Aku Tanttu

TEKOÄLY PELIOHJELMOINNISSA

Case: Stratogear

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Marraskuu 2013

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Opinnäytetyön päivämäärä** 3.12.2013
Tekijä(t) Aku Tanttu	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittelyn ko.
Nimeke Tekoäly peliohjelmoinnissa	

Tiivistelmä

Tekoäly tuli peleihin jo 1970-luvulla ja on kulkenut pitkän matkan sieltä nykypäivään. Nykyään se on hyvin oleellinen osa tietokone- ja konsolipelaamista. Tekoälyä on kaikkialla peleissä, ja se kehittyy koko ajan

Teimme tässä projektissa neljän hengen ryhmässä steampunk-tyylisen tietokonepelin. Ryhmämme koostui suunnittelijasta, mallintajasta ja kahdesta koodarista, joista minä olin toinen koodari.

Opinnäytetyössäni kävin läpi, millaista tekoäly on peleissä, miten sitä käytetään, ja mikä ylipäätään luokitellaan tekoälyksi peleissä. Lisäksi kerroin hiukan tekoälyn historiasta sekä peleissä, että ylipäätään. Kerroin myös opinnäytetyössä esimerkkejä erilaisista tekoälyistä

Käytännöntoteutuksena tosiaan teimme tietokonepelin, josta minä vastasin erityisesti tekoälystä. Pelissä oli tarkoitus hallita ilmalaivoja, joilla pelaaja pystyi käymään kauppaa kaupunkien välillä, tai jopa ryöstää tai ampua muita laivoja.

Pelin voittaisi se, kuka lopulta hallitsee pelimaailman ilmatilaa, joko kaupankäynnillä tai tuhoamalla. Opinnäytetyön puitteissa teimme pelistä vain demon, mutta lopullinen tavoite olisi ollut monin peli useamman pelaajan ja/tai tekoälyn kesken

Tässä opinnäytetyössä käsittelen tekoälyä lähinnä siltä kantilta, miten sitä käytetään peliohjelmoinnissa ja miten pienillä asioilla saadaan pelin tekoäly näyttämään paremmalta, mitä se oikeasti on. En käsittele tässä kovinkaan paljon itse tekoälyn koodamista, vaan sen keinoja.

Asiasanat (avainsanat)

Tekoäly, heuristiikka, äärellinen automaatti, päätöspuu, sumea logiikka

Sivumäärä	**Kieli**	**URN**
29	Suomi	

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi	**Opinnäytetyön toimeksiantaja**
Jukka Selin	

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis** 3 December 2013
Author(s) Aku Tanttu	**Degree programme and option** Business Information Technology

Name of the bachelor's thesis

Artificial intelligence in game development

Abstract

Artificial intelligence (AI) arrived to the game industry in the 1970s and it has come a long way since then to the present day. Nowadays it is a very essential part of gaming. Artificial intelligence can be found everywhere in games and it is developing all the time

This thesis went through, what kind of different artificial intelligence there is in games. And how this was used in games? I also told about the history of artificial intelligence and the difference between game AI and academic AI

In this project, our group of four students made a steam-punk computer game where the player controls airships. With airships player could make money by trading goods between cities, or even rob or shoot the other ships. In the end winner would be the one who control the sky. This study only involved a demo, but the game could also be developed further.

Subject headings, (keywords)

Artificial intelligence, heuristic, finite state machine, decision tree, fuzzy logic

Pages	**Language**	**URN**
29	Finnish	

Remarks, notes on appendices

Tutor	**Bachelor's thesis assigned by**
Jukka Selin	Mikkeli University of Applied Sciences

SISÄLTÖ

1 JOHDANTO

Opinnäytetyöni aihe on siis tekoäly peliohjelmoinnissa. Kiinnostuin tekoälystä noin puoli vuotta ennen, kun aloin tehdä tätä opparia. Siitä lähtien tekoäly on alkanut kiinnostamaan koko ajan vain enemmän. Tekoälyssä riittää tarpeeksi haastetta ja siinä pääsee välillä oikeasti miettimään, miten tämä voisi onnistua. Siksi sen tutkiminen ja ohjelmoiminen onkin varmaan minusta niin hauskaa, joten aihe oli minulle selvillä.

Tulin myös siihen tulokseen lopulta, että haluaisin demota sitä jossain pelissä. Juttelin ideasta parin kaverin kanssa ja kävi ilmi, että hekin olivat suunnitelleet pelin tekemistä oppariksi. Näin syntyi Stratogear- peli. Tavoitteenamme tähän opinnäyte työhön oli kuitenkin vain tehdä tästä pelistä demo, sillä koko pelin valmiiksi asti vieminen olisi paljon suurempi urakka, mitä opinnäytetyön rajoissa ehdittäisiin tehdä. Pyrimme silti saamaan mielenkiintoisen demon aikaiseksi, josta saisi jonkinlaista ideaa, minkälainen peli se voisi lopulta olla. Jos demo vaikuttaa lupaavalta niin todennäköisesti viemme sen myös loppuun asti myöhemmin.

Oma tutkimusongelmani projektissa oli miten tekoälyä käytetään peliohjelmoinnissa, ja mikä ylipäätään luokitellaan tekoälyksi peleissä? Lopuksi aion vielä soveltaa sitä käytännössä meidän projektiimme. Pyrin siis opinnäytetyössäni sisällyttämään peliimme tekoälyä, joka auttaisi pelaajaa helpottamalla pelaajan työtä sekä tekoälyn ohjaaman pelaajan, jota vastaan pelaaja voisi pelata.

Alkuun kerron hiukan tekoälyn historiasta, sen synnystä ja siitä milloin ja missä tekoäly sai alkunsa. Olen myös sitä mieltä, että Turingin testi liittyy oleellisesti tekoälyyn joten kerron myös siitä hiukan. Lisäksi kerron hiukan yleistä asiaa tekoälystä peleissä, millaista tekoälyä käytetään, minkälaisissa tilanteissa sitä käytetään, sekä minkälaisia tekoälyjä on käytetty peleissä ennen ja milloin niitä käytettiin. Teoriaosuuden jälkeen kerron sitten enemmän meidän pelistämme. Siitä mitä siihen tein, minkälaista tekoälyä käytin ja mihin. Ylipäätään hiukan minkälainen peli se on.

2 TEKOÄLYN HISTORIAA

Tekoälyä on tutkittu jo vuosikymmeniä ja nykyään sitä löytyy melkein kaikkialta laidasta laitaan. Kodinkoneissa, tietokonepeleissä, autoissa, jossain tekoäly vain helpottaa tai nopeuttaa käyttäjän toimia, kun taas jossain tekoäly ajattelee ja toimii täysin itsenäisesti.

Kaikki alkoi 40- ja 50-luvulla kun useat tutkijat eri aloilta alkoi keskustella mahdollisuudesta luoda keinotekoiset aivot. Kun 50-luvun puolessavälissä kehiteltiin numeeriset tietokoneet, tutkijat keksivät, että jos kone pystyy käsittelemään numeroita, niin se pystyy myös käsittelemään kuvioita. Kuvioiden tunnistus ja käsittely voisi hyvin olla ihmisen ajattelun perusolemusta.

Vuonna 1956 pidettiin tekoälyn kannalta merkittävä Dartmouthin konferenssi. Konferenssiin osallistui useita tutkijoita, jotka olivat luoneet tärkeitä ohjelmia tekoälyn tutkinnan kannalta. Kyseisessä konferenssissa tekoälyä alettiin kutsua ensimmäisen kerran tekoälyksi. Lisäksi siellä koettiin ensimmäiset onnistumiset tekoälyn saralla. Siellä muun muassa Newell ja Simon loivat The Logic Theorist –ohjelman, jota monet pitävätkin ensimmäisenä oikeana tekoäly sovelluksena. Sovelluksen ideana oli esittää ongelmat puumallina ja ratkoa niitä valitsemalla aina haara joka todennäköisimmin johtaisi oikeaan ratkaisuun. Darthmouthin konferenssia pidetäänkin yleisesti tekoälyn syntynä

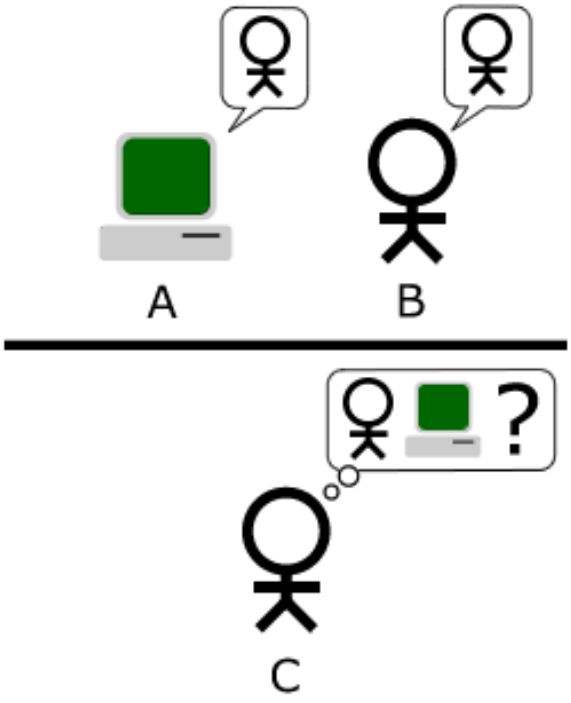
Seuraavan seitsemän aikana tekoälytutkimusten suosio kasvoi. Tekoälytutkimuksen pääpaikoiksi nousi kaksi suurta nimeä MIT ja Carnegie Mellon. Tekoälyn kehitys seurasi lähinnä kahta linjaa; Sitä kehitettiin systeemiksi, joka osaisi tehokkaasti ratkaista ongelmia. Toiseksi kehiteltiin systeemiä, jolla olisi kyky oppia ja kehittyä itsenäisesti.

Vuonna 1957 ensimmäinen versio uudesta sovelluksesta *General Problem Solver*(GPS) testattiin. Ohjelman takana oli sama kaksikko, joka oli luonut the Logic Theorist -sovelluksen. GPS pystyi ratkaisemaan entistä paremmin perusongelmia. Muutama vuosi GPS:n jälkeen IBM kasasi tiimin tutkimaan tekoälyä. Vuonna 1958 John McCarthy julkisti seuraavan läpimurron tekoälyn historiassa. Hän oli kehitellyt LISP- ohjelmointikielen, joka on vielä tänä päivänäkin käytössä. LISP tulee sanoista LISt Processing ja siitä tuli pian hyvin suosittu ohjelmointikieli tekoälytutkijoiden

keskuudessa. Vuonna 1963 USA:n hallitus antoi MIT:lle 2,2 miljoonan dollarin rahoituksen tekoälyn tutkimiseen. Rahoitus tuli asevoimien osaston *advanced research projects agency*(ARPA) –tutkimusorganisaatiolta. Rahoitus annettiin lähinnä siksi, että Yhdysvallat pysyisi Neuvostoliittoa edellä teknologiassa. Tämä kuitenkin mahdollisti tekoälytutkimuksen kiihtyvän kehityksen. (Oracle Thinkquest)

2.1 Turingin testi

Alun perin vuonna 1951 Alan Turing suunnitteli testin nimeltä ”*The Imitation Game”* Ensimmäinen versio tosin ei sisältänyt ollenkaan tekoälyä. Kuvittele päässäsi kolme huonetta. Huoneet ovat yhteydessä toisiinsa vain tietokoneiden kautta. Ensimmäisessä huoneessa istuu mies, toisessa huoneessa nainen ja kolmannessa istuu henkilö, jota kutsutaan nyt tuomariksi(judge). Tuomarin tehtävänä on selvittää, kumpi tietokoneella keskustelevista henkilöistä on mies. Mies yrittää auttaa tuomaria todistamaan itsensä mieheksi. Naisen tehtävä tässä on kuitenkin yrittää huijata tuomaria luulemaan toisin.



KUVA 1. Turingin testin perusidea (Wikimedia.org 2013)

Mitä tekemistä tällä on tekoälyn kanssa? No, Alan Turing muokkasi tätä ideaa myöhemmin. Hän teki siitä version, jossa miehen ja naisen sijaan testissä olikin ihminen, kumpaa tahansa sukupuolta, ja häntä vastassa oli tietokone.

Tuomarin tehtävänä oli nyt valita, kumpi näistä testattavista oli ihminen. Jos tuomari olisi yhtä varmasti valinnut tietokoneen kuin ihmisen, niin tietokone olisi kelvollinen simulaatio ihmisen mielestä. Nykyään kuitenkin testissä on vain yksi osallistuja ja tuomarin tehtävänä on päättää onko hän ihminen vai tietokone. (Bradley 2002.)

2.2 Tekoäly peleihin

Shakkia on aina pidetty älykkäänä pelinä, joten shakki oli ensimmäisiä pelejä missä tekoälyä kokeiltiin. Shakkia käytettiinkin paljon tekoälyn opiskelussa ja esittelyssä. Monet asiantuntijat pitävätkin suurena virstanpylväänä tekoälylle kun vuonna 1997 kun Deep Blue tekoäly ohjelma voitti shakin maailman mestarin Gary Kasparovin.

Tietokone- ja konsolipeleihin tekoäly tuli mukaan hyvin varhaisessa vaiheessa, jopa 1970-luvulla kun Atari julkaisi pelin nimeltä *Computer Space*. Peli itsestään oli hyvin yksinkertainen, mutta siinä oli tekoälyn liikuttamia aluksia, jotka pelasivat pelaajaa vastaan. Siitä lähtien pelitekoäly on kehittynyt valtavasti. Nykyään tekoäly on välttämätöntä ja sillä on tärkeä osa pelin pelattavuudessa ja laadussa. (Xu 2008)

AI game programmers guild jakaa pelien tekoäly historian ajanjaksoihin. Tällä hetkellä he ovat luokitelleet tekoälyn kuuteen ajanjaksoon. Nämä ajanjaksot kertovat minkälaista tekoälyä peleissä on käytetty ja milloin.

Basic patterned

Ensimmäisenä mainittakoon *Basic patterned*, jota käytettiin aikaisemmissa arcade peleissä, sekä vanhoissa tietokone peleissä. Siinä viholliset käyttävät lähinnä ennalta määritettyjää kaavoja liikkumisessa. Nämä kaavat ovat yleisesti lyhyitä, yksinkertaisia ja ne toistavat itseään. Lisäksi kaavat eivät ollenkaan tai hyvin vähän huomioi pelaajan liikkeitä. Tällaista tekoälyä käytettiin 70-luvusta 80 –luvun puoleenväliin muun muassa vanhoissa arcade peleissä kuten Space Invaders ja Donkey Kong.

Simple Hard-coded Rules

Toiseksi kilta sanoo *Simple Hard-coded Rules.* Tässä yksittäiset olennot käyttävät koodattuja algoritmeja määrittääkseen liikkumiseen tai toiminnan. Nämä saattavat ottaa jopa huomioon pelaajan sijainnin tai liikkeet, mutta yleensä vain hyvin rajoitetusti. Tällaista tekoälyä käytettiin 70–luvulta 80-luvun loppupuolelle. Esimerkki pelinä kilta sanoo Pac-Manin, jossa jokaisella kummituksella on omat liikkumisohjeensa labyrintissä liikkumiseen.

Advanced Patterned

Kolmantena tulee *Avanced Patterned*, jota käytettiin lähinnä 80-luvulla. Siinä vihollisia liikutetaan ennalta koodatuilla säännöillä, jotka ovat hiukan monimutkaisempia kuin yksinkertaiset koodit, mutta eivät vieläkään huomioi pelaajan kaikkia toimintoja. Pelejä, missä tällaista tekoälyä käytettiin, on esimerkiksi Zelda ja Super Mario Brothers.

Tactical Reaction

Seuraavana tulee *Tactical Reaction*. Tässä tekoäly sisältää lyhytaikaisia reaktioita lähellä tapahtuviin vaikutteisiin. Moni taistelupeli, kuten Tekken tai Street Fighter, käyttää tätä torjumaan pelaajan hyökkäykset ja hyökkäämään itse kun näkee siihen mahdollisuuden. Tällainen tekoälyä käytettiin 80-luvun puolestavälistä alkaen ja perustasolla tätä ideaa käytetään edelleen tänä päivänäkin.

Tactical Reasoning

Viidentenä he sanovat *Tactical Reasoning*, jossa yksittäiset tekoälyn ohjaamat olennot tekevät jo edistyneempiä päätöksiä, jotka ottavat huomioon ympäristön. Tämä eroaa Tactical Reactionista siten, että tässä toiminnot ovat yleensä pitempiaikaisia ja saattaa sisältää useitakin toimenpiteitä. Tämän tyylinen käyttäytyminen sisältää esimerkiksi järkevää reitinhakua, sekä suojaan menemistä. Tätä alettiin käyttää 90-luvun puolivälissä ja tätäkin käytetään vielä nykypäivänä.

Strategic

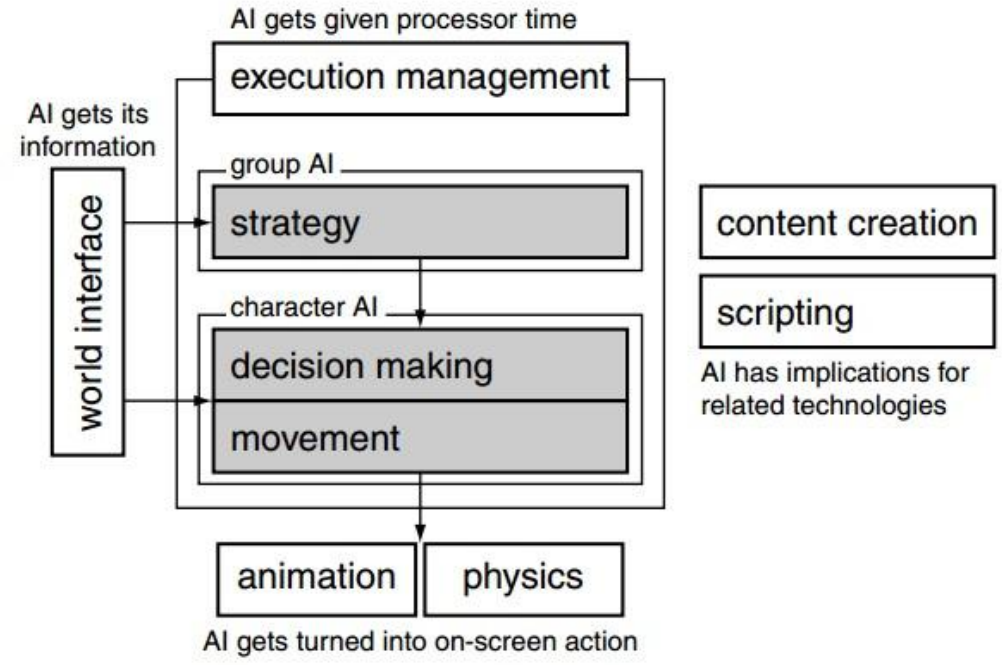
Viimeisenä tulee *Strategic*, joka kehiteltiin 90-luvun loppupuolella. Tässä tekoälyn ohjaamat yksittäiset olennot ovat enemmän tietoisia toisistaan. Moni tällaista tekoäly ajattelua hyödyntävä peli käyttää *"Overseer"* tekoälyä kordinoimaan useita yksittäisiä olentoja, jotta suurempi ryhmä osaisi toimia paremmin keskenään. Esimerkkejä tämänkaltaisista tekoälyistä löytyy muun muassa Halo –pelisarjasta sekä Starcraft 2:sta. (AI game programmerr guild 2011)

3 TEKOÄLY OSANA PELIÄ

Nykyään lähes kaikissa peleissä on tekoälyä jossain muodossa. Oli se sitten nopeimman reitin etsimistä, yksinkertaista asioiden ohjailua tai tekoälyn ohjaama pelaaja. Millingtonin mukaan suurimassa osassa moderneissa peleissä tekoälyllä on kolme perustarvetta. Kyky liikuttaa hahmoa, kyky tehdä päätös siitä mihin liikkua ja kyky ajatella taktisesti tai strategisesti.

Kaikki pelit eivät kuitenkaan tarvitse näitä kaikkia. Esimerkiksi lautapelit, kuten shakki tai risk käyttävät vain strategista ajattelua. Siinä yksittäiset hahmot tai nappulat eivät tee itse päätöksiä siitä mihin ne liikkuvat.

Toisaalta taas monissa peleissä ei ole strategista ajattelua ollenkaan. Esimerkiksi monissa tasohyppely peleissä viholliset toimivat itsenäisesti reagoiden vain pelaajaan ja sen liikkeisiin. Näissä peleissä viholliset eivät toimi yhtenäisesti, joka on usein helpompi pelaajalle.



KUVA 2. Tekoälyn tyyli peliohjelmoinnissa

Liikkumisella Millington tarkoittaa algoritmeja, jotka kääntävät päätökset jonkinlaiseksi liikkeeksi. Kun joissain peleissä vihollinen, jolla ei ole pidemmän kantaman asetta, haluaa hyökätä pelaajaan, niin sen pitää ensin liikkua pelaajan luo. Kun vihollinen on tarpeeksi lähellä, niin sitten se voi vasta hyökätä. Päätös hyökkäämisestä johtaa sarjaan liikkumista hoitavia algoritmeja, jotka ohjaavat vihollisen pelaajan luo ja vasta sitten voidaan hyökkäys animaatio ja vähentää pelaajan elämäpisteitä.

Liikkuminen voi toki olla monipuolisempaakin kuin pelkästään pelaajan luo hakeutuminen. Hahmon täytyy ehkä väistellä esteitä tai jopa kulkea useamman huoneen lävitse. Esimerkiksi joissain *Splinter Cell*-pelin tasoissa vartija pelaajan huomattuaan tekee hälytyksen. Tässä tilanteessa vartijan pitää löytää nopeasti tiensä lähimmälle hälytysnappulalle, joka voi olla pidemmänkin matkan päässä ja matkalla pitäisi väistellä esteitä ja kulkea usean käytävän halki.

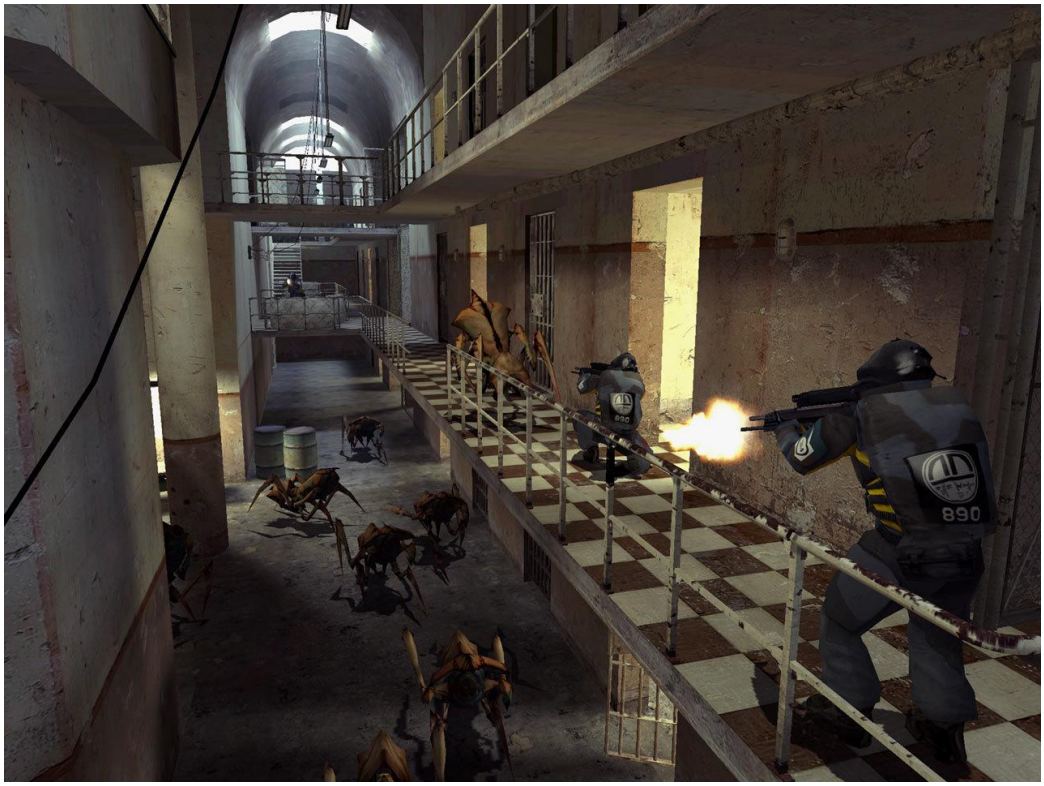
Moni toiminta suoritetaan ihan vain käyttämällä animaatiota. Esimerkiksi pelissä *The Sims* kun sim istuu pöydän ääressä ja hänellä on ruokaa edessä. Jos sim haluaa nytten syödä, niin syömistä kuvaava animaatio laitetaan vain käyntiin. Kun hahmo on päättänyt, että se haluaa syödä, niin tekoälyä ei siihen enää tarvita. Jos kuitenkin sama hahmo olisi oven takana, niin liikkumista ohjaavan tekoälyn pitäisi ensin ohjata hahmo pöydän ääreen.

Päätöksenteossa hahmo taas miettii, mitä se tekisi seuraavaksi. Yleensä jokaisella peliobjektilla on useita eri vaihtoehtoja mitä se voisi tehdä. Objekti voisi esimerkiksi hyökätä, piiloutua tai seisoa paikallaan. Tekoälyn päätettäväksi jääkin miettiä mikä näistä olisi paras ratkaisu. Objekteilla voi yksinkertaisimmillaan olla hyvin simppeli käyttäytymismalli.

Esimerkiksi joissain Zelda peleissä farmin eläimet seisoo paikallaan, kunnes pelaaja tulee liian lähelle, jolloin eläin liikkuu hiukan kauemmaksi. Toisessa ääripäässä taas viholliset *Half-Life 2* -pelissä käyttävät hyvinkin monimutkaista päätöksentekoa. Pelissä viholliset käyttävät useita erilaisia strategioita päästäkseen pelaajan lähelle, Nämä strategiat sisältävät usein useita pienempiä toimintoja, kuten kranaatin heittäminen tai suojatulen antaminen, saavuttaakseen tavoitteensa.

Jotkut päätökset tarvitsevat liikkumista ohjaavaa tekoälyä toteuttamaan päätöksen. Lähitaistelu hyökkäyksen tehdäkseen hahmon pitää liikkua lähelle kohdetta. Toiset päätökset toteutetaan helposti animaatiolla, kuten vaikka sim syömässä. Jotkut päätökset vain yksinkertaisesti muuttamalla pelin tai hahmon tilaa ilman minkäänlaista näkyvää muutosta. Esimerkiksi vaikka kun *Sid Meier's Civilisation III* -pelissä tekoälyn valtio valitsee uuden teknologian tutkittavakseen. Se vain tapahtuu ilman mitään näkyvää ilmoitusta.

Jo liikkumista ohjaavalla ja päätöksenteko tekoälyillä voi päästä pitkälle. Moni toimintapainotteinen kolmiulotteinen peli käyttääkin vain näitä kahta osaa. Mutta ohjatakseen kokonaista tiimiä tai joukkuetta, niin tarvitset strategista tekoälyä. Strategisella tekoälyllä Millington tarkoittaa tekoälyä, joka ohjaa yhden hahmon sijaan kokonaista joukkoa. Jokaisella yksittäisellä hahmolla usein kuitenkin on edelleen oma päätöksenteko ja liikkumista ohjaavat algoritmit, mutta kokonaisuudessaan ryhmän toimii keskenään ja yksittäiset päätökset ajaa ryhmän etua. Esimerkiksi alkuperäisessä *Half-Life* -pelissä viholliset toimivat tiiminä piirittääkseen ja tuhotakseen pelaajan. Yksi vihollisista usein yritti päästä pelaajan taakse, jotta pelaajalla olisi hankalampaa suojautua, kun molemmista suunnista ammutaan.



KUVA 3. Half-life pelissä tekoäly osasi ohjata hahmoja tiimissä (Half-life)

Tekoäly algoritmit eivät kuitenkaan yksinään riitä. Kehitelläkseen tekoäly peliin tarvitaan paljon muutakin. Liikkuminen pitää luoda peliin toiminnoksi, joko animaatiolla fysiikan mallinnuksella. Lisäksi tekoäly tarvitsee tietoa pelistä, että se voisi tehdä järkeviä päätöksiä. Englanniksi tätä kutsutaan nimityksellä *”perception”*. Tällä määritellään mitä hahmo tietää.

Käytännössä se on kuitenkin paljon enemmän kuin se mitä hahmo kuulee tai näkee. Näiden lisäksi pitää ottaa huomioon kaikki tekoälyn ja pelimaailman välillä olevat rajapinnat, mitä kaikkea hahmo ylipäätään voi tehdä, onko sillä jotain esteitä. Millington nimeääkin tämän osa-alueen tekoälyn ohjelmoinnissa eniten aikaa vieväksi. (Millington 2006.)

3.1 Tekoälyn tyyli peleissä

Peliohjelmoinnissa käytetään paljon pieniä kikkoja nopeuttaakseen peliä tai luodakseen hienoja efektejä. Pelien tekoäly ei juuri tästä eroa. Suurin ero pelitekoäly koodaajien ja tekoäly tutkijoiden välillä on se, mitä he luokittelevat tekoälyksi. Millingtonin kokemuksen mukaan tekoäly pelissä sisältää yhtä paljon *hackingia*, heuristiikkaa ja algoritmeja.

Hacking, eli ne pienet illuusiot, jotka saavat pelin näyttämään järkevämmältä kuin se onkaan. *Hackingiksi* voidaan luokitella monia erilaisia asioita, joista kerron tässä muutaman. Se näyttää kissalta ja tuoksuu kissalta; se on luultavasti kissa. Tällaista ajattelutapaa sanotaan behaviorismiksi. Tämäntapainen ohjelmointi on juuri sopivaa pelitekoälyyn.

Yleensä ohjelmoija ei tarvitse hahmoa, joka ajattelee ja toimii kuin ihminen, vaan ohjelmoija tarvitsee hahmon, joka tekee jonkun tietyn asian. Hyvä tekoäly peleissä toimiikin tällä lailla. Tämä tarkoittaa sitä, että mikä voidaan lukea pelin tekoälyksi, ei välttämättä luokitella tekoälyn tekniikaksi. Esimerkiksi satunnaisluvun luomista ei luokitella tekoälytekniikaksi, mutta sitä voi käyttää monissa paikoissa peleissä luomaan illuusion tekoälystä.

Toinen hyvä esimerkki luovasta tekoäly suunnittelusta löytyy *The Sims* -pelistä. Vaikka pinnan alla onkin paljon puhdasta tekoäly koodia, niin suuri osa hahmojen käyttäytymisestä tuodaan esille animaatioilla. *Star Wars: Episode 1 Racer* -pelissä taas hahmot, jotka ovat vihaisia, tönäisevät muita hahmoja.

Näistä kumpikaan ei tarvitse kovinkaan suurta tekoälyä taakseen. Ne tarvitsevat vain pienen pätkän koodia, joka suorittaa animaation oikeassa kohtaa. Pelien tekoälyohjelmoinnissa pienet asiat, jotka luovat illuusion älykkyydestä, saavat pelin näyttämään heti paljon paremmalta.

KUVA 4. The Sims pelissä pienet animaatiot tuovat tekoälyn tuntua (The Sims)

Heuristiikka; arvioitu ratkaisu, joka toimii useimmissa tapauksissa, mutta mitä luultavimmin ei kaikissa. Ihmiset käyttävät heuristiikkaa jatkuvasti. Me emme mieti kaikkia mahdollisia ideoita toimiessamme, vaan luotamme perus periaatteisiin, mitkä olemme, joko huomanneet toimivaksi aikaisemmin tai oppineet jostain. Se voi yksinkertaisesti olla vaikka ”Ei kannata ottaa karkkia tuntemattomalta hiipparilta”.

Peliohjelmoinnissa on välillä tehtävä valintoja koskien vaikkapa päätöksentekoa tai liikkumista, valintoja nopeuden ja tarkkuuden välillä. Kun tarkkuus ei ole niin tärkeää ja halutaan saada tulos nopeasti tai vähällä vaivalla, niin hankala algoritmi korvataan heuristiikalla.

Tekoälyohjelmoinnissa voi käyttää hyvinkin erilaista heuristiikkaa ongelmissa, joissa ei tarvita tarkkaa algoritmia. Monissa RTS (*Real-time strategy*) tyylisissä peleissä heuristiikalla liikutetaan hahmoja, joilla on pitkän kantaman aseita, hiukkasen eteenpäin, jos vihollinen on juuri ja juuri tavoittamattomissa. Suurimmassa osassa tapauksista tämä onkin paras vaihtoehto, mutta jos sillä pienellä liikkumisella hahmo liikkuu vihollisen suuren tykin kantamalle, niin se ei ollutkaan välttämättä enää niin hyvä idea.

Monissa strategia peleissä eri joukoille tai nappuloille on annettu numeerinen arvo kuvaamaan sitä kuinka hyviä ne ovat. Tämäkin on heuristiikkaa ja se korvaa yhdellä ainoalla numerolla monimutkaisen laskutoimituksen siitä mitä kaikkea kyseinen nappula tai joukko voi tehdä. Ohjelmoija voi määrittää numeron helposti ennakkoon ja tekoäly vain lisää numeron omiin päätöksenteko algoritmeihinsa ja löytää parhaimman vaihtoehdon eri joukoista vertaamalla numeroa ja hintaa.

KUVA 5. Tekoälyn hahmot tulivat liian lähelle torneja (wonderhowto.com)

Peliohjelmoinnissa käytetään monenlaisia heuristiikoita, moniin eri tilanteisiin. Alla on esitelty yleisimpiä heuristiikoita mitä pelien tekoälyssä tulee vastaan.

Harvinaisin

Valitsee aina harvinaisimman mahdollisuuden. Esimerkiksi, jos hahmolla on edessä 4 vihollista ja yksi niistä käyttää kilpeä. Jos kilvellä varustettu vihollinen laskee suojauksensa alas, joka kolmas vuoro. Niin tekoäly päättäisi tässä tilanteessa päättää hyökätä joka kolmas vuoro siihen, koska tilaisuus hyökätä siihen on harvinaisin tapahtuma.

Vaikein ensin

Vaikeimman asian tekeminen vaikuttaa monesti muihin asioihin. On parempi tehdä se ensin kuin tehdä helpot asiat ensin ja sitten huomata, että vaikeinta ei enää voikaan tehdä. Esimerkiksi jos armeijan pitäisi jakaa joukko sotilaita kahteen tasavahvaan joukkoon ja armeijalla on 9 talonmiestä ja 1 ritari. Ritari olisi vaikein sijoittaa, joten se pitäisi tehdä ensimmäisenä. Jos talonmiehet jaetaan kahteen ensin, niin joukoista ei saa mitenkään tasavahvoja

Lupaavin ensin

Jos tekoälyllä on valittavana useasta vaihtoehdosta, niin usein on mahdollista tehdä jokaiselle vaihtoehdolle nopea arviointi ja jokaiselle numero. Vaikka arvio olisikin epätarkka niin kokeilemalla eri vaihtoehtoja parhaiten arvioidusta heikoimpaan, tuottaa paljon paremman tuloksen, kuin täysin satunnaisesti kokeiltu vaihtoehto.

Heuristiikalla ja Hackingilla pääsee jo pitkälle, mutta jos haluaa tarkan tuloksen. niin täytyy ottaa mukaan viimeinen kolmannes, eli algoritmit, jotka mahdollistavat monimutkaisemmat käyttäytymiset. Algoritmit ovat yleensä tarkempia, mutta siksi myös yleensä ovat raskaampia. Algoritmeja käytetään myös paljon peliohjelmoinnissa perusongelmiin, jotka esiintyvät usein, kuten reitinhaku (*Pathfinding*) tai Päätöksen teko(*decision making*). Joten ei aina välttämättä kannata keksiä pyörää uudestaan, vaan voi ihan hyvin käyttää näitä yleisiä algoritmeja. (Millington 2006)

3.2 Tekoälyn rajoitukset

Suurin rajoitus pelien tekoälylle on itse laite millä peliä pelataan. Pelin tekoäly yleensä ei voi miettiä monta päivää seuraava askelta, eikä se saa varata koko muistiakaan omaan käyttöön. Suurin syy, miksei kaikkia tekoälyn tekniikoita käytetä pelien tekoälyssä, on juuri se, että ne tarvitsisivat paljon muistia tai aikaa. Ohjelmoijat joutuvatkin suunnittelemaan pelien tekoälyä näitä asioita silmällä pitäen.

Prosessorin nopeus on näistä syistä itsestään selvin. Aikaisemmin grafiikka vei suuren osan tästä, mutta nykyään suuremmissa määrin grafiikkaa ja animaatioita piirretään nykyään grafiikalle varatuilla laitteistolla. Tämä on luonnollisesti mahdollistanut peleissä muiden osa-alueiden käyttävän prosessoria enemmän. Tämä taas mahdollistaa peleissä paljon kehittyneemmän tekoälyn käyttöä. Suurin osa tekoäly algoritmeista ei tarvitse kovinkaan suuria määriä muistia, mutta muistin määrä ei ole ainoa rajoitus muistin käytössä. Tärkeää on myös miten nopeasti muistiin pääsee käsiksi. Prosessori odottaa ihan turhaan, jos se joutuu odottamaan jatkuvasti tietoa muistista.

Tietokoneet itsessään ovat kaikkein vahvimpia pelilaitteita, sekä samalla kaikkein heikoimpia. Siinä missä konsolit ovat aina samanlaisia, niin tietokoneita on monenlaisia ja tehot vaihtelee todella paljon. Tietokoneelle peliä tehdessä, pitää ottaa huomioon niin satunnaisten pelaajien rajoitetut koneet, kuin kovimpien pelaajien viimeisimmän päälle päivitetyt koneet. Grafiikka on näissä helppo ottaa huomioon, vain säätämällä tarkkuutta ja varjoja heikompaan tai tarkempaan suuntaan, mutta tekoälyssä se ei toimikaan niin.

Jos tekoälyä lähtisi skaalaamaan samalla tavalla, niin peli olisikin helpompi heikommilla koneilla ja tämä ei välttämättä ole kovin suotavaa. Tai vastaavasti, jos tekoäly yrittää tehdä saman työn heikommalla koneella, minkä se voi tehdä paremmalla, niin se vie enemmän aikaa, joka taas saattaa hidastaa peliä.

Yleisin ratkaisu tähän onkin se, että tehdään tekoäly heikompien koneiden ehdoilla, jolloin tekoäly ei skaalaudu ollenkaan paremmille koneille. Monissa peleissä kuitenkin tekoälyä skaalataan, mutta hiukan eri tavalla. Näissä peleissä yksinkertaisesti lisätään hahmoja, jotka ei varsinaisesti vaikuta pelaamiseen, kuten linnut lentämässä taivaalla tai kadulla kävelevät ihmiset. (Millington 2006.)

3.3 Tekoäly pelaajan tukena

Yleensä peliobjektien halutaan liikkuvan sujuvasti ja luonnollisesti peleissä. Ilman minkäänlaista tekoälyä, pelaajan pitäisi ohjailla todella monia asioita yhtaikaa. Mitä esimerkiksi hahmo tekee, jos se juoksee seinää päin. Jatkaako se juoksemista paikallaan, pysähtyykö se, vai törmääkö hahmo seinään ja kaatuu. Ilman minkäänlaista ohjelmointia hahmo varmaan vain juoksisi paikallaan seinää päin, mutta tämähän ei vaikuta kovin luonnolliselta.

Kyllähän tämänlainenkin toiminta riittää useisiin peleihin, mutta jos pelaajalla on hallittavissa useita hahmoja yhden sijaan, niin hahmojen olisi hyvä osata toimia itse perustilanteissa. Hahmo voisi tosiaan pysähtyä juuri ennen seinää ja hahmon pitäisi kääntyä ennen kuin se jatkaisi kulkuaan. Tai yleensä vielä parempi olisi jos hahmo osaisi itse väistää ja kiertää esteet kuten seinät.

Monesti peleissä onkin tekoäly auttamassa pelaajaa pikkuaskareissa. Esimerkiksi monissa RTS peleissä, pelaajan ohjaamilla hahmoilla on jonkinlainen reitinetsintä (*Path-finding*) tekoäly tukena. Jos pelaaja laittaa armeijansa kulkemaan vuoren toiselle puolelle, niin armeija osaa tällöin kiertää sen, eikä välttämättä yritä suoraan läpi.

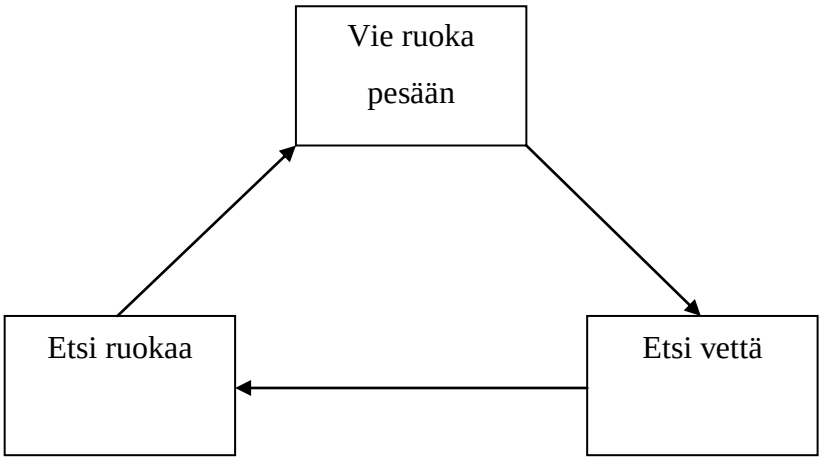
3.4 Tekoäly pelaajana

Tekoäly pelaajana on mahdollistanut muun muassa yleisesti monin peleiksi miellettyjen pelien pelaamisen yksin. Jotta tekoäly osaisi toimia pelaajan sijasta, tekoäly pitää ottaa kaikki asiat huomioon ja valita paras vaihtoehto. Koodatessakin pitää siis miettiä kaikki mahdolliset tilanteet mihin tekoäly voi joutua ja kertoa tekoälylle mitä tällaisessa tilanteessa pitää tehdä.

Yksi tapa tähän on Äärellinen automaatti (Finite-State Machine), jossa jokaisella pelattavalla objektilla on jokin tila, kun objekti täyttää tietyt kriteerit se siirtyy toiseen tilaan. Tämä lienee helpommin hahmotettavissa esimerkin kautta. Bourg ja Seemann esittivät kirjassaan hyvän esimerkin tästä. Siinä seurataan muurahaisia, jokaisella muurahaisella on siinä tiloja.

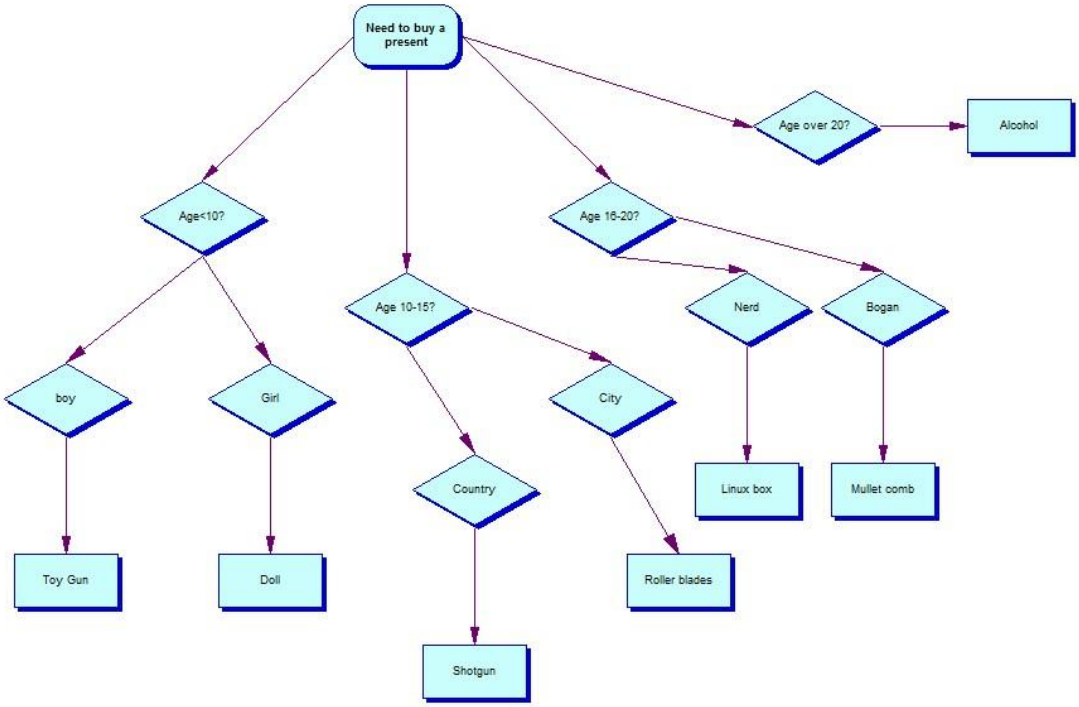
Kaikki muurahaiset aloittavat etsien ruokaa. Kun ne löytävät ruokaa, niiden tila vaihtuu. Nyt muurahainen vie ruoan pesään. Tämän tehtyään muurahaisen tila vaihtuu taas ja nyt se on janoinen raatamisesta. Tästä johtuen se etsii vettä. Kun se lopulta löytää vettä tila siirtyy takaisin ruoan etsimiseen ja kaikki alkaa alusta.

Tämä oli kuitenkin hyvin yksinkertainen demo, jossa oli vain 3 eri tilaa. Suuremmissa peleissä objekteilla todennäköisesti olisi paljon enemmän tiloja, eikä niiden vaihtuminen välttämättä olisi noin suoraviivaista. Tiloja voisi olla vaikkapa hyökkääminen, resurssien kerääminen, pakeneminen tai jopa keskusteleminen. (Bourg, & Seemann 2004.)



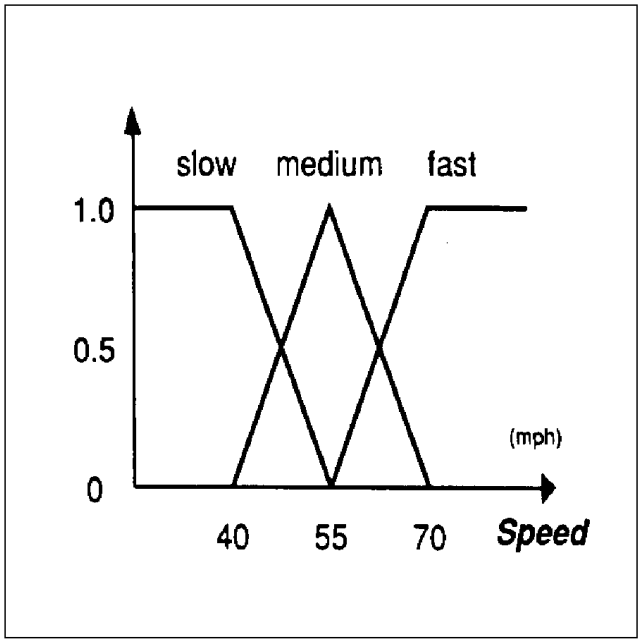
KUVA 6. Äärellinen automaatti

Toinen vaihtoehto, mitä käytetään pelitekoälyssä, on päätöspuu(*decision tree*), joka muistuttaa rakenteeltaan puuta. Siinä tekoäly etsii ratkaisua pienempien päätösten ja niiden seurauksien kautta. Päätöspuita käytetään lähinnä tekemään analyysi siitä, mikä vaihtoehto todennäköisimmin johtaisi haluttuun lopputulokseen. Päätöspuussa ongelman ratkaiseminen aloitetaan aina juuresta, eli siitä mistä kaikki nuolet lähtevät. Siitä mennään aina siihen suuntaan, mikä vastaa ongelman tilannetta. Kun haaranpäähän, joka ei enää jatku, niin ratkaisu on löytynyt. (vceit.com.)



KUVA 7. Päätöspuu (vceit.com)

Lisäksi pelitekoälyssä voi käyttää sumeaa logiikkaa. Sumea logiikka mahdollistaa ohjelmoinnissa käsitteen osittain totta. Eli esimerkiksi hahmo ei ole enää joko nuori tai vanha, vaan hän voi olla jotakin siltä väliltä. Ei ihminenkään ajattele, että on jokin tietty ikä milloin henkilö ei ole enää nuori vaan nyt hän muuttui vanhaksi. Näin ollen 30-vuotias olisi osittain nuori.



KUVA 8. Sumea logiikka kuvaamassa nopeutta (Localdesigns.com)

Koodissa se voisi näkyä vaikka asteikolla 0,6 nuori. Sumeaa logiikkaa voi käyttää pelissä vaikka tekoälyn aseen valitsemiseen. Aseen valinta voisi riippua vaikka kuinka kaukana vihollinen on. Jos vihollinen on melko lähellä, niin tekoäly voisi valita hahmolle miekan ja hyökätä päälle. Jos taas vihollinen olisi kauempana, niin jousi olisi paljon parempi ase.

4 CASE: STRATOGEAR

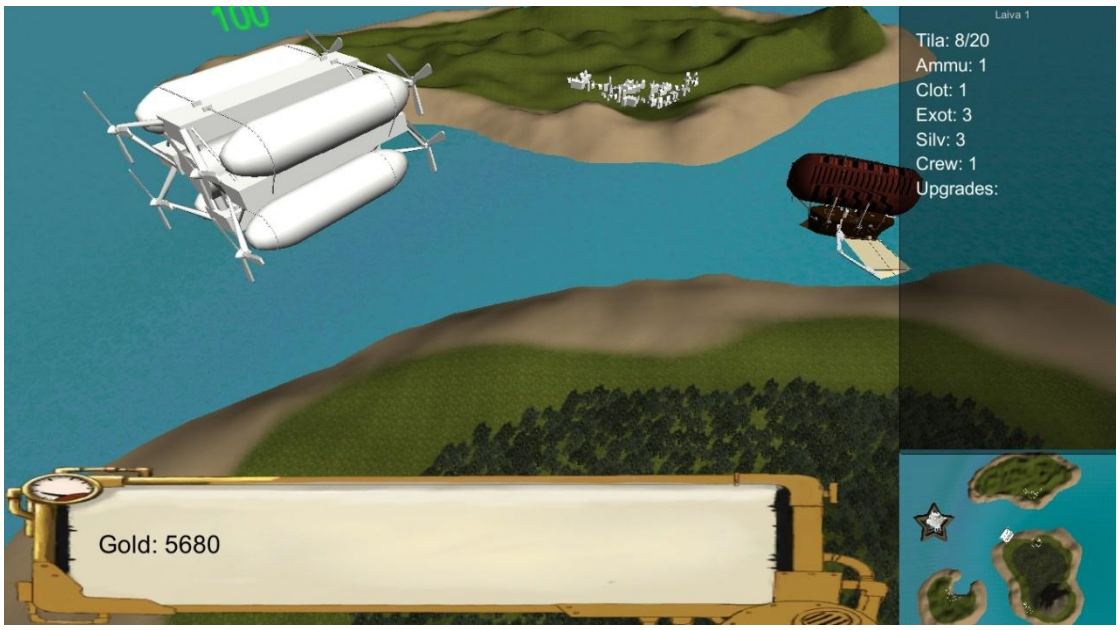
Projektimme oli siis tietokonepeli, jossa hallitaan steampunk-tyylisiä ilmalaivoja. Näillä laivoilla olisi tarkoitus kasvattaa laivaimperiumiasi jollain kolmesta eri tavasta, tai miksei vaikka useammallakin. Tapoja on siis kauppaaminen, jossa luonnollisesti lennetään kaupunkien välillä ja ostetaan halvalla ja myydään kalliilla. Tämä on ehkä pelin turvallisin tapa tienata. Toinen tapa on toisten laivojen ryöstäminen, tätä varten pelaajan pitää siis aseistaa laivaansa, että pelaajalla olisi minkäänlaista mahdollisuutta onnistua. Kolmas tapa on vartiosto, jolla pelaaja puolustaa muita laivoja rosvoilta ja pidättää tai ampuu ryöstäjät.

KUVA 9. Stratogearin logo

Peli suunniteltiin useamman pelaajan pelattavaksi samanaikaisesti, joten pelaajat voisivat ryöstellä ja jopa taistella toistensa kanssa. Peliin kuitenkin tarvittiin tekoäly pelaajia, jotta sitä voisi pelata yksin ja jotta pienemmissä useamman pelaajan pelissä laivoja olisi hiukan enemmän. Opinnäytetyönä tällainen peli olisi ollut liian suuri tehtäväksi kokonaan. Tästä syystä pelistä oli siis tarkoitus alun perinkin tehdä pelkästään pieni demo, sillä koko pelin valmiiksi asti vieminen vaatisi monen vuoden työpanoksen.

Demo sisältäisi vain kaikkein olennaisimmat osat. Demoon kuuluu pieni saaristo, johon kuuluu neljä saarta ja neljä kaupunkia. Siinä voisi käydä kauppaa kaupunkien välillä ja jokaisessa kaupungissa tuote tarjonta vaihtelisi ja hinnat muuttuisi tuotteiden saatavuuden mukaan.

Kaupungeissa voi myös ostaa muutamia erilaisia tykkejä laivaan tai hankkia kokonaan uuden laivan. Lisäksi demossa olisi myös tekoäly, joka ohjaa omaa laivaansa ja käy kauppaa. Halutessaan pelaaja voisi myös hyökätä tekoälylaivan kimppuun.



KUVA 10. Pelin perusnäkymä

Perusnäkymässä ruudulla on kartta, sivupalkki, jossa näkyy oman laivan tiedot ja kyydissä olevat tuotteet, sekä alapalkki, jossa näkyy yleistä tietoa pelitilanteesta. Tässä näkymässä hoidetaan kaikki laivojen liikkumiset ja toiminnat muiden laivojen kanssa. Lisäksi demossa on kaupunki näkymä, joka ilmestyy pelaajalle, kun laiva saapuu kaupunkiin. Tässä näkymässä pystyy ostamaan ja myymään tuotteita, varustamaan laivaa paremmilla tykeillä, sekä ostamaan ihan uusia laivoja.

KUVA 11. Pelin kaupunkinäkymä

Demo alkaa tutorial tyylisellä opastuksella, jossa opas neuvoo pelaajaa ja opettaa demon perusasiat, kuten sen mistä varustat laivaa ja mistä ostat tuotteita. Opas myös kertoo hiukan pelin maailmasta. Näin pelaaja pääsee helpommin peliin sisälle ja hän pääsee nopeammin nauttimaan itse pelistä

4.1 Työvälineet ja menetelmät

Projektissa käytimme lähinnä Unity3D pelimoottoria ja koodikielenä toimi C#. Vaihtoehtoja kuitenkin olisi ollut enemmänkin. Unityn lisäksi mietimme myös Microsoft studiota, jota olimme kaikki käyttäneet aikaisemmin. Molemmilla oli, sekä hyviä että huonoja puolia. C Sharpin lisäksi olisimme myös voineet käyttää Javaa, sillä Unitylla onnistuu myös Javalla koodaaminen.

Projektissa tarvitsimme myös useita malleja peliin, joten tarvitsimme myös 3d-mallinnusohjelman. Tähän tehtävään käytimme Autodeskin 3ds maxia. Itse en tosin sitä omassa osuudessa käyttänyt, joten ei siitä sen enempää.

Unity3D

Unity3D on siis pelimoottori, joka on tällä hetkellä hyvin yleisesti käytetty pelifirmoissa ympäri maailmaa. Valitsimme sen tähän projektiin lähinnä sen helpon ja kätevän terrain editorin takia, joka tässä projektissa mahdollisti nopean alun projektille, koska emme joutuneet erikseen mallintamaan pelialuettamme. Unity3D vaikutti meistä myös helpolta käyttää ja oppia.

C Sharp

C Sharp eli C# on Microsoftin kehittämä koodikieli, joka yhdistää C++:n tehokkuuden ja Javan helppokäyttöisyyden. Valitsimme ohjelmointikieleksi C#:in, koska pidimme sitä riittävän monipuolisena, mutta kuitenkin helppona käyttää.

4.2 Tekoälyn toteutus

Itse opinnäytetyön aiheena minulla oli siis tekoäly, joten siirrytäänpä nyt siihen. Tekoäly oli hyvin olennainen osa projektiamme. Tekoälyä tarvittiin projektissamme enimmiltä osin tietokoneen ohjaamiin laivoihin, mutta lisäksi sitä tarvittiin. Pelaajankin laivoihin. Siihen miten laivat toimii kun käsket niiden hyökätä tai miten ne puolustautuvat muilta hyökkääjiltä.

4.2.1 Tekoälyn laivanhallinta

Pelissä tarvittiin hyvin paljon tekoälyä juuri ilmalaivoihin. Laivat kuitenkin olivat keskeisessä osassa peliä, joten niiden piti toimia hyvin. Tekoälyjen ohjaamat laivat kokonaisuudessaan menisi äärellisellä automaatilla, mutta pelaajan ohjaamatkin laivat tarvitsivat hiukan tekoälyä.

Ensinnäkin, laivan piti lentää luonnollisen näköisesti. Ei riitä, että laiva vaan lentää suoraan sinne, mihin on klikattu. Sen pitäisi lähteä hitaasti liikkeelle, jotta se näyttäisi kiihdyttävän, ja sama kohteeseen saavuttaessa. Ei laiva pysähdy kuin seinään perille saapuessa, se hidastaa sinne hitaasti. Näin myös halusin meidän laivojen toimivan. Käytännössä tämä onnistui muutamilla *if*- lauseilla ja pienellä laivan ohjauksella. Jos laivan kohdesijainti olisi kauempana, niin laivan lentonopeus kasvaisi aina tiettyyn nopeuteen asti. Jos taas laivan kohdesijainti olisi tarpeeksi lähellä, laiva hidastaisi ja lopulta myös pysähtyisi.

Se yksinään ei kuitenkaan riittänyt laivan sujuvan näköiseen lentämiseen. Joten seuraavaksi piti miettiä laivojen kääntymistä. Ohjelmoidessahan jos laittaa objektin kulkemaan johonkin pisteeseen, niin sehän lähtee kulkemaan sinnepäin, oli objekti alun perin miten päin tahansa. Joten laivat piti saada kääntymään sujuvasti kohdetta päin.

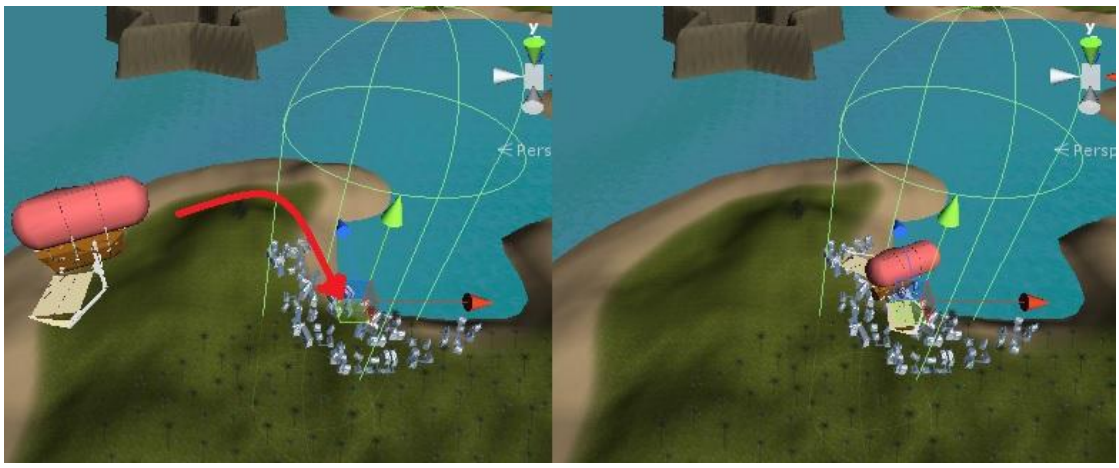
KUVA 12. Laivan kääntyminen

Tämä toteutui demossa koodilla, joka lukitsi laivat siten, että ne voivat liikkua vain eteenpäin. Tämän jälkeen kuvan 13 koodinpätkällä laivat laitettiin kääntymään pikkuhiljaa kohdetta päin. Näin saatiin haluttu lopputulos, jossa laivat liikkuivat sulavasti ja näyttivät heti paljon realistisemmilta.

```
targetRotation = Quaternion.LookRotation((TargetPosition - this.transform.position).normalized);
PlayerTransform.rotation = Quaternion.Slerp(PlayerTransform.rotation, targetRotation, Time.deltaTime * 2);
```

KUVA 13. Laivan Kääntymistä hoitavan koodin oleellinen pätkä

Näiden kahden seikan jälkeen laivat näyttivät lentävän jo sujuvasti. Laivojen piti kuitenkin myös laskeutua sujuvasti. Laivojen piti lentää muuten korkealla taivaalla, kunnes sen käskettäisiin laskeutua kaupunkiin, yleensä nappia painamalla. Laiva ei saanut laskeutua heti nappia painattaessa vaan vasta kun se on kaupungin lähellä ja vain sen kaupungin lähellä mihin sen on käsketty laskeutua.



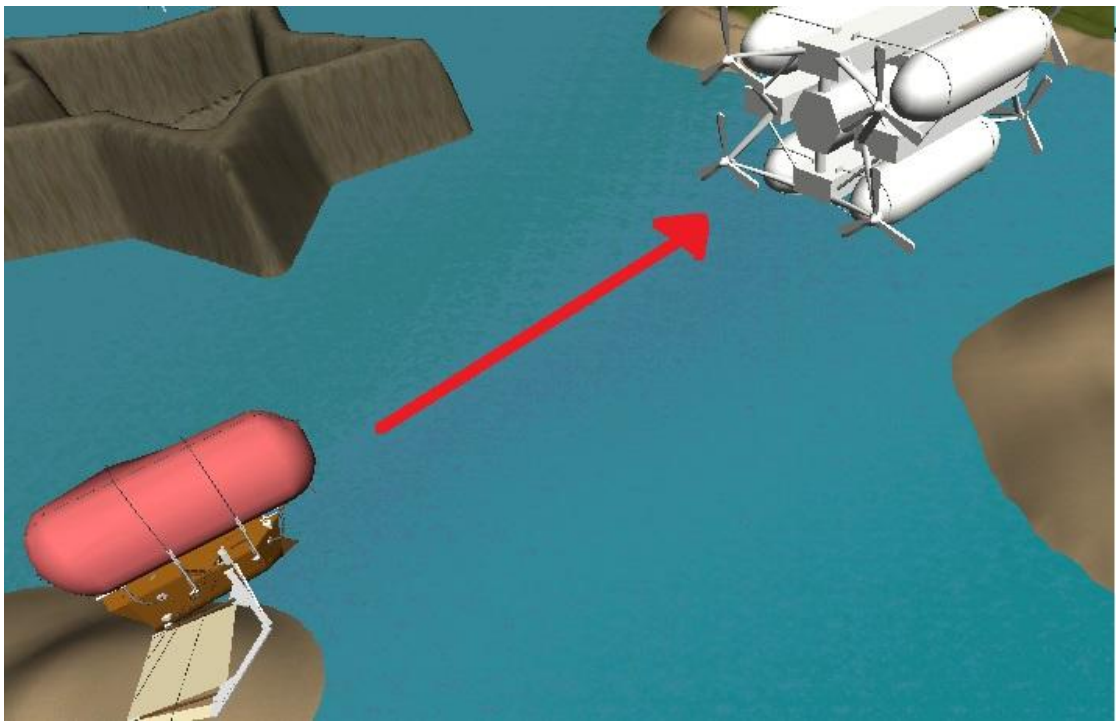
KUVA 14. Laivan laskeutuminen

Aluksi kaupungeille ja laivoille piti asettaa *hitbox*, jolla määritellään osuuko jokin johonkin. Näillä hitboxeilla tiedetään milloin laiva saapuu kaupungin lähelle. Sitten vain lisättiin laivalle nappula, jolla se sai komennon laskeutua kaupunkiin. Tämän jälkeen kun laiva saapuu sen kaupungin *hitboxiin*, mihin laivan on käsketty laskeutua, niin se alkaa laskeutumaan kaupunkiin samalla tavalla niin kuin kuvasssa X. Sama logiikkaa toimii myös kaupungista pois lähdettäessä. Kun laiva tuli ulos kaupungin alueelta, niin laiva lähti nousemaan takaisin lentokorkeuteensa.

```
61      //Jos ollaan laskeutumassa kaupunkiin
62      void OnTriggerEnter(Collider other) {
63          if(other.name.Substring(0,5) == "laiva")
64          {
65          }
66          else
67          {
68              if (this.GetComponent<Ominaisuudet>().attackCommand == true)
69                  {
70                  this.GetComponent<Ominaisuudet>().laskussa = true;
71                  this.TargetHeight = other.transform.position.y+25;
72                  print(this.GetComponent<Ominaisuudet>().laskussa);
73                  }
74          }
75      }
```

KUVA 15. Laivan laskeutumista hoitavan koodin pätkä

Nyt laivat osasivat lentää, nousta ja laskeutua. Seuraavaksi laivojen pitäisi osata käyttäytyä muiden laivojen kanssa. Mitä silloin tapahtuisi kuin laiva hyökkäisi toisen laivan kimppuun? Yleensä laiva varmaan lähtisi toista laivaa päin ja aloittaisi ampumaan tätä.



KUVA 16. Laivan hyökkääminen

Tätä varten kehitin laivalle tilan, jonka aikana laiva ampuisi hyökkäyksen kohteena olevaa laivaa. Nappia painattaessa laivan tila vaihtuisi ja se alkaisi seurata kohdetta ja ampuisi sitä. Kohteen tuhouduttua, laiva lopettaisi ampumisen ja tila vaihtuisi takaisin normaaliin.

```
// Jos laivassa on tykki aloitetaan ampuminen
aLaiva = GameObject.Find("Main Camera").GetComponent<PeliTiedot>().aktivLaiva;
if (GameObject.Find("laiva"+aLaiva).GetComponent<Ominaisuudet>().hyokkaysTila == 1)
{
    aLaiva = GameObject.Find("Main Camera").GetComponent<PeliTiedot>().aktivLaiva;
    GameObject.Find("laiva"+aLaiva).GetComponent<Ominaisuudet>().kohdeLaiva = "rahtilaiva-k";
    GameObject.Find("laiva"+aLaiva).transform.FindChild("laiva"+aLaiva+"-k")
        .FindChild("autocannon").FindChild("smallexplosion").particleEmitter.emit = true;
}
```

KUVA 17. Laivan ampumista hoitavan koodin pätkä

Lopullisessa pelissä laivoja voisi myös ryöstää, joten se tarvitsee jälleen uuden tilan laivalle. Siinä kohteen tuhoamisen sijaan laiva yrittäisikin vain päästä vierelle, jolloin miehistö voisi loikata toiseen laivaan ryöstämään sen. Peliin suunnittelimme myös useita erilaisia aseita, jotka toimivat erilailla. Tällöin jos laivassa olisi erilaisia aseita, niin sumea logiikka voisi valita parhaan tavan ampua sillä hetkellä.

4.2.2 Tekoälyn päätöksien teko

Projektissamme tekoälyn piti lähinnä osata pelata peliämme, joten piti miettiä. Miten se saadaan ostamaan tuotteita halvalla ja sitten myymään ne kalliilla toisessa kaupungissa. Pidemmälle vietynä sen olisi myös hyvä seurata hiukan missä kaupungeissa saa ostettua tiettyjä tuotteita halvalla ja mihin kannattaa mennä, että saa tuotteesta hyvän hinnan.

Äärellinen automaatti osoittautui tässä projektissa juuri sopivaksi. Perus tekoälyn laivat tarvitsivat alkuun tilan, missä ne lentelisivät kaupungista toiseen. Kaupungeissa tila vaihtuisi kaupanteko tilaan, jossa laiva ostaisi tuotteita, jotka se saisi halvalla ja myisi tuotteet, mistä saa paremman hinnan kuin millä tekoäly tuotteen osti.

Tämän jälkeen laiva lähtisi pois kaupungista ja tila vaihtuisi takaisin lentotilaan. Jotta tekoäly osaisi tehdä kannattavia kauppoja, niin sen pitää seurata hintoja kaupungeissa. Tämän tekoäly voisi ratkaista päätöspuun avulla. Jos hinta on halvempi kuin suurimmassa osassa kaupungeista, niin osta. Jos taas tuote on tavallista kalliimpi nykyisessä kaupungissa ja laivalla on tuotetta mukanaan, niin myy tuotteet. Yksinkertaisimmillaan se toimisi juuri näin.

Lisäksi tekoälyn pitäisi jossain vaiheessa valita alkaako se ryöstämään kauppaamaan tosissaan vai pitämään yllä järjestystä. Tässä kuitenkin halusin käyttää hiukan satunnaisuutta, jotteivät kaikki tekoälylaivat alkaisi tekemään ihan samaa asiaa. Valinnasta riippuen tekoäly alkaisi toimia hiukan erilailla. Joten siihen pitäisi tehdä samasta tekoälystä kolme eri suuntaa.

Kauppiaana tekoäly ostaisi lähinnä isompia rahtilaivoja ja kauppaisi suuria määriä tuotteita. Aseita se ostaisi lähinnä puolustustarkoitukseen, jotta laivoja ei niin helposti kaapattaisi. Tämän tyylisiä tekoälyn ohjaamia laivoja tarvittaisiin eniten, jotta maailma näyttäisi vilkkaalta eikä kaikkialla vain sodittaisi.

Ilmarosvona tekoäly keskittyisi toisten laivojen ryöstämiseen ja kaappaamiseen sekä myöhemmin kokonaisten kaupunkien ryöstäminen. Tällainen tekoäly parantelisi ensisijaisesti laivojen aseistusta ja hankkisi lähinnä nopeita laivoja joilla pääsee, sitten helposti pakoon vartiostoa. Rosvot käyvät kauppaa lähinnä vain ryöstämällään saaliilla.

Viimeiseksi on vielä ilmavartioston tekoäly. Tekoälyn tarkoitus olisi napata ryöstäjät ja pitää kaupankäynti turvallisena. ilmavartiostona pelaaminen on käytännössä freelancer lainvartijan hommia, jolla tienaa rosvoja pidättämällä. Tämä ei kuitenkaan toimi, jos lain vartijoita on liian paljon suhteessa rosvoihin, koska silloin ei tienaa tarpeeksi. Tästä syystä tällaisia tekoälyn laivoja on vähiten. Ilmavartiostona tekoäly ostaisi raskaita puolustustarkoitukseen tarkoitettuja laivoja, joilla pystyy auttamaan kauppalaivoja ja olemaan jopa valmis sotaan rosvojen kanssa. Tekoälyn tarkoitus olisi siis etsiä rosvoja ja turvata tärkeimmät kauppareitit.

5 YHTEENVETO

Pelin koodaaminen oli jo pitkään ollut minulla mielessä, mutta tämä projekti antoi lopulta siihen mahdollisuuden. Loppujen lopuksi projektina tämä oli erittäin antoisa. Opin itse matkan varrella paljon uusia asioita joista osasta en ollut kuullutkaan ennen. Kiinnostuin tekoälystä tämän aikana vain lisää. Opinnäytetyö oli myös mukavaa tehdä ryhmässä ja suosittelen sitä lämpimästi kaikille.

Vaikka meitä olikin neljä henkeä tämän projektin kanssa, niin jokainen kirjoitti kuitenkin oman raporttinsa ja jokaisella oli oma tutkimusongelma. Se ehkä mahdollistikin näin sujuvan yhteistyön ryhmän välillä. Jokainen pystyi tekemään sitä mikä itseään kiinnosti. Tero Parkkinen teki suunnittelusta ja siinä sivussa projektinhallinnasta, Heikki Mäki oli meidän mallintaja, jonka ansiosta peli edes näyttää joltain. Jari Vitikainen oli meidän ryhmämme toinen koodari ja perehtyi enemmän satunnaisuuteen, jota esiintyi pelissämme kaupankäynnissä ja taistelussa. Minä taas vastasin työssämme tekoälystä. Käytännössä minä ja Jari myös koodattiin pelin pohja.

Lisäksi meillä oli vähänaikaa apuna graafikko Sami Vitikainen, joka teki upeaa työtä piirtämällä pelin hahmot. Meillä oli myös muusikko tekemässä meille pelin musiikit, mutta tämän kirjoitushetkellä se oli vielä epävarmaa, ehtiikö hän saada ne tämän aikataulun rajoissa demoon mukaan.

Kuitenkin ydinporukkaamme kuului neljä ensin mainittua henkeä. Minusta se oli juuri sopivan kokoinen ryhmä. Jos meitä olisi ollut vähemmän, niin pelistä olisi jäänyt jotain uupumaan. Viides taas olisi ehkä ollut jo liikaa. Enemmän henkilöitä olisi tarkoittanut, enemmän mielipiteitä. Välillä neljänkin hengen ryhmässä oli ristiriitoja pelinsuunnasta, joten viidellä niitä olisi varmaan ollut vielä enemmän. Lisäksi mitä useampi tekijä on mukana, niin sitä hankalampaa voi olla hallinnoida koko projektia.

Osittain tämä myös onnistui siksi, että pystyimme hallinnoimaan ryhmän sisällä projektia. Kaikilla oli aina uusin versio pelistä saatavilla. eikä koodaajat tehneet samaan aikaan muutoksia peliin. Tällä estettiin lähinnä se, ettei kumpikaan tehnyt turhaa työtä, jos toinen tallensi oman uuden version toisen uuden version päälle.

Oma tavoitteeni oli siis tutkia miten tekoälyä käytetään peliohjelmoinnissa ja myös tehdä peliimme tekoälyä soveltamalla teorian oppeja. Työn jälkeen tiedän nyt minkälaista ja millä tavalla tekoälyä peleissä käytetään. Itse ajattelin ennen pelitekoälyn paljon suppeammaksi, kuin mitä se oikeasti on. Mutta nyt tiedän paremmin. Tekoälyä on tosiaan kaikkialla peleissä nykyään.

Tekoäly myös kehittyy jatkuvasti peleissä, sekä muuallakin. Ennen tekoälyn kehitystä peleissä on rajoittanut aika paljon koneiden tehot ja se, että muutkin pelin osa-alueet tarvitsevat koneen tehoja itselleen. Pelien graafinen puoli alkaa nyt kohta olla jo sitä luokkaa, että se muistuttaa pelottavan paljon oikeaa maailmaa. Tästä johtuen grafiikka peleissä ei välttämättä enää tarvitse tulevaisuudessa, niin suurta osuutta jatkuvasti tehokkaampien koneiden tehoista ja tekoälylle ja muille osa-alueille jää enemmän tehoja.

Projektiimme en tosin saanut tehtyä niin paljon tekoälyä, kuin olin alun perin suunnitellut. Pelin koodaaminen ei tosiaan sekään tapahdu ihan hetkessä. Tästä syystä osa tekoälystä jäi pelkästään tutkinnan ja suunnittelun puolelle. Olisinko kenties voinut tehdä jotain toisin? Todennäköisesti en. Tekoäly olisi ehkä voinut demota enemmänkin ennen pelin koodaamista, mutta silloin peli ei välttämättä olisi yhtä pitkällä kuin nyt tätä kirjoittaessani. Ja mitä hyötyä olisi peliin suunnitellusta tekoälystä, jota ei voi käyttää, koska peli ei ole vielä siinä kunnossa?

Alkuperäinen tavoite oli tehdä myös peli. Nopeasti siinä tosin kävi ilmi, että edes neljällä hengellä me ei tulla saamaan tätä peliä edes lähelle valmista tuotosta opinnäytetyön aikarajoissa. Ehkä ideamme oli vähän kunnianhimoinen. Tällainen suurempi peli vaatii helposti vuosien työn. Meidän aikataululla saimme kuitenkin pelistä toimivan proton. Demon, jota pystyy jo pelaamaan ja josta saa jo kuvan, minkälaisesta pelistä olisi kyse.

Sitä emme vielä tiedä jatkammeko pelin kehittämistä vai emme. Pelinkehitys oli kiinnostavaa ja hauskaa ja tulen toivottavasti jatkossakin tekemään jotain vastaavaa. Jos en peliä, niin ehkä sitten tekoälyä muualla. Näin suuren projektin loppuun vieminen kuitenkin tulisi maksamaan ainakin paljon aikaa. Ideoita kyllä peliin riittäisi ja ehkä jopa jonkinlaista osaamistakin tämän jälkeen.

LÄHTEET

AI Game programmers guild. 2011. History Of Game AI. gameai.com. http://gameai.com/wiki/index.php?title=History_Of_Game_AI Päivitetty: 7.18.2011. Luettu: 17.10.2013.

Bourg, David. M. & Seemann, Glenn. 2004. AI for Game Developers. O'Reilly Media.

Bradley, Peter. 2002. Turing Test and Machine Intelligence. mind.ilstu.edu. http://www.mind.ilstu.edu/curriculum/turing_machines/turing_test_and_machine_intelligence.php?modGUI=240&compGUI=1145&itemGUI=1956 Luettu: 9.10.2013.

Half-life. picgifs.com. 2013. picgifs.com. WWW-dokumentti. http://www.picgifs.com/wallpapers/half-life/animaatjes-half-life-77606-958374/ Luettu: 7.11.2013

Localdesigns.com. 2013. Fuzzy logic. logicaldesigns.com. WWW-dokumentti. http://www.logicaldesigns.com/Ldfuz1.gif Luettu: 12.11.2013

Mark, Dave. 2012. AI Architectures: A Culinary Guide (GDMag Article). intrinsicalgorithm.com. http://intrinsicalgorithm.com/IAonAI/2012/11/ai-architectures-a-culinary-guide-gdmag-article/ Luettu: 2.11.2013.

Millington, Ian. & Fudge, John. 2006. Artificial intelligence for games. Morgan Kaufmann Publishers

Oracle Thinkquest. 2013. The History of Artificial Intelligence. thinkquest.org. http://library.thinkquest.org/2705/history.html Luettu: 17.10.2013.

Pirovano, Michele. 2012. The use of Fuzzy Logic for Artificial Intelligence in Games. pdf-dokumentti. http://homes.di.unimi.it/~pirovano/pdf/fuzzy_ai_in_games.pdf Luettu: 9.10.2013

The Sims: Life Stories.2013. Mobygames.com. WWW-dokumentti. http://www.mobygames.com/images/shots/l/303594-the-sims-life-stories-windows-screenshot-talking-to-neighbourss.jpg Luettu: 2.11.2013.

unitygems.com. 2013. Learn how artificial intelligence has been applied to computer games and the various techniques that are used. unitygems.com WWW-dokumentti. http://unitygems.com/ai-games/ Luettu 24.10.2013

vceit.com. 2013. Decision tree. vceit.com WWW-dokumentti. http://vceit.com/m/visthink-decisiontree.jpg Luettu: 7.11.2013

wikimedia.org. 2013. Turing test. wikimedia.org. WWW-dokumentti. http://upload.wikimedia.org/wikipedia/commons/e/e4/Turing_Test_version_3.png Luettu: 5.11.2013

wonderhowto.com. 2013. Tower defense. wonderhowto.com. WWW-dokumentti. http://social-games.wonderhowto.com/inspiration/attack-balloons-and-supermonkeys-tower-defense-retrospective-0127076/ Luettu: 13.11.2013

Xu, Siyuan. 2008 . History of AI design in video games and its development in RTS games. WWW-dokumentti. https://sites.google.com/site/myangelcafe/articles/history_ai Luettu:9.11.2013