Paloranta Wilma

Koneoppiminen videopelien NPC-hahmojen kehityksessä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

30.4.2024

Tiivistelmä

Tekijä:	Wilma Paloranta
Otsikko:	Koneoppiminen videopelien NPC-hahmojen kehityksessä.
Sivumäärä:	32 sivua
Aika:	30.4.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaajat:	Lehtori Miikka Mäki-Uuro

Insinöörityön tarkoituksena oli tutkia tekoälyn ja koneoppimisen hyödyntämistä NPC-hahmojen (ei-pelattavien hahmojen) kehittämisessä. Työn tavoitteena oli tutkia, voiko pienen projektin kautta luoda älykkään ja järkevän tekoälyn, jonka toiminnot vastaavat haluttua lopputulosta. Lopputuloksena haluttiin tekoälymalli, joka pärjää erilaisia vihollisia vastaan.

Projekti tehtiin Unity-pelimoottorissa ja sen pohjalla käytettiin ML-Agents- sekä TensorBoard-kirjastoja. Tekoäly saatiin luotua ML-Agentsien avulla ja koulutusta pystytään seuramaan Tensorboardin kaavojen avulla. Ohjelmoinnissa käytettiin C#-ohjelmointikieltä.

Työn lopuksi todettiin, että ML-Agents toimii hyvin erilaisten yksinkertaisten tekoälymallien kehittämisessä. Koulutuksessa meni hetki aikaa, mutta haluttu lopputulos saatiin kuitenkin. Koneoppimisen käyttäminen yksinkertaisen NPC:n luomiseen saattaa viedä enemmän aikaa kuin manuaalisesti halutun käytöksen ohjelmointi, mutta koneoppiminen on hyvä työkalu monimutkaisille tapauksille.

Yksi suurimmista haasteista projektin aikana oli koulutuksen kesto. Alussa liian lyhyet tekoälymallin johtivat siihen, ettei malli ehtinyt tutkia erilaisia ratkaisuja parhaiden tuloksien saamiseksi. Hyperparametrit aiheuttivat jonkin verran ongelmia, mutta ne ratkaistiin pitkäjänteisen hienosäädön jälkeen.

Insinöörityössä onnistuttiin tekemään pieni ampumasimulaatio, jossa agentti osaa ampua kohti tulevia vihollisia tarkasti ja oikeassa järjestyksessä.

Avainsanat: tekoäly, koneoppiminen, Unity, pelikehitys

Abstract

Author:	Wilma Paloranta
Title:	Machine learning in video game NPC character development
Number of Pages:	32 pages
Date:	30 April 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Game Applications
Supervisors:	Miikka Mäki-Uuro, Senior Lecturer

The purpose of this engineering thesis was to investigate the utilization of artificial intelligence and machine learning in the development of NPC (non-playable character) characters. The goal of the work was to see if through a small project, it is possible to create an intelligent artificial intelligence whose actions correspond to the desired outcome.

The project was carried out using the Unity game engine, utilizing the ML-Agents and TensorBoard libraries. Artificial intelligence was created using ML-Agents, and training progress was monitored using TensorBoard charts. C# programming language was used for coding.

In conclusion, it was found that ML-Agents work well for developing various simple artificial intelligence models. Training took some time, but the desired outcome was achieved, nonetheless. Using machine learning with simple NPCs may take more time than manually programming desired behavior, but it is a useful tool for more complex cases.

The main challenge during the project was found to be the duration of training. Initially, training sessions that were too short, caused confusion and unnecessary tweaking of code or parameters. There were also some issues with hyperparameters, but these were overcome when suitable values were found for the intended purpose.

The result was a success in creating a small shooting simulation where the agent is able to shoot accurately and in the correct order towards incoming enemies.

Keywords: artificial intelligence, machine learning, Unity, game development

Sisällys

Lyhenteet

1 Johdanto 1

2 Koneoppiminen 2

2.1 Koneoppimisen perusteet 2
2.2 Koneoppiminen videopeleissä 3
2.3 Neuroverkot 7
2.4 Vahvistusoppiminen 12

3 Projektin työkalut ja parametrien säätäminen 13

3.1 ML-Agents 13
3.2 Hyperparametrit 15

4 Neuroverkon kehittäminen ampumapelissä 17

4.1 Suunnittelu ja ympäristön luonti 18
4.2 Agentti 21
4.3 Koulutus 24

5 Projektin arviointi 26

5.1 Ongelmatilanteet 27
5.2 Loppuanalyysi 30

6 Yhteenveto 32

Lähteet 1

Lyhenteet

GTA: *Grand Theft Auto*. Rockstar Gamesin luoma toimintapelisarja.

MOBA: *Multiplayer Online Battle Arena*. Taisteluareenamoninpeli eli reaaliaikainen strategiapelien lajityyppi.

NPC: *Non-playable character*. Termi, jota käytetään ei-pelattavista hahmoista videopeleissä.

1 Johdanto

Tekoäly on teknologia, jolla yritetään luoda erilaisia älykkäitä ohjelmia erityisesti tietokonejärjestelmille. Tekoälyn on tarkoitus pystyä tekemään erilaisia tehtäviä, jotka normaalisti ovat vaatineet ihmisten osallistumista. Näihin tehtäviin kuuluu ongelmanratkaisu, havaitseminen, päättely, oppiminen ja kielen ymmärtäminen.

Tämän päivän tunnetuimpia esimerkkejä tekoälyn käytöstä ovat ohjelmat kuten ChatGPT, erilaiset kuvanmuokkausohjelmat ja erilaiset digitaaliset apuvälineet, joita jokainen käyttää melkein päivittäin. Tekoälyn käyttäminen on edistynyt videopeleissä mahdollistaen erilaisen ja satunnaisemman tavan luoda tarinaa ja erilaisia pelaajan ja pelimaailman välisiä kohtauksia.

Insinöörityössä tehdään yleiskatsaus tekoälyn mahdollisuuksista videopeleissä. Työssä katsotaan mitä tekoäly on, ja vielä tarkemmin, mitä koneoppiminen on. Tätä havainnollistamaan on luotu Unityssa ampumapelisimulaatio, jossa pelin sisäinen kone, eli agentti, ampuu ja välttää vihollista käyttäen koneoppimista. Tässä projektissa on käytetty Unity-kirjastossa olevaa ML-Agents-työkalua. Tämän kirjaston lisäksi määritellään muut vaikutteet, kuten hyperparametrit, parhaan oppimistuloksen saamiseksi.

Luvussa 2 perehdytään perusmääritelmiin projektissa käytetyistä teknisistä menetelmistä, kuten koneoppimisesta, vahvistusoppimisesta ja neuroverkoista. Sen jälkeen luvussa 3 perehdytään hieman suppeampiin menetelmiin, kuten ML-Agentsit ja hyperparametrit. Lopussa, miten itse projektin suunnittelu ja toteutus onnistui ja mitä projektista opittiin.

2 Koneoppiminen

Koneoppiminen on tekoälyn alaluokka, jossa tietokone oppii itsenäisesti tekemään oikeita päätöksiä ilman ihmisen suoraa vaikutusta. Konetta ohjataan oikeaan ja haluttuun suuntaan erilaisilla kannustimilla. Muuten sen annetaan itse etsiä parhaat lopputulokset. Yleensä koneoppiminen sisältää mallin kouluttamisen erilaisilla data-aineistoilla, jotta malli voi tunnistaa halutut suhteet tai kuviot. Tätä opittua dataa malli käyttää soveltaessaan uutta, ennalta näkemätöntä dataa ja tekee itsenäisesti uusia päätöksiä tai ennusteita parhaaksi näkemällään tavalla.

2.1 Koneoppimisen perusteet

Koneoppimisen voi määritellä monella erilaisella tavalla. Data-analyytikko Tom M. Mitchell tiivistää määritelmän kaavaan, jossa kone oppii kokemuksesta E (experience) suhteessa tiettyyn tehtäväluokkaan T (tasks) ja suoritusmittaan P (performance measure) [1]. Gollapudi Sunila on määritellyt koneoppimisen älylliseksi rakenteeksi, joka pystyy oppimaan omasta kokemuksestaan [2]. Koneoppiminen ja sen sisältämä mallin haku tai hahmontunnistus on pohjimmiltaan tutkimusala siitä, kuinka koneet havaitsevat ympäristön, oppivat erottamaan halutun käyttäytymisen muusta ja pystyvät tekemään järkeviä päätöksiä käyttäytymisen luokittelusta. Tavoitteena on edistää tarkkuutta, nopeutta ja välttää järjestelmän vääränlaista oppimista [3].

Koneoppiminen jaetaan yleisesti kolmeen eri menetelmään: ohjattuun oppimiseen, ohjaamattomaan oppimiseen ja vahvistusoppimiseen. Ohjatussa koneoppimisessa malli oppii merkityn tietoaineiston avulla, mitä sille syötetään [3]. Tässä tapauksessa haluttu lopputulos tai tavoite on jo tiedossa. Ohjattua koneoppimista käytetään yleensä kuvien tunnistamisessa, riskien arvioinnissa ja erilaisissa ennakoivissa analytiikoissa.

Ohjaamattomassa mallissa valmiiksi määriteltyjä luokkia ei ole. Mallille syötetystä ei-merkitystä datasta koneen on tarkoitus löytää jonkinlainen rakenne ja jakaa data ryhmiin, ryppäisiin (clusters) tai esitysmuotoon.

Vahvistusoppimismuodossa mallin tai agentin, eli koulutettavan hahmon, pitää itse pystyä operoimaan ympäristössään. Malli saa palautetta suorituksestaan mahdollisesti pienellä viiveellä ja sitten kehittää itseään [4]. Työn projektissa tullaan käyttämään vahvistusoppimista, joten se määritellään tarkemmin luvussa 2.4.

Koneoppiminen on nykyään yksi tietotekniikan suosituimmista ja kiinnostavimmista aiheista, ja sen suosio on kasvanut huomattavasti viime vuosina. Koneoppimista hyödynnetään laajasti eri aloilla, kuten roskapostisuodattimissa, personoiduissa suosituksissa verkkosivuilla, itseohjautuvissa autoissa ja lääkkeiden kehityksessä [5]. Koneoppiminen on osa monen ihmisen arkipäivää, vaikka sitä ei aina huomaakaan. Tulevaisuudessa odotettavissa on entistä enemmän kehitystä, kun aineistoa kerätään lisää, laskentateho kasvaa ja algoritmit kehittyvät edelleen.

Turingin testi on yksi yksinkertaisimmista tavoista, jolla voi demonstroida tietokoneen tai ohjelman älykkyyttä verrattuna ihmisen älykkyyteen. Kone läpäisee testin, jos ihminen luulee tekoälyä toiseksi ihmiseksi sen kanssa keskustellessa. Testi luotiin vuonna 1950 ja on ollut perusta tekoälyn teorialle ja kehitykselle [6].

2.2 Koneoppiminen videopeleissä

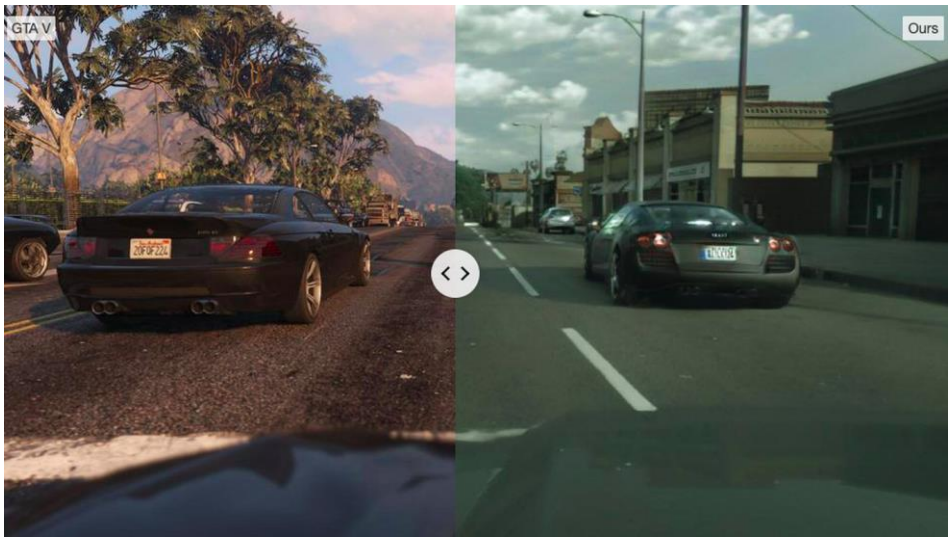
Koneoppiminen ja sen kehitys on lisääntynyt paljon videopelien ansiosta. Ensimmäiset tunnetut tapaukset tekoälyn hyödyntämisestä videopeleissä ovat vuodelta 1951 pelistä nimeltä Nim [7]. Kyseisessä pelissä pelaajat vuorotellen poistavat tulitikkuja pelilaudalta, kunnes tikkuja ei enää ole jäljellä. Voittaja riippuu käytettävistä säännöistä. Jo tuon ajan agentti oppi pelaamaan ihmistä vastaan yllättävän hyvin. Toinen varhaista tekoälyä hyödyntävä peli oli shakki. Vuonna 1997 Deep Blue -shakkirobotti voitti Garry Kasparovin, venäläisen shakin maailmanmestarin [8].

Vuoden 1997 Deep Blue -shakkirobotti ohjelmoitiin vielä erilaisten shakkimestarien sääntöjen ja syötteiden avulla, mutta nykyaikaiset shakkibotit usein koulutetaan koneoppimisen avulla. Yksi ensimmäinen koneoppimista käyttävä videopeliesimerkki on vuodelta 1993, kun opiskelijat Carnegie Mellon yliopistossa tutkivat konvoluutioneuroverkkoja (CNN, Convolutional Neural Network). He loivat koneoppimismallin videopeliin nimeltä DOOM, jossa agentti pystyi pelaamaan pelin läpi käyttäen vain pelin kuvasyötteitä [9].

Videopelien tekoäly on toisinaan joutunut väittelyn alaiseksi. Tämä johtuu siitä, että perinteinen videopelien tekoäly ei ole sisältänyt koneoppimista tai ihmisen aivojen simulointia, jotka läpäisisivät Turingin testin. Sen sijaan videopelien tekoäly on ollut lähempänä automatisoitua koneellistamista. Nykypäivänä suurien kielimallien lisäys esimerkiksi NPC-hahmoihin (ei-pelattaviin hahmoihin) on lisääntynyt. Tätä ovat käyttäneet modit (mod), eli pelaajien itse luoma lisäsisältö sekä uusimmat videopelit [10].

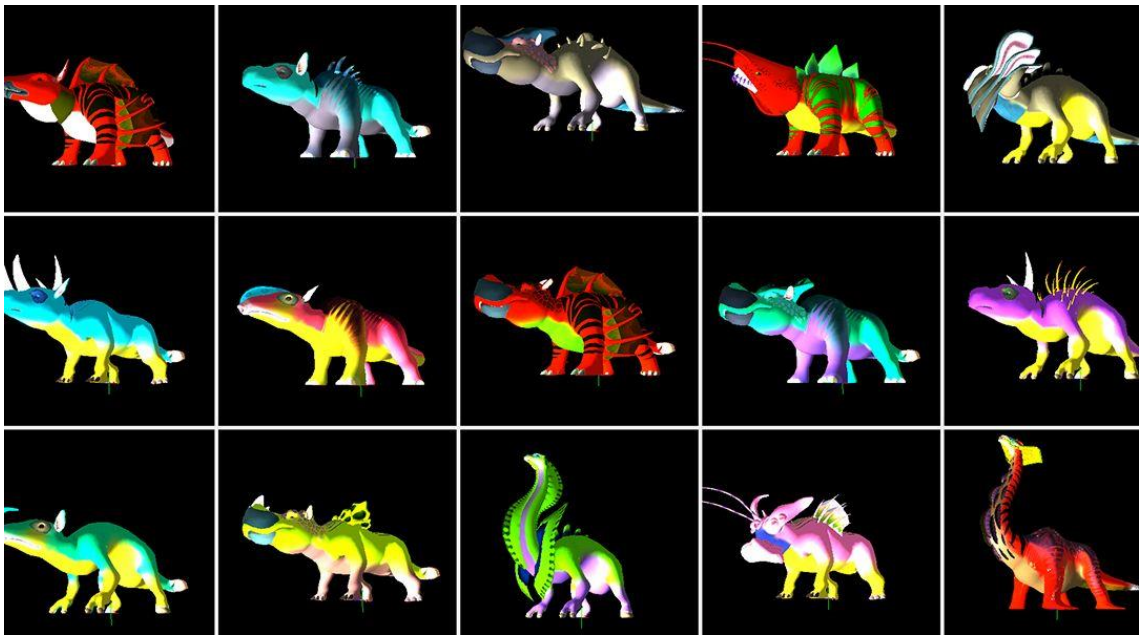
NPC:t, eli ei-pelattavat hahmot, ovat pelin sisäisiä hahmoja, jotka ovat vuorovaikutuksessa pelaajan kanssa. Tekoälyn avulla kyseisistä hahmoista voidaan tehdä älykkäitä ja realistisia. Algoritmien avulla NPC:t voivat mukautua ja muuttaa päätöksiään pelaajan toiminnan perusteella [10]. Videopeleissä on monenlaisia erilaisia tekoälyjä, kuten polunetsintä, jossa algoritmi määrittelee optimaalisen reitin NPC:ille navigoida pelimaailmassa esteitä välttäen Päätöksenteon avulla NPC:t voivat tehdä älykkäitä päätöksiä ennalta määritettyjen sääntöjen tai opitun käyttäytymisen perusteella. Oppiva tekoäly mukautuu ajan myötä ja parantaa suorituskykyään koneoppimisen ja neuroverkon kaltaisilla tekniikoilla.

Esimerkkejä koneoppimisen käytöstä on monista eri videopeleistä. Grand Theft Auto 5 (GTA) käyttää koneoppimista luodakseen hyperrealistisen pelimaailman grafiikat [11]. Kuvassa 1 näkyy eroja normaalien perinteisten grafiikkojen ja Intelin tekemän koneopilla kehitettyjen grafiikkojen välillä. GTA:n kehittäjät käyttävät tekoälyä parantaakseen tekstuureita. He käyttävät erilaisia data-aineistoja ja neuroverkkoja löytääkseen erilaisia malleja, joita kone osaa käyttää tekstuurien parantamisessa [10].



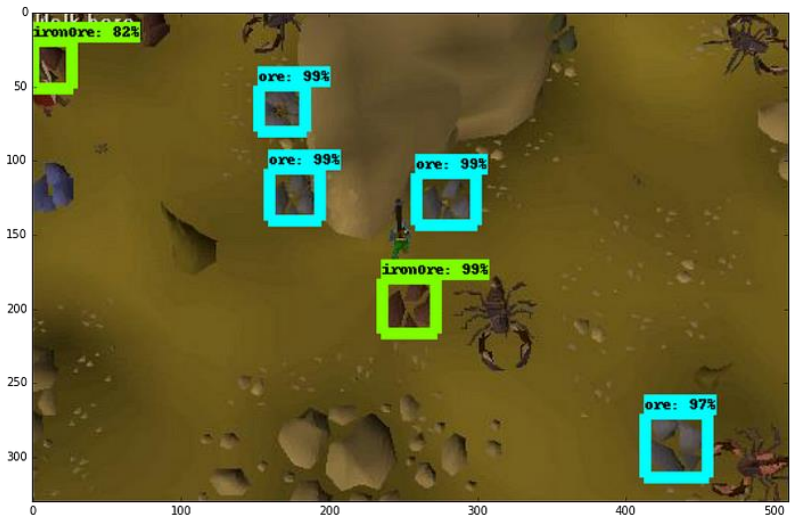
Kuva 1. GTA 5 ennen (vasemmalla) ja jälkeen (oikealla) Intelin koneoppimisella parannelluilla grafiikoilla [11].

Videopeli No Man’s Sky tunnetaan laajasta maailmasta ja miljoonista planeetoista. Planeettojen luominen käsin ei olisi ollut mahdollista rajatussa ajassa. Manuaalisen ratkaisun sijaan pelissä käytetään proseduraalista luomista, jossa maailma luodaan algoritmien avulla [12]. Videopeli Minecraft käyttää samanlaista ideaa, jossa jokaisella pelikerralla pelaaja saa uuden ja ainutlaatuisen maailman. Minecraft-peliin verrattuna No Man’s Sky on hieman edellä ja hienovaraisempi maailman luonnissa. Jokaisella planeetalla on omat säännöt luonnon ja ilmaston luomiseen. Kuvassa 2 näkyy, kuinka pelin sisäiset olennot muuttuvat ja kehittyvät olentojen oman algoritmin mukaan.



Kuva 2. No Man’s Sky -pelin koneoppimisella generoituja olentoja [13].

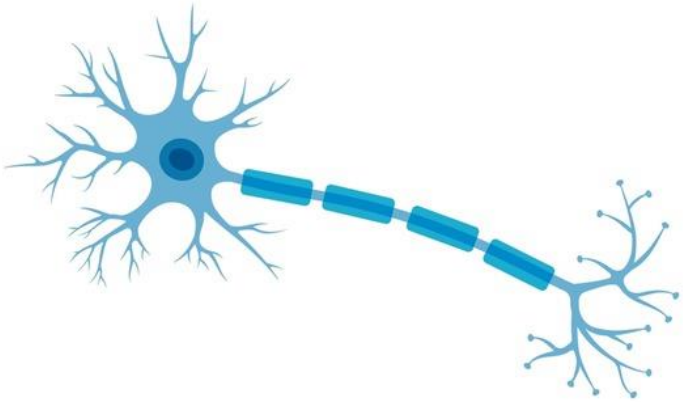
Ohjattua ja vahvistusoppimista käytetään usein erilaisten bottien tai agenttien luomisessa. Pelaajat voivat tehdä peleihin omia koodeja, joiden avulla luodaan ihmistä parempia pelaajia tai pelaajan suoritusta edistäviä apuvälineitä. [8] Tällaisia botteja käytetään usein erilaisissa moninpeleissä tai MOBA-peleissä (Multiplayer Online Battle Arena), eli suurissa moninpeli areenoissa. RuneScape on yksi suositumpia pelejä, joita varten botteja tehdään. Kuvassa 3 näkyy, kuinka kyseisessä pelissä käytetään kuvantunnistusta ja koneoppimista objektien tunnistamisessa [14].



Kuva 3. Videopelin RuneScape kuvakaappaus, jossa näkyy eri objekteja korostava kuvantunnistusohjelma [14].

2.3 Neuroverkot

Neurobiologit eivät vielä ole löytäneet tarkkaa rakennetta ja prosessia, jotka tapahtuvat ihmisten hermosoluissa. Siitä huolimatta ihmisiä kiinnostaa, mistä ajatukset lähtevät liikkeelle ja miltä ajatukset näyttävät. Tämä kiinnostus on auttanut ihmisiä luomaan uusia teknologioita, jotka heijastavat aivojen kapasiteettia. Kuvassa 4 on havainnekuva ihmisen biologisesta neurohermosta, jonka avulla tekoälyssä neuroverkkoja on lähdetty kehittämään.



Kuva 4. Ihmisen biologinen hermo [15].

Neuroverkot (neural network) on koneellinen tekoälyn muoto, joka sisältää prosessointielementtejä (neuronit) ja yhteyden niiden välillä. Tämä tekoälyn muoto on saanut inspiraationsa oikeiden hermostojen ajatusprosessista. Nämä yhteydet luovat neuronaalisen rakenteen, johon on yhdistetty erilaiset mallit ja algoritmit [16].

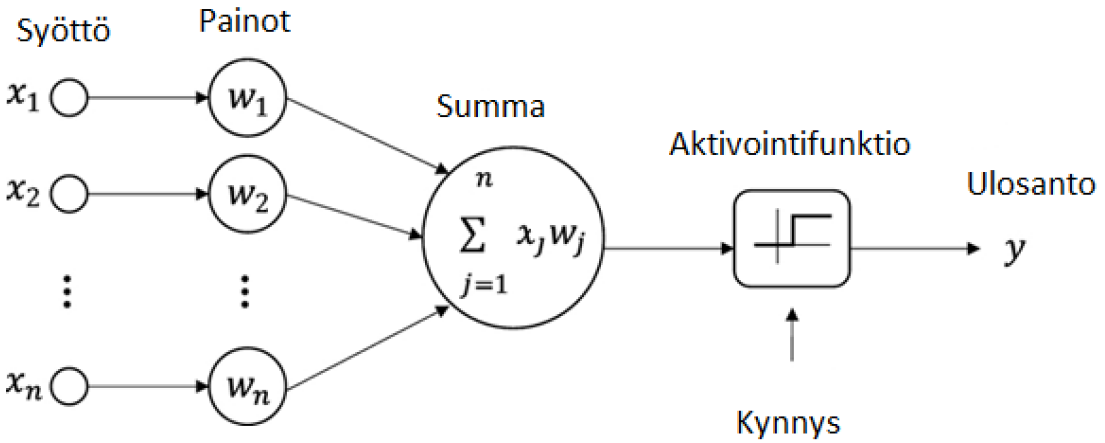
Syväoppimista ja neuroverkkoja käytetään usein synonyymeina, mikä voi olla hieman hämmentävää. Syväoppimisen osa neuroverkoissa yleensä viittaa neuroverkkojen erilaisiin kerroksiin. Neuroverkkoa, jossa on enemmän kuin kolme kerrosta, voidaan kutsua syväoppimiseksi [17]. Neuroverkot, joissa on kolme kerrosta tai vähemmän ovat tavallisia neuroverkkoja.

Jokainen neuroverkko sisältää erilaisia solmuja (node) tai toisin sanoen keinotekoisia neuroneita, syöttökerroksen, yhden tai sitä enemmän piilotettuja kerroksia ja ulosantokerroksen. Jokainen solmu yhdistyy toisiin ja sillä on oma painoarvonsa sekä kynnyksensä. Jos ulos annetun tiedon arvo on korkeampi kuin asetetun kynnyksen, solmu aktivoituu ja tieto lähetetään seuraavaan kerrokseen kyseisessä verkossa [17].

Jokainen solmu on oma yksittäinen lineaarinen regressiomallinsa, joka koostuu tulotiedoista, painotuksista, kynnysarvosta ja ulosannosta. Kuvassa 5 havainnollistetaan keinotekoista neurosolmua. Kaava näyttää tältä:

$$\sum wixi + kynnys = w1x1 + w2x2 + w3x3 + kynnys$$

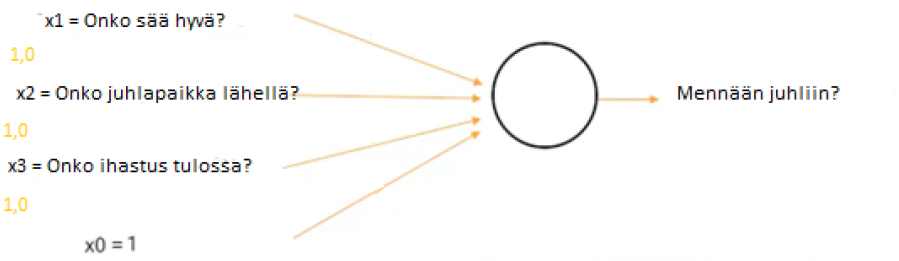
$$ulosanto = f(x) = 1\ if\ \sum w1x1 + b >= 0;\ 0\ if\ \sum w1x1 + b < 0$$



Kuva 5. Neuroverkon solmujen kaava [18].

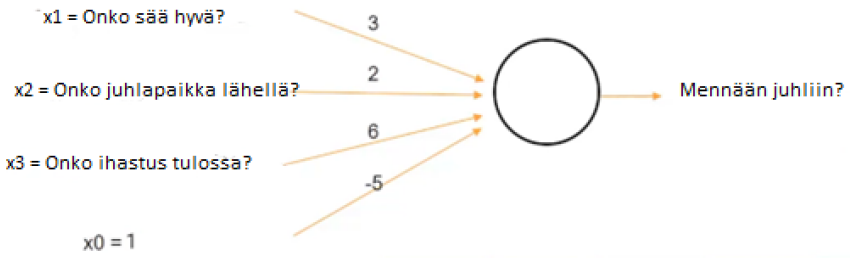
Kun syöttökerros on päätetty, painoarvot lasketaan. Nämä painoarvot auttavat määrittämään minkä tahansa tietyn muuttujan tärkeyden, ja suuremmat arvot vaikuttavat enemmän lopputulokseen verrattuna muihin syötteisiin. Kaikki syötteet kerrotaan sitten vastaavilla painoilla ja lasketaan yhteen. Tämän jälkeen saatu arvo ohjataan aktivointitoiminnon läpi, mikä määrittää lähdön. Jos tämä lähtö ylittää tietyn kynnyksen, se aktivoi solmun siirtäen tiedot verkon seuraavaan kerrokseen. Tämä johtaa siihen, että yhden solmun lähtö tulee seuraavan solmun tuloksi.

Esimerkkinä voidaan tutkia melkein mitä tahansa arkisia päätöksiä, joita ihmiset tekevät. Oletetaan, että huomenna on juhlat ja mietitään, osallistutaanko niihin. Alla olevassa kuvassa näkyy kolme syötettä: Onko sää hyvä? Onko juhlapaikka lähellä? Onko ihastus tulossa? Kuvissa 6 ja 7 kuvataan kyseinen ongelma ja siihen tarvittavat syötteet.



Kuva 6. Havainnekuva päätösfunktiosta [19].

Tiedetään syötteet, joten annetaan niille painoarvot. Sanotaan, että sää saa painoarvon 3, juhlapaikan läheisyys arvon 2 ja ihastuksen näkeminen arvon 6. Kynnyksenä pidetään arvoa 5. Lopputulos tässä tapauksessa ylittää kynnyksen, eli agentti menisi juhliin.

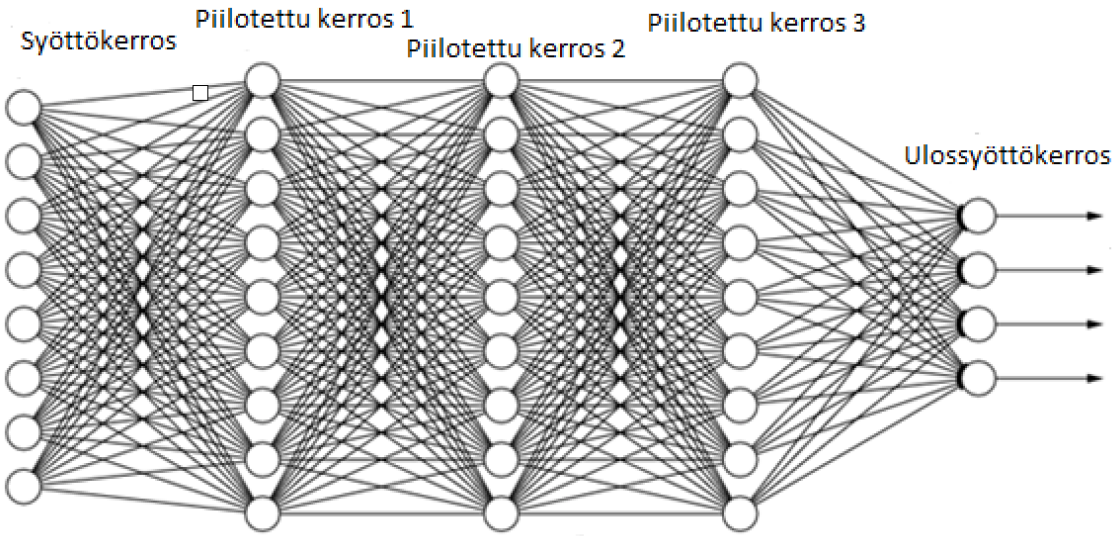


Kuva 7. Havainnekuva päätösfunktiosta [19].

Koneoppimisesta puhuttaessa puhutaan sen tarkkuudesta ja siitä, kuinka hyvin se toimii. Tärkein tutkittava on koneoppimisessa ennustevirheet. Koneoppimismallia suunniteltaessa, malli luokitellaan hyväksi koneoppimismalliksi, jos se yleistää ongelma-alueen uuden syöttödatan oikein. Tämä auttaa tekemään ennusteita tulevasta datasta, jota tietomalli ei ole koskaan nähnyt. Halutaan tarkistaa, kuinka hyvin koneoppimismalli oppii ja mukautuu uuteen dataan. Sitä varten on hyvä tarkistaa, aiheutuuko mallissa yli- tai alisovitusta, jotka ovat pääosin vastuussa koneoppimisalgoritmien huonosta suorituskyvystä [20].

Tietoa ylisovittamisesta neuroverkossa voidaan ilmaista seuraavalla tavalla: kun neuroverkko on ylisovitettu, siinä on liikaa kerroksia, neuroneita tai molempia. Näin ollen se yrittää löytää liian monimutkaisia suhteita datasta, mikä ei välttämättä ole olennaista ongelman ratkaisun kannalta. Sen sijaan verkko voi kiinnittyä tiettyihin piirteisiin aineistossa, mikä tekee siitä herkän muutoksille ja vaikeuttaa sen soveltamista uusiin tilanteisiin. Alisovittaminen puolestaan tarkoittaa, että neuroverkossa ei ole tarpeeksi neuroneita. Näin se ei kykene käsittelemään kaikkea oleellista tietoa aineistosta, mikä rajoittaa sen suorituskykyä ongelmaa ratkaistaessa.

Kuvassa 8 on havainnekuva kokonaisen neuroverkon rakennelmasta, jossa on solmuja tai neuroneja eri tasoilla. Verkossa näkyy syöttökerros, piilotetut kerrokset ja ulossyöttökerrokset. Kuten aiemmin mainittiin, solmut ovat linkitettynä toisiinsa ja pitävät sisällään eri painoarvot ja kynnykset. Kun solmun arvo on suurempi kuin sen kynnys, se aktivoituu ja data viedään eteenpäin seuraavalle tasolle verkossa [21].



Kuva 8. Kokonainen neuroverkko [21].

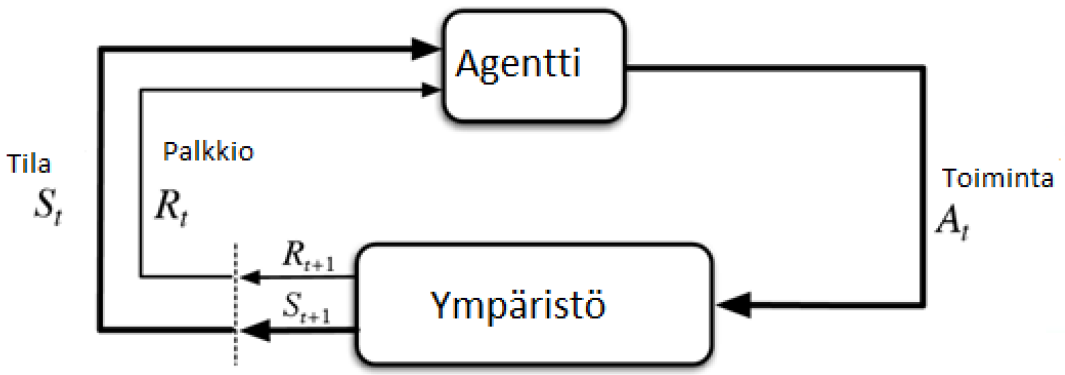
Kun neuroverkon koulutus aloitetaan, painoarvot asetetaan aluksi satunnaisesti. Tämä ei yleensä johda hyvään tulokseen. Koulutuksen aikana tavoitteena on aloittaa huonosti suoriutuvalla neuroverkolla ja päättyä verkkoon, joka saavuttaa parhaan tarkkuuden. Häviöfunktion, eli virheitä kuvaavan funktion, halutaan olevan paljon pienempi koulutuksen lopussa. Verkkoa voidaan parantaa muuttamalla sen painoarvoja. Tavoitteena on löytää funktio, joka suoriutuu paremmin kuin alkuperäinen [22].

On monia erilaisia algoritmeja, joita voidaan käyttää koulutuksen aikana. Algoritmit voivat olla gradienttipohjaisia, riippuen käyttävätkö ne vain funktion informaatiota vai myös gradientin. Yleinen gradienttipohjainen algoritmi on backpropagation-algoritmi. Kyseinen algoritmi kouluttaa neuroverkkoa käyttäen menetelmää nimeltä ketjusääntö. Jokaisen vaiheen jälkeen verkko suorittaa

takaisinsyötön ja samalla säätää mallin parametrejä, kuten painoarvoja ja vääristymiä [23].

2.4 Vahvistusoppiminen

Vahvistusoppiminen (engl. Reinforcement Learning) sisältää agentin, joka tutkii ennestään tuntematonta ympäristöä saavuttaakseen asetetun tavoitteensa. Agentin lisäksi pääelementteihin kuuluu ympäristö, jossa agentti toimii, menettelytapa, jolla agentti valitsee toimintonsa ja palkkiosignaali. Palkkiosignaalin agentti saa tehdessään toimintonsa loppuun. Koneoppiminen perustuu hypoteesiin, että tavoitteet voidaan kuvata odotetun kumulatiivisen palkkion maksimoinnilla. Agentin on tarkoitus ymmärtää ja oppia ympäristöä käyttämällä erilaisia asetettuja toimintoja saavuttaakseen parhaimman palkkion [24]. Agentti toimii päätöksentekijänä, joka käyttää saatavilla olevaa palautetta ja tuntemattomia vaihtoehtoja parantaakseen suoritustaan. Sen tavoitteena on maksimoida positiivinen palaute ja minimoida negatiivinen palaute. Kuvassa 9 kuvataan, kuinka agentti tekee päätöksiä kokeilemalla erilaisia vaihtoehtoja ja oppimalla niiden seurauksista. Se tekee päätökset ilman kokonaiskuvan tietoisuutta, keskittyen pelkästään parhaan palautteen saavuttamiseen.



Kuva 9. agentin päätöksentekosilmukka [25].

Kuvassa 9 näkyy toimintalogiikka vahvistuskoneoppimiselle. Agentti tekee toiminnan, josta se saa palautetta. Tämän jälkeen agentille annetaan uusi ympäristön tila, jossa se aloittaa alusta.

3 Projektin työkalut ja parametrien säätäminen

Projektin ideana oli saada aikaan yksinkertainen ampumissimulaatio, jossa agentti yrittää suoriutua mahdollisimman hyvin vihollisia vastaan. Projektin kehityksessä käytettiin Unitya ja Unityn ilmaista kirjastoa ML-Agents. Työn toimivuuden kannalta käytettiin Unity versiota 2022.3.14f1 ja Pythonista versiota 3.9.13.

3.1 ML-Agents

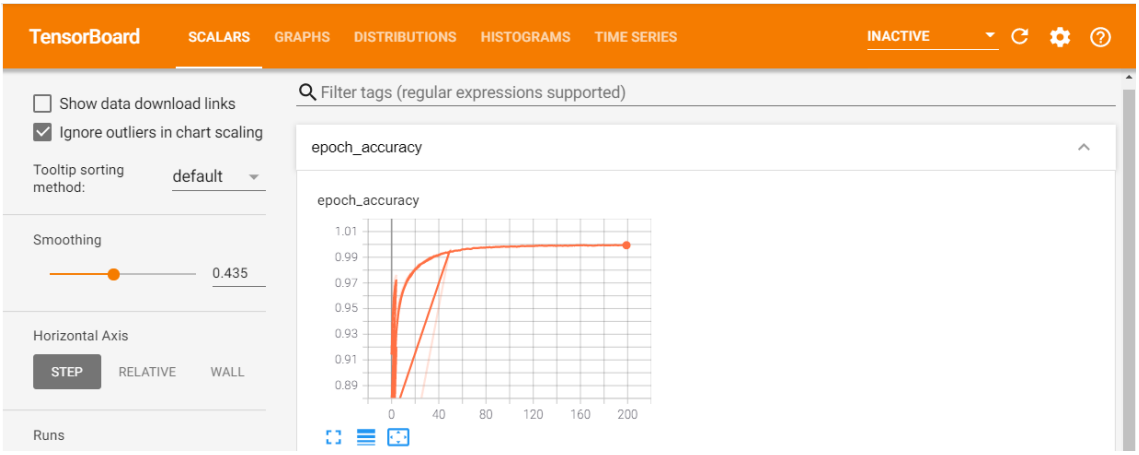
Insinöörityön projektin pohjana toimi Unityn ML-Agents-kirjasto. Se on avoin lähdekoodi, joka mahdollistaa pelien toiminnan ympäristönä älyllisten agenttien luomiseen. ML-Agents-kirjastoa käytetään vahvistusoppimisessa Unity-pelimoottorissa. PyTorch-kirjastoon perustuva algoritmi mahdollistaa erilaisten älyllisten agenttien luomisen 2D-, 3D-, VR- ja AR-peleihin [26]. Itse ML-Agents-kirjaston käyttäminen on yksinkertaista, ja alkuun pääsee, vaikkei tietäisikään aiheesta.

Projekti tehtiin Unity-pelimoottorilla, johon ML-Agents on rakennettu. Unity on vuonna 2005 Unity Technologiesin kehittämä monialustainen pelimoottori, joka on ilmainen harrastajille ja opiskelijoille. Alkuperin Unity julkaistiin Mac OS X:n pelimoottoriksi, mutta julkaisun jälkeen se on levinnyt toisille alustoille kuten puhelimille, pöytätietokoneisiin ja VR, sekä AR:aan [27]. Moottori pystyy luomaan 2D- ja 3D-pelejä sekä erilaisia interaktiivisia kokemuksia. Unitya käytetään myös peliteollisuuden ulkopuolella, kuten arkkitehtuurissa, rakennus- ja elokuva-alalla.

ML-Agentsin asentaminen on helppoa, mutta eri versioiden kanssa voi nopeasti tulla ongelmia, jos ei ole aiempaa kokemusta. ML-Agentsien lisäksi täytyy asentaa Python, ja Unityyn unity.ml-agents-paketti. Pythonin ja Unityn versioiden kanssa voi tulla ongelmia, jotka yleensä johtuvat liian varhaisesta Python-versiosta. Sen lisäksi Python ilmoittaa nopeasti, jos joissain kirjastoissa on vääränlaisia versioita. Projektia varten on suositeltavaa luoda oma virtuaaliympäristö, jotta projektin tiedostot ja erilaiset asennukset pysyvät sen sisällä. Virtuaaliympäristön avulla vältetään tulevaisuudessa ongelmia muiden projektien kanssa.

ML-Agentsien lisäksi tärkeä kirjasto on TensorFlow, joka mahdollistaa koulutetun agentin tietojen ja kehittymisen tarkkailun. Googlen Google Brain -tiimin kehittämä TensorFlow-kirjasto käyttää tietovirtakaavoja tulosten laskentaan. TensorFlow pystyy mallin koulutuksen päätteeksi tallentamaan neuroverkon TensorFlow.nn-tiedostoon, jota voi myöhemmin käyttää Unityssa agentin käyttäytymisen ohjausta varten [28].

TensorBoard on kätevä työkalu TensorFlow’n koulutuksen visualisointiin. Se hyödyntää TensorFlow’n tallentamaa dataa ja esittää sen graafisesti erilaisten kuvaajien avulla. Näin käyttäjä voi helposti seurata ja vertailla koulutuksen etenemistä, saadun palautteen määrää ja koulutusjaksojen pituuksia. Kuvassa 10 havaitaan esimerkki siitä, miltä sivusto näyttää ja miten tieto on käyttäjälle ilmaistu kaavion avulla.



Kuva 10. TensorBoard kaavio [29].

TensorBoard säilyttää ja näyttää kaikkien koulutuksien tulokset, joita käyttäjä on käynyt läpi. Ne tallentuvat suoraan käyttäjän tietokoneelle, josta TensorBoard lukee datan ja tekee siitä omat havainnollistavat kaavat. TensorBoardissa seuraavat kaaviot on hyvä huomioida mallia kouluttaessa:

1. Kumulatiivinen palkkio (Cumulative Reward) kertoo agenttien keskiarvoisen palkkion per jakso, jonka pitäisi nousta onnistuneen kouluttautumisen aikana.

2. Ympäristön tai jakson pituus (Environment/Episode Lenght) on keskiarvo jokaisen agentin jakson pituudesta.

3. Häviö tai käytännön häviö (Losses/Policy Loss) on käytännön menetysfunktion keskimääräinen suuruus. Kertoo siitä kuinka paljon käytäntö, toimintojen päätös, muuttuu joka jaksossa. Arvon pitäisi laskea onnistuneen kouluttamisen aikana.

4. Oppimisnopeus (Learning Rate) kertoo, kuinka suuria askeleita neuroverkkoa koulutetaan. Graafin pitäisi laskea koulutuksen myötä.

5. Haje (Entropy) kertoo käyttäjälle kuinka satunnaisia mallin päätökset ovat. Tuloksen pitää laskea hitaasti onnistuneen koulutuksen aikana. Jos tulos laskee liian nopeasti, beeta-hyperparametrejä täytyy suurentaa.

Tensorboardia voi myös seurata koulutuksen aikana, jolloin käyttäjä näkee heti jos ohjelmassa tai mallissa on ongelmia.

3.2 Hyperparametrit

ML-Agents käyttää vahvistusoppimisen menetelmää nimeltä Proximal Policy Optimization (PPO). Tämä tekniikka hyödyntää neuroverkkoa, joka arvioi optimaalisen toimintatavan agentille tietyssä tilanteessa. PPO-algoritmi on toteutettu TensorFlow’ssa, ja se suoritetaan erillisessä Python-prosessissa, joka kommunikoi aktiivisen Unity-sovelluksen kanssa socket-yhteyden välityksellä

[30]. Tärkeimpiä hyperparametrejä, jotka vaikuttavat koulutustulokseen ovat esimerkiksi buffer_size, batch_size, learning_rate, time_horizon, max_steps, ja beta parametrit. Buffersize määrittelee, kuinka paljon dataa (agentin havaintoja, tekoja ja saatuja palkintoja) kerätään, ennen kuin aloitetaan mallin koulutus tai päivitys. Yleensä tämä arvo on batch_sizea suurempi. Suurempi buffer_size yleensä tuottaa vakaampia tuloksia. Buffer_sizen tyypillinen skaala on 2 048–409 600.

Batch_size kuvaa, kuinka monta koulutusaineiston havaintoa kerrallaan algoritmi käyttää neuroverkon päivitykseen. Tämän arvon tulisi olla aina pienempi kuin buffer_sizen. Jatkuvassa toimintatilassa käytettäessä tämän arvon tulisi olla suuri (noin 1 000), kun taas erillisessä toimintatilassa se voi olla pienempi (noin 10). Tyypillinen skaala bazh-sizelle on 512–5 120 tai 32–512.

Learning_rate määrittää gradienttien päivityksen voimakkuuden jokaisessa koulutusvaiheessa. Jos harjoittelu on epävakaata, eikä palkkio jatkuvasti kasva, tätä arvoa tulisi pienentää. Tyypillinen skaala learning_ratelle on 1e–5 - 1e–3.

Time_horizonin avulla voidaan muuttaa, kuinka monta vaihetta kokemusta kerätään jokaiselta agentilta, ennen kuin kerätty tieto lisätään kokemuspuskuriin. Jos tämä raja saavutetaan ennen jakson päättymistä, arvoestimaattoria käytetään ennustamaan odotettu kokonaispalkkio agentin nykyisestä tilasta. Jos jaksoissa on usein palkintoja tai ne ovat pitkiä, pienempi arvo voi olla suotuisampi. Time_Horizonin tulisi kuitenkin olla riittävän suuri, jotta se kattaa kaikki tärkeät toimet agentin toimintosarjassa. Time_horizonin tyypillinen skaala on 32–2 048.

Max_steps määrittää, kuinka monta simulaation vaihetta suoritetaan koulutuksen aikana. Tämän arvon tulisi olla korkeampi monimutkaisissa ongelmissa, jotta malli saa riittävästi tietoa ympäristöstä ja pystyy oppimaan tehokkaasti. Max_stepsin tyypillinen skaala on 5e5–1e7.

Betalla säädetään entropian vahvuutta, mikä vaikuttaa toimintojen satunnaisuuteen. Tämä arvo tulisi säätää niin, että entropia, jota voi seurata TensorBoardin

avulla, vähenee koulutuksen aikana tasaisesti ja hitaasti. Betan skaala on yleensä 1e-4–1e-2.

Koulutuksen onnistuminen riippuu suuresti hyperparametrien oikeasta määrittelystä, joten hyperparametrien asettaminen ennen koulutuksen aloittamista on tärkeää. Tarvittaessa niitä kannattaa myös säätää, jos koulutus ei etene halutulla tavalla. Aloittelijan on hyvä etsiä samanlaatuinen malli ja katsoa, millaisia hyperparametrejä samankaltaisissa tilanteissa on käytetty.

Konfiguraatiotiedostossa on määriteltävä vähintään yksi palkintosignaali. Yleisin näistä on ulkoinen palkintosignaali, joka antaa palautetta agentille sen tekemien päätösten perusteella, kuten esimerkiksi maaliin pääsemisestä. Lisäksi voidaan määritellä uteliaisuus- ja GAIL (Generative Adversarial Imitation Learning) -palkintosignaali [31].

Uteliaisuuspalkintosignaali voi olla hyödyllinen ympäristöissä, joissa palautetta annetaan vähän tai harvoin. Se kannustaa agenttia kokeilemaan erilaisia toimintoja antamalla palautetta sen perusteella, kuinka paljon toiminto eroaa sen normaalista toiminnasta. GAIL-palkintosignaali puolestaan hyödyntää tallennettua demonstraatiotiedostoa koulutuksessa, jossa tallennettuja toimintoja pyritään jäljittelemään.

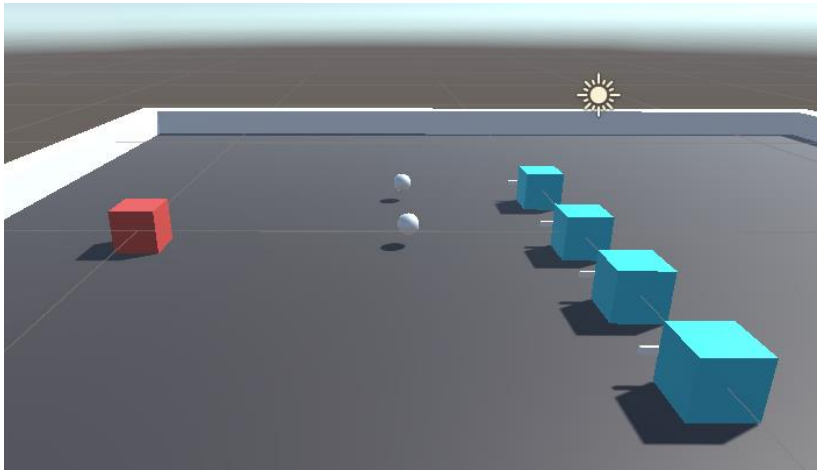
4 Neuroverkon kehittäminen ampumapelissä

Ensimmäinen vaihe insinöörityössä oli asentaa tarvittavat työkalut ja ohjelmistot. Asennuksessa on mukaan luettuna Unity-pelimoottori, sekä Python ja sen monet eri kirjastot. Tämä oli työn ensimmäinen haaste, sillä versioiden yhteensopivuus ei ollut itsestäänselvyys. Eri versioiden testaamisen ja lataamisen jälkeen, onnistuttiin lataamaan kaikki tarvittavat kirjastot oikeilla versioilla. Aluksi harjoiteltiin kahden erilaisen idean väliltä, jotka olivat yksinkertainen pisteen luokse kulkeva malli ja objekteja siirtävä malli. Lopuksi perehdyttiin malliin, jota voitaisiin mahdollisesti käyttää myös tulevaisuuden peliprojekteissa. Lopputuloksena saatiin hahmo, joka osaa tarkasti ampua vihollisia.

4.1 Suunnittelu ja ympäristön luonti

Projektia varten ei tehty erillisiä 3D-malleja, vaan käytettiin Unitysta löytyviä neliön muotoisia objekteja, joille annettiin erilainen materiaali vihollisten ja agentin erottamiseksi. Agentti on sininen, kun taas sen viholliset ovat punaisia. Pieni valkoinen pallo asetettiin projektiiliksi. Projektiilia varten agentille lisättiin pieni ampumakohta, josta se lähtee eteenpäin. Pelikentäksi luotiin ensimmäisenä yksinkertainen alusta, jonka reunoille asetettiin seinät. Seinät estivät hahmojen joutumista tarkoitetun testialustan ulkopuolelle. Myöhemmin lisättiin enemmän seiniä ympäri pelikenttää, mikä teki koulutuksesta mielenkiintoisemman kokonaisuuden. Koulutuksen aikana Debug.Log-funktiota käytettiin paljon koodin purkamista varten.

Kaiken tarvittavan ollessa pelikentällä oli aika suunnitella, mitä agentin tulee tehdä. Yksinkertaisessa ampumapelissä tärkeintä on se, että agentti osaa ampua. Ensimmäisenä täytyi siis lisätä koodi, jolla agentti alkaa itsenäisesti ampumaan. Tämä oli myös ensimmäinen kerta, kun ML-Agents-toiminto aloitettiin ja sen toimivuutta päästiin testaamaan. Kun huomattiin, että agentti osaa ampua eteenpäin, lisättiin liikkuvuutta. X- ja Z-akseleille laitettiin eteenpäin liikkuminen, ja Y-akselille ympäripyöriminen. Unityn puolella joudutaan lukitsemaan agentin Y-positio ja X - ja Y-rotaatio, koska agentti pyörähti liian helposti sivulle ilman rajoituksia. Näiden muutosten jälkeen testattiin toimivuus, jossa ei tähän asti ollut mitään ongelmia. Kuvassa 11 näkyy, että agentit osasivat ampua projektiilin. Tässä kohtaa yhdessä kentässä oli monta agenttia, mikä muutettiin myöhemmin.



Kuva 11. projektin ensimmäinen koulutuskerta.

Esimerkkikoodissa 1 näkyy, että alussa agentille annettiin pisteitä ampumisesta AddReward()-funktion avulla, jotta se osaisi tehdä sille tärkeimmän tehtävän, eli ampumisen. Loppukoodissa palkintoja tullaan muuttamaan jonkin verran.

```
    private void Shoot()  {
        if (!ShotAvaliable)
            return;

        var layerMask = 1 << LayerMask.NameToLayer("En-
emy");
        var direction = transform.forward;

        // Luoti
        var spawnedProjectile = Instantiate(projectilePre-
fab, shootPoint.position, Quaternion.Euler(0f, -90f, 0f));
        spawnedProjectile.SetDirection(direction);

        if (Physics.Raycast(shootPoint.position, direction,
out var hit, 200f, layerMask)){
            hit.transform.GetComponent<En-
emy>().GetShot(damage, this);
            Debug.Log("Hit enemy");}
        else {
            AddReward(1.0f);
            Debug.Log("Missed enemy");}

        // Lisätään tauko, jotta agentti ei ammu koko ajan
        ShotAvaliable = false;
        StepsUntilShotIsAvailable = minStepsBetweenShots;}
```

Esimerkkikoodi 1. Agentin ampumisfunktio.

Seuraavaksi lisättiin vihollinen, joka aluksi oli vain paikoillaan. Oli tärkeää ensin saada agentti ampumaan oikeaa päämäärää ja nähdä, ettei suurempia ongelmia tullut tässä kohtaa. Myöhemmin vihollisia lisättiin ja niille annettiin myös päämäärä liikkua agenttia kohti, jotta agentilla olisi syytä ampua vihollisia tarpeeksi ajoissa.

Taulukko 1 kuvastaa loppupalkkioita, eli kannustinta. Niitä muutettiin projektin ja koulutuksen aikana useasti. Tärkein muutos oli, että agentti saa suuremman palkkion jokaisesta päihitetystä vihollisesta mitä vähemmän vihollisia oli jäljellä. Muuten agentti helposti tyytyi vain yhden päihittämiseen, ennen kuin se saatiin kiinni. Lisää eri palkkioiden vaikutuksesta 4.2-luvussa, jossa käydään koulutusta tarkemmin läpi.

Taulukko 1. Agentin palkkiot.

Agentti ampuu jotain muuta kuin vihollista	-0.050f
Kannustin, ettei agentti lopeta ampumista kokonaan	-1.0f / aika viime kerrasta, kun agentti ampui
Agentti ampuu vihollista	1.0f / vihollisten määrä jäljellä
Agentti törmää viholliseen	-1.0f
Agentti törmää seinään	-0.5f

Esimerkkikoodissa 2 näkyy selvästi kaikki agentin toiminnot. Projektin lopussa sen täytyi ampua, liikkua eteenpäin ja osata pyöriä itsensä ympäri.

```
  public override void OnActionReceived(ActionBuffers ac-
tions)
    {
        float shoot = actions.ContinuousActions[0];
        float movex = actions.ContinuousActions[1];
        float movey = actions.ContinuousActions[2];
        float movez = actions.ContinuousActions[3];

        if (Mathf.RoundToInt(shoot) >= 1)
        {
            Shoot();
        }

        Rb.velocity = new Vector3(movex * speed, 0f, movey
* speed);
        transform.Rotate(Vector3.up, movez * rotation-
Speed);}
```

Esimerkkikoodi 2. Agentin toiminnot.

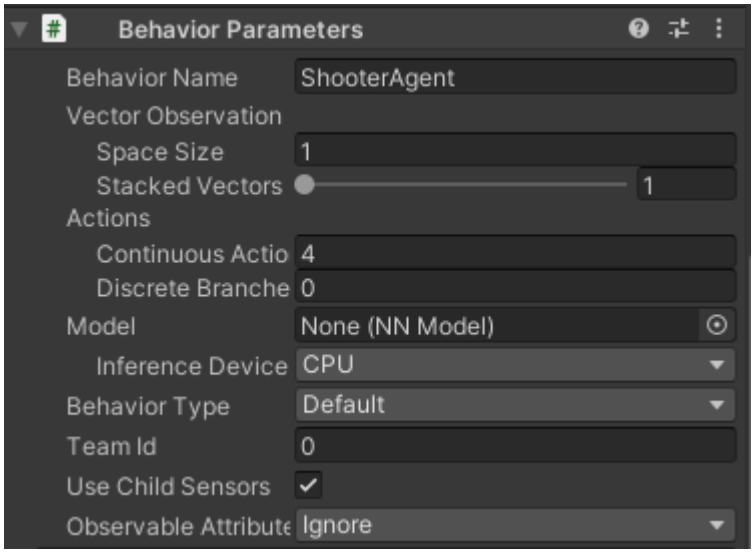
4.2 Agentti

Agenttia ohjattiin skriptin avulla, joka liikuttaa hahmoa ympäri kenttää ja saa sen ampumaan eteensä. Skriptissä on mahdollista säätää agentin ampumis-, pyörimis- ja liikkumisnopeutta. Projektiilin vahinkomäärää pystyttiin myös säätämään, mikä aluksi asetettiin päihittämään viholliset yhdellä kerralla. Vaikeustasoa koulutuksessa olisi voinut nostaa tekemällä vihollisista hieman vahvempia, mutta tärkeintä oli saada perustoiminnot agentille toimiviksi. Agentille asetetiin Max step-arvo, eli kuinka monta toimintoa agentti tekee, ennen kuin jakso aloitetaan alusta. Tällä vältetään se, ettei agentti jää jumiin, jos koulutuksessa tulee jokin ongelma tai odottamaton vaihe. Liian pieni luku tarkoittaa myös sitä, että agentti ei välttämättä ehdi tehdä toimintojaan loppuun. Tässä tapauksessa Max step asetettiin 10 000 yksikköön, liikkumisnopeus 4 yksikköön ja pyörimisnopeus 3 yksikköön.

Agentille lisättiin Box Collider- ja RigidBody-komponentit, jotta se osaa kulkea oikealla tavalla, eikä leikkaudu esimerkiksi pelikentän tai seinien läpi. Sen lisäksi agentti sai Behavior Parameters -komponentin, jolla voi säätää agentin käytösparametreja. Tärkein käytösparametri asettaa on Actions, jossa merkitään, kuinka monta erilaista toimintoa agentilla on. Ampumisagentilla on neljä

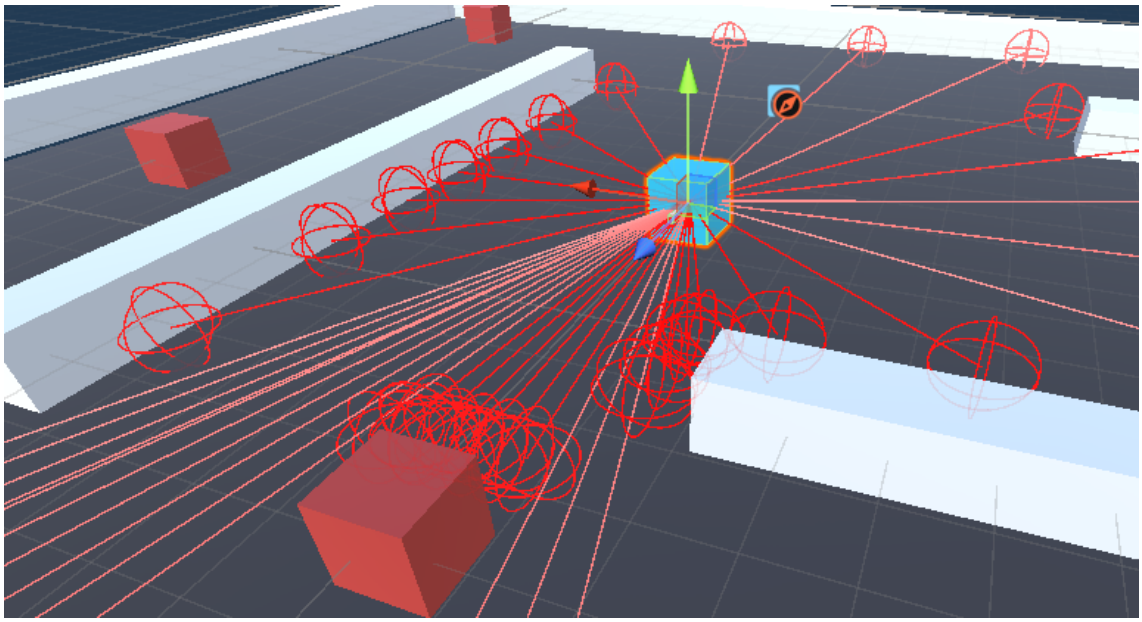
toimintoa: ampuminen, liikkuminen X- ja Z-akseleilla ja pyöriminen Y-akselilla. Käytösparametreihin voidaan myös asettaa Tensorflow'n luoma nn.-tiedosto-malli, jolla koulutettu käytös saadaan asennettua agentille.

Behavios Parameters -komponentissa annetaan agentin käytösnimi ja kerrotaan, kuinka monta asiaa agentti havaitsee. Lisäksi siinä määritellään, kuinka monta ja millaisia toimintoja agentti voi suorittaa. Agentille voi antaa valmiin koneoppimismallin tiedostomuodossa (.nn-tiedosto). Valmiit koneoppimismallit saadaan tuloksena koulutuksen päättyessä. Kuvassa 12 näkyy, että projektin agentti tarvitsee neljää jatkuvaa toimintoa: ampuminen, liikkuminen x- ja z-akseleilla ja pyöriminen y-akselilla.



Kuva 12. Agentin Behavior Parameters -komponentti.

Agentille asetetaan kaksi Ray Perception Sensor 3D -komponenttia, jotka lähettävät säteitä ja osuvat eri objekteihin pelikentällä. Säteiden avulla agentti osaa tunnistaa eri kohteet edessään, kuten viholliset ja seinät. Ensimmäinen komponentti lähettää tiiviissä rivissä säteitä suoraan eteen, jossa agentti tarvitsee eniten tarkkuutta ampuessaan vihollisia. Toinen komponentti lähettää säteitä harvemmassa taakse ja sivulle, jotta agentti osaa arvioida myös lähestyvät viholliset ja reagoida niihin oikeassa järjestyksessä. Kuvassa 13 näkyy säteiden määrä ja suunnat.



Kuva 13. Agentin Ray Perception Sensor -komponentti toiminnassa.

Vihollisilla oli alussa hyvin yksinkertaiset funktiot. Agentti osuu niihin projektiilillaan, jolloin ne katoavat, ja jakso alkaa alusta. Myöhemmin vihollisille annettiin kyky liikkua agenttia kohti, jolloin niiden täytyi myös tietää, missä ne saavat liikkua. Varsinkin seiniä lisättäessä pelikenttään tämä oli tärkeää rajata. Vihollisille annettiin Nav Mesh Agent -komponentti, ja pelikenttään lisättiin tyhjään objektiin NavMeshSurface-komponentti. NavMeshSurface-komponentilla saadaan leivottua (bake) pelikenttään kerros, josta vihollinen näkee, missä se voi kulkea.

Uuden alueen avulla pystytään tuomaan vaihtelua agentin koulutukseen. Jakson alettua alusta vihollinen ilmestyy uudelle paikalle sallitun alueen sisällä. Tämä saadaan aikaan satunnaisilla, positiivisilla ja negatiivisilla X- ja Z-akselilukemilla. Vihollisen Enemy-skriptissä oleva funktio luo uuden satunnaisen paikan, jolloin agentti ei päädy ampumaan vain tiettyä paikkaa tietyssä järjestyksessä. Agentti oppii ampumaan lähimpänä näkyvää vihollista. Esimerkkikoodissa 3 näkyy vihollisten satunnaisfunktio. Ensin kaikki viholliset resetoidaan, jotta ongelmatilanteita ei tulisi. Sen jälkeen jokainen vihollinen asetetaan uudelle aloituspaikalleen.

```
    public void SetEnemiesActive()
    {
        int counter = 0;
        EnemyCount = Mathf.FloorToInt(EnvironmentParame-
ters.GetWithDefault("amountZombies", 4f));

        startingPoint = Mathf.FloorToInt(Random.Range(0f,
enemies.Length - EnemyCount));

        foreach (var enemy2 in enemies)
        {
            enemy2.gameObject.SetActive(false);
        }

        for (int i = startingPoint; i < EnemyCount + start-
ingPoint; i++)
        {
            counter++;
            enemies[i].gameObject.SetActive(true);
        }
     }
```

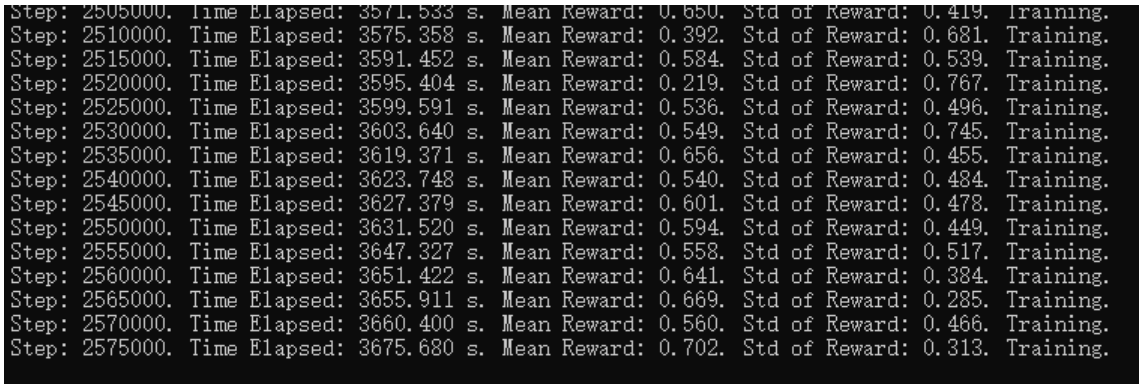
Esimerkkikoodi 3. Vihollisten aktivointifunktio.

4.3 Koulutus

Projektissa käytettiin koneoppimismenetelmänä vahvistusoppimista, jonka avulla agentti koulutettiin. Ympäristön ollessa haluttu ja agentin sekä vihollisten toimintojen toimiessa oikealla tavalla, oli aika aloittaa mallin koulutus. Koulutus aloitettiin avaamalla Windows-komentorivi ja syöttämällä komento mlagents-learn <configfile.name> --run-id=<runidname>. Configfile.name kohtaan asetetaan aiemmin luodun ja määritetyn hyperparametrien yaml-konfiguraatiotiedoston nimi. Runidname on ainutlaatuinen avain jokaiselle tallennetulle koulutuskerralle, jotta koulutusta voi tarkkailla erikseen TensoBoardin kautta.

Koulutuksen alkaessa komentoikkunassa ilmoitetaan, jos jokin on menee pieleen. Jos kaikki toimii odotetulla tavalla, Unityn puolelta saa koulutuksen käynnistettyä play-nappia painamalla. Koulutuksen ollessa päällä käyttäjä voi seurata agentin kehittymistä Unityn kautta ja nähdä miten se pärjää joka jaksossa. Kamera on keskitetty vain yhden agentin seuraamiseen, mutta lopullisessa koulutuksessa oli 20 kopiota kentästä, jossa eri agentit kouluttautuvat

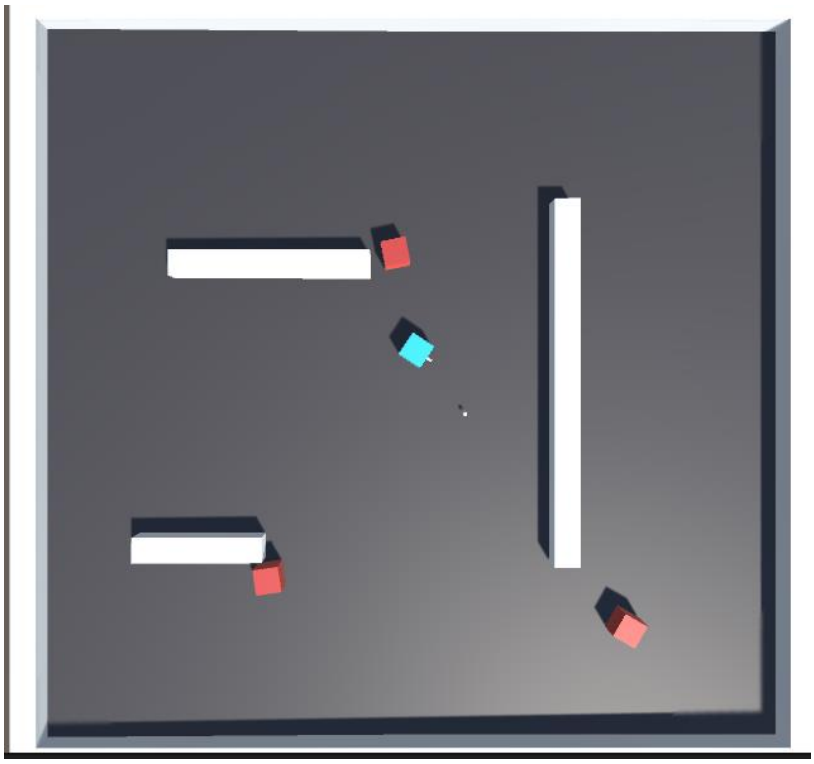
samanaikaisesti. Kuvassa 14 nähdään koulutuksen aikana komentoikkunasta näkyvät tulosrivit. Mitä korkeammalle Mean Reward eli palkkion keskiarvo nousee, sitä paremmin agentti toimii halutulla tavalla.



Kuva 14. Komentorivi agentin koulutuksen aikana.

TensoBoardista löytyy myös koulutuksen etenemisen tarkkailun kannalta käteviä taulukkoja ja graafeja. Ne kertovat kaiken informaation mitä kuvassa 14 näytetään, mutta visualisoituna käyttäjälle, jotta tulokset olisivat helppolukuisempia. TensorBoardin tuloksia tarkkaillaan tarkemmin luvussa 5.

Kuvassa 15 näkyy lintuperspektiivistä yhden agentin suoritus koulutuksen aikana. Alun perin kentällä on ollut neljä vihollista, joten agentti on onnistunut päihittämään yhden lähestyvistä vihollisista. Seuraava projektiili näkyy menevän ohi. Agentti ei siis selvästi ole vielä täydellinen. Se kuitenkin ymmärtää paremmin jo tässä kohtaa, mitä sen kuuluu tehdä saadakseen parhaan mahdollisen palkkion.



Kuva 15. Lintuperspektiivi agentin kouluttamisesta.

Koulutus loppuu joko kun max_steps-arvo, joka on määritetty hyperparametreissä, tulee täyteen tai kun käyttäjä manuaalisesti lopettaa koulutuksen komentorivillä tehdyn ctrl + left shift -komennolla. Koulutuksen jälkeen TensorFlow tallentaa koulutuksen tulokset ja mallin. Tallennetun nn.-tiedoston voi siirtää Unityyn ja asettaa agentille Behavious Parameters -komponenttiin. Tällöin agentti saa itselleen koulutetun tekoälymallin ja pelin alkaessa tekee oppimiaan toimintoja.

5 Projektin arviointi

Vaikeinta projektissa oli päästä alkuun asennuksien kanssa. Sen lisäksi hyperparametrien ja palkintojen muuttamisessa tuli omanlaisia ongelmia. Suurin ongelma tuli siitä, ettei mallin annettu kouluttautua tarpeeksi pitkään. Alussa agentti kouluttautui noin 20 minuuttia, mutta myöhemmin koulutusaikaa kasvatettiin muutamasta tunnista jopa puoleen päivään. Muutama tunti olisi riittänyt, mutta oli mielenkiintoista nähdä, jos malli parantuisi esimerkiksi kolmen tunnin

koulutuksen jälkeen. Huomattiin kuitenkin, ettei malli kahden tunnin jälkeen kehittynyt.

5.1 Ongelmatilanteet

Insinööriprojekti oli täynnä erilaisia ongelmatilanteita, jotka onneksi saatiin ratkaistua. Ensimmäiset ongelmat tulivat positiivisen ja negatiivisen palkkion tasapainottamisesta. Palkkioita täytyi muuttaa heti projektin alussa, kun agentille annettiin liikaa miinusta esimerkiksi ohi ampumisesta. Liian suuren negatiivisen palautteen seurauksena agentti alkoi pyöriä itsensä ympäri ja päätti olla tekemättä mitään. Tämän jälkeen lisättiin positiivista palkkiota ampumisesta agentille. Seurauksena agentti päätyi ampumaan joka suuntaan, sen sijaan että osaisi ampua vihollista kohti. Tässä kohtaa täytyi etsiä tasapainoa agentin ohi ampumisen rangaistuksille ja vihollisen päihittämisen palkkiolle. Lopputuloksena ohi ampumisen suora miinuksen antaminen poistettiin ja miinusta lisättiin kumulatiivisesti, jos agentti ei ampunut pitkään aikaan. Tämä tuntui antavan tarpeeksi tasapainoa ampumiseen.

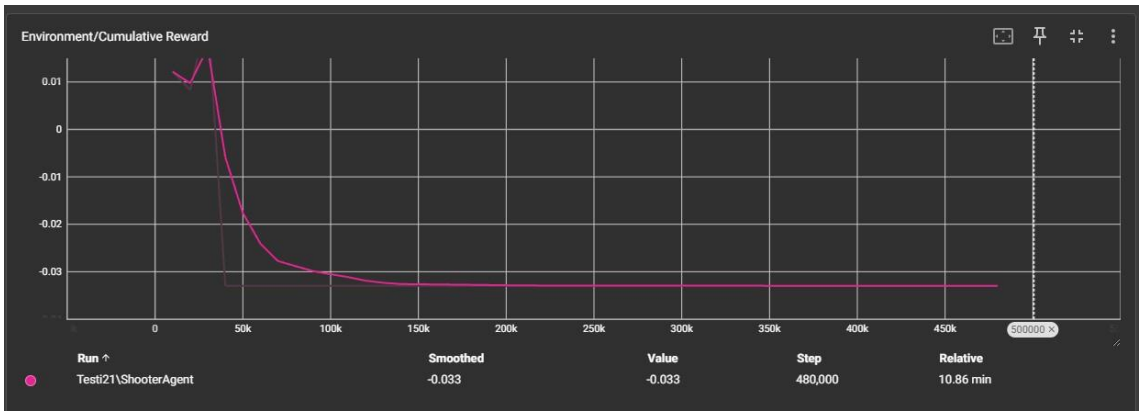
Hyperparametreillä oli iso vaikutus koulutuksen suorituksessa. Niitä jouduttiin säätämään usein ja yksi kerrallaan, jotta ongelmakohta löytyisi. Yksi tärkeimmistä parametreista, joka muutti huomattavasti agentin kouluttamista, oli curiosity, eli uteliaisuusparametri. Uteliaisuutta lisäämällä sai agentin kokeilemaan asioita, joita se ei välttämättä ensin olisi tajunnut. Tämän myötä agentti löysi parempia ja tehokkaampia tapoja saada maksimaalinen palkkio.

Uteliaisuusparametrin lisäksi muitakin hyperparametrejä jouduttiin säätämään. Alla lista parametreistä, jotka kokivat isoimmat muutokset ja joiden muuttaminen paransi koulutuksen tuloksia:

- batch_size: 10 → 2 048.
- buffer_size: 100 → 20 480.
- beta: 5.0e-4 → 1e-2.
- learning_rate_schedule: linear → constant.
- hidden_units: 128 → 512.
- num_layers: 2 → 3.
- time_horizon: 64 → 1 000.

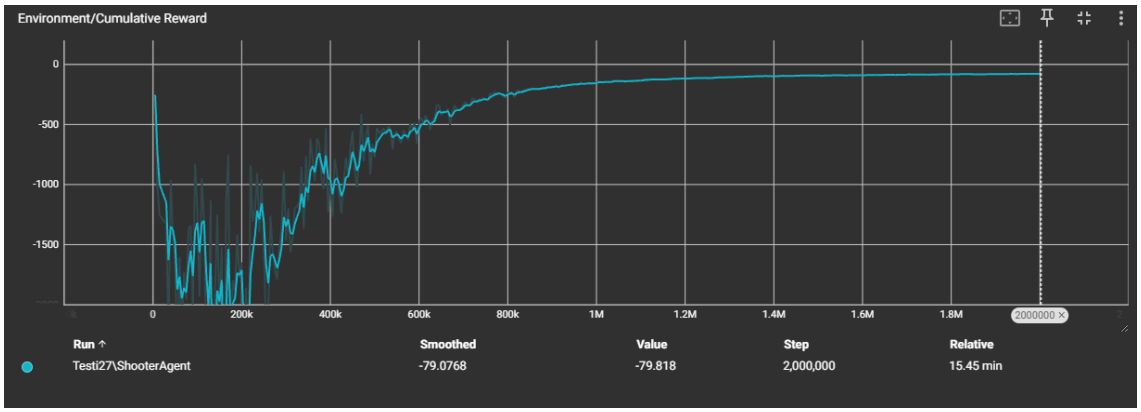
Alun perin hyperparametreissä batch_size oli hyvin pieni, mikä ei toimi hyvin, jos agentin toiminnot ovat jatkuvia. Tämän vuoksi sen arvo lisättiin tuhansiin. Buffer_sizen arvo, jonka täytyy olla kymmenkertainen batch_sizesta, muutettiin myös. Beta vastaa mallin hajetta, joka tekee koulutuksesta satunnaisempaa. Tämä varmistaa, että agentti kokeilee mahdollisimman monta vaihtoehtoa löytääkseen parhaan tavan toimia. Tätä pienentämällä huomattiin, että agentti oppi nopeammin oikeat toiminnot. Learning_rate, eli oppimisaste, lineaarisena tarkoittaa sitä, että sen arvo laskee koulutuksen aikana. Tämän muuttaminen jatkuvaksi auttoi tuloksia ja teki mallin kouluttamisesta vakaampaa. Hidden_units oli liian pieni agentin tarpeisiin.

TensorBoard auttoi huomattavasti datan analysoinnissa. Siitä näki helposti, miten hyvin malli kouluttautui. Kuvassa 16 näkyy ongelmatilanne, jolloin agentti lopetti ampumisen kokonaan liian ison rangaistuksen vuoksi.



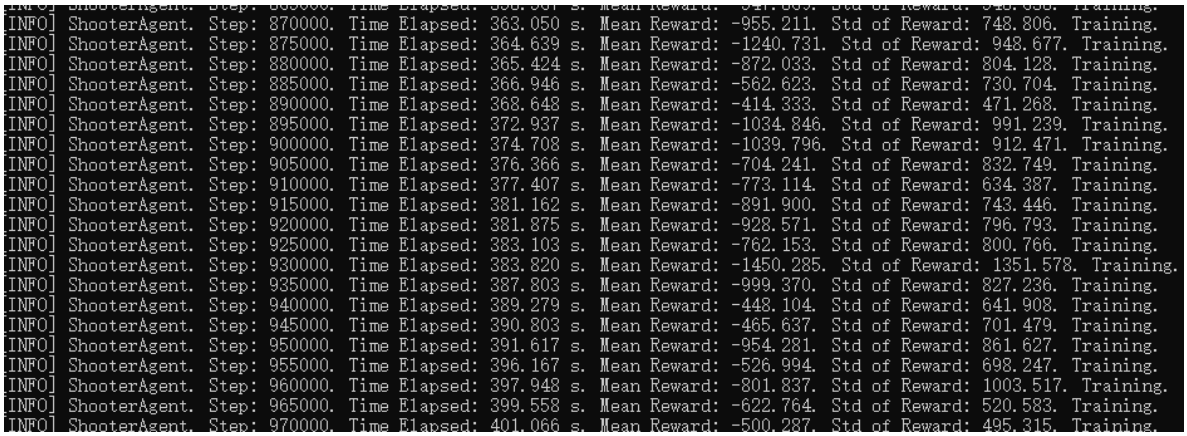
Kuva 16. TensorBoard kaavio, joka laskee jyrkästi.

Kuvassa 17 hahmottuu se, kuinka agentti kehittyi. Kaaviosta nähdään, miten kasvu päättyy yhtäkkiä. Syynä on luultavasti hyperparametrien tai palkkioiden säätöongelmat. Liian lyhyt 15 minuutin koulutusaika vaikutti myös agentin kehittymiseen. Koulutusaikaa lisättiin, jotta nähtäisiin, ettei mallin kehityksen puute johdu vain siitä. Koulutuksen nopeuttamiseksi usea agentti kouluttautui samaan aikaan. Projektissa käytettiin 20:tä agenttia omine kenttineen. Kun agenteilla on sama Behavior Parameters -komponentin määrittelemä Behavior Name, agentit keräsivät oppimisdatan samaan malliin. Pelikenttien lisääminen vaatii tietokoneelta paljon lisäsuorituskykyä, varsinkin keskusmuistista.



Kuva 17. TensorBoard kaavio, joka nousee jyrkästi.

Kouluttamisen alussa oli liian suuret palkkiot ja rangaistukset, jolloin kaavio näyttää hieman jyrkemmältä muutoksiensa kanssa. Kuvasta 18 näkyy, kuinka isot rangaistukset agentti sai projektin alussa verrattuna lopulliseen versioon, joka ilmenee kuvassa 14.



Kuva 18. Komentorivi koulutuksen aikana.

5.2 Loppuanalyysi

Projektin myötä opittiin, että agentti oppii paremmin koneoppimisessa positiivisen kuin negatiivisen palautteen avulla. Tämä huomattiin alussa, kun annettiin liian suurta negatiivista palautetta esimerkiksi ohi ampumisesta. Tämän vuoksi kannatti välttää liian suuren negatiivisen palautteen antamista ja antaa agentin etsiä parhaimmat tavat saada eniten positiivista palkkiota. Positiivisen palautteen kumulatiivinen lisäys jokaisen vihollisen päihittämisen jälkeen edisti koulutusta huomattavasti. Agentti oppi, että sen kannattaa päihittää mahdollisimman monta vihollista jokaisessa jaksossa sen sijaan, että se yrittäisi mahdollisimman nopeasti saada muutaman pisteen, ennen kuin viholliset saavat sen kiinni.

Insinööriprojektissa päästiin lopulta haluttuun tulokseen, vaikka jossain määrin agentin tekoälyssä on paranneltavaa. Agentti saatiin ampumaan tulevia vihollisia kohti, vaikka viholliset tulivat eri kulmilta ja suunnilta. Koulutuksen lopussa agentti onnistui päihittämään kaikki viholliset ongelmitta, ennen kuin ne ehtivät agentin luokse. Oppimiskokonaisuutena projektin teko oli hyvin antoisaa, vaikkakin aikaa vievää koneoppimisen perusasioita opeteltaessa. Kokonaisuudessaan tekoäly on hyvin mielenkiintoinen aihe ja nyt sitä on päästy tutkimaan pelien koneoppimisen kautta.

Hyperparametrit olivat yksi isoimpia kompastuskiviä, mikä oli yllättävää. Pienikin muutos niiden luvuissa muutti koko koulutuksen tuloksen ja itse ongelman

löytäminen oli haastavaa ja aikaa vievää. Tietoa parametrien säätämiseen löytyi niukasti, joten käyttäjän on parempi itse kokeilla kaikki mahdolliset yhdistelmät parasta lopputulosta varten.

Projektin agentin koulutuksessa käytettiin uteliaisuuspalkintosignaalia, joka toimi hyvin loppupeleissä. Jos projekti tehtäisiin uudestaan, uteliaisuuspalkintosignaalin sijaan olisi luultavasti käytetty GAIL-palkintosignaalia. GAILin avulla käyttäjä pystyy manuaalisesti luomaan pohjan agentin käytökselle, jota se yrittää imitoida koulutuksen aikana. Tämän pohjan luominen olisi nopeuttanut kouluttamista, mutta yksinkertaisen ampumisagentin kouluttamisessa onnistuttiin onneksi ilman.

Koneoppiminen on loistava työkalu peleihin, mutta se tuo myös omanlaiset haasteensa ja kysymyksensä. Jos agentin opettaa pelaamaan pelaajaa vastaan liian hyvin, peli tai sen sisäinen taistelu ei ole enää mieluisa kokemus. Päinvastainen tilanne on myös hyvin mahdollista, jolloin agentti tekee kummallisia ja huonoja päätöksiä. Koneoppimisen satunnaisuus on sen viehätys ja kompastuskivi samaan aikaan. Perinteistä videopelien NPC:n tekoälyä koneoppiminen ei välttämättä korvaa ikinä, mutta se voi tuoda omanlaisen hauskan puolen peleihin tulevaisuudessa. Yleisesti samankaltaisiin yksinkertaisiin tilanteisiin, kuten ampuva vihollinen tai ympäristöä valvova vartija videopelissä, kannattaa itse ohjelmoida haluamansa toimivuus. Työkaluna koneoppiminen on kuitenkin hyödyllistä oppia, ja se voi auttaa monimutkaisissa ongelmissa tulevaisuudessa.

Koneoppimisen opettelu vie aikaa. Se on kuitenkin todella hyvä taito pelikehittäjälle, sillä tekoäly lisää mahdollisuuksia luoda monimutkaisia ja mielenkiintoisia pelikokemuksia. Koneoppimisen kehitys ja resurssit lisääntyvät koko ajan. Sen lisäksi tekoälyn käyttö on yleistynyt huomattavasti pelikehityksen eri puolilla, kuten musiikissa ja taiteessa viime vuosien aikana. On mielenkiintoista nähdä, millainen vaikutus koneoppimisella on videopeleihin tulevaisuudessa.

6 Yhteenveto

Koneoppimista ei käytetä vielä kovin laajasti peliteollisuudessa, fanien tekemien bottien tai muiden projektien ulkopuolella. Koko ajan kehittyvien koneoppimiskirjastojen ja -työkalujen määrä kasvaa kuitenkin koko ajan, ja ne tuovat uusia mahdollisuuksia tulevaisuuden videopelikehitykseen. Unity pyrkii kehittämään ML-Agentseja, jotta niiden käyttö olisi vaivattomampaa kaikenlaisissa projekteissa. Tälläkin hetkellä koneoppimisen käyttöä ja tutkimista kannattaa harrastaa pelikehittäjänä, sillä se tuo aivan uudenlaisia tapoja kehittää älykkäitä malleja omiin peleihin.

Insinöörityössä luotiin ampumissimulaatio, jotta voidaan harjoitella ja testata koneoppimisen hyötyjä videopeleissä. Projekti pidettiin yksinkertaisena, jotta voitiin keskittyä mahdollisimman hyvän lopputuloksen saamiseen. Lopputuloksena saatiinkin toimiva tekoäly, joka osasi ampua vihollisia päin, vaikka niitä olisi satunnaisesti kentällä eri määriä.

Mallia olisi voinut kouluttaa vielä paremmaksikin, mutta rajaa on vaikea tällaisissa projekteissa asettaa. Tekoälyä olisi voinut esimerkiksi kouluttaa erilaisissa kentissä tai niihin olisi voinut lisätä vaikeampia vihollisia. Kouluttamisajan lisääminen auttaa myös huomattavasti. Itse agentin toiminnallisuuttakin olisi ollut hauska monipuolistaa, esimerkiksi lisäämällä mahdollisuuden väistää vihollisluoteja, joutua lataamaan asetta silloin tällöin tai muilla yleisillä pelimekaniikoilla. Tärkeintä on, että saatiin maltillinen idea toimimaan, jolloin voidaan tulevaisuudessa implementoida samankaltaisia mekaniikkoja.

Lähteet

1 Tom, M. Mitchell. 1997. Machine Learning. E-kirja. McGraw-Hill Science/Engineering/Math.

2 Sunila, Gollapudi. 2016. Practical Machine Learning. E-kirja. Packt Publishing Ltd.

3 Chrystal, R. China. 2023. Five machine learning types to know. Verkkoaineisto. IBM 12/2023. <https://www.ibm.com/blog/machine-learning-types/>. 20.12.2023. Luettu 13.3.2024.

4 Koneoppimisen lajit. Verkkoaineisto. Elements of AI. <https://course.elementsofai.com/fi/4/1>. Luettu 13.3.2024.

5 IBM Data and AI Team. 2023. 10 everyday machine learning use cases. Verkkoaineisto. IBM 10/2023. <https://www.ibm.com/blog/10-everyday-machine-learning-use-cases/>. 16.10.2023. Luettu 14.3.2024.

6 Ivan, Koswara; Mateo, Matijasevick: Kai, Hsien Boo; Ivan, Koswara; Mateo, Matijasevick & Kai, Hsien Boo. Nim. Verkkoaineisto. Brilliant. <https://brilliant.org/wiki/nim/>. Luettu 14.3.2024.

7 The Investopedia Team. 2023. The Turing Test: What Is It, What Can Pass It, and Limitations. Verkkoaineisto. Investopedia 7/2023. <https://www.investopedia.com/terms/t/turing-test.asp>. 31.7.2023. Luettu 4.5.2024.

8 Jose, J. Martinez. 2023. AI in video games: a historical evolution, from Search Trees to LLMs. Chapter 1: 1950–1980. Verkkoaineisto. Medium 11/2023. <https://medium.com/@jjmcarrascosa/ai-in-video-games-a-historical-evolution-from-search-trees-to-llms-chapter-1-1950-1980-f3b04d6e9dc8>. 4.11.2023. Luettu 14.3.2024.

9 Wikipedia. 2024. Machine learning in video games. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Machine_learning_in_video_games>. 14.1.2024. Luettu 4.5.2024.

10 Avcontentteam. 2024. Machine Learning and AI in Game Development in 2024. Verkkoaineisto. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2023/03/ml-and-ai-in-game-development/>. 6.2.2024. Luettu 15.3.2024.

11 Ian Carlos, Campbell. 2021. Intel is using machine learning to make GTA V look incredibly realistic. Verkkoaineisto. The Verge. <https://www.theverge.com/2021/5/12/22432945/intel-gta-v-realistic-machine-learning-cityscapes-dataset>. 13.5.2021. Luettu 20.3.2024.

12 Simon Parkin. 2014. No Man’s Sky: A Vast Game Crafted by Algorithms. Verkkoaineisto. MIT Technology Review. <https://www.technologyreview.com/2014/07/22/12940/no-mans-sky-a-vast-game-crafted-by-algorithms/>. 22.7.2014. Luettu 20.3.2024.

13 No Man's Sky Planet and Creature Creation Shown Off. 2016. Verkkoaineisto. Gamerant. <https://gamerant.com/no-mans-sky-planet-creature-creation/>. 19.10.2016. Luettu 20.3.2024.

14 Gornall, Joshua. 2021. How I made a Runescape bot. Verkkoaineisto. <https://joshuagornall.medium.com/how-i-made-a-runescape-bot-90248acae34>. 17.5.2021. Luettu 20.3.2024.

15 Drypsiak, Oleksandr. Human Neuron Structure. Verkkoaineisto. Shutterstock. <https://www.shutterstock.com/fi/image-vector/human-neuron-structure-brain-cell-illustration-2166870873>. 13.6.2022. Luettu 30.3.2024.

16 Subana, Shanmuganathan; Sandhya, Samarasinghe. 2016. Artificial Neural Network Modelling. E-kirja. Springer International Publishing.

17 What is a neural network? Verkkoaineisto. IBM. <https://www.ibm.com/topics/neural-networks>. Luettu 30.3.2024.

18 Illustration of an artificial neuron. Verkkoaineisto. ResearchGate. <https://www.researchgate.net/figure/Illustration-of-an-artificial-neuron_fig1_343829199>. Luettu 30.3.2024

19 Himanshi, Singh. 2023. Deep Learning 101: Beginners Guide to Neural Network. Verkkoaineisto. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/>. 27.7.2023. Luettu 30.3.2024.

20 ML | Underfitting and Overfitting in Machine Learning. 2024. Verkkoaineisto. Geeks for Geeks. <https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/>. 11.3.2024. Luettu 10.4.2024

21 Hossein, Ashtari. 2022. What Is a Neural Network? Definition, Working, Types, and Applications in 2022. Verkkoaineisto. Spiceworks. <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/>. 3.8.2022. Luettu 4.5.2024

22 Bushaev, Vitaly. 2017. How do we ‘train’ neural networks? Verkkoaineisto. Towards Data Science. <https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73>. 27.11.2017. Luettu 4.5.2024

23 Kostadinov, Simeon. 2019. Understanding Backpropagation Algorithm. Verkkoaineisto. Towards Data Science. <https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd>. 8.8.2019. Luettu 4.5.2024

24 What is Reinforcement Learning? Verkkoaineisto. Synopsys. <https://www.synopsys.com/ai/what-is-reinforcement-learning.html>. Luettu 10.4.2024.

25 Bhatt, Shweta. 2018. Reinforcement Learning 101. Verkkoaineisto. Towards Data Science. < https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292>. 19.3.2018. Luettu 10.4.2024.

26 ML-Agents. Verkkoaineisto. Unity Technologies. <https://github.com/Unity-Technologies/ml-agents>. 9.10.2023. Luettu 11.4.2024.

27 Unity (game engine). Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Unity_(game_engine)>. 7.4.2024. 12.4.2024

28 Tutorials. Verkkoaineisto. TensorFlow. <https://www.tensorflow.org/tutorials>. 19.9.2023. Luettu 11.4.2024.

29 Jyoti, Yadav. 2021. A Complete Guide to Tensorboard. Verkkoaineisto. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/04/a-complete-guide-to-tensorboard/>. 27.4.2021. Luettu 11.4.2024

30 Training PPO. Verkkoaineisto. Unity Technologies. <https://github.com/gzrjzcx/ML-agents/blob/master/docs/Training-PPO.md>. 19.10.2018. Luettu 12.4.2024

31 Training with Imitation Learning. 2019. Verkkoaineisto. Yosider. <https://github.com/yosider/ml-agents-1/blob/master/docs/Training-Imitation-Learning.md>. 12.12.2019. Luettu 12.4.2024