Mark Romanov

VAHVISTUSOPPIMINEN PELIMOOTTORISSA

Opinnäytetyö

Tekniikan ammattikorkeakouluntutkinto

Tieto- ja viestintätekniikka

2022

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Insinööri (AMK)
Tekijä	Mark Romanov
Työn nimi	Vahvistusoppiminen pelimoottorissa
Toimeksiantaja	Gamelab
Vuosi	2022
Sivut	28 sivua, liitteitä 0 sivua
Työn ohjaaja	Pekka Vilpponen

TIIVISTELMÄ

Tämä opinnäytetyö käsittelee itseoppivaa tekoälyä, koneoppimista ja tekoälyn työkaluja pelimoottoreissa sekä niiden käyttämistapoja. Työ käsittelee myös vahvistusoppimisen teoriaa ja sen käytäntöä pelialalla.

Työn lähtökohtana toimii kirjoittajan mielenkiinto aiheeseen sekä toimeksiantajana toimivan Kaakkois-Suomen ammattikorkeakoulun Gamelab-oppimisympäristön kiinnostus siitä, miten vahvistusoppimista voisi käyttää pelimoottorien kanssa.

Opinnäytetyön aihe on keskitetty tekoälyn toteutukseen pelimoottoreissa. Pelimoottorien tekoälyn kehitykseen pääsemiseksi ensin on käsiteltävä aiheeseen liittyvää teoriaa.

Työssä perehdytään neuroverkkojen ja vahvistusoppimisen algoritmin teoriaan sekä tekoälyn toteutukseen pelimoottorissa. Työ on toteutettu Unity Engine pelimoottoria käyttäen. Työn ohjelmointikielenä toimi C#-ohjelmointikieli.

Opinnäytetyössä esitellään koneoppimisalgoritmeja ja erityisesti vahvistusoppimista, koska tätä algoritmityyppiä käytetään useimmiten pelimoottoreissa. Työssä tuodaan esiin myös jatkokehitysehdotuksia, joita työn pohjalta voisi kehittää.

Asiasanat: tekoäly, peliohjelmointi, neuroverkko, koneoppiminen

South-Eastern Finland University of Applied Sciences

Degree	Bachelor of Engineering
Author	Marl Romanov
Thesis title	Reinforcement Learning in a Game Engine
Commissioned by	Gamelab
Time	November 2022
Pages	28 pages, 0 pages of appendices
Supervisor	Pekka Vilpponen

ABSTRACT

This thesis process provides an insight into self-learning artificial intelligence, machine learning and artificial intelligence tools in game engines and the ways they are used. The work also process with the theory of reinforcement learning and it's practice in the gaming industry.

The idea for this work is the author's interest in the topic and the interest of South-Eastern Finland University of Applied Sciences how reinforcement learning could be used with game engines. The topic of the thesis is focused on the implementation of artificial intelligence in game engines. Before going into development of artificial intelligence in game engines, need to understand the theory of artificial intelligence.

The work introduces a theory of neural networks and reinforcement learning algorithm, as well as the implementation of the artificial intelligence in game engine. The project was created in Unity-game engine using C# programming language.

The thesis presented machine learning algorithms and especially reinforcement learning, because this type of algorithm is mostly used in game engines. The work also brings forward further development proposals that could be developed based on the work.

Keywords: machine learning, reinforcement learning, neural networks

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena on kehittää Kaakkois-Suomen ammattikorkeakoulun tieto- ja viestintätekniikan Gamelab-oppimisympäristölle tekoäly, joka perustuu vahvistusoppimisen tekniikkaan. Työn tarkoitus on tutkia vahvistusoppimisen luomisen periaatteita ja luoda itseohjautuva auto, joka toimii ympäristön mukaisesti ja etsii vapaata paikkaa parkkipaikalla. Projekti simuloi tekoälyjen toimintoja, jota käytetään esimerkiksi robottipölynimurissa ja automaattisessa auton pysäköinnissä.

Työn alussa selvitetään tarvittava tekoälyn teoria ja erilaiset koneoppimisvaihtoehdot, sekä mitä niistä käytetään pelialalla. Tämän jälkeen tarkistetaan olemassa olevia pelimoottoreiden työkaluja. Selvitystyön perusteella valitaan koneoppimisen menetelmä, sekä toteutettiin projekti käyttämällä sopiva työkalu. Toinen osa käsittelee vahvistusoppimisen toteutuksen prosesseja sekä koulutusympäristöjen tuloksia.

Opinnäytetyö käsittelee tekoälyn viitekehystä ja niiden käyttötapoja pelimoottorien työkaluina. Työ käsittelee myös koneoppimisen toteutusta Unity-pelimoottorissa ja tekoälyn koulutuksen dataa käyttämällä TensorFlow-tekoälykirjasto.

2 TUTKIMUSASETELMA

Tutkimustehtävänä on tutkia erilaisia koneoppimisen tekniikoita, niiden käyttämistapoja ja kehittää projekti pelimoottorille sopivalla tekniikalla. Tutkimus rajoitetiin yhteen koneoppimista käyttävään toteutukseen, koska tarkoitus, miten koneoppiminen luodaan.

Tutkimusote jakautuu kahteen osaan. Alkuosassa selvitetään tekoälyn teoria, koneoppimisen algoritmit ja mitä niistä käytetään pelialalla. Toisessa osassa toteutetaan projekti, jossa käytetään pelimoottorin työkalua tekoälyn toteutukseen. Selvitystyön tuloksena saadaan koulutuksen dataa ja käsitellään sitä tekoälyn toimintatavoitteiden mukaan.

Aineistona toimii tekoälyn ja vahvistusoppimisen teorian kirjat, kurssien opetusmateriaali ja työkalun dokumentaatio sekä työkalun omat esimerkit.

2.1 Tutkimusongelma ja -kysymykset

Tutkimusongelmana on selvittää tekoälyn toteutuksen periaatteet sekä ML-Agents-työkalun käyttämistapoja ja mahdollisuuksia. Edellä mainitusta ongelmanasettelusta on johdettu seuraavat tutkimuskysymykset:

1. Mitä työkaluja on olemassa tekoälyn luomiseen pelimoottorissa?
2. Miten optimoidaan agenttikoulutus?
3. Miten käsitellään agenttikoulutuksen dataa?

Tutkimuskysymysten lisäksi aihe vaatii syvää tekoälyn ja vahvistusoppimisen tuntemusta.

2.2 Aikaisemmat opinnäytetyöt ja tutkimukset

Tekoälyn toteutus pelimoottorissa ei ole uusi asia. Theseuksesta löytyy vastaavanlaisia opinnäytetöitä, jotka käsittelevät tekoälyn teoria ja niiden toteutusta pelimoottorissa. Esimerkiksi Tiia Sarell käsittelee luonnollisen kielen prosessointia opinnäytetyössään (2021) ”Luonnollisen kielen prosessointi ja sen hyödyntäminen peliohjelmoinnin koulutuksessa”, jossa on hyvin avattu tekoälyn teoriaa ja koneoppimisen algoritmeja. Myös Teemu Ropilo käsittelee opinnäytetyössä (2019) ”Teaching a machine learning agent to survive in a 2D topdown environment” Unity Engine -pelimoottorin ML-Agents-työkalua.

3 TEKOÄLY

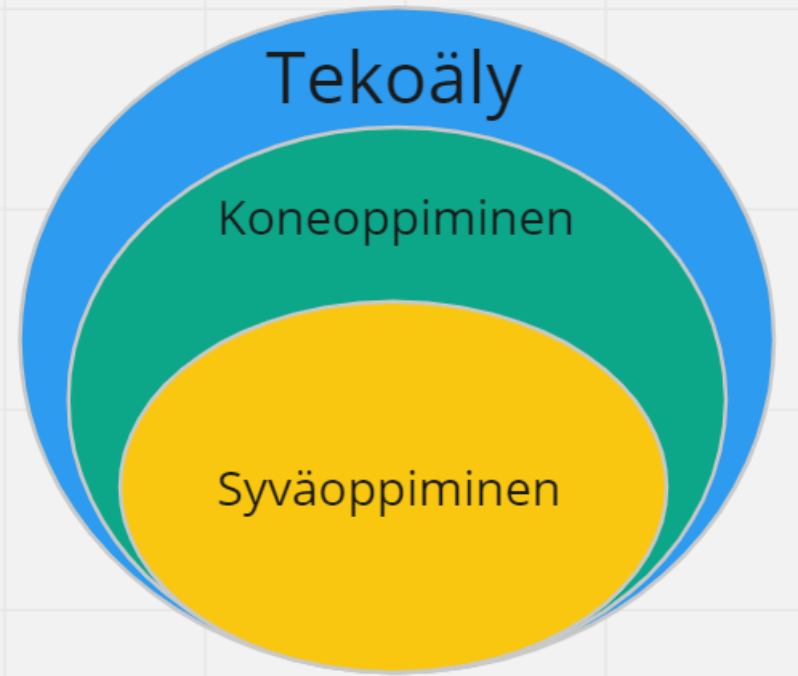
Termillä tekoäly ei ole yhtä oikeaa määritelmää, sillä se viittaa suureen määrään erilaisia käsitteitä. Tekoälyllä itsessään ei ole tietoisuutta eikä älyä. Älykkäiksi katsotaan kuitenkin tekoälyn suorittamat tehtävät, kuten matemaattiset tehtävät. Tekoäly on tietokoneohjelma, joka toimii älykkäästi ja automaattisesti. Älykäs toiminto sisältää havainnon, päättelyn, oppimisen, kommunikoinnin ja toimimisen monimutkaisissa ympäristöissä. (Winston 1992,5)

Tietojenkäsittelytieteilijä Nils John Nilsson kuvailee tekoälyä seuraavasti: Tekoäly (engl. *AI*), joka on laajasti ja hieman ympäripyöreästi määritelty, koskee älykästä käyttäytymistä esineissä. Älykäs käyttäytyminen puolestaan sisältää havainnon, päättelyn, oppimisen, kommunikoinnin ja toimimisen monimutkaisissa ympäristöissä (Nilsson 1998,1). Nilssonin kollega Patrick Winston esittää toisen määritelmän. Tekoäly on tutkimus laskelmista, jotka mahdollistavat havaitsemisen, päättelyn ja toiminnan. (Winston 1992,5)

Molemmissa määritelmissä voidaan nähdä samat kohdat: havaitseminen ja toiminnallisuus. Nämä kohdat ovat perusasioita, joiden perusteella määritellään ja käytetään tekoälyä.

Tavoitteiden osalta tekoäly yleensä jaetaan teknisiin ja tieteellisiin tavoitteisiin. Tekoälyn insinööritavoitteena on ratkaista todellisia ongelmia käyttämällä tekoälyä ideoiden työkaluna tiedon edustamiseen, tiedon käyttämiseen ja järjestelmien kokoamiseen. Tekoälyn tieteellinen tavoite on määrittää, mitkä ajatukset tiedon esittämisestä, tiedon käyttämisestä ja järjestelmien kokoamisesta selittävät erilaista älykkyyttä (Winston 1992, 6).

Käsiteltäessä tekoälyä puhutaan yleensä vielä koneoppimisesta ja syväoppimisesta. Jokainen niistä on laaja ja monimutkainen asia, joka sisältää paljon omia vivahteita.

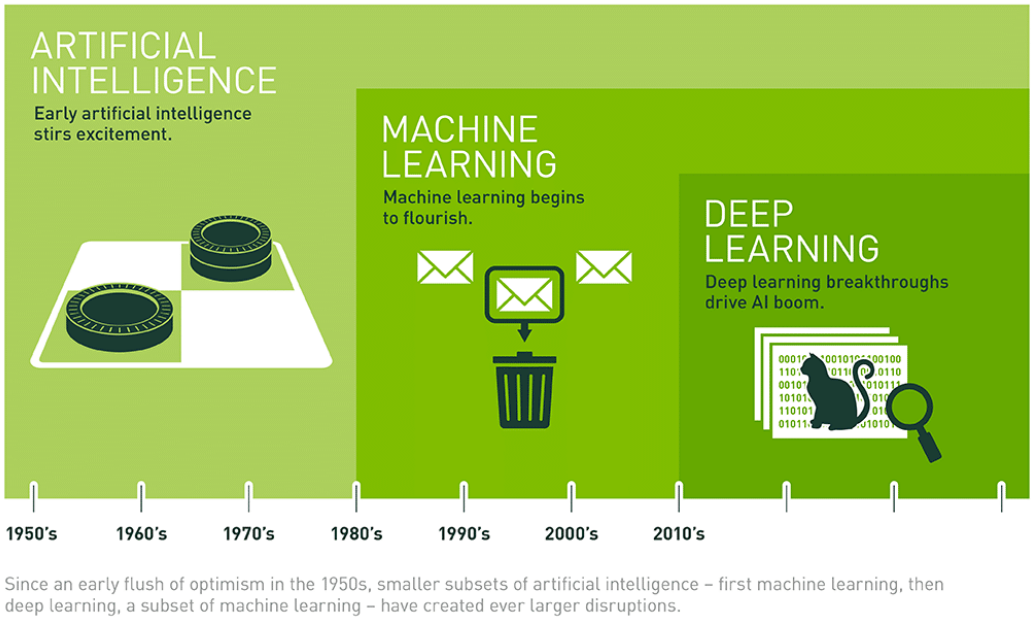


Kuva 1. Tekoälyn termien suhteet (IESC 2018).

Tältä näiden termien riippuvuus näyttää (kuva 1) toisistaan:

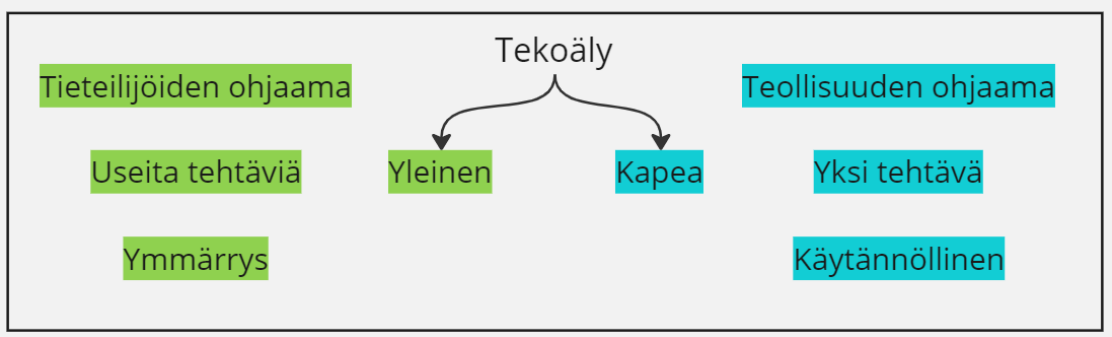
- Tekoäly: Suurempi ympyrä, idea, joka syntyi ensimmäisenä tällä alalla
- Koneoppiminen: Keskiympyrä, termi on kehitetty tekoälyn jälkeen
- Syväoppiminen: Pienin ympyrä, joka on kasvava tekoälyn osa tällä hetkellä

Nvidian visuaalinen esitys havainnollistaa asiaa hyvin. Se tarjoaa hyvän lähtökohdan tekoälyn, koneoppimisen ja syväoppimisen erojen ymmärtämiselle (Artificial Intelligence, Machine Learning, and Deep Learning 2018).



Kuva 2. Nvidian tekoälyn kehittäminen ajan kuluessa (Copeland 2016).

Alkuperäisesti tekoälyn konseptia kutsuttiin nimellä *General AI* eli yleinen tekoäly, jolla on kaikki ihmisen aistit ja ideat. Tavoitteena oli rakentaa kone, joka ajattelee kuten ihmiset. Nykyinen teknologian kehitys ei mahdollista luomaan yleistä tekoälyä. Voidaan kuitenkin luoda koneita, joita kutsutaan kapeaa tekoälyä hyödyntäviksi (engl. *Narrow AI*). Ne voivat suorittaa tiettyjä tehtäviä ihmisen ihmistasolla (Artificial Intelligence 2018).



Kuva 3. Yleisen ja kapean tekoälyn tarkoitus (IESC 2018).

Koneoppiminen tarkoittaa tekoälyn osa-aluetta, jossa tekoäly oppii itsenäisesti annetun datan perusteella. Koneoppimisessa on erilaisia koneoppimisalgoritmeja, jotka sopivat erilaisiin tehtäviin.

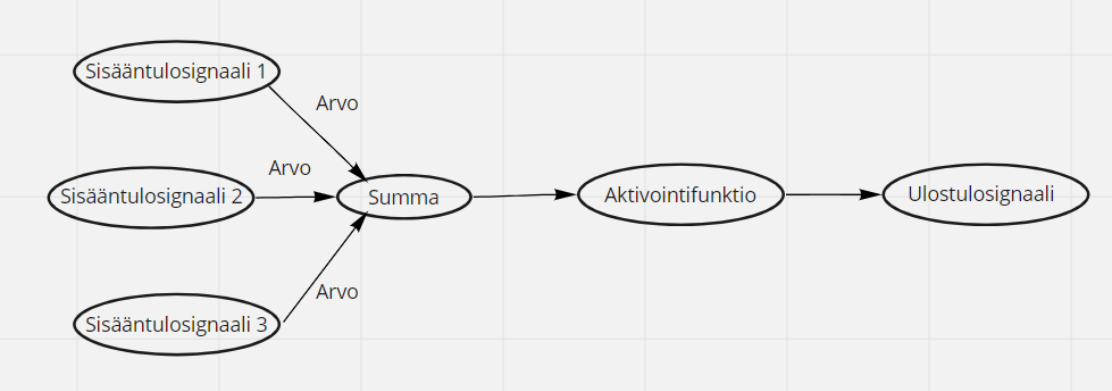
Koneoppiminen on pohjimmiltaan käytäntö käyttää algoritmeja datan jäsentämiseen. Datan jäsentämisestä päästään oppimiseen, päätöksentekoon ja lopulta ennusteiden tekoon. Sen sijaan, että kone koodataan käsin ohjelmistorutiineja käyttäen ja tietyillä ohjeilla tietyn tehtävän suorittamiseksi, kone koulutetaan käyttämällä suuria tietomääriä ja algoritmeja, jotka antavat sille mahdollisuuden oppia suorittamaan tehtävä (Michael Copeland 2016).

Syväoppiminen on uusi koneoppimisen tutkimusalue, joka on otettu käyttöön tavoitteena siirtää koneoppiminen lähemmäksi alkuperäistä tekoälyn tavoitetta. (Deep Learning 2018) Syväoppimisessa käytetään neuroverkkoa, jossa on erilaisia neuroverkkokerroksia, joilla jokaisella on erilainen tarkoitus.

Neuroverkko

Neuroverkko on tietojen käsittelyä, joka perustuu yhdistävään laskentaan. Neuroverkon idea perustui biologisten neuronien ominaisuuksiin, kuten ihmisaivossa. Datan siirtymässä käytetään neuroneja.

Neuroni ei tee mitään, ellei sen kaikkien syötteiden kollektiivinen vaikutus saavuta kynnystasoa. Aina kun tämä kynnystaso saavutetaan, neuroni tuottaa täyden voimakkaan ulostulon kapean pulssin muodossa, joka etenee solurungosta alas aksonia pitkin ja aksonin haaroihin (Winston 1992, 43).



Kuva 4. Neuronin toiminnon esimerkki (Winston 1992, 43).

Neuronissa on sisääntulosignaali, aktivointifunktio ja ulostulo. Jokainen sisääntulosignaali lähettää oman arvon tai painon ja kuin signaali on riittävän vahva, neuroni aktivoituu. Aktivointifunktio on matemaattinen funktio, joka

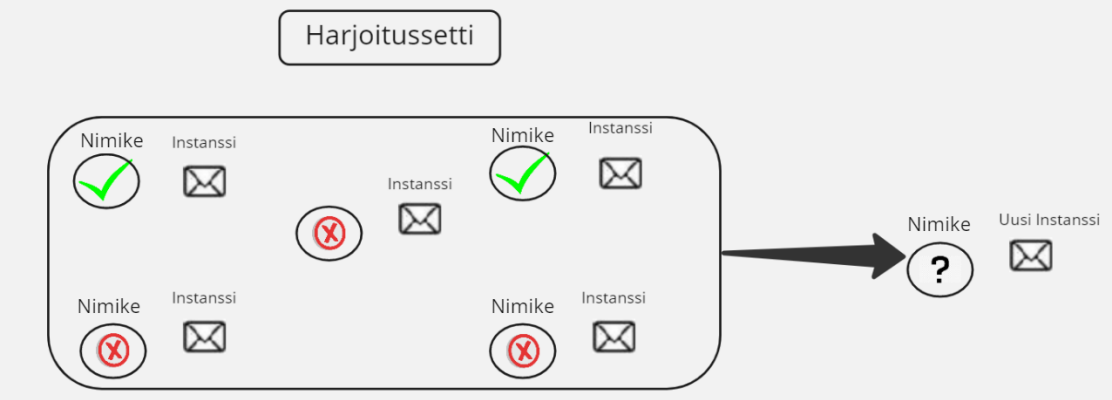
käsittelee sisääntulosignaalia. Aktivointifunktio lähettää ulostulosignaalin eteenpäin toisille neuroneille ja niin luodaan neuroverkko.

Neuroverkon koulutus sisältää vain oikeiden painojen etsimisen. Painojen arvoja muutetaan pienillä askeleilla, kunnes neuroverkko toimii halutulla tavalla.

4 KONEOPPIMISALGORITMIT

4.1 Ohjattu oppiminen

Ohjattu oppiminen on algoritmi, jossa opetusdata tiedetään ennen ulostuloa. Järjestelmä pystyy ennustamaan tulevia tuloksia menneiden tietojen perusteella. Ohjatussa oppimisessa käytetään regressio- ja luokittelutekniikoita (engl. *regression and classification*). Tätä algoritmia käytetään esimerkiksi kuvien luokituksessa ja tietojen rinnakkaisriippuvuuksien etsimisessä.



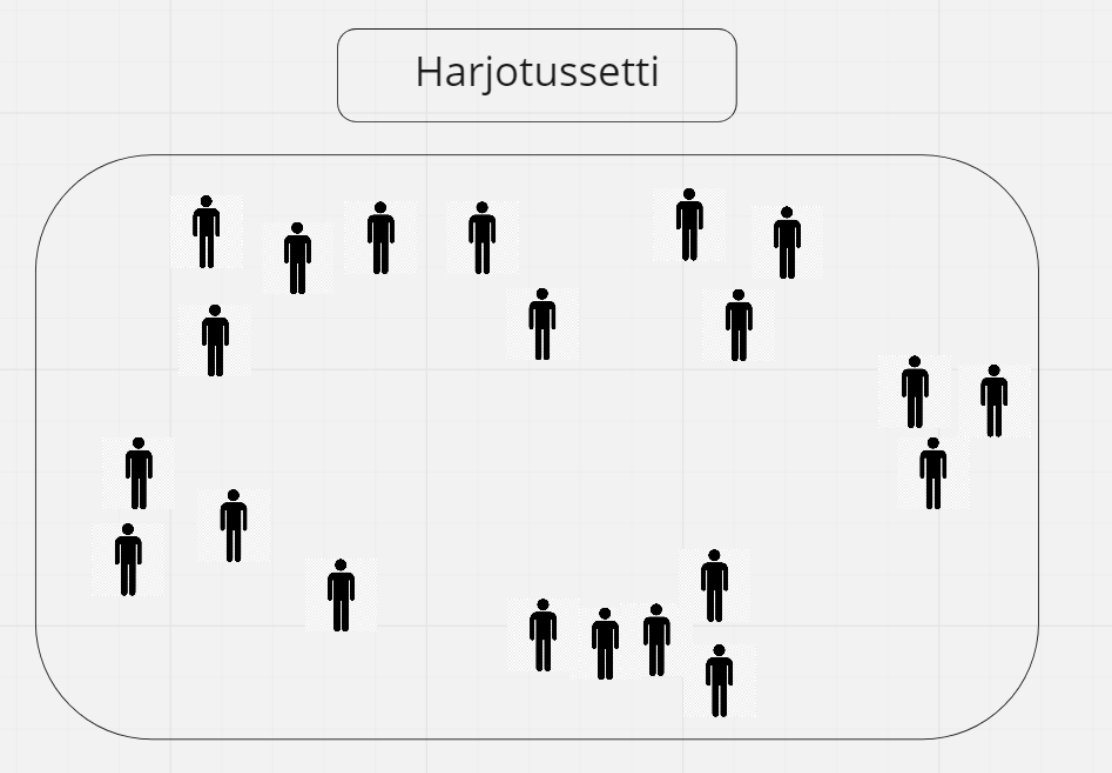
Kuva 5. Ohjattu oppiminen esimerkki (Russell 2018,14).

Tämän tyyppisessä koneoppimisjärjestelmässä tietoja, jotka syötetään algoritmiin halutun ratkaisun kanssa, kutsutaan nimikkeiksi (engl. *label*). Ohjattu oppiminen ryhmittelee luokittelutehtävät. Yllä oleva ohjelma (kuva 8) on hyvä esimerkki tästä, koska sitä on koulutettu useiden sähköpostien avulla, joissa lähtötiedossa on asetettuja nimikeitä ja niiden mukaan luokitellaan tuntemattomia sähköposteja (Russell 2018,14).

4.2 Ohjaamaton oppiminen

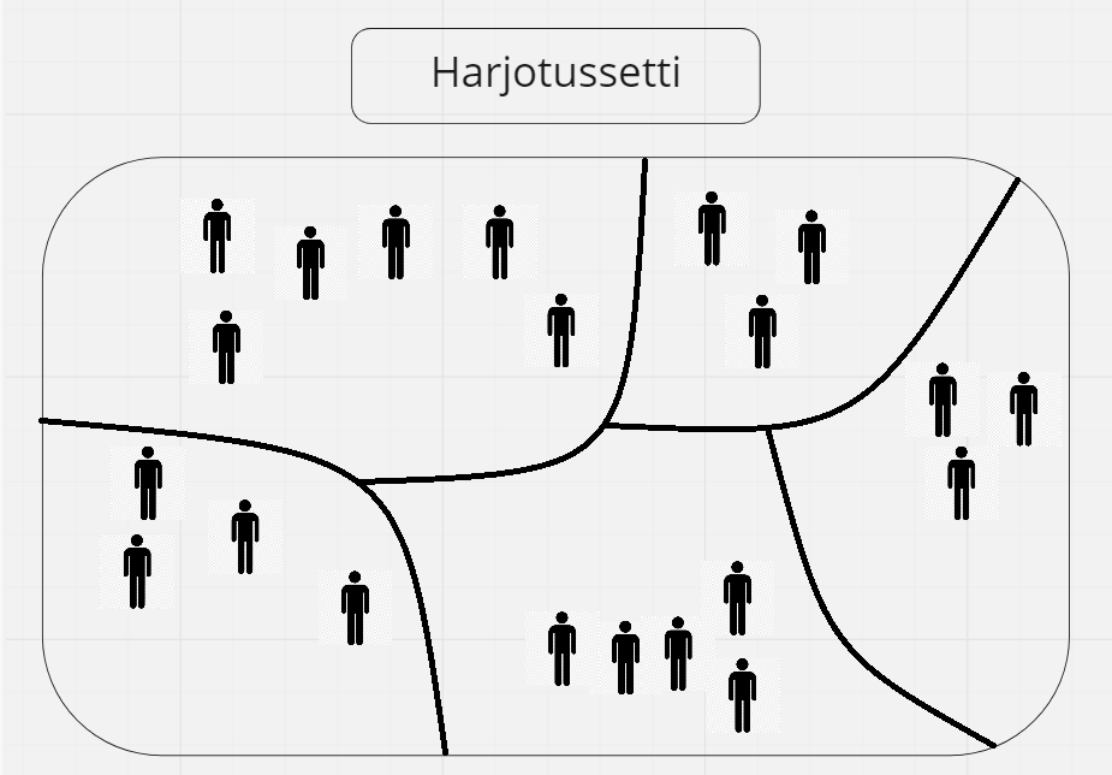
Ohjaamattomassa oppimisessa ei ole opetusdataa alussa ja sitä saadaan prosessin aikana. Järjestelmä pystyy tunnistamaan piilotetut mallit tai trendit

syöttötiedoista, jossa ei ole annettu lähtötietoja tai nimikkeitä. Tiedot järjestetään mallien mukaan, jolloin yhtäläisyydet ja poikkeavuudet tulevat yhä selvemmiksi. Ohjaamattomassa oppimisessa käytetään ryhmittymistä (engl. *clustering),* yhtymistä (engl. *association)* ja poikkeamien havaitsemista.



Kuva 6.Ohjamaton oppiminen. Massadatan lähtötietojen esimerkki (Russell 2018).

Tätä algoritmia käytetään esimerkiksi ostajien ryhmittelyssä ostohistorian ja selaushistorian perusteella.



Kuva 7. Ohjamaton oppiminen. Massadatan esimerkki ryhmittymisen jälkeen (Russell 2018)

Kuvassa (kuva 6) on esitetty asiakasdatan esimerkki. Algoritmin käyttö auttaa jakamaan (kuva 7) asiakkaat ryhmiin annettujen tietojen perusteella.

4.3 Vahvistusoppiminen

Vahvistusoppiminen on koneoppimisen algoritmi, jossa opetusdataa saadaan vuorovaikutuksesta ympäristön kanssa. Se on tavoitteellista oppimista, jossa oppijalle ei opeteta, mitä toimia hänen tulee tehdä. Sen sijaan oppija oppii tekojensa seurauksista (Ravichandiran 2018, 6). Oppimisobjektia kutsutaan agentiksi.

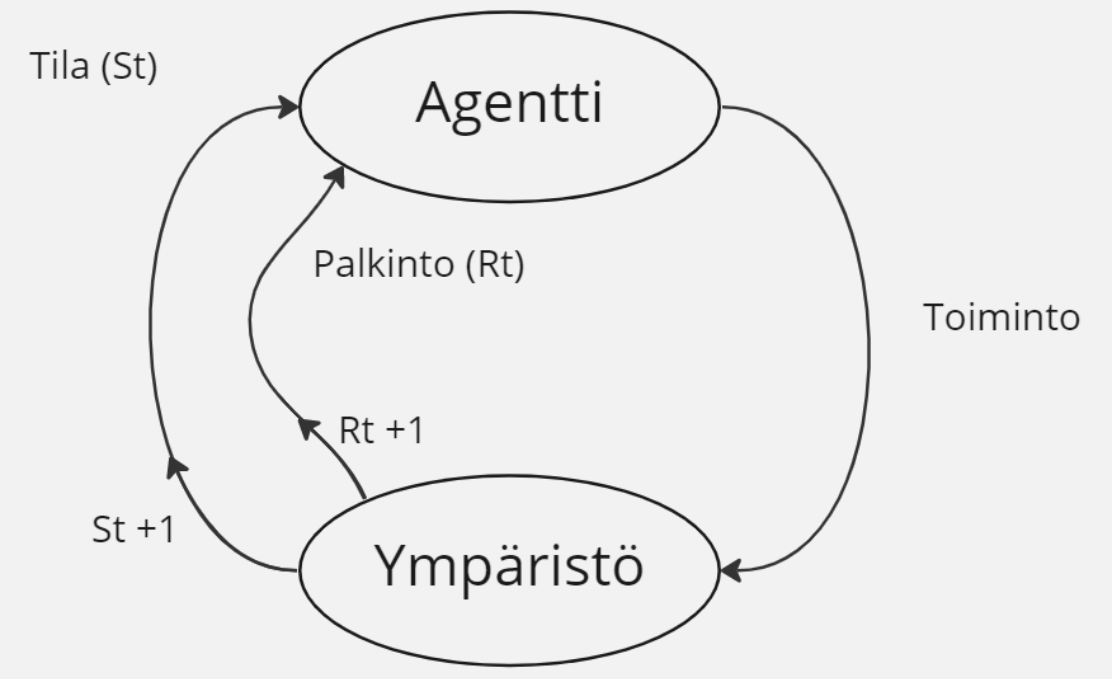
Agentit ovat ohjelmistoja, jotka tekevät älykkäitä päätöksiä ja ne ovat periaatteessa oppilaita vahvistusoppimisessa. Agentit toimivat vuorovaikutuksessa ympäristön kanssa ja ne saavat palkintoja toimiensa perusteella.

Agentin toiminto ympäristössä määrittyy käytännön (engl. *policy function)* kautta. Agentin päätös, minkä toiminnon se suorittaa, riippuu käytännöstä. Käytäntö voi olla hakutaulukon tai monimutkaisen hakuprosessin muodossa (Ravichandiran 2018, 10).

Agentin toiminnon jälkeen ympäristö antaa palautteen, joka voi olla positiivinen tai negatiivinen. Sen perusteella agentti alkaa suorittamaan toimia, jotka saivat positiivisen palautteen ja ohittaa negatiivisen palautteen. Kyseessä on yritys-erehdys-oppiminen.

Arvofunktio (engl. *value function*) osoittaa, kuinka hyvä agentin tila on tietyssä tilassa. Arvofunktio riippuu käytännöstä, ja se on yhtä suuri kuin agentin odotettu kokonaispalkinto alkuperäisestä tilasta alkaen. Arvofunktioita voi olla useita.

Agentin esitystä ympäristöstä kutsutaan malliksi (engl. *model*). Oppimista voi olla kahta tyyppiä: mallipohjaista oppimista ja mallitonta oppimista. Mallipohjaisessa oppimisessa agentti hyödyntää aiemmin oppimaansa tietoa suorittaakseen tehtävän. Mallittomassa oppimisessa agentti vain luottaa yrityksen ja erehdyksen kokemukseen oikean toiminnan suorittamiseksi.



Kuva 8. Vahvistusoppimisen algoritmin vaiheet (Ravichandiran 2018).

Tyypillisen vahvistusoppimisen algoritmin vaiheet ovat:

1. Agentti tekee havaintoja ja on vuorovaikutuksessa ympäristön kanssa.
2. Agentti suorittaa toiminnon ja siirtyy yhdestä tilasta (engl. *state*) toiseen.
3. Agentti saa palautteen suorittamansa toiminnon perusteella.
4. Palautteen perusteella agentti päättyy, oliko toiminta hyvä vai huono.
5. Jos agentti sai positiivisen palautteen, agentti mieluummin suorittaa kyseisien toiminnon, tai agentti yrittää suorittaa toisen toiminnon, joka antoi positiivisen palautteen.

5 TEKOÄLYN TYÖKALUT PELIMOOTTOREISSA

Pelialalla Unity- ja Unreal Engine -pelimoottorit ovat kehittyneimpiä ja suosituimpia monialustaisia pelimoottoreita. Niille on kehitetty monia erilaisia laajennuksia, myös tekoälyn kehittämiseen tarkoitettuja. Nämä laajennukset, kuten monet tekoälyprojektit, ovat avoimen lähdekoodin projekteja. Pelinkehittäjille tällaisten laajennusten pääasiallinen käyttötarkoitus voi olla ei-pelaajahahmo ohjaaminen muuttuvassa ympäristössä.

5.1 Unreal Engine ja MindMaker

MindMaker AI- liitännäisohjelma on avoimen lähdekoodin laajennus, jonka avulla Unreal Engine 4- ja Unreal Engine 5 pelit ja -simulaatiot voivat toimia OpenAI Gym -ympäristöinä autonomisten koneoppimisagenttien kouluttamiseen. (Krumins ym 2022.)

Laajennus helpottaa verkkoyhteyttä Unreal Enginen oppimisympäristön ja Pythonin ML-kirjaston välillä, joka vastaanottaa koulutustiedot Unreal Enginestä ja jäsentää OpenAI Gym -ympäristössä agentin kouluttamiseksi. Soveltamismahdollisuudet sisältävät tieteellisiä ja teknisiä tehtäviä. Laajennusta voidaan käyttää robottisimulaatioihin, autonomiseen ajamiseen, prosessigrafiikkaan ja muihin tehtäviin.

Laajennusta hallitaan pelimoottorin sisällä Blueprints-tekniikan avulla. Blueprints on visuaalinen komentosarjajärjestelmä, joka perustuu käyttöliittymien käyttöön pelielementtien luomiseen. (Introduction to Blueprints s.a.) Tämä

helpottaa laajennuksen käyttämistä käyttäjille, jotka eivät tunne C++-ohjelmointikieltä. MindMaker tarjoaa mahdollisuuden valita useista määrästä vahvistusoppimisien algoritmeja, joita käytetään agentin kouluttamiseen.

5.2 Unity ja ML-Agents

Unity Machine Learning Agents Toolkit (ML-Agents), kuten myös MindMaker ovat avoimen lähdekoodin projekteja, joiden avulla pelit ja simulaatiot voivat toimia ympäristöinä älykkäiden agenttien kouluttamiseen. ML-Agents tarjoa algoritmit, jotka perustuvat PyTorch-ohjelmistokehykseen.

Python-sovellusliittymän kautta käyttäjät voivat kouluttaa agentteja vahvistusoppimisen, jäljitelmän oppimisen, neuroevoluution tai muiden menetelmien avulla. Koulutettuja agentteja voidaan käyttää useisiin tarkoituksiin, esimerkiksi NPC-käyttäytymisen ohjaamiseen eri asetuksissa, kuten moniagenteissa ja taisteluissa pelaajiaan vastaan.

Yksi agentin koulutuksen haasteista on suuri pelidatan määrän efektiivinen käyttäminen oppimisessa. Sen mukaan olemassa olevat pelit kehittyvät jatkuvasti, joten harjoitusaikojen tulee olla riittävän nopeita. Unity kehitti yhteistyössä Jam City yhteisön kanssa asynkronisen ympäristön (engl. *Asynchronous Environments*), GAIL-algoritmin (engl. *Generative Adversarial Imitation Learning*) ja Soft Actor-Critic-algoritmin. Näiden työkalujen avulla voidaan kouluttaa useita agentteja samanaikaisesti eri ympäristöissä, simuloida ihmisten käyttäytymistä ja ohjata agentin käytäntöjä niin, että koulutus on monta kertaa nopeampaa.

6 PROJEKTIN SUUNNITTELU

Suunnittelu on tärkeä vaihe tekoälyn luomisessa. Tekoälykehityksessä pieni virhe voi johtaa agenttikoulutusta väärään suuntaan. Oikea suunnittelu säästää paljon aikaa kehityksessä.

Suunnitelmassa selvitetään projektin rajoja. Ensin määritellään agentin tavoitteet ja mahdolliset toimet. Agentin tavoitteiden perusteella luodaan useita

melkein samanlaisia ympäristöjä, jossa jokaisella on oma tehtävänsä. Luodaan ympäristöön tarvittavat mekaniikat ja aloitetaan agentin kouluttaminen.

Työn kehitykseen valittiin Unity-pelimoottori, koska ML-Agents laajennusta ohjataan C#-ohjelmointikielellä. ML-Agents-laajennuksen käyttäminen helpottaa koodin testaamista Unreal Engine ja Blueprint-koodaukseen verrattuna. Toinen tärkeä etu ML-Agentissa on, että koulutetut mallit voidaan visualisoida diagrammissa TensorFlowin kautta. Näin voidaan ymmärtää paremmin, miten agentti on koulutettu ja tarvittaessa muuttaa koulutusta.

6.1 Agentin tavoitteet ja toiminnot

Projektin tarkoituksena on luoda tekoäly, joka simuloi koneen automaattista pysäköintiä. Agentin tehtävänä on löytää ja päästä tarvittavaan paikkaan muuttuvassa ympäristössä. Agentin tulee myös välttää törmäyksiä muihin esineisiin.

Tarvittavien tehtävien suorittamiseksi agentin tulee pystyä ajamaan eteenpäin ja taaksepäin sekä kääntämään pyöriä molempiin suuntiin. Tällaisten mekaniikkojen luomiseen on monia vaihtoehtoja, joten ensimmäisessä versiossa käytetään yksinkertaista syötejärjestelmää, joka simuloi pelaajan syöte. Projektin parannuksien osiossa tarjotaan mahdollisia parannuksia näihin mekaniikkoihin.

6.2 Toiminnon ympäristöt

Alkuympäristössä näytetään agentille, mitä siltä odotetaan. Tätä varten ympäristössä on oltava piste, johon agentin tulisi mennä, sekä esineitä, joihin sen ei pitäisi törmätä. Törmäyksen jälkeen asetetaan negatiivinen palkinto ja aloitetaan agentin koulutus uudestaan. Jokaisessa uudessa koulutusiteraatiossa asetetaan agentille ja sen tavoitteille uudet sijainnit pelimaailmassa.

Kun agentti löytää tarvittavan kohteen nopeasti ja pääsee siihen virheettömästi, voidaan tallentaa neuroverkon malli ja aloittaa sen perusteella harjoittelu monimutkaisemmassa ympäristössä.

Seuraavassa koulutuksen syklissä luodaan muuttuva ympäristö, jossa vaikeutetaan tavoitteen saavuttamista ja siten päästään lähemmäksi realistista käyttäytymistä.

7 VAHVISTUSOPPIMINEN TOTEUTUS PELISSÄ

Suunnittelun luomisen jälkeen, jossa oli päätetty tekoälyn tehtävät ja tavoitteet, voidaan siirtyä mekaniikan ja ympäristöjen luomiseen. Koska tämän projektin luominen tiivistettiin lyhyessä ajassa, kaikki 3D-mallit ja -tekstuurit otettiin avoimista lähteistä. Samasta syystä koodin päätehtävänä oli sen yksinkertaisuus ja testauksen nopeus.

Tässä osassa tarkastellaan agentin liikkumisen mekaniikkaa ja ML-Agents-komponenttien asennusta sekä niiden yhdistäminen koodin kanssa. Ympäristöissä luodaan satunnaisesti esteiden ja agentin tavoitteen ilmaantumisen mekaniikkaa, joka vaikeuttaa agentin tehtävää.

Lopuksi eritellään kerätyt koulutustiedot ja esitetään useita vaihtoehtoja agentin koulutuksen parantamiseksi. Tämä auttaa tuomaan simuloinnin lähemmäksi reaalimaailmaa, jossa ympäristö on monta kertaa monimutkaisempi kuin pelimaailmassa.

7.1 Koodi

Agentin tulee pystyä ajamaan eteenpäin ja taaksepäin, sekä kääntämään pyöriä molempiin suuntiin. Toteutukseen valittiin yksinkertaisin syötejärjestelmä, koska se nopeuttaa kehitystä. Agentin syötteet välitetään carController-skriptille ja niin agentti saa mahdollisuuden ohjata konetta itsenäisesti (kuva 9).

```
public override void OnActionReceived(ActionBuffers actions)
{
    float forwardSpeed = 0f;
    float turnSpeed = 0f;

    switch (actions.DiscreteActions[index: 0])
    {
        case 0: forwardSpeed = 0f; break;
        case 1: forwardSpeed = +1f; break;
        case 2: forwardSpeed = -1f; break;
    }
    switch (actions.DiscreteActions[index: 1])
    {
        case 0: turnSpeed = 0f; break;
        case 1: turnSpeed = +1f; break;
        case 2: turnSpeed = -1f; break;
    }

    carController.GetInput(_verticalInput: forwardSpeed, _horizontalInput: turnSpeed);
    AddReward(increment: -0.001f);
}
```

Kuva 9. Agentin toiminnot.

Agentti tarkistaa törmäävät kohteet ja objektista riippuen agentti saa palkkion. Agentin skriptissä tarkistetaan olemassaolo komponentit ympäristöjen objektien sisällä. Siten kohteissa, joihin agentti ei voi törmätä on laitettu EnvironmentCar-skripti. Samalla periaatteella asetetaan Goal-skripti objektille, johon agentin pitäisi päästä. Hoidetaan agentin osuman OnTriggerEnter-metodin avulla.

```
void Start()
{
    x_MinCoord = transform.localPosition.x;
    z_MinCoord = transform.localPosition.z;

    x_StartMinCoord = x_MinCoord;
    z_StartMinCoord = z_MinCoord;
}

0 references
public void Spawner()
{
    int randomPlace = UnityEngine.Random.Range(minInclusive: 0, maxExclusive: Cars.Count);
    for (int i = 0; i < Cars.Count; i++)
    {
        if (randomPlace != i)
        {
            float size = Cars[index: i].GetComponent<BoxCollider>().size.x;
            Vector3 position = new Vector3(x: x_MinCoord, y: 0.0f, z: z_MinCoord);
            var GameObject Car = Instantiate(original: Cars[index: i], position: position, rotation: Quaternion.identity);

            Car.transform.SetParent(gameObject.transform.parent, worldPositionStays: false);

            x_MinCoord += size + 0.4f;
            CreatedPlaces.Add(item: Car);
        }
        else
        {
            float freePlaceSize = FreePlace.GetComponent<BoxCollider>().size.x;
            Vector3 freePlacePosition = new Vector3(x: x_MinCoord, y: 0.0f, z: z_MinCoord);
            var GameObject freePlace = Instantiate(original: FreePlace, position: freePlacePosition, rotation: Quaternion.identity);

            freePlace.transform.SetParent(gameObject.transform.parent, worldPositionStays: false);

            x_MinCoord += freePlaceSize + 0.4f;
            CreatedPlaces.Add(item: freePlace);
        }
    }
}
0 references
public void SpawnReset()
{
    for (int i = 0; i < CreatedPlaces.Count; i++)
    {
        GameObject.Destroy(obj: CreatedPlaces[index: i]);
    }
    CreatedPlaces.Clear();

    x_MinCoord = x_StartMinCoord;
    z_MinCoord = z_StartMinCoord;
}
```

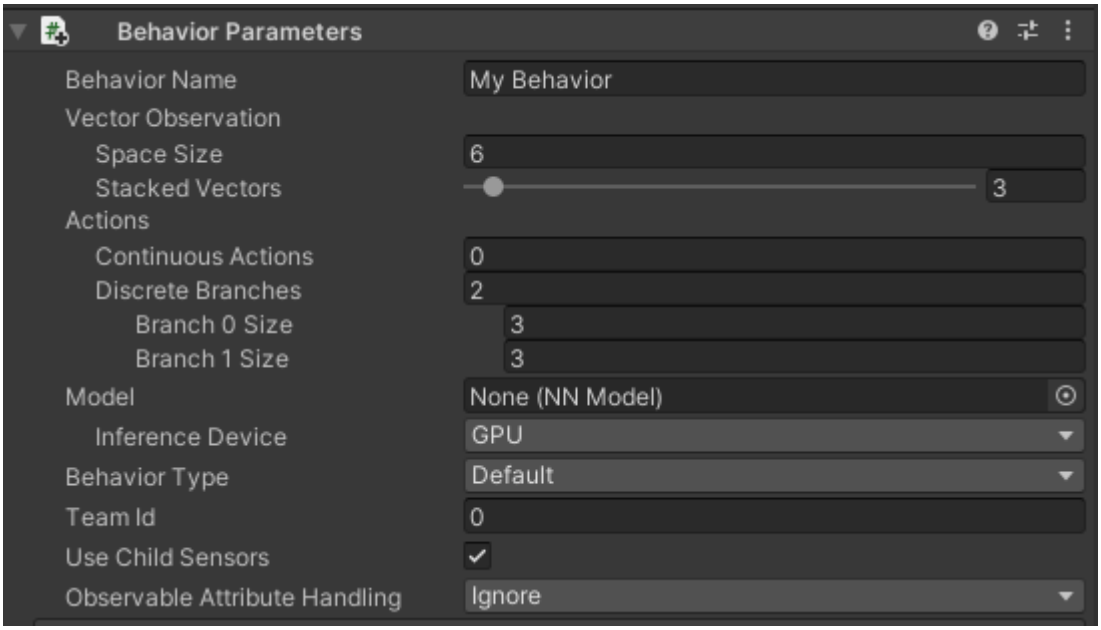
Kuva 10. Autojen satunnainen syntyminen.

CarSpawner-skripti generoi autoja ympäristöön. Niiden välissä pitää olla vähintään yksi vapaa paikka, mihin agentti yrittää päästä. Objektien satunnainen syntyminen luodaan paikallisiin muuttujiin mukaan, jotta on mahdollista monistaa ympäristöjä agentin nopeaa koulutusta varten (kuva 10).

7.2 Unity

Asynkronista oppimista varten luodaan Prefab-objektit projektin kaikille tärkeille osille niin kuin agentti, agentin tavoite ja agentin koulutuksen ympäristö. Unityn Prefab-järjestelmän avulla voidaan luoda, asettaa ja tallentaa peliobjektit sekä niiden komponentit ja asetukset. Prefab-objekti toimii kuin malli ja sen avulla voidaan nopeasti käyttää, kopioida ja muokata peliobjektia. (Prefabs s.a.) ML-Agents laajennuksen käyttämiseksi asetetaan komponentit agentille.

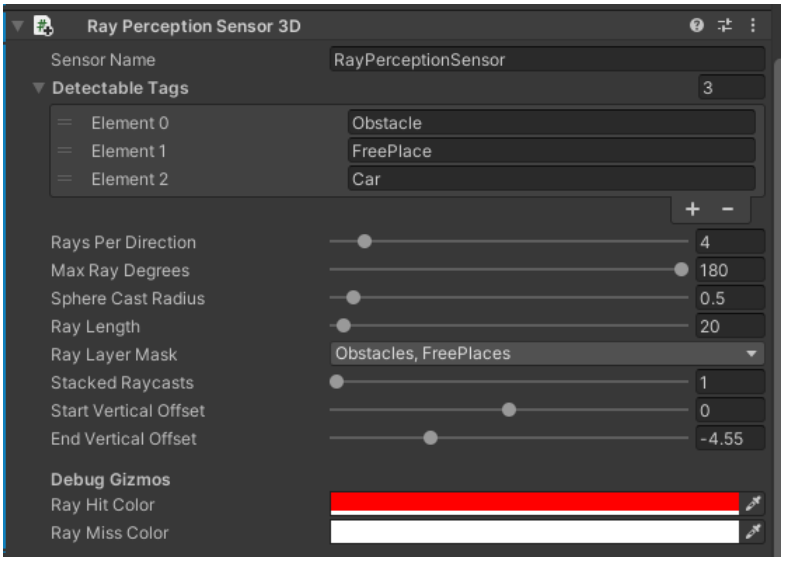
Agentin skripti periytyy ML-Agents-laajennuksesta. Kun skripti on lisätty agenttiin, laajennus luo automaattisesti Behavior Parameters -komponentin (kuva 11). Vector Observation vastaa siitä, mikä dataa tallennetaan malliin treenaamisessa. Sen perusteella mallia koulutetaan. Tässä tapauksessa tallennetaan objektit, jotka eivät saa törmätä, sekä tavoitteen paikallinen sijainti lokaalinen paikka.



Kuva 11. Behavior Parameters-komponentti.

Vector Action-funktio on vastuussa agentin toiminnasta. Funktio voi ottaa kahden tyyppisiä arvoja. Diskreetti toiminta ottaa kokonaislukuja, kun taas jatkuva toiminta ottaa liukulukuja. Koska agentin toiminta simuloi syötettä, käytetään diskreettiä vektoria. Ensimmäinen haarautuminen vastaa ajon suunnasta ja toinen pyörien kääntymistä. Projektin parannukset -osassa käsitellään jatkuva vektori ja sen käytön syitä.

Agentille asetetaan Raycast Perception Sensor 3D -komponentti, jossa ilmoitetaan, kuinka monta säteensuuntausta tarvitaan ja mihin suuntiin (kuva 12). Määritetään myös, mitkä säteensuuntaukset kohtaavat ja tallennetaan vastaanotettu data automaattisesti malliin.

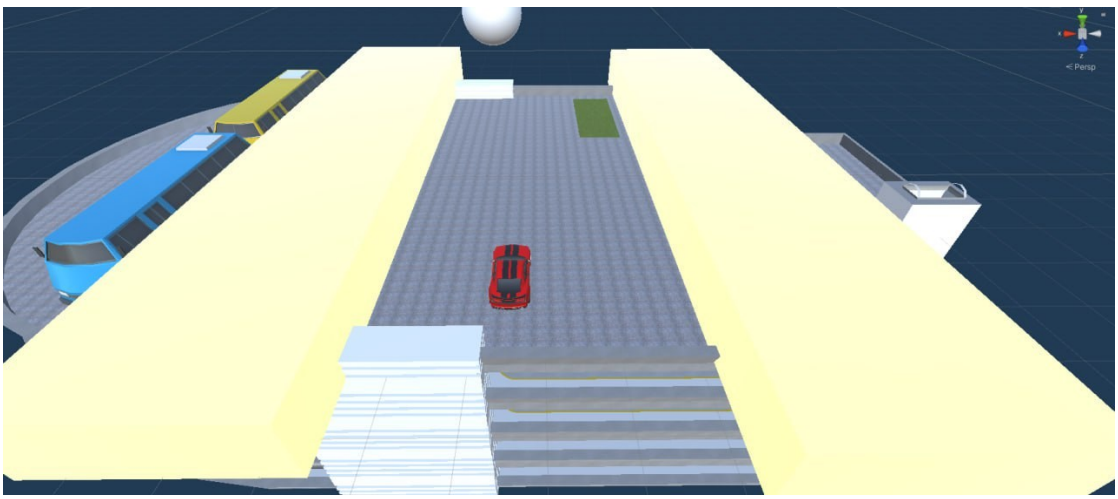


Kuva 12. Ray Perception Sensor 3D-komponentti.

Decision Requester -komponentti vastaa toimintapyynnön lähettämisestä tietyllä tiheydellä. Tätä tarvitaan määrittelemään agentin suurin askelmäärä. Tämä on tarpeen, jotta agentti pyrkii suorittamaan tehtävän mahdollisimman optimaalisesti.

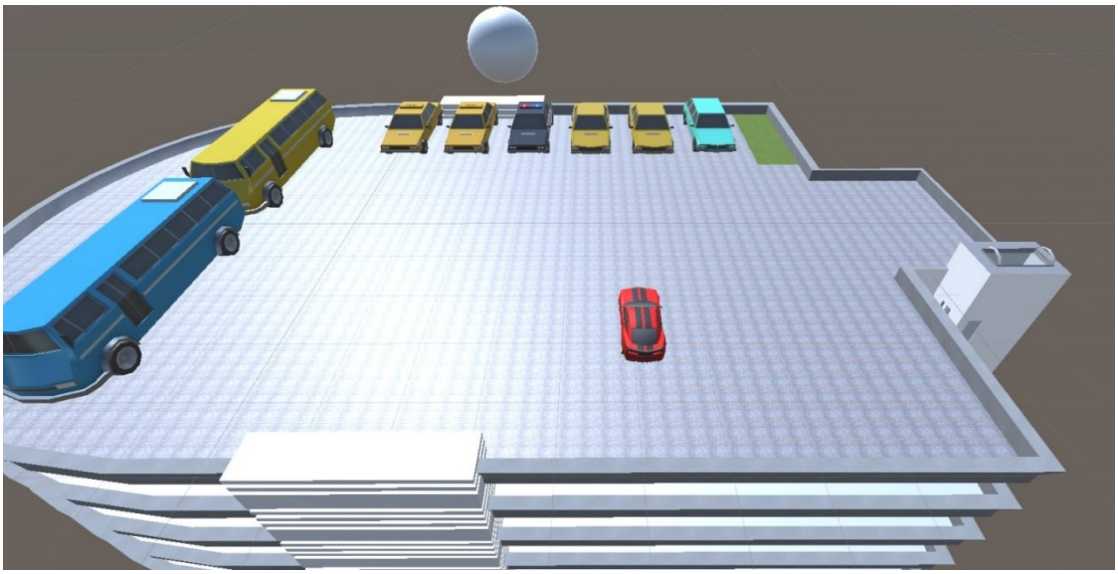
7.3 Koulutuksen ympäristöt

Ensimmäinen ympäristö sisältää vain agentin tavoitteen ja kaksi seiniä simuloivaa estettä. Voidaan myös huomata valkoisen pallon ympäristön yläpuolella. Sen väri muuttuu sen mukaan, suorittaako agentti tehtävää vai törmääkö se esteeseen. Tämä auttaa näkemään, eteneekö oppiminen oikeaan suuntaan koulutuksen ajassa.



Kuva 13. Ensimmäinen koulutuksen ympäristö.

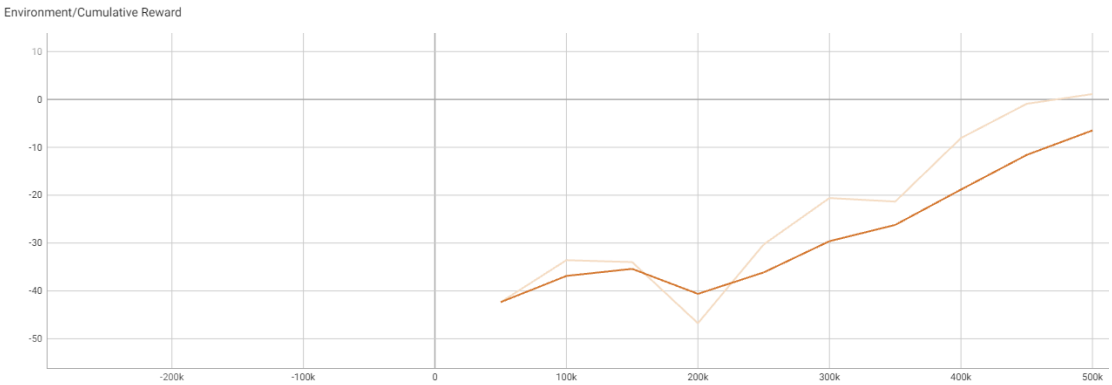
Viimeinen ympäristö eroaa siinä, että agentti voi täysin liikkua sen ympärillä. Autoja ilmestyy myös satunnaisesti agentin edessä ja niiden välissä on agentin tavoite.



Kuva 14.Viimeinen koulutuksen ympäristö.

7.4 Koulutusten tietojen esittäminen

Koulutustietojen visualisoimiseksi ML-Agents-laajennus tarjoa TensorBoard-työkalun. Sen avulla voidaan seurata kokeilumittareita, kuten häviötä ja tarkkuutta, visualisoida mallikaaviota ja paljon muuta. Tämä on erittäin tehokas työkalu, joka vaatii kykyä työskennellä datan kanssa.

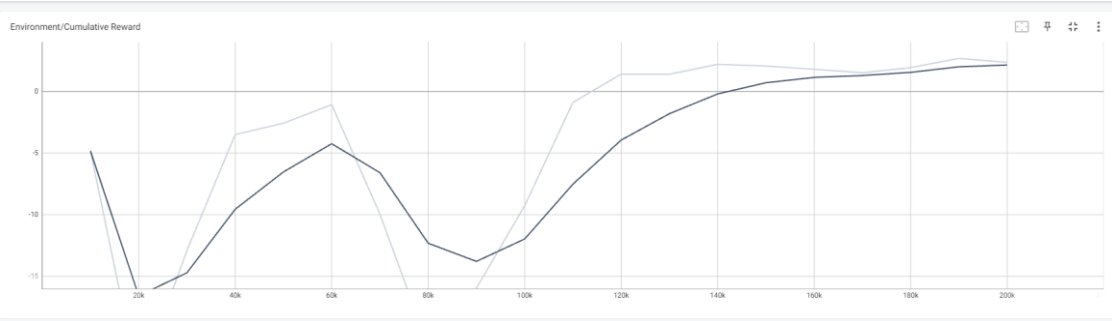


Kuva 15. Ensimmäinen koulutuksen malli.

Kuvassa 13 on esitetty ensimmäinen malli, jonka aikana agentti oppii ymmärtämään, mitä siltä vaaditaan, eli pääsemään lopulliseen tavoitteen ja välttämään törmäyksiä. Voidaan nähdä, että ensimmäisellä 200 tuhannella

iteraatiolla agentti kokeilee mahdollisia toimia ja sen jälkeen tekee yhä vähemmän virheitä saavuttaakseen tavoitteen.

Tämän mallin perusteella aloitetaan monimutkaistaa ympäristöä ja tuoda sen haluttuihin tuloksiin. On huomioitava, että agentin palkkio kasvaa harjoittelun keston myötä.



Kuva 16. Toinen koulutuksen diagrammi.

Kuvassa 14 nähdään, että toisella harjoittelulla agentti tekee alussa uusia virheitä, koska ympäristö on muuttunut. Noin puolet annetuista iteraatioista jälkeen se alkaa tehdä vähemmän virheitä ja saavuttaa tavoitteensa paremmin.

7.5 Projektin parannukset

Koulutuksen jälkeen agentti oppi useimmissa tapauksissa löytämään tarvittavan kohteen ja tulemaan sen luo. Vaikka agentin oppimistavoitteet on saavutettu, hankkeessa on vielä paljon parantamisen varaa. Tässä luvussa käsitellään sitä, kuinka tätä projektia voidaan parantaa tulevaisuudessa, jotta päästäisiin lähemmäs todellisia olosuhteita.

Aloitetaan agentin liikkeestä. Tällä hetkellä agentin liikettä simuloidaan pelaajan syötteellä. Todellisuudessa liikettä on rajoitettava moottorin teholla. Agentin on valittava, kuinka paljon voimaa liikkeeseen kohdistaa ja mihin suuntaan. Sama koskee pyörien kääntämistä. Tämä parantaa agentin liikkeen minimaalisiin liikkeisiin.

Voidaan myös pienentää agentin askelmäärää, jotta se liikkuu aina optimaalisella tavalla. Säteensuuntauksen muutokset voivat myös monipuolistaa agentin käyttäytymistä.

Seuraava mahdollinen suuri parannus on palkitsemisjärjestelmän parantaminen. Tällä hetkellä agentti saa palkinnon vain, kun hän koskettaa oikeaa kohtaa. Todellisuudessa on mahdollista antaa pieni palkkio liikuttaessa oikeaan suuntaan, esimerkiksi säteensuuntauksen avulla. Näin voidaan ohjata agenttia ympäristössä oikeaan suuntaan ihan alussa ja nopeuttaa agentin koulutusta.

Aiemmin ehdotettuja parannuksia luotaessa voidaan luoda vieläkin monimutkaisempi ympäristö valmistelemaan agentti käyttämään kaikkia saatavilla olevia kykyjä. Siten agentti voi oppia liikkumaan oikeaan suuntaan tietyssä paikassa ympäristössä välttäen törmäyksiä.

8 TUTKIMUSTULOKSET JA JOHTOPÄÄTÖKSET

Pelimoottorit tarjoavat avoimen lähdekoodin laajennuksia, joiden avulla voidaan luoda tekoälyä. Laajennukset tarjoavat mahdollisuuden käyttää erilaisia matemaattisia algoritmeja vahvistusoppimisen sisällä, mikä mahdollistaa tekoälyn luomisen tiettyjä tavoitteita varten. Tekoälyn luominen on mahdollista eri ohjelmointikielten sekä visuaalisten komentosarjojen avulla.

Kun agentti on oppinut suorittamaan annettuja tehtäviä, oppimisprosessin analysoinnin jälkeen voidaan aloittaa agenttikoulutuksen optimoinnin. Optimointi tapahtuu koodin avulla, jonka tarkoituksena on välittää agentille mahdollisimman tarkasti mitä siltä odotetaan. Tämän avulla voidaan tasapainottaa neuroverkkoja ja tehdä agentin reagointi parempi ympäristön muutoksille.

Koulutustietojen käsittelyyn on olemassa erityisiä avoimen lähdekoodin työkaluja. Lataamalla sinne koulutuksen dataa kehittäjä saa hyvin visualisoitua agentin koulutuksen dataa, jonka avulla ymmärretään miten koulutus etenee.

Tekoälyn toteutuksen periaatteiden ymmärtäminen liittyy matemaattisen teorian opiskeluun sekä itse tekoälyn kokemuksen luomiseen. Pelimoottorien laajennukset tarjoavat yksityiskohtaista dokumentaatiota ja laajennuksien toteuttajien omija esimerkkejä, jotka auttavat käyttäjää luomaan tarvitsemansa tekoälyn.

9 POHDINTA

Opinnäytetyön tarkoitus on muuttunut alkuperäisestä suunnitelmasta kirjoittamisprosessissa. Alun perin ajattelin kouluttaa agenttia niin että se suorittaa aina annetun tehtävän virheettömästi ja käyttää kaikkia käytettävissä olevia toimintoja. Teoriaan tutustumisen ja projektin luomisen aikana tuli selväksi, että tämä vaatii paljon enemmän aikaa agentin kouluttamiseen ja koodin parantamiseen. Siitä huolimatta sain agentin suorittamaan tehtävän ja pohtimaan mahdollisia parannuksia projektin tarkoituksiin.

Tutkimuksen tulokset voidaan toteuttaa uudestaan lähteiden avulla. Laajennuksen kehittäjät tarjoavat pääsyn omiin projekteihin, joissa laajennuksen käyttäjä voi tutustua erilaisiin tekoälyn projekteihin. Tekoälyn ja vahvistusoppimisen teorian avulla voidaan ymmärtää paremmin, mitä tekoälyn työkalun sisällä tapahtuu ja käyttää sitä mahdollisimman tehokkaasti.

Tutkittu teoria osoittaa, kuinka laaja tekoälyn teoria on. Agentti koulutettiin vahvistusoppimisen perussääntöjen mukaan, mutta neuroverkon tasapainotus on tarkempi ja vaikeampi kuin ajattelin teorian tutkimuksen aikana. Neuroverkon tasopainotuksen sääntöjen ymmärtäminen tuli minulle enemmän erilaisista testeistä saatujen kokemusten myötä.

Minulla on selkeä käsitys siitä, kuinka tätä projektia voi parantaa jatkossa. Projektin parannukset -osiossa annettiin esimerkkejä siitä, miten projektia voidaan parantaa ja mitä sillä voidaan saavuttaa.

10 YHTEENVETO

Tekoäly ja neuroverkot integroituvat jokapäiväiseen elämään. Lähitulevaisuudessa niiden käyttäminen helpottaa ja nopeuttaa kehitystä monella alalla. Nykymaailma koostuu massadatasta ja vain neuroverkot voivat auttaa sen käsittelyssä. Saatujen tietojen käyttäminen on myös suuri taito, joka vaatii paljon kokemusta ja ymmärtämistä.

Pelialalla tekoälyn käyttäminen monipuolistaa pelikokemusta ja voi antaa ainutlaatuisen käyttökokemuksen pelaajalle. Vahvistusoppiminen tekee

muutakin kuin vain monipuolistaa ei-pelaajahahmojen käyttäytymistä. Tekoälyn avulla voidaan kerätä pelaajien tietoja ja kehittää tuoteitä niiden perusteella. Sen avulla voidaan ymmärtää paremmin pelien yleisöä ja parantaa pelikokemusta.

Tekoälyn työkalujen käyttö pelimoottoreissa antaa mahdollisuuden simuloida todellisen maailman ympäristöjä. Sen avulla voidaan löytää mahdolliset ongelmat ja niiden ratkaisut lyhyessä ajassa, ilman varsinaista luomista. Nämä työkalut säästävät paljon henkilötunteja sekä todellisia resursseja, kuten robotin osia.

Tämän työn tehtäviin valmistautuessa löydettiin monia erilaisia projekteja. Niiden tutkiminen nopeuttaa työkalujen ymmärtämistä ja käyttämistä. Myös laajennusten tekijöiltä on omija projekteja, jotka auttavat ymmärtämään tekoälyn toimintaa.

Perusmekaniikan ja ympäristön luominen ei vaatinut paljon aikaa. Suurin osa ajasta kului agentin kouluttamiseen ja virheiden löytämiseen koulutuksen jälkeen. Rajallisen kokemuksen vuoksi ongelmakohtien etsintä kesti kauemmin. Tekoälyn ja neuroverkkojen kehityksessä aina saadaan kokemusta, joka parantaa seuraavien projektien kehitystä.

LÄHTEET

Artificial Intelligence, Machine Learning, and Deep Learning: Same context, Different concepts. 2018. IESC. WWW-dokumentti. Päivitetty 16.4.2018. Saatavissa: https://master-iesc-angers.com/artificial-intelligence-machine-learning-and-deep-learning-same-context-different-concepts/ [viitattu 6.11.2022].

Copeland, M. 2016. What's the Difference Between Artificial Intelligence, Machine Learning and Deep Learning. WWW-dokumentti. Päivitetty 29.7.2016. Saatavissa: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/ [viitattu 6.11.2022].

Krumins, A. 2020. MindMaker. WWW-dokumentti. Saatavissa: https://github.com/krumiaa/MindMaker [viitattu 6.11.2022].

Nilsson, N. 1998. Artificial Intelligence: A New Synthesis. San Francisco: Morgan Kaufmann Publisher.

Ravichandiran, S. 2018. Hands-On Reinforcement Learning with Python. Birmingham: Packt Publishing.

Russell, R. 2018. Step-by-Step Guide To Implement Machine Learning Algorithms with Python.

Unity Documentation. 2018. Prefabs. WWW-dokumentti. Päivitetty 31.7.2018. Saatavissa: https://docs.unity3d.com/Manual/Prefabs.html [viitattu 26.3.2023].

Unity Technologies. 2017. ML-Agents. WWW-dokumentti. Päivitetty 14.1.2022. Saatavissa: https://github.com/Unity-Technologies/ml-agents [viitattu 6.11.2022].

Unreal Engine Documentation. Introduction to Blueprints. s.a. WW-dokumentti. Saatavissa: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/ [viitattu 26.3.2023].

Winston, P. 1992. Artificial Intelligence Third Edition. Addison-Wesley Publishing Company.