Atte Ahapainen

Tekoälyn vaikutus haastavuuteen hiiviskelypeleissä

Tietotekniikan Pro gradu -tutkielma

27. maaliskuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Atte Ahapainen

Yhteystiedot: atsaahap@student.jyu.fi

Ohjaaja: Vesa Lappalainen, Jukka Varsaluoma

Työn nimi: Tekoälyn vaikutus haastavuuteen hiiviskelypeleissä

Title in English: The effect of AI on difficulty in stealth games

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Pelit ja pelillisyys

Sivumäärä: 74+2

Tiivistelmä: Tutkielman tarkoituksena on selvittää, miten pelitekoäly vaikuttaa pelaajan kokemaan haastavuuteen hiiviskelypeleissä. Hiiviskelypeleissä pelaajan tavoitteena on selvitä erilaisista tehtävistä pysytellen samalla mahdollisimman huomaamattomana. Olen itse kiinnostunut pelitekoälystä ja siksi päädyin myös kyseiseen tutkielman aiheeseen.

Teoreettisessa viitekehyksessä käyn läpi pelitekoälyä eri näkökulmista. Esittelen pelitekoälyä yleisesti, ja kerron miten se eroaa yleisestä tekoälyn käsitteestä. Käyn myös läpi tekoälyn eri osa-alueiden toteutustapoja ja annan esimerkkejä erilaisista toteutuksista oikeissa peleissä. Lisäksi esittelen pelikokemuksen käsitettä ja sen tutkimiseen käytettäviä menetelmiä.

Käytin tutkimuksessani kvalitatiivista tutkimusmenetelmää, jonka lisäksi keräsin myös kvantitatiivista dataa. Haastattelin joukkoa pelaajia, joilla oli jo ennestään kokemusta genren peleistä. Haastateltavat pelasivat kehittämääni peliprototyyppiä ja keskustelimme heidän kokemuksistaan prototyyppiin liittyen. Peliprototyyppi koostuu kolmesta samanlaisesta pelikentästä, joissa kussakin on käytetty eri tavalla käyttäytyvää tekoälyvastustajaa. Käyn prototyypin koostumuksen myös tarkemmin läpi tutkielmassa.

Haastatteluiden perusteella voin todeta, että peliprototyypin tekoälyillä oli suuri vaikutus koettuun haastavuuteen. Tekoäly ei ainoastaan vaikuttanut haastavuuden tasoon, vaan myös siihen, millä tavalla tekoäly haastoi pelaajan. Staattisesti käyttäytynyt tekoäly tarjosi mekaanisen ja muistamista vaativan haasteen, kun taas satunnaisesti käyttäytyvä tekoäly pakotti

haastateltavan reagoimaan dynaamisesti muodostuviin tilanteisiin.

Avainsanat: Videopelit, tekoäly, pelikokemus, haastaavuus, peliprototyyppi

Abstract: The purpose of this thesis is to examine how game artificial intelligence (AI) effects the difficulty of the game perceived by the player. In stealth games players objective is to manage different kinds of assignments without being seen or heard by the AI opponent. I selected this subject for the thesis because I personally find game AI very interesting.

In the theory part of the thesis I will discuss game AI from different points of view. I will describe game AI in general and how it differs from generic term of AI. I will also go through different implementation methods of game AI components and give some real-life examples. In addition, Im going to explain the term game experience and what kind of research methods can be used to research it.

In my research I used a qualitative research method and I also collected some quantitative data. I interviewed a group of gamers who had prior experience of playing stealth games. The subjects played the game prototype that I developed for the research and we talked about their experience with the prototype. The game prototype consists of three identical levels with three differently behaving AI opponents. I will also discuss the composition of the game prototype in more detail.

According to the results the AI in game prototype had great impact on the perceived difficulty. Not only did it affect the level of difficulty, but it also affected the way the player was challenged. Statically behaving AI offered challenge that was mechanical and demanded remembering behavioral patterns of the AI whereas randomly behaving AI forced the player to react to dynamically emerging situations.

Keywords: Video games, artificial intelligence, gaming experience, difficulty, game prototype

Kuviot

Sisältö

1 Johdanto

Pelitekoäly on tärkeä osa pelaajan kokemaa kokonaisuutta. Se määrittelee esimerkiksi pelihahmojen tai vastustajan toimintatavat eri tilanteissa ja siten myös vaikuttaa suuresti pelaajan kokemukseen. Tekoäly tasapainoilee pelattavuuden ja uskottavuuden välillä tavoitteenaan luoda uskottavasti käyttäytyvä vastustaja, mutta kuitenkin samalla säilyttäen riittävän pelillisen haasteen ja viihdyttävyyden. Tässä eivät kuitenkaan kaikki pelit aina onnistu, joten mielestäni on kiinnostavaa tarkastella pelitekoälyä lähemmin.

Gradu-tutkielmani käsittelee hiiviskelypelien (stealth games) tekoälyä. Hiiviskely pelimekaniikkana esiintyy useissa eri genrejen peleissä, kuten toiminta- ja seikkailupeleissä. Pelien ideana on edetä mahdollisimman huomaamattomasti vastustajia vältellen ja samalla suoriutua erilaisista tehtävistä ja tilanteista. Usein pelaajalla on mahdollisuus valita, miten edetä eri tehtävien suhteen. Esimerkiksi vastustajan voi jättää täysin koskemattomaksi tai vastustajan voi mahdollisesti tainnuttaa ja niin edelleen. Valitsin kyseisen genren, koska siihen kuuluvien pelien tekoälyllä on mielenkiintoisia erikoispiirteitä, joita ei monessa muussa peligenressä välttämättä esiinny. Genre on myös yksi suosikeistani, joten se toimii omakohtaisena motivaationa tutkielman toteutuksessa.

Yleensä hiiviskelypeleissä tekoäly ohjaa pelihahmoja, jotka vartioivat tiettyä aluetta pelimaailmasta. Pelaajan tehtävänä on ohittaa hahmot kiinnittämättä huomiota itseensä (esimerkiksi Metal Gear pelisarja). (Alkaisy 2011) Pelaajan huomatessaan tekoäly osaa tehdä hälytyksen, jonka ansiosta muutkin peliympäristössä olevista tekoälyhahmoista tulevat tietoisiksi pelaajan sijainnista ja pyrkivät pysäyttämään pelaajan eri keinoin. Tekoäly osaa yleensä tarkkailla erilaisia merkkejä pelaajasta, kuten pelaajan tuottamia ääniä tai vaikkapa lumihankeen jääneitä jälkiä. Käsittelen hiiviskelypelien tekoälyn erityispiirteitä tarkemmin tutkielman teoriaosuudessa.

Tutkielman tavoitteena on selvittää, miten tekoälyn käyttäytyminen vaikuttavat koettuun haastavuuteen hiiviskelypeleissä. Tätä varten luon peliprototyypin teoriaosuudessa tekemieni löydösten pohjalta. Käyn prototyypin suunnittelu- ja toteutusprosessin läpi tutkielman toisessa osassa. Prototyyppiä testaa joukko halukkaita, joiden kokemukset pelistä kerätään

haastattelun avulla. Lopputuloksena on tieto siitä, miten eri tavalla käyttäytyvät tekoälyt vaikuttavat pelaajan kokemaan haastavuuteen. Tietoa voidaan käyttää apuna uusien genreen kuuluvien pelien suunnittelussa. Tutkimuksen tulokset ja pohdintaa niihin liittyen käyn läpi viidennessä luvussa.

2 Tekoäly

Tässä luvussa esittelen pelitekoälyn käsitettä. Kerron, mitkä ovat pelitekoälyn keskeiset erikoispiirteet ja miten pelitekoäly eroaa perinteisestä akateemisen tekoälyn käsitteestä. Esittelen myös pelitekoälyn toteutuksessa käytettäviä toteutustapoja.

Tarkastelen pelitekoälyn yleisen käsitteen lisäksi myös pelitekoälyä hiiviskelypeleissä. Esittelen tekoälylle ominaisia piirteitä ja niihin liittyviä vaatimuksia, jotka erityisesti liittyvät kyseiseen peligenreen. Tässä käytän apuna genren eri peleissä tehtyjä ratkaisuja tekoälyn toteutuksen suhteen.

2.1 Pelitekoäly

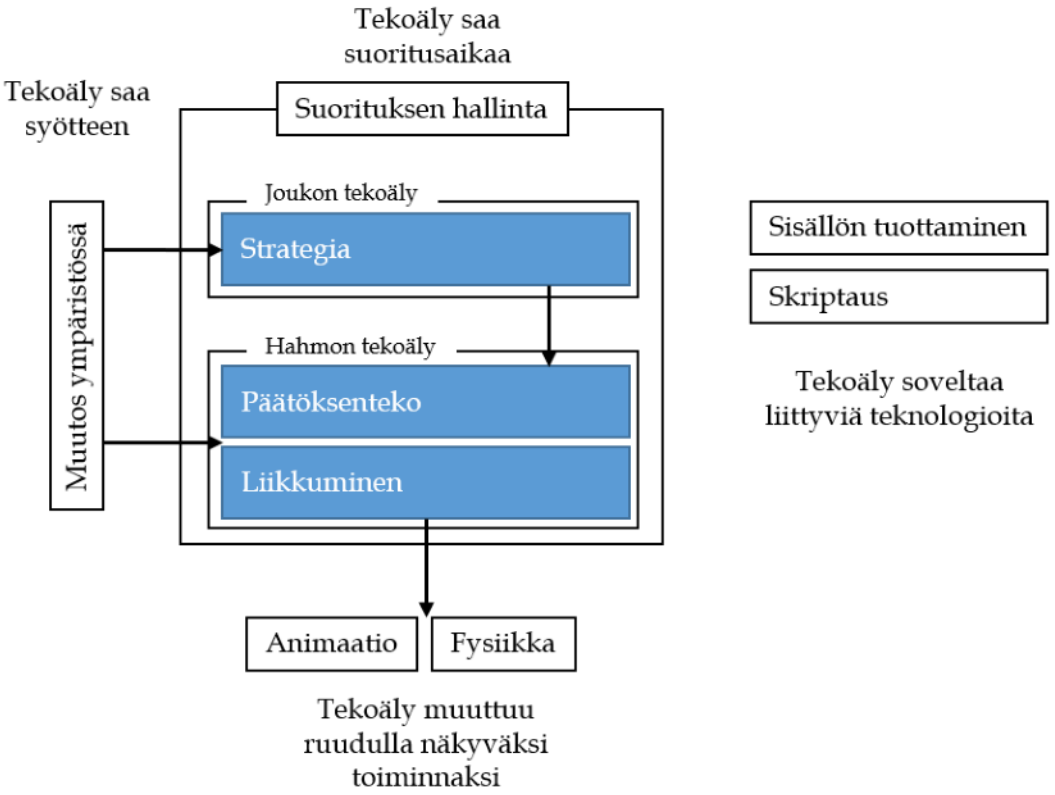
Neil Kirby (2010, 1) määrittelee pelitekoälyn siten, että tekoälyn tulee pystyä reagoimaan älykkäästi sen ympäristössä tapahtuvien muutosten pohjalta. Määritelmä siis selkeästi esittää, että tekoälyllä täytyy olla mahdollisuus toimia. Tekoäly on täysin hyödytön ilman minkäänlaista mahdollisuutta reagoida ympärillä tapahtuviin muutoksiin.

Tekoälyn täytyy kyetä myös tekemään älykkäitä päätöksiä eritoten silloin, kun pelaaja näkee tekoälyn toiminnassa. Se ei pysty antamaan kovinkaan älykästä vaikutelmaa, jos tekoäly toimii älykkäästi pelaajan näkökentän ulkopuolella, mutta töpeksii pelaajan nähdessä. Kirbyn (2010, 2) mukaan tekoälyn saaminen näyttämään fiksulta onkin helpompaa kuin sen estäminen toimimasta typerästi pelaajan edessä. Itse päätöksentekohetken ei tarvitse tapahtua pelaajan nähden, vaan päätöksenteosta johtuva toiminta on pelaajan kannalta tärkeämpi.

Kolmantena osana Kirbyn (2010, 2) määritelmää tekoälyn täytyy pystyä reagoimaan ympäristöönsä. Ympäristö toimii tekoälyn tapauksessa syötteenä, jota tekoäly seuraa ja tekee toimintoja saamansa syötteen mukaan. Erityisen huomionarvoista on, että pelaaja vaikuttaa tiettyjen rajojen puitteissa tähän syötteeseen. Siksi onkin tärkeää, että tekoäly pystyy havaitsemaan ympäristön muutokset ja reagoimaan pelaajan tekemisiin.

Myös Millington ja Funge (2009, 9) ovat selkeästi samoilla linjoilla Kirbyn kanssa esittelemänsä tekoälymallin (kuvio 1) perusteella. Kyseisessä mallissa tekoäly saa syötteen ympä-

ristössä tapahtuvasta muutoksesta, jonka jälkeen tekoäly tekee päätöksen toiminnasta. Lopputuloksena on esimerkiksi pelaajalle näkyvä animaatio ja liike.



Kuvio 1. Tekoälymalli (Millington ja Funge 2009, 9).

Millingtonin ja Fungen mallissa tekoälyn tehtävät on jaettu kolmeen osioon, jotka ovat liikkuminen, päätöksenteko ja strategia. Liikkuminen ja päätöksenteko sisältävät algoritmeja, jotka hallinnoivat yksittäisen hahmon toimintaa, kun taas strategiaosio hallitsee joukon yleistä toimintaa. Malli on erittäin yleinen, eikä kaikki tekoälyt käytä kaikkia esiteltyjä tasoja. Esimerkiksi useissa lautapeleissä tarvitaan vain strategiakomponenttia, kun taas joissain toimintapeleissä tekoäly on täysin reaktiivista, eivätkä siten tarvitse strategiakomponenttia lainkaan.

2.2 Pelitekoäly vs. yleinen tekoäly

Perinteisen, ”yleisen” tekoälyn määritelmiä on todella monta ja aiheesta on mahdotonta löytää yhtä ainoaa totuutta. Tekoäly voidaan määritellä olevan koneen osoittamaa älykkyyttä. Tekoälytutkimuksessa tutkitaan ”älykkäitä toimijoita”, jotka pystyvät havaitsemaan ympäristönsä ja suorittamaan toimintoja, joiden tarkoitus on maksimoida jonkin tavoitteen onnistumisen todennäköisyys. (Poole, Mackworth ja Goebel 1998, 1)

On tärkeää huomata eroavaisuus pelitekoälyn ja akateemikkojen tutkiman tekoälyn välillä. Akateeminen tutkimus jakautuu Bucklandin (2005, xix) mukaan kahteen leiriin: vahva tekoäly ja heikko tekoäly. Vahvan tekoälyn tutkimus keskittyy kehittämään järjestelmiä, jotka matkivat ihmisen ajatusprosesseja, ja heikon tekoälyn tutkimus keskittyy soveltamaan tekoälyteknologioita tosimaailman ongelmien ratkaisemiseen. On tärkeää huomata, että molemmat akateemiset lähestymistavat pyrkivät ratkaisemaan ongelman optimaalisesti, eivätkä aika- ja laitteistoresurssit ole kovinkaan rajoitettuja.

Pelitekoälyn ohjelmoijien täytyy taas työskennellä rajoitettujen resurssien puitteissa. Tekoälylle jää usein vain pieni osa käytettäväksi laitteistoresursseista, kun verrataan pelin muiden komponenttien käyttämään resurssien määrään (esim. grafiikka, fysiikanmallinnus jne.). Tämän takia tekoälyä kehitettäessä joudutaan tekemään kompromisseja hyväksyttävän suorituskyvyn säilyttämiseksi. Lisäksi tekoälyn tulee olla viihdyttävä, ja tämän saavuttamiseksi tekoäly toteutetaan ”epäoptimaalisesti”. (Buckland 2005, xix-xx) Yleinen tekoäly keskittyy siis selvästi enemmän prosesseihin, kun taas pelitekoälylle tärkeämpää on lopputulos.

2.3 Pelitekoälyn lyhyt historia

DaGraça (2017, 6) aloittaa pelitekoälyn historian shakkia pelaavista tietokoneista. Shakki oli täydellinen peli tekoälyn testaamista varten, sillä se vaati paljon ajattelua ja ennakointia. 50-luvulla tietokoneet eivät olleet vielä kykeneviä tällaiseen ihmismäiseen toimintaan pelatakseen ja voittaakseen shakkipelissä. Ensimmäinen askel olikin saada tietokone prosessoimaan pelin sääntöjä ja ”ajattelemaan” itsenäisesti, jotta kone pystyisi tekemään hyviä siirtoja ja sitä kautta myös voittamaan pelin. Ongelmia aiheuttivat shakin monet mahdollisuudet. Vaikka koneella olikin hyvä strategia pelin voittamiseen, strategiaa täytyi aina laskelmoida uudel-

leen ja mahdollisesti kehittää täysin uusi strategia, kun ensimmäinen meni pieleen.

Ihmiset pystyvät pelaamaan aina eri tavalla, minkä vuoksi ohjelmoijilla olisi kova työ syöttää kaikki mahdollinen data, jotta kone pystyi voittamaan shakissa. Siksi kaikkien mahdollisuuksien kirjoittaminen ei ollut kovin fiksu ratkaisu, ja tästä syystä ohjelmoijien oli ajateltava ongelmaa uudestaan. Ratkaisuna ongelmaan kone laitettiin päättämään itsenäisesti siirrosta jokaisella vuorolla, mikä mahdollisti koneen sopeutumisen jokaiseen pelitilanteeseen. Tämä aiheutti kuitenkin uuden ongelman: kone ei suunnitellut pitkän aikajänteen strategiaa, joten sitä vastaan oli melko helppoa pelata. Vuosikymmeniä myöhemmin ongelma ratkaistiin ja tekoälyn käsite syntyi. Amerikkalainen Arthur Samuel oli 50-luvulla keskeinen tekijä tässä prosessissa. Hän kehitti koneen, joka pystyi oppimaan itse ja tällä tavalla muistamaan kaikki shakin mahdolliset siirtokombinaatiot. Koneoppimisen ansiosta ihmisen avustukselle ei ollut enää tarvetta ja kone pystyi itsenäiseen ajatteluun. (DaGraça 2017, 6-7)

Ensimmäiset videopelit, joissa on tekoälyvastustajia, alkoivat ilmestyä 1970-luvulla. Pian myös alkoi ilmestyä pelejä, jotka nostivat tekoälyn laatua ja sitä myötä myös odotuksia tulevia pelejä kohtaan. Tällaisia pelejä olivat mm. arcade-laitteilla pelattavat Speed Racer (Taito, 1974), Qwak (Atari, 1974) ja Pursuit (Atari, 1975). Muita mainitsemisen arvoisia pelejä ovat tekstiseikkailupelit, jotka julkaistiin suurtietokoneille ja myöhemmin ensimmäisille PC-laitteille (esimerkiksi Hunt the Wumpus (1975) ja Star Trek (1971)). Tekoälyn vastukset omasivat perinteisten käyttäytymismallien lisäksi satunnaisuutta, minkä vuoksi tekoäly vastus oli ennalta-arvaamaton ja siten jokainen pelikerta oli erilainen pelikokemus. Tällaiset uudet pelikokemukset mahdollistuivat mikroprosessoreiden tulon myötä. Space Invaders (Taito, 1978) toi liikkumismallit, ja Galaxian (Namco, 1979) paransi näitä malleja lisäämällä vaihtelua, ja siten myös tekoälyn monimutkaisuus lisääntyi. Myöhemmin Pac-Man (Namco, 1980) toi liikkumismallit myös sokkelogenreen. (DaGraça 2017, 7)

Pac-Manin vaikutus tekoälyn suunnitteluun ja koko pelialaan on valtavan suuri. Myös hiiviskelypelien genre on saanut paljon vaikutteita Pac-Manista. Peli saa pelaajan uskomaan aaveiden jahtaavan häntä, muttei kuitenkaan siten, että aaveet pyrkisivät mahdollisimman suoraviivaisesti ja yksinkertaisesti saamaan pelaajan kiinni. Aaveet jahtaavat tai välttelevät pelaajaa eri tavoilla aivan kuin jokaisella aaveella olisi oma persoonallisuus. Tämän ansiosta syntyy illuusio siitä, että pelaaja pelaa yksilöllisiä aaveita vastaan sen sijaan, että kaikki

aaveet olisivat toistensa kopioita. Pac-Manin jälkeen Karate Champ (Data East, 1984) loi ensimmäisen tekoälyvastuksen taistelupeliin ja Dragon Quest (Enix, 1986) toi taktisen järjestelmän roolipeliin. (DaGraça 2017, 8)

Suurin osa yllä mainituista peleistä edustavaa eri genrejä ja on yksilöllisiä genrensä edustajia. Kaikki nämä pelit käyttivät kuitenkin samaa metodia tekoälyn luomiseksi, mikä tunnetaan äärellisenä tilakoneena (finite-state machine eli FSM). Tilakone luodaan antamalla koneelle kaikki mahdolliset käyttäytymismallit, mitä se tarvitsee haastaakseen pelaajan. Aivan kuten ensimmäiselle koneelle, joka pelasi shakkia. Tekoälyn ohjelmoija määrittää tarkalleen, miten koneen tulisi käyttäytyä eri tilanteissa liikkuakseen, vältelläkseen, hyökätäkseen tai tehdäkseen mitä tahansa muuta haastaakseen pelaajan, ja tätä tapaa luoda tekoäly peliin käytetään vielä tänä päivänäkin lähes kaikissa peleissä. (DaGraça 2017, 8)

2.4 Pelitekoälyn erikoispiirteitä ja toteutustapoja

Kuten aiemmin mainitsin, pelitekoälyn päävastuualueet voidaan jakaa karkeasti kolmeen komponenttiin: liikkuminen, strategia ja päätöksenteko (kuvio 1). Kaikki komponentit eivät kuitenkaan ole aina pakollisia pelissä. Seuraavaksi esittelen muutamia yleisimpiä toteutustapoja ja tekniikoita, joita käytetään jokaisen komponentin toteutuksessa.

2.4.1 Liikkuminen ja reitinhaku

Päästäkseen paikasta toiseen ja suorittaakseen eri tehtäviä pelihahmo tarvitsee tavan liikkua ja ymmärtää, minne on mahdollista siirtyä ja minne ei. Lisäksi hahmon on pystyttävä välttelemään esteitä matkan varrella. Tätä varten peleissä käytetään usein erilaisia algoritmeja liikkumisen mahdollistamiseksi.

Kaikki liikkumiseen luodut algoritmit käyttäytyvät samalla periaatteella: ne ottavat syötteenä geometristä dataa omasta tilastaan ja ympäristöstä. Tuloksena tästä syntyy geometristä dataa, joka esittää haluttua liikettä. Jotkin algoritmit tarvitsevat hyvin vähän syötetietoa, kuten esimerkiksi hahmon sijainnin ja jahdattavan vastustajan sijainnin. Monimutkaisemmat algoritmit taas vaativat enemmän tietoa pelin tilasta ja ympäristön geometriasta. Myös syötteestä seuraava tulos vaihtelee algoritmin mukaan. (Millington ja Funge 2009, 40)

Liike voidaan jakaa kahteen luokkaan: kineettinen ja dynaaminen liike. Vanhemmissa peleissä oli hyvin yleistä, että hahmoilla oli vain kaksi tai kolme eri liikkumisnopeutta (esimerkiksi paikallaan pysyminen, kävely ja juoksu). Dynaaminen liike ottaa taas huomioon hahmon hetkittäisen nopeuden ja sijainnin ja tällä tavoin joko hidastaa tai nopeuttaa liikettä tarpeen mukaan. Liikettä ei siis ole lukittu muutamaan eri vakioon. Kineettisessä liikkeessä algoritmi antaa tuloksena halutun nopeuden, kun taas dynaamisessa liikkeessä algoritmi palauttaa kiihtyvyyden, jolla hahmon nopeutta tulee muuttaa. (Millington ja Funge 2009, 40-41)

Hahmojen täytyy myös pystyä liikkumaan pelialueella. Joskus liikkuminen pelialueella on suunniteltu etukäteen, kuten vaikkapa hiiviskelypelien vartioiden partiointireitit. Toisaalta taas hahmo voi harhailla satunnaisesti ympäriinsä. Näissä liikkumisen määrittämisissä on selkeitä heikkouksia. Staattisen reitin voi helposti katkaista siirtämällä jokin objekti reitille. Satunnaisen harhailun seurauksena hahmo vaikuttaa tyhmältä ja saattaa jäädä jumiin. Hahmon täytyy myös pystyä suunnittelemaan etukäteen, minne liikkua. Hiiviskelypelin vartijan on vaikkapa liikuttava lähimmällä hälytyspisteelle kutsuakseen apua ja niin edelleen. Tätä varten tekoälyn täytyy kyetä laskemaan sopiva reitti senhetkisestä sijainnista tavoitteeseen. Reitin tulisi olla myös mahdollisimman järkevä ja nopea, jotta hahmo vaikuttaisi fiksulta. (Millington ja Funge 2009, 197)

Reitin löytämistä varten on olemassa erilaisia reitin etsimiseen tarkoitettuja algoritmeja, joista käytetyimmät ovat Dijkstra ja A* algoritmit. Reitinhaku sijoittuu tekoälymallissa jonnekin liikkumisen ja päätöksenteon välimaastoon. Määränpään yleensä määrittää jokin päätöksentekokomponentti. Reitinhaku määrittää määränpään saavuttamiseksi reitin, ja liikekomponentti toteuttaa liikkumisen reittiä pitkin. Reitinhaku voi myös toimia päätöksentekokomponenttina, jolloin se päättää halutun määränpään ja sinne vievän reitin. (Millington ja Funge 2009, 197-198)

Xiao Cui ja Hao Shi (2011) esittävät A*-algoritmin seuraavanlaisena pseudokoodina:

```
1. Add the starting node to the open list.
2. Repeat the following steps:
    a. Look for the node which has the lowest
       f on the open list. Refer to this node
       as the current node.
    b. Switch it to the closed list.
    c. For each reachable node from the current
       node
        i. If it is on the closed list, ignore
           it.
       ii. If it isnt on the open list, add it
           to the open list. Make the current
           node the parent of this node. Record
           the f, g, and h value of this node.
      iii. If it is on the open list already,
           check to see if this is a better
           path. If so, change its parent to the
           current node, and recalculate the f
           and g value.
    d. Stop when
        i. Add the target node to the closed
           list.
       ii. Fail to find the target node, and the
           open list is empty.
3. Tracing backwards from the target node to the
   starting node. That is your path.
```

Pseudokoodissa $g(n)$ edustaa tarkkaa hintaa aloituspisteestä mihin tahansa pisteeseen n, $h(n)$ taas edustaa arvioitua hintaa pisteestä n määränpäähän ja $f(n) = g(n) + h(n)$. (Cui ja Shi 2011)

Suurin osa peleistä käyttää A* algoritmia. Vaikka A* onkin tehokas ja helppo toteuttaa, se ei pysty toimimaan suoraan pelin ympäristödatan perusteella. A* vaatii ympäristödatan esittämistä erityisessä datarakenteessa, jota kutsutaan suunnatuksi epänegatiiviseksi painotetuksi verkoksi (directed non-negative weighted graph). Dijkstra puolestaan on yksinkertaisempi

algoritmi A*:en verrattuna, ja sitä käytetäänkin useammin taktisessa päätöksenteossa kuin reitinhaussa. (Millington ja Funge 2009, 197-198)

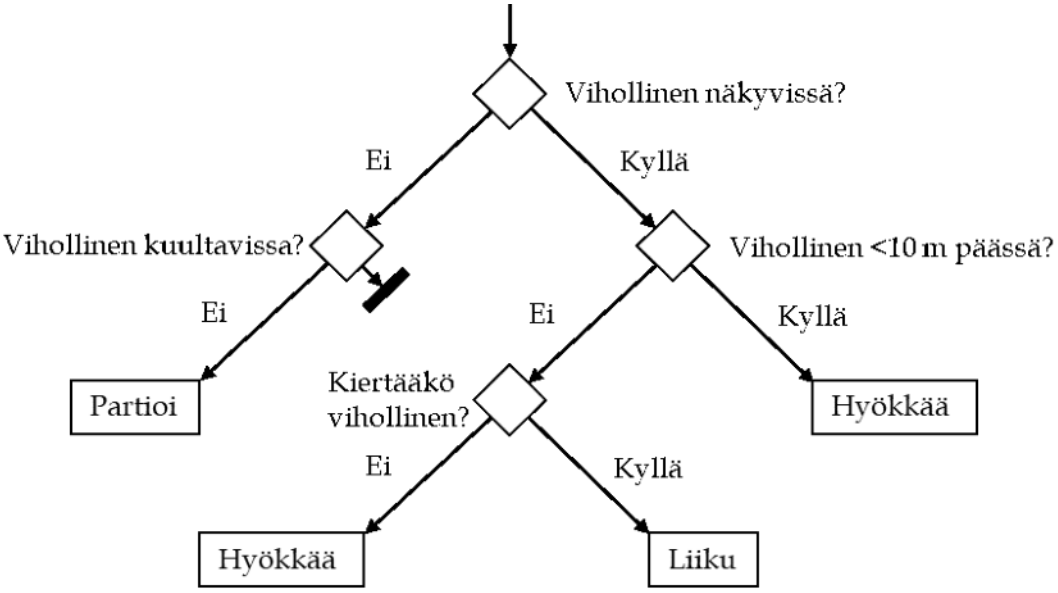
2.4.2 Päätöksenteko

Päätöksentekokomponentti mahdollistaa hyvin pitkälle sen, että tekoäly näyttää tekevän fiksuja asioita. Päätöksenteon seurauksena tekoäly alkaa suorittaa toimenpiteitä (esimerkiksi liikkuminen tai animaatio), joilla se pystyy saavuttamaan halutun tavoitteen. Komponentti on suhteellisen pieni osa siitä, mitä vaaditaan uskottavan tekoälyn luomiseksi, ja se on myös useimmiten toteutettu yksinkertaisella tilakoneella tai päätöspuulla. Myös sääntöpohjaisia järjestelmiä on olemassa, vaikkakin hieman vähemmän. Monimutkaisempia päätöksentekoon käytettäviä järjestelmiä (esimerkiksi sumea logiikka tai neuroverkot) on käytettävissä, mutta niiden toteuttaminen on haastavaa, kuten myös niiden saaminen käyttäytymään halutulla tavalla. (Millington ja Funge 2009, 293)

Vaikka toteutustekniikoita onkin monia, voidaan niiden havaita käyttäytyvän samalla tavalla: tekoäly prosessoi saamansa syötteen, jonka perusteella se tuottaa tavoitteita, jotka se pyrkii täyttämään. Päätöksenteon syötteenä toimii tekoälyn senhetkinen tietämys ja tulosteena toimintapyyntö. Tietämys voidaan jakaa ulkoiseen ja sisäiseen tietämykseen: ulkoinen tietämys tarkoittaa tekoälyn tietämystä sen ympäristöstä, kuten hahmojen sijainnit, ympäristön ominaispiirteet, onko esimerkiksi jokin vipu kytketty, äänen suunta ja niin edelleen. Sisäinen tietämys on taas tekoälyn ymmärrys sen omasta tilasta, kuten terveyspisteet, päätavoitteet ja mitä tekoäly teki muutama hetki sitten. Vastaavasti toiminnot voidaan myös jakaa ulkoisiin ja sisäisiin toimintoihin: ulkoiset toiminnot ovat pelaajan helposti nähtävissä (liike, vivun kytkeminen, ampuminen jne.), kun taas sisäiset toiminnot ovat paljon hienovaraisempia, mutta ovat silti erittäin tärkeitä joissain päätöksentekoalgoritmeissa. Sisäisellä toiminnolla voidaan muuttaa vaikkapa tekoälyn suhtautumista pelaajaan, tekoälyn tunnetilaa tai päätavoitetta. (Millington ja Funge 2009, 293-295)

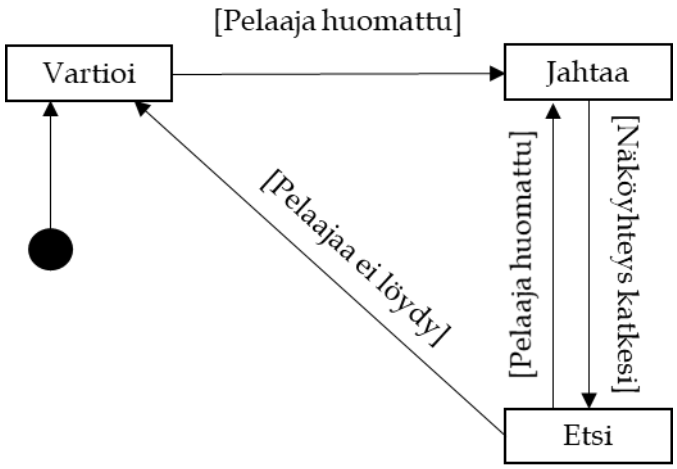
Päätöspuut (kuvio 2) ovat nopeita, helposti toteutettavia ja yksinkertaisia ymmärtää. Niihin on kuitenkin helppo lisätä monimutkaisuuta puun modulaarisen luonteen takia. Päätöspuu sisältää yhdistettyjä päätöspisteitä ja aloittava päätös sijaitsee päätöspuun juuressa. Jokais-

ta päätöstä kohti juuresta alkaen valitaan tilannetta kuvaava vaihtoehto. Jokainen valinta tehdään tekoälyn tietämyksen perusteella. Algoritmi jatkaa päätöspisteitä pitkin, kunnes päätösprosessissa ei ole enää valintoja jäljellä. Puun lehdet edustavat toteutettavaa toimintaa ja kun päätöspuu saapuu lehteen, toteutetaan lehden edustama toiminta välittömästi. (Millington ja Funge 2009, 295-296)



Kuvio 2. Päätöspuu (Millington ja Funge 2009, 296)

Pelihahmoilla on usein rajattu joukko käyttäytymistapoja. Esimerkiksi monissa hiiviskelypelissä vartija partioi tiettyä aluetta kunnes näkee pelaajan ja lähtee jahtaamaan tätä. Jahtaaminen loppuu vartijan menettäessä näköyhteyden pelaajaan. Tällaisen käyttäytymisen pystyy toteuttamaan päätöspuulla, mutta usein on helpompaa käyttää juuri tätä tarkoitusta varten suunniteltua tekniikka, eli tilakonetta. Tilakoneet ovatkin todella laajasti käytettyjä peleissä ja usein skriptauksen tukemana käsittää suurimman osan päätöksentekojärjestelmästä. Tilakoneet (kuvio 3) pystyvät ottamaan huomioon sekä ympäristönsä, että hahmon sisäisen tilan. (Millington ja Funge 2009, 309)

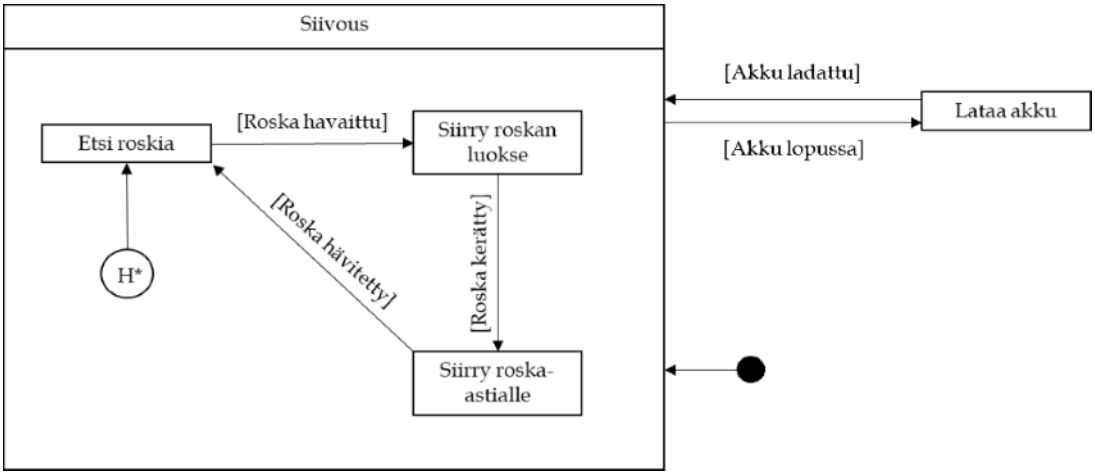


Kuvio 3. Tilakone

Tilakoneessa jokainen pelihahmo omaa yhden tilan. Normaalisti toiminnot ja käyttäytymiset liittyvät jokaiseen tilaan. Hahmo jatkaa toiminnon suorittamista niin kauan, kun tila on voimassa. Tilat ovat yhdistetty toisiinsa siirtymillä. Jokainen siirtymä vie yhdestä tilasta toiseen kohdetilaan, ja jokaisella siirtymällä on jokin ehto siirtymän toteutumiseksi. Jos huomataan, että siirtymän ehdot täyttyvät, muuttaa hahmo tilaansa siirtymän kohdetilaan. Päätöspuussa jokainen toiminto on aina saavutettavissa samojen päätöksien kautta, kun taas tilakoneessa vain sen hetkisen tilan siirtymät ovat mahdollisia, joten kaikki toiminnot eivät ole aina saatavilla. (Millington ja Funge 2009, 309-310)

Perinteisillä tilakoneilla voi kuitenkin olla haastavaa toteuttaa joitain käyttäytymisiä. Tästä hyvä esimerkki ovat ns. ”hälytyskäyttäytymiset”. Esimerkiksi siivousrobotin (joka toimii tilakoneen avulla) täytyy jossain vaiheessa lähteä lataamaan akkua latauspisteelle. Lataustoiminnon täytyy pystyä keskeyttämään sen hetkinen toiminto, oli se mikä tahansa. Latauksen päätyttyä täytyy robotin jatkaa siitä, mihin se jäi ennen latausta. Kyseessä on siis hälytysmekanismi. Jokin keskeyttää normaalin käytöksen suorittaakseen jotakin tärkeämpää. Perinteisessä tilakoneessa tällaisen käytöksen toteuttaminen tuplaisi tilojen määrän. Yhden tason hälytyksessä tämä ei ole ongelma, mutta entä jos hälytystiloja onkin useampia? Sen sijaan, että kaikki logiikka pyritään toteuttamaan yhdessä tilakoneessa, voidaan tilakone jakaa useampaan tilakoneeseen, jotka järjestetään hierarkkisesti. Tällä tavoin hierarkiassa alin tilakone otetaan käyttöön vain, jos ylemmän tason hälytykset eivät ole aktiivisia. Tällaisia tilakoneita

kutsutaan hierarkkisiksi tilakoneiksi (kuvio 4). (Millington ja Funge 2009, 318)



Kuvio 4. Hierarkkinen tilakone (Millington ja Funge 2009, 320)

Tilakoneen toteuttaminen vaati kehittäjältä jokaisen tilan ja siirtymän ennakointia. Lisäksi tilakoneen tiloja ja siirtymiä on erittäin vaikeaa käyttää, sillä niillä on usein hyvin täsmällinen konteksti ja käyttötarkoitus. Jotta tilakoneen toteutus olisi joustavampi, voidaan käyttää niin sanottua komponenttipohjaista hierarkkista tilakonetta (Component-based Hierarchical State Machine). Komponenttisoitujen tilojen ja siirtymien ansiosta tilakonetta voidaan joustavasti muuttaa ohjelman kääntämisen, pelin suunnittelun tai sen ajon aikana.(Hu, Zhang ja Mao 2011)

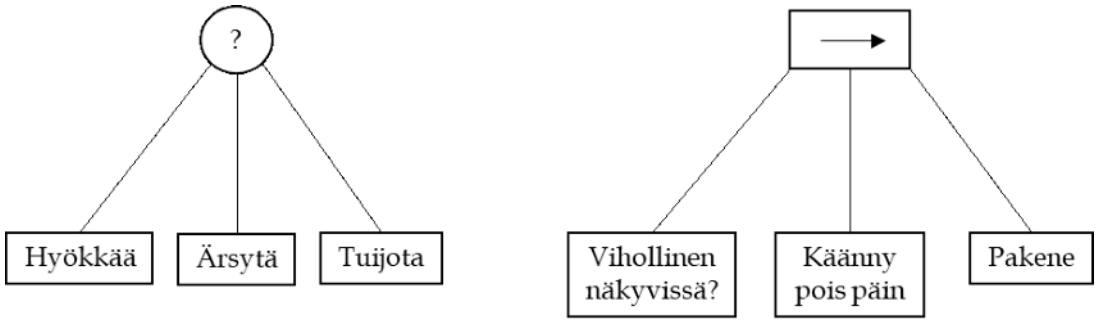
Käytöspuut ovat useiden eri tekoälytekniikoiden (hierarkkiset tilakoneet, aikataulutus, suunnittelu ja toiminnan suoritus) yhdistelmä. Niiden vahvuutena on kyky lomittaa eri tekniikoiden vastuut siten, että niitä on helppo ymmärtää ja luoda. Vaikka käytöspuut ovat yhä enemmän käytössä, eivät ne aina sovellu hyvin päätöksentekoon. Käytöspuilla on paljon yhteistä hierarkkisten tilakoneiden kanssa, mutta tilan sijaan niiden tärkein elementti on tehtävä (a task). Tehtävä voi olla esimerkiksi pelitilan muuttujan arvon tarkastaminen tai animaation suorittaminen. Tehtävät koostetaan alipuiksi, jotka edustavat monimutkaisempia toimintoja. Näistä toiminnoista voidaan taas koostaa korkeamman tason toimintoja. Juurikin koostettavuutensa ansiosta käyttäytymispuut ovat tehokkaita. Koska kaikilla tehtävillä on yhteinen rajapinta ja ne ovat suurimmilta osin itsenäisiä, voidaan ne järjestää hierarkkisesti ilman huolta siitä, miten yksityiskohdat eri alitehtävissä on toteutettu. (Millington ja Funge 2009, 334)

Kaikilla käytöspuun tehtävillä on sama perusrakenne. Niille annetaan suoritusaikaa oman tehtävänsä toteuttamiseksi. Kun ne ovat valmiita, palauttavat ne tilakoodin, joka indikoi joko onnistumista tai epäonnistumista. Jotkut saattavat käyttää useampaa palautusarvoa, kuten virheen tila silloin, kun jokin epäonnistuu tai ”tarvitaan lisää aikaa” –viesti aikatauluttamista varten. Tehtävät mahdollistavat suuren joustavuuden, jos jokainen niistä on rikottavissa pienempiin käyttökelpoisiin komponentteihin tehtävän sisältämän koodin monimutkaisuudesta riippumatta. Tämän ansiosta käytöspuut ovat erittäin tehokas työkalu, kun siihen liitetään myös muokkaus mahdollisuus graafista käyttöliittymää käyttäen. Näin myös muut käytöspuista vähemmän tietävät pystyvät suunnittelemaan suhteellisen monimutkaisia tekoälyn käytöksiä. (Millington ja Funge 2009, 334)

Millington & Funge (2009, 335) antavat esimerkin yksinkertaisesta käytöspuusta, joka koostuu kolmesta eri tehtävästä: ehdosta (condition), toiminnosta (action) ja yhdistelmästä (composite). Ehto testaa jotain pelin ominaisuutta (esimerkiksi kohteen etäisyyttä, näköyhteyttä, hahmon tilaa jne.). Jokainen testi täytyy toteuttaa erillisenä tehtävänä, johon usein liitetään parametrejä tehtävän uudelleenkäyttämisen mahdollistamiseksi. Jokainen ehto palauttaa joko onnistumista tai epäonnistumista indikoivan tilakoodin. Toiminnot puolestaan muuttavat pelin tilaa. Toimintoja voi olla esimerkiksi animaatioille, hahmon liikkumiselle, hahmon sisäisen tilan muutoksille, äänen toistamiselle, dialogin suorittamiselle tai vaikkapa erityisen tekoälykoodin suorittamiselle (kuten reitinhaku). Aivan kuten ehdot, myös jokainen toiminto tarvitsee oman toteutuksen, ja toimintoja on todennäköisesti useita. Useimmiten toiminnot onnistuvat. Jos epäonnistumiselle on mahdollisuus, on hyvä käyttää ehtoa tarkistamaan tämä mahdollisuus ennen toiminnon suorittamista. Epäonnistuvia toimintoja on kuitenkin mahdollista luoda. Ehdot ja toiminnot sijaitsevat käytöspuun lehdissä. Suurin osa puun haaroista on toteutettu yhdistelmillä. Nimensä mukaisesti yhdistelmät pitävät kirjaa lapsitehtävistään (ehdoista, toiminnoista ja muista yhdistelmistä) ja yhdistelmän käytös perustuu sen lapsiin. Toisin kuin ehtoja ja toimintoja, käytöspuussa on vain muutamia yhdistelmiä. Tämä johtuu siitä, että jo kourallisella erilaisia käytösryhmiä voidaan luoda hyvinkin monimutkaisia käytöksiä.

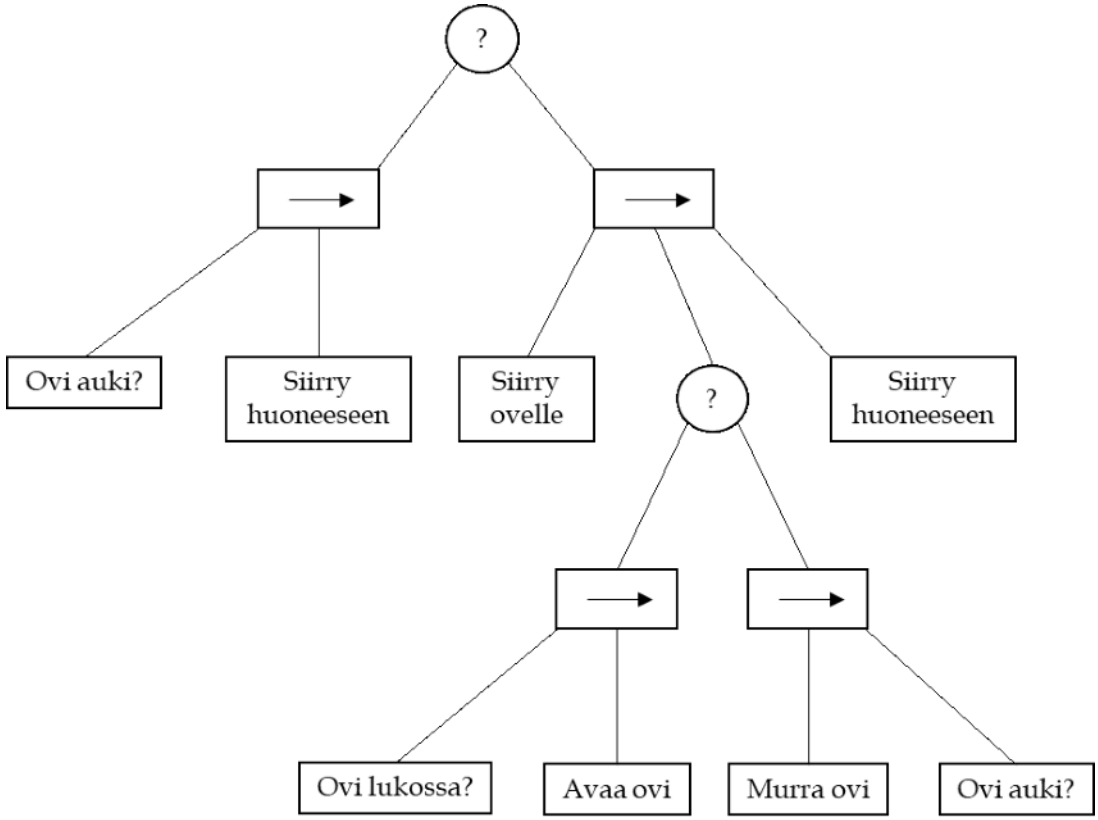
Esimerkissä käytetään kahta eri yhdistelmätyyppiä (kuvio 5): valitsin (selector) ja jono (sequence). Molemmat suorittavat omat lapsikäytöksensä vuoron perään. Kun lapsikäytös on suoritettu,

yhdistelmä päättää, jatketaanko lapsien läpikäymistä vai lopetetaanko käytös. Valitsin lopettaa suorituksen välittömästi, kun yksi sen lapsista suorittaa käytöksensä onnistuneesti. Jos mikään lapsista ei onnistu käytöksen suorittamisessa, valitsin palauttaa epäonnistumista indikoivan koodin. Jono palauttaa epäonnistumista indikoivan tilakoodin heti, jos yksikin lapsista epäonnistuu. Jos kaikki lapset onnistuvat käytöksensä suorittamisessa, se palauttaa onnistumista indikoivan tilakoodin. (Millington ja Funge 2009, 335)



Kuvio 5. Esimerkki valitsin (vasen) ja jono (oikea) tehtävistä (Millington ja Funge 2009, 336)

Valitsin ja jono mallitehtävistä (kuvio 5) voidaankin huomata, että tällaisia kokonaisuuksia (ts. käytöksiä) yhdistelemällä voidaan saada aikaa hyvinkin monimutkaisia käytöspuita. Esimerkki käytöspuusta (kuvio 6) antaa jo jonkinlaisen viiteen tehtävien yhdistelemisen mahdollisuuksista.



Kuvio 6. Esimerkki käytöspuusta (Millington ja Funge 2009, 339)

Aiemmat päätöksentekoon käytetyt tekniikat ovat olleet hyvin selkeitä siinä mielessä, että päätöksen tai ehdon vaihtoehdot ovat erittäin yksiselitteisiä (esim. kyllä tai ei vaihtoehdot). Vaihtoehtojen välillä ei ole ”harmaata aluetta”. Sumea logiikka (fuzzy logic) on joukko matemaattisia tekniikoita, joita käytetään ”harmaita alueita” käsitellessä. Sumean logiikan avulla voidaan esimerkiksi ”sumentaa” rajaa kahden eri tilakoneen tilan välillä. Sen sijaan, että tilasta hypättäisiin suoraan toiseen, voidaan tilaa muuttaa asteittain, jolloin mahdollisesti saadaan aikaan realistisempi käytöksenmuutos. Vaikka sumeaa logiikkaa käytetäänkin peleissä paljon, on hyvä huomata, että suomea logiikka on suurilta osin osoitettu epäuskottavaksi valtavirran akateemisessa tekoäly-yhteisössä. (Millington ja Funge 2009, 371)

Perinteisessä logiikassa käytetään predikaatin käsitettä. Predikaatti on jonkin ominaisuus tai kuvaus. Hahmo voi olla vahingoittunut. Tässä tapauksessa predikaatti on ”vahingoittunut” ja jokainen hahmo on joko vahingoittunut tai ei. Vahingoittumisen asteesta ei ole tietoa. Nämä predikaatit voidaan käsittää joukkoina. Kaikki, joihin predikaatti vaikuttaa, kuuluvat jouk-

koon, ja kaikki muut ovat joukon ulkopuolella (klassiset joukot). Sumea logiikka laajentaa predikaatin käsitettä antamalla sille numeerisen arvon. Hahmo voi olla vahingoittunut arvolla 0,5. Tällainen hahmo on enemmän vahingoittunut kuin hahmo, joka on vahingoittunut arvolla 0,3. Sen sijaan, että asiat kuuluvat tai eivät kuulu joukkoon, kaikki asiat voivat kuulua osittain joukkoon. Jotkin enemmän kuin toiset. Tällaisia joukkoja kutsutaan sumeiksi joukoiksi ja numeerista arvoa kutsutaan jäsenyydenasteeksi. Jos jokin kuuluu täysin joukkoon, sen jäsenyydenaste on 1 ja vastaa silloin joukkoon kuulumista perinteisen logiikan säännöillä. Jäsenyydenasteen ollessa 0 asia ei kuulu lainkaan joukkoon. (Millington ja Funge 2009, 372)

Sumeissa ja klassisissa joukoissa mikä tahansa voi olla osa useaa joukkoa samanaikaisesti. Hahmo voi olla esimerkiksi sekä nälkäinen että haavoittunut. Perinteisessä logiikassa on usein joukkoja, jotka ovat toisensa poissulkevia. Hahmo ei voi olla samanaikaisesti terve ja haavoittunut. Sumeassa logiikassa näin ei enää kuitenkaan ole. Hahmo voi olla terve ja haavoittunut tai pitkä ja lyhyt. Hahmolla on vain eri jäsenyydenaste jokaiseen joukkoon. Hahmo voi olla vahingoittunut arvolla 0,5 ja terve arvolla 0,5. Sumean logiikan vastine toistensa poissulkeville joukoille on vaatia, että jäsenyydenasteiden yhteissumma on 1. Tällöin hahmo ei voi olla terve arvolla 0,4 ja haavoittunut arvolla 0,8. On kuitenkin harvinaista, että sumean logiikan toteuttamisessa päätöksenteossa käytetään toisensa poissulkevia joukkoja. Useimmat toteutukset sallivat minkä tahansa jäsenyysarvon käyttämällä sumentamismetodia. Tällä tavoin arvojen summa on noin 1 ja käytännössä arvojen pienellä heittelyllä ei ole juurikaan vaikutusta. (Millington ja Funge 2009, 372-373)

Koneoppiminen on kuuma aihe myös peleissä. Periaatteessa oppivalla tekoälyllä on potentiaalia sopeutua jokaiseen pelaajaan oppimalla tämän temput ja tekniikat ja tätä kautta antamaan yhtenäisen haasteen. Potentiaalia on myös uskottavampien hahmojen luomiseksi. Hahmot voivat oppia ympäristönsä ja käyttää sitä mahdollisimman tehokkaasti hyödykseen. Koneoppimisella voidaan myös vähentää tarvittua vaivaa luoda pelikohtainen tekoäly. Hahmon pitäisi kyetä oppimaan kaiken ympäristöstään ja sen tuomista taktisista vaihtoehdoista. Käytännössä koneoppiminen ei ole kuitenkaan vielä kantanut hedelmää peleissä. Sen käyttäminen vaatii tarkkaa suunnittelua ja ymmärrystä kaikista mahdollisista sudenkuopista. ”Hype” on usein paljon todellisuutta parempaa, mutta jos ymmärtää jokaisen tekniikan oikut ja

on realistinen toteutuksessa, ei ole syytä, miksei koneoppimista voisi onnistuneesti käyttää peleissä. Oppimistekniikoita on useita aina yksinkertaisesta numeroiden pikku säädöistä monimutkaisiin neuroverkkoihin asti. Jokaisella tekniikalla on ominaispiirteensä, jotka täytyy ymmärtää ennen niiden käyttämistä peleissä. (Millington ja Funge 2009, 579)

Hyvänä esimerkkinä koneoppimisen käyttämisestä videopelien tekoälynä on MazeBase tutkimus. MazeBase on yksinkertainen kaksiulotteinen hiekkalaatikko, jonka avulla voidaan testata erilaisia koneoppimismalleja yksinkertaisissa järkeilyä ja suunnittelua vaativissa pulmissa. On kuitenkin selvää, että kaikki mallit eivät pärjänneet kaikissa pulmissa kovinkaan hyvin. MazeBasella koulutetut mallit pystyivät kuitenkin päihittämään Star Craft (Blizzard Entertainment, 1998) pelin tekoälyn kohtuullisen toistuvasti. (Sukhbaatar ym. 2015)

2.4.3 Strategia

Päätöksentekoon käytetyillä tekniikoilla on kaksi tärkeää rajoitetta: ne on tarkoitettu vain yhden hahmon käytettäväksi, ja ne eivät yritä päätellä omaamistaan tiedoista pelitilanteen kokonaiskuvan ennustetta. Molemmat näistä rajoitteista kuuluvat suurimmalta osin taktisiin ja strategisiin tekoälykomponentteihin. Tässä luvussa käsittelen muutamia tekniikoita, joiden avulla voidaan luoda hahmoille kehys taktista ja strategista päättelyä varten. Näihin kuuluu taktisen tilanteen päättely epätäydellisestä tiedosta, taktisen tilanteen käyttäminen päätöksenteossa ja usean hahmon välinen koordinointi. (Millington ja Funge 2009, 494)

Etappitaktiikkaan (waypoint tactics) kuuluu etappien eli yksittäisten pelimaailman sijaintien käyttäminen taktiseen päätöksentekoon. Normaalisti etappeja käytetään reitinhaussa, mutta reitinhakuun käytettyä dataa laajentamalla on etapit pystytty ottamaan käyttöön myös muissa tehtävissä. Reitinhaussa yksittäinen etappi on esitetty yhtenä solmuna reitinhakuverkossa. Tähän solmuun on myös liitetty algoritmin vaatimaa dataa, kuten yhteydet, painot ym. Etappien taktista käyttöä varten solmuihin on lisättävä dataa, ja lisättävä data riippuu siitä, mihin etappeja käytetään. (Millington ja Funge 2009, 494-495)

Etappeja, joita käytetään taktisten sijaintien kuvaamiseen, voidaan kutsua ”kokoontumispisteiksi” (rally point). Kokoontumispisteitä käytettiin esimerkiksi simulaatioissa (erityisesti sotasimulaatiot), joissa ne olivat määrätty turva-alue, jonne hahmot pystyivät pakenemaan

ollessaan alakynnessä tulitaistelussa. Samaa periaatetta käytetään myös oikean maailman taistelusuunnitelmissa: kun joukkue hyökkää vihollista vastaan, on sillä ainakin yksi ennalta määrätty vetäytymispiste, jonne siirtyä tilanteen niin vaatiessa. Tällä tavoin hävitty taistelu ei aina pääty täydelliseen verilöylyyn. Peleissä yleisemmin käytetään taktisia sijainteja osoittamaan puolustuspisteitä tai suojia. Kun hahmo hyökkää pelaajan kimppuun, käyttää se suojia hyväkseen. Myös muita suosittuja taktisia sijainteja on olemassa, kuten sijainti tarkka-ammuntaan tai vaikkapa varjoisat sijainnit, jonne pääsee piiloon. Taktisia sijainteja on lukematon määrä, ja riippuen pelistä voivat hahmot käyttää useita eri taktiikoita, joita varten on omat etappinsa. (Millington ja Funge 2009, 494-495)

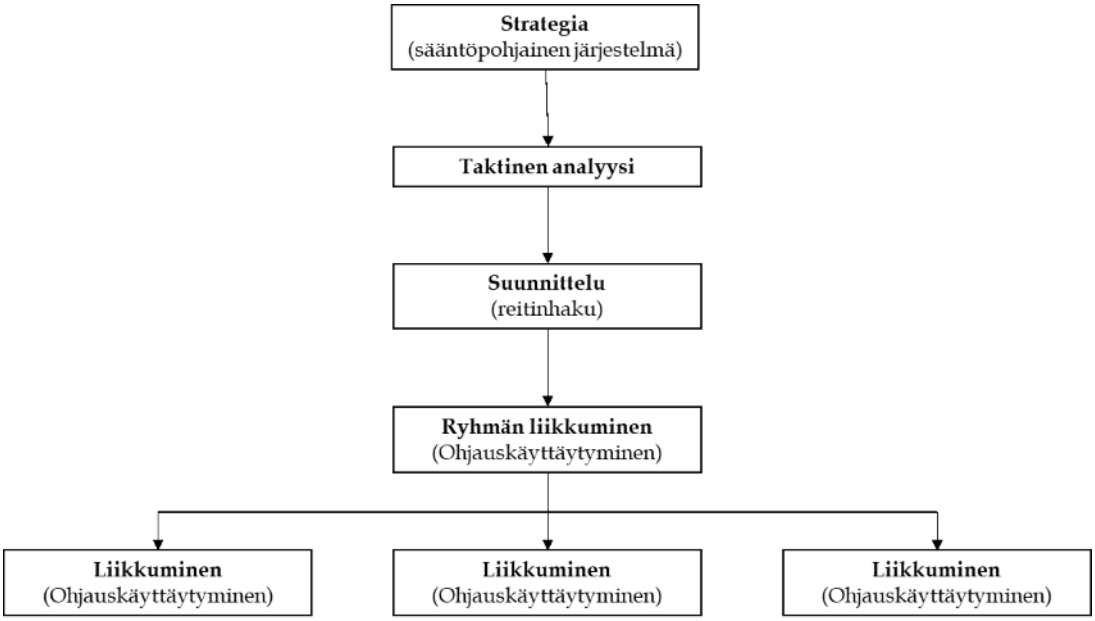
Kaiken tyyppistä taktista analyysiä kutsutaan usein vaikutuskartoiksi (influence map). Vaikutuskartat on luotu ja ovat suurimmaksi osaksi käytössä strategiapeleissä, joissa tekoäly seuraa omia ja pelaajan hallussa olevia alueita. Samoja tekniikoita on myös käytetty joukkuepohjaisissa ammuskelupeleissä ja MMO (massively multiplayer online) peleissä. Sotatilanteissa lähes vastaavaa analyysiä kutsutaan maastoanalyysiksi (terrain analysis). Vaikutuskarttojen ja etappistrategioiden pääperiaatteiden välillä on yllättävän vähän eroavaisuuksia. Jos kaikki nämä toteutettaisiin erikseen esimerkiksi ammuskelupelissä, olisi tuloksena hyvin samankaltaiset strategiakehykset. (Millington ja Funge 2009, 518)

Pelialue täytyy jakaa paloihin taktista analyysiä varten. Palojen sisältämillä alueilla tulisi olla suunnilleen samat ominaisuudet mille tahansa taktiikalle, josta ollaan kiinnostuneita. Jos kiinnostuksen kohteena on varjot, tulisi jokaisen palan valaistus olla karkeasti samalla tasolla. Pelialueen jakamiseen on useita eri tapoja. Ongelma on hyvin samankaltainen kuin reitinhaussa, ja kaikkia samoja lähestymistapoja voidaan käyttää. Koska taktisen analyysin juuret ovat vahvasti RTS-peleissä (Real-Time Strategy), enemmistö nykyisistä toteutuksista pohjautuu laattapohjaiseen (tile-based) ruudukkoon. (Millington ja Funge 2009, 518)

Taktinen reitinhaku on myös tärkeässä asemassa pelikehityksessä. Sen avulla voidaan saada aikaan hyvinkin vakuuttavia tuloksia, kun esimerkiksi joukkue liikkuu ympäristössä ottaen huomioon taktisen ympäristönsä, käyttäen hyväksi suojaa ja vältellen vihollisen tulilinjoja ja selviä väijytyspisteitä. Vaikka taktinen reitinhaku kuulostaakin monimutkaiselta, se ei suuresti eroa tavallisesta reitinhausta. Ainoa muutos algoritmissa tulee kustannusfunktioon (cost function), johon lisätään taktinen tieto ja välimatka tai aika. (Millington ja Funge 2009, 553)

Yhä useammassa pelissä useat hahmot tekevät yhteistyötä tehtävien suorittamiseksi. Myös tekoälyn yhteistyö pelaajan kanssa on lisääntynyt. Monissa peleissä tämä on toteutettu siten, että pelaaja antaa käskyjä ryhmän jäsenille ja he suorittavat käskyn parhaansa mukaan. Tekoäly on kuitenkin muuttunut siten, että ryhmän jäsenet pyrkivät itsenäisesti tukemaan pelaajaa tämän toimintojen mukaan ja tällä tavoin pyrkivät edesauttamaan tehtävän suorittamista. (Millington ja Funge 2009, 559)

Monitasoinen tekoäly (kuvio 7) sisältää käytöksiä useilla eri tasoilla: jokaisella hahmolla on oma tekoälynsä, ryhmällä on yksi yhteinen tekoäly, joka sisältää eri algoritmeja yksittäiseen hahmoon verrattuna, ja lisää tasoja voi olla esimerkiksi usean ryhmän tasolla. Taktiset algoritmit jaetaan kaikkien hahmojen kesken. Hahmot pyrkivät ymmärtämään pelitilanteen ja siten mahdollistamaan laajan päätöksenteon. Myöhemmin yksittäiset hahmot voivat tehdä omat päätöksensä tämän yleiskuvan pohjalta. (Millington ja Funge 2009, 559)

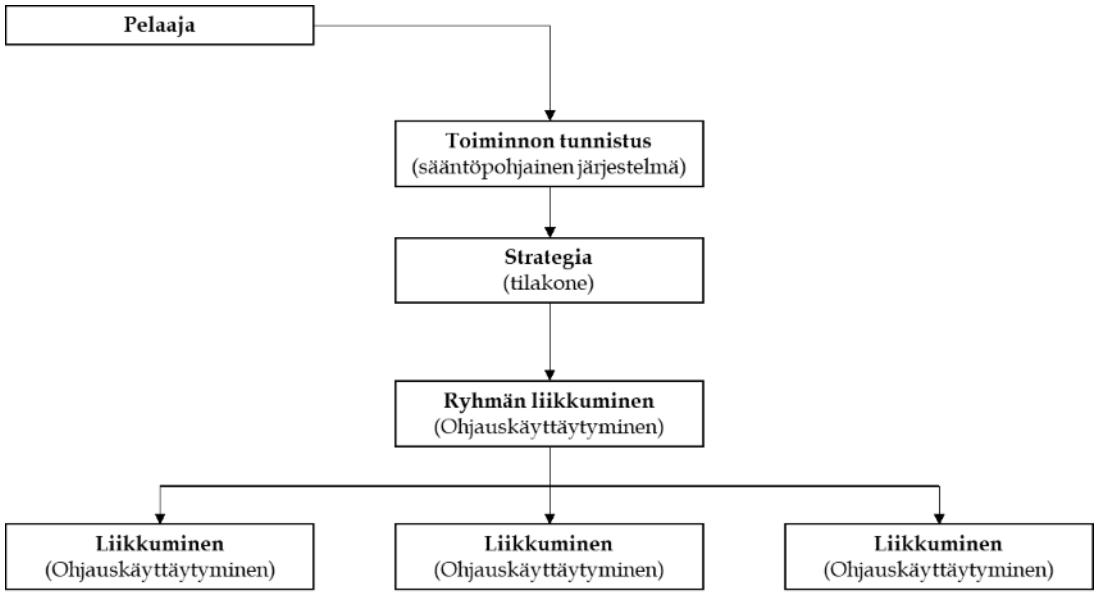


Kuvio 7. Esimerkki monitasoisesta tekoälystä (Millington ja Funge 2009, 560)

Monitasoinen tekoäly voi toimia lukemattomilla eri tavoilla. Esimerkiksi tekoälyn korkein taso tekee päätöksen, välittää sen alemmalle tasolle, joka tekee annetun päätöksen pohjalta oman päätöksensä ja niin edelleen, kunnes päätösketju saapuu alimmalle tasolle (ns. top-down lähestymistapa). Toisaalta alimman tason tekoälyalgoritmit voivat olla oma-aloitteisia

ja välittää päätöksensä ylemmälle tasolle, joiden avulla ylemmän tason algoritmit voivat tehdä päätöksensä (ns. bottom-up lähestymistapa). (Millington ja Funge 2009, 560)

Vaikka monitasoinen tekoäly toimiikin erinomaisesti ryhmä- ja joukkuepohjaisissa peleissä, se ei toimi kovinkaan hyvin pelaajan ollessa osa ryhmää. Monitasoista tekoälyä täytyy muokata siten, että pelaaja toimii korkeimman tason päätöksentekijänä (kuvio 8). Jos pelaaja on hierarkian huipulla, tekoälyn ohjaamat hahmot pohjaavat toimintansa siihen, mitä ne luulevat pelaajan haluavan, eikä johonkin muuhun korkean tason päätöksentekokerrokseen. Tämä ei tarkoita, että hahmot pystyvät ymmärtämään mitä pelaaja haluaa. Tällä tavoin hahmojen toiminnot eivät ole ristiriidassa pelaajan toimintojen kanssa. (Millington ja Funge 2009, 562-563)



Kuvio 8. Esimerkki pelaajan huomioivasta monitasoisesta tekoälystä (Millington ja Funge 2009, 564)

Emergentti yhteistyö on toinen suosittu tapa toteuttaa ryhmätekoäly. Toisin kuin monitasoisessa tekoälyssä, hahmojen ei tarvitse koordinoida samalla tavoin, vaan ne voivat yksinkertaisesti ottaa huomioon, mitä muut hahmot tekevät ja tällä tavoin vaikuttaa yhtenäiseltä ryhmältä. Jokaisella hahmolla on oma päätöksentekojärjestelmä, joka ottaa huomioon toisen hahmon toiminnan. Jos yksi ryhmän hahmoista poistetaan, ei se suuresti vaikuta lopun ryhmän toimintaan. (Millington ja Funge 2009, 566)

2.5 Pelitekoäly hiiviskelypeleissä

Tässä luvussa keskityn tarkemmin hiiviskelypeligenren (stealth genre) tekoälyyn. Ensin esittelen yleisesti tekoälyn erikoispiirteitä hiiviskelypeleissä, jonka jälkeen esittelen muutamia tärkeitä genreen kuuluvia pelejä ja niiden tekoälyn toteutusta.

Hiiviskelypelien genre on oikeastaan useiden eri genrejen alagenre, kuten seikkailu, toiminta, simulaatio ja roolipeli. Hiiviskelyä pelimekaniikkana on käytetty useissa eri peleissä. Pelien perusideana on silmittömän räiskimisen sijaan vältellä vastustajia ja suorittaa erilaisia tehtäviä mahdollisimman huomaamattomasti ja hiljaisesti. Yleensä tällaisissa peleissä pelaajalla on useita eri mahdollisuuksia edetä ja ratkaista erinäisiä tilanteita. Tunnetuimpia genren pelisarjoja ovat mm. *Metal Gear Solid* (Konami), *Splinter Cell* (Ubisoft), *Hitman* (Warner Bros Games) ja *Thief* (Square Enix). (Alkaisy 2011)

2.5.1 Erikoispiirteitä

Aiemmissa luvuissa esittelin pelitekoälyn koostumusta (kuvio 1) ja sitä, miten eri tekoälykomponentteja toteutetaan. Kuten aiemmin mainitsin, kaikkia tekoälykomponentteja ei aina tarvita tekoälyä toteutettaessa. Hiiviskelypelit ovat toimintapelien alagenre, jossa pelaajan on tarkoitus vältellä vastustajia suorittaessaan erilaisia tehtäviä. Toimintapeleille tyypillisesti strategiakomponenttia ei yleensä tarvita muutamia poikkeuksia lukuun ottamatta.

Tekoälyn keskiössä on vastustajien ”aistien” simulointi. Tekoäly voi havaita pelaajan käyttäen näkö- ja kuuloaistejaan. Näköaistin simulointi on yleisin ja itsestään selvin aisti, joka tekoälyvastustajilla on. Tekoälyllä on yleensä annettu tietty näkökenttä ja -etäisyys, jonka ulkopuolelle tekoäly ei näe kääntymättä tai liikkumatta. Tekoäly pystyy myös usein havaitsemaan erilaisia merkkejä pelaajasta, kuten esimerkiksi pelaajan jäljet lumihangessa tai ruumiit, joita pelaaja ei ole piilottanut. Näköaisti toimii usein myös asteittain. Jos pelaaja on heikosti nähtävissä esimerkiksi pimeyden takia, pelaajaa ei ole vielä ”löydetty”, vaan tekoäly joutuu tarkistamaan tarkemmin näkemänsä esimerkiksi liikkumalla lähemmäs pelaajaa.

Useissa peleissä pelaajalle on annettu erilaisia työkaluja tekoälyn näkökentästä poissapysymiseksi. Pelaaja voi esimerkiksi piiloutua esteiden taakse, kaappien sisälle, pöytien alle tai pelaaja voi yksinkertaisesti liikkua piilosta toiseen tekoälyn katsoessa muualle. Joissain pe-

leissä pelaaja voi myös käyttää hyväksi ympäristöään pukeutumalla valeasuun tai sopivaan maastokuvioon. Myös pimeyteen piiloutuminen on tärkeä työkalu monessa genren pelissä. Näkö on ainoa aisti, jonka avulla tekoäly pystyy havaitsemaan ja pysäyttämään pelaajan. Kuulo toimii ”hälyttävänä” aistina, jonka lisäksi tekoälyltä vaaditaan näköyhteys pelaajaan, jotta pelaaja tulisi ”löydetyksi”.

Tekoäly pystyy kuulemaan esimerkiksi askeleita tai aseen laukauksia. Äänien kuulemiseen vaikuttavat etäisyys äänen lähteestä ja ympäristön äänet, joihin pelaajan tuottamat äänet mahdollisesti hukkuvat. Tekoäly pystyy seuraamaan äänien lähteitä ja tällä tavoin mahdollisesti näkemään pelaajan.

Pelaajan löydettyään tekoäly alkaa jahdata pelaajaa, kunnes pelaaja on päihitetty tai pelaaja kadottaa jahtaajat. Tekoälyllä täytyy olla siis tieto siitä, missä pelaaja on viimeksi nähty. Jos tekoäly kadottaa pelaajan eikä pelaajaa löydy viimeksi nähdystä pisteestä etsimisen jälkeen, palaa tekoäly normaaliin käytökseen.

On myös tärkeää, että tekoälyä on helppoa ja mielenkiintoista seurata. Tätä varten tekoälyhahmon animaation ja käytöksen tulisi olla riittävän selkeää ja ennakoitavaa, jotta pelaaja pystyy riittävän helposti välttelemään tekoälyä. Esimerkiksi katseen suunnan pitäisi olla helposti havaittavissa tekoälyhahmon pään asennosta.

2.5.2 Toteutuksia eri peleissä

Tässä luvussa esittelen tekoälyn toteutuksia seuraavissa genren pelissä: *Thief: The Dark Project* (Eidos Interactive, 1998), *Tom Clancy's Splinter Cell Blacklist* (Ubisoft, 2013) ja *Alien: Isolation* (Sega, 2014). Pelit ovat hyvin erityyppisiä, ja erityisesti Thiefillä on ikäeroa muihin yli kymmenen vuotta. Kuitenkin pelien tekoälyn toteutuksissa on paljon yhteistä.

2.5.2.1 Thief: The Dark Project

Thief (kuva 1) on ensimmäisestä persoonasta kuvattu hiiviskelypeli, joka sijoittuu synkkään fantasiamaailmaan. Pelissä ohjataan mestarivaras Garrettia, jonka tavoitteena on varastaa mahdollisimman paljon arvotavaraa erilaisista kohteista samalla vältellen nähdyksi tulemis-

ta. Garrettilla on käytettävissä useita eri työkaluja, kuten pamppu, miekka ja jousi. Tärkein työkalu Garrettille on kuitenkin pimeys, jonka avulla pelaaja pysyy vartijoilta näkymättömissä. Pelaaja pystyy myös manipuloimaan valonlähteitä sammuttamalla tai sytyttämällä kynttilöitä, takkoja ja soihtuja.

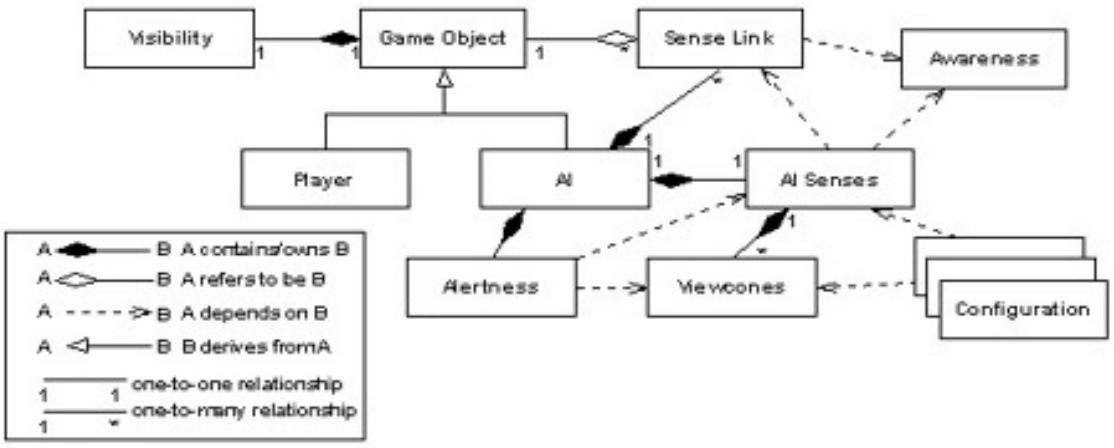
Kuva 1. Thief: The Dark Project kuvakaappaus

Thiefin tekoälyn päävaatimuksina oli luoda erittäin tarkasti säädettävä aistijärjestelmä, joka toimii yhteen useiden tilojen kanssa. Jotta pelikokemus olisi hauska, turvan ja vaaran välistä harmaata aluetta laajennettiin, mikä on useissa tavanomaisissa toimintapeleissä hyvin vähäinen. Tällä tavoin pelistä saatiin paljon viihdyttävämpi, kun tilasta toiseen siirtyminen ei ollut niin binääristä. (Leonard 2003)

Toissijaisena tavoitteena aistijärjestelmän oli oltava aktiivinen paljon useammin ja sen täytyi operoida useammissa objekteissa verrattuna tyypilliseen toimintapeliin. Pelin aikana pelaaja pystyy manipuloimaan pelimaailmaa siten, että tekoälyn pitäisi huomata nämä muutokset jopa silloin, kun pelaaja ei ole lähettyvillä. Nämä vaatimukset aiheuttivat haasteita pelin

suorituskyvyn suhteen. (Leonard 2003)

Pelin aistijärjestelmä (kuvio 9) perustuu näköalueiden ja säteensuuntauksen (raycasting) avulla toteutettuun näköjärjestelmään ja yksinkertaiseen kuulojärjestelmään, josta löytyy rajapinnat optimointia, pelimekaniikkoja ja pseudo-aistidataa varten. Suurin osa aistidatan keräämisestä on erillään päätöksentekoprosessista, joka toimii datan perusteella. (Leonard 2003)

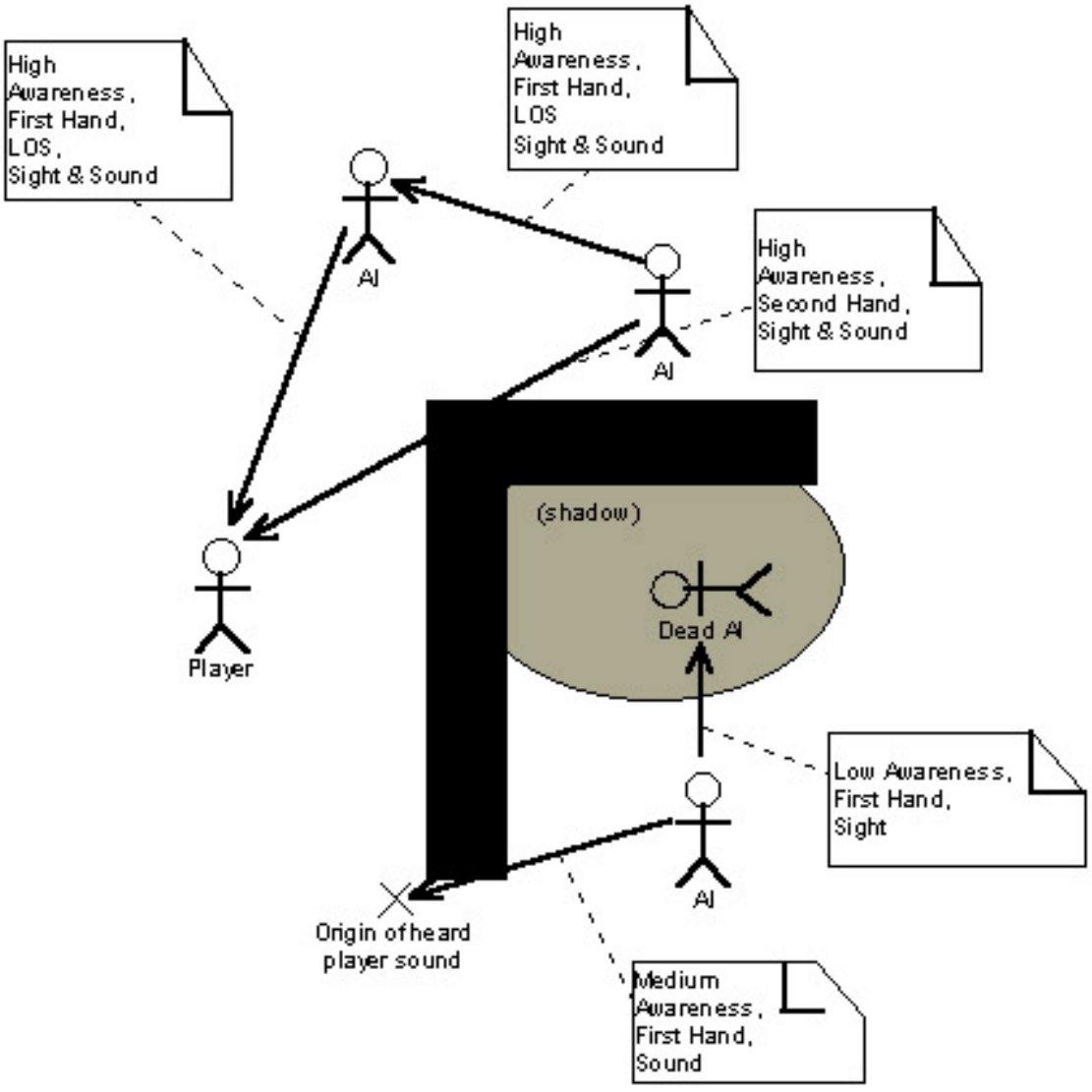


Kuvio 9. Peruskomponentit ja -suhteet (Leonard 2003)

Järjestelmän suunnittelu ja datan kulkeminen järjestelmän läpi johtavat juurensa pitkälti siitä, että kyseessä on informaation keräämiseen tarkoitettu järjestelmä. Se on muokattavissa ja säädettävissä, mutta sen tuloste on silti vakaata ja ymmärrettävää. Tässä järjestelmässä tekoälyn aisteja kutsutaan “tietoisuudeksi”. Tietoisuutta ilmaisee joukko erillisiä tiloja, jotka edustavat tekoälyn varmuutta läsnäolosta, sijainnista ja mielenkiinnon kohteena olevan objektin identiteetistä. Erilliset tilat ovat ainoa suunnittelijalle näkyvä sisäinen järjestelmä, ja ne korreloivat korkeamman tason tekoälyn valppaustilan kanssa. Valppaustilat ovat samantapaisia kuin tietoisuustilat. Tekoälyn valppaustila syötetään takaisin aistijärjestelmälle eri muodoissa järjestelmän käyttäytymisen muuttamiseksi. (Leonard 2003)

Tietoisuus on sijoitettu aistilinkkeihin (kuvio 10), jotka yhdistävät joko annetun tekoälyn johonkin toiseen pelin entiteettiin tai sijaintiin. Nämä suhteet pitävät kirjaa pelille relevanteista aistimisen yksityiskohdista (aika, sijainti, näköyhteys jne.) sekä välimuistiin talletetuista ar-

voista, joita käytetään laskuissa ajattelusyklien aikana. Aistilinkit ovat tekoälyn ensisijainen muisti. Linkkejä voidaan välittää muille tekoälyille puheen ja havainnoinnin välityksellä. Tätä kuitenkin rajoitetaan ohjaimilla, jotta tietoisuus ei leviä kaikkialle pelialueella. Pelilogiikka pystyy myös manipuloimaan aistilinkkejä perusprosessoinnin jälkeen. (Leonard 2003)



Kuvio 10. Aistilinkit (Leonard 2003)

Jokaisella “mielenkiintoisella” objektilla on luontainen näkyvyysarvo, joka on täysin katsojista riippumaton. Näkyvyysarvon tarkkuus ja päivittämisen tiheys skaalataan pelitilasta ja objektin ominaisuuksista riippuen, jotta arvon laskentaan käytettävä prosessointiaika pysyisi

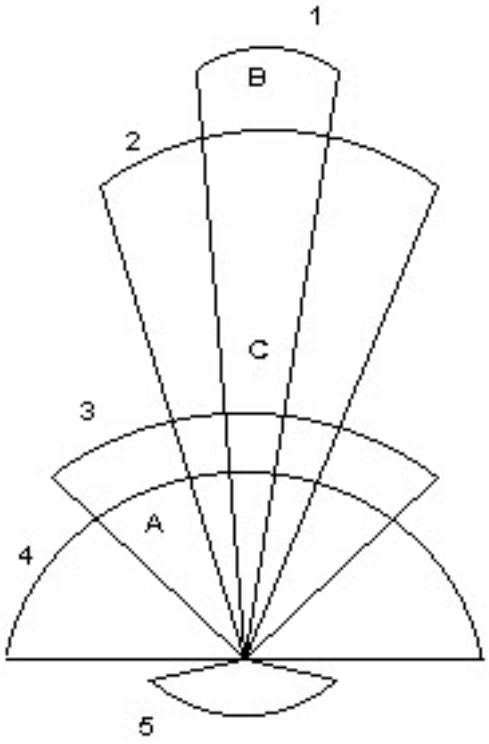
budjetin puitteissa. (Leonard 2003)

Näkyvyys määräytyy entiteetin valaistuksen, liikkeen ja altistumisen (koko, erottuminen muista objekteista) mukaan. Näiden tarkoitus on läheisesti sidottu pelin vaatimuksiin. Esimerkiksi pelaajan valaistus määräytyy pelaajan läheisyydessä olevan lattian valaistuksen mukaan. Tällä tavoin pelaajalla on objektiivinen ja helposti havaittava tapa ennakoida omaa turvassa pysymistä. (Leonard 2003)

Pelin näkökenttä koostuu useista järjestetyistä kolmiulotteisista "näkökartioista" yhden kaksiulotteisen näkökentän sijaan. Näkökartiot kuvataan XY-kulmana, Z-kulmana, pituutena, joukkona parametrejä, jotka kuvaavat sekä tarkkuutta että herkkyyttä erilaisille ärsykkeille ja relevanssia tekoälyn valppauden mukaan. Näkökartiot orientoituvat tekoälyn pään asennon mukaan. (Leonard 2003)

Minä hetkenä tahansa tekoäly näkee jonkin objektin, ainoastaan ensimmäinen näkökartio, jossa objekti on sisällä, otetaan huomioon aistilaskennassa. Jokaisen näkökartion oletetaan tuottavan jatkuvan tulosteen riippumatta siitä, missä kohtaa näkökartiota kohde sijaitsee. Tämä yksinkertaistaa laskentaa ja auttaa pelattavuuden säätämisessä. (Leonard 2003)

Esimerkiksi kuviossa 11 tekoälyllä on viisi näkökartiota. Objekti pisteessä A arvioidaan käyttäen kartiota 3. Pisteiden B ja C arviointiin käytetään kartiota 1. Identtiset näkyvyysarvot tuottavat saman tuloksen. (Leonard 2003)



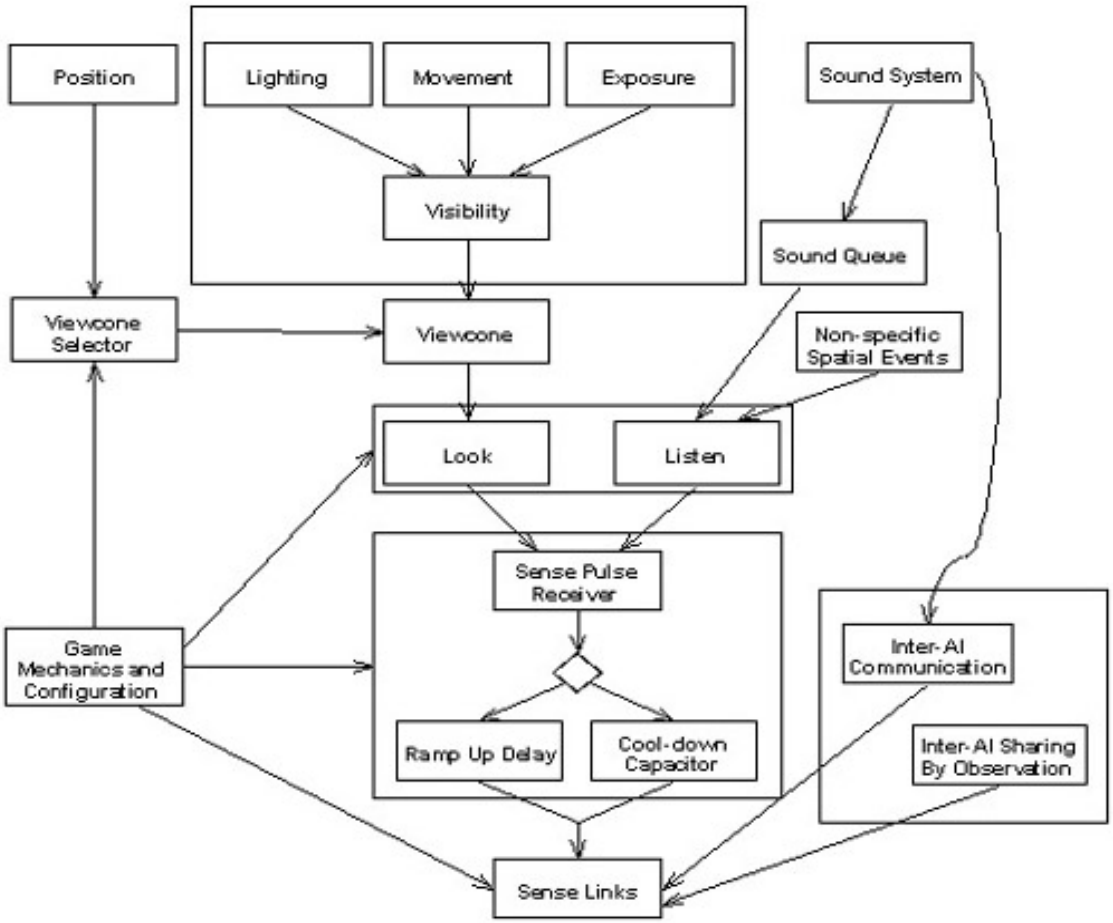
Kuvio 11. Näkökartiot ylhäältäpäin kuvattuna (Leonard 2003)

Mielenkiintoisia kohteita luodatessa aistit määrittävät ensin mitä näkökartiota, jos mitään, käytetään kohdetta arvioidessa. Tämän jälkeen kohteen luontainen näkyvyys ja näkökartio syötetään katseheuristiikan läpi, jonka tulosteena saadaan erillinen tietoisuusarvo. (Leonard 2003)

Usean näkökartion ratkaisuun päädyttiin, jotta pystyttiin simuloimaan suoraa näköä, reuna-näköä ja mahdollisuutta havaita, onko kohde samalla tasolla vai ylä- tai alapuolella. Kartio 5 kuviossa 11 on suunnattu katsomaan taakse. Se on herkkä liikkeelle ja simuloi pelaajan aistimista, jos tämä seuraa tekoälyä liian lähellä pelaajan ollessa muutoin huomaamaton. (Leonard 2003)

Aistien hallintajärjestelmä on suunniteltu joukoksi komponentteja (kuvio 12), jotka ottavat syötteenä rajatun ja hyvin määritellyn joukon dataa ja jotka antavat tulosteena vieläkin rajatumpia arvoja. Jokaisen vaiheen on tarkoitettu olevan itsenäisesti skaalautuvia pelin proses-

sointivaatimusten mukaisesti. Kyseinen skaalautuva “kerrosarkkitehtuuri” kykenee erittäin tehokkaaseen suorituskykyyn. (Leonard 2003)



Kuvio 12. Informaation kulku (Leonard 2003)

Aistijärjestelmän ydin toteuttaa heuristiikan näkyvyyden, äänitapahtumien, senhetkisten tietoisuuslinkkien, asetusdatan ja senhetkisen tekoälyn tilan hyväksymiseksi ja mielenkiinnon kohteena oleva objektin yksittäisen tietoisuusarvon tulostamiseksi. Nämä heuristiikat toimivat mustana laatikkona, jota tekoälyn kehittäjä säätää pelin kehityksen aikana. (Leonard 2003)

Näkö on toteutettu suodattamalla objektin näkyvyysarvo tilanteeseen sopivan näkökartion läpi, mikä muokkaa tulosta yksittäisen tekoälyn ominaisuuksien mukaan. Tavallisissa tapauksissa käytetään yksinkertaista säteensuuntausta näköyhteyden tarkistamista varten. Mo-

nimutkaisemmissa tapauksissa (esimerkiksi pelaajan tapauksessa) käytetään useita säteensuuntauksia, jotta tekoälyn avaruudellinen suhde kohteeseen saadaan mukaan, kun kohteen altistumista arvioidaan. (Leonard 2003)

Thief sisältää myös monimutkaisen äänijärjestelmän, jossa sekä renderöity että renderöimätön ääni on merkitty semanttisella datalla ja levitetään pelialueen kolmiulotteisen geometrian läpi. Kun ääni saapuu tekoälylle, se saapuu äänilähteen todellisesta suunnasta ja ääni on myös merkattu vaimennetuilla tietoisuusarvoilla, mahdollisesti välittäen tietoa muilta tekoälyiltä. Äänet yhdistävät muita tietoisuutta aikaansaavia asioita, jotka ilmentyvät tietoisuussuhteina eri sijainteihin. (Leonard 2003)

Kun katse- ja kuunteluoperaatiot on suoritettu, niiden tietoisuustulokset annetaan metodille, joka on vastuussa jaksottaisten aistipulssien vastaanottamisesta ja niiden muuntamisesta yksittäiseksi tietoisuussuhteeksi, jonka yksityiskohdat talletetaan asiaankuuluvaan aistilinkkiin. Kyseisen prosessin data on täysin erillistä aiemmin kuvatusta analogisesta datasta. Prosessin tarkoituksena on luoda, päivittää tai poistaa aistilinkkejä oikeita tietoisuusarvoja käyttäen. (Leonard 2003)

Prosessi on kolmivaiheinen. Ensin ääni- ja näköparametreja verrataan, toinen julistetaan hallitsevaksi ja hallitsevasta parametrista tulee tietoisuuden arvo. Ylimääräinen data sisällytetään aistitapahtuman yhteenvetoon. (Leonard 2003)

Toiseksi, jos tietoisuuspulssi on suurempi aiempiin lukemiin verrattuna, se suodatetaan aikapohjaisen suodattimen läpi, joka päättää todellisesta tietoisuuden lisääntymisestä. Aikaviive on vain senhetkisen tilan ominaisuus. Tällä tavoin reaktioon saadaan viivettä ja anteeksiantavuutta. Kun aikaraja on ohitettu, tietoisuus etenee tavoitetilaan kulkematta välillisten tilojen kautta. (Leonard 2003)

Lopuksi jos uusi pulssiarvo on senhetkisten lukemien alapuolella, tietoisuus hiipuu vähitellen. Tietoisuus hiipuu jonkin aikaa käyden läpi kaikki välilliset tilat. Tämä pehmentää tekoälyn käytöstä kun mielenkiinnon kohdetta ei enää pystytä havaitsemaan. Täällä mekanismilla ei kuitenkaan hallita tekoälyn valppautta. (Leonard 2003)

Jos mielenkiinnon kohde ei enää tuota pulsseja, aistit perustavat tietyn asteisen vapaan tietä-

myksen, joka skaalautuu tekoälyn tilan pohjalta. Tämä mekanismi antaa vaikutelman tekoälyn päättelykyvystä, kun objekti on kadonnut sen näkökentästä. Näin myös vältetään tekoälyn “huijauksen” paljastumista pelaajalle. (Leonard 2003)

2.5.2.2 Splinter Cell Blacklist

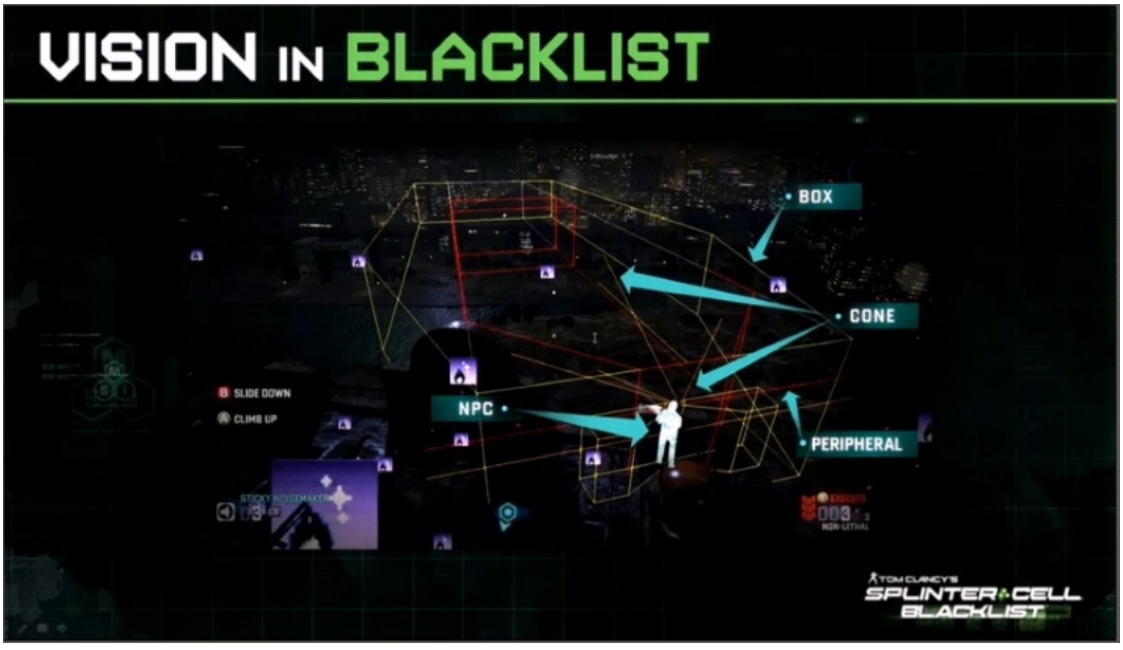
Blacklist (kuva 2) on Splinter Cell-sarjan viimeisin osa, jonka pitkäaikainen päähenkilö Sam Fisher kuuluu Yhdysvaltain salaiseen tiedusteluhaaraan Fourth Echeloniin. Samin tehtävänä pelissä on pysäyttää terroristijärjestö Blacklist, ennen kuin järjestön suunnitelmat toteutuvat. Pelaajalla on käytössään uusinta ja melko futuristista teknologiaa, jonka avulla terroristeja pystyy neutralisoimaan hiljaisesti tai halutessaan hieman äänekkäämminkin. Kuten Thiefissä, pelaaja pystyy käyttämään pimeyttä hyväkseen pysytellessään poissa tekoälyn näkyviltä. Peli ei seuraa perinteisiä hiiviskelypelejä siinä mielessä, että peliä on täysin mahdollista pelata tavallisena kolmannen persoonan räiskintäpelinä, mutta pelin mekaniikat nojaavat silti vahvasti hiiviskelyyn ja huomaamattomaan etenemiseen.

Kuva 2. Splinter Cell Blacklist kuvakaappaus

Blacklistin tekoäly perustuu räätälöityihin havaitsemis- ja tietoisuusmalleihin, joiden tavoitteena on olla reilu pelaajaa kohtaan, antaa yhtenäistä palautetta pelaajalle tämän toimista

(animaatiot, puhe, näytön HUD (heads-up display) elementit) ja osoittaa älykkyyttä. Näiden kolmen tavoitteen välillä on kuitenkin ristiriitoja: miten tehdä tekoälystä älykäs pitämällä se samalla reiluna ja tulkittavana? Tärkeämpää pelaajan kannalta onkin, että tekoäly vaikuttaa uskottavalle. Edellä mainittuihin malleihin kuuluvat näkö, ympäristö, ääni, sekä sosiaalinen ja kontekstuaalinen havaitseminen ja tietoisuus. (Walsh 2014)

Pelin näkömallista (kuvio 13) löytyvät perinteiset näkökartiot hyvin samantapaisesti kuin aiemmin mainitussa Thiefissä, joiden avulla tekoäly pystyy näkemään tarkasti ja havaitsemaan pelaajan melko nopeasti. Lisäksi näkömalliin kuuluvat suuret ruumisarkun muotoiset näköalueet, jotka määrittävät tekoälyn koko näköalueen ja simuloivat kaukonäköä. Näiden näköalueiden avulla voidaan määrittää, kuinka nopeasti pelaaja tulee nähdyksi riippuen pelaajan etäisyydestä, asennosta, pelin vaikeusasteesta ja tekoälyn arkkityypistä. Pelin suoritusaikana näköalueet tarkastetaan järjestyksessä ja ensimmäistä näköaluetta, jossa pelaaja on läsnä, käytetään havaitsemisajan arviointiin. Perustilassa näköalue orientoituu hahmon kehon suunnan mukaisesti, kun taas hälytystilassa se orientoituu pään asennon mukaisesti. (Walsh 2014)



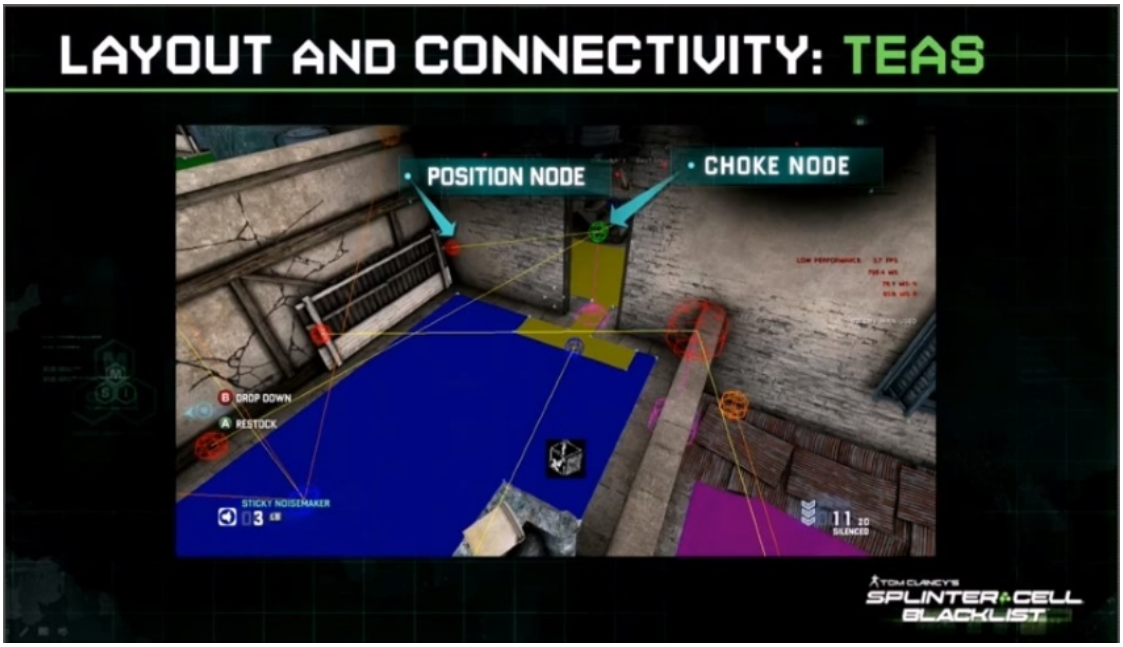
Kuvio 13. Näköalueet (Walsh 2014)

Tämän lisäksi pelaajan tulee olla näkyvillä ja valaistuna tullakseen nähdyksi. Tätä varten pe-

lissä tarkastetaan pelaajan pelihahmon kahdeksan eri “luuta” säteensuuntauksen avulla (kuvio 14). Havaitsemisen käynnistymiseksi luita tulee olla näkyvillä tietty määrä pelihahmon asennon mukaan. Pelaajalle myös viestitään ruudulla näkyvällä mittarilla, että hänet ollaan näkemässä. (Walsh 2014)

Kuvio 14. Luiden tarkastaminen säteensuuntauksella (Walsh 2014)

Tekoäly on myös tietoinen ympäristöstään ja ympäröivistä objekteista. Tekoäly ymmärtää, miten tilat ovat rakentuneet, mistä löytyvät ikkunat ja ovet, minne pääsee suojaan ja niin edelleen. Tämän ansiosta tekoäly pystyy esimerkiksi “järkeilemään” millaisessa asemassa pelaaja mahdollisesti on. Tätä varten kehitettiin TEAS-järjestelmä (Tactical Environment Awareness System) (kuvio 15). Kyseinen järjestelmä on hyvin samantyyppinen mitä kuvailin aiemmin strategiajärjestelmistä. TEAS koostuu sijaintisolmuista, jotka linkittyvät toisiinsa pullonkaulasolmujen (choke node) kautta. Pullonkaulasolmut esittävät erilaisten alueiden, kuten huoneiden liittymäkohtia (esim. ovet ja ikkunat). Tämän tiedon avulla tekoäly pystyy esimerkiksi vahtimaan tärkeitä ovia ja tarvittaessa vetäytymään suojaan toisessa huoneessa sijaitsevalle sijaintisolmulle. Tällä tavoin saatiin aikaan tekoälyn paljon uskottavampi käyttäytyminen. TEAS:ia käytettiin myös koordinoitujen sisääntulojen ja hierarkkisen reitinhaun toteutuksessa, sekä äänen etäisyyden laskemisessa. (Walsh 2014)



Kuvio 15. TEAS-järjestelmä (Walsh 2014)

Tekoäly pystyy myös havaitsemaan eri objektit ja niiden tilojen muutokset (esim. avattu ovi tai rikottu lamppu). Tätä varten kehitettiin muuttunut toimijajärjestelmä (changed actor). Kun toimijan tila muuttuu, siitä luodaan tapahtuma. Ensimmäinen tämän tapahtuman huomaava tekoäly oman tilansa perusteella saattaa alkaa tutkimaan tapahtumaa. Järjestelmän avulla saatiin helposti luotua voimakas vaikutelma tekoälyn kyvystä havainnoida muutoksia ympäristössä. (Walsh 2014)

Blacklistin äänimallissa jokaisella äänitapahtumalla on etäisyys ja tärkeysaste. Jos tekoäly on äänen kantaman sisällä, ääni tulee kuulluksi ja reagoi ääneen riippuen äänen tyypistä, läsnä olevista tekoälyistä ja senhetkisestä tilasta. Äänen matka suhteessa tekoälyyn lasketaan TEAS-järjestelmän avulla. TEAS:n avulla pystytään saamaan reitti eri tilojen läpi tekoälyn ja äänilähteen välillä ja tällä tavoin saatiin laskettua äänilähteen kokonaisetäisyys. Pelisuunnittelun näkökulmasta oli kuitenkin tärkeää, että pelaajan tuottamat äänet eivät aina tulleet kuulluksi ja että pelaajan näkökulmasta tekoälyn kuulo tuntuu uskottavalta, joten tekoäly, joka ei ole kuvassa ja on riittävän kaukana usein kuulee hieman huonommin verrattuna pelaajan näkökentässä olevaan. (Walsh 2014)

Lisäksi pelissä on käytössä BARKS-järjestelmä, jonka avulla tekoäly pystyy esimerkiksi

kommentoimaan pelaajan toimintaa (esim. “Kuulen askelia!”) tai kommunikoimaan muiden tekoälyjen kanssa. Järjestelmä on kolmikerroksinen, jossa ensimmäinen kerros sisältää hyvin spesifejä kommentteja, kun taas toinen ääripää sisältää yleisempiä kommentteja. BARKS:n avulla tekoälyn puhe on paljon vaihtelevampaa, mikä taas antaa tekoälystä älykkäämmän vaikutelman. Tekoälyllä on myös sosiaalinen ja kontekstuaalinen tietoisuus, jonka avulla esimerkiksi kahden tekoälyn keskustelun katkettua keskustelua pystytään jatkamaan luontevasti oikeasta kohtaa. Sosiaalisen tietoisuuden avulla tekoäly myös huomaa, jos toinen tekoälyhahmo on kadonnut pelaajan toimien takia. (Walsh 2014)

Tekoälyjä hallitsee ohjaava tekoäly ja tilanteen vaatiessa se valitsee tekoälyistä yhden, joka johtaa muita. Tekoälyt pystyvät “koordinoimaan” yhdessä taktiikoita pelaajan kukistamiseksi. Esimerkiksi yksi tekoäly antaa suojatulta toisen heittäessä kranaatin pelaajan oletettuun sijaintiin. Myös tällainen käytös on sosiaalisen tietoisuuden ansiota. (Walsh 2014)

2.5.2.3 Alien: Isolation

Isolation (kuva 3) sijoittuu Ridley Scottin alkuperäisessä *Alien*-elokuvassa (1979) luomaan scifi-maailmaan 15 vuotta elokuvan tapahtumien jälkeen. Pelaaja ohjaa pelissä ensimmäisestä persoonasta elokuvan päähenkilön, Ellen Ripleyn tytärtä Amandaa. Pelaajan tavoitteena on selvitä hengissä ulos Sevastopol avaruusasemalta vältellen tappavaa Alienia, seonneita androideja ja muita ihmisiä. Pelin keskiössä on yksi Alien, jota pelaaja ei pysty päihittämään tavanomaisilla aseilla. Pelaaja voi käyttää tulta ajaakseen Alienin vain hetkellisesti pois tai harhauttaa sitä eri välineillä. Alien on taas hyvin arvaamaton ja tarkka-aistinen. Se metsästää pelaajaa pitkin Sevastopolin käytäviä ja kuulee pelaajan tuottamat äänet, joiden perusteella se pystyy paikantamaan pelaajan. Jos Alien saa pelaajan kiinni, peli päättyy.

Kuva 3. Alien: Isolation kuvakaappaus

Koska kyseessä on kauhupeli, pelin rytmittäminen on avainasemassa hyvän pelikokemuksen saavuttamiseksi. Siksi pelaaja ei saisi olla paniikissa koko aikaa, vaan Alienin luomaa uhkaa täytyy välillä vähentää. Muutoin pelaaja saattaa tottua uhkaavaan tilanteeseen ja peli tuntuu tylsältä. Myös liiallinen koordinoitu säikyttely menettää tehonsa hyvin nopeasti. Tätä varten peliin on otettu vaikutteita alkuperäisen Alien-elokuvan lisäksi muista klassisista kauhu- ja seikkailuelokuvista. Lopputuloksena on ns. psykopaattisen onnellisen sattuman (psychopathic serendipity) filosofia, jonka muukaan Alien päätyy usein oikeaan paikkaan oikeaan aikaan. Vaikka pelaaja onkin piilossa ja Alien ei pysty näkemään tätä tai tiedä pelaajan tavoitetta, pystyy se silti sotkemaan pelaajan suunnitelmat. (Thompson 2016)

Alienin tekoälyn toteutuksessa käytetään kaksitasoista käytöksenhallintajärjestelmää: makro- ja mikrotekoälyä, joista makro on korkean tason ohjaava tekoäly ja mikro on itse Alienin tekoäly. Ohjaava tekoäly seuraa jatkuvasti pelaajan ja Alienin tilaa, kun taas Alienin tekoäly on reaktiivinen ja “aistien” varassa toimiva tekoäly, joka reagoi pelaajan tekemisiin ja ohjaajan käskyihin. Ohjaavan tekoälyn tehtävänä on osoittaa Alien määräajoin pelaajan suuntaan paljastamatta pelaajan tarkkaa sijaintia, josta Alien aktiivisesti etsii pelaaja. (Thompson 2016)

Ohjaavan tekoälyn vastuulla on niin sanotun “uhkamittarin” hallitseminen. Mittari antaa käsityksen siitä, kuinka paljon järjestelmä painostaa pelaajaa. “Uhan” määrä lasketaan käyttämällä tietoa siitä, kuinka lähellä pelaaja on, kuinka paljon aikaa on kulunut pelaajan läheisyydessä ja kuinka kauan Alien on ollut pelaajan näkökentässä ja liikkeentunnistimessa. Ohjaava tekoäly kasvattaa uhkaa lähettämällä Alienin lähelle pelaajan sijaintia, kun taas uhan tason ollessa tarpeeksi korkea, Alien lähetetään pois joksikin aikaa. (Thompson 2016)

Alienilla on myös tehtäväjärjestelmä, joka määrää Alienin tehtävät ja tehtävien suorittamisen sijainnit. Jokaisella tehtävällä on myös tärkeysaste, minkä mukaan Alien mahdollisesti pysäyttää senhetkisen tehtävänsä ja aloittaa seuraavan korkeamman tärkeysasteen tehtävän tai suorittaa senhetkisen tehtävän loppuun. Tehtäväjärjestelmä mahdollistaa Alien toimimisen aktiivisessa ja passiivisessa tilassa. Aktiivisessa tilassa Alien etsii pelaajaa jonkin tapahtuman takia (esim. pelaajan tuottama ääni, pelin tuottama tapahtuma, ohjaavan tekoälyn määräys) tai toteuttaa pelin vaatiman tapahtuman. Passiivisessa tilassa Alien taas vetäytyy ilmastointikanaviin tai tutkimaan avaruusaseman muita osia. (Thompson 2016)

Alienin tekoäly on toteutettu käytöspuulla, jossa on yli sata solmua, ja puun ylimmällä tasolla on noin kolmekymmentä solmua, jotka määrittelevät suoritettavan käytöksen. Nämä ylimmän tason solmut jakautuvat useisiin alipuihin, jotka määräävät Alienin eri alikäytökset. Pelin alussa osa käytöspuusta on lukittu, ja lukitut osat aukeavat joko pelin edetessä tai pelaajan toimien mukaan. Tästä syystä Alien vaikuttaa oppivan pelaajan tekemisistä. Käytöspuun osien avautuminen ei kuitenkaan liity koskaan pelaajan kuolemaan, sillä se saattaisi antaa Alienille epäreilun edun. (Thompson 2016)

Saadessaan tehtävän ohjaavalta tekoälyltä, Alien käyttää nopeaa ja tehokasta reitinhakujärjestelmää, jonka heuristiikat auttavat määrittämään Alienin tiettyjen sensoreiden arvot. Alienilla on joukko sensoreita, jotka mahdollistavat pelaajan askelten, aseiden laukausten ja liiketunnistimen äänen havaitsemisen. Alien ei kuitenkaan liiku optimaalisesti ympäristössä, vaan sen liikkuminen muistuttaa enemmän etsimistä tai metsästämistä. Tämän takia Alien välillä palaa takaisin samaa reittiä, mikä antaa vaikutelman siitä, että Alien haluaa tarkastaa alueen uudestaan tai epäilee itseään. Reitinhaussa käytetään kahta eri sijaintityyppiä. Alien yksinkertaisesti liikkuu kohti “etsimissijainteja” ja se pysähtyy katsomaan “paikkasijainneissa”. (Thompson 2016)

3 Pelikokemus

Tässä luvussa esittelen pelikokemuksen käsitettä ja sitä, miten pelitekoäly vaikuttaa pelikokemukseen. Käyn myös läpi, miten pelikokemusta tutkitaan.

3.1 Mistä pelikokemus koostuu?

Kirjassa *Rules of Play* (Salen ja Zimmerman 2004, 23.1) pelikokemuksen sanotaan olevan itse pelin pelaamista. Sen näkemistä, kuulemista, koskemista, haistamista ja maistamista, kuten myös vartalon liikuttamista pelin aikana, tunteiden tuntemista eri lopputulemien takia, muiden pelaajien kanssa kommunikointia ja normaalien ajattelumallien muuttamista pelin mukaisesti. Toisin kuin selkeät matemaattiset säännöt, pelin kokemuksellinen pelaaminen on sumeaa ja häilyvää. Pelaaja kuitenkin pelaa peliä kokemuksen puitteissa.

Pelikokemus on kokemus siinä missä mikä tahansa muu kokemus. Kokemuksella tavallisesti viitataan seuraaviin:

1. Objektin, ajatuksen tai tunteen ymmärtäminen aistien tai mielen avulla.
2. Tietämykseen tai taitoon johtavaa aktiivista tapahtumaan tai aktiviteettiin osallistumista.
3. Tapahtuma tai joukko tapahtumia, joihin on osallistuttu tai joiden läpi on eletty.

Toisin sanoen, kokemus on osallistumista. Jokainen peli luo omanlaisensa kokemuksen. Ei ole olemassa yhtä tiettyä kokemusta, jota kaikkien pelien tulisi yrittää tarjota. On kuitenkin olemassa merkityksellisen pelaamisen periaatteita, joita voidaan käyttää peleissä eri suunnittelukonteksteissa. (Salen ja Zimmerman 2004, 23.2)

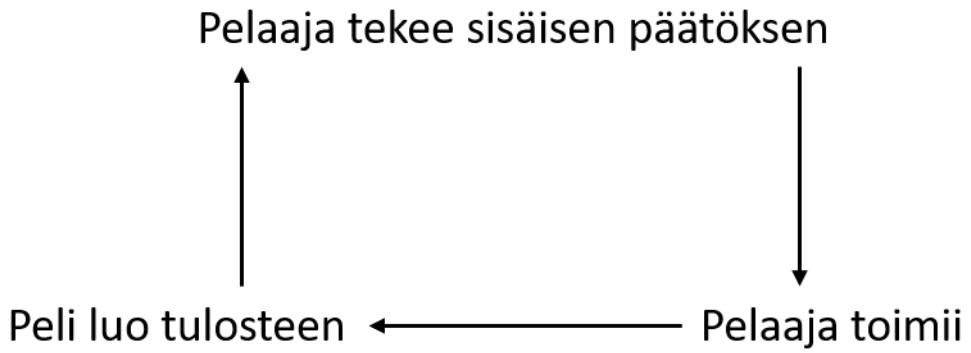
Pelaamisen kokeminen esiintyy niin monessa eri muodossa, että kaiken sisältävää listaa on mahdotonta laatia. Tämä ei kuitenkaan tarkoita, etteikö pelaamisen kategorisointiin käytettävä järjestelmä olisi hyödyllinen suunnitteluongelmia ratkaistaessa. (Salen ja Zimmerman 2004, 23.2) Kirjassa *Toys as Culture* (Sutton-Smith 2004, 69-72) esitetään malli, joka listaa psykologiset prosessit, joita käyttäen pelit koetaan. Vaikka malli käsittelee juurikin videopelejä, on se relevantti myös muita pelejä tarkasteltaessa. Mallin mukaan pelikokemukseen

kuuluu viisi elementtiä, jotka ovat

- *Visuaalinen tarkastelu:* visuaalinen havaitseminen erityisesti koko kuvaruutua tarkasteltaessa.
- *Auditiivinen erottelu:* pelin tapahtumien ja signaalien kuuntelu.
- *Motoriset reaktiot:* pelaajan suorittama fyysinen toiminto ohjainta käyttäessä.
- *Keskittyminen:* pelaamiseen kohdistuva tiivis keskittyminen.
- *Oppimisen aistimallit:* pelin rakenteen oppiminen.

Malli tarjoaa melko suppean listan elementeistä, joista pelikokemus koostuu. Visuaalinen tarkastelu ja auditiivinen erottelu edustavat pelaajan aistimuksellisia toimintoja, motoriset reaktiot pelaajan fyysistä toimintaa ja kaksi viimeistä elementtiä edustavat pelaajan sisäisiä kognitiivisia mekanismeja, jotka yhdistävät aiemmin mainitut syötteet ja reaktiot. Vaikka malli soveltuukin hyvin perinteisiin yksinpelikokemuksiin, ne eivät kuitenkaan kata kaikkia pelejä. Malli ei ota huomioon esimerkiksi sosiaalista kanssakäymistä, mikä on läsnä monissa tämän päivän peleissä. (Salen ja Zimmerman 2004, 23.2-23.3)

Mallin elementit voidaan kuitenkin abstrahoida, jotta malli olisi laajemmin sovellettavissa (kuvio 16). Mallin ytimessä ovat aistisyötteiden, reaktioiden ja pelaajan sisäisten mekaniikkojen suhde. Tämä kolmiosainen malli on hyödyllinen yleisrakenne, kun pelaajan pelikokemusta halutaan ymmärtää. Pelaajan tapa hahmottaa peli ja toimia sen sisällä tulee aina olemaan pelinsuunnittelukohtainen. Nämä yksityiskohdat sisältyvät suurempaan kokemuksen järjestelmään, johon kuuluu aina jonkinlainen aistisyöte, pelaajan reaktio ja pelaajan sisäinen kognitio. (Salen ja Zimmerman 2004, 23.2-23.3)



Kuvio 16. Pelikokemusmalli (Salen ja Zimmerman 2004, 23.3)

Pelinkehityksessä haasteena on, ettei pelikehittäjä suoraan suunnittele pelikokemusta. Sen sijaan pelaaminen on emergentti ominaisuus, joka nousee pelistä pelaajan ollessa vuorovaikutuksessa järjestelmän kanssa. Pelikehittäjä luo joukon sääntöjä, joita pelaaja noudattaa, tutkii ja manipuloi. Tällä tavalla pelaaja kokee pelin. Kehittäjä siis epäsuorasti suunnittelee pelikokemuksen suunnittelemalla pelin säännöt. (Salen ja Zimmerman 2004, 23.4)

Kehittäjä muokkaa kokemusta luomalla useita “toiminta- ja lopputulemaketjuja”, joista koostuu suurempi merkityksellisen pelaamisen järjestelmä erityisesti silloin, kun lopputulema on havaittavissa ja integroitu peliin kokonaisuudessaan. Pelit voidaan siis kehystää järjestelmiksi, joiden merkitys kumpuaa pelaajien tekemistä valinnoista johtuvasta kokemuksesta. Jokainen valintakomponentti on kokemuksen kannalta relevantti. (Salen ja Zimmerman 2004, 23.4)

Jokainen peli sisältää ydinmekaniikan, jonka pohjalta pelaaja on vuorovaikutuksessa peliympäristön kanssa, ja siten ydinmekaniikka on erittäin tärkeä pelikokemuksen kannalta. Ydinmekaniikka edustaa keskeistä pelaajan toimintaa, jota pelaaja suorittaa uudestaan ja uudestaan läpi pelin. Pelin aikana ydinmekaniikat luovat käyttäytymismalleja, jotka ilmentyvät pelaajalle kokemuksena. Ydinmekaniikan avulla pelaaja pystyy tekemään pelin kannalta merkityksellisiä päätöksiä ja tällä tavoin myös saavuttamaan merkityksellisen kokemuksen. Siksi onkin tärkeää tunnistaa pelin ydinmekaniikka suunnitteluprosessin alussa, vaikka se muuttuisikin kehityksen aikana. Ydinmekaniikan löytämisen avulla kehittäjä pystyy luomaan yhteenvedon pelin interaktiivisuudesta. Hyvin usein syy sille, miksi peli ei ole hauska, löytyy

pelimekaniikasta. (Salen ja Zimmerman 2004, 23.4-23.5)

Pelikokemuksen tärkein osa on pelaaja itse ja siksi onkin myös tärkeää pystyä luokittelemaan erityyppisiä pelaajia ja pelityylejä. Pelaaja voidaan luokitella viiteen eri luokkaan, jotka eivät ole toisiaan poissulkevia: saavutus (achievement), tutkimus (exploration), seurallisuus (sociability), dominointi (domination) ja immersio (immersion). Kyseisten luokkien avulla voidaan havainnollistaa pelaajan motivaatioita peliä pelattaessa ja sitä, millaisesta perspektiivistä pelaaja kokee pelin. (Hamari ja Tuunanen 2014)

3.2 Miten pelikokemusta tutkitaan?

Kirjassa Game Research Methods (Björk ja Lankoski 2015) esitellään eri tutkimusmenetelmiä, jotka soveltuvat hyvin pelikokemuksen tutkimiseen ja mittaamiseen. Tällaisia ovat esimerkiksi haastattelu, virikkeellinen haastattelu, täsmäryhmähaastattelu ja pelaajan kokemuksen audiovisuaalinen analyysi, joista viimeinen on kvantitatiivinen lähestymistapa, kun taas kolme ensimmäistä ovat kvalitatiivisia. Tässä luvussa käyn läpi näiden menetelmien hyviä ja huonoja puolia pelikokemusta tutkittaessa.

Haastattelu ei tuota kovinkaan yleistettäviä tuloksia. Sen avulla ei voi esimerkiksi tutkia, kuinka usein ihmiset tekevät jotain, tai kuinka todennäköinen jokin seuraamus on. Tämä johtuu suurilta osin siitä, että tutkimukseen käytettävä otos usein valitaan tutkimuksen tarkoitusten mukaisesti. Tutkija ei siis voi tehdä päätelmiä koko populaatiosta, sillä haastateltavat eivät edusta laajempaa ryhmää. Haastattelun avulla voidaan kuitenkin saavuttaa henkilökohtainen syvyys hyvinkin yksityiskohtaisesti ja pystytään kuvaamaan tietty pienempi ryhmä. (Björk ja Lankoski 2015, 93)

Yksilöhaastattelussa tutkijalla on korkeatasoinen hallinta haastattelutilanteessa, sillä haastateltavia on yksi, jonka kanssa on muodostettu yhteisymmärrys. Tutkija kykenee vaikuttamaan vahvasti keskustelun suuntaan, jos keskustelu tarvitsee ohjausta. Epäselviä kysymyksiä voi muotoilla uudestaan ja aiheesta eksyttäessä haastateltava voidaan hienovaraisesti ohjata takaisin aiheen pariin. Tutkijan täytyy kuitenkin käyttää aikaa yhteisymmärryksen muodostamiseksi ja tutkimuksen tarkoituksen selittämiseksi. Tutkijan täytyy myös kyetä pitämään haastattelun sisältö luottamuksellisena, eikä yksittäisen haastateltavan henkilöllisyys saa sel-

vitä kirjatuista tuloksista. (Björk ja Lankoski 2015, 96)

Virikkeellisessä haastattelussa haastateltavaa muistutetaan aiemmasta osallistumisesta (esim. pelin pelaamisesta) käyttämällä nauhoitteita (esim. äänite, video tai kuva). Pelien immersiivisyys saattaa hankaloittaa ajatteluprosessien muistamista jälkikäteen. Virikkeiden avulla haastateltavaa voidaan auttaa muistamaan kokemuksen aikaisia ajatusprosesseja. (Björk ja Lankoski 2015, 117)

Virikkeellisessä haastattelussa haastateltava joutuu kohtaamaan omat toimensa niiden tapahtuessa (nauhoituksien sallimissa rajoissa), mikä on selvä etu perinteiseen haastatteluun nähden. Kun haastattelu on käynnissä, haastateltava puhuu toimista, joita hän oikeasti teki, eikä muistamiseen liittyviä erheitä pääse yhtä helposti tapahtumaan, mikä vähentää haastattelun hypoteettisuutta. (Björk ja Lankoski 2015, 119)

Selvänä haasteena on kameran läsnäolo ja sen vaikutus haastateltavan käyttäytymiseen. Tämä saattaa aiheuttaa hermostuneisuutta ja haastateltava saattaa tuntea painetta suoriutua. Kameran läsnäolo vaikuttaa enemmän pinnalliseen käyttäytymiseen kuin pysyvämpiin käytösmalleihin. Yksilöiden on vaikeaa muuttaa pinnallista käyttäytymistä, sillä he eivät ole niistä tietoisia. Menetelmä on myös paljastava, sillä se auttaa osallistujaa antamaan oikeamman kuvan itsestään, koska hänen toimensa ovat nauhoitettu ja siten niitä on vaikeaa muuttaa. (Björk ja Lankoski 2015, 119-120)

Reliabiliteetin suhteen menetelmä on kaksiteräinen miekka, sillä ihmisillä on taipumus selittää asioita. Haastateltava voi pyrkiä järkeilemään toimiaan ja selittämään niitä ymmärtämättä omia ajatusprosessejaan. Toisaalta tutkimusaiheen antamat yleiset periaatteet ja haastattelun sisältämä järkeily saattavat johtaa laajempaan tutkimusmateriaaliin. (Björk ja Lankoski 2015, 120)

Täsmäryhmähaastattelu on ryhmäkeskustelu, joka keskittyy tiettyyn haastateltaville ja tutkijalle mielenkiintoiseen tai relevanttiin aiheeseen. (Björk ja Lankoski 2015, 133) Menetelmän vahvuutena on sen joustavuus. Keskustelut voidaan toteuttaa monella tapaa, osallistujien määrä voi vaihdella, voidaan käyttää erilaisia näytteenottotapoja ja niin edelleen. Täsmäryhmähaastattelut soveltuvat erityisen hyvin tutkittaessa aiheita, joista tiedetään tai ymmärretään vähän. Keskustelun aikana osallistujista tulee aiheen asiantuntijoita, joilta tutkija voi op-

pia. Tämän mahdollistaa ryhmäkeskusteluiden avoin rakenne ja osallistujien tasa-arvoisempi asema. Menetelmä voi olla myös voimaannuttava, sillä se antaa osallistujille mahdollisuuden keskustella yhteisistä kokemuksista ja ymmärtää, etteivät he ole yksin. (Björk ja Lankoski 2015, 136)

On kuitenkin muistettava, että ryhmät ovat pieniä ja tarkoin valittuja, eivätkä siten edusta suurempaa populaatiota. Tästä syystä menetelmän tulokset eivät ole helposti yleistettävissä. Ryhmiä voi olla myös haastavaa saada kasaan. (Björk ja Lankoski 2015, 136)

Pelikokemuksen audiovisuaalinen analyysi käyttää hyväksi pelin pelaajalle antamia audiovisuaalisia syötteitä ja mittareita pelikokemusta tutkittaessa. Syötteitä ja mittareita analysoimalla pystytään myös suoraan mitattavasti analysoimaan pelaajan yksilöllistä pelikokemusta. Menetelmän avulla pystytään tarkastelemaan dataa eri näkökulmista tutkimuskysymyksen mukaan ja käyttämään erilaisia audiovisuaalisia prosessointitekniikoita tutkimuskysymyksiin vastattaessa. (Björk ja Lankoski 2015, 207)

Analyysin toteuttaminen on kuitenkin melko monimutkainen tehtävä. Analysoitava pelivideo sisältää usein tuntikaupalla pelikuvaa, joka syntyy pelaajan yksilöllisestä lähestymistavasta tai pelityylistä, tavasta oppia asioita, ymmärtämis- ja havainnointikyvystä. Ennen analyysin toteuttamista pelin rakenne ja pelaajan suhteet pelin eri osa-alueisiin täytyy ymmärtää hyvin. (Björk ja Lankoski 2015, 207-208)

4 Peliprototyyppi

Tässä luvussa esittelen tutkimusta varten kehittämäni peliprototyypin. Esittelen prototyyppiä korkealla tasolla ja sen lisäksi käyn yksityiskohtaisemmin läpi prototyypin tekoälyjen toteutusta. Prototyypin lähdekoodi löytyy GitHubista (*https://github.com/Atte89/ProjectStealth*).

4.1 Prototyypin toteutus

Tutkimusta varten toteutin peliprototyypin (kuva 4) Unity-pelimoottorilla, joka pohjautuu Unityn omaan *Project: Stealth* –tutoriaaliin. Valitsin Unityn kehitysalustaksi, sillä se oli itselleni jo entuudestaan tuttu ja alustalle valmiina ollut pelipohja helpotti pelin kehittämistä huomattavasti. Kaikki tarvittavat resurssit olivat jo valmiina, ja pelin perusmekaniikat vaativat vain vähän viilausta. Kirjoitushetkellä alkuperäinen Unity-tutoriaali on vedetty pois vanhentuneena Unityn kauppapaikalta, eikä siksi ole enää virallisesti saatavilla.

Kuva 4. Peliprototyyppi kuvakaappaus

Suurin työ prototyypin suhteen oli saada se toimimaan uudemmalla Unityn versiolla, sillä *Project: Stealth* on tehty vanhemmalle Unityn versiolle (4.x), eikä taaksepäin yhteenso-

pivuus uudemmissa versioissa ole täydellinen. Unityn keskustelupalstoilta kuitenkin löytyi hyviä ohjeita erinäisten yhteensopivuusongelmien korjaamiseksi. Suurin alkuperäiseen peliin tekemäni muutos liittyy tekoälyyn.

Project: Stealth on pelinä melko perinteinen hiiviskelypeli, jossa pelaajan tehtävänä on ohjata hahmo hissille samalla vältellen ympäristössä partioivia robotteja. Pelaajan hahmo pystyy liikkumaan normaalisti ja hiipien, käyttämään ympäristöstä löytyviä painikkeita, keräämään avainkortteja sekä kiinnittämään robottien huomion itseensä huudahtamalla. Pelissä käytetään lintuperspektiiviä kameran kuvakulmana ja kamera seuraa pelaajan hahmoa tämän liikkuessa.

Peliympäristönä (kuva 5) toimii maanalainen bunkkeri. Bunkkerista löytyy robottien lisäksi esteitä, joita pelaajan tulee vältellä. Turvakameran näkökenttään joutuessaan tai laseraitaan koskiessaan pelaaja laukaisee hälytyksen, jolloin vartijarobotit tulevat tarkastamaan hälytyksen laukaisseen turvalaitteen. Edetäkseen pelissä pelaajan täytyy etsiä ympäristöstä kytkinpaneeleja, joita käyttämällä laseraidat voi sammuttaa. Pelaaja ei pysty vaikuttamaan turvakameroiden toimintaan, joten pelaajan täytyy kiertää kameran näkökenttä, joka on heijastettuna maahan punaisena valona. Paetakseen bunkkerista pelaajan täytyy käyttää bunkkerin yläosasta löytyvää hissiä. Hissi on kuitenkin lukittu, joten pelaajan on etsittävä bunkkerista avainkortti, jolla hissin ovet aukeavat.

Kuva 5. Peliprototyyppin peliympäristö

Peliprototyyppiin kuuluu kolme eri variaatiota samasta pelikentästä. Erilaista kussakin variaatiossa on vartijarobottien käyttäytyminen ja lukumäärä. Pelin käynnistyessä käyttäjä ohjataan alkuvalikkoon, josta valitaan pelattava variaatio. Pelin päättyessä valittu variaatio alkaa aina uudestaan. Pelattavaa variaatiota pystyy vaihtamaan palaamalla alkuvalikkoon.

Moniulotteisen pelitypologian (Aarseth, Solveig ja Sunnanå 2003) mukaan prototyypin voi kuvata seuraavasti:

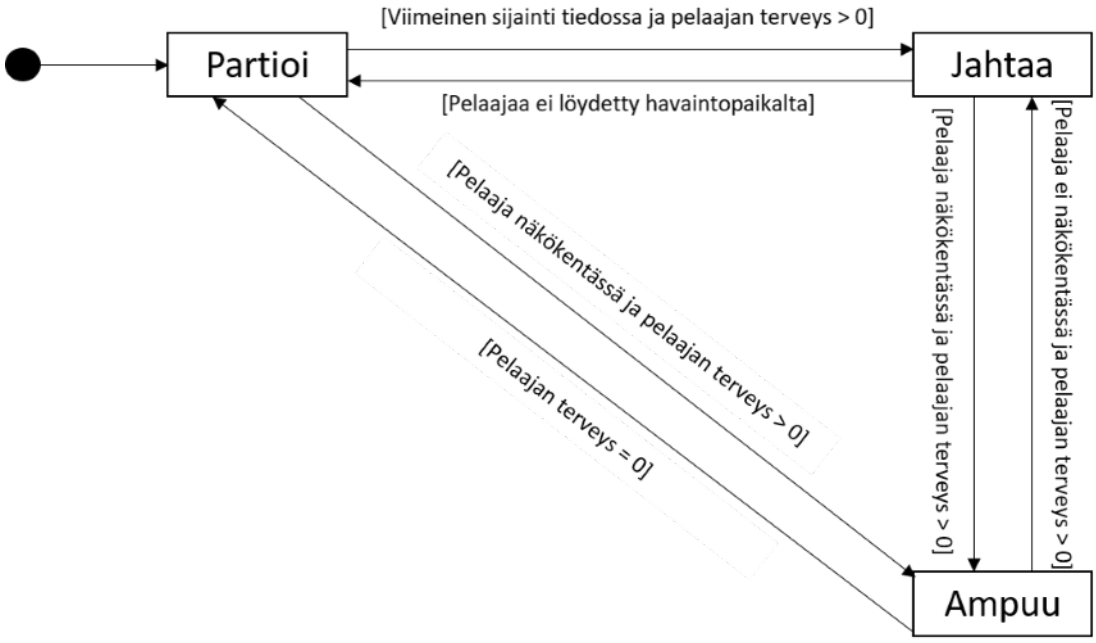
- **Tila**:
  - Perspektiivi – vaeltava (vagrant):
    - prototyypin perspektiivi on sidottu pelaajan ohjaamaan hahmoon
  - Topologia – geometrinen:
    - pelaaja pystyy vapaasti liikkumaan pelialueen sallimissa rajoissa
  - Ympäristö – staattinen:
    - pelaaja ei pysty vaikuttamaan pelin ympäristöön

- **Aika**:
  - Tahti – reaaliaikainen:
    - prototyypin tapahtumat tapahtuvat reaaliajassa
  - Esitystapa – mimeettinen (mimetic):
    - prototyypissä tapahtuva toiminta jäljittele oikeaa maailmaa ajallisesti
  - Tavoitteellisuus – rajallinen:
    - prototyypissä on selkeä tavoite, eli pelialueelta pakeneminen hissiä käyttäen
- **Pelaajarakenne** – yksipelaajainen:
  - peliin osallistuu ainoastaan yksi pelaaja
- **Hallinta**:
  - Muuttuvuus – staattinen:
    - pelaajaan hahmo ei muutu pelin aikana
  - Tallennettavuus – ei tallennettava:
    - peli alkaa aina alusta epäonnistumisen jälkeen
  - Määräytyneisyys – määräytynyt/ei määräytynyt
    - määräytyneisyys vaihtelee variaatioiden mukaan
    - variaatio A:ssa tekoäly toimii aina samalla tavalla ja kulkee aina samaa reittiä
    - variaatio B:ssä ja C:ssä tekoäly valitsee satunnaisesti kuljettavan reitin
- **Säännöt**:
  - Topologiset säännöt – kyllä/ei:
    - säännöt vaihtelevat variaatioiden mukaan
    - variaatio A:ssa ja B:ssä ei ole sijaintiin liittyviä sääntöjä
    - variaatio C:ssä tekoäly pysyttelee samalla alueella pelaajan kanssa
  - Aikaperusteiset säännöt – ei:
    - prototyypissä ei ole aikaperusteisia sääntöjä
  - Tavoiteperusteiset säännöt – kyllä:
    - Pelaajan tavoitteena on paeta käyttämällä hissiä

4.2 Tekoäly peliprototyypissä

Peliprototyypin robottivartijoiden tekoäly on toteutettu melko perinteisellä ja yksinkertaisella tilakoneella. Valitsin tilakoneen, koska prototyyppi itsessään on yksinkertainen eikä vaadi tekoälyltä monimutkaisia käyttäytymismalleja. Lisäksi tekoälylle on toteutettu näkö- ja kuulosensorit ympäristön havainnointia varten.

Tilakoneessa (kuvio 17) on kolme tilaa: partioi (patrolling), jahtaa (chasing) ja ampuu (shooting). Aloitustilana toimii partioi-tila, jossa tekoäly kulkee ympäristöön ennalta määrättyjen etappien (waypoint) välillä. Ampuu-tilassa tekoäly ampuu pelaajan hahmoa ja jahtaa-tilassa tekoäly "jahtaa" pelaajaa juoksemalla viimeiselle sijainnille, jossa pelaajan hahmo havaittiin. Partioi-tilasta voidaan siirtyä suoraan kahteen muuhun tilaan. Ampuu-tilaan siirrytään, jos pelaajan hahmo on tekoälyn näkökentässä ja pelaajan hahmolla on enemmän kuin nolla terveyspistettä. Ampuu-tilasta siirrytään takaisin partioi-tilaan, kun pelaajan hahmon terveyspisteet putoavat nollaan. Partioi-tilasta siirrytään jahtaa-tilaan, kun tekoälyllä on tieto viimeisestä havaitusta pelaajan sijainnista ja pelaajan hahmolla on enemmän kuin nolla terveyspistettä. Jahtaa-tilasta siirrytään takaisin partioi-tilaan, kun tekoäly saavuttaa viimeisimmän sijainnin, jossa pelaajan hahmo havaittiin, eikä kykene enää pelaajaa havaitsemaan. Jahtaa-tilasta siirrytään ampuu-tilaan, kun pelaaja on tekoälyn näkökentässä ja pelaajalla on enemmän kuin nolla terveyspistettä, ja vastaavasti ampuu-tilasta siirrytään jahtaa-tilaan, kun pelaaja poistuu tekoälyn näköpiiristä ja pelaajalla on enemmän kuin nolla terveyspistettä.



Kuvio 17. Tilakone vartijarobotille

Prototyypin koodissa tilakoneen toteutus näyttää tältä:

```
void Update()
{
    // If the player is in sight and is alive...
    if (enemySight.playerInSight && playerHealth.health > 0f)
        // ... shoot.
        Shooting();

    // If the player has been sighted and isnt dead...
    else if (enemySight.personalLastSighting !=
        lastPlayerSighting.resetPosition &&
        playerHealth.health > 0f)
        // ... chase.
        Chasing();

    // Otherwise...
    else
        // ... patrol.
        Patrolling();
}
```

Toteutuksessa käytetään Unityn `Update()` funktiota, jota kutsutaan aina jokaisen kehyksen päivityksen yhteydessä (Unity Technologies 2020). Jos pelin kehystaajuus (frame rate) on esimerkiksi 60 kehystä sekunnissa (frames per second tai FPS), kutsutaan `Update()` funktiota 60 kertaa sekunnissa.

Kuten aiemmin mainitsin, peliprototyyppi on jaettu kolmeen variaatioon samasta pelikentästä (taulukko 1). Tämän seurauksena pelaaja joutuu aina muuttamaan omaa strategiaansa eri variaatioita pelatessaan, sillä samoilla olettamuksilla ei pelikenttää aina välttämättä pysty läpäisemään. Variaatioiden erona on tekoälyn käyttämät partiointistrategiat, joihin vaikuttavat tapa valita seuraava etappi sekä etappien määrä ja sijoittelu.

Variaatio	Tekoälyjen määrä	Etappien määrä per tekoäly	Kaikki etapit käytettävissä	Partiointistrategia	Etapin valintastrategia	Kykenee palaamaan takaisin	Toteutustapa
A	3	4	Ei	Ennalta määrätty	Ennalta määrätty	Ei	Tilakone
B	2	13	Kyllä	Sattumanvarainen	Sattumanvarainen	Ei	Tilakone
C	1	18	Kyllä	Pysyttelee pelaajan lähellä	Sattumanvarainen	Kyllä	Hierarkkinen tilakone

Taulukko 1. Variaatiot

Variaatiossa A tekoälyllä on käytettävissään rajattu määrä etappeja (kuva 6), eikä tekoälyllä ole varaa valita seuraavaa etappia, vaan seuraava etappi on aina valmiiksi määritelty. Tämä on yksinkertaisesti toteutettu siten, että tekoälyllä on järjestetty lista etapeista, joita se iteroi järjestyksessä, ja tällä tavoin kulkee aina samojen etappien kautta samassa järjestyksessä. Tekoälyn rajatun liikkuvuuden paikkaamiseksi variaatiosta löytyy kolme vartijarobottia, joilla kullakin on omat eriävät etappilistat. Tekoälyskriptin `Patrolling()` funktio on seuraavanlainen:

```
void Patrolling()
{
    // Set an appropriate speed for the NavMeshAgent.
    nav.isStopped = false;
    nav.speed = patrolSpeed;

    // If near the next waypoint or there is no destination...
    if (nav.destination == lastPlayerSighting.resetPosition ||
        nav.remainingDistance < nav.stoppingDistance)
    {
        // ... increment the timer.
        patrolTimer += Time.deltaTime;

        // If the timer exceeds the wait time...
        if (patrolTimer >= patrolWaitTime)
        {
            // ... increment the wayPointIndex.
            if (wayPointIndex == patrolWayPoints.Length - 1)
                wayPointIndex = 0;
            else
                wayPointIndex++;

            // Reset the timer.
            patrolTimer = 0;
        }
    }
    else
        // If not near a destination, reset the timer.
        patrolTimer = 0;

    // Set the destination to the patrolWayPoint.
    nav.destination = patrolWayPoints[wayPointIndex].position;
}
```

`Patrolling()` funktio määrittää tekoälyn toiminnan partioi tilassa. Variaatio A:n tekoälyllä on tarkoitus matkia vanhempien hiiviskelypelien (esim. *Metal Gear Solid* (Konami, 1998) ja *Tom Clancy's Splinter Cell* (Ubisoft, 2002)) tekoälyjen käyttäytymistä.

Kuva 6. Tekoälyn reitti A-variaatiossa

Variaatiossa B tekoälyllä on taas käytettävissä kaikki pelikentästä löytyvät etapit. Jokaisella etapilla on lisäksi tieto siitä, mitkä muut etapit ovat sen sijainnista saavutettavissa. Tämän avulla tekoäly pystyy tekemään päätöksen siitä, mille etapille se seuraavaksi lähtee siirtymään. Tekoäly ei pysty kuitenkaan palaamaan takaisin omia jälkiään, joten sen on aina valittava jokin muu etappi kuin edellinen. Kun etappiin on linkitetty useampi kuin yksi etappi edellisen etapin lisäksi, tekoäly valitsee sattumanvaraisesti eri vaihtoehdoista seuraavan etapin. Tätä strategiaa käyttämällä pelaajan on astetta vaikeampaa ennakoida tekoälyn liikkumista tai sijaintia tietyn ajan kuluttua. Pelikenttään on lisätty useita etappeja lisää, mikä mahdollistaa tekoälyn laajemman liikkumisen. Tästä syystä B-variaatiossa on kaksi vartijarobottia kolmen sijaan.

Tekoälyskriptin `Patrolling()` funktion etapin valinta Variaatio B:n tekoälyn toteutuksessa on seuraavanlainen:

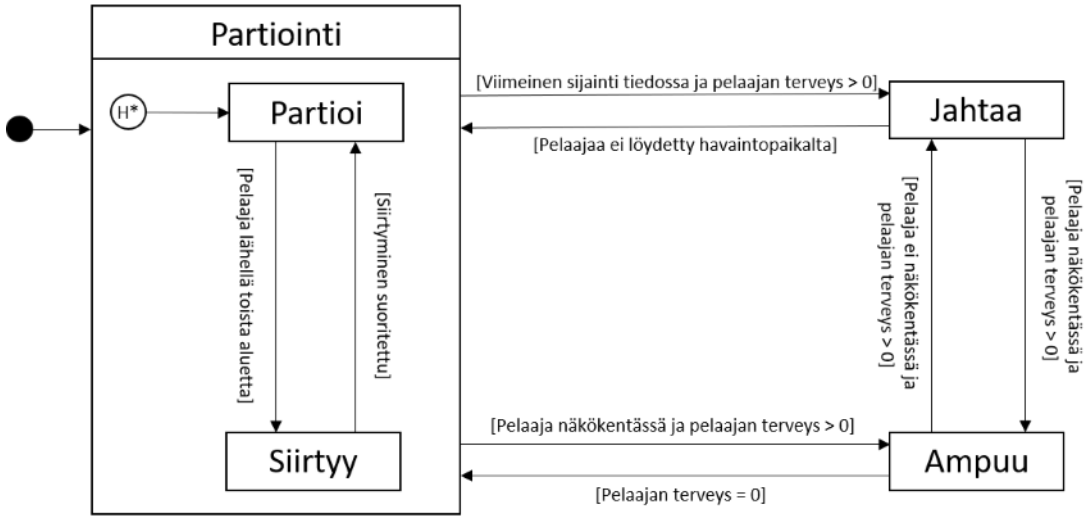
```
// If the timer exceeds the wait time...
if (patrolTimer >= patrolWaitTime)
{
    if (currentWaypoint == null)
    {
        currentWaypoint = GetClosestWaypoint(patrolWaypoints);
    }
    else
    {
        GameObject newWaypoint = lastWaypoint;
        GameObject[] linkedWaypoints =
            currentWaypoint.GetComponent<DoneWayPointGizmo>().linkedWaypoints;

        while (newWaypoint == lastWaypoint)
        {
            newWaypoint =
                linkedWaypoints[Random.Range(0, linkedWaypoints.Length)];
        }

        lastWaypoint = currentWaypoint;
        currentWaypoint = newWaypoint;
    }
    // Reset the timer.
    patrolTimer = 0;
}
```

Variaatiossa C pelikenttä on jaettu ylä- ja alaosaan, joilla molemmilla on omat erilliset etappinsa. Näitä kahta etappiryhmää yhdistävät ns. “siirtymäetapit”, jotka muodostavat reitin näiden kahden alueen välille. Alueen jakamisen tarkoituksena on mahdollistaa tekoälyn kyky keskittyä partioimaan kumpaa tahansa aluetta ja vaihtamaan aluetta tiettyjen ehtojen vallitessa. Variaatio C:ssä on ainoastaan yksi vartijarobotti, joka pyrkii aina partioimaan samalla alueella, jolla pelaaja on. Se vaihtaa aluetta, kun pelaajaa lähinnä oleva etappi kuuluu eri alueeseen, jossa robotti sillä hetkellä liikkuu. Tätä varten tilakonetta täytyi muuttaa lisäämällä

uusi siirtymätila, jonka aktivoituessa robotti vaihtaa aluetta käyttämällä siirtymäetappeja ja siirtymisen jälkeen jatkaa taas partioi-tilassa. Siirtymätilasta täytyy myös pystyä siirtymään samoihin tiloihin kuin partioi-tilasta, joten tilakoneesta muodostui hierarkkinen tilakone (kuvio 18). Lisäksi tekoäly pystyy palaamaan takaisin omia jälkiään, joten sen käyttäytymistä on entistäkin vaikeampaa ennakoida. Muutoin robotti valitsee seuraavan etapin samalla tavalla kuin variaatiossa B.



Kuvio 18. Hierarkkinen tilakone vartijarobotille

Koodissa hierarkkiseen tilakoneeseen siirtyminen vaati seuraavanlaisen muutoksen `Update()` funktioon:

```
void Update()
{
    // If the player is in sight and is alive...
    if (enemySight.playerInSight && playerHealth.health > 0f)
        // ... shoot.
        Shooting();

    // If the player has been sighted and isn't dead...
    else if (enemySight.personalLastSighting !=
        lastPlayerSighting.resetPosition &&
        playerHealth.health > 0f)
        // ... chase.
        Chasing();

    else if (doTransition)
        Transition();

    // Otherwise...
    else
        // ... patrol.
        Patrolling();
    }
```

Myös `Patrolling()` funktion etapin valinta muuttui seuraavasti:

```
// If the timer exceeds the wait time...
if (patrolTimer >= patrolWaitTime)
{
   if (currentWaypoint == null)
   {
       currentWaypoint = GetClosestWaypoint();

       if (upperPatrolWaypoints.Contains(currentWaypoint))
       {
           currentWaypoints = upperPatrolWaypoints;
       }
       else
       {
           currentWaypoints = lowerPatrolWaypoints;
       }
   }
   else
   {
       lastWaypoint = currentWaypoint;
       GameObject[] linkedWaypoints =
           currentWaypoint.GetComponent<DoneWayPointGizmo>().linkedWaypoints;
       currentWaypoint = linkedWaypoints[
           Random.Range(0, linkedWaypoints.Length)];
       while (!currentWaypoints.Contains(currentWaypoint))
       {
           currentWaypoint =
               linkedWaypoints[Random.Range(0, linkedWaypoints.Length)];
       }

   }
   // Reset the timer.
   patrolTimer = 0;
}
```

Lisäksi `Patrolling()` funktiossa tarkastetaan myös tarve vaihtaa aluetta:

```
if (currentWaypoints != null)
{
    GameObject waypointClosestToPlayer =
        lastPlayerSighting.WaypointClosestToPlayer();

    if (!currentWaypoints.Contains(waypointClosestToPlayer) &&
        !transitionWaypoints.Contains(waypointClosestToPlayer))
    {
        doTransition = true;
        lastWaypoint = currentWaypoint;
        currentWaypoints = transitionWaypoints;

        if (upperPatrolWaypoints.Contains(lastWaypoint))
        {
            currentWaypoint = startTransitionToLower;
            transitioningTo = lowerPatrolWaypoints;
        }
        else
        {
            currentWaypoint = startTransitionToUpper;
            transitioningTo = upperPatrolWaypoints;
        }

        return;
    }
}
```

Variaatio C:n tavoitteena on matkia *Alien: Isolation* -pelin tekoälyä, jossa Alien on ennaltaarvaamaton ja vaikuttaa aina löytävän itsensä pelaajan läheisyydestä.

Tekoälylle on toteutettu yksinkertaiset näkö- ja kuulosensorit käyttämällä yhtä pyöreää törmäyksentunnistinta (sphere collider) (kuva 7). Tekoälyn näkökenttä on 110 astetta eteenpäin. Jos pelaajan hahmo on törmäyksentunnistimen kanssa kosketuksessa, pelaajaan ja tekoälyn välille piirretään vektori. Jos kyseinen vektori osuu tekoälyn näkökenttään, testataan säteensuuntauksella, onko pelaajan ja tekoälyn välissä näköesteitä. Jos näköesteitä ei ole, tulee

pelaaja nähdyksi. Kuulo taas toimii siten, että jos pelaaja joko liikkuu normaalisti tai yrittää kiinnittää tekoälyn huomion huudahtamalla niin, että pelaajan hahmo on törmäyksen tunnistimen kanssa kosketuksissa, tulee pelaaja kuulluksi.

Kuva 7. Näkö- ja kuulosensori

5 Haastattelututkimus

Tässä luvussa esittelen tutkimussuunnitelman ja tutkimuksesta saadut tulokset. Käyn läpi tutkimuksessa käytettyjä metodeja ja tarkastelen tutkimustuloksia hyvin yleistetyllä tasolla. Lisäksi teen tulkintoja saaduista tuloksista ja siitä, miten saadut tulokset vastaavat tutkimuskysymykseen.

5.1 Tutkimussuunnitelma

Tämän tutkielman tutkimuskysymyksenä on se, miten tekoälyn käyttäytyminen vaikuttaa hiiviskelypelin haastavuuteen. Vastatakseni tutkimuskysymykseen suoritan pienimuotoisen haastattelututkimuksen. Valitsin laadullisen tuitkimusmenetelmän, koska se on joustavampi määrälliseen tutkimukseen verrattuna. Laadullinen tutkimus ei aina sisällä ennalta määrättyä hypoteesia. Tutkimusprosessi laadullisessa tutkimuksessa on hermeneuttinen, mikä sopii hyvin tutkimuskysymykseeni vastaamiseen. Lisäksi tutkimusongelmaa ja tutkimuskysymystä voi tarkentaa myös aineiston keräämisen jälkeen. (Puusa ja Juuti 2011, 88)

Tutkimuksessa haastateltava pelaa edellisessä luvussa esittelemääni peliprototyyppiä ja vastaa siihen liittyviin kysymyksiin. Tutkimukseen kuuluu myös pienimuotoinen kvantitatiivisen aineiston kerääminen. Kehitin tutkimusta varten haastattelukehyksen (liite 1), joka toimii haastattelun käsikirjoituksena. Kehys antaa mahdollisuuden jatkokysymyksille ja tarkentavalle keskustelulle haastateltavien kanssa. Kehys määrittää pääpiirteissään haastattelutilanteen kulun, joka etenee seuraavasti:

- Aloitus
- Variaatio A:n pelaaminen
- Kysymyksiä pelikokemuksesta
- Variaatio B:n pelaaminen
- Kysymyksiä pelikokemuksesta
- Variaatio C:n pelaaminen
- Kysymyksiä pelikokemuksesta ja lopetus

Data on pääasiassa kvalitatiivista, haastattelemalla kerättyä dataa, mutta lisäksi kerään kvantitatiivista dataa haastateltavan pelisuorituksesta. Mittaan pelisuorituksiin kuluneen ajan ja epäonnistumisten määrän. Voin verrata kvantitatiivista dataa kvalitatiiviseen dataan ja mahdollisesti löytää joko korrelaatiota tai ristiriitoja näiden kahden välillä. Kerään myös tietoa haastateltavan pelaajatyypistä käyttämällä Hamarin ym. (2014) esittämiä pelaajatyyppejä. Tämän avulla voin mahdollisesti ymmärtää erilaisia toimintamalleja, joita haastateltavat käyttävät pelitilanteessa.

Tutkimuksen kohderyhmänä toimii 5-7 haastateltavaa, joilla on jo ennestään kokemusta hiiviskelypeleistä. Haastateltavat vastaavat kaikkiin kysymyksiin täysin anonyymisti, ja käsittelen tuloksia siten, ettei niistä ole mahdollista tunnistaa haastateltua yksilöä. Tutkimuksen lopputuloksena on suuntaa antava ymmärrys siitä, miten tekoälyn käytös vaikutta haastavuuteen hiiviskelypelissä. Tulosta voidaan hyödyntää esimerkiksi jatkotutkimuksessa tai mahdollisesti pelikehityksessä.

5.2 Tutkimustulokset

Tutkimukseen osallistui viisi melko erityyppistä haastateltavaa. Kysyin kaikilta heidän omaa mielipidettänsä siitä, mihin Hamarin ym. (2014) esittämiin pelaajatyyppeihin he kokevat kuuluvansa. Pelaajatyypit ja heidän pelaamiaan hiiviskelypelejä olivat

- Haastateltava 1: Tutkimus/Saavutus
  - Spider-Man
- Haastateltava 2: Saavutus/Dominointi
  - The Elder Scrolls V: Skyrim
  - Divinity: Original Sin 2
  - Call of Duty: World at War
  - Men of War: Assault Squad 2
- Haastateltava 3: Seurallisuus/Dominointi/Tutkimus
  - The Elder Scrolls V: Skyrim
  - Dishonored
- Haastateltava 4: Dominointi
  - Dishonored
  - Deus Ex
  - Vampire: The Masquerade
  - Metal Gear Solid
  - Alien: Isolation
- Haastateltava 5: Immersio
  - Tom Clancy's Splinter Cell
  - Assassin's Creed

Kaikki haastateltavat pitivät keskinkertaisesta haasteesta. Kysyin heiltä, minkä vaikeusasteen he usein peleissä valitsevat, kun heillä on mahdollisuus valita, ja kaikki suosivat keskitason vaikeusastetta. Toisaalta myös mainittiin, että jos peli on mieleinen, vaikeusasteen nostaminen tekee pelistä usein entistäkin mielenkiintoisemman. Vastaavasti peleissä, jotka eivät ole kovinkaan mieluisia, liiallinen haaste ei välttämättä ole hyväksi.

Haastateltavien mielestä tekoäly hiiviskelypeleissä on melko usein epärealistinen, anteeksiantava ja jopa naurettava, eikä usein tarjoa kovinkaan suurta haastetta. Epärealistisen vaikutelman aiheuttaa heidän mielestänsä tekoälyn erittäin rajatut sensoriominaisuudet, joiden takia tekoälyltä jää usein monia asioita huomaamatta, ja tästä syystä tekoälyn toiminta vaikuttaa epärealistiselta. Tekoälyllä ei myöskään usein ole minkäänlaista sosiaalista tietoisuutta, joten he eivät useinkaan välitä muista neutralisoiduista tekoälyhahmoista.

Tekoälyn toimintaa pidettiin helposti ennakoitavana toistuvien liikkeiden ja vartiointirutiinien ansiosta. Joissain peleissä tekoälyn näkökenttä on selvästi indikoitu pelaajalle, mikä entisestään helpottaa pelaamista. Jos pelaaja tuli havaituksi, on hänellä useimmiten mahdollisuus ratkaista tilanne muullakin tavalla kuin hiiviskelemällä.

Joitain pelejä myös pidettiin haastavina. Esimerkiksi Dishonoredissa tekoälyhahmojen määrä kasvoi huomattavasti pelin loppua kohden ja hahmoilla oli käytettävissä korkeat robottijalat, jotka antoivat tekoälylle edun paremman näköalan muodossa. Myös joitain Assassin's Creedin tehtäviä pidettiin haastavina, mutta kuitenkin suoraviivaisina, sillä kaavamaisuutensa ansiosta tehtävät yleensä pystytään läpäisemään kokeilemalla ja oppimalla virheistä.

Aloituskysymysten jälkeen siirryimme pelaamaan peliprototyypin variaatioita. Taulukossa 2 näkyvät variaatiokohtaiset keskimääräiset peliajat ja yritysten määrät. Variaatio A oli kaikille ensimmäinen, ja kuten keskimääräisestä peliajasta huomaa, kesti sen läpäisy pisimmän aikaa. Tämä johtui pääosin siitä, että haastateltavat opettelivat prototyypin pelaamista ja pelialuetta kyseisen variaation aikana. Muissa variaatioissa pelaajat käyttivät keskimäärin huomattavasti vähemmän aikaa ja yrityksiä, sillä pelialue ja pelimekaniikat olivat jo tutut.

Variaatio	Käytetty aika (min)	Yritysten määrä	Haastavuus
A	23	16	5
B	8	6	4,5
C	2	1	4,2

Taulukko 2. Käytetty aika, yritysten määrä ja haastavuus keskimäärin per variaatio

Taulukossa 3 näkyy tarkemmin haastateltavakohtaiset tulokset haastattelussa kerätystä kvantitatiivisesta datasta. Taulukossa on haastateltavakohtaisesti esitetty pelaajatyyppi sekä variaatiokohtaiset tulokset, joihin kuuluu koettu haastavuus (asteikolla 1-10, missä 1 on vähiten haastava ja 10 on haastavin), käytetyt yritykset ja kulunut aika minuuteissa. Lisäksi taulukosta näkee jokaisen haastateltavan mielipiteen haastavimmasta, helpoimmasta ja viihdyttävimmästä variaatiosta.

Haastateltava	1	2	3	4	5
Pelaajatyyppi	Tutkimus/ Saavuttaminen	Saavuttaminen/ Dominointi	Seurallisuus/ Dominointi/ Tutkimus	Dominointi	Immersio
Variaatio A					
Haastavuus	7-8	5	4	5	4
Yritykset	15	17	26	4	19
Aika (min)	24	19	25	6	41
Variaatio B					
Haastavuus	6	4	8	3	1-2
Yritykset	14	4	11	2	1
Aika (min)	17	7	10	4	3
Variaatio C					
Haastavuus	8	2	2	2	7
Yritykset	1	2	1	2	1
Aika (min)	1	2	1	2	5
Haastavin variaatio	B	A	B	A	C
Helpoin variaatio	A	C	C	C	B
Viihdyttävin variaatio	C	C	A	B	C

Taulukko 3. Kvantitatiiviset tulokset

Kaikkien haastateltavien mielestä variaatio A tarjosi hyvin keskitasoisen haasteen. Haastateltavat kokivat, että tekoälyn käyttäytyminen oli helppoa havaita ja että variaatiossa täytyi edetä tekoälyn liikkumista tarkkaillen. Koska tekoälyn käyttäytyminen oli erittäin kaavamaista, haastateltavien mielestä variaation pystyi läpäisemään helposti kokeilemalla eri lähestymistapoja eri tilanteissa ja käyttämällä aina toimivaksi havaittua ratkaisua.

Variaatio B puolestaan jakoi haastateltavien mielipiteitä. Tekoälyn satunnaisen luonteen takia joidenkin haastateltavien kohdalla tekoäly ei ollut riittävästi läsnä oikeissa paikoissa, jolloin pelaajan oli helppoa edetä pelissä. Tällöin haastateltavat eivät myöskään kyenneet havaitsemaan tekoälyn käyttäytymistä.

Toisaalta tekoälyn satunnainen käyttäytyminen loi täysin uudenlaisen haasteen, joka vaati

pelaajalta enemmän sopeutumista erilaisiin tilanteisiin. Niiden haastateltavien mielestä, joiden kohdalla tekoäly oli enemmän läsnä oikeissa paikoissa, tekoäly oli paljon "elävämpi" ja arvaamattomampi variaatio A:n tekoälyyn verrattuna ja siksi myös paljon kiinnostavampi ja viihdyttävämpi. Variaatio B:tä pidettiin siis joko haastavana tai helppona sen mukaan, miten tekoäly sattui asettumaan jokaisen haastateltavan kohdalla.

Myös variaatio C jakoi mielipiteitä samoista syistä kuin variaatio B. Tekoälyn oli entistäkin vaikeampaa olla riittävästi läsnä varsinkin, kun vartijarobotteja oli vain yksi. Luonnollisesti haastateltavat eivät silloin kyenneet havaitsemaan tekoälyn käyttämiä rutiineja.

Haastateltavien 1 ja 5 kohdalla tekoäly kuitenkin pystyi olemaan läsnä ja luomaan jopa jännittävän, painostavan ja haastavan pelikokemuksen. Tällöin haastateltavien mielestä tekoälyä vastaan oli mielenkiintoista pelata. Tekoälyn arvaamattomuus piti pelaajan varpaillaan ja selvästi loi enemmän viihdearvoa. Haastateltavat luonnehtivat variaatio C:n tekoälyä "eläväksi" ja "inhimilliseksi".

Suurin osa haastateltavista piti variaatio A:ta haastavimpana. Haasteen kokemiseen kuitenkin vaikutti perusmekaniikkojen opettelu kyseisen variaation aikana. Myös variaatio B:tä pidettiin haastavana, mutta tekoälyn läsnä olemisen vähyys helpotti monilla variaatiota A:han verrattuna. Variaatio C koettiin helpoimmaksi variaatioksi ainoastaan yhden robottivartijan takia. Poikkeuksellisesti haastateltava 5 koki variaatio C:n olevan haastavin. Hänen kohdallansa tekoäly pystyi olemaan läsnä ja aiheuttamaan jännittäviä ja haastavia tilanteita.

Vaikka variaatio A olikin yleisesti koettu haastavimmaksi, ei sitä koettu kovinkaan viihdyttäväksi verrattuna B- ja C-variaatioihin. Muista poiketen haastateltava 3 piti A variaatiota viihdyttävimpänä, sillä hän pitää ongelmanratkaisupeleistä ja variaatio A hänen mielestään muistutti paljon tällaisia pelejä. Kuten aiemmin mainitsin, B- ja C-variaatiot koettiin mielenkiintoisina ja viihdyttävinä dynaamisen, arvaamattoman ja inhimillisen tekoälyvastustajan takia. Usean haastateltavan mielestä eri variaatioita yhdistelemällä voisi saada aikaan hyvin viihdyttävän ja haastavan pelikokemuksen.

5.3 Pohdintaa

Tulosten perusteella on selvää, että tekoäly vaikutti suuresti haastateltavien kokemaan pelin haastavuuteen. Tekoälyt eivät ainoastaan vaikuttanut haastavuuden tasoon, vaan myös kykenivät haastamaan pelaajan toisistaan poikkeavilla tavoilla. Variaatio A tarjosi perinteisen, mekaanisen sekä yrityksen ja erehdyksen kautta opittavan kokemuksen, jossa haasteena oli löytää variaation eri vaiheissa toimivat toimintatavat. Variaatio B puolestaan haastoi pelaajan reagoimaan tekoälyn tuottamiin emergentteihin tilanteisiin, jollaisia ei variaatio A:ssa tullut vastaan. Myös variaatio C toi pelaajalle uudenlaisen haasteen muihin verrattuna, kun tekoäly vei pelaajalta turvalliset alueet, oli entistäkin arvaamattomampi partiointireitin suhteen ja pyrki aina pysymään pelaajan lähettyvillä.

Toisaalta tekoälyjen heikkoudet tulivat myös hyvin esille tutkimuksen aikana. Variaatio A:n tekoäly koettiin yllätyksettömänä ja puuduttavana haasteena. Variaatio B:n ja C:n tekoäly oli kaksiteräinen miekka: vaikka toimiessaan odotetusti tekoäly tarjosi viihdyttävän ja haastavan pelikokemuksen, sen satunnaisen luonteen takia tekoäly ei kyennyt aina olemaan riittävästi paikalla, jolloin pelaajalla oli helppo tehtävä edetä pelissä. Variaatio C:ssä tekoälyn siirtyminen pelaajan perässä alueelta toiselle oli myös liian hidasta. Pelaaja ehti yleensä helposti ylemmälle alueelle ja hakemaan hissin avainkortin ennen kuin tekoäly oli ehtinyt siirtymään kunnolla samalle alueelle. Tästä syystä toinen samanlainen tekoäly variaatiossa C ei korjaisi ongelmia kokonaan.

Eräs haastateltava sanoi, että “totuus löytyy jostain näiden (variaatioiden) välistä” ja olen samaa mieltä. Variaatioita selvästi vaivasi tekoälyjen yksipuolisuus, jolloin niiden heikkoudet voimistuivat. Aiemmin mainitsemani Alien: Isolation käyttää arvaamattoman Alienin lisäksi vastustajina ihmisiä ja androideja, jotka partioivat Sevastopolin käytävillä. Tällä tavalla peli kykenee täyttämään Alienin poissaolon tuottaman haasteellisen tyhjiön. Peliprototyypin variaatiot hyötyisivät todennäköisesti suuresti eri tekoälyjen yhdistelemisestä, jolloin variaatio B:n ja C:n satunnaisuuden tuottama vähäinen haaste saadaan paikattua variaatio A:n mekaanisella haasteella ja variaatio A:n tylsyys ja yllätyksettömyys voidaan korjata variaatio B:n ja C:n dynaamisuudella.

6 Lopuksi

Tutkielmani perusteella on mielestäni selvää, että satunnaisen liikkumisstrategian omaavalla pelitekoälyllä on paljon potentiaalia hiiviskelypeleissä viihdyttävyyden näkökulmasta. Kuten mainitsin luvussa 2.4.1, satunnainen liikkuminen ja reitinhaku saattaa vaikuttaa pelaajan näkökulmasta tyhmältä, mutta kun tekoälyn suunnittelee hyvin hahmolle sopivaksi voi lopputulos olla myös erittäin uskottava ja viihdyttävä (esim. Alien: Isolation). On selvää, että satunnaisen tekoälyn luonteeseen kuuluu, että se ei pysty ylläpitämään jatkuvaa läsnäoloa, joten pelisuunnittelijan on otettava tämä huomioon täyttämällä tekoälyn puutteellista läsnäoloa muilla keinoin.

Mielestäni peliprototyypin työstäminen oli kiinnostavaa ja erittäin opettavaista. Huomasin helposti, kuinka kuviossa 1 esitetty tekoälymalli toteutui myös omassa prototyypissäni jokaisen tekoälyvariaation kohdalla. Oli myös hauska huomata, kuinka tarkoituksenmukaista pelitekoälyn kehittäminen on verrattuna omaan varsinaiseen työhöni, johon kuuluu neuroverkkojen parissa työskentely. Pelitekoälyä työstäessä oli tärkeämpää saada tekoäly liikkumaan oikeaan paikkaan, vaikka tekoälyskriptin koodin laatu saattoikin kärsiä, kun taas neuroverkkojen parissa työskennellessä on tärkeää tehdä asioita oikein, jotta verkon luokittelutarkkuus olisi mahdollisimman korkea. Tästä myös mainitsin luvussa 2.2.

Haastattelututkimus sujui myös mielestäni erittäin hyvin. Kaikki haastateltavat olivat innostuneita ja kiinnostuneita tutkielmani aiheesta, mikä auttoi entisestään yhteisymmärryksen muodostamista haastateltavien kanssa. Kaikilla oli selvästi hauskaa haastattelun aikana ja sain erittäin hyviä tuloksia. Itselleni tuli hieman yllätyksenä B- ja C-variaatioiden heikko suoriutuminen ja läsnäolo, mutta oli myös yllättävää huomata, kuinka paljon enemmän haastateltavat niistä pitivät verrattuna A variaatioon.

Tutkielman valmistuminen kesti melko pitkään. Tähän vaikutti erittäin paljon omassa elämässäni tapahtuneet jopa rajut muutokset. Tutkielman työstämisen aikana sain mm. kaksi työpaikkaa, joista jälkimmäinen on vakituinen ja josta myös nautin tälläkin hetkellä. Olen kuitenkin ylpeä itsestäni pystyessäni saattamaan tutkielmani loppuun kaikista hidasteista ja vastoinkäymisistä huolimatta.

Lähteet

Aarseth, Espen, Marie Smedstad Solveig ja Lise Sunnanå. 2003. “A Multidimensional Typology of Games”. Teoksessa *DiGRA ’03 - Proceedings of the 2003 DiGRA International Conference: Level Up.* ISBN: ISSN 2342-9666. http://www.digra.org/wp-content/uploads/digital-library/05163.52481.pdf.

Alkaisy, Casey. 2011. “The history and meaning behind the ’Stealth genre’”. Viitattu 6. tammikuuta 2019. http://www.gamasutra.com/blogs/MuhammadAlkaisy/20110610/7764/The_history_and_meaning_behind_the_Stealth_genre.php.

Björk, Staffan, ja Petri Lankoski. 2015. *Game Research Methods.* Pittsburgh, PA: ETC Press.

Buckland, Mat. 2005. *Programming game AI by example.* Plano, TX: Wordware Publishing.

Cui, Xiao, ja Hao Shi. 2011. “A*-based Pathfinding in Modern Computer Games” [kielellä English]. *IJCSNS International Journal of Computer Science and Network Security* 11, numero 1 (tammikuu): 125–130.

DaGraça, Micael. 2017. *Practical Game AI Programming.* Birmingham: Packt Publishing Ltd.

Hamari, Juho, ja Janne Tuunanen. 2014. “Player types: a meta-synthesis” [kielellä English]. *Transactions of the Digital Games Research Association* 1 (2): 29–53. ISSN: 2328-9422. doi:10.26503/todigra.v1i2.13.

Hu, Wenfeng, Qiang Zhang ja Yaqin Mao. 2011. “Component-based hierarchical state machine A reusable and flexible game AI technology”. Teoksessa *2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference,* 2:319–324. Elokuu. doi:10.1109/ITAIC.2011.6030340.

Leonard, Tom. 2003. “Building an AI Sensory System: Examining The Design of Thief: The Dark Project”. Viitattu 30. joulukuuta 2018. https://www.gamasutra.com/view/feature/131297/building_an_ai_sensory_system_.php.

Millington, Ian, ja John Funge. 2009. *Artificial intelligence for games.* Burlington: Elsevier.

Poole, David, Alan Mackworth ja Randy Goebel. 1998. *Computational Intelligence: A Logical Apporoach.* New York: Oxford University Press.

Puusa, Anu, ja Pauli Juuti. 2011. *Menetelmäviidakon raivaajat: perusteita laadullisen tutkimuslähestymistavan valintaan.* Helsinki: JTO.

Salen, Katie, ja Eric Zimmerman. 2004. *Rules of Play.* Cambridge, MA: MIT Press.

Sukhbaatar, Sainbayar, Arthur Szlam, Gabriel Synnaeve, Soumith Chintala ja Rob Fergus. 2015. "MazeBase: A Sandbox for Learning from Games". *ArXiv* abs/1511.07401.

Sutton-Smith, Brian. 2004. *Toys as Culture.* New York: Gardner Press.

Thompson, Tommy. 2016. "The AI of Alien: Isolation". Viitattu 3. tammikuuta 2019. https://www.youtube.com/watch?v=Nt1XmiDwxhY.

Unity Technologies. 2020. "Unity - Scripting API: MonoBehaviour.Update()". Viitattu 8. maaliskuuta 2020. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html.

Walsh, Martin. 2014. "Modeling AI Perception and Awareness in Splinter Cell: Blacklist". Viitattu 30. joulukuuta 2018. https://www.youtube.com/watch?time_continue=1905&v=RFWrKHM0vAg.

Liitteet

Haastattelukehys

Aloitus

Mainittavat asiat:

- Haastattelutilanne nauhoitetaan myöhempää analyysiä varten
- Pelitilanteesta kerätään dataa (yritysten määrä, kulunut aika, videonauhoitus)
- Haastateltava vastaa kysymyksiin täysin anonyymisti
- Tulokset käsitellään siten, ettei haastateltavan henkilöllisyys ole niistä selvitettävissä
- Peliprototyypin esittely

Kysymykset:

- Millainen pelaaja mielestäsi olet?
  - Saavuttaja, tutkija, sosiaalinen, dominoiva vai immersio
- Miten suhtaudut haasteeseen peleissä?
- Oletko aiemmin pelannut hiiviskelypelejä?
- Ovatko pelaamasi pelit mielestäsi haastavia?
- Mikä mielestäsi tekee niistä haastavan/helpoin?
- Mitä mieltä olet pelaamiesi genren pelien tekoälystä?

Variaatio A

- Oliko variaatio haastava? (asteikolla 1-10)
  - Mikä teki variaatiosta haastavan/helpon?
- Mitä mieltä olit tekoälyn käyttäytymisestä?

Variaatio B

- Oliko variaatio haastava? (asteikolla 1-10)

    - Mikä teki variaatiosta haastavan/helpon?
- Mitä mieltä olit tekoälyn käyttäytymisestä?
- Oliko variaatio edellistä haastavampi?
    - Mikä teki variaatiosta edellistä haastavamman/helpomman?

Variaatio C

- Oliko variaatio haastava? (asteikolla 1-10)
    - Mikä teki variaatiosta haastavan/helpon?
- Mitä mieltä olit tekoälyn käyttäytymisestä?
- Oliko variaatio edellistä haastavampi?
    - Mikä teki variaatiosta edellistä haastavamman/helpomman?

Lopetus

- Mikä oli mielestäsi haastavin variaatio?
    - Mikä teki variaatiosta haastavan?
- Mikä oli mielestäsi helpoin variaatio?
    - Mikä teki variaatiosta helpon?
- Oliko tekoälyllä mielestäsi suuri vaikutus variaatioiden haastavuuteen?
- Mikä variaatioista oli mielestäsi viihdyttävin?