Riku Heino

Itseoppiva tekoäly pelissä

Opinnäytetyö
Tieto- ja viestintätekniikka

2019

Kaakkois-Suomen
ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Riku Heino	Insinööri (AMK)	Joulukuu 2019
Opinnäytetyön nimi Itseoppiva tekoäly pelissä		36 sivua 0 liitesivua
Toimeksiantaja Kaakkois-Suomen ammattikorkeakoulu, Gamelab		
Ohjaaja Niina Mässeli		

Tiivistelmä

Opinnäytetyö käsittelee oppivaa tekoälyä eli koneoppimista, sen historiaa ja kehittymistä tavallisesta tekoälystä oppivaan tekoälyyn sekä koneoppimisen neuroverkko menetelmää. Työ käsittelee myös peliä, joka soveltaa koneoppimista pelin vihollishahmoon.

Idea työhön lähti Kaakkois-Suomen ammattikorkeakoulun tieto- ja viestintätekniikan Gamelab-oppimisympäristön mielenkiinnosta pelillistää koneoppiminen sekä seurata sen oppimistoimintaa. Työssä seurataan peliin luodun vihollishahmon reagoimista eri tilanteissa.

Peliprojekti on luotu Unity-pelimoottorilla käyttäen C#-ohjelmointikieltä. Peliin luotu vihollinen käyttää neuroverkoilla toimivaa koneoppimisalgoritmia ja oppii jokaisen kuoleman jälkeen suojautumaan paremmin. Opinnäytetyössä käsitellään projektin koodin kulkua sekä perehdytään matriisiin, jonka ympärille neuroverkko rakentuu.

Opinnäytetyössä esiteltiin yksi mahdollinen tapa toteuttaa koneoppimista pelissä. Lopputuloksena saavutettiin tavoitteiden mukainen käyttäytyminen vihollisella, joka oppi muuttamaan käyttäytymistään pelaajan tuhottuaan sen.

Asiasanat

neuroverkot, peliohjelmointi, tekoäly, koneoppiminen

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Riku Heino	Bachelor of Engineering	December 2019
Thesis title Self-learning AI in game		36 pages 0 pages of appendices
Commissioned by South-Eastern Finland University of Applied Sciences, Gamelab		
Supervisor Niina Mässeli		

Abstract

This thesis process provides an insight into self-learning artificial intelligence also known as machine learning, its history and evolution from normal artificial intelligence and neural networks. The work also processes with a game that applies machine learning to the enemy character of the game.

The idea for this work came from the desire of the information and communication technologies of South-Eastern Finland University of Applied Sciences to build a game that uses machine learning and to follow its learning activity. The game project also investigated how the enemy character reacts with different situations.

The project was created with Unity-game engine using C#-programming language. In the game the enemy uses a neural network algorithm and learns to defend himself better after each death. The thesis process examined the code flow of the project and the report explains the matrix around which the neural network is built.

The thesis presented one possible way to create a game that uses machine learning. The programming goal of this study was reached, as the enemy learned to change its behavior after each time it was destroyed by the player.

Keywords

neural network, game programming, artificial intelligence, machine learning

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena oli toteuttaa Kaakkois-Suomen ammattikorkeakoulun (XAMK) tieto- ja viestintätekniikan Gamelab-oppimisympäristölle peli, jossa pelin vihollinen toimisi koneoppimisella. Tarkoitus oli luoda vihollinen, joka oppisi jokaisella kierroksella suojautumaan paremmin käyttäen neuroverkkoja.

Opinnäytetyössä luodaan peleille tunnetun tekoälyn sijasta kehittyvää tekoälyä eli koneoppimista. Työssä käytetään neuroverkkoja, Unity-pelimoottoria sekä Blender-mallinnusohjelmaa. Ohjelmointi toteutettiin C#-ohjelmointikielellä.

Työn ensimmäisessä versiossa selvitettiin, mitä koneoppimisvaihtoehtoja on olemassa ja mikä niistä kaikista tukisi opinnäytetyön toteuttamista. Selvitystyön perusteella valittiin neuroverkkomenetelmä, koska se toimii kuin ihmisaivot.

Työ aloitettiin keräämällä tietoa neuroverkkojen luonnista sekä niiden toiminnasta. Kaakkois-Suomen ammattikorkeakoulussa järjestettiin ”Machine learning” -opintojakso, jossa käsiteltiin koneoppimista. Kurssilla luotiin kaksi koneoppivaa sovellusta, ensimmäinen käsitteli ”naive”-algoritmia, joka oppi laskemaan binäärejä ja toinen ”back-propagation”-algoritmia, joka oppi tunnistamaan roskapostin.

Opinnäytetyö käsittelee koneoppimista, koneoppimisen neuroverkkomenetelmää, sen historiaa ja kehittymistä tavallisesta tekoälystä sekä koneoppimisen ja tekoälyn eroja. Työ käsittelee myös Unity-pelimoottorilla luotua peliä, joka soveltaa koneoppimista pelin vihollishahmoon.

2 TEKOÄLY JA KONEOPPIMINEN

Tekoäly on tietokoneohjelmassa tai pelissä käytetty tapa saada jotain tehtyä automaattisesti sekä älykkäästi. Peleissä, joissa käytetään tekoälyllä toimivia hahmoja, kutsutaan NPC-pelaajiksi (non playable character). NPC-pelaajat pyritään ohjelmoimaan kuin se olisi ihmispelaaja. (Mills 2018.)

Tekoäly ja koneoppiminen voivat mennä usein sekaisin. Koneoppimista voidaan pitää yhtenä tekoälyn suuntana. Koneoppimisen idea perustuu datan prosessoimiseen, jota se hyödyntää oppimiseensa ilman ihmisen jatkuvaa valvontaa. (Mills 2018.)

Koneoppimiselle on olemassa monia valmiita kirjastoja, jotka lisättynä projektiin tekevät suuren osan työstä. Näistä kirjastoista suurin osa on kuitenkin kirjoitettu Python-ohjelmointikielelle, joka on hyvin yleiskäyttöinen ja moniin eri käyttötapauksiin soveltuva. (Boiko 2018.) Työn tarkoitus oli kuitenkin tutustua koodin sisältöön ja siksi kirjoittaa koodi itse käyttämättä valmiita koneoppimiskirjastoja.

2.1 Katsaus tekoälyn historiasta tähän päivään

Toisen maailmansodan aikaan vuonna 1940 saksalaiset käyttivät Enigma nimistä salauslaitetta viestien lähettämisessä. Englantilaiset matemaatikot purkivat kyseisen salauslaitteen koodeja käsin. Koodinpurku käsin oli kuitenkin hyvin hidasta, joten matemaatikko Alan Turing alkoi kehitellä konetta, joka purkaisi koodit. Tämä kone kokeili eri asetuksia sekä poisti ne, jotka oli jo kokeiltu. Tätä konetta kutsuttiin nimellä Bombe ja se oli ensimmäinen tekoälyä käyttävä tietokone. (Turing 2012, 7-9.)

Ensimmäinen koneoppimiseen perustuva termi ”neural network” (neuroverkko) keksittiin jo vuonna 1943. Se oli Waren McCullochin sekä Walter Pittsin suunnittelema teoriamalli. (Mohammed ym. 2017, 20.)

Kuva 1. Alan Turingin luoma ensimmäinen tietokone (Guinness 2018)

Kuvassa 1 on ensimmäiseksi tietokoneeksi luokiteltu Bombe-kone. Samalla tavalla kuin koodinpurkajat kokeilevat eri merkkivariaatioita paperille, kone lukee ja kirjoittaa symboleja pitkään nauhaan, jota se käyttää datanaan. Tämän jälkeen kone siirtää nauhaa oikealle tai vasemmalle. (Turing 2012, 10.)

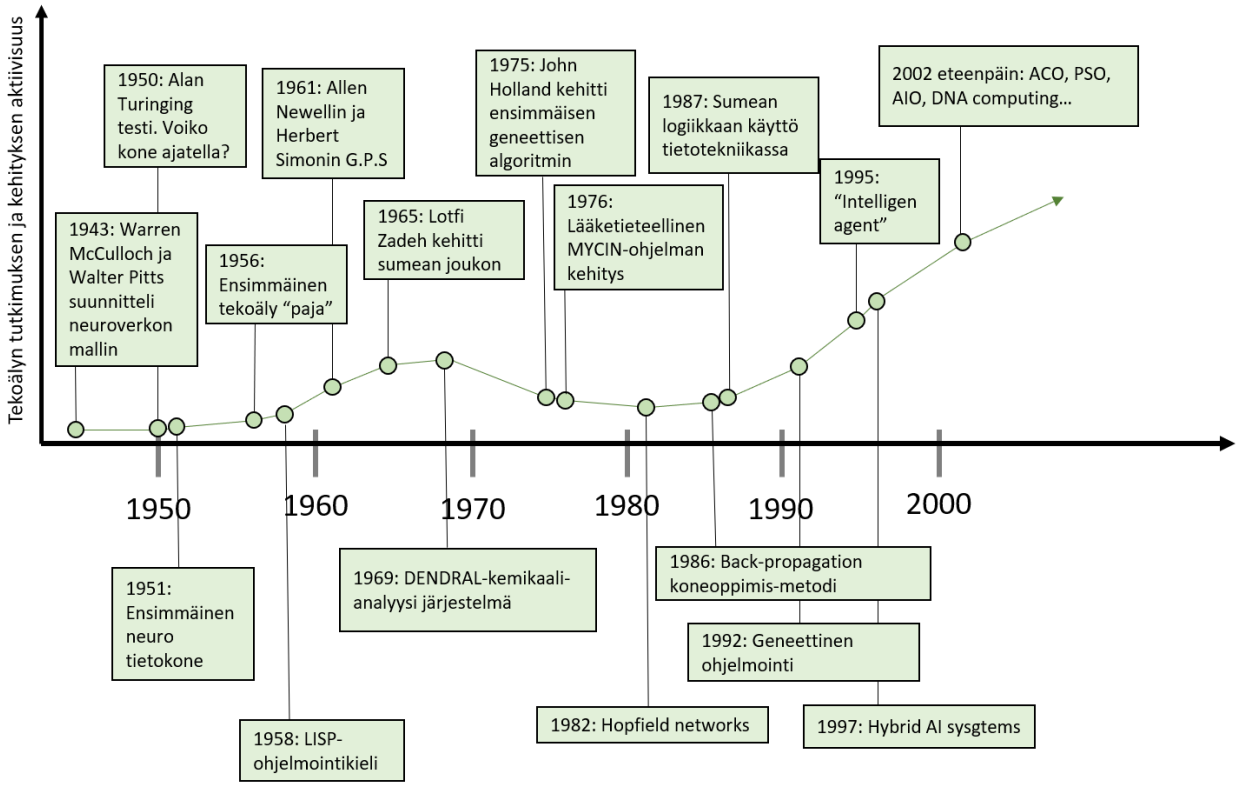
Loogisen teoreetikon ohjelmaa esitettiin vuonna 1956 Claude Shannonin sekä IBM-teknologiayrityksen John McCarthyn, Marvin Minksyn sekä Nathan Rochesterin järjestämässä Darthmouth Summer Research on Artificial Intelligence -konferenssissa Yhdysvalloissa. Samana vuonna pidettiin myös toinen Darthmouth-konferenssi, jossa termiä ”artificial intelligence” (tekoäly) käytettiin ensimmäistä kertaa. (Mohammed ym. 2017, 20)

Vuonna 1957 keksittiin ensimmäinen perceptron binäärin lasku tietokoneohjelma, sen kehitti Frank Rosenblatt. Seuraavana vuonna John McCarthy kehitteli LISP-ohjelmointikielen. (Mohammed ym. 2017, 20)

Massachusettsin teknologiainstituutin professori Joseph Weizenbaum loi vuosina 1964–1965 ELIZA nimisen keskustelurobotin. Eliza osasi tulkita tietokoneelle kirjoitetun englannin kielisen tekstin ja vastata siihen. (Markoff 2008.)

Shakin pelaamiseen luotu IBM-teknologiayrityksen Deep Blue voitti shakin hallitsevan maailmanmestarin Garry Kasparovin vuonna 1997. Tämä oli suuri askel tekoälyohjelmoinnissa. (Smith ym. 2006, 10.)

Vuonna 2016 Google DeepMindin luoma AlphaGo-tekoälyohjelma voitti Go-lautapelin hallitsevan maailman mestarin Lee Sadolin. AlphaGo-tekoälyohjelma käytti pilvilaskentaa sekä etsintäalgoritmin ja neuroverkolla toimivan päätöksentekoalgoritmin yhdistelmää. (Wang 2019.)



Kuva 2. Tekoälyn aikajana (Wang 2019)

Kuvassa 2 näkyvä käyrä kuvaa tekoälyn kehitystä ennen 1950-lukua aina 2000-luvulle asti. Kehityksessä näkyy, kuinka tekoälyn kehitys on kiihtynyt

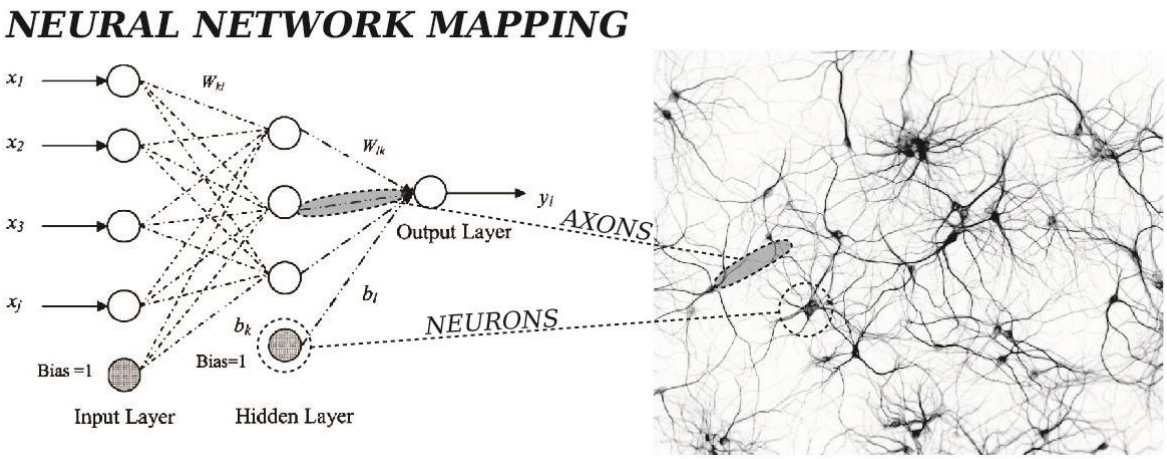
1960-luvulla, mutta lähtenyt pieneen laskuun hieman ennen 1970-lukua. Kuvassa luetellaan muun muassa sumeajoukko, DENDRAL-kemikaalianalyysijärjestelmä, MYCIN-ohjelma, Hopfiel networks, geneettinen ohjelmointi sekä monia muita, jotka ovat olleet historian saatossa tärkeitä tutkimus- ja kehitysmenetelmiä tekoälylle sekä koneoppimiselle.

Tietojenkäsittelyn ja teknologian uudistuessa koneoppiminen on muuttunut hyvin paljon. Monen koneoppimisalgoritmin pitkä ikä on luonut niistä tehokkaampia. Ne voivat laskea suuria datamääriä sisältäviä monimutkaisia laskutoimituksia paljon nopeammin kuin ennen. Nykyään koneoppimista käytetään hyvin arkipäiväisissä asioissa, kuten esimerkiksi Amazon-verkkokaupan ja Netflix-suoratoistopalvelun suosituksissa sekä sosiaalisessa mediassa. Suurimpia innovaatioita ovat itseajavat autot. (Evolution of machine learning 2019.)

Tekoälyn ja koneoppimisen edistyessä ja kasvaessa jokapäiväisessä elämässä on pelko niitä kohtaan noussut. Jotkut ihmiset pitävät termiä ”robotti” hyvin negatiivisena. Uskotaan, että tämä robotti vie ihmisten työpaikat tai pahimmassa tapauksessa alkaa kehittää itseään ja orjuuttaa ihmiskunnan. Toiset taas ajattelevat positiivisemmin ja uskovat tekoälyn ja robottien ottavan ihmisille sopimattomia sekä vähemmän sopivia työtehtäviä ja näin tuovan uusia työtehtäviä. (Jääskeläinen 2019.)

2.2 Neuroverkot

Neuroverkko on ihmisaivoja simuloiva verkosto, jossa kulkee dataa neuronilta toiselle. Niiden kerrotaan simuloivan ihmisten aivoja. Aivoissa on soluja nimeltä neuronit. Jokainen neuroni saa sisääntulosignaaleja aisteista. Kun signaali on tarpeeksi vahva, neuroni aktivoituu. Kun neuroni aktivoituu, se lähettää signaaleja eteenpäin toisille neuroneille. Tätä tapahtuu koko ajan pään sisällä, joka liikuttaa lihaksia, hallitsee tunteita ja kaikkea ihmisen käyttäytymistä. (Hulten 2018, 205.)



Kuva 3. Neuroverkkojen vertaus aivosoluihin (Stanton 2014)

Kuvassa 3 verrataan kuvitettua neuroverkkoa aivosolujen kanssa. Molemmista löytyy neuronit sekä niistä lähtevät haarakkeet.

2.2.1 Neuroverkkojen luonti

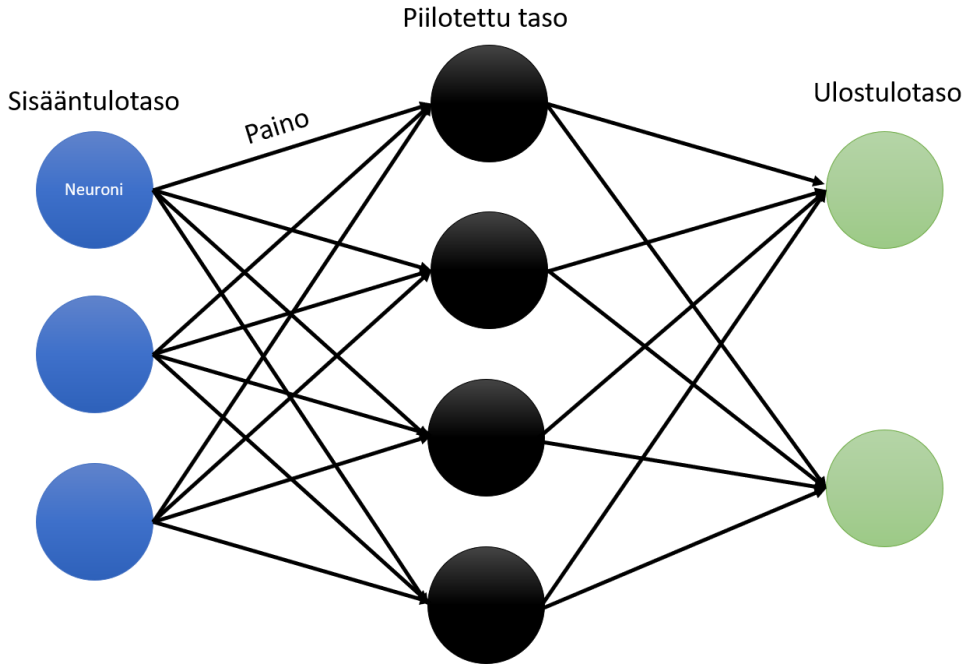
Neuroverkot luodaan usein kolmella neuronitasolla. Sisääntulotaso, piilotettu-taso sekä ulostulotaso. Jokaisella neuronilla on arvo nollasta yhteen mikä aktivoi sen. Neuronit ovat liitoksissa toisiinsa yksisuuntaisilla painoilla. Kyseiset painot ovat vertauskuvia biologisille hermoliitoksille, jotka aktivoivat seuraavan neuronin. (Robbins 2014, 392.)

Dataa syötetään verkkoon asettamalla aktiivitasot sisääntuloneuroneille. Tämän jälkeen piilotetut neuronit ottavat tietyn verran stimulaatiota, joka on sisäisen vaikutuksen kanssa saman arvoinen. Tämä summataan sisääntulokerroksen kunkin neuronin tulojen sekä siihen yhdistetyn painon kanssa. Sen jälkeen jokaisen neuronin aktiivisuus piilotetussa kerroksessa lasketaan asettamalla ei-lineaarinen aktivointi funktioarvoon, joka osoittaa sen stimulaatiotasoa. Tämä toistetaan verkoston jokaisessa tasossa ottamalla stimulaatio edeltävistä tasoista ennen kuin verkoston aktiivisten tasojen ulostuloneuronit päivitetään. Tämä muodostaa verkoston ulostulon. (Robbins 2014, 392.)

Koska neuroverkot sisältävät piilotettua dataa niiden virheiden etsintä on hyvin vaikeaa. Selkeää syytä päätöksien tekoon ei voi saada selville, eikä painojen

arvoja muuttaa. Yleisesti ottaen, jos ohjelma ei esiinny halutulla tavalla, syy on sisään tulevassa datassa (Robbins 2014, 396.)

Neuroverkkojen oppiessa ne osaavat tehdä tarkempia päätöksiä. Niiden vahvuus on kyky oppia annetun datan tiedosta ja miten se voidaan parhaiten yhdistää ennustettavaan lopputulokseen. (Brownlee 2016.)



Kuva 4. Yksinkertainen neuroverkko

Kuvassa 4 on kuvitettu hyvin yksinkertainen neuroverkko. Kyseisessä neuroverkossa on kolme sisääntuloneuronia, jotka ottavat dataa. Sen jälkeen aktivoitunut paino siirtää dataa piilotettuun tasoon. Piilotetun tason neuronit aktivoivat seuraavan painon, joka siirtää sen ulostulotasolle.

Piilotettujen tasojen ja niiden neuronien oikeaan määrään ei ole oikeaa kaavaa tai ratkaisua. Ainoastaan lineaarisissa ohjelmissa ei ole tarvetta piilotetuille kerroksille. Ei-lineaarisille ohjelmille kuitenkin on olemassa suuntaa antavia ohjeita. Mitä enemmän neuroneja piilotetulla tasolla on, sitä monimutkaisempia asioita se voi oppia. Tämä kuitenkin vaikuttaa opetukseen tarvittavaa aikaan. Piilotettujen neuronien paras mahdollinen määrä on määritettävä kokeilemalla. Yksi tapa on aloittaa kahdella tai kolmella piilotetulla neuronilla ja lisätä yksitellen lisää. (Robbins 2014, 393.)

2.2.2 Neuroverkkojen opettaminen

Neuroverkkojen opetus tapahtuu syöttämällä toistuvasti dataa paritetun sisääntulon sekä kohdennetun ulostulon kanssa. Joka kerta kun sisääntulodata syötetään verkostoon, verkosto laskee sen ulostulon, jonka jälkeen verkoston painot sekä poikkeavuus muuttuu lähemmäksi kohdennettua tulosta. (Robbins 2014, 394.)

Tässä projektityössä opetetaan viholliselle suojautumista. Vihollisen neuroverkko ottaa dataa kahdelle sisääntuloneuronille. Sisään tuleva data on vihollisen kuolinpiste pelikentällä. Toinen ottaa x-akselin koodinaatit ja toinen z-akselin koordinaatit. Sen jälkeen neuroverkko prosessoi dataa neljässä piilotetussa tasossa, joissa kussakin on viisi neuronia. Ulostuloneuroneita on myös kaksi, joiden data antaa viholliselle uuden pisteen, johon pyrkiä.

3 PELIT JA KONEOPPIMINEN

Suosittu tapa käyttää koneoppimista videopeleissä on opettaa tekoäly pelaamaan kyseistä peliä. Tätä on käytetty mm. Mortal Kombat -tappelupelissä sekä Sonic the Hedgehog- ja Super Mario -tasohyppelypeleissä. (Luo s.a.)

Pelejä, joissa käytetään koneoppimista osana pelimekaniikkaa, kuten tässä työssä, on myös muutamia. Tunnettu rallipelisarja Forza Motorsport käyttää koneoppimista opettaakseen NPC-kuskeja ajamaan paremmin.

Cambridgen Microsoftin tutkimuskampuksella työskentelevä Drivatar-tiimi on luonut Forza Motorsport -rallipeliin pelaajan taitoihin perustuvan mukautuvan tekoälyn. Drivatar tallentaa telemetrisesti missä pelaaja on pelikentällä, mitä autoa pelaaja käyttää, kaasun hallinnan sekä muiden autojen sijainnin. Peli seuraa myös relaatiotietoja sekä korreloi pelaajan toimintaa muiden autojen kanssa. Tämä opettaa algoritmille, miksi ja milloin pelaajat tekevät tiettyjä toimia. (Orland 2013.)

Toinen hyvin yksinkertainen neuroverkkoa käyttävä peli on Googlen luoma ”Quick, Draw!”. Peli on rakennettu koneoppimisella, jossa pelaajan on piirrettävä asia, jonka peli kertoo. Neuroverkko yrittää arvata mitä pelaaja piirtää.

Mitä enemmän peliä pelataan, sitä enemmän peli oppii. (Google Creative Lab 2017.)

Koneoppimisen käytettävyys peleissä on tällä hetkellä hyvin pientä, johtuen sen arvaamattomuudesta. Vain tutkijat sekä indie-kehittäjät käyttävät tämäntyyppisiä kokeiluja. On helpompaa ohjelmoida tekoäly tekemään rajoitettuja asioita ja viedä pelaajan huomio osittain johonkin muuhun. Näin pelaaja kokee tekoälyn viisaaksi, vaikka tarkemmin tutkittuna se suorittaa vain yksinkertaisia käskyjä. (Statt 2019.)

4 PELIPROJEKTIN TOTEUTUS

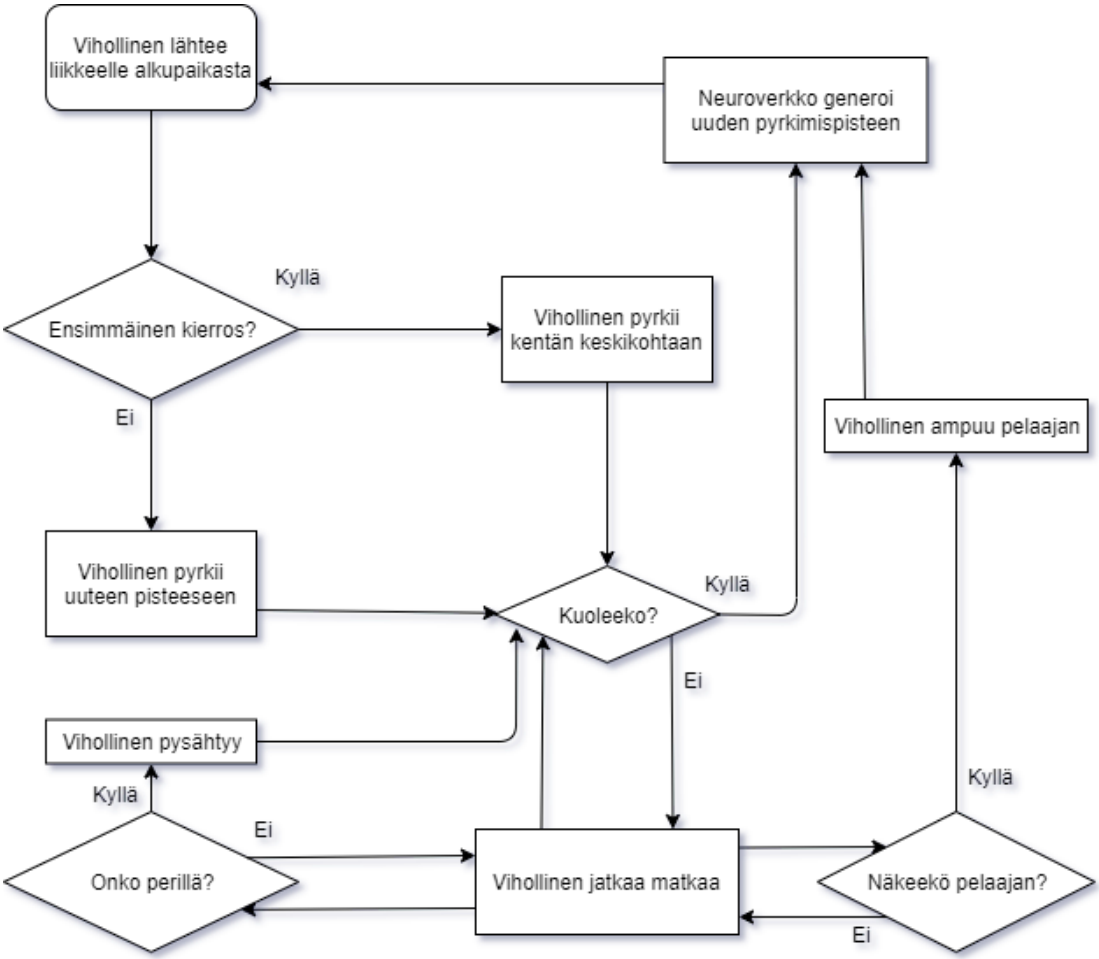
Ennen peliprojektin aloitusta selvitettiin, mikä pelimoottori tukee parhaiten algoritmikirjastoja ja onko niistä hyötyä peliprojektissa. Pelimoottorivaihtoehtoina olivat joko Unreal Engine tai Unity.

Muutama yleinen pelimoottori mukaan lukien Unreal Engine ja Unity sisälsivät oman navigointialgoritmikirjaston, joka oli projektin tärkein ominaisuus. Koska myös Unity sisälsi kyseisen kirjaston, oli se paras valinta projektin pelimoottoriksi.

Esimerkkejä neuroverkkojen luomiseen löytyi todella vähän, varsinkin C#-ohjelmointikielelle. YouTube-videopalvelusta kuitenkin löytyi tätä työtä koskettava ohjevideo. Siinä luotiin juuri Unity-pelimoottorilla sekä C#-ohjelmointikielellä neuroverkkoa käyttäviä objekteja. Erona tähän työhön oli kaksiulotteisuus sekä se, että siinä objektit opettelivat seuraamaan käyttäjän kursoria. Kyseisestä videomateriaalista sai kuitenkin vahvan pohjan neuroverkkoprojektiin.

4.1 Tavoitteet

Ensimmäisenä tuli suunnitella vihollisen toiminta ja käyttäytyminen. Toimeksiantajalta saatiin idea suojaan pyrkivään viholliseen, josta peliprojekti oli hyvä aloittaa. Neuroverkkojen tiedonpuute esti sen suoran aloittamisen, joten ensimmäisessä versiossa tutustuttiin erilaisiin menetelmiin toteuttaa projekti ja opeteltiin Unity-pelimoottorin Navigation tool -reitinhakutyökalun käyttöä sekä suunniteltiin tarkoitukseen soveltuva kenttä.



Kuva 5. Vihollisen käyttäytymisen vuokaavio

Kuvassa 5 on tuotu vihollisen käyttäytymisen yksinkertaisuus vuokaaviona. Vihollisen lähtiessä liikkeelle pyrkii se heti pisteeseen, jonka neuroverkko on sille generoinut, ellei kyseessä ole pelin ensimmäinen kierros. Vihollisen pyrkiessä pisteeseen peli tarkastelee, kuoleeko vihollinen tai havaitseeko se pelaajan. Jos vihollinen kuolee, generoi neuroverkko sille uuden pyrkimispisteen käyttäen sisääntuloarvoina vihollisen kuolin pistettä ja alkaa uusi kierros. Jos vihollinen havaitsee pelaajan, pelaaja kuolee ja alkaa uusi kierros. Vihollisen päästessä pisteeseen, johon se alun perin pyrki, jää vihollinen paikalleen pelin silti tarkastellessa sen tilaa.

Alun perin vihollisen ei kuulunut hyökätä, mutta lopulliseen versioon luotiin ampuva vihollinen. Kentästä oli tarkoitus saada tasapainoinen, paljon esteitä, joiden taakse piiloutua, mutta ei sokkeloinen. Tästä alettiin muuttamaan projektia oikeaan suuntaan. Suunnitelmiin kuului myös pelaajan peliin liittyvien tietojen tallentaminen tekstitiedostolle.

Tavoite oli saada aikaan ”itse ajatteleva” ja ”virheistään oppiva” vihollinen. Vihollisen tarkoitus oli aluksi olla ”tyhmä”, mutta ”viisastua” joka kuoleman jälkeen. Mitä enemmän dataa vihollinen saa, sitä ”viisaammaksi” se tulee.

Peliä on tarkoitus testauttaa eri henkilöillä ja saada heidän tietonsa tallennettua testituloksia varten. Tämän vuoksi pelin oli oltava tietokoneen kapasiteetin kannalta kevyt käyttää. Pelin oli tarkoitus myös kirjata tuloksia tekstitiedostolle, mutta tämä toimi ainoastaan Unity-pelimoottorin pelitilassa.

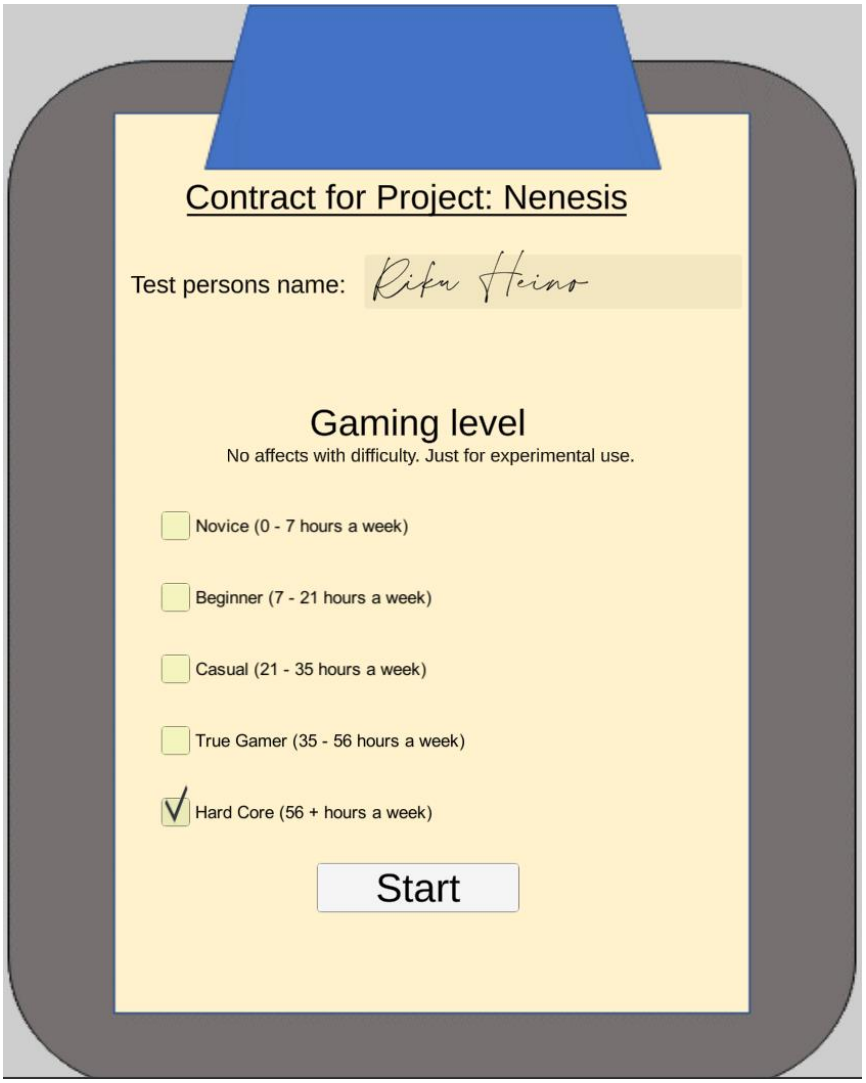
4.2 Valitut menetelmät

Seuraavaksi tuli ottaa selvää sopivasta koneoppimismenetelmästä. Koska neuroverkot toimivat kuten ihmisaivot, olivat ne täydellinen valinta projektiin.

Pelikenttänä toimii varastohalli. Hallissa on laatikoita, joiden taakse piiloutua. Pelaaja sekä vihollinen syntyvät sopivan matkan päässä toisistaan, jotta vihollisella on mahdollisuus päästä karkuun. Alkuperäisessä ideassa oli tarkoitus, että Pelaajalla on viisi minuuttia aikaa tuhota vihollinen tai alkaa uusi kierros ja pelaaja ei saa pistettä. Vihollinen ei myöskään ampunut alkuperäisessä versiossa. Lopullisessa versiossa aikaraja poistettiin sen haitattua pelaamista sekä vihollisen oppimista. Viholliselle lisättiin myös ampuminen suorassa katseyhteydessä pelaajaan.

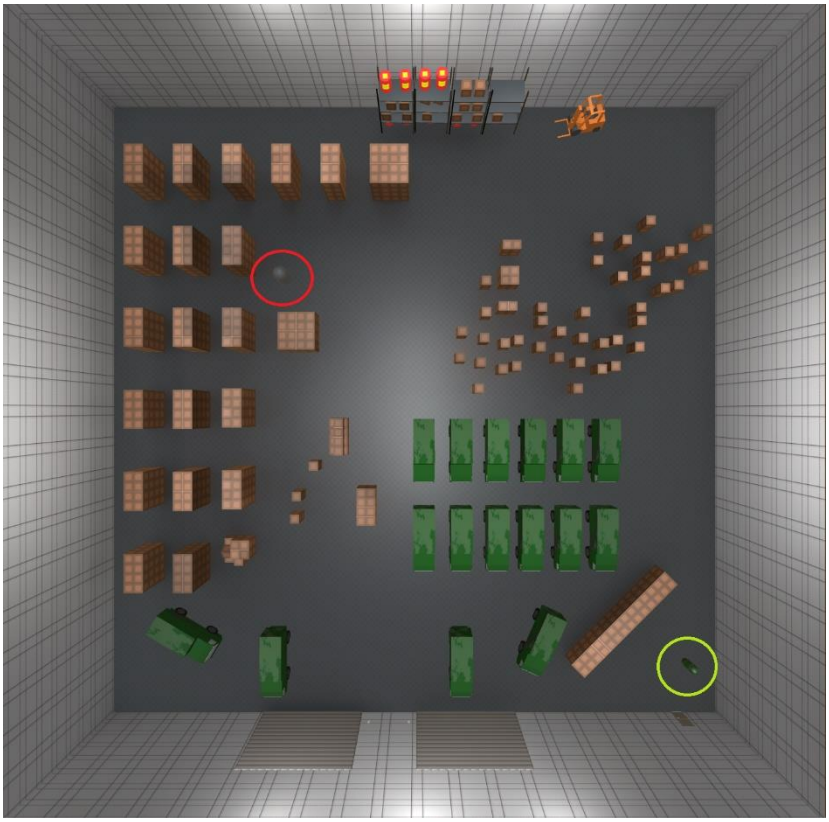
Vihollisen ampuminen luotiin Unity-pelimoottorin raycast-toiminnolla. Raycast-toiminto luo alkupisteestään suoraa viivaa eteenpäin, niin pitkälle kuin se on asetettu. Se katkeaa törmätessään esteeseen. (Physics.Raycast 2019.)

Pelin oli myös tarkoitus kirjata itsestään tuloksia tekstitiedostoon, josta käy selville pelaajan nimi, pelaajan taso, tapot, häviöt, kokonaisaika sekä kierrokset. Pelikentässä on myös koko kenttää kuvaava kamera, jolla kuvataan tekoälyn toimintaa. Näillä metodeilla seurataan tekoälyn toimintaa erityypin pelaajilla.



Kuva 6. Pelin alussa avautuva ”sopimus”.

Kuvassa 6 näkyy sopimus, johon on syötetty testattavan henkilön nimi kohtaan test persons name sekä valintaruutuun asetettu testihenkilön pelitaso. Tiedot oli tarkoitus tallentaa tekstitiedostoon.



Kuva 7. Pelin kenttää kuvaava kamera.

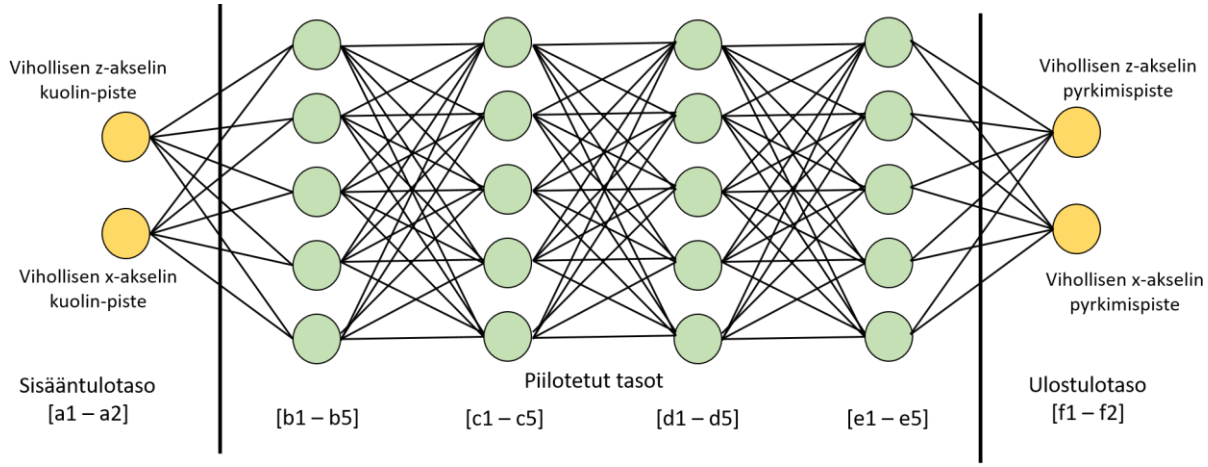
Kuva 7 on otettu peliin luodusta kamerasta joka kuvaa pelikenttää ylhäältä alaspäin. Punaisessa ympyrässä vihollisen syntymispiste ja vihreässä ympyrässä pelaajan.

Neuroverkon tasot ovat luotu matriisilla 1.

(1)

$$[a^1 a^2] \begin{bmatrix} b^1 b^2 b^3 b^4 b^5 \\ c^1 c^2 c^3 c^4 c^5 \\ d^1 d^2 d^3 d^4 d^5 \\ e^1 e^2 e^3 e^4 e^5 \end{bmatrix} [f^1 f^2]$$

jossa	*a1-a2*	sisääntulo taso
	b1-b5	ensimmäinen piilotettu taso
	c1-c5	toinen piilotettu taso
	d1-d5	kolmas piilotettu taso
	e1-e5	neljäs piilotettu taso
	f1-f2	ulostulo taso



Kuva 8. Pelissä käytetty neuroverkko visualisoituna

Kuvassa 8 näkyy peliin luotu neuroverkko. Kuvassa oleva sisääntulotaso koostuu kahdesta sisääntuloneuronista. Nämä neuronit ottavat dataa vihollisen kuolinpisteestä pelikentällä. Tämä data kulkeutuu piilotetuiden kerrosten läpi neuroverkon prosessoitua sitä ulostulotasoon. Ulostulotasossa olevat kaksi ulostuloneuronia asettavat uuden pisteen, johon vihollishahmo pyrkii.

Jos vihollinen kuolisi esimerkiksi pisteessä x = 4 ja z = 8, ottaisi kuvassa 8 näkyvä sisääntulotason alempi sisääntuloneuroni arvokseen 4 ja ylempi 8. Tämän jälkeen arvojen data kulkeutuu painoja pitkin piilotetuille neuroneille, jotka prosessoivat saatua dataa. Lopuksi prosessoitu data siirtyy kuvassa näkyvälle ulostulotason ulostuloneuroneille. Ulostuloneuronien data määrää vihollisen seuraavan kierroksen pyrkimispisteet, esimerkiksi x = 10 ja z = 20.

Taulukko 1. Eri neuroverkko matriiseilla testattuja tuloksia

Kierrokset	Matriisi	Lopputulos
100	[a1 - a2], [b1 - b3], [c1 - c2]	Vihollinen ei liikkunut millään kierroksella
100	[a1 - a2], [b1 - b10], [c1 - c2]	Vihollinen ei liikkunut millään kierroksella
50	[a1 - a2], [b1 - b3], [c1 - c3], [d1 - d2]	Vihollinen liikkui pienen matkan päähän aloituspaikasta
50	[a1 - a2], [b1 - b4], [c1 - c4], [d1 - d2]	Vihollinen liikkui useasti samoihin paikkoihin
50	[a1 - a2], [b1 - b5], [c1 - c5], [d1 - d2]	Vihollinen liikkui muutaman kymmenen kierroksen jälkeen vain yhteen paikkaan
3 x 100	[a1 - a2], [b1 - b5], [c1 - c10], [d1 - d5], [1e - 2e]	vihollinen jäi n. 2 - 5 kierroksen jälkeen paikalleen aloitus pisteeseen

100	[a1 - a2], [b1 - b5], [c1 - c5], [d1 - d5], [1e - 2e]	Vihollinen jäi n. 50 kierroksen jälkeen paikalleen
100	[a1 - a2], [b1 - b3], [c1 - c5], [d1 - d3], [1e - 2e]	Vihollinen juoksi usein oikeaan ylänurkkaan, mutta välillä muuallekin
150	[a1 - a2], [b1 - b2], [c1 - c2], [d1 - d2], [1e - 2e]	Vihollinen juoksi kahden pisteen välillä
100	[a1 - a2], [b1 - b5], [c1 - c5], [d1 - d5], [1e - 5e], [1f - 2f]	Vihollinen vaihtoi paikkaa joka kierroksella

Taulukossa 1 tulee ilmi vihollisen käyttäytymisen erot eri neuroni ja tasomäärillä. Pienemmässä tasomäärässä huomasi pienen eron, mutta lopullisesta määrästä tasojen lisääminen ei tuottanut eroa. Eron olisi voinut huomata vasta monen sadan tai jopa tuhannen kierroksen jälkeen.

4.3 Pelin ulkonäkö

Pelin ulkonäkö oli alusta asti selvillä. Avara kenttä sekä yksinkertaiset mallit olivat tärkeä osa pelin ulkonäköä ja pelattavuutta. Liika yksityiskohtien luominen olisi vienyt aikaa sekä tietokoneen kapasiteettia.

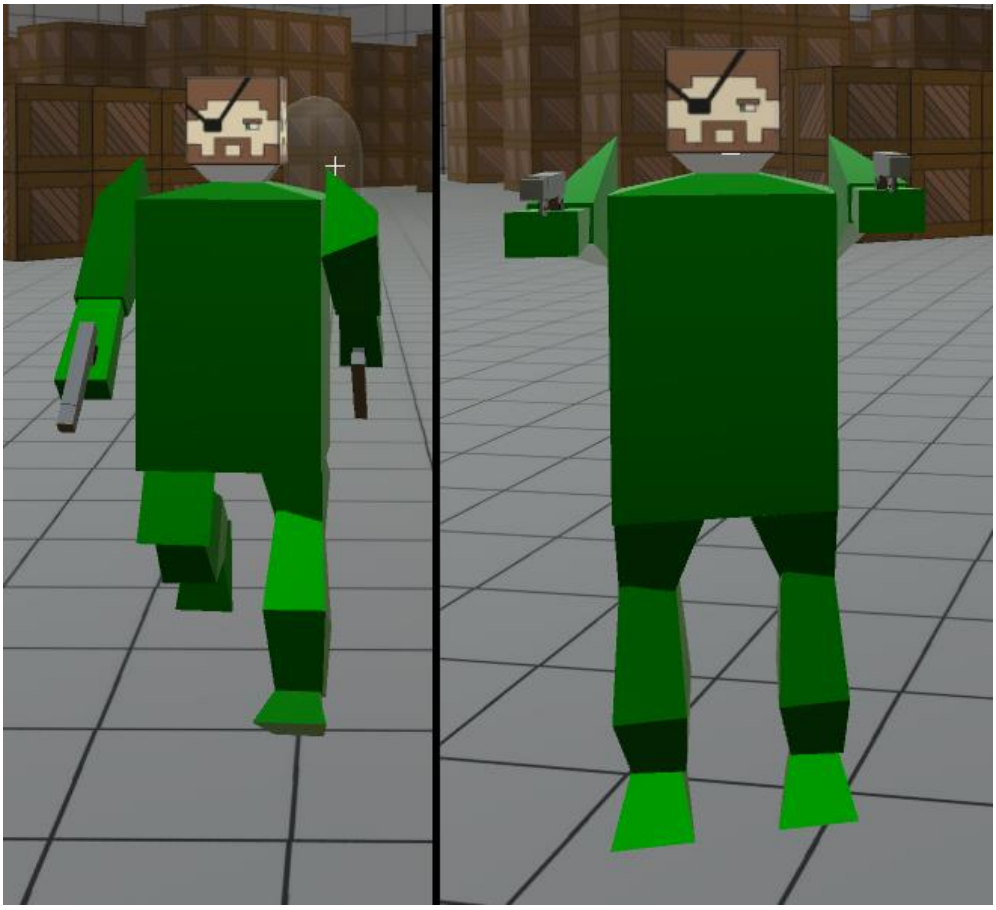
Kaikki työhön on valmistettu itse kentästä 3D-malleihin sekä tekstuureihin. Tarkoitus oli saada sarjakuvamainen, mutta selvä ulkonäkö ilman ylimääräisiä tehosteita. Muutaman pieni tehoste löytyy, mm. aseen suuliekki. Mallit ovat luotu Blender 3D -mallinnus ohjelmalla sekä tekstuurit GIMP-kuvankäsittelyohjelmalla.

Kuva 9. Peliin luodut mallit

Kuvassa 9 näkyy pelikentän esteitä. Ylemmässä kuvassa esteinä toimii hyllyjä ja trukki, kun taas alemmassa laatikoita sekä kuorma-autoja. Kuvassa näkyy myös koristeeksi luotu liukuovi sekä sille painikkeet. Kuvassa näkyvät esineet ja asiat eivät kuitenkaan ole interaktiivisia. Peli ei myöskään piirrä varjoja kevyen muistinkäytön takaamiseksi.

Kuva 10. Pelaajan näkökulma.

Kuvassa 10 näkyy aseesta tuleva suuliekki sekä vasemmassa ylänurkassa debug-infoa, josta käy selville: vihollisen syntymä- ja kuolemakerrat, vihollisen sisään- ja ulostulevien neuronien arvot, joista Input1 on vihollisen x-akselin kuolinpiste, Input2 z-akselin kuolinpiste, Output1 x-akselin pyrkimispiste sekä Output2 z-akselin pyrkimispiste. Kaksi alimmaista ovat fitness-arvo sekä koneoppimisen opetustila.



Kuva 11. Pelin vihollinen

Kuvan 11 vasemmanpuoleisessa ruudussa näkyy vihollinen, joka kävelee. Vihollisen ollessa tässä tilassa, se ei ole nähnyt pelaajaa. Oikeanpuoleisessa ruudussa vihollisella on suora katseyhteys pelaajaan, joten se ampuu.

4.4 Koodin kulku

Peliin on luotu C#-ohjelmointikielellä koodiluokkia, jotka toteuttavat pelin toiminnallisuuden. Vihollisen kannalta tärkeimmät luokat ovat NeuralNetwork, AImanager sekä enemyAI. NeuralNetwork-luokassa luodaan itse neuroverkot, joista luodaan instanssi AImanager-luokkaan. Tämän jälkeen AImanager-luokka kykenee käyttämään NeuralNetwork-luokassa luotua neuroverkkoa.

```
private int[] layers; //layers
private float[][] neurons; //neuron matrix
private float[][][] weights; //weight matrix
private float fitness; //fitness of the network
```

Kuva 12. NeuralNetwork-luokan muuttujat

Yllä olevassa kuvassa on luotuna NeuralNetwork-luokkaan integer-taulukko ”layers” (tasot), float-matriisit ”neurons” (neuronit) ja ”weights” (painot) sekä float-muuttuja ”fitness” (kunto). Jokainen näistä muuttujista on yksityisiä muuttujia, eli niihin ei pääse käsiksi muissa luokissa kuin NeuralNetwork-luokassa.

AImanager-luokka on kiinni peliobjektissa, joka ei tuhoudu pelin aikana, joten siihen on luotu funktiot, joiden on toimittava koko ajan. Kyseisessä luokassa luodaan neuroverkon tasojen koko ja määrä sekä operoidaan neuroverkkoja.

```
//Initilizes and neural network with random weights
public NeuralNetwork(int[] layers){…

//Deep copy constructor
public NeuralNetwork(NeuralNetwork copyNetwork){…

private void CopyWeights(float[][][] copyWeights){…

//Create the neuron matrix
private void InitNeurons(){…

//Create weights matrix
private void InitWeights(){…

//Feed forward this neural network with a given input array
public float[] FeedForward(float[] inputs){…

//Mutate neural network weights
public void Mutate(){…

public void AddFitness(float fit){…

public void SetFitness(float fit){…

public float GetFitness(){…

//Compare two neural networks and sort based on fitness
public int CompareTo(NeuralNetwork other){…
```

Kuva 13. NeuralNetwork-luokan funktiot

Kuvassa 13 näkyy NeuralNetwork-luokan funktiot. NeuralNetwork-luokkaan on saatu ohjeistusta YouTube-videosta. (Tutorial On Programming An Evolving Neural Network In C# w/ Unity3D 2017).

Ensimmäisenä kuvassa näkyy funktio nimeltä NeuralNetwork. Tämä funktio toimii NeuralNetwork-luokan konstruktorina eli rakentajana. Funktio ottaa parametriksi integer-taulukon ”layers” ja asettaa luokan oman ”layers”-arvon parametriin annettuun arvoon, lisäksi funktio kutsuu myös kuvassa näkyviä InitNeurons- ja InitWeights-funktioita.

Seuraavana toinen NeuralNetwork-funktio. Tämä toimii luokan kopiokonstruktorina. Sitä kutsutaan AImanager-luokassa, jossa sille annetaan kuvassa 14 näkyvästä ”NeuralNetwork”-listasta parametreja. Funktio ottaa parametrikseen saman luokan. Kopiokonstruktoin tarkoitus on luoda uusi instanssi vanhojen instanssien arvoille (Saini s.a.). Funktio kutsuu myös InitNeurons-, InitWeights- sekä CopyWeights-funktioita.

CopyWeights-funktio asettaa arvot kuvassa 12 näkyvälle ”weights”-matriisille, jotka se saa copyWeights-parametristaan. Tätä funktiota käytetään vain luokan kopiokonstruktorissa.

InitNeurons-funktio luo neuronit ”neurons”-matriisille sekä InitWeights-funktio luo painot ”weights”-matriisille. Näitä funktioita kutsutaan konstruktorissa sekä kopiokonstruktorissa.

FeedForward-funktio siirtää neuroverkon sisään tulevaa dataa eteenpäin. Lopuksi se palauttaa ulostulevan datan, jota käytetään vihollisen liikuttamiseen. FeedForward-funktiota kutsutaan vihollisen enemyAI-luokassa, jossa neuroverkon ulostulon arvo asetetaan tämän funktion palauttamaan arvoon.

Mutate-funktio muuttaa sattumanvaraisesti joitain ”weights”-muuttujia. Tässä pyritään satunnaistamaan vihollisen toimintaa, jotta löydettäisiin erilaisia ratkaisuja, jotka voisivat toimia paremmin.

Yksi tärkeimmistä neuroverkkojen funktioista on fitness-funktio, joka arvioi annetun päätöksen läheisyyttä optimiin ratkaisuun. Jokaiselle päätökselle on annettava yleisen vaatimuksen osoittava pistemäärä. Se luodaan soveltamalla Fitness-funktiota testituloksista saatuihin tuloksiin. (Mallawaarachchi 2017.)

Kuvassa 13 näkyvät ”fitness”-arvoa prosessoivat funktiot nimeltä AddFitness, SetFitness sekä GetFitness. AddFitness-funktio lisää ”fitness”-arvoon parametriin tuodun määrän, jonka se saa enemyAI-luokassa. Sen parametriksi asetetaan absoluuttinen sisääntuloarvo. SetFitness-funktio taas asettaa ”fitness”-arvon parametriin tuotuun määrään. Tämä funktio vain nollaa ”fitness”-arvon AImanager-luokassa silloin, kun vihollinen aloittaa uuden kierroksen. GetFitness-funktio palauttaa ”fitness”-arvon. Tätä funktiota käytetään vain fitness-arvon tulostamiseen enemyAI-luokassa.

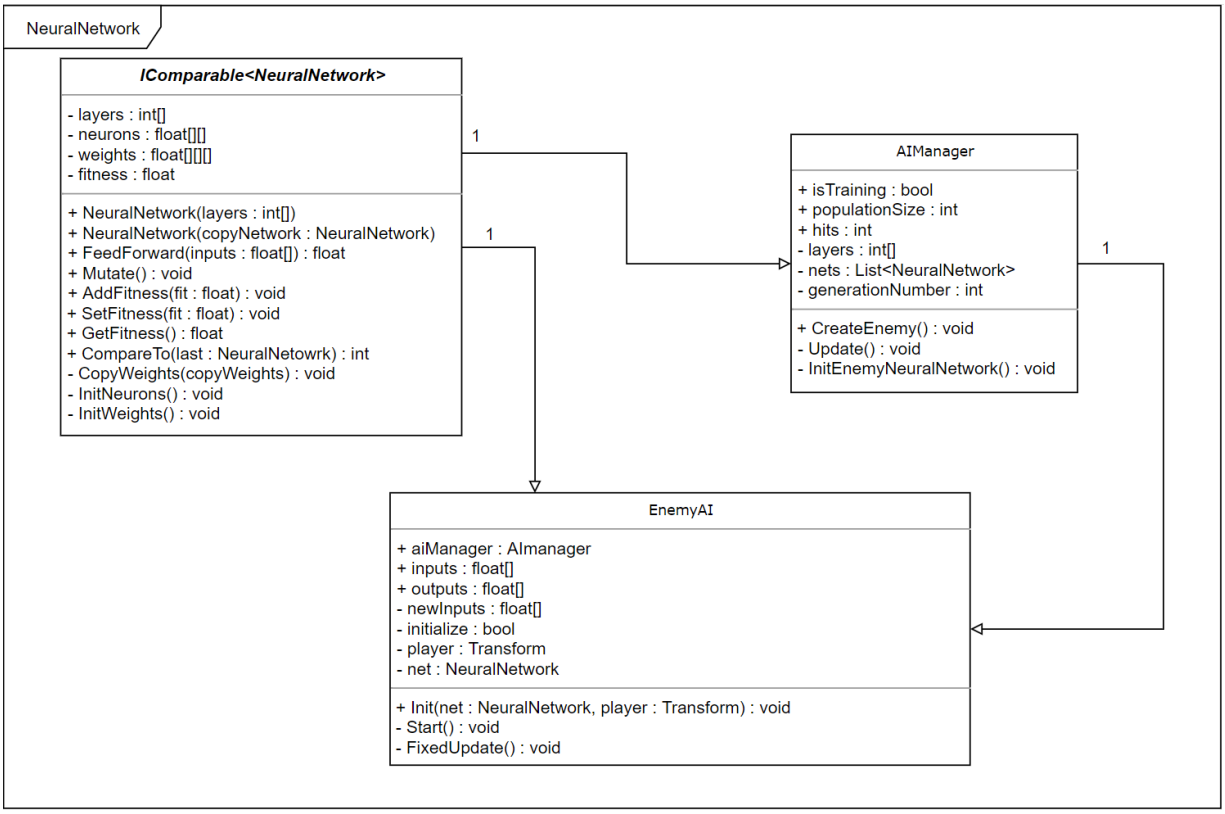
Viimeisenä CompareTo-funktio, joka vertaa aiemman kierroksen ”fitness”-arvoa uuteen. Jos uusi arvo on suurempi kuin aiempi palautetaan 1. Jos taas uusi arvo on pienempi kuin aiempi, palautetaan -1. Jos uusi ja aiempi fitness-arvo ovat samat, palautetaan 0.

```
//Neural network variables
public bool isTraining = false;
private int[] layers = new int[] {2, 5, 5, 5, 5, 2};
private List<NeuralNetwork> nets;
private int generationNumber = 0;
public int populationSize = 30;
public int hits = 0;
```

Kuva 14. AImanager-luokan neuroverkon kannalta tärkeät muuttujat

Yllä olevasta kuvassa on boolean-muuttuja ”isTraining”, joka kertoo, onko vihollinen oppimassa kyseisellä hetkellä vai ei. Seuraavana integer-taulukko ”layers” (tasot), jota operoidaan koodissa ja joka asettaa ne NeuralNetwork-luokan saman nimiseen muuttujaan. Kolmantena lista NeuralNetwork-luokasta nimeltä ”nets”, jolla päästään käsiksi NeuralNetwork-luokan neuroverkkoihin. Integer-muuttuja ”generationNumber” kasvaa aina, kun vihollinen syntyy uudestaan. Integer-muuttuja ”populationSize” alustaa päätösten määrää. Viimeisenä vihollisen kuolemakerrat laskeva integer-muuttuja ”hits”. Muuttujat ”isTraining”, ”populationSize” sekä ”hits” ovat julkisia muuttujia, koska niihin on päästävä käsiksi muissa luokissa, kuten esimerkiksi aseen Gun-luokassa, jossa vihollisen kuollessa kasvatetaan yhdellä ”hits”-muuttujaa sekä asetetaan ”isTraining”-boolean epätodeksi.

AImanager-luokan tehtävänä on ylläpitää vihollisen neuroverkoilla saatua dataa sekä prosessoida sitä. Koska luokka ei koskaan häviä pelistä sen tehtävä on myös luoda uusi vihollinen sekä neuroverkkojen tiedon tulostus pelinäkymän vasempaan ylänurkkaan.



Kuva 15. Pelin luokkakaavio neuroverkosta

Kuvassa 15 on visualisoitu peliin luotu neuroverkko ja miten EnemyAI- sekä AImanager-luokka perii NeuralNetwork-luokan.

4.5 Unity

Unity on alustariippumaton pelimoottori. Unity-pelimoottorissa ohjelmointi tapahtuu C#-ohjelmointikielellä. Unity sisältää monia pelimoottorille tyypillisiä toimintoja kuten 2D- ja 3D komponentteja, käyttöliittymäeditorin, fysiikkamoottorin sekä efektityökaluja (äänet, valot jne.) (Unity 2019: Performance by default, high-fidelity real-time graphics, and artist tools 2019.)

Tässä työssä käytetään Unity-pelimoottorin 2019.2.0f1-versiota, jolla saa rakennettua pelejä Windows- ja Linux-käyttöjärjestelmille sekä kaikille Apple-laitteille. Myös Android- ja Lumin-käyttöjärjestelmät, Samsung TV, PlayStation 4, Xbox One, WebGL sekä Facebook ovat tuettuna. (Build Settings 2019.)

Unity-pelimoottorille on myös olemassa valmiita koneoppimis lisäosia, esimerkiksi ML-agent eli Machine Learning-Agent, joka on kyseisen pelimoottorin avoimen lähdekoodin liitännäinen. ML-Agent antaa mahdollisuuden opettaa pelimekaniikkaa peliobjekteille, kuten esimerkiksi NPC-pelaajan käyttäytymistä. Opetusmenetelmiä on monia, kuten esimerkiksi neuroverkot. ML-Agent toimii niin kaksiulotteisissa, kolmiulotteisissa kuin virtuaali- ja lisätyissä todellisuuksissa. (Juliani ym. 2018.)

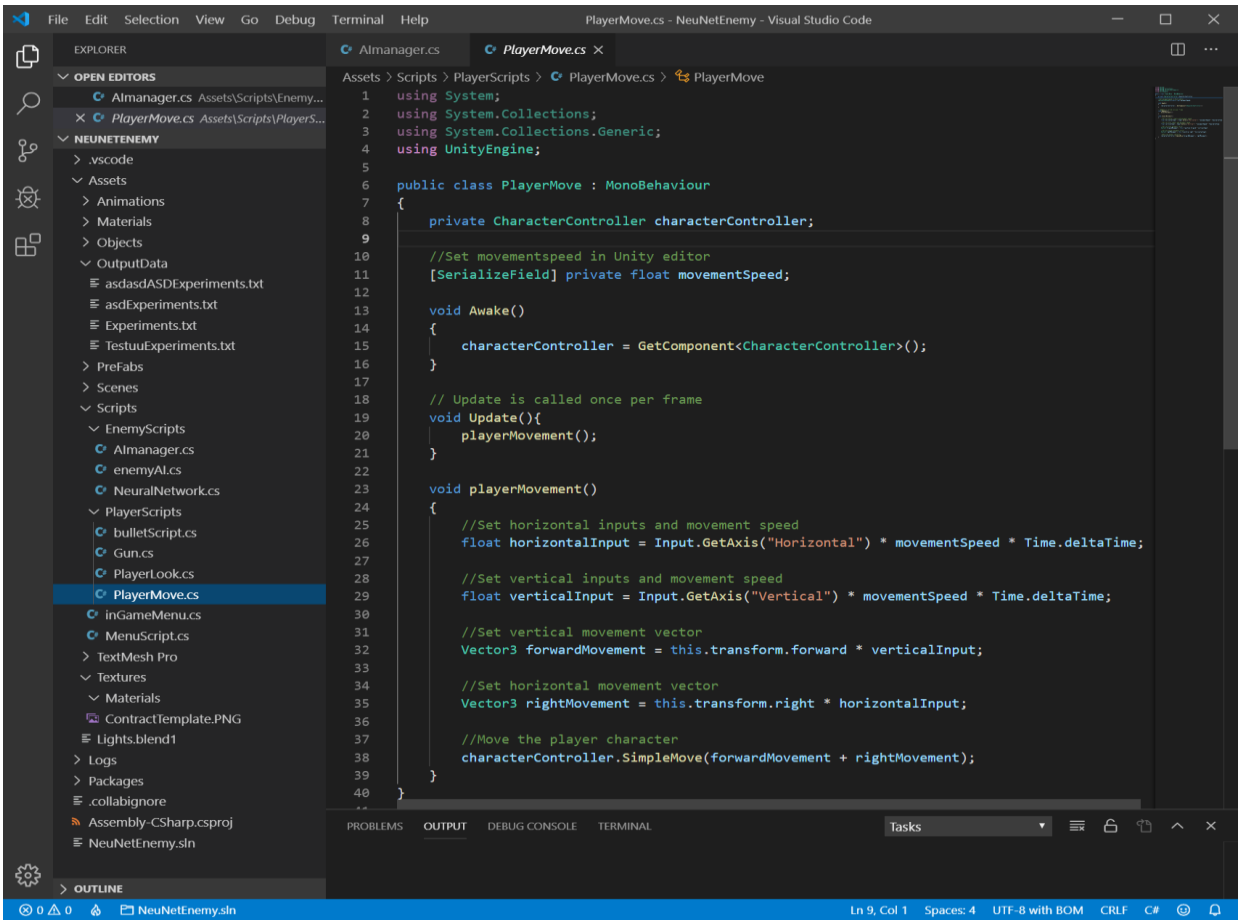
Myös ML-Agent -työkalusta otettiin selvää, siitä mihin se olisi kyennyt ja mitä kaikkea se vaati. Yksi tapa olisi ollut luoda peli ML-Agent -työkalulla. Tarkoitus kuitenkin oli neuroverkon luonti täysin tyhjästä sekä ottaa selvää mitä kaikkea se vaatii. Toinen tarkoitus oli myös seurata itseluoman neuroverkon toimintaa, joten oli päätettävä, että tätä työkalua ei käytetä.

4.6 C#

C# eli C sharp on vuonna 2000 Microsoft .NET-ohjelmistokehyksellä luotu olio-ohjelmointikieli, johon on peritty C- ja C++ -ohjelmointikielien paranneltuja ominaisuuksia. Sitä käytetään sovellusten, pelien, nettisivujen sekä tietokantojen käsittelyssä. Helpon opittavuuden ja käytettävyyden sekä suuren dokumentaation takia se on yksi käytetyimmistä ohjelmointikielistä. (C# s.a.)

Visual Studio Code on ohjelmointiin tarkoitettu lähdekoodieditori, joka tukee ohjelmoijalle tärkeitä ominaisuuksia kuten IntelliSense ja tekstin korostus**,** debugging (virheidenkorjaus) sekä Git-versionhallintatyökalu. Visual Studio Code myös laajenee monelle ohjelmointikielelle. Visual Studio Code-työkalua itseään saa myös muokattua paljon. Omille silmille sopiva tausta, tekstinkorostusväri sekä fontti on mahdollista muuttaa. (Why did we build Visual Studio Code? 2019.)

IntelliSense on useiden koodieditorien ominaisuus, joka auttaa muun muassa painovirheissä. Se myös kertoo esimerkiksi funktioiden parametrien infot sekä antaa valmiita vaihtoehtoja. (IntelliSense, 2019.)



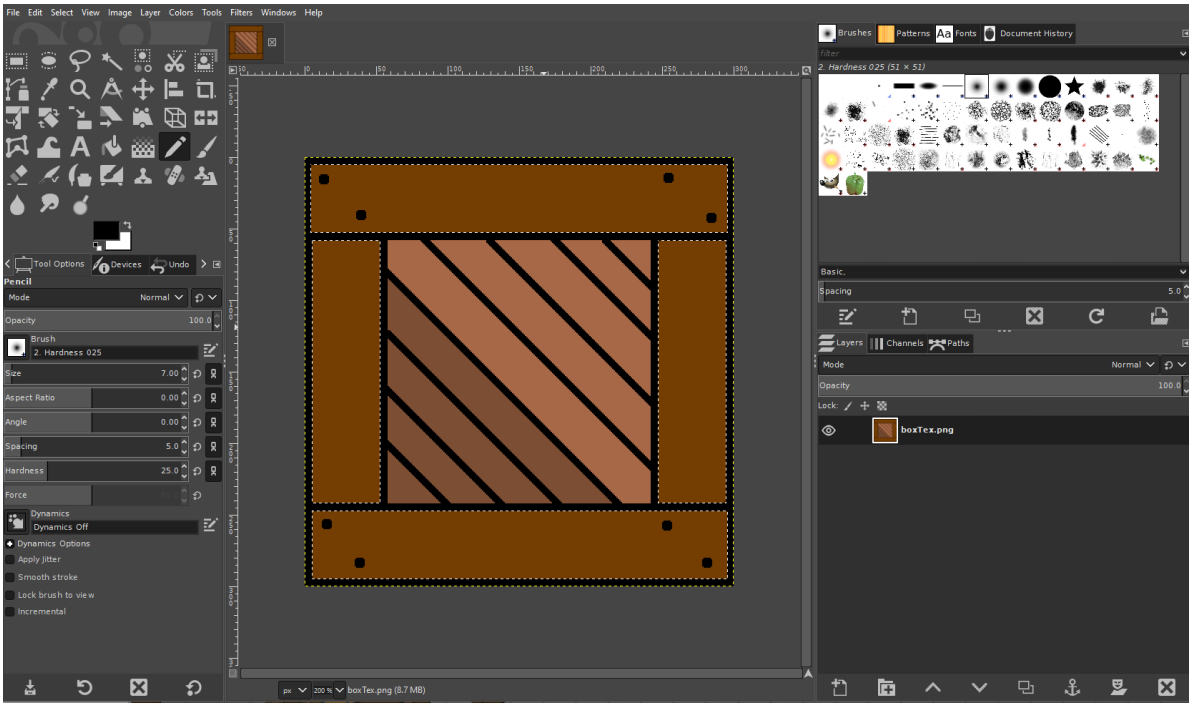
Kuva 16. Visual Studio Coden käyttöliittymä. Versio: 1.39.2

Kuvassa 16 on luotu C#-ohjelmointikielellä PlayerMove-luokka, jossa tapahtuu pelaajan liikkuminen. Olennaisin funktio on nimeltään ”playerMovement”, jossa asetetaan horizontal- sekä vertical-painikkeet. Funktiossa luodaan myös pelaajan nopeus. Kyseistä funktiota kutsutaan Update-funktiossa. Update-funktiota taas kutsutaan pelin joka framella. Kuvan vasemmassa reunassa myös lista peliin luoduista luokista.

Työssä käytetään Visual Studio Code -työkalua sen kevyen muistin käytön takia. Siitä löytyy myös ”lisäosakauppa”, josta käyttäjä saa ladattua esimerkiksi eri ohjelmointikielille, ohjelmointirajapinnoille tai pelimoottoreille suunnattuja kolmannen osapuolen luomia lisäosia.

4.7 GIMP

GIMP on kuvankäsittelyyn luotu vapaanlähdekoodin työkalu. GIMP-kuvankäsittelyohjelma tukee Linux-, OS X- sekä Windows-käyttöjärjestelmiä. GIMP lyhenne tulee sanoista GNU Image Manipulation Program. (The Free & Open Source Image Editor s.a.)

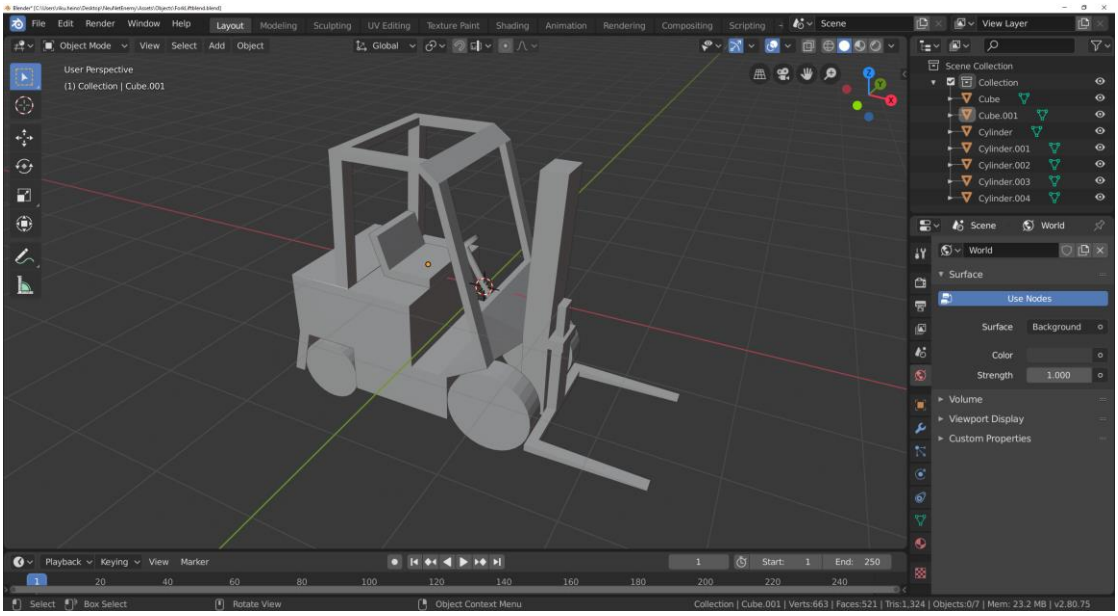


Kuva 17. GIMP-kuvankäsittely työkalun käyttöliittymä. Versio: 2.10.12

Työn graafinen osa on luotu kaikessa yksinkertaisuudessaan GIMP-kuvankäsittely työkalulla. Sillä sai nopeasti piirrettyä tarpeeksi hyvät grafiikat. Kuvassa 17 näkyy pelissä käytetyn laatikon tekstuuri.

4.8 Blender

Blender on alustariippumaton avoimenlähdekoodin 3D-mallinnusohjelma. Se tukee 3D-mallinnusta, animointia, simulointia, renderöintiä, yksityiskohtien luomista sekä liikkeen seurausta ja toimii myös videonkäsittelyyn sekä 2D-animaatioiden luomiseen. (Introduction s.a.)



Kuva 18. Blender 3D-mallinnusohjelman käyttöliittymä. Versio: 2.80

Kuvassa 18 on luotu trukki Blender 3D-mallinnusohjelmalla. Kuvan oikeassa ylänurkassa näkyy rakennepuu, josta saa selville millä osilla trukki on rakennettu.

5 YHTEENVETO

Neuroverkkojen ohjelmointi sekä dokumentaation löytäminen näinkin suuressa skaalassa oli erittäin vaikeaa. Johtuen Python-ohjelmointikielen monikäyttöisyydestä useimmat ohjeet olivat juuri kyseiselle ohjelmointikielelle luotuja. Nämä ohjeet olivat silti tyhjää parempia.

Tekoäly on kiinnostanut monia jo useita vuosikymmeniä. Sen tuomat edut ovat helpottaneet meidän elämäämme. Historian saatossa tekoälyn valmistaminen on tullut tehokkaammaksi ja läheisemmäksi kuluttajille. Kiitos tästä kuuluu monelle tutkijalle historian saatossa. Se että kuuluuko oppiva tekoäly peliin, on monen tekijän summa. Sen sopivuus on kyseenalainen pelin vihollisen roolissa. Oppivaa tekoälyä voi soveltaa peleissä muullakin tavalla kuin vihollisen tai NPC-pelaajien päätösten tekemisessä ja pelaajan liikkeiden oppimisessa. Sitä voisi soveltaa mm. suositusten muodossa pelaajalle pelaajan tehtyjen valintojen perusteella.

Kuten luvussa 3 käy ilmi koneoppiminen on vielä sen verran arvaamatonta, että siitä saisi järkevän pelikäyttöön. Myös tavallinen tekoäly on hyvin yksinkertaistettua. Monessa pelissä huomio halutaan kiinnittää muihin asioihin kuin itse tekoälyyn. Viholliset voi luoda käyttämättä hienoja ja raskaita tekoälylle ominaisia luuppeja. Vihollinen voidaan saada huijattua näyttämään viisaammalta kuin mitä se todellisuudessa on. Tämän tapainen niin sanottu kovakoodaus on nykyajan peleissä hyvin optimiratkaisu johtuen muun muassa laajasta ja näyttävästä pelimaailmasta ja sen ulkonäöstä. Luvussa 3 mainittua Quick Draw! -peliä pelatessa voi huijata koneoppimista siten, että piirtää tarpeeksi monta kertaa saman kuvan väärin. Näin koneoppimisalgoritmi saa väärää tietoa ja täten oppii väärin**.**

Työhön käytettyä aikaa kului paljon suunniteltua enemmän. Piilotetuiden kerrosten neuronien tuntematon arvo teki projektin tekemisestä sekä korjaamisesta erittäin vaikeaa ja aikaa vievää. Projektin luominen alkoi lukemalla lähteitä sekä piirtelemällä kaavioita. Eri vaihtoehtoja selvittäessä oli hyvä aloittaa ensimmäisen testiprojektin rakentaminen.

Lopullisessa versiossa oli tapaus, jossa vihollinen luultavasti oppi suojautumaan koko sen ajan mitä oli hengissä. Aina kun pelaaja käveli vihollisen luo, vihollinen pyrki silloin poispäin pelaajasta. Kuitenkin näytti siltä, ettei se tallentanut oppimaansa kuollessaan. Aivan kuin se olisi unohtanut mitä edellisellä kierroksella tapahtui. Tässä samaisessa testissä vihollinen juoksi kuvasta 12 katsottuna joko vasempaan alanurkkaan tai oikeaan ylänurkkaan.

Lopulta testasimme, josko neuroverkko toimisi oikein saadessaan sisääntulo arvoiksi vihollisen kuolema pisteen pelikentällä. Ulostulo toimisi vihollisen pisteenä, johon se pyrkisi. Näin saatiin projektista järkevämmän oloinen ja vihollinen käyttäytyi odotetulla tavalla.

Projektin lopussa tarvitsi enää testata neuroverkon neuronien sekä tasojen määrä. Peliin luotiin K-näppäimelle funktio, joka tappoi vihollisen samalla tavalla, kuin pelaaja olisi sitä ampunut. Näin säästettiin aikaa testauksessa.

Jatkoa ajatellen projektia voisi edelleen optimoida sekä esimerkiksi pelikenttää muokata. Vihollisen opetuksesta tuotettu data voitaisiin tallentaa. Peliä voisi

testata monella vihollisella niin, että pelaaja ei siirtyisi alkupaikkaan tuhottuaan vihollista. Projektiin voisi lisätä myös asetuksen, jossa pelaajana toimisi niin sanottu botti, eli peli pelaisi itsestään. Näin koneoppivaa vihollista saisi koulutettua yötä päivää.

Neuroverkkojen osalta työ aloitettiin täysin nollapisteestä. Työ antoi kuvan siitä, mitä koneoppiminen peleissä on. Se on suurimmassa osin matematiikkaa, loogista päättelykykyä sekä vähäisestä lähteiden määrästä johtuvaa kokeilua. Se antoi myös kuvan siitä, miten sitä tulee tai kannattaa käyttää. Kiinnostus tekoälyä sekä koneoppimista kohtaan kasvoi tätä projektia tehdessä. Samalla kasvoi ymmärrys ja tieto kyseisiä asioita kohtaan. Seuraavaksi voisi luoda samantyylisen työn esimerkiksi Unity-pelimoottorin omalla ML-Agent -työkalulla.

LÄHTEET

Boiko, S. 2018. Best Libraries and Tools to Start off with Machine Learning and AI. Blogi. Saatavissa: https://railsware.com/blog/best-libraries-and-tools-to-start-off-with-machine-learning-and-ai/ [viitattu: 16.11.2019].

Brownlee, J. 2016. Crash Course On Multi-Layer Perceptron Neural Networks. Blogi. Päivitetty 19.8.2019. Saatavissa: https://machinelearningmastery.com/neural-networks-crash-course/ [viitattu: 4.10.2019].

Build Settings. 2019. Unity Documentation. WWW-dokumentti. Saatavissa: https://docs.unity3d.com/2019.2/Documentation/Manual/BuildSettings.html [viitattu:21.11.2019].

C#. s.a. Technopedia. WWW-dokumentti. Saatavissa: https://www.techopedia.com/definition/26272/c-sharp [viitattu: 16.11.2019].

Evolution of machine learning. 2019. SAS. Verkkolehti. Saatavissa: https://www.sas.com/en_us/insights/analytics/machine-learning.html [viitattu: 26.10.2019].

Guinness, R. 2018. What is Artificial Intelligence? Part 2. Verkkolehti. Saatavissa: https://towardsdatascience.com/what-is-artificial-intelligence-part-2-bad0cb97e330 [viitattu: 25.11.2019].

Hulten, G. 2018. Building Intelligent System A Guide to Machine Learning Engineering. Washington: Apress.

IntelliSense. 2019. Visual Studio Code Documentation. WWW-dokumentti. Saatavissa: https://code.visualstudio.com/docs/editor/intellisense [viitattu: 21.11.2019].

Introduction. s.a. Blender. WWW-dokumentti. Saatavissa: https://docs.blender.org/manual/en/dev/getting_started/about/introduction.html [viitattu 28.10.2019].

Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M. & Lange, D. 2018. Unity: A General Platform for Intelligent Agents. WWW-dokumentti. Saatavissa: https://github.com/Unity-Technologies/ml-agents [viitattu: 16.11.2019].

Jääskeläinen, A. 2019. Mitä tapahtuu huomenna, kun tekoäly poistaa järjettömyyden. Saatavissa: https://kaakkuri.finna.fi/ [viitattu 10.12.2019].

Luo, J. s.a. An Exploration of Neural Networks Playing Video Games. Verkkolehti. Saatavissa: https://towardsdatascience.com/an-exploration-of-neural-networks-playing-video-games-3910dcee8e4a [viitattu: 13.11.2019].

Mallawaarachchi, V. 2017. How to define a Fitness Function in a Genetic Algorithm? Verkkolehti. Saatavissa: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4 [viitattu 18.11.2019].

Markoff, J. 2008. Joseph Weizenbaum, Famed Programmer, Is Dead at 85. Verkkolehti. Satavissa: https://www.nytimes.com/2008/03/13/world/europe/13weizenbaum.html [viitattu: 27.9.2019].

Mills, T. 2018. Machine Learning Vs. Artificial Intelligence: How Are They Different? Verkkolehti. Saatavissa: https://www.forbes.com/sites/forbestechcouncil/2018/07/11/machine-learning-vs-artificial-intelligence-how-are-they-different/#7ef74e103521 [viitattu: 19.9.2019].

Mohammed, M., Khan, M. & Bashier, E. 2017. Machine Learning Algorithms and Applications. 1. painos. Florida: CRC Press. Saatavissa: https://doi.org/10.1201/9781315371658 [viitattu: 10.12.2019].

Orland, K. 2013. How Forza 5 and the Xbox One use the cloud to drive machine-learning AI. Verkkolehti. Saatavissa: https://arstechnica.com/gaming/2013/10/how-forza-5-and-the-xbox-one-use-the-cloud-to-drive-machine-learning-ai/ [viitattu 16.9.2019].

Physics.Raycast. 2019. Unity Documentation. WWW-dokumentti. Saatavissa: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html [viitattu: 25.11.2019].

Robbins, M. 2014. Using Neural Networks to Control Agent Threat Response. Teoksessa Rabin, S. (toim.) GAME AI PRO. Florida: CRC Press.

Smith, C., McGuire, B., Huang, T., Yang, G. 2006. The History of Artificial Intelligence. Washington: University of Washington. PDF-dokumentti. Saatavissa: https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf [viitattu 10.12.2019].

Stanton, C. 2014. When will technology surpass the complexity and intelligence of the human brain? WWW-dokumentti. Saatavissa: https://www.quora.com/When-will-technology-surpass-the-complexity-and-intelligence-of-the-human-brain [viitattu: 20.11.2019].

Statt, N. 2019. How Artificial Intelligence Will Revolutionize the Way Video Games Are Developed And Played. WWW-dokumentti. Saatavissa: https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning?fbclid=IwAR1owmTiM7Py5D6qFRP-COn1gp3W1tBWR7QnXF01q6rWGozS36R9EgWx_g4 [viitattu: 24.11.2019].

The Free & Open Source Image Editor. s.a. GIMP. WWW-dokumentti. Saatavissa: https://www.gimp.org/ [viitattu: 15.11.2019].

Turing, S. 2012. Alan M. Turing. 1. – 2. painos. Cambridge: Cambridge University Press. Saatavissa: https://kaakkuri.finna.fi/ [viitattu 9.12.2019].

Tutorial On Programming An Evolving Neural Network In C# w/ Unity3D. 2017. YouTube. WWW-dokumentti. Saatavissa https://www.youtube.com/watch?v=Yq0SfuiOVYE&t=1459s [viitattu: 28.11.2019].

Unity 2019: Performance by default, high-fidelity real-time graphics, and artist tools. 2019. Unity. WWW-dokumentti. Saatavissa: https://unity3d.com/unity#editor [viitattu 26.10.2019].

Why did we build Visual Studio Code? 2019. Visual Studio Code Documentation. WWW-dokumentti. Saatavissa: https://code.visualstudio.com/docs/editor/whyvscode [viitattu: 16.11.2019].

Wang, L. 2019. From Intelligence Science to Intelligent Manufacturing. *Elsevier.* PDF-dokumentti. Saatavissa: https://www.sciencedirect.com/science/article/pii/S2095809919301821?via%3Dihub# [viitattu 25.11.2019].

KUVALUETTELO