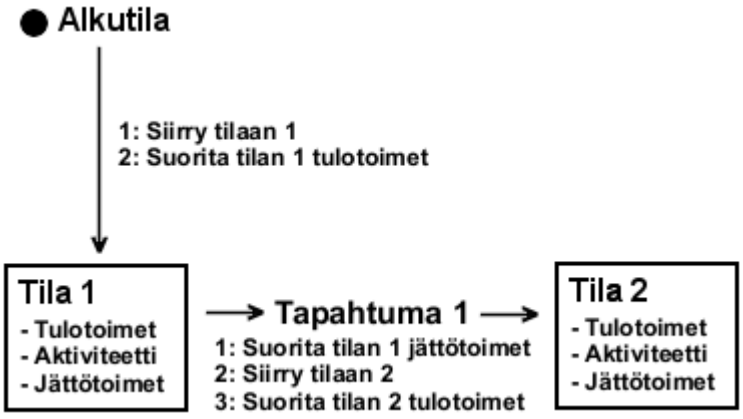
A-algoritmin* sovellukset eivät rajoitu pelkästään reitinhakuun. Sen sijaan, että algoritmille annettaisiin kaksi pistettä, joiden välille voitaisiin löytää reitti, algoritmille voitaisiin antaa kaksi pistettä, joiden välille ei löytyisi missään tapauksessa reittiä. Tätä tulosta voitaisiin käyttää kun määritetään esimerkiksi samankaltaisen ympäristön laajuus peliympäristössä. Jos haluttaisiin esimerkiksi määrittää pelissä esiintyvän metsäalueen laajuus, algoritmin avulla voitaisiin tutkia, kuinka laajalla alueella puut ovat lähellä toisiaan aloittamalla reitinhaku metsän sisältä, antamalla päämääräksi saavuttamaton piste ja määrittämällä metsässä sijaitsevat puut ainoiksi alueiksi, joilla on mahdollista liikkua. (Higgins 2002, 114.)

4 TEKOÄLYN SUORITTAMA PÄÄTÖKSENTEKO

4.1 Tilakone

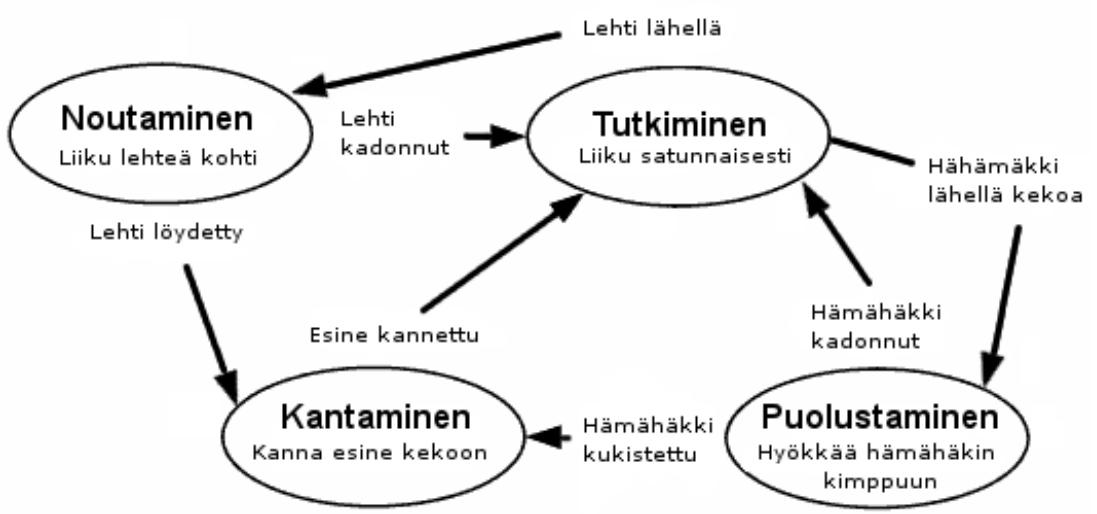
Koska tekoälyn on yleensä tarkoitus reagoida sovelluksessa tapahtuviin muutoksiin ja tilanteisiin, tarvitaan jonkinlainen tapa tai menetelmä toteuttaa tekoälyn suorittama päätöksenteko. Esimerkiksi videopelissä esiintyvän pelihahmon on päätettävä vaaran uhatessa, pitäisikö lähteä pakoon vai puolustautua. Yksi tapa toteuttaa tekoälyn suorittama päätöksenteko on tilakone.

Tilakone (state machine) on toimintamalli, joka sisältää tietyn määrän tiloja, ohjeet tilojen välillä siirtymiselle ja tilojen toiminnot. Yksittäinen tila voi määrittää esimerkiksi sen, mitä videopelin pelihahmo on tekemässä kyseisellä hetkellä ja tilojen väliset siirtymiset voivat mallintaa pelihahmon päätöksentekoa erilaisissa tilanteissa. (de Sousa 2002, 713; McGugan 2007, 141; Varanese 2003, 791.) Tilakoneen toimintaa voidaan mallintaa tilakaavion avulla.



Kuvio 11: Tilakoneen toimintaperiaate.

Kuvio 11 mallintaa tilakoneen toimintaperiaatteen tilakaavion avulla. Tilakaavion tilojen välisiä siirtymisiä kutsutaan tilasiirtymiksi. Alkutila kuvaa esimerkiksi sovelluksen käynnistämistä. Molemmat tilat sisältävät tulotoimen, aktiviteetin ja jättötoimen. Tulotoimi on toiminto, joka suoritetaan tilaan saavuttaessa. Jättötoimi on toiminto, joka suoritetaan tilasta poistuttaessa. Aktiviteetti on toiminto, jota suoritetaan koko ajan tilassa ollessa. Tapahtuma 1 kuvaa tapahtumaa, joka aiheuttaa tilasiirtymisen tilasta 1 tilaan 2. (Haikala–Märijärvi 2006, 138–139; Koskimies 2000, 131–132, 149–151.)



Kuvio 12: Esimerkki tilakoneen käytöstä.

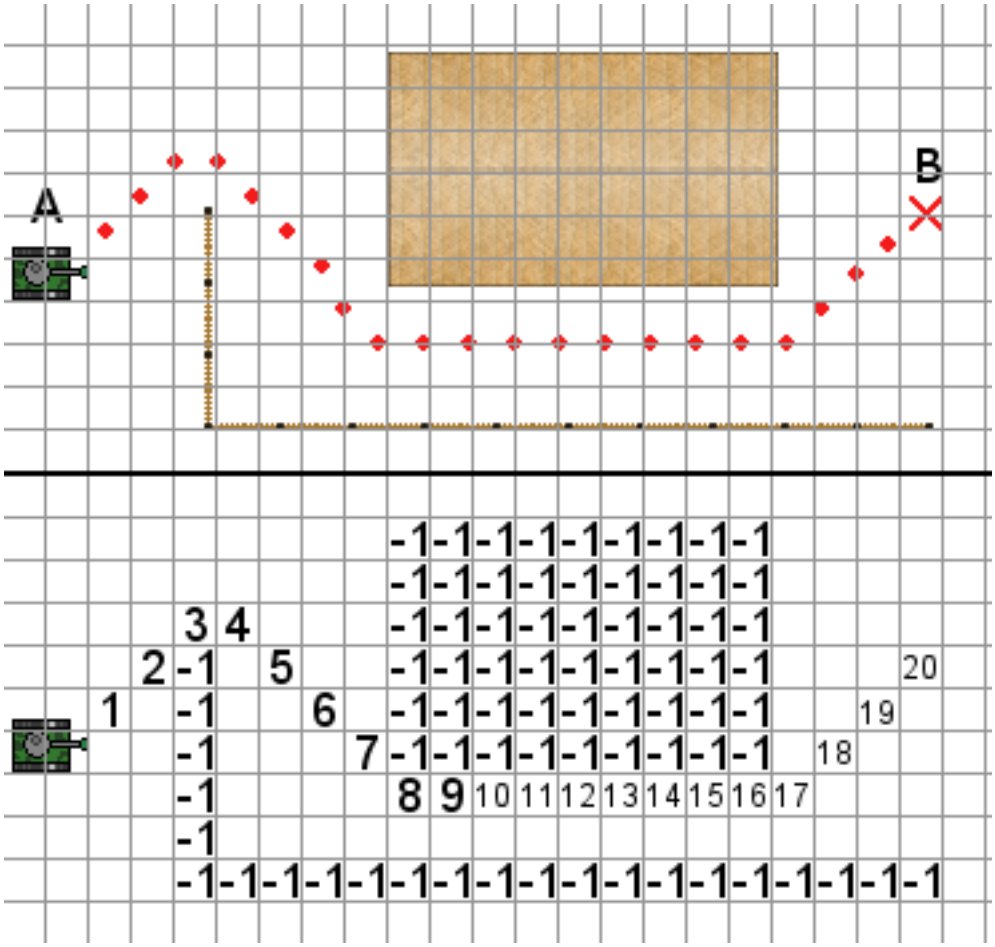
Esimerkkinä tilakoneesta toimii kuviossa 12 mallinnettu ohjelma, joka simuloi muurahaisten toimintaa muurahaiskeon ympärillä. Ohjelmassa on muurahaisia, puiden lehtiä, hämähäkkejä ja muurahaiskeko. Ohjelman käynnistyessä muurahaisilla on tilakoneen tilana ”tutkiminen”. Kun muurahaiset näkevät maassa lehden, tilaksi tulee ”noutaminen”, jolloin muurahaiset liikkuvat lehteä kohti. Kun muurahaiset saavuttavat lehden, tilaksi tulee ”kantaminen”, jolloin ne raahaavat lehden kekoon. Kun lehti on raahattu kekoon, tilaksi tulee uudestaan ”ympäristön tutkiminen”. Jos keon lähelle eksyy hämähäkki, muurahaisten tilaksi tulee ”puolustaminen”, jolloin kaikki lähellä olevat muurahaiset hyökkäävät hämähäkin kimppuun. Kun hämähäkki on kukistettu, tilaksi tulee ”kantaminen” ja muurahaiset kantavat raadon kekoon. Jos hämähäkki pääsee karkuun, tilaksi tulee uudestaan ”tutkiminen”. (McGugan 2007, 142–163.) Näinkin yksinkertaisella tilakoneella saadaan simuloitua hetkellisesti tarkkailtuna melkein luonnollisesti käyttäytyviä muurahaisia.

Tilakoneet ovat käytännöllinen ja helppo tapa luoda päätöksenteko videopelin tekoälylle, koska tilakone pilkkoo monimutkaisen rakenteen pieniin palasiin, jotka on helppo toteuttaa. Tilakoneita ei ole vaikea suunnitella, koska ihmisillä on tapana kuvitella, mitä muut ihmiset tai eläimet ajattelevat tehdessään jotain. Ei ole tosin käytännöllistä muuttaa jokaista ajatusta ohjelmakoodiksi, sillä videopelin tekoälyn toteuttamiseksi tarvitaan vain normaalia käyttäytymistä läheisesti muistuttava käyttäytyminen. (McGugan 2007, 164.)

4.2 Tavoitekartta

Tekoälyn suorittama päätöksenteko voidaan toteuttaa myös siten, että tekoäly tutkii lähiympäristöään, ja päättelee ympäristön perusteella seuraavan toimintonsa luonteen. Pelihahmo voi esimerkiksi huomata edessään olevan räjähtämässä olevan dynamiitin ja lähteä juoksemaan lähintä suojaa kohti. Tämänkaltaisen päätöksentekomenetelmän voi luoda esimerkiksi käyttämällä tavoitekarttaa.

Tavoitekartalla (priority map, priority grid) tarkoitetaan esimerkiksi ruudukkoon jaettua graafia, joka kuvaa pelihahmon lähiympäristön mielenkiintoisuutta ja vaarallisuutta. Jos pelihahmon lähellä on ruutu, joka kuvaa vaarallisuutta, pelihahmo pyrkii tästä ruudusta poispäin. Jos pelihahmo huomaa ruudun, joka on mielenkiintoinen, se pyrkii kulkemaan tätä ruutua kohti.



Kuvio 13: Esimerkki tavoitekartasta.

Kuvio 13 on esimerkki peliympäristöä mallintavasta tavoitekartasta. Kuviossa negatiivisen arvon sisältävät ruudut ovat esteitä ja positiivisen arvot sisältävät ruudut mielenkiintoisia kohteita. Pelihahmo tutkii vieressään olevat ruudut ja

liikkuu aina omalla kohdallaan olevan ruudun arvoa suuremman arvon sisältävään ruutuun. Tällä tavalla pelihahmo saavuttaa päämäärän vain ympäristöään tutkimalla.

Siinä missä tilakone on hyvä tapa toteuttaa tekoälyn käyttäytymismallit, tavoitekarttaa voidaan tilakoneeseen yhdistettynä käyttää toteuttamaan tekoälyn päätökset pienessä mittakaavassa. Esimerkiksi pelihahmon ollessa nälkäinen, tilakoneen avulla voitaisiin vaihtaa pelihahmon prioriteetteja siten, että hahmo alkaa etsiä ympäristöstä jotain syötävää. Tämän jälkeen pelihahmolle voitaisiin tavoitekartan avulla kertoa, missä mahdollinen ruoka on ja mihin ei ole turvallista mennä.

5 TEKOÄLYN TOTEUTTAMINEN VIDEOPELISSÄ

5.1 Käytetyt työvälineet ja menetelmät

Videopeli toteutettiin *Game Maker 8* -ohjelmistolla. *Game Maker 8* on kaksiulotteisten videopelien kehittämiseen suunnattu ohjelma, joka sisältää työkalut kuvien muokkaamiseen ja ohjelman sisäisen skriptin luomiseen. Ohjelman toimintaperiaate on yksinkertainen: siihen lisätään tarvittavat resurssit (kuvat, äänet, fontit ja skriptit) sekä luodaan pelin sisäiset entiteetit ja linkitetään niihin resurssit.

Esimerkiksi luotaessa pelihahmona toimivaa virtuaalista autoa, ensimmäiseksi luodaan autoa esittävä kuva kuvankäsittelyohjelmalla ja ladataan se ohjelmaan. Tämän jälkeen luodaan varsinainen entiteetti ohjelman sisälle, johon linkitetään juuri ladattu kuva. Lopuksi autolle kirjoitetaan skripti, joka saa auton liikkumaan tietokoneen ruudulla.

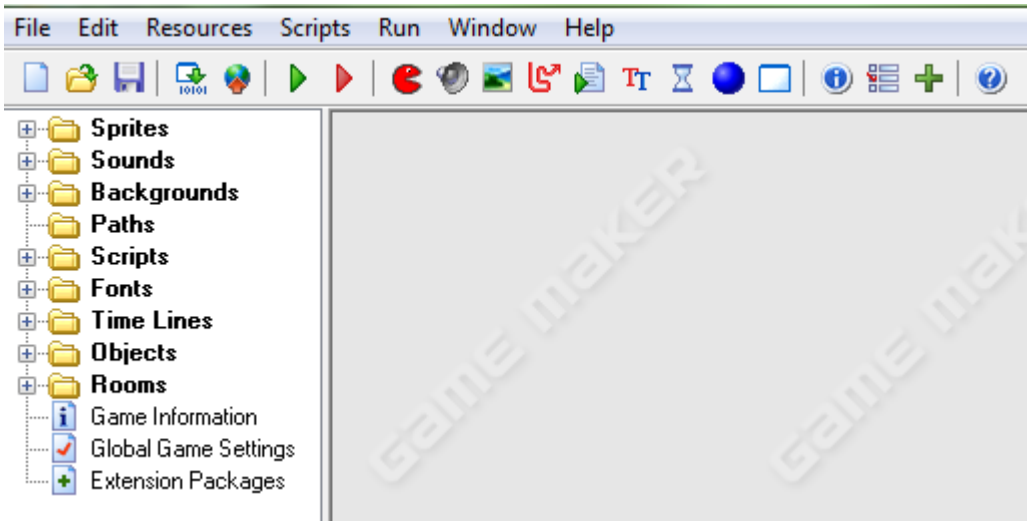
Ohjelman sisäinen skriptikieli on nimeltään *GML* (Game Maker Language), joka on *heikosti tyypitetty* kieli. *Heikosti tyypitetty* tarkoittaa sitä, että skriptissä esiintyville muuttujille ei tarvitse määrittää tietotyyppiä. Skriptiä kirjoittaessa voidaan myös valita, lisätäänkö jokaisen komennon sisältävän rivin perään merkki ”;”, kuten esimerkiksi *C-ohjelmointikielessä*.

Seuraavassa skriptissä on esimerkki *GML*:n syntaksista:

```
1. action = "";
2. next_action = "";
3. …
4. if (next_action != "") {
5.     action = next_action;
6.     next_action = " ";
7. } else {
8.     action = "";
9. }
```

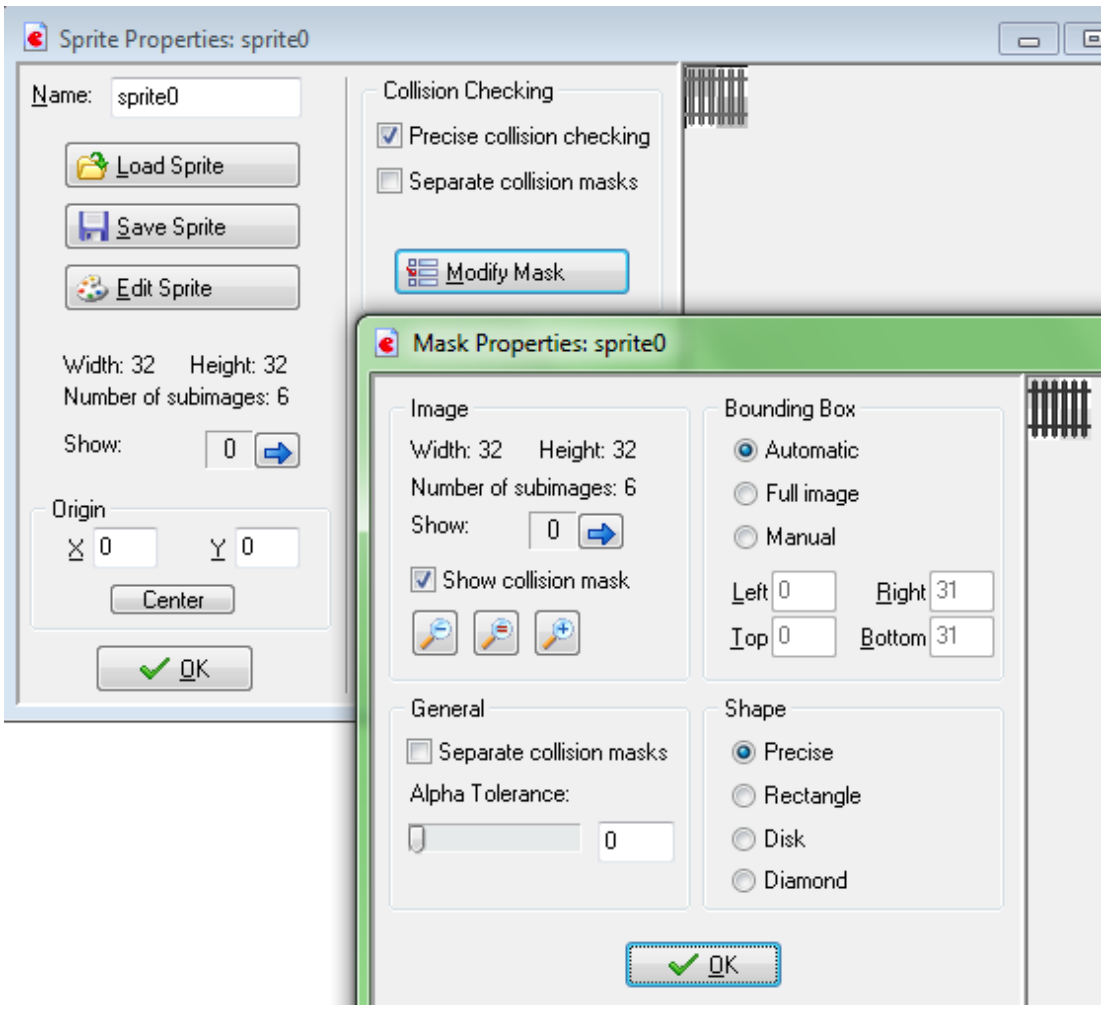
Kohdassa 1 ja 2 luodaan kaksi muuttujaa, joiden arvoiksi annetaan tyhjä merkkijono. Kohta 3 ei ole komento, vaan kirjoittamatta jätetty kohta koodissa, joka voi sisältää esimerkiksi pelin ohjelmakoodin sisäisiä komentoja, jotka käsittelevät juuri luotuja muuttujia. Kohdassa 4 ja 7 on kontrollirakenne, jonka avulla tarkistetaan muuttujan ”next_action” arvo. Jos muuttuja ei ole tyhjä merkkijono, sijoitetaan muuttujalle ”action” muuttujan ”next_action” arvo ja

muuttujalle ”next_action” arvoksi tyhjä merkkijono (kohdat 5 ja 6). Jos muuttuja ”next_action” on tyhjä merkkijono, sijoitetaan muuttujalle ”action” arvoksi tyhjä merkkijono (kohta 8). *GML* sisältää paljon valmiita komentoja, joiden avulla voidaan esimerkiksi laskea kahden pisteen välinen etäisyys tai kulma tai tarkistaa milloin kaksi objektia törmää toisiinsa.



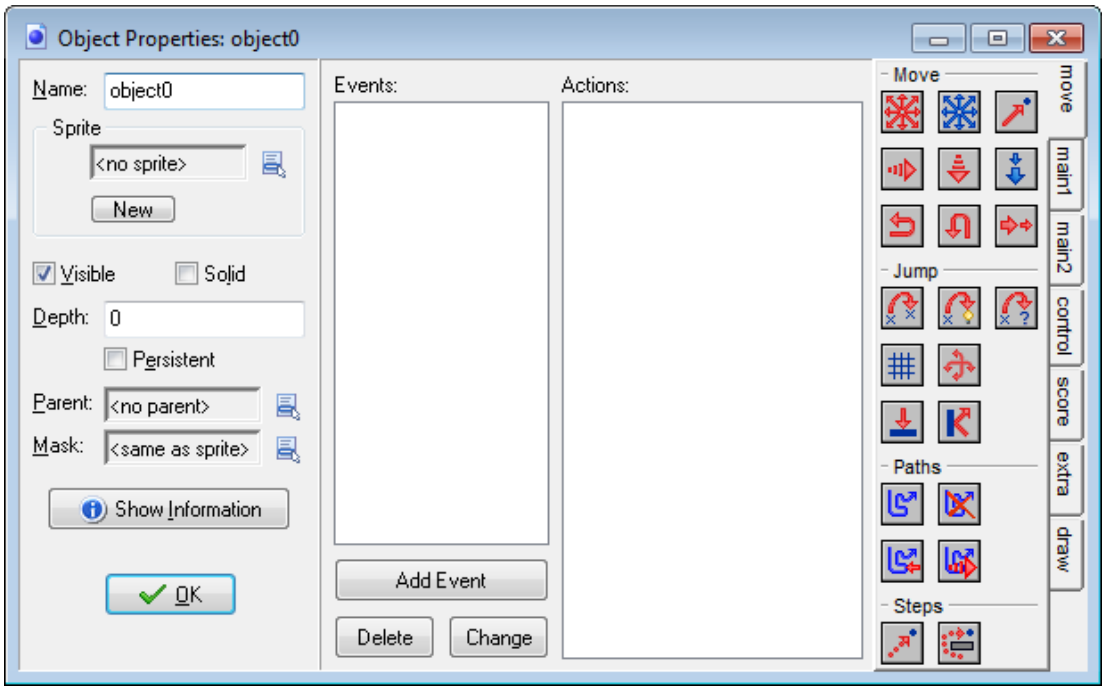
Kuvio 14: Game Maker 8 -ohjelman käyttöliittymä.

Kuvio 14 on kuvankaappaus *Game Maker 8* -ohjelman laajennetusta (advanced) päänäkymästä. Ylimmällä rivillä on päävalikko, toisella rivillä on valikko ohjelman omille toiminnoille ja vasemmassa sarakkeessa näkyy peliin ladatut resurssit sekä ohjelman avulla luodut entiteetit. Aktiiviset alaikkunat tulevat näkyviin oikeaan sarakkeeseen. Vasemmassa sarakkeessa on myös toiminnot pelin sisäisten tietojen muokkaamiseen (Game Information), pelin asetusten muokkaamiseen (Global Game Settings) ja lisäosien asentamiseen (Extension Packages). Lisäosien avulla ohjelmaan voidaan lisätä toiminnallisuuksia esimerkiksi DLL-tiedostoja käyttämällä.



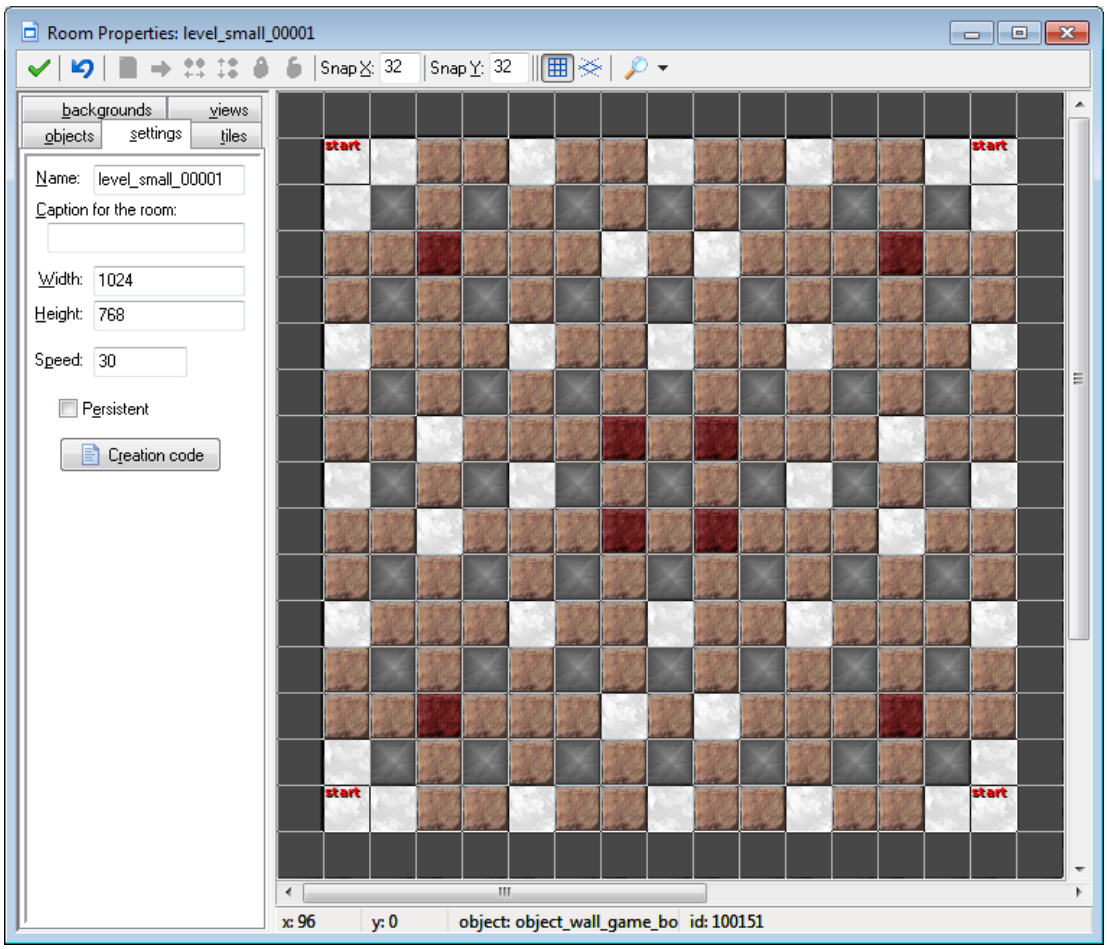
Kuvio 15: Game Maker 8 -ohjelman sprite-editorin käyttöliittymä.

Kuvio 15 on kuvankaappaus *Game Maker 8* -ohjelman sisäisestä sprite-editorista (Sprite Editor). Editorin avulla voidaan luoda ohjelmassa käytettäviä spritejä, jotka sisältävät kuvan tai kuvasarjan, sekä muokata kuvaan liittyviä tietoja. Kuvan voi ladata ohjelmaan (Load Sprite) tai luoda ohjelman avulla (Edit Sprite). Kuvaan liittyviä tietoja ovat muun muassa kuvan keskipiste (Origin) sekä törmäyslaskentatiedot (Collision Checking). Törmäyslaskentaan liittyen kuvaan voi liittää tiedot törmäyslaskennassa huomioitavasta alueesta, maskista (Mask). Maski voi olla kuvan muotoinen (Precise), neliö (Rectangle), pyöreä (Disk) tai timantin muotoinen (Diamond). Maskille voidaan määrittää reunat (Bounding Box) automaattisesti tai manuaalisesti.



Kuvio 16: Game Maker 8 -ohjelman objektieditorin käyttöliittymä.

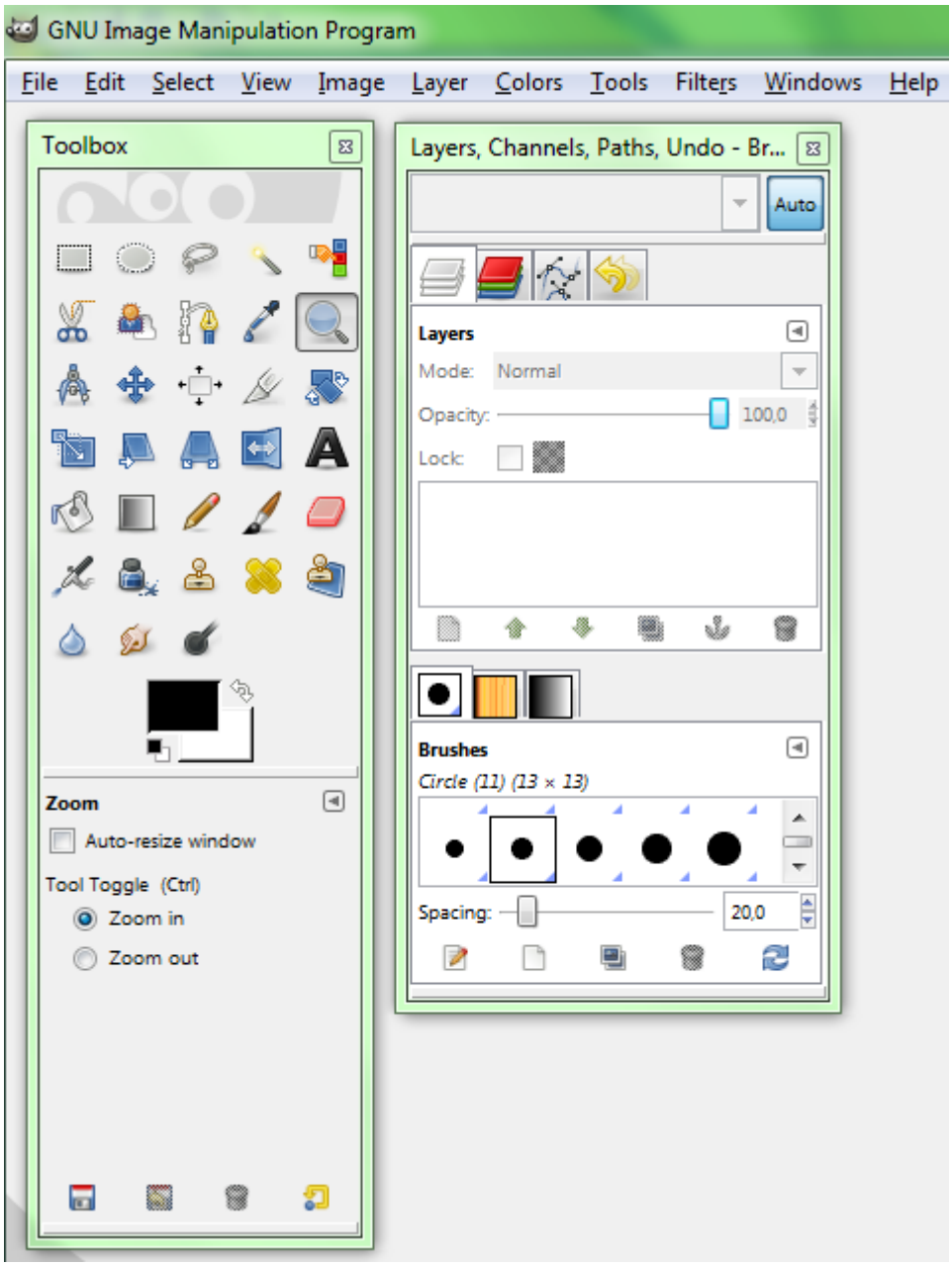
Kuvio 16 on kuvankaappaus *Game Maker 8* -ohjelman sisäisestä objektieditorista (Object editor). Editorin avulla ohjelmaan voidaan luoda sen sisäisiä entiteettejä, objekteja. Objektin päätarkoitus on toimia pelaajalle näkyvänä entiteettinä, kuten esimerkiksi pelin sisäisenä pelihahmona. Siihen voidaan liittää sprite sekä määrittää onko se näkyvä (Visible), kiinteä (Solid) ja pysyvä (Persistent). Näkyvä objekti on näytöllä näkyvä, piirrettävä objekti. Kiinteä objekti tarkoittaa objektia, joka huomioidaan pelin sisäisissä laskuissa kiinteänä, kosketeltavana objektina. Pysyvät objektit ovat objekteja, jotka eivät tuhoudu pelikenttien välillä siirryttäessä. Objektille voidaan myös määrittää syvyys (Depth) joka määrittää sen, missä vaiheessa objekti piirretään näytölle. Esimerkiksi syvyys 1 piirretään syvyyden 2 päälle. Objektiin voidaan liittää tapahtumia (Events), joiden seurauksena suoritetaan toimintoja (Actions). Esimerkiksi pelin käynnistyessä objekti voisi suorittaa resurssien ja asetusten lataamisen.



Kuvio 17: Game Maker 8 -ohjelman kenttäeditorin käyttöliittymä.

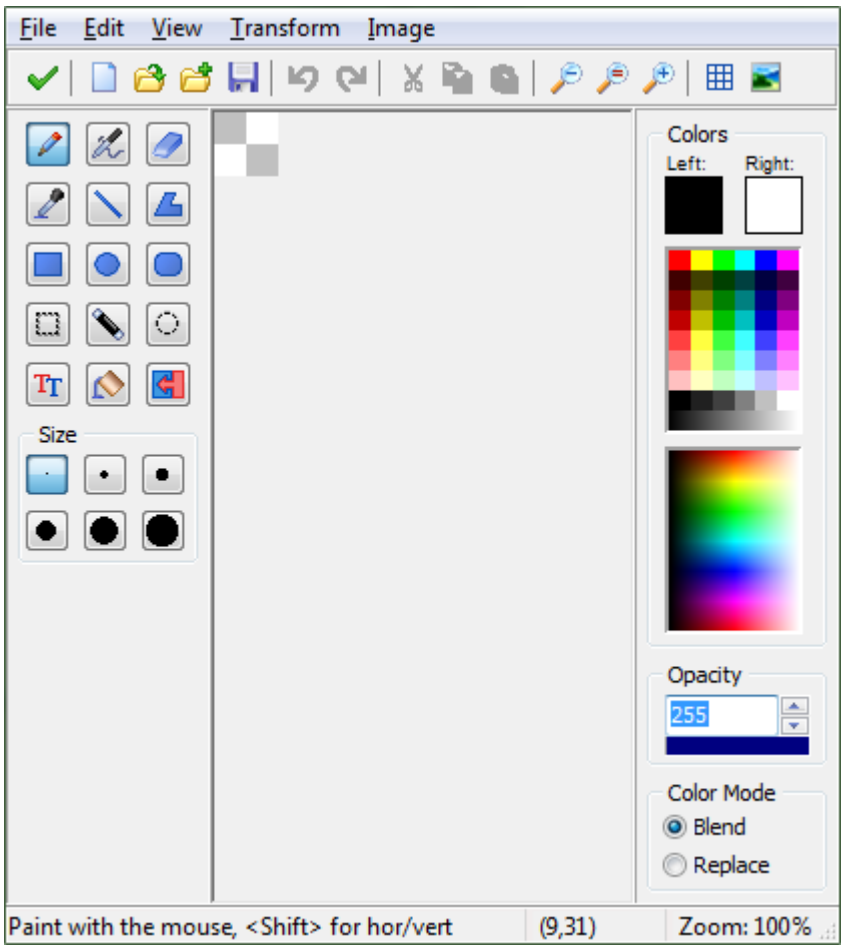
Kuvio 17 on kuvankaappaus *Game Maker 8* -ohjelman sisäisestä kenttäeditorista. Kenttäeditorin avulla voidaan luoda ja muokata pelin sisäisiä pelikenttiä ja valikon näkymiä. Pelikenttä on se alue, jossa kaikki pelin toiminta tapahtuu. Pelikenttä voi olla esimerkiksi jalkapallopelin kenttä tai autopelin kilparata. Pelikentälle voidaan määrittää muun muassa koko (Width, Height) sekä siihen voidaan liittää objekteja, taustakuvia ja skriptejä. Valikot ovat ikkunoita tai näkymiä, joiden avulla pelin sisäisiä valintoja voidaan tehdä. Esimerkiksi uuden ottelun aloittaminen tehdään otteluvalikosta käsin.

Videopelin grafiikka toteutettiin luomalla pikseligrafiikkaa *GIMP 2* -ohjelmalla (GNU Image Manipulation Program) ja *Game Maker 8* -ohjelman sisäisellä kuvaeditorilla. *GIMP* on ilmainen, avoimen lähdekoodin kuvankäsittelyohjelma, joka toimii usealla alustalla. Se on suunniteltu erityisesti digitaalisten valokuvien ja Internet-sivujen grafiikan muokkaamiseen, mutta soveltuu myös muun kaksiulotteisen grafiikan luomiseen. (Peck 2008, 1–2.)



Kuvio 18: GIMP 2 -ohjelman käyttöliittymä.

Kuvio 18 on kuvankaappaus *GIMP 2* -ohjelman työkalupalkista (Toolbox), taso-, kanava- ja polkuikkunasta (Layers, Channels, Paths, Undo) sekä pääikkunasta. Muokattava kuva näkyy pääikkunassa, joka on tilan säästämiseksi siirretty kahden muun ikkunan alle. Työkalupalkki sisältää yleisimmät työkalut, kuten esimerkiksi valintatyökalut ja piirtotyökalut. Taso-, kanava- ja polkuikkuna sisältää tiedot ja työkalut kuvien tasoihin, kanaviin ja polkuihin liittyen sekä pikavalinnat aktiiviselle työkalulle. Pääikkunasta voidaan valita edistyneempiä työkaluja, kuten esimerkiksi kuhmutus (Bumb Map) ja animaatiotyökalut (Animation).



Kuvio 19: Game Maker 8 -ohjelman kuvaeditorin käyttöliittymä.

Kuvio 19 on kuvankaappaus *Game Maker 8* -ohjelman sisäisestä kuvaeditorista. Editorin avulla voidaan luoda nopeasti yksinkertaisia kuvioita, mutta se ei sisällä edistyneempiä muokkausominaisuuksia, kuten esimerkiksi kuhmutusta tai valokuvan muokkaamiseen tarkoitettuja työkaluja.

Toteutetun videopelin grafiikan voi jakaa kahteen osa-alueeseen, spriteihin ja taustakuviin. Sprite on yksittäinen, yleensä osittain läpinäkyvä kuva joka edustaa pelissä olevaa entiteettiä, kuten esimerkiksi pelihahmoa tai seinäkappaletta. Taustakuva on kaiken muun grafiikan taustalla näkyvä kuva, kuten esimerkiksi pelialueen lattia, jos peli on kuvattu ylhäältä päin.

Kuvio 20: Esimerkki sprite-grafiikasta.

Kuviossa 20 on esimerkki pelissä esiintyvästä sprite-grafiikasta. Kuvion sprite näytetään niiden pelihahmojen nimien vieressä, jotka eivät selviytyneet pe-

lisessiosta tuhoutumatta. Kuviossa 20 näkyvä sprite on osittain läpinäkyvä, joten se voidaan piirtää suoraan taustakuvan päälle, ilman että sen vieressä näkyisi suorakaiteen muotoista valkoista aluetta.

Kuvio 21: Esimerkki taustakuvasta.

Kuviossa 21 esitettyä taustakuvaa käytetään videopelin peliympäristön lattian kuvaamiseen. Kuva piirretään peliympäristön pohjalle niin monta kertaa vierekkäin ja päällekkäin, että se peittää koko tietokoneen ruudulla näkyvän pelialueen. Kuvan reunat on tehty yhteneviksi siten, että sen ollessa piirrettynä itsensä kanssa vierekkäin, reunat eivät erotu lopputuloksesta.

5.2 Päämäärä ja elementit

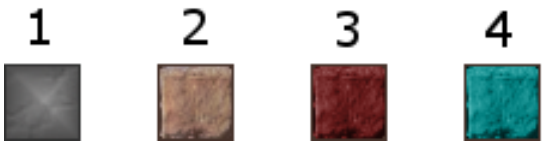
Toteutettu videopeli on kaksiulotteinen, usean yhtäaikaisen pelaajan toimintapeli ja sen esikuvana toimii *Hudson Softin* vuonna 1983 julkaisema videopeli *Bomberman*. Pelin pelaajien päämääränä on eliminoida muut pelissä esiintyvät pelaajien pelihahmot käyttäen peliympäristössä sijaitsevia seiniä suojina, keräämällä pelihahmon ominaisuuksia parantavia esineitä ja asettamalla maahan itsekseen räjähtäviä pommeja. Yhdessä pelisessiossa voi pelata 2–8 pelaajaa, jotka voivat olla joko ihmispelaajia tai tekoälyn ohjaamia pelaajia. Ihmispelaaja voi ohjata pelihahmoaan näppäimistön, hiiren tai peliohjaimen avulla. Videopeliin ei toteutettu verkkopelitoimintoa, eli kaikkien ihmispelaajien on pelattava saman tietokoneen ääreltä. Videopelin käyttöliittymä sekä pelin sisäiset ohjeet ja viestit on toteutettu englanniksi.

Pelissä esiintyy neljä täysin erilaista entiteettiä: pelihahmot, seinät, pommit ja esineet. Pelihahmot esittävät peliympäristössä liikkuvia pelaajia, jotka voivat liikkua alueilla, joissa ei esiinny seiniä sekä pudottaa maahan pommeja ja kerätä esineitä. Pelihahmoilla on kolme ominaisuutta: nopeus, pommin voimakkuus ja pudotettujen pommien enimmäismäärä. Nopeus määrittää pelihahmon liikkumisnopeuden peliympäristössä. Pommin voimakkuus määrittää maahan pudotettavan pommin räjähdyksen enimmäisetäisyyden. Pudotettujen pommien enimmäismäärä määrittää, kuinka monta pommia kukin peli-

hahmo voi pitää yhtäaikaisesti maahan pudotettuna. Tämän lisäksi pelihahmot voivat saada ominaisuuden pommin heittämiseen ja potkaisemiseen. Pommin potkaiseminen saa aikaan pommin liikkumisen potkaisijasta poispäin, kunnes vastaan tulee pelihahmo, seinä tai toinen pommi. Pommin heittäminen siirtää pommin joko yhden seinäkappaleen ylitse, jos pelihahmon edessä on seinä, tai tietyn matkan päähän, jos edessä oleva peliympäristö on tyhjä seinistä.

Kuvio 22: Videopelissä esiintyvän pelihahmon animaatio kuvasarjana.

Kuviossa 22 on esitetty pelissä esiintyvän pelihahmon animaatio kuvasarjan avulla. Jokainen pelihahmo käyttää pohjanaan samaa kuvasarjaa, mutta kuvat väritetään pelihahmosta riippuen eri värillä, jotta pelaajat erottaisivat oman pelihahmonsa muiden joukosta. Kuvasarja esittää ylhäältä päin kuvattua ihmistä.



Kuvio 23: Videopelissä esiintyvien seinäkappaleiden grafiikka.

Seinät ovat läpipääsemättömiä alueita pelikentällä, ja niitä on neljää erilaista tyyppiä, jotka on esitetty kuviossa 23: särkymättömiä (kohta 1), normaaleja särkyviä (kohta 2), särkyviä joista saa aina esineen (kohta 3) ja yhtäaikaisesti särkyviä (kohta 4). Särkymättömät seinät eivät tuhoudu koskaan. Kaikkia särkyviä seiniä voidaan tuhota pommien avulla, jolloin niistä ilmestyy tietyn prosenttiluvun perusteella pelikentälle esine. Kun yhtäaikaisesti särkyvä seinä tuhoutuu, tuhoutuvat myös kaikki muut samanlaiset seinät.

Kuvio 24: Videopelissä esiintyvän pommin animaatio kuvasarjana.

Pommi on pelaajan maahan pudottama ase, joka räjähtää tietyn ajan päästä pudottamisesta. Kuvio 24 kuvaa pommin animaatiota kuvasarjan avulla. Pommi räjähtää animaation saavuttaessa loppunsa, jolloin siitä lähtee nel-

jään pääilmansuuntaan liekki, joka tuhoaa eteensä tulevat pelihahmot, esineet ja seinät. Jos liekki osuu seinään, sen kulku pysähtyy. Muussa tapauksessa liekki pysähtyy vasta sen voimakkuuden loputtua. Pommin räjähdystä ja pommin liekkiä kuvaavat animaatiot luodaan *Game Maker 8* -ohjelmiston avulla ohjelman ajon aikana.

Kuvio 25: Videopelissä esiintyvien esineiden grafiikka.

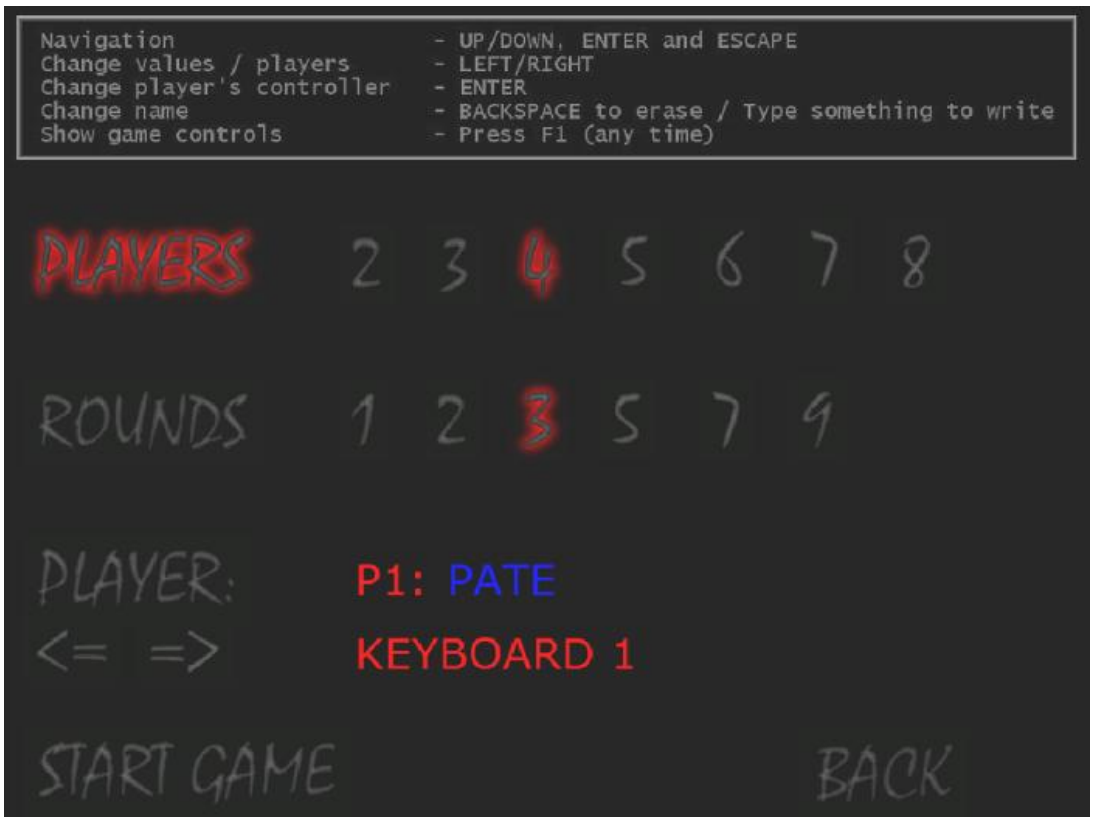
Esineet ovat pelikentälle ilmestyviä pelihahmojen ominaisuuksia parantavia objekteja. Kuviossa 25 on kuvattu pelissä esiintyvien esineiden grafiikat: nopeutta lisäävä esine (kohta 1), pommien voimakkuutta lisäävä esine (kohta 2), yhtäaikaisten pudotettujen pommien lukumäärää nostava esine (kohta 3), potkuominaisuuden antava esine (kohta 4), heitto-ominaisuuden antava esine (kohta 5) ja lahja (kohta 6). Lahja parantaa jotain pelaajan ominaisuutta sattumanvaraisesti seuraavassa pelisessiossa. Esineitä ilmestyy tuhotuista seinistä ja pelihahmoista sekä pelisession loppuvaiheessa niitä ilmestyy pelikentälle tyhjästä ottelun nopeuttamiseksi. Esine tuhoutuu, kun pelihahmo on kerännyt sen.

5.3 Käyttöliittymä

Videopeliin toteutettiin moniosainen valikko sekä käyttöliittymä pelikentän valitsemiseen ja ottelun aikaisen tiedon esittämiseen. Valikko koostuu päävalikosta, otteluvalikosta, ottelua valmistelevasta valikosta, asetusvalikosta ja kiitokset sisältävästä valikosta. Asetusvalikkoa ja kiitokset sisältävää valikkoa ei käsitellä tässä työssä.

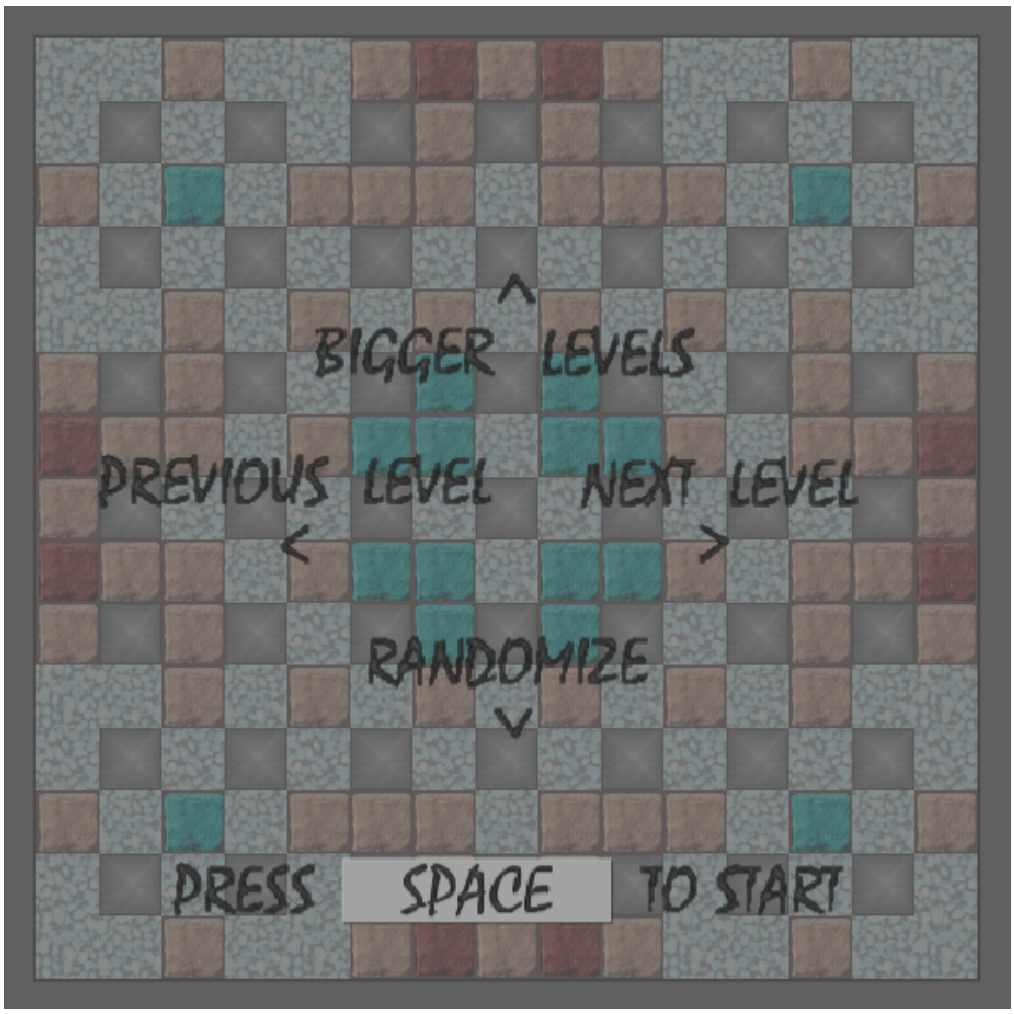
Kuvio 26: Videopelin päävalikko.

Kuvio 26 on kuvankaappaus videopeliin toteutetusta päävalikosta. Päävalikosta päästään otteluvalikkoon (New Game), asetuksiin (Options), kiitoksiin (Credits) ja takaisin käyttöjärjestelmään (Quit Game). Valikon alaosassa on ohjeet valikossa navigoimiseen. Valikon kuvankaappaus on esitetty pienennetyssä mittakaavassa suuren kokonsa vuoksi.



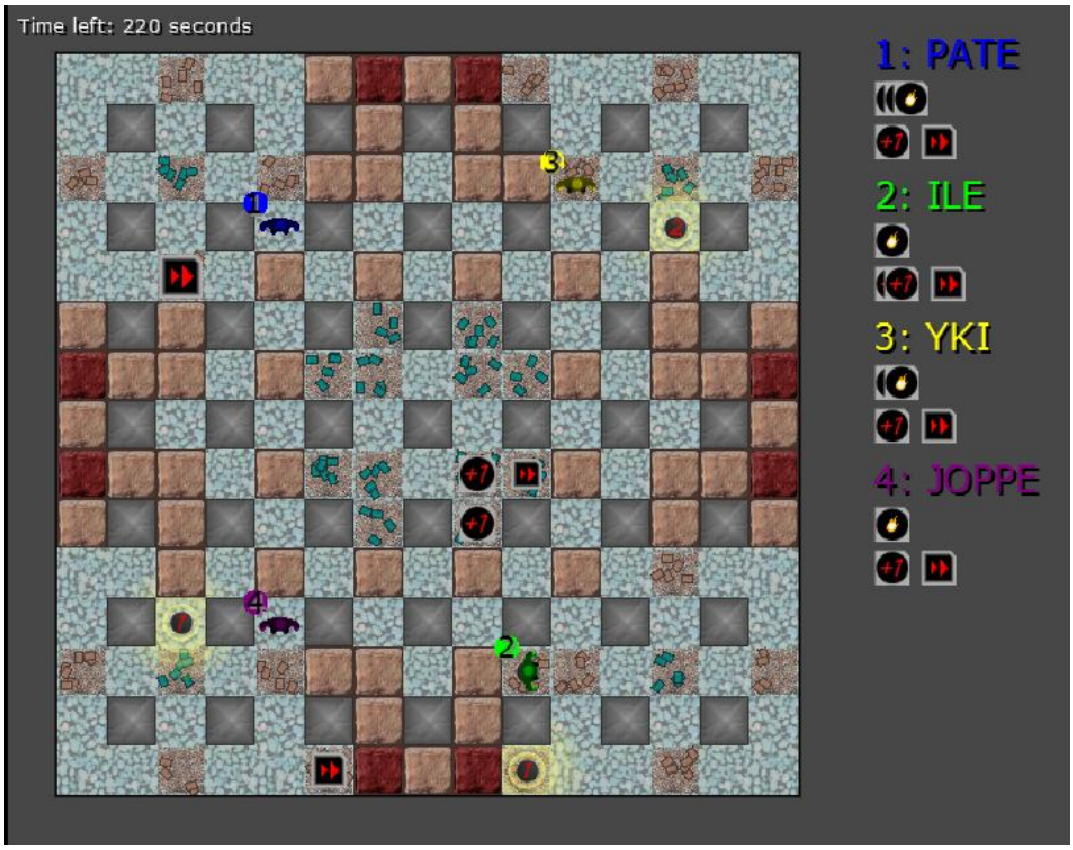
Kuvio 27: Videopelin otteluvalikko.

Kuvio 27 on kuvankaappaus videopeliin toteutetusta otteluvalikosta. Otteluvalikossa valitaan pelaajien lukumäärä (Players), otteluiden määrä ottelusarjassa (Rounds) sekä pelaajien nimi ja ohjainlaitteisto (Player). Otteluvalikosta päästään otteluun (New Game) ja takaisin päävalikkoon (Back). Valikon yläosassa on ohjeet valikossa navigoimiseen ja näppäimet yleisille toiminnoille. Valikon kuvankaappaus on esitetty pienennetyssä mittakaavassa suuren kokonsa vuoksi.



Kuvio 28: Videopelin ottelua valmisteleva valikko.

Kuvio 28 on kuvankaappaus peliin toteutetusta ottelua valmistelevasta valikosta. Valikosta voidaan valita ottelussa käytettävä pelikenttä (Previous Level, Next Level) tai satunnainen kenttä (Randomize) sekä vaihtaa kentän kokoa (Bigger Levels). Valikon kuvankaappaus on esitetty pienennetyssä mittakaavassa suuren kokonsa vuoksi.



Kuvio 29: Videopelin ottelun aikainen käyttöliittymä.

Kuvio 29 on kuvankaappaus ottelun aikaisesta käyttöliittymästä. Ottelun aikalaskuri näkyy vasemmassa yläreunassa (Time left), ottelun osanottajat, pelaajat, ovat listattuina oikeassa reunassa ja pelikenttä on sijoitettu keskelle. Pelaajien tiedoissa näkyvät pelaajan nimi ja kerätyt esineet. Ottelusta päästään otteluvalikkoon painamalla ESC-näppäintä. Käyttöliittymän kuvankaappaus on esitetty pienennetyssä mittakaavassa suuren kokonsa vuoksi.

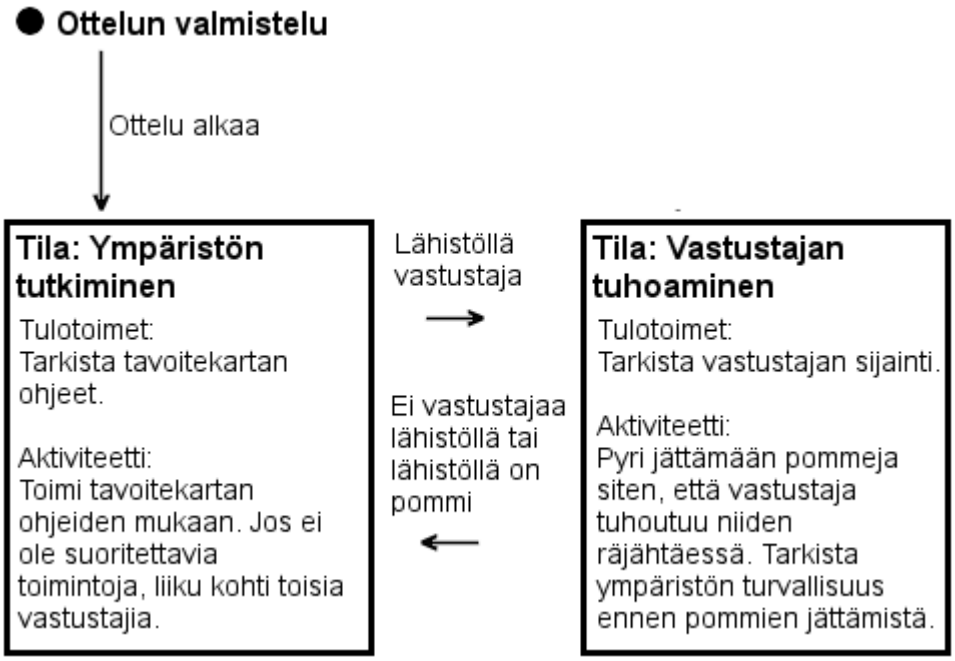
5.4 Tekoälyn päätöksenteko

Toteutetun videopelin tekoälyn päämääränä on toimia ihmispelaajaa läheisesti vastaavalla tavalla. Tekoälyn on tarkoitus tuhota edessä olevat seinänkappaleet pommien avulla, kerätä lähellä olevat esineet maasta ja pyrkiä asettamaan maahan pommeja siten, että vastustajien pelihahmot tuhoutuisivat pommien räjähtäessä. Tekoälyn on myös tarkoitus tutkia paikan turvallisuus ennen pommin asettamista ja suojautua vastustajien pelihahmojen jättämiltä pommeilta sienien taakse. Tutkiessaan alueen turvallisuutta tekoäly pyrkii myös olemaan tuhoamatta lähistöllä olevia esineitä.

Yllä kuvattujen toimintojen perusteella voidaan luoda lista, minkä perusteella tekoäly asettaa toiminnot tärkeysjärjestykseen tehdessään päätöksiä:

1. Tutki, yltääkö jonkin pommin räjähdysalue omalle kohdalle. Jos näin on, pyri suojautumaan seinien taakse.
2. Jos lähellä on vastustajan pelihahmo, tutki alueen turvallisuus ja aseta pommi siten, että vastustajan pelihahmo saattaisi tuhoutua pommin räjähtäessä.
3. Kerää lähellä olevat esineet.
4. Jos lähellä on seinänkappale, tutki alueen turvallisuus ja aseta pommi siten, että seinänkappale tuhoutuu pommin räjähtäessä.
5. Siirry kohti muita pelihahmoja.

Jos pelikentällä on enemmän kuin yksi pelihahmo, tekoälyn ohjaamalla pelihahmolla on aina toteutettavana jokin kohta listasta. Jos pelihahmoja on jäljellä vain yksi, ottelu on päättynyt.

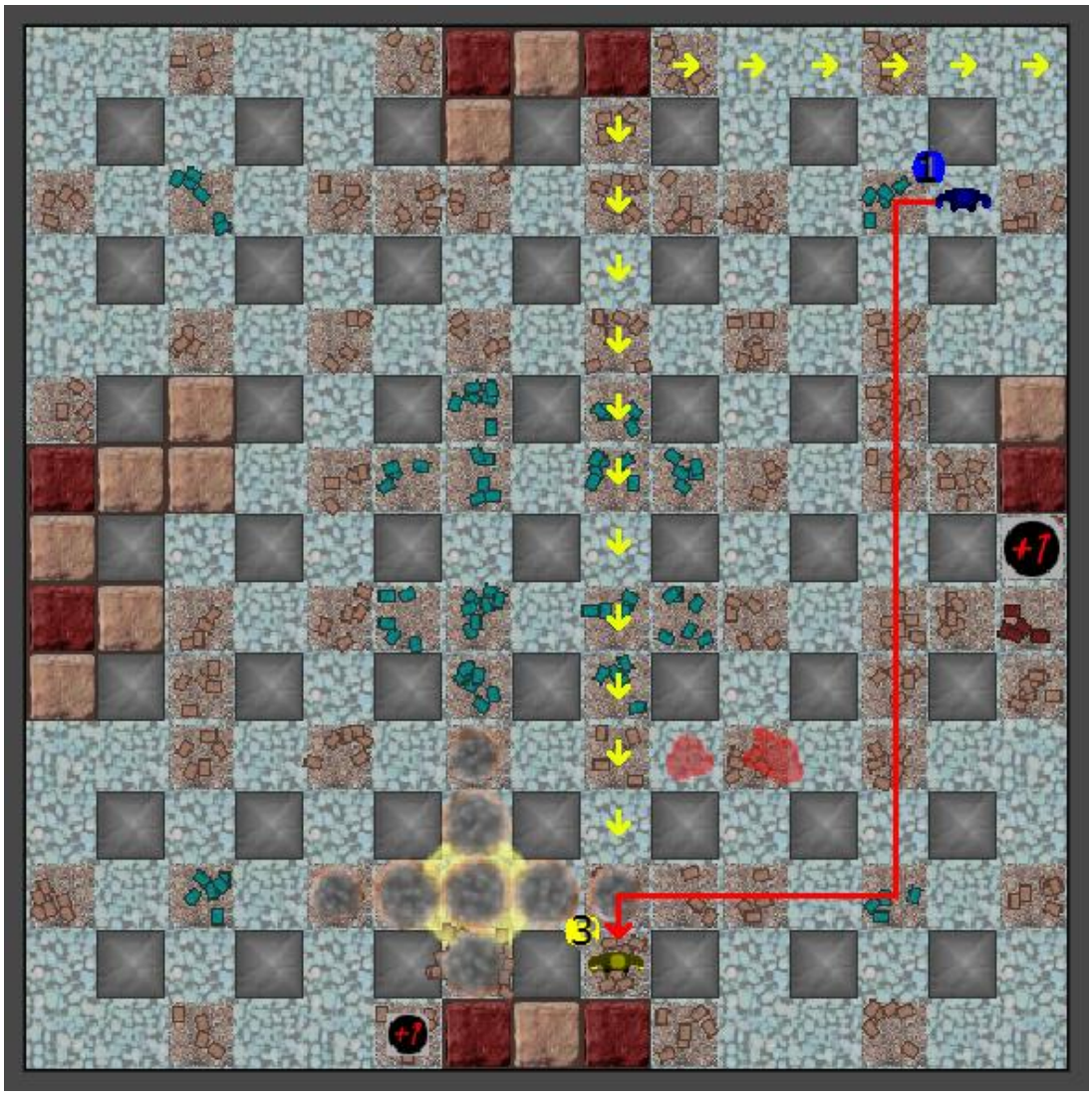


Kuvio 30: Videopeliin toteutettu tilakone.

Tekoälyn päätöksenteko toteutettiin yhdistämällä tilakoneen ja tavoitekartan ominaisuudet. Kuviossa 30 kuvattu tilakone selvittää, onko tekoälyn ohjaaman pelihahmon lähellä muita pelihahmoja. Jos on, tekoäly pyrkii tuhoamaan lähistöllä olevat pelihahmot pommien avulla. Pelihahmojen, esineiden ja seinänkappaleiden paikat ilmoitetaan tekoälylle tavoitekartan avulla. Jos tekoäly ei saa ohjeita tilakoneelta, se toimii tavoitekartan ohjeiden mukaisesti. Ainoastaan vaarallisella etäisyydellä olevien pommien aiheuttama suojautuminen menee edelle tilakoneen antamista käskyistä.

5.5 Reitinhaku ja tavoitekartta

Videopeliin toteutettiin *A*-algoritmia* käyttävä reitinhaku, jonka tarkoitus on ilmoittaa tekoälylle mihin suuntaan sen on liikuttava, päästäkseen lähelle vastustajien pelihahmoja. Videopelin peliympäristön yksinkertaisuuden vuoksi muiden entiteettien, kuten seinänkappaleiden ja esineiden etsimiseen ei ole tarvetta käyttää reitinhakualgoritmia, vaan niiden paikka on ilmoitettu tekoälylle tavoitekartan avulla.



Kuvio 31: Videopelissä sovellettu reitinhaku ja tavoitekartta.

Kuviossa 31 on esitetty, miten reitinhakua ja tavoitekarttaa on sovellettu toteutetussa videopelissä. Reitinhaun avulla tekoäly määrittää, mihin suuntaan sen on liikuttava saavuttaakseen sopivan etäisyyden vastustajan pelihahmon tuhoamiseen. *A*-algoritmin* avulla etsittyä reittiä kuvataan yhtenäisellä punaisella nuolella. Tätä reittiä etsitään vain silloin, kun pelihahmo ei saa selville

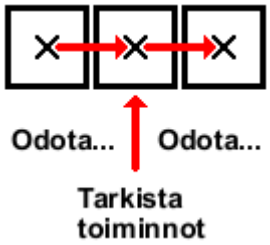
tavoitekartan avulla, mitä sen pitäisi tehdä. Kuviossa on esitetty keltaisten nuolten avulla seinänkappaleen paikka siten, kuin se on tekoälyn käyttämässä tavoitekartassa esitetty. Jos tekoälyn pelihahmon paikasta on suora näköyhteys keltaiseen nuoleen, pelihahmo suunnistaa tätä nuolta kohti. Joissakin tapauksissa pelihahmon kohdalta ei ole suoraa näköyhteyttä keltaiseen nuoleen, vaikka kohde olisi aivan vieressä. Tällöin tekoäly etsii reitinhaulla suunnan vastustajan pelihahmoa kohti. Liikkuessaan uuteen suuntaansa tekoälyn pelihahmo saa yleensä näköyhteyden piilossa olleeseen kohteeseensa, jolloin ylempänä kuvattu ongelmatilanne on ratkaistu. Tekoälyn päätöksentekoon on lisätty satunnaistekijä, jonka ansiosta tekoäly saattaa päätöstä tehdessään odottaa pienen hetken tekemättä mitään. Tämä saa tekoälyn tuntumaan vähemmän konemaiselta ja lisää tekoälyn erehtymismahdollisuutta.

Tavoitekarttojen päivittäminen on tietokoneen suorituskyvyn kannalta raskasta, jos sitä tehdään jatkuvasti. Tämän takia niitä on pyritty päivittämään vain silloin, kun se on oleellista videopelin toiminnan kannalta.

Tavoitekartat päivitetään videopelissä ainoastaan silloin, kun

- ottelu alkaa
- pommi on asetettu pelikentälle
- pommi on tuhoutumassa
- pommi on tuhoutunut
- seinänkappale on tuhoutunut
- esine on ilmestynyt pelikentälle
- esine on tuhoutunut tai pelihahmo on kerännyt sen.

Kun jokin listassa esitetty asia on tapahtunut, ohjelmalle kerrotaan, että tavoitekarttojen päivittäminen suoritetaan muutaman sekunnin sadasosan odotusajan päästä. Jos tuona odotusaikana tulee uusia samanlaisia tapahtumia, odotusaika aloitetaan alusta. Tämä vähentää tavoitekarttojen päivittämisen tarvetta esimerkiksi siinä tapauksessa, jos yhtäaikaisesti tuhoutuisi neljä seinänkappaletta. Tällöin tavoitekartat päivitetään vain kerran eikä neljä kertaa peräkkäin.



Kuvio 32: Tekoälyn toimintojen tarkistaminen pelikentällä.

Jokainen tekoälyn ohjaama pelihahmo tarkistaa suoritettavat toimenpiteensä kuvion 32 kuvaamalla tavalla, eli ainoastaan ollessaan keskellä ruudukkograafin ruutua. Ruudukkograafin avulla pelikenttä on mallinnettu yksinkertaisempaan muotoon laskutoimitusten helpottamiseksi. Suoritettaessa tekoälyn toimintojen tarkistaminen ainoastaan pelihahmon ollessa keskellä ruutua, vähennetään videopelin aiheuttamaa kuormaa tietokoneelle.

5.6 Tekoälyn käyttäytyminen

Tekoälyn käyttäytymistä voidaan seurata videopelin ottelusta otetuilla kuvankaappauksilla. Kuvankaappauksissa on vain osa pelin pelikentästä oleellisen tiedon esittämiseksi.

Kuvio 33: Tekoälyn käyttäytyminen ottelun alussa.

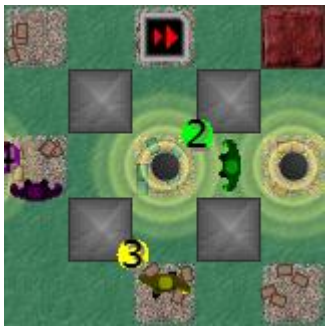
Kuviossa 33 videopelin ottelu on alkanut ja tekoälyn ohjaama pelihahmo on asettanut pommin lähimmän särkyvän seinänkappaleen vierelle, siirtyen samalla pois pommin vaikutusalueelta. Ottelun alussa pommien vaikutusalue on yleensä hyvin pieni, joten pelihahmon ei tarvitse suojautua seinien taakse ollakseen vaikutusalueen ulkopuolella. Tekoäly osaa huomioida pommien vaikutusalueen tarkasti tutkiessaan ympäristön turvallisuutta.

Kuvio 34: Tekoälyn käyttäytyminen pommien ja esineiden lähellä.

Kuviossa 34 tekoälyn pelihahmo on asettanut pelikentälle pommin ja on poimimassa pelikentältä esinettä. Tällaisessa tilanteessa tekoälyn tärkeimpänä toimintona on pommilta suojautuminen, jonka jälkeen se keskittyy vieressä olevan esineen poimimiseen.

Kuvio 35: Tekoälyn käyttäytyminen vastustajien lähellä.

Kuviossa 35 tekoälyn ohjaamat pelihahmot ovat huomanneet toisensa ja pyrkivät tuhoamaan vastustajansa. Kun tekoäly olettaa vastustajan olevan pommin vaikutusalueella, se asettaa pommin tai useita pommeja pelikentälle sellaisiin paikkoihin, joista sen on itse mahdollista päästä suojaan. Tällaisissa tilanteissa tekoälyn ohjaamat pelihahmot käyttäytyisivät liian samankaltaisesti ilman satunnaista päätöksentekoa eivätkä todennäköisesti koskaan tuhoaisi toinen toistaan. Lisäämällä päätöksentekoon pieni mahdollisuus epäröimiseen toinen tekoälyn ohjaamista pelihahmoista tekee lopulta virheen ja tuhoutuu.



Kuvio 36: Tekoälyn kukistama vastustaja.

Kuviossa 36 on esimerkki siitä, miten tekoäly joskus onnistuu kukistamaan vastustajansa. Vihreä pommien ympäröimä pelihahmo ei pääse tilanteestaan tuhoutumatta, ellei sillä ole esineistä saatua ominaisuutta, jonka avulla se voisi potkaista pommia eteenpäin. Tämänkaltainen tulos ei johdu siitä, että tekoäly suunnittelisi ympäröivänsä vastustajansa pommeilla, vaan tekoälyn

satunnaisen epäröinnin ja ympäristön tarkkailun puutteiden vuoksi se ei aina osaa arvioida kulkureittinsä turvallisuutta, ennen kuin on liian myöhäistä.

Kuvio 37: Videopelin ottelun pitkittyminen.

Kuviossa 37 on kuvankaappaus koko pelikentästä ottelun ajalta, jolloin jäljellä on vain kaksi pelihahmoa. Tällaisessa tilanteessa tekoälyn ohjaamat pelihahmot keskittyvät melkein yksinomaan vastustajansa tuhoamiseen, keräten eteensä tulevat pelikentällä sijaitsevat esineet. Joissakin tapauksissa tämä tilanne pitkittyy ja ottelu loppuu vasta aikalaskurin (Time left) näyttäessä nollaa. Johtuen pelikentän avoimuudesta ja yksinkertaisuudesta ottelun loppuvaiheessa kaksi tekoälyn ohjaamaa pelihahmoa taistelee hyvin samankaltaisesti, vaikka niiden päätöksenteossa olisikin satunnaisesti epäröintiä.

6 YHTEENVETO JA JATKOKEHITYS

Opinnäytetyön tavoitteena oli perehtyä tekoälyn historiaan sekä selittää tekoälyn merkitys videopeleissä. Tavoitteena oli myös antaa esimerkkejä tekoälyn ja reitinhaun soveltamisesta kaksiulotteisissa videopeleissä sekä kehittää tekoälyn ohjaama ihmispelaajan korvike usean yhtäaikaisen pelaajan mahdollistavaan kaksiulotteiseen videopeliin.

Tekoälyn historiaa, reitinhakua ja tekoälyn määritelmää käsitteleviä teoksia löytyi helposti niin kirjastosta kuin Internet-lähteistäkin. Tekoälyn merkitys videopeleissä oli vaikeampi selittää kuin tekoälyn yleinen määritelmä aiheen uutuuden ja lähteiden vaikean löydettävyyden vuoksi.

Tekoälyn ohjaama ihmispelaajan korvike onnistui kokonaisuudessaan hyvin. Toteutettu tekoäly osasi tuhota edessä olevat seinänkappaleet turvallisesti ja poimi lähelle ilmestyneet esineet. Nähdessään vastustajan pelihahmon tekoäly asetti pommeja pelikentälle tuhotakseen vastustajansa.

Suurimmat työssä esiintyneet ongelmat liittyivät tekoälyn toteuttamiseen. Tietyissä tilanteissa tekoäly ei osannut ennakoida maahan asetettujen pommien turvallisuutta. Tämä tapahtui tilanteissa, joissa tekoälyn ohjaama pelihahmo oli kerännyt huomattavan määrän pelihahmon ominaisuuksia parantavia esineitä, jolloin maahan asetetut pommit olivat tavallista paljon voimakkaampia. Kun pommeja asetettiin maahan useampi peräkkäin, syntyi ketjureaktio. Tällöin viimeisenä asetetut pommit räjähtivät aikaisemmin kuin tekoäly osasi arvioida. Toinen toteutetun tekoälyn käytöksessä esiintynyt ongelma on pelikentällä sijaitsevien esineiden tuhoutuminen vahingossa. Tekoälyn pelihahmon asettaessa useita pommeja peräkkäin se ei osannut arvioida ensimmäisen räjähdyksen jälkeisten räjähdysten aiheuttamia tuhoja. Tekoälyä ei kehitetty hyödyntämään pommien heitto- ja potkaisuominaisuutta, mikä asetti tekoälyn ohjaamat pelihahmot huonompaan asemaan verrattuna ihmispelaajien ohjaamiin pelihahmoihin.

Jotta tekoälystä saataisiin viihdyttävämpi ja tekoälyn ohjaamien pelihahmojen toiminnoista luonnollisempia, videopelin jatkokehityksessä on otettava huomioon yllä kuvattujen ongelmien ratkaiseminen. Videopelin tekoälyn persoonallisuuden kasvattamiseksi tekoälylle voitaisiin luoda erilaisia profiileja tai

käyttäytymismalleja. Esimerkkejä tällaisista profiileista ovat esimerkiksi *aggressiivinen pelaaja*, *varovainen pelaaja* ja *keräilijä*. Aggressiivinen pelaaja yrittää aina tilaisuuden tullen tuhota vastustajansa kaikin mahdollisin keinoin. Varovainen pelaaja suojautuu asettamiltaan ja vastustajien asettamilta pommeilta erityisen huolellisesti sekä pyrkii ennakoimaan vastustajiensa suunnitelmat asettaessaan pommeja pelikentälle. Keräilijä pyrkii keräämään mahdollisimman paljon ominaisuuksiaan parantavia esineitä pelikentältä, vältellen samalla mahdollista kontaktia vastustajien pelihahmojen kanssa. Näitä profiileja voitaisiin vaihtaa tekoälyn ohjaamalle pelihahmolle tilanteesta riippuen.

Lisäämällä videopeliin monitasoisen, pelaajan valittavissa olevan tekoälyn vaikeusasteen saataisiin aikaan usealle osaamistasoltaan erilaiselle pelaajalle sopiva peli. Tällainen ominaisuus on yleensä suuren työn takana ja vaatii huolellista suunnittelua onnistuakseen. Peliin voitaisiin sisällyttää esimerkiksi kohderyhmät *aloittelija*, *harrastelija* ja *kokenut*. Aloittelijalle tarkoitetut tekoälyn ohjaamat pelihahmot voisivat olla erehtyväisiä, hitaasti reagoivia vastustajia, kun taas kokeneelle pelaajalle tarkoitetut tekoälyn ohjaamat pelihahmot voisivat olla aggressiivisia, laskelmoivia vastustajia.

Tämän opinnäytetyön avulla olen oppinut paljon tekoälyn kehittymisen eri vaiheista, sen sovellusalueista, vahvuuksista ja puutteista sekä ihmisten odotuksista tekoälysovellusten suhteen. Opin erityisen paljon tekoälyn luomisen vaikeuksista luodessani näinkin yksinkertaista tekoälyä toteuttamaani videopeliin. Tämä työ lisäsi entisestään kiinnostustani videopeleihin ja tekoälyn tutkimiseen ja kehittämiseen.

Vaikka työssä keskitytään kaksiulotteisiin videopeleihin, siinä esitellyt algoritmit ja menetelmät sekä niiden vertailu ovat hyödyllisiä kenelle tahansa videopelien kehittämisestä kiinnostuneelle. Työssä toteutettua videopeliä on mahdollista kokeilla osoitteessa www.yoyogames.com/games/110832-demolisher.

LÄHTEET

Campbell, M. 2001. Deep Blue. Osoitteessa http://sjeng.org/ftp/deepblue.pdf. 1.8.2001.

Cormen, T. H. – Leiserson, C. E. – Rivest, R. L. – Stein, C. 2001. Introduction to Algorithms. Second Edition. Massachusetts: The MIT Press.

de Sousa, B. M. T. 2002. Game Programming All in One. Ohio: Premier Press, Inc.

Etteplan 2010. Asiantuntijajärjestelmä takaa masuunin tasaisen prosessin. Osoitteessa www.etteplan.fi/referenssit/case_raaheatj.php. 14.1.2010.

Haikala, I. – Märijärvi, J. 2006. Ohjelmistotuotanto. Helsinki: Talentum Media Oy.

Higgins, D. 2002. Generic A* Pathfinding. Teoksessa AI Game Programming Wisdom (toim. S. Rabin), 114–121. Massachusetts: Charles River Media, Inc.

Honkela, T. 2009. Tekoälyn menneisyydestä tulevaisuuteen: Faktoista viisauteen? Arpakannus 1/09, 12–15.

Jones, M. T. 2003. AI Application Programming. Massachusetts: Charles River Media, Inc.

Kokkarinen, I. 2003. Tekoäly, laskettavuus ja logiikka. Helsinki: Talentum Media Oy.

Koskimies, K. 2000. Oliokirja. Helsinki: Satku.

LaMothe, A. 2000. Inside Peliohjelmointi. Helsinki: IT Press.

Lester, P. 2005. A* Pathfinding for Beginners. Osoitteessa www.policyalmanac.org/games/aStarTutorial.htm. 18.6.2005.

Matthews, J. 2002. Basic A* Pathfinding Made Simple. Teoksessa AI Game Programming Wisdom (toim. S. Rabin), 105–113. Massachusetts: Charles River Media, Inc.

McGugan, W. 2007. Beginning Game Development with Python and Pygame. From Novice to Professional. Berkeley: Apress.

McMillan, M. 2005. Data Structures and Algorithms Using Visual Basic.NET. Cambridge: Cambridge University Press.

Patel A. 2009. Amit's A* Pages, Introduction. Osoitteessa http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. 31.12.2009.

Peck, A. 2008. Beginning GIMP. From Novice to Professional. Second Edition. Berkeley: Apress.

Penttonen, M. 1997. Johdatus algoritmien suunnitteluun ja analysointiin. Helsinki: Otatieto Oy.

Rouse III, R. 2001. Game Design Theory and Practice. Plano: Wordware Publishing, Inc.

Russell, S. – Norvig, P. 2003. Artificial Intelligence A Modern Approach. Second Edition. New Jersey: Pearson Education, Inc.

The Loebner Price 2007. Home Page of The Loebner Prize in Artificial Intelligence. Osoitteessa www.loebner.net/Prizef/loebner-prize.html. 26.10.2007.

Tozour, P. 2002. The Evolution of Game AI. Teoksessa AI Game Programming Wisdom (toim. S. Rabin), 3–15. Massachusetts: Charles River Media, Inc.

Varanese, A. 2003. Game Scripting Mastery. Ohio: Premier Press, Inc.