Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Markus Pietilä

Koneoppiminen tasohyppelypelissä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

1.12.2019

Tekijä Otsikko Sivumäärä Aika	Markus Pietilä Koneoppiminen tasohyppelypelissä 34 sivua 1.12.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Pelisovellukset
Ohjaaja	Lehtori Miikka Mäki-Uuro

Insinöörityön päätarkoituksena oli selvittää koneoppimisen soveltuvuutta 3D-tasohyppelyyn. Erityisesti tutkittiin, voidaanko koneoppimista hyödyntää yhden henkilön projekteissa. Tavoitteeksi asetettiin saada valmiiksi järkevältä vaikuttava tekoäly, jonka toiminnasta pystytään hahmottamaan sen oppimat asiat.

Tekoälyn koulutusta varten luotiin prototyyppi 3D-tasohyppelypelin tasosta, johon tehtiin mahdollisimman monipuolisesti haasteita. Projektin tasot tehtiin Unity-pelimoottorilla, ja tekoälyn tekemiseen käytettiin Unity ML-Agents-työkalupakkia. Tekoälyn oppimisen seuraamiseen käytettiin TensorBoard-työkalua.

Projekti aloitettiin tekemällä yksinkertainen taso, josta tekoälyn piti päästä läpi. Kun tekoäly läpäisi tason, nostettiin tason vaikeusastetta ja kehitettiin tekoälyä. Tasojen tekemisessä keskityttiin esteiden monimuotoisuuteen, jotta tekoäly kehittyisi monipuoliseksi.

Lopputuloksena todettiin, että ML-Agents on sopiva pieniin projekteihin, vaikka tekoäly ei ollut täysin aukoton. Koneoppimisen ML-Agents-työkalupakin huomattiin olevan huomioitava vaihtoehto varsinkin indie-tason projekteissa.

Työssä saatiin aikaiseksi pienimuotoinen tasohyppelypeli, jonka kehitystä on tarkoitus jatkaa tulevaisuudessa ja opettaa tekoälyä selviytymään vielä monimutkaisemmista ongelmista.

Avainsanat	pelikehitys, koneoppiminen, Unity, tasohyppelypeli

Author Title Number of Pages Date	Markus Pietilä Machine Learning in a Platform Game 34 pages 1 December 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Game Applications
Instructor	Miikka Mäki-Uuro, Senior Lecturer

The main purpose of this final year project was to clarify the suitability of machine learning in a 3D platform game. The study especially strived to investigate can machine learning be utilized in a one person project. The goal was to create an artificial intelligence that clearly displays the logic it has developed from learning.

A prototype of a 3D-platform game with as diverse obstacles as possible was created for the artificial intelligence. The project's levels were created with the Unity game engine and Unity's ML-Agents toolkit was used to create the artificial intelligence. The artificial intelligence's progress was monitored using TensorBoard toolkit.

The project was started by creating a simple level for the artificial intelligence to get through. When the AI passed the level the level's difficulty was increased and the AI was developed further. When creating levels the focus was set on the obstacles' diversity in order for the AI to develop to be as versatile as possible.

The conclusion was that ML-Agents is a suitable option for a small project, although the artificial intelligence was not flawless. Machine learning with the ML-Agents toolkit was noted to be a considerable option especially in indie-style projects.

As a result a small-scale platform game was created that is meant to be further developed in the future along with teaching the artificial intelligence to get through even more complicated problems.

Keywords	game development, machine learning, Unity, platform game

Sisällys

1 Johdanto

Tekoälylle on monia eri määritelmiä. Yksi määritelmistä määrittelee tekoälyn järjestelmän kyvyksi käsitellä dataa, oppia datasta sekä käyttää dataa ja siitä opittuja asioita päästäkseen päämääräänsä. Tekoälylle on myös eri alakäsitteitä, jotka sopeutuvat eri tarkoituksiin.

Koneoppiminen on yksi suurimmista tekoälyn sovelluksien osa-alueista. Se antaa järjestelmille kyvyn oppia ja kehittyä kokemuksista autonomisesti. Tämä eroavaisuus muista tekoälyn muodoista mahdollistaa sen, että koneella on mahdollisuus kehittyä tehtävässään paremmaksi kuin ihminen tai saavuttaa koneelle asetetut tavoitteet ilman ihmisen asettamia raja-arvoja ja malleja.

Insinöörityön tarkoitus on tutkia koneoppimisen soveltamista itse tehtyyn tasohyppelypeliin. Työn pääasia on selvittää, millaisilla algoritmeilla, hyperparametreilla ja ympäristön tutkinnalla tekoäly oppii parhaiten. Pelin ensisijainen tarkoitus ei ole olla laaja, vaan mahdollisimman monipuolinen tekoälylle tehtyjen haasteiden puolesta. Työtä varten luotu kenttä tehtiin Unity-pelimoottorilla, jonka kanssa käytettiin Unity ML-Agents -työkalupakkia tekoälyn opettamiseen.

Toisessa luvussa käydään läpi tekoälyn historiaa ja kehitystä sekä tuodaan esille muutama tärkeä tekoälyn saavutus. Kolmas luku sisältää koneoppimisen kolme eniten käytettyä paradigmaa. Neljännessä luvussa käydään läpi tässä työssä käytettyjä ja koneoppimiselle usein tärkeitä neuroverkkoja. Viidennessä luvussa käsitellään insinöörityössä käytettyjä työkaluja ja niiden toimintaa. Kuudes luku sisältää insinöörityön toteutuksen askeleittain ja seitsemännessä luvussa kerrotaan työn lopputulokset sekä yleinen yhteenveto työstä.

2 Tekoälyn historia

Ensimmäinen neuroverkkojen ilmentymä tapahtui vuonna 1943, kun neurofysiologi Warren McCulloch ja matemaatikko Walter Pitts kirjoittivat tutkielman neuronien toimintatavasta [1]. Vuonna 1952 Arthur Samuel loi yhden maailman ensimmäisistä itseoppivista tietokoneohjelmista ja virallisti termin koneoppiminen vuonna 1959. Arthur Samuel kehitti myös tekoälylle yleisesti tärkeän alfa-beetakarsintahakualgoritmin. Koneoppimisen yleistymisen myötä vuonna 1959 kehitettiin myös yksikerroksinen neuroverkko ADALINE, joka osasi tutkia binäärisiä kaavoja. Seuraavalle mallille annettiin nimeksi MADALINE, joka on kolmikerroksinen neuroverkko ja osaa häivyttää kaiun puheluista. Se on käytössä vielä nykypäivänä.

Vuonna 1986 keksittiin backpropagation-algoritmi, jota käytetään monikerroksisten neuroverkkojen opettamiseen. Noin vuosikymmen myöhemmin tekoäly sai lisää huomiota, kun shakin maailmanmestari Garry Kasparov hävisi IBM:n kehittämälle tietokoneelle. Tämän jälkeen tekoälylle on löydetty monia erilaisia tarkoituksia. Googlen kehittämä GoogleBrain vuonna 2012 tutki suurta määrää kuvia ja lopulta oppi tunnistamaan asioita Youtube-videoista [2]. Vuonna 2014 Facebook julkisti, että se oli kehittänyt neuroverkon, joka osasi tunnistaa ihmisiä yhtä tarkasti kuin ihminen itse [3]. Kaksi vuotta jälkeenpäin Googlen kehittämä AlphaGo voitti ammattilaispelaajan viiden pelin ottelusarjassa voittaen kolme ensimmäistä peliä Go-lautapelissä.

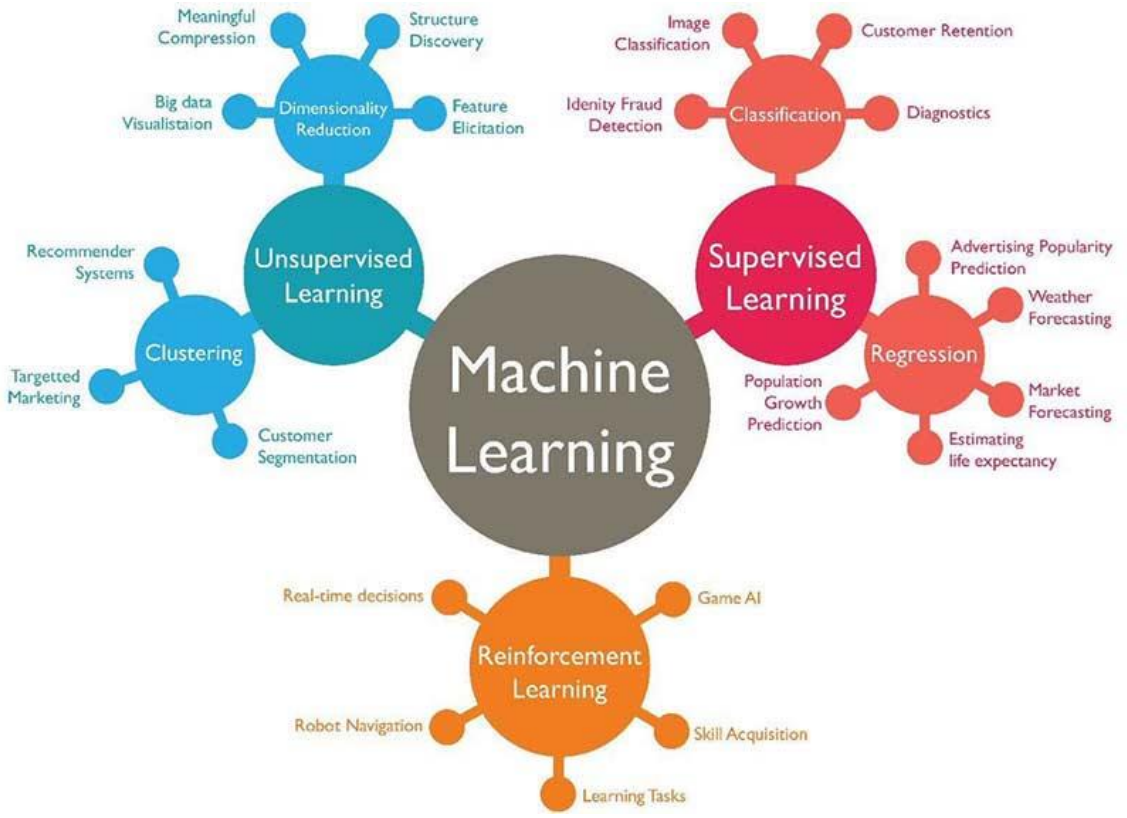
Luultavasti vakuuttavin peleihin liittyvä koneoppimisprojekti nähtiin vuonna 2017, kun OpenAI:n tekemä OpenAI Five voitti kärkipelaajia ja ammattilaispelaajia vastaan Dota 2 –tietokonepelissä [4]. OpenAI:n tekoäly toimi alussa vain kahden pelaajan otteluissa, mutta myöhemmin siitä kehitettiin toimiva viisi vastaan viisi -otteluihin. Otteluissa oli muutama lisätty sääntö, mutta lähes koko pelin kattava tekoäly, joka pystyi voittamaan parhaita pelaajia, oli huikea harppaus, sillä Dota 2:ta pidetään tietokonepelien keskuudessa yhtenä monimutkaisimmista.

Koneoppimisen saavutukset ovat varsinkin viime aikoina olleet yrityksien kiinnostusten kohteena. Tämän myötä ala on saanut uusia työkaluja, jotka sopivat yleensä myös yksityishenkilön projekteihin. Tekoälyn kehittyessä on myös herännyt idea niin sanotusta yleisestä tekoälystä, jolla on kyky ymmärtää tai oppia mikä vaan älykkyyttä vaativa tehtävä, jonka ihminen voi oppia. Tällaista tekoälyä on jo kauan teorioitu tieteisfiktioissa.

Nykyään lähes tietoisuuden saavuttanutta, tai kokonaan tietoista konetta kutsutaan usein vahvaksi tekoälyksi, ja se on joidenkin organisaation tutkimuskohde.

3 Koneoppimisen paradigmat

Koneoppimisen kolme paradigmaa (kuva 1) ovat ohjattu oppiminen, ohjaamaton oppiminen ja vahvistusoppiminen. Ohjattuun oppimiseen voidaan vielä mieltää kuuluvan regressio ja luokittelu sekä ohjaamattomaan oppimiseen klusterointi ja datan ulottuvuuksien vähentäminen.



Kuva 1. Koneoppimisen paradigmat [5].

3.1 Ohjattu oppiminen

Ohjattu oppiminen (engl. supervised learning) on hyödyllinen silloin, kun tekoälylle opetettava data sisältää sekä syötteen että tuloksen. Kone pyrkii opettelemaan yhteyden

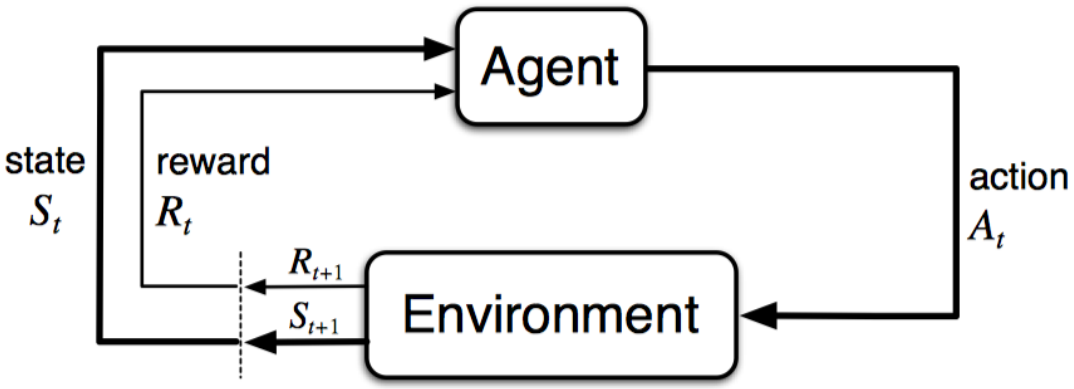
syötteen ja siitä saadun tuloksen välillä. Hyvin oppinut kone osaa harjoitusdatasta tehdyllä mallilla laskea erittäin tarkan lopputuloksen syötteestä, joka ei ollut harjoitusdatassa. Ohjattu oppiminen on usein hyvällä toteutuksella tarkka ja nopea. Sitä käytetään esimerkiksi populaation kasvun ennustusten ja eliniän odotteen laskentaan.[6.]

3.2 Ohjaamaton oppiminen

Kun käytettävä data ei sisällä tuloksia, vaan pelkkiä datapisteitä, käytetään tekoälyn opettamiseen usein ohjaamatonta oppimista (engl. unsupervised learning). Tämä algoritmi tutkii yhtenäisyyksiä datassa. Harjoitusdatasta rakennetun mallin perusteella algoritmi etsii löytyykö datasta samoja yhtenäisyyksiä vai ei. Klusterointi on yleinen osa ohjaamatonta oppimista, jossa data jaetaan osajoukkoihin jonkun elementin perusteella. Kohdistettu markkinointi on yksi ohjaamattoman oppimisen käyttöalueista.[7.]

3.3 Vahvistusoppiminen

Vahvistusoppiminen (engl. reinforcement learning) on tässä insinöörityössä käytetty koneoppimisen algoritmi. Vahvistusoppiminen on yleensä laajempi käsite kuin ohjattu tai ohjaamaton oppiminen. Kuvasta 2 on nähtävissä, että jokaisessa vahvistusoppimista käyttävässä koneessa on agentti, joka yrittää etsiä ratkaisun saadakseen suurimman palkinnon. Palkintojen saamisessa vahvistusoppiminen eroaa ohjatusta oppimisesta siinä, että epätäydellisiä tuloksia ei välttämättä jätetä pois.[8.]



Kuva 2. Agentin ja ympäristön palautesilmukka [8].

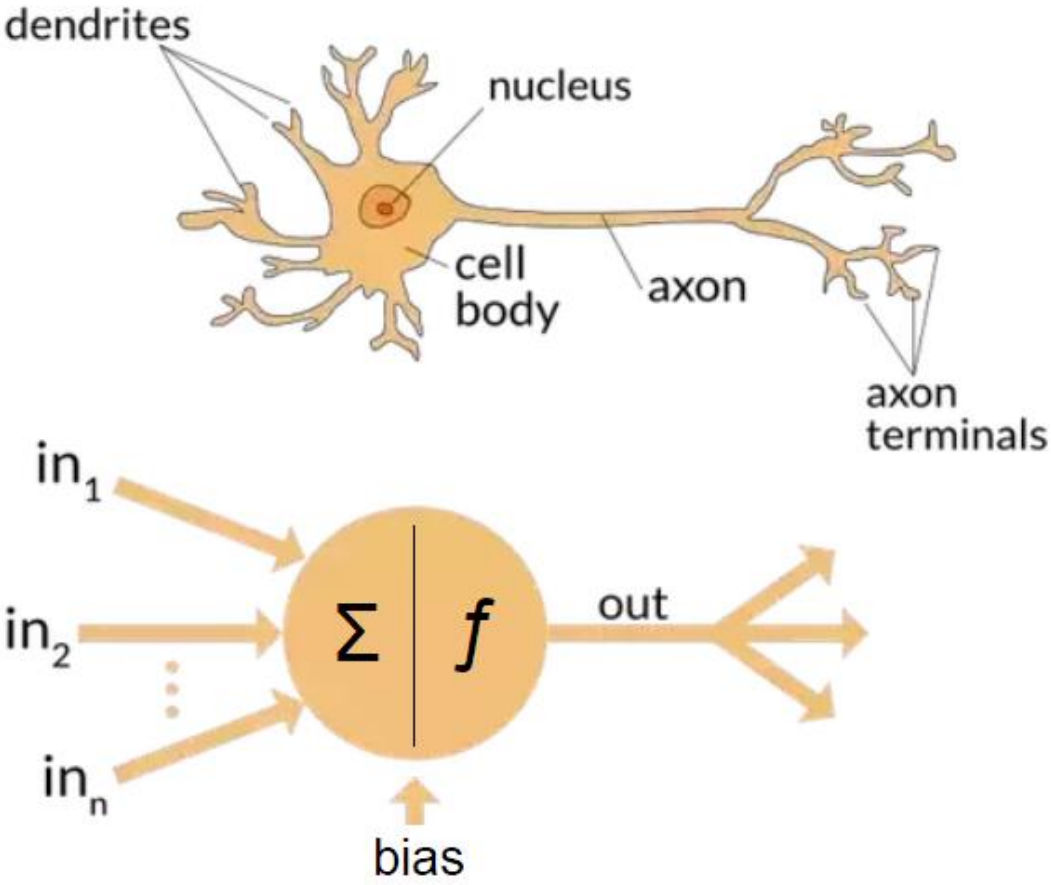
Vahvistusoppimisen agentin tehtävänä on tutkia ympäristöään ja päättää sen perusteella mitä seuraavaksi tulisi tehdä. Agentin päätökset riippuvat suuresti palkinnoista. Agentille annetaan palkintoja sen suoriutumisen perusteella, esimerkiksi maaliin pääsy tarkoittaa usein suurinta palkintoa. Pelin tekoälyn suunnittelemisessa palkintofunktion toteuttaminen oikein on erittäin tärkeää koneen oppimisen kannalta. Agentti toimii vertailemalla senhetkistä ympäristöä sen oppimaan malliin ja päättää sen perusteella, mikä toiminto siinä tilassa antaisi parhaan palkinnon. Agentti ei kuitenkaan voi aina valita suoraan optimaalisinta toimintoa, sillä se johtaisi usein hyvin yksinkertaiseen ja putkinäköiseen tekoälyyn, joka ei välttämättä opi kaikkea ympäristöstään. Tämän takia agentilla tulee olla myös uteliaisuutta, joka saa agentin poikkeamaan optimaalisimmasta reitistä ja tutkimaan ympäristöään enemmän.

3.4 Hyperparametrit

Koneoppimisessa käytetyt hyperparametrit ovat parametreja, joiden arvot asetetaan ennen oppimisprosessia. Tekoäly ei opettele hyperparametreja koulutuksen aikana, vaan niillä ohjataan oppimisen etenemiseen vaikuttavia elementtejä, kuten tekoälyn monimutkaisuutta ja oppimistahtia. Hyperparametrit voidaan jakaa kahteen eri luokkaan, jotka ovat optimointihyperparametrit ja mallihyperparametrit. Optimointihyperparametrit vaikuttavat oppimisen optimointiin harjoitteluprosessin aikana esimerkiksi määrittelemällä oppimisjaksojen laajuudet. Mallihyperparametrit vaikuttavat tekoälyn mallin rakentamiseen. Esimerkiksi oppimisessa luodun neuroverkon koko ja kerrokset määritellään mallihyperparametreissa.[9.]

4 Neuroverkot

Neuroverkot ovat kuvan 3 tavoin pääpiirteiltään ihmisaivoja jäljitteleviä algoritmeja, joiden tehtävänä on oppia tunnistamaan yhtenäisyyksiä esimerkeistä. Yhtenäisyydet ovat mistä vain datan ominaisuuksista luotuja numeerisia vektoreita, joiden avulla tekoäly tekee päätöksensä seuraaviin toimintoihin. Neuroverkot koostuvat keinotekoisista neuroneista, joilla on sisääntulo, painoarvo, bias-arvo, aktivaatiofunktio ja ulostulo. [5.] Neuroverkot ovat erittäin hyvin soveltuvia esimerkiksi kasvojentunnistukseen, mutta ne toimivat myös pelien tekoälyissä oikein suunniteltuina.

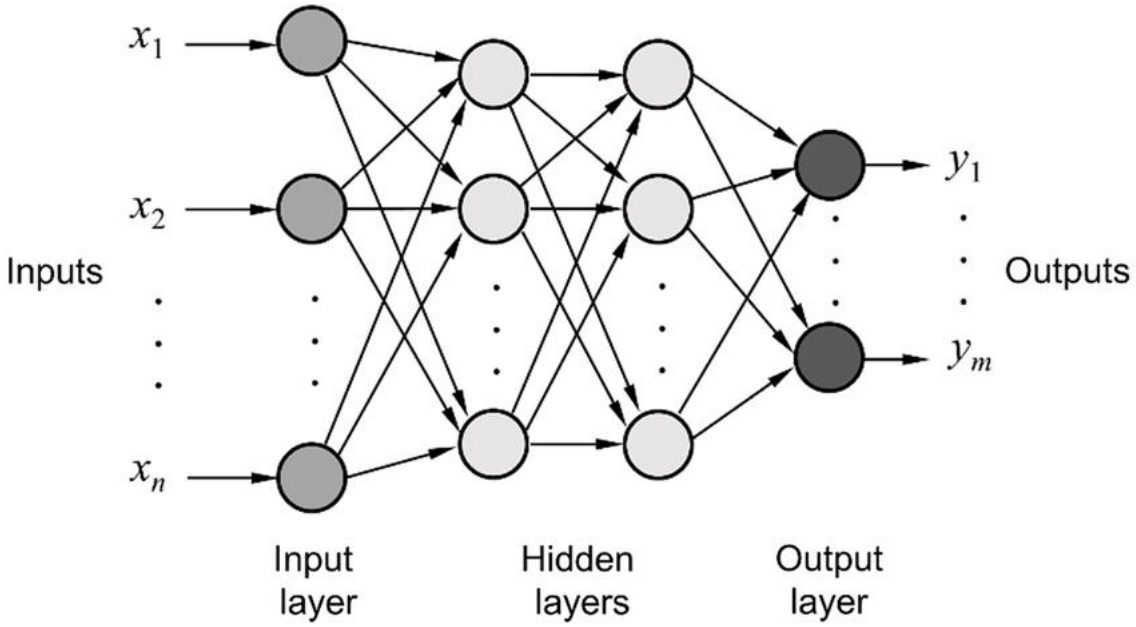


Kuva 3. Biologinen ja keinotekoinen neuroni [10].

Jokainen neuroni sisältää oman tietonsa siitä, mitä se on oppinut, sekä asetetuista säännöistä tai sen itselleen asettamista säännöistä. Jokaisessa kerroksessa neuroneilla on monia yhteyksiä seuraavaan sekä viime kerroksen neuroneihin, joiden välillä neuronit joko saavat tai kuljettavat dataa. Ensimmäisen kerroksen neuronin saadessa dataa, se aktivoi seuraavan kerroksen neuroneita tietyn kuvion mukaisesti lukemansa datan perusteella aktivaatiofunktiolla. Tämä ketju jatkuu ulostulokerrokseen saakka kuvan 4 mukaisesti, jolloin neuroverkko valitsee todennäköisimmän vastauksen. [10.]

Jokaiselle neuronille on määrätty painoarvo, jota neuroverkko voi muuttaa simuloinnissa. Oppiessa neuroverkossa käytettävän algoritmin (esim. backpropagation) tärkeä tehtävä onkin muuttaa neuroneiden painoarvoja vastaamaan datan tärkeitä positiivisia ja negatiivisia piirteitä sekä vähentää painoarvoja mitättömille piirteille. Esimerkiksi kissaa tunnistettaessa hännällä tulee olla korkea positiivinen painoarvo ja sarven painoarvo korkea negatiivinen. Neuronin data usein kartoitetaan 0:n ja 1:n välille käyttämällä logistista

funktiota, jolloin neuronien ulostulo voidaan kuvitella olevan 0 %:n ja 100 %:n välillä. Kuitenkin ennen datan normalisointia siihen lisätään usein luku, jolla pyritään saada vain halutun arvon ylittäneet neuronit aktivoimaan seuraavan kerroksen neuroneita. Tätä arvoa kutsutaan bias-arvoksi, ja se voi myös muuttua oppimisen aikana.[10.]



Kuva 4. Monikerroksinen neuroverkko [11].

Vahvistumisoppimisessa käytetään yleensä monikerroksisia neuroverkkoja. Monikerroksissa neuroverkoissa on monta piilotettua kerrosta sisääntulokerroksen ja ulostulokerroksen välillä, ja ne ovat käytännössä vain erinäisiä neuronikerroksia. Neuronien painoarvojen tarkentamiseen käytetään yleensä vastavirta-algoritmia, jossa edetään viimeisestä neuronikerroksesta ensimmäiseen.

Vahvistusoppimisessa usein käytetyssä Q-oppimisessa on tarkoitus oppia optimaalinen etenemistapa jokaisessa tilassa, jossa järjestelmä voi olla. Järjestelmän agentti valitsee toiminnon ja saa palautteen toiminnosta, minkä perusteella sen tila-toimintoparin Q-arvoa muutetaan. Jos palaute on positiivista, tila-toimintoparin Q-arvoa nostetaan, ja lasketaan, jos palaute on negatiivinen. Kun tila-toimintoparien määrä kasvaa suureksi, ei ole mielekästä pitää jokaisen parin Q-arvoja tallessa erikseen. Tällöin toimintojen Q-arvot usein tallennetaan neuroverkkoon, josta ne voidaan hakea. Neuroverkon painoarvoja päivitetään odotettujen Q-arvojen ja oppimisessa saatujen Q-arvojen erojen mukaan.[12.]

5 Työkalut ja menetelmät

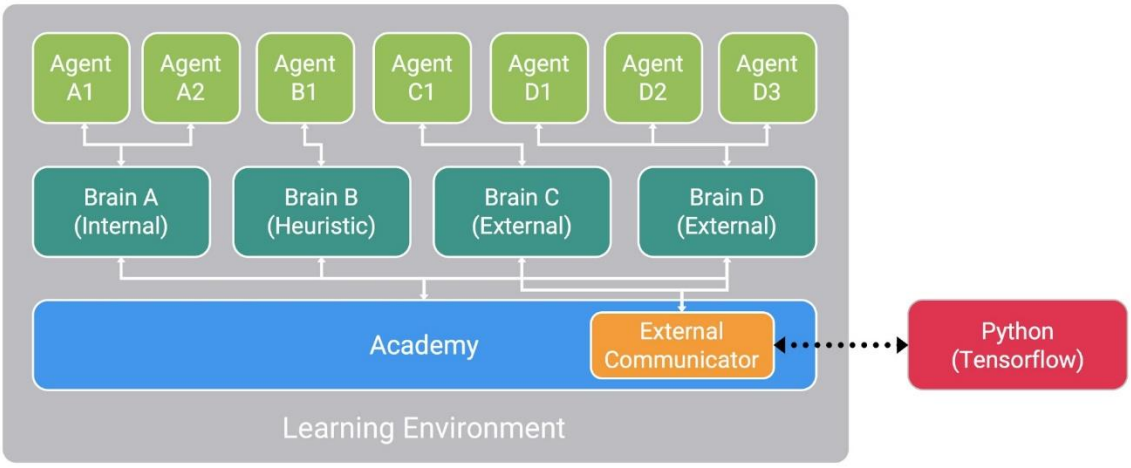
Insinöörityön pelin toteuttamiseen käytettiin Unity-pelimoottorin 2019.1.5f1-versiota. Tekoäly toteutettiin Unityn ML-Agents-versiolla 0.9.2, joka toimii yhdessä version 3.7.3 Python-prosessin kanssa. ML-Agents käyttää avoimen lähdekoodin kirjastoa nimeltä TensorFlow, joka rakentaa syväoppimisen mallit, ja sen toteutus on suurimmaksi osaksi abstraktoitu.

5.1 Unity-pelimoottori

Unity on vuonna 2005 julkaistu suosittu järjestelmäriippumaton pelimoottori, jolle on eri lisenssejä mukaan lukien ilmainen yksityislisenssi. Nykyään Unity käyttää pääosin C#-kieltä, mutta se tuki ennen Boo-ohjelmointikieltä sekä Unityn versiota JavaScriptistä nimeltä UnityScript, jonka tuki lopetettiin myös vuonna 2017. Unity tukee sekä 2D- että 3D-pelejä eri fysiikkamoottoreilla.

5.2 Tekoälyn komponentit

ML-Agents-työkalupakin kolme pääkomponenttia, jotka ovat jokaisessa skenessä, jossa sitä käytetään, ovat agentti (engl. agent), aivot (engl. brain) ja akatemia (engl. academy) (kuva 5). Vahvistusoppimiseen ML-Agents käyttää PPO-algoritmia, joka on toteutettu TensorFlow-tekoälymoottorilla ja suoritetaan erillisessä Python-prosessissa, joka kommunikoi Unityn kanssa. PPO-vahvistusoppimisalgoritmi käyttää neuroverkkoja agentin tilojen ja toimintojen tallentamiseen ja niiden perusteella parhaan vaihtoehdon valitsemiseen. PPO-algoritmi kerää tekoälyn kokemuksia sen ympäristöstä, ja se käyttää niitä päätöksenteon muokkaamiseen, minkä jälkeen se toistaa saman uudella joukolla kokemuksia. PPO-algoritmille olennaista on, että sen päivitykset muokkaavat käytäntöjä lievästi verrattuna moniin muihin samankaltaisiin algoritmeihin. PPO-algoritmi on esimerkiksi OpenAI:n käyttämä ja eniten suosima algoritmi sen selkeyden, keveyden ja hyvän lopputuloksen takia.[13.]



Kuva 5. ML-Agentsin pääkomponentit [13].

Agentti on yleisnimitys asialle, joka tekee päätöksiä ja toimii sen oppineen tiedon mukaan. Pelissä agentti on usein tekoälyn omaava hahmo, joka on vuorovaikutuksessa sen ympäristön eli pelin kentän kanssa. Kentästä kerätään usein tietoa keinotekoisin aistein ja mahdollisesti vaikutetaan myös kenttään esimerkiksi raajoilla. Agentti pyrkii pääosin valitsemaan senhetkisessä tilassa toiminnon, joka on todennäköisin paras valinta sen oppiman tiedon mukaan. ML-Agentsissa agenttiin on liitetty aivot, jossa agentin päätöksenteko kiteytyy. Simuloinnissa yksiin aivoihin voi olla liitettynä monta agenttia.[13.]

Aivoja on kolmea eri tyyppiä: oppivat aivot, heuristiset aivot ja pelaaja-aivot. Oppivia aivoja käytetään simuloinnissa, kun tekoälylle opetetaan uusi malli. Oppivat aivot ovat yleisimmät agentin logiikan opettamiseen käytetyt aivot. Heuristiset aivot mahdollistavat agentin toimintalogiikan käsikoodaamisen ilman simulointia. Pelaaja-aivoja käytettäessä ei tapahdu oppimista, vaan ne mahdollistavat itse pelin pelaamisen. Ne ovat hyödylliset kokeilemaan, että kenttä on järkevä, sekä varmistamaan agentin liikkumisen toimivuuden. Aivoissa määritellään myös agentin tutkimien asioiden vektorin pituus ja agentin toimintojen vektori.

Akatemia hallinnoi kaikkia skenen agentteja sekä aivoja, ja se on aina oltava skenessä, jossa on agentti. Akatemiassa voi säätää simuloinnin parametreja, kuten simuloinnin kuvataajuus ja renderöinnin laatu. Nollausparametrit asetetaan myös akatemiassa. Nollausparametrit ovat käsintehtyjä parametreja, jotka muuttuvat simuloinnin aikana. Niillä

voidaan esimerkiksi merkitä kentän vaikeutta, joka nousee simuloinnin edetessä. Akatemiassa tehdään myös kaikkia skenen agentteja ja aivoja koskevat päätökset sekä kentän nollaus ja muokkaus.[14.]

5.3 Hyperparametrit

PPO-algoritmin käyttämät hyperparametrit ovat muokattavissa config-tiedostossa. Yksinkertaisten ongelmien selvittämiseen tekoäly pärjää usein oletusparametreilla, mutta monimutkaisemmissa tilanteissa hyperparemetrien säätäminen sopivaksi parantaa huomattavasti oppimista ja vähentää siihen tarvittua aikaa. Palkintosignaaleilla voi antaa agenteille palkintoja muustakin kuin kentässä etenemisestä sekä muokata niiden vahvuutta ja heikkenevyyttä simuloinnin edetessä. Uteliaisuuspalkintosignaali antaa agentille palkintoja silloin, kun sen toiminto eroaa sen laskelmoidusta toiminnosta nostaen palkintoa sen perustella, mitä suurempi eroavaisuus on. GAIL-palkintosignaali tutkii toisen neuroverkon avulla eroavaisuuksia agentin toiminnassa ja nauhoitettujen demonstraatioiden välillä. Agentille annetaan palkinto sen perusteella, kuinka lähelle se pääsee demonstraatiota. Simuloinnissa käytetyt hyperparametrit ovat seuraavat:

- Lambda, eli kuinka paljon agentti luottaa laskelmiinsa nykyisestä arvon ennustuksesta, kun lasketaan seuraavaa. Pieni lambda tarkoittaa suurempaa luottoa nykyiseen ennustukseen kuin korkea lambda, joka luottaa enemmän ympäristöstä saatuihin palkintoihin. Oikea lambda johtaa usein tasaisempaan oppimiseen.

- Buffer size viittaa puskurin kokoon eli siihen, kuinka paljon agentin tulee tutkia, toimia ja saada palkintoja, ennen kuin tapahtuu oppimista ja mallia päivitetään. Puskurin koon nostaminen usein johtaa tasaisempaan oppimisprosessiin, mutta saattaa myös nostaa oppimiseen vaadittua aikaa eikä ole aina tarpeellinen yksinkertaisten ongelmien ratkaisemiseen.

- Batch size tarkoittaa agentin kokemuksien määrää, joka saavutetaan yhdessä gradient descentalgoritmin iteraatiossa. Sen tulee olla suuri, jos agentti käyttää jatkuvaa toimintotilaa, ja pieni, jos agentti käyttää diskreettiä toimintotilaa. Parametrin tulee myös olla murto-osa puskurin koosta.

- Number of Epochs säätää gradient descentalgoritmin iteroinnissa läpäisyjen määrää kokemuspuskurissa. Sen koko tulisi rinnastaa Batch sizen koon kanssa. Pienentäminen antaa tasaisempia päivityksiä, mutta hidastaa oppimista.

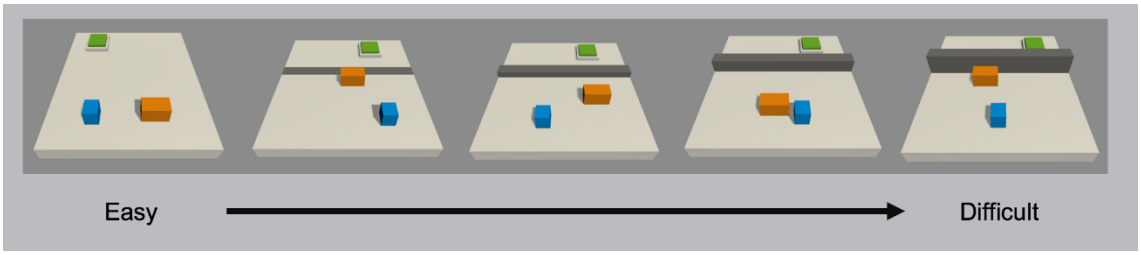
- Learning Rate vastaa jokaisen gradient descentalgoritmin päivityksen vahvuutta. Jos palkinnot ovat heitteleviä tai jos palkinto ei nouse ollenkaan simuloinnin aikana, tulisi pienentää tätä parametria.

- Time Horizon tarkoittaa, kuinka monta askelmaa kokemusta agentin tulee kerätä, ennen kuin kokemukset lisätään kokemuspuskuriin. Kun asetettuun rajaan päästään, ennen kuin oppimisepisodi on loppunut, ennustetaan odotetun palkinnon kokonaismäärä agentin nykyisestä tilasta. Tämän parametrin tulisi olla kuitenkin tarpeeksi suuri, jotta saadaan tärkeimmät agentin ratkaisut talteen.

- Max Steps määrää, kuinka monta askelta simulaatiota suoritetaan harjoitusprosessin aikana. Suurempi määrä vastaa suurempaa harjoitusaikaa, joten tätä parametria tulisi nostaa sen perusteella, mitä monimutkaisempi ratkaistava ongelma on.[15.]

5.4 Kurssioppiminen

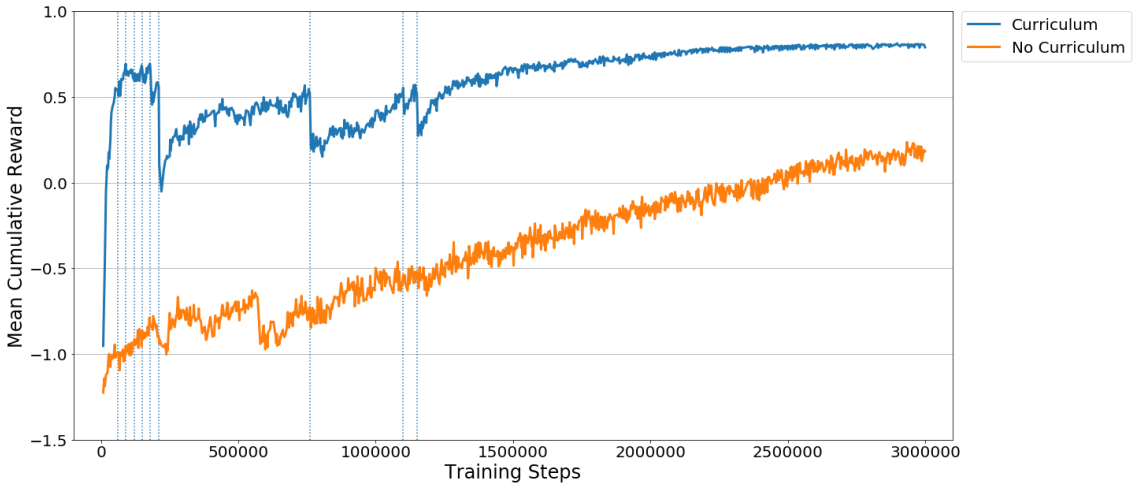
ML-Agents-työkalupakissa on haluttaessa mahdollista käyttää kurssioppimisominaisuutta, jonka periaate on nähtävissä kuvassa 6.



Kuva 6. Kurssioppimisen eteneminen [16].

Kurssioppiminen perustuu ihmisen opetuskaareen, joka alkaa helpoimmista ongelmista, kuten yhteenlaskuista matematiikassa. Ratkaistuista ongelmista saatua tietoa käyttäen ihmisen on helpompi ratkaista etenevästi vaativampia ongelmia. Etenevän vaikeustason

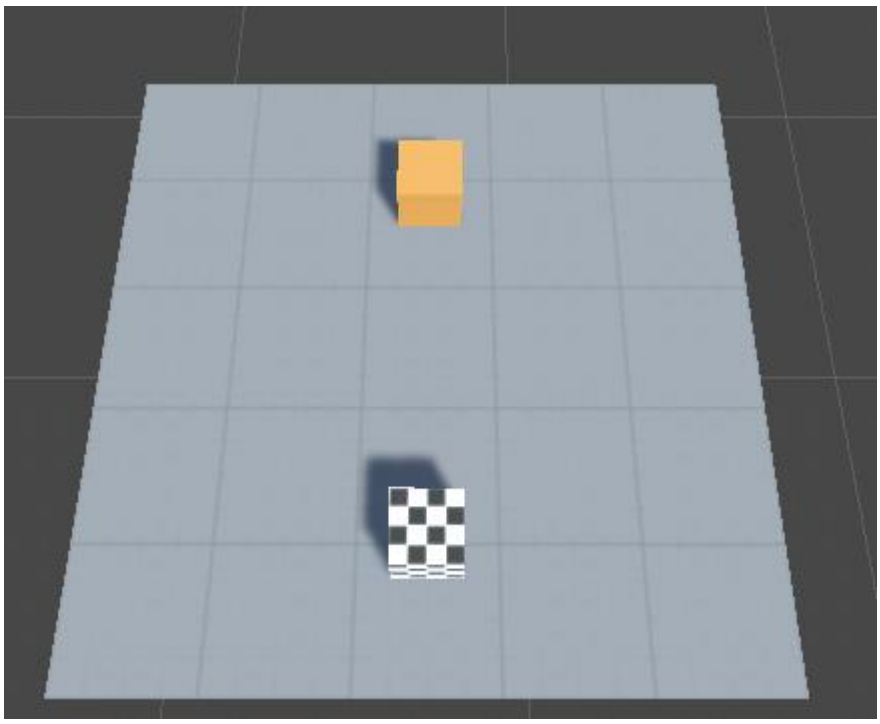
tarkoituksena on edistää oppimisen nopeutta ja laatua. Tätä yritetään hyödyntää myös tekoälyn kurssioppimisessa. Esimerkiksi tasohyppelypelissä kentän vaikeutta voidaan tekoälyn suoriutumisen perusteella nostaa lisäämällä esteitä simulaatioon, kun tekoäly on ylittänyt halutun suoriutumispisteen. Asteittain etenemällä tekoälyn tulisi suoriutua koko kentästä nopeammalla ja tasaisemmalla oppimisella, kuin aloittamalla oppimaan suoraan koko kenttää, kuten nähdään kuvasta 7.



Kuva 7. Kurssioppiminen verrattuna tavalliseen oppimiseen [16].

6 Tekoälyn kehittäminen tasohyppelyssä

Insinöörityö lähti liikkeelle luvussa 5 esiteltyjen työkalujen asentamisesta. Kun työkalut oli saatu toimimaan, kokeiltiin tekoälyn oppimista lähes sellaisenaan hyvin yksinkertaisessa tasohyppelypelin ongelmassa. (Kuva 8.)



Kuva 8. Tekoälyn ensimmäinen ongelma: maalikuution löytäminen.

6.1 Tekoälyn perusteiden rakentaminen

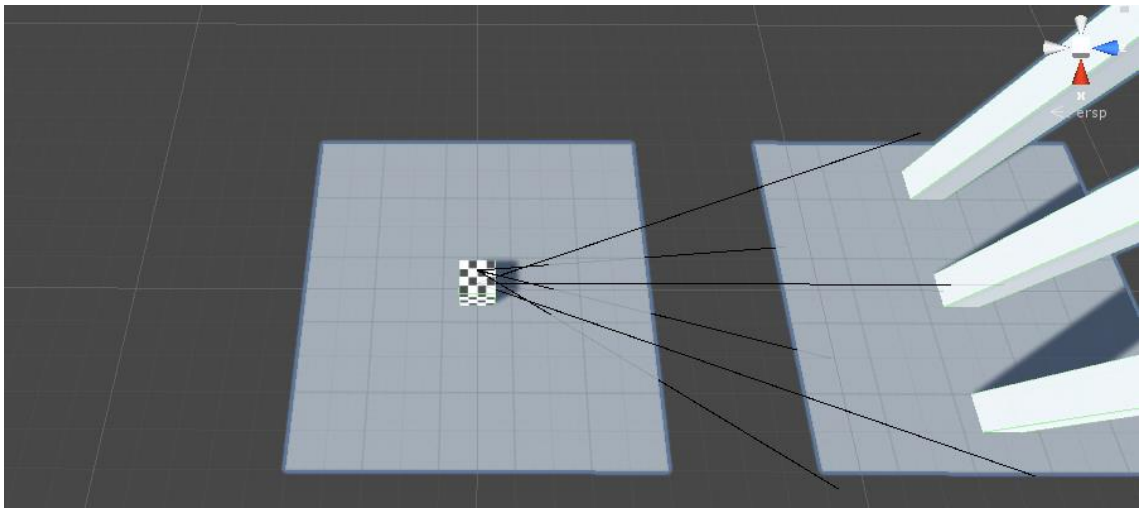
Tekoälyn oli ensimmäisessä ongelmassa tarkoitus vain löytää kuutio ja olla putoamatta tasohyppelypelin alustalta. Kuitenkin näinkin yksinkertaisessa ongelmassa huonolla palkintoalgoritmilla tekoäly alkoi toimia ei-toivotulla tavalla. Pelkästään antamalla koneelle palkintoja maalin löytämisestä ja rankaisuja putoamisesta tekoäly ajoi välillä suoraan ulos tasanteelta. Pidemmän harjoittelun jälkeen se alkoi välttellä kokonaan liikkumista miinuspisteiden pelossa putoamisesta. Ratkaisuksi tekoälylle lisättiin palkintoalgoritmiin pieni palkinto siitä, kun se pääsi lähemmäksi maalikuutiota, joka on yleisesti toimiva ratkaisu ongelmiin, jossa koneen halutaan etenevän jotain kohti. Tekoälylle lisättiin myös tutkittavaksi jatkuvasti sen etäisyys alustan keskipisteestä. Näillä muutoksilla kone löysi maalikuution luotettavasti putoamatta.

Testiongelman jälkeen alkoi projektiin sopivan pelikentän suunnittelu. Insinöörityön pääaiheena oli tekoälyn suoriutuminen, joten päädyttiin siihen ratkaisuun, että kentän ei tule olla pitkä, vaan mieluummin monipuolinen esteiden kannalta. Kenttää tehtiin myös as-

teittain, lisäten haasteita, kun tekoäly suoriutui halutun hyvin. Pelin ulkoasulla ei ole tekoälylle myöskään merkitystä, joten päätettiin pitäytyä Unityn tarjoamissa 3D-malleissa ja jättää animaatiot mahdollisesti myöhemmäksi.

Agentin toiminnot tehtiin AgentAction-funktiossa kolmen pituisella vektorilla. Vektorin ensimmäinen alkio päättää, kuinka paljon ja mihin suuntaan agentti kääntää itseään y-akselin ympäri, eli mihin agentti katsoo. Toinen vektorin alkio lisää agentin eteenpäin osoittamaan suuntavektoriin sille määritellyn määrän voimaa, eli se saa agentin kulkemaan eteenpäin. Kolmas alkio saa agentin hyppäämään, jos se on maassa ja arvottu määrä on tarpeeksi suuri.

Ympäristön tutkimiseen riitti helpommille kentän vaikeusasteille agentin nopeus, agentin sijainti verrattuna alustan keskipisteeseen ja agentin rotaatio. Kuitenkin tuli huomatuksi, että näistä saadut tiedot eivät riittäneet, kun kenttään lisättiin esteitä. (Kuva 9.)



Kuva 9. Agentin aistiminen säteillä.

Tämän takia agentille kehiteltiin keinotekoiset tavat aistia ympäristöään Unityn raycasteilla. Aluksi säteitä oli enemmän, mutta niiden määrää vähennettiin, kun huomattiin, että kaikki niistä saatu tieto ei ollut välttämätöntä ja se lisäsi oppimisaikaa. Suoraan eteenpäin lähtevien säteiden tarkoituksena on tutkia agentin edessä olevia esteitä, jotta se oppii niistä opitun tiedon perusteella tekemään ratkaisuja, joiden avulla agentti väistää esteet.

Esimerkiksi pylväiden väistö onnistui suhteellisen hyvin, kun niihin törmäämisestä annettiin pieniä miinuspisteitä. Tämä tosin vaikutti huonolla tavalla muuhun agentin toimimiseen, joten se jätettiin lopulta pois. Alaviistoon osoittavien säteiden tarkoituksena on tutkia, milloin agentin edessä ei ole alustaa, joten se osaa hypätä seuraavalle alustalle. Säteille kokeiltiin eri kulmia, ja lopulta todettiin, että parhaat tulokset saatiin, kun säteet osoittivat melko lähelle agenttia.

Agentti tutkii jokaisella simuloinnin askeleella ympäristöään esimerkkikoodi 1 tavoin. Ympäristön tutkimisen asetuksiin tehtiin muutoksia projektin edetessä. Parhaat tutkimisen kohteet löydettiin yleensä puhtaan kokeilemisen kautta, sillä asiasta ei ole juurikaan löydettävissä tietoa. Yleisesti agentti kuitenkin ainakin merkitsi muistiin sen oman nopeuden ja rotaation sekä agentin etäisyyden maaliin.

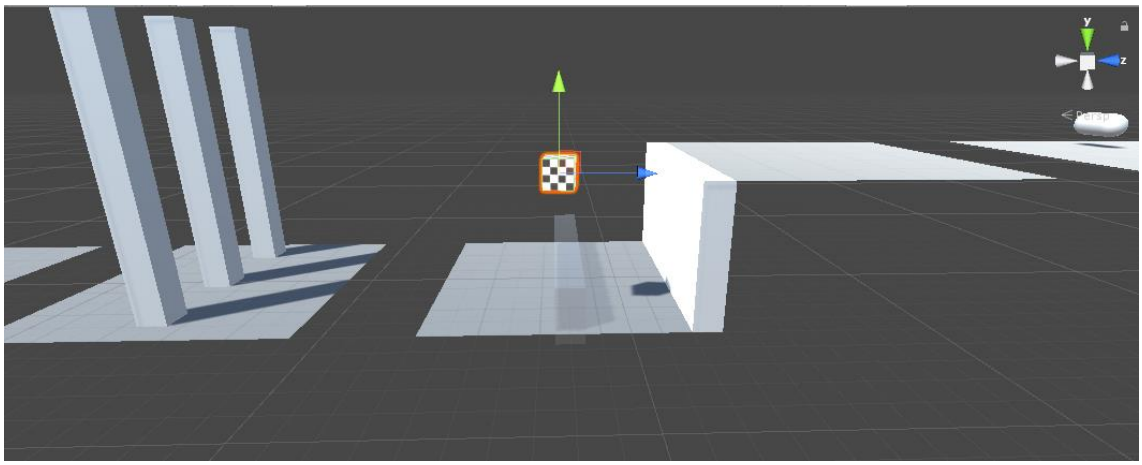
```
public override void CollectObservations()
    {
        Vector3 relativePosition = Target.position - this.transform.position;

        AddVectorObs(relativePosition);
        AddVectorObs(transform.rotation.eulerAngles.y);

        AddVectorObs(rBody.velocity);
        AddVectorObs(gameObject.transform.rotation.eulerAngles);

        foreach(Transform spike in Spikes)
        {
            AddVectorObs(spike.GetComponent<Rigidbody>().velocity);
        }

        float rayDistance = 15f;
        float[] rayAngles = {70f, 90f, 110f};
        string[] detectableObjects;

        detectableObjects = new[] {"obstacle", "platform", "spike", "bubble"};
        AddVectorObs(rayPer.Perceive(rayDistance, rayAngles, detectableOjects,
0f, 0f));
        AddVectorObs(rayPer.Perceive(rayDistance, rayAngles, detectableOjects,
1f, -10f));
    }
```

Esimerkkikoodi 1. Yksi versio agentin tiedonkeräyksestä.

Agentin keinotekoiset aistit toimivat Unityn raycasteilla. Raycastit palauttavat agentille tiedon osumista silloin, kun ne osuvat tutkimisfunktiossa määritettyihin esineisiin. Agenttia opettaessa huomattiin, että liian monta sädettä ja liian monta tutkittavaa kohdetta saavat agentin harjoitusajan nousemaan huomattavasti. Pidemmilläkin harjoitusajoilla silloin tällöin ilmeni odottamattomia lopputuloksia, kun kerättävää tietoa oli liikaa. Tämän

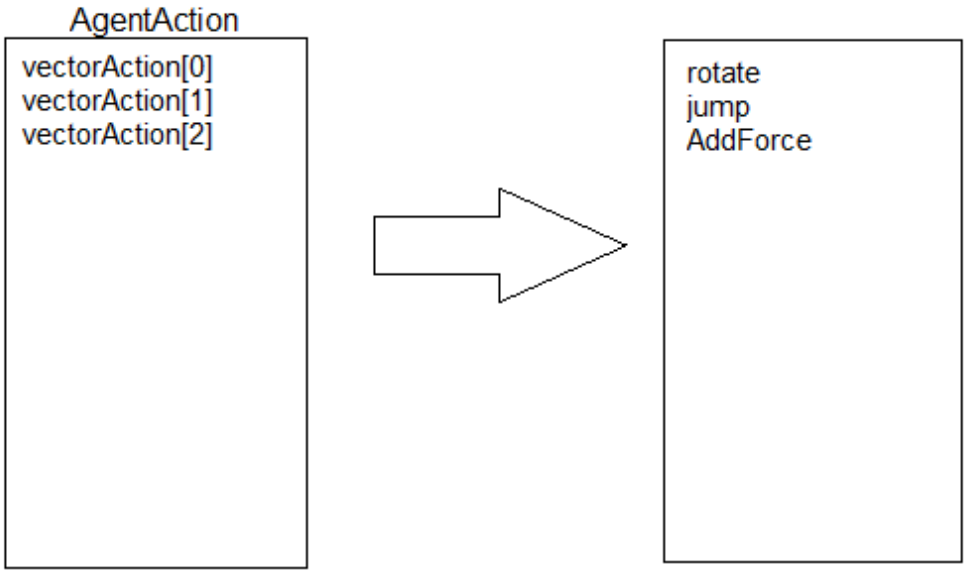
takia agentista lähtevien säteiden määrä vähennettiin lopulta kuuteen, joista kolme osoittaa eteenpäin 40 asteen levinneisyydellä ja loput kolme samalla hajoamalla, mutta alaspäin.

Agentin palkintofunktiota muutettiin myös projektin aikana pääosin kokeilun perusteella. Jo alussa huomattiin, että palkintojen muuttaminen edes vähän vaikuttaa usein vahvasti agentin toimintaan. Alusta pitäen agentti kuitenkin sai pienen palkinnon jokaiselta simulaation askeleelta, jossa se oli lähempänä maalia kuin viime askeleessa. Työn edetessä agentin suoriutumista seurattiin ja sen perusteella annettiin lisää palkintoja, kun nähtiin sen olevan tarpeen. Esimerkiksi hyppyriin osuminen, jonka avulla pääsee yli korkeasta seinästä, oli aluksi hankalaa agentille. Harjoitusaikaa saatiin huomattavasti lyhennettyä hyppyrin osalta, kun agentille annettiin pieni palkinto sen löytämisestä (kuva 10).



Kuva 10. Esimerkki hyppyrin toiminnasta.

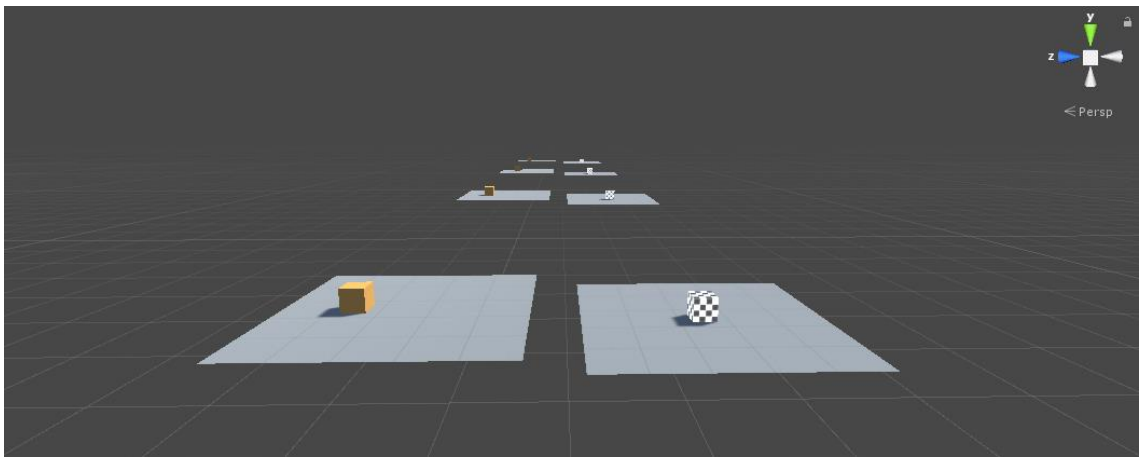
Agentin valitsemat toiminnot ovat kääntyminen, hyppääminen tai meneminen eteenpäin, joiden valitsemistapa on kuvassa 11. Oppimisen aikana agentti sai jokaisella askeleella kolmialkioisen vektorin, jonka alkiot vastasivat näitä toimintoja ja vaihtelevat -1:n ja 1:n välillä. Projektin edetessä huomattiin, että hyppääminen oli tärkeä olla todennäköinen vaihtoehto, sillä agentin on tärkeää oppia, milloin sen tulee hypätä. Kääntymistä myös kokeiltiin rajoittaa hieman antamalla rangaistus, kun agentti on kääntynyt pois maalista, tai palkinto kun agentti on kääntynyt maaliin päin.



Kuva 11. Agentin toimintofunktio kuvainnollisesti.

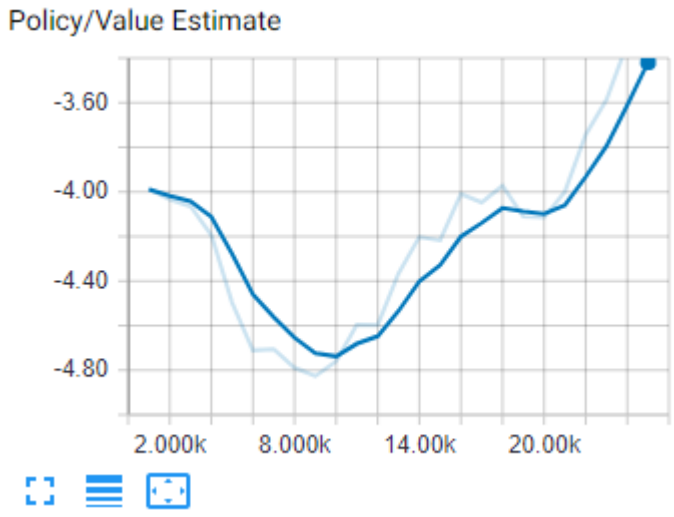
6.2 Työn alku

Työ toteutettiin lisäämällä tekoälylle vaikeutta, kun se oli selvinnyt edellisestä halutulla tasolla. Ensimmäiseksi aloitettiin yksinkertaisella tasolla, jossa oli kaksi alustaa, joista toinen oli hieman korkeammalla (kuva 12).



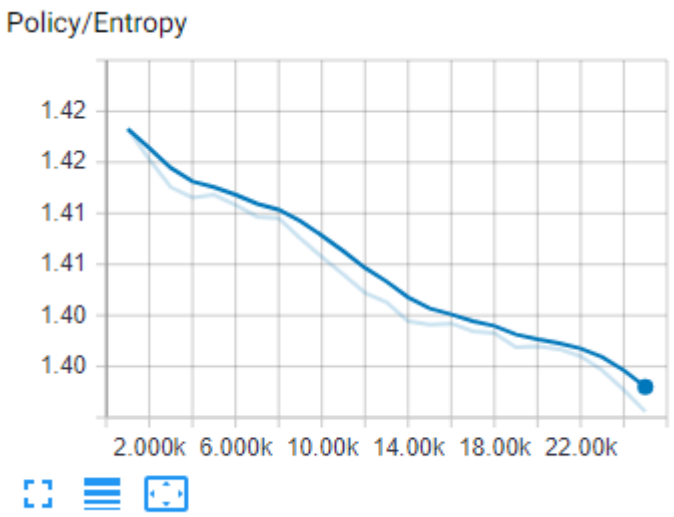
Kuva 12. Projektin ensimmäinen taso: kaksi alustaa eri korkeuksilla.

Agentti toimi tässä tasossa ilman keinotekoisia aisteja. Ympäristöstä kerättiin tietoa agentin sijainnista alustojen keskiosaan verrattuna, agentin nopeudesta, rotaatiosta ja etäisyydestä maaliin. Palkintoa agentille annettiin maaliin pääsemisestä ja siitä, kun se oli lähempänä maalia kuin edellisen simuloinnin askeleella. Agenttia rangaistiin suuresti, jos se putosi alustalta, ja vähän jokaisella simuloinnin askeleella oletuksena, minkä tarkoituksena on nopeuttaa maaliin pääsemistä. Noin viiden minuutin harjoituksen tuloksena agentti oppi pääsemään maaliin suurimman osan ajasta. (Kuva 13.)



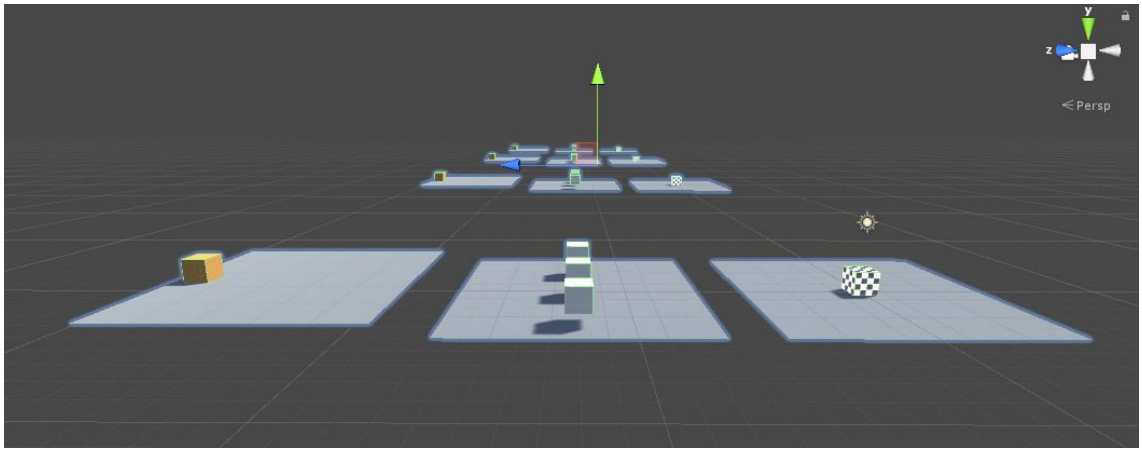
Kuva 13. Agentin palkinnot suhteutettuna simuloinnin askeleisiin ensimmäisessä tasossa.

Agentin hyperparametreihin ei tarvinnut tässä vaiheessa vielä puuttua paljon, mutta kuvasta 14 huomataan, että beeta-arvoa laskemalla saatiin myös haje suhteellisen tasaiseen laskuun, mikä on suositeltavaa.



Kuva 14. Hajeen lasku kuvainnollistettuna.

Tason vaikeutta nostettiin, kun agentti pääsi halutulle tasolle. Kenttään lisättiin yksi alusta, johon laitettiin kolme palikkaa esteeksi (kuva 15).

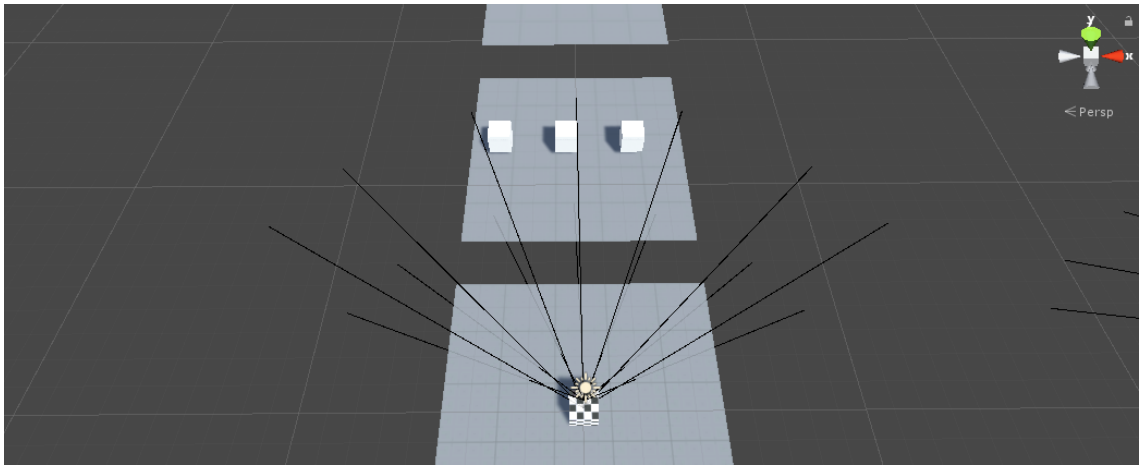


Kuva 15. Toinen taso: kolme alustaa ja ensimmäiset esteet.

Ensin tasossa käytettiin samoja algoritmeja kuin ensimmäisessä tasossa. Noin viiden minuutin harjoittelulla tekoälyn toiminta oli kuitenkin varsin huojuvaa. Tekoäly putosi välillä alustoilta, mutta se varsinkin oppi käyttämään hyväksi palkintoalgoritmia. Agentille

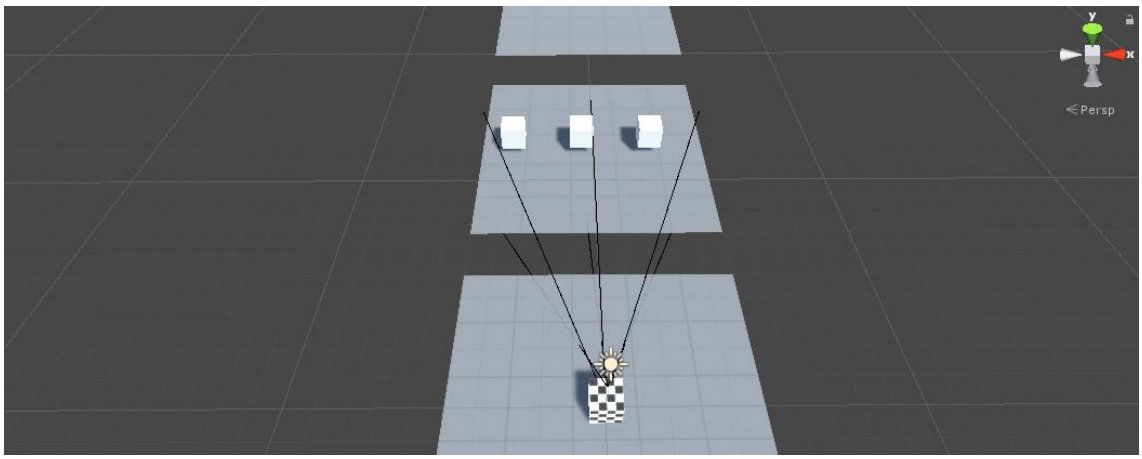
annettiin vieläkin pieni palkinto etenemisestä maalia kohti, joten se oppi käyttämään hyväksi toisella alustalla olevia esteitä liikkumaan hieman eteenpäin jokaisella simulaation askeleella samalla jumittaen esteissä. Tämän vuoksi palkintoa ei enää annettu, jos agentti oli kontaktissa esteisiin samalla, kun se eteni. Törmäyksestä esteisiin kokeiltiin myös antaa pieni rangaistus, mutta se toimi vain erittäin pitkillä harjoitusajoilla, sillä muuten agentti ei yleensä ottanut enää riskiä edetäkseen.

Toisessa tasossa agentille luotiin myös keinotekoiset aistit, joilla sen on tarkoitus huomata esteet (kuva 16).



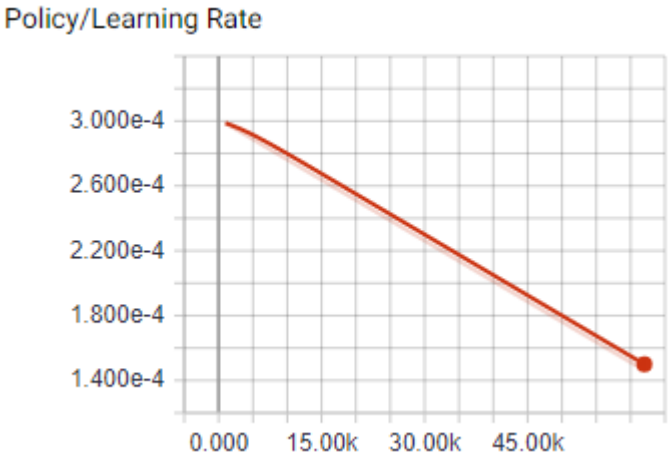
Kuva 16. Agentin alkuperäiset aistit, joilla se tutkii ympäristöään.

Alkuperäisesti agentti aisti asioita sen edestä 30:n ja 150 asteen välillä. Tällöin tutkittavien asioiden vektori, joka sijaitsee agentin aivoissa, oli 109 alkiota pitkä. Luku ei ole kovin massiivinen monimutkaisiin tehtäviin, mutta insinöörityötä tehdessä kiinnitettiin myös huomiota simuloinnin optimoimiseen, ja tutkittavien asioiden määrä vaikuttaa huomattavasti tekoälyn harjoitteluaikaan. Tässä vaiheessa päätettiin, että agentin ei todennäköisesti tarvitse aistia lähes puoliympyrän leveydellä eteenpäin. (Kuva 17.)



Kuva 17. Agentti vähennetyillä aisteilla.

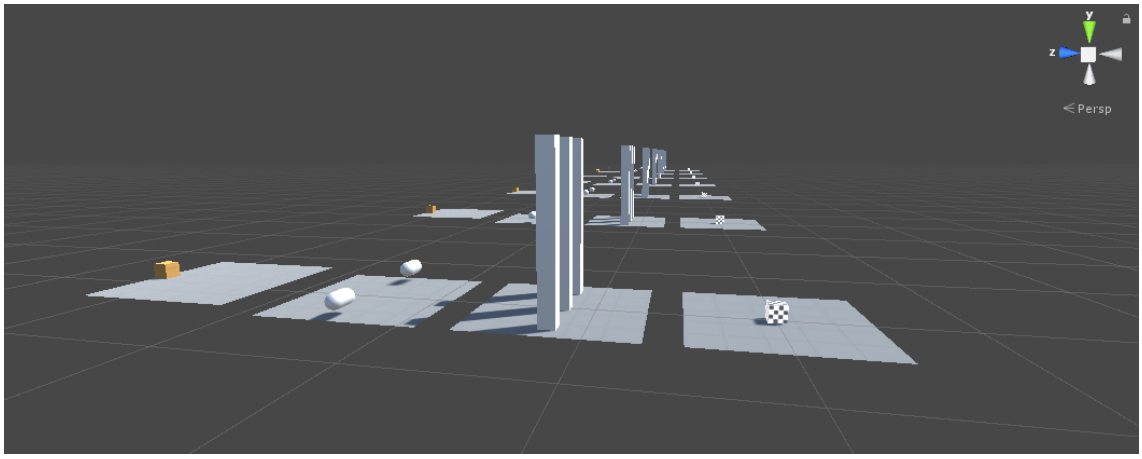
Agentilta vähennettiin molemmalta puolelta kaksi reunimmaista raycastia, jolloin se aisti 70:n ja 110 asteen välillä. Tutkittavien asioiden vektori väheni lukuun 53. Raycastien vähentäminen sai olennaisesti harjoitusajan vähenemään ja parempia tuloksia, sillä agentilla oli vähemmän turhaa tietoa käsiteltävänä. (Kuva 18.)



Kuva 18. Oppimistahdin kuvaaja, josta näkyy oppimisen tasaisuus.

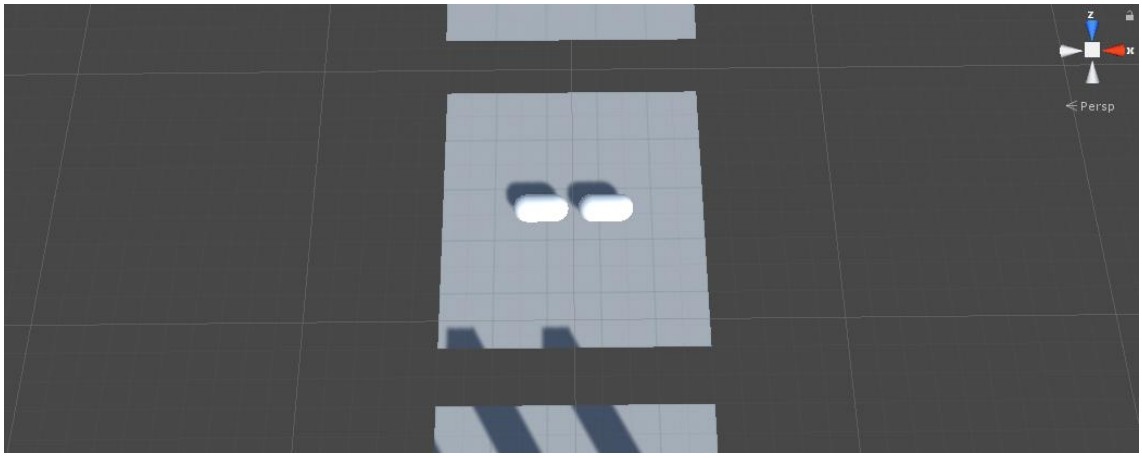
6.3 Liikkuvat esteet

Jälleen kun projektissa päästiin haluttuihin tuloksiin, kenttään lisättiin vaikeusastetta. Kolmannessa versiossa kenttään lisättiin liikkuvia esteitä, joihin törmätessä agentille annettiin sama rankaisu kuin alustalta putoamisesta, ja simulaatio alkoi alusta. (Kuva 19.)



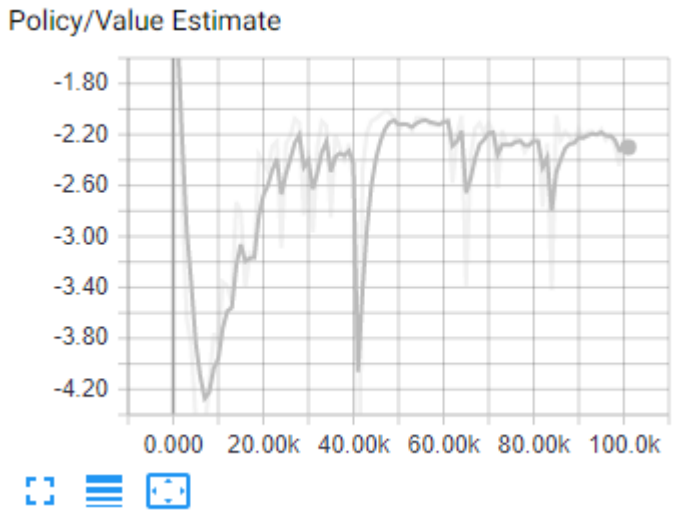
Kuva 19. Projektin kolmas taso: liikkuvat esteet.

Toisella alustalla olevia esteitä myös pidennettiin, jotta tekoäly ei pystyisi menemään niiden yli. Suurin vaikeus kuitenkin aiheutui selvästi liikkuvista esteistä. Esteet liikkuvat edestakaisin lähes kohdaten ja palaten takaisin alkupisteeseensä. (Kuva 20.)



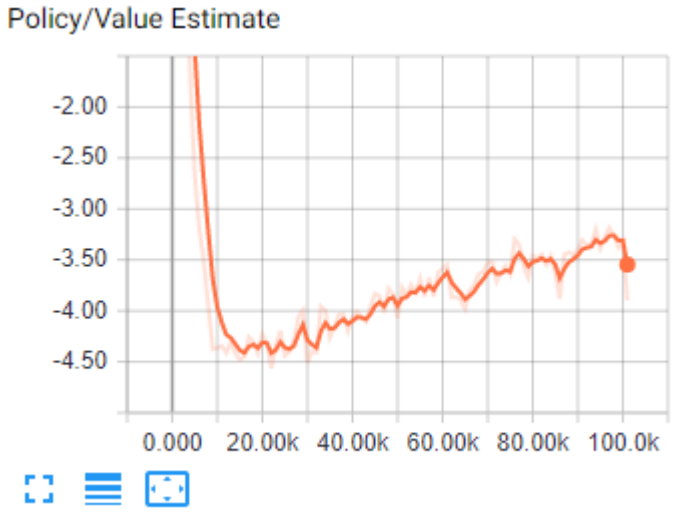
Kuva 20. Liikkuvat esteet lähellä toisiaan.

Heti ensimmäisillä harjoituskerroilla ilmeni, että pelkästään esteiden aistiminen ei riitä, sillä tekoäly oppii nopeasti välttelemään osumia jarruttamalla paikallaan. Esteiden välttämisen tarkentamiseksi lisättiin agentin tutkintafunktioon esteiden senhetkinen liikuntanopeus. Tämä paransi tulosta hieman, mutta kuitenkaan haluttuun lopputulokseen ei vielä päästy. Agentin etenemiselle asetettiin kannusteeksi 0,2:n arvoinen palkinto, kun se ohitti liikkuvat esteet. Palkinto yhdistettynä pieneen uteliasuusarvoon sai agentin kokeilemaan liikkuvien esteiden ohittamista. (Kuva 21.)



Kuva 21. Tason 3 palkintojen kuvaaja: uteliaisuudesta johtuva palkintojen hajanaisuus.

Palkinnot olivat hieman heittelevämpiä kuin ennen (kuva 22), osittain luultavasti johtuen uteliaisuudesta, joka laitettiin päälle tässä tasossa. Uteliaisuuden tarkoituksena on saada agentti valitsemaan silloin tällöin jokin muu kuin optimaalisin ratkaisu sen laskelmien mukaan. Tässä tasossa uteliaisuudella oli tarkoitus saada agentti kokeilemaan pääsyä esteiden ohi, vaikka se oli jo törmännyt niihin ennen ja mahdollisesti ymmärtänyt, että on parempi olla ottamatta sitä riskiä.

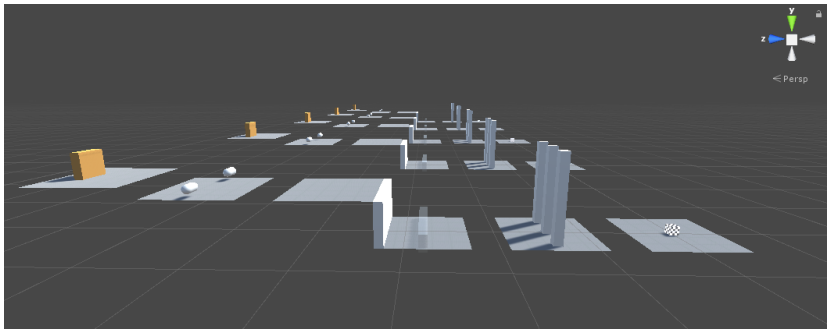


Kuva 22. Tason 3 tasaisemmat palkinnot ilman uteliaisuutta.

Kuvassa 22 näkyy, kuinka uteliaisuuden jättäminen pois saa palkinnot nousemaan tasaisemmin, mutta lopputulos on huonompi. Lopulta uteliaisuudella kehitetty tekoäly pääsi maaliin noin 10 minuutin opetuksella halutun usein, mutta liikkuvat esteet olivat silti vielä selvästi vaikeimmat ja aiheuttivat häviötä silloin tällöin. Yleisin onnistumiseen johtanut tekoälyn strategia oli lopulta ohittaa esteet hyppäämällä sivulta.

6.4 Neuroverkon kerroksien lisäys ja kurssioppiminen

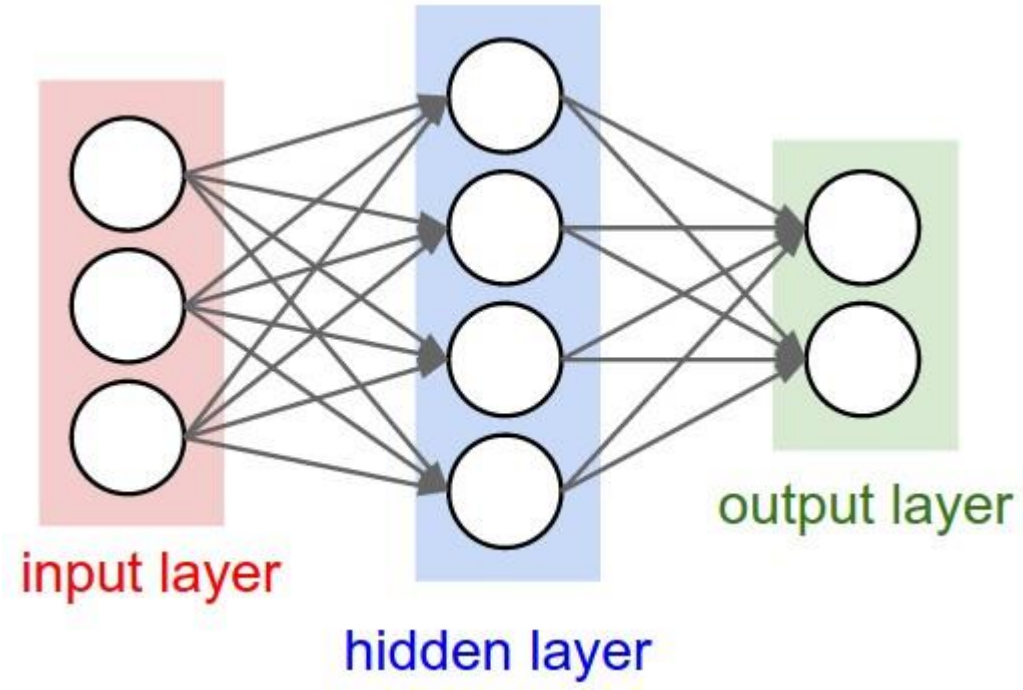
Seuraavassa tasossa kenttään lisättiin hyppyri, jonka kautta agentin täytyy mennä päästäkseen sen edessä olevasta seinästä eteenpäin (kuva 23).



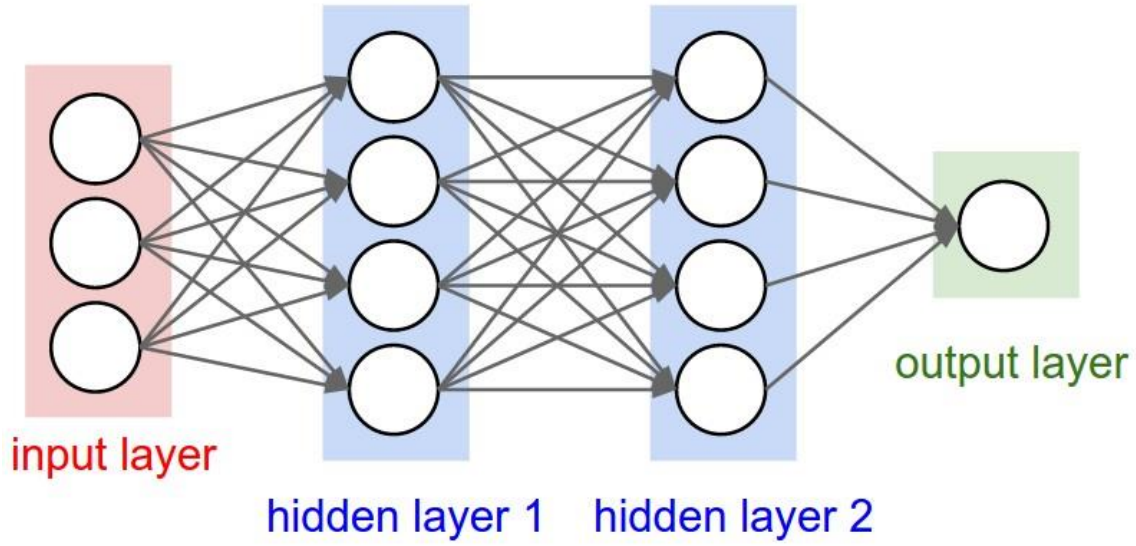
Kuva 23. Projektin neljäs taso: hyppyri ja korkea seinä.

Kolmannella alustalla oleva hyppyri tehtiin antamalla agentille impulsiivinen voima y-akselin suuntaisesti agentin osuessa hyppyriin. Tämä voima tehtiin tarpeelliseksi, sillä hyppyrin edessä olevaa seinää ei pysty ylittämään pelkästään hyppäämällä. Agentin tuli myös osua hyppyriin oikeassa kulmassa, jotta se ei lentäisi viistossa ulos alustalta.

Aluksi agentilla oli selvästi ongelmia löytää hyppyri ja pysyä kentällä hyppyriin osumisen jälkeen. Edelliseen kenttään käytetyllä agentin aivojen algoritmeilla ei saatu haluttuja tuloksia, joten agentin toimintoja oli muutettava. Agentille annettiin ensimmäiseksi pieni palkinto, kun se osui hyppyriin. Tämä sai agentin selvästi löytämään hyppyrin useammin ja siten kehittymään sen käytössä sekä vähentämään jarruttelua ja riskien välttämistä. Oppimisen optimoinnin vuoksi hyppyrin palkintoa muokattiin myöhemmin antamalla hieman suurempi palkinto sen perusteella, kuinka lähellä hyppyrin keskiosaa agentti osui hyppyriin. Taso oli kuitenkin jo tässä vaiheessa sen verran monimutkainen, että heittelevää toimintaa esiintyi silloin tällöin noin 10 minuutin harjoitusjaksoilla. Tässä vaiheessa projektia käytettiin kaksikerroksista neuroverkkoa (kuva 24), joka on oletuksena PPO-algoritmin hyperparametreissa. Kenttä oli kuitenkin neljännessä tasossa kehittynyt tarpeeksi monimutkaiseksi, että neuroverkon kerroksia kokeiltiin nostaa. (Kuva 25.)



Kuva 24. Esimerkki kaksikerroksesta neuroverkosta [17].



Kuva 25. Esimerkki kolmikerroksisesta neuroverkosta [17].

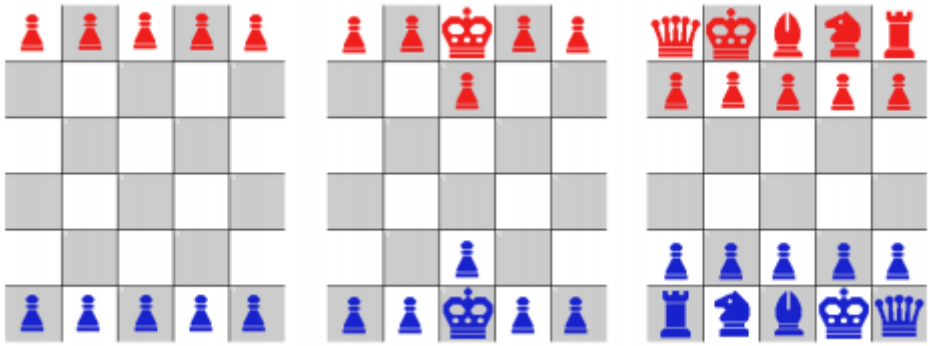
Neuroverkkojen kerroksien lisäämisellä oli tarkoitus saada tekoälyn päätöksentekoon lisää dynaamisuutta ja monimutkaisuutta. Nostamalla samalla harjoitteluaikaa noin 15 minuuttiin huomattiin, että tekoäly osaa selvästi käsitellä ongelmia korkeammalla tasolla. Simulaation alussa huomattiin myös, kuinka yhden kerroksen lisääminen neuroverkkoon hidasti hieman tekoälyn etenemistä helpommista ongelmista. Tuloksena saatiin siis tekoäly, joka käsittelee ongelmat kehittyneemmin, mutta vaatii enemmän aikaa toimiakseen paremmin.

Unityn ML-Agents-työkalupakissa olevaa kurssioppimista kokeiltiin myös ottaa käyttöön tässä projektin vaiheessa. Kurssioppiminen toteutettiin lukemalla JSON-tiedostosta parametrit kentän päivittämiseen (esimerkkikoodi 2).

```
{
    "measure" : "reward",
    "thresholds" : [0.1, 0.3, 0.5],
    "min_lesson_length" : 10,
    "signal_smoothing" : true,
    "parameters" :
    {
        "map_difficulty" : [0.0, 1.0, 2.0, 3.0]
    }
}
```

Esimerkkikoodi 2. Kurssioppimiseen käytetty JSON-tiedosto.

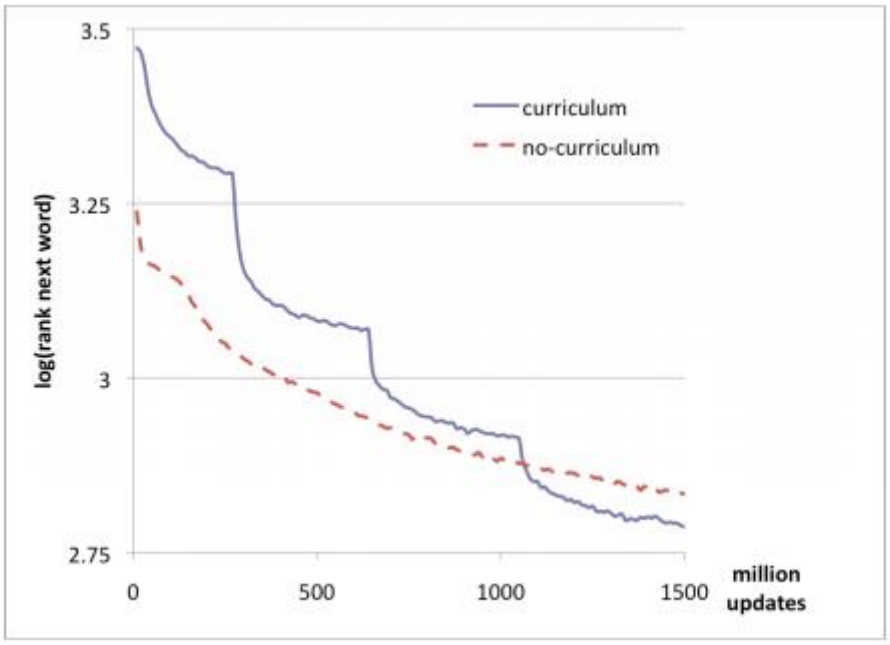
Kurssioppiminen toteutettiin perustuen agentin saamiin palkintoihin. Ainoana parametrina käytettiin tason vaikeutta, jonka arvon perusteella tasoa muokattiin koodissa. Kentälle tehtiin neljä eri vaikeusastetta, joita nostettiin kun agentti pääsi seuraavaan palkintoasteeseen 0,1:n, 0,3:n ja 0,5:n suuruisten palkintojen välillä, samalla tavoin kuin mahdollisesti kuvassa 26.



Kuva 26. Mahdollisen kurssioppimisen eri vaiheet shakissa [18].

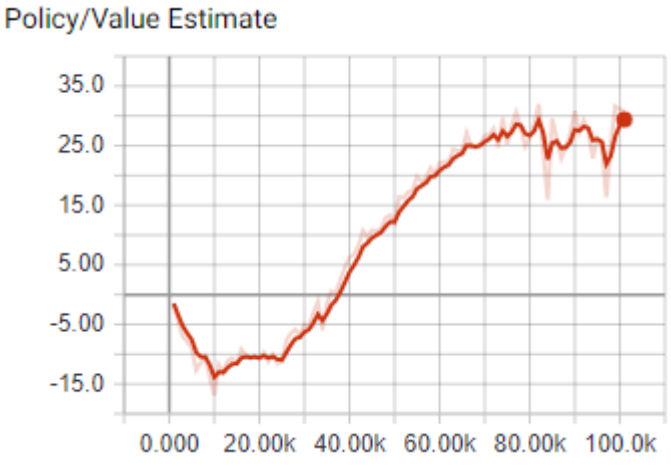
Ensimmäisellä asteella agentin tuli vain hypätä toiselle alustalle. Toisessa vaikeusasteessa agentilla oli myös edessään pylväät, jotka agentin tuli ohittaa. Kolmannessa versiossa tasoon lisättiin hyppyri, ja viimeinen vaikeusaste sisälsi koko neljännen tason.

Kurssioppimisesta puhutaan usein tehokkaampana ja nopeampana tapana opettaa tekoäly kuin ilman kurssioppimista, kuten nähdään kuvassa 27. Projektissa käytetyn kurssioppimisen kanssa tehdyissä simulaatioissa ei kuitenkaan ilmennyt huomattavia eroja verrattuna simulaatioihin, joissa ei käytetty kurssioppimista. Välillä tulokset olivat jopa hajanaisempia kurssioppimisen kanssa. Tämän vuoksi tultiin lopputulokseen, että kurssioppiminen olisi luultavasti vaatinut enemmän hiomista toimiakseen kunnolla. Lisäksi todettiin, että projekti ei todennäköisesti ollut liian monimutkainen toteutettavaksi ilman tätä tekniikkaa, joten kurssioppiminen päätettiin jättää kokeilun jälkeen pois.



Kuva 27. Wikipedian sanojen tunnistamiseen opetetun tekoälyn kuvaajassa vertailtuna kurssiop-piminen ja tavallinen oppiminen [19].

Lopulta noin 15 minuutin harjoittelulla ja 3 kerroksen neuroverkolla päästiin tuloksiin, joissa tekoäly pääsi maaliin saakka usein (kuva 28).

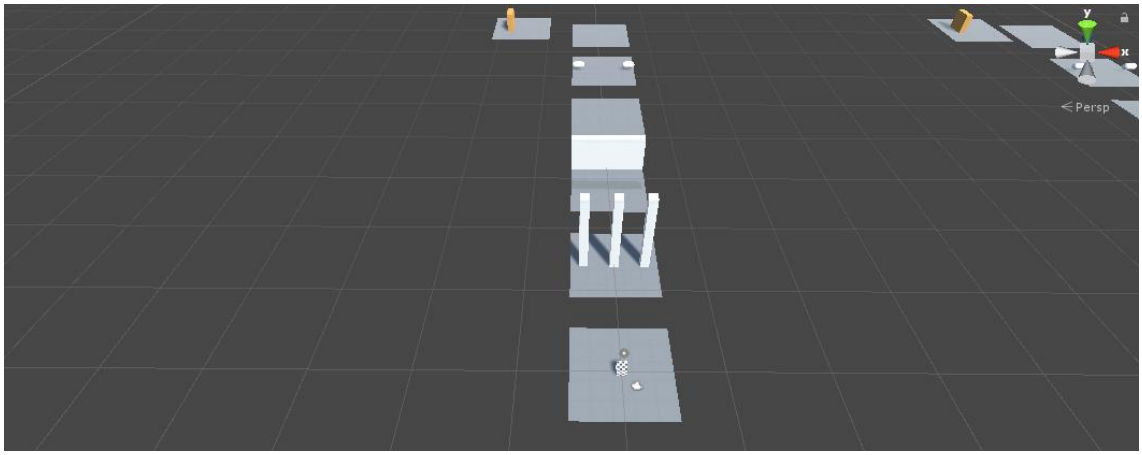


Kuva 28. Tason 4 palkintokuvaaja: palkintojen nousu tasaantuu tekoälyn päästyä hyvälle tasolle.

Agentin saamat palkinnot tasaantuivat noin 80 000 simuloinnin askeleen kohdalla, jolloin suurin hyöty oppimisesta käytetyillä asetuksilla aikaan suhteutettuna oli saavutettu. Vieläkin tekoälylle vaikein ongelma oli kuitenkin selvästi liikkuvat esteet, joihin tekoäly osui silloin tällöin. Välillä tekoäly ohjasi hyppyriin myös liian kovalla nopeudella, jolloin se lensi selvästi ulos kentältä.

6.5 Imitaatio-oppiminen ja uteliaisuus

Projektin viimeiseen kenttään lisättiin tekoälylle opeteltavaksi kääntyminen. Kenttään lisättiin alusta 90 asteen kulmassa ja selvitettiin, kuinka koneelta onnistuu sen havaitseminen ja sen mukaan kääntyminen. (Kuva 29.)

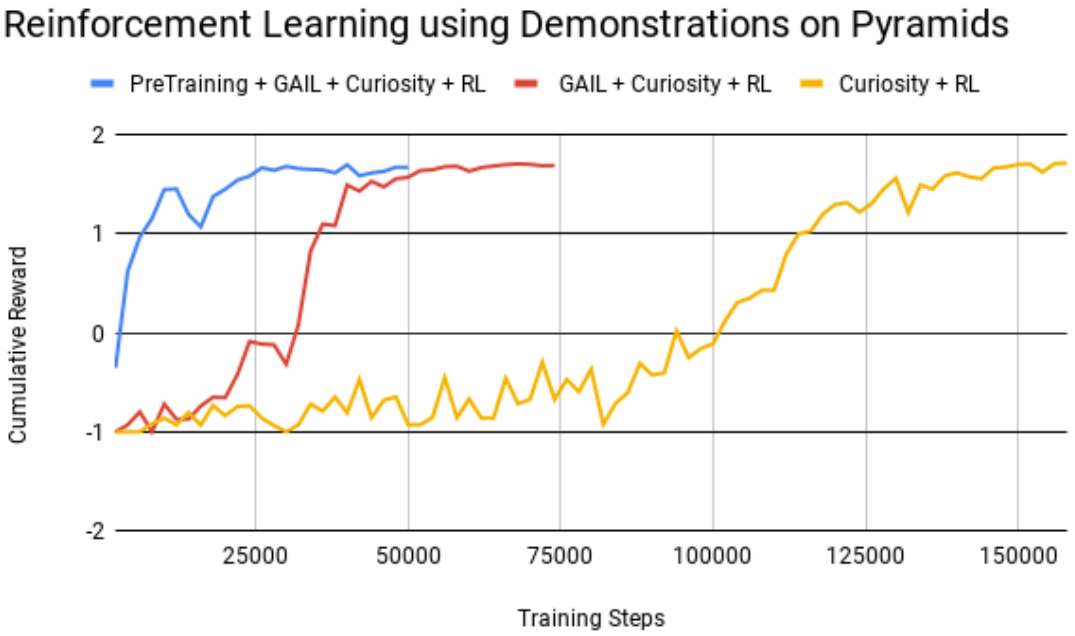


Kuva 29. Projektin viides taso: kääntyminen sivulle.

Suurimmaksi vaikeudeksi koneelle aiheutui sen nopeuden säätely, sillä viime tasossa kone pääsi usein maaliin ottaen hyppyristä nopeutta ja menemällä vauhdilla ohi esteistä. Hyppyrin ja liikkuvien esteiden vaikeus aiheutti tekoälylle hankaluuksia päästä siihen kohtaan, jossa sen tulisi oppia kääntymään maaliin kohti.

Unityn ML-Agents-työkalupakissa on myös käytössä imitaatio-opettelu, joka otettiin käyttöön tässä tasossa. Imitaatio-opettelu käyttää valmiiksi nauhoitettua dataa pelaajan pelauskerroista, minkä perusteella se oppii etenemään tasossa. Opetetun datan vaikutusta tekoälyn toimintaan voi säätää hyperparametreilla joko lähes kloonattuun toimintaan tai suuntaa antavaksi. Imitaatio-oppimisen voi yhdistää vahvistusoppimiseen, kuten projek-

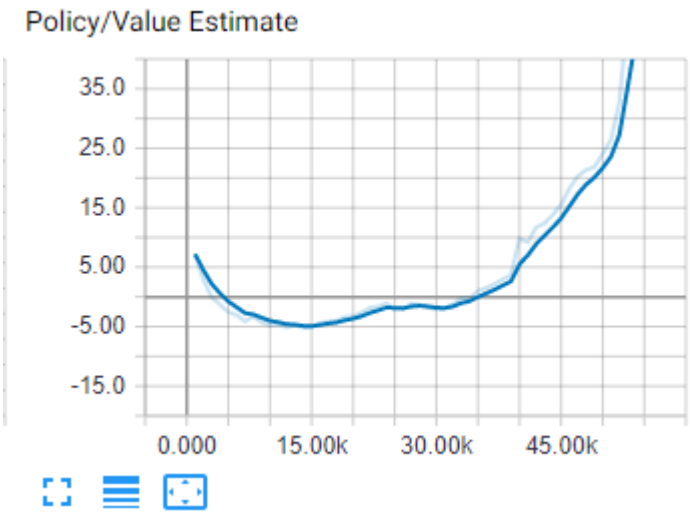
tissa tehtiin, jolloin se toimii ympäristöstä saatujen palkintojen kanssa. Imitaatio-oppimisen tulisi nostaa oppimisnopeutta ja tehdä oppimisesta tasaisempaa, kuten nähdään kuvassa 30.



Kuva 30. Vertailu eri oppimistyylien välillä: imitaatio-oppimisella yhdistettynä uteliaisuuteen saadaan usein paremmat tulokset kuin tavallisella oppimisella [20].

Imitaatio-oppimisen käyttöönoton jälkeen huomattiin heti, kuinka tekoäly osaa jo aikaisessa vaiheessa harjoittelua yrittää hakea palkintoja tason loppuosista. Ilmeni, että imitaatio on erinomainen tapa saada tekoäly hakemaan suurempia palkintoja vaikeiden esteiden takaa ottaen riskin häviöstä sen sijaan, että se välttelisi riskejä. Projektissa käytetyssä imitaatio-oppimisessa käytettiin esioppimista, GAIL-palkintosignaalia, uteliaisuutta ja vahvistusoppimista, sillä se on usein paras tyyli auttaa tekoälyä oppimaan ja silti käyttää vahvistusoppimisalgoritmeja.

Lopulta päästiin tulokseen, jossa imitaatiota käyttämällä tekoäly pääsi useasti maaliin saakka noin 20 minuutin harjoittelulla (kuva 31).



Kuva 31. Viidennen tason palkintokuvaaja, josta on nähtävissä, kuinka imitaatio-oppiminen mahdollistaa myös vaikeiden palkintojen saamisen, vaikka tekoäly ei löytäisi niitä simulaation alussa.

Yhden simulaation palkintokuvaajasta näkyy, kuinka tekoälyn oppiessa imitaation ja vahvistusoppimisen kautta, miten kenttä tulee pelata, sen saamat palkinnot lähtivät jyrkkään nousuun. Imitaatio-oppimisen todettiin olevan sopiva tämänkaltaisiin ongelmiin, joiden ratkaiseminen vaatii muuten tekoälyltä paljon riskien ottoa ja kokeilua.

6.6 Tulokset

Tekoäly saavutti riittävän tason selviytyäkseen suurimmasta osasta sille tehdyistä ongelmista. Tekoälyn liikkumisen todettiin olevan pääpiirteiltään järkevän kaltaista, mutta hienovaraisiin liikkeisiin se ei kyennyt. Todettiin, että hyperparametrit on tärkeää säätää oikein riippuen tekoälylle asetetusta ongelmasta. Kokeilluista oppimistekniikoista kurssioppimiselle ei löydetty tässä projektissa toteutustapaa, jolla olisi saatu oppimiselle suunnattua hyötyä. Imitaatio-oppimisen todettiin kuitenkin olevan tekoälyn oppimista edistävä tekniikka, varsinkin tilanteissa, joissa tekoäly löytää harvoin palkintoja vaikeiden esteiden takia.

7 Yhteenveto

Koneoppiminen on suhteellisen uusi asia, ja varsinkin sen käyttö pienissä projekteissa on mahdollistettu vasta viime aikoina. Asian teknisistä ominaisuuksista on löydettävissä paljon tietoa, mutta itse toteutus omissa projekteissa on usein omista kokemuksista saadun tiedon ja kokeilun varassa.

Insinöörityössä selvitettiin koneoppineen tekoälyn toimivuutta sille tehdyssä pienimittaisessa 3D-tasohyppelypelissä. Tasohyppely tehtiin pitäen prioriteetti tekoälyn toiminnassa, joten itse tason tarkoitus oli tarjota tekoälylle mahdollisimman monipuolisia esteitä. Tämän myötä saatiin kehitettyä tekoäly, joka luultavasti selviäisi pidemmässäkin tasohyppelypelissä, jossa esteet ovat usein jollain tapaa samanlaisia. Samalla todettiin, että koneoppineen tekoälyn on mahdollista selvitä suuremmastakin pelistä ja vaikeista ongelmista.

Projektissa tehty tekoäly ei toimi täysin aukottomasti, mutta siinä on nähtävissä elementtejä, joista huomaa, että koneoppimisella on selvät mahdollisuudet ylittää ihmisen potentiaali tasohyppelypeleissä. Pitkillä harjoitusajoilla tekoälyn suoritusta oli usein vaikea voittaa silloin, kun kone oli opetettu hyvillä asetuksilla.

Erittäin suurilla harjoitusajoilla ja monimutkaisemmilla neuroverkoilla sekä yleisesti päätöksenteolla saadaan aikaan jo nykypäivänä kehittyneitä tekoälyjä, jotka voittavat ihmisiä monimutkaisissa peleissä. Tulevaisuudessa koneoppiminen luultavasti saa jatkuvasti lisää suosiota, sillä sen saavutukset ja käytettävyys vetävät puoleensa yrityksiä ja harrastelijoita. Varsinkin indie-projekteissa on tulevaisuudessa todennäköisesti helpompaa ottaa käyttöön koneoppimista.

Lähteet

1 McCulloch, Warren & Pitts, Walter. A Logical Calculus of the Ideas Immanent in Nervous Activity. Verkkoaineisto. <https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf> Luettu 2.9.2029.

2 Markoff, John. 2012. How Many Computers to Identify a Cat? 16,000. Verkkoaineisto <https://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html> Luettu 15.11.2019.

3 Chowdhry, Amit. 2014. Facebook's DeepFace Software Can Match Faces With 97.25% Accuracy. Verkkoaineisto. <https://www.forbes.com/sites/amitchowdhry/2014/03/18/facebooks-deepface-software-can-match-faces-with-97-25-accuracy/#7480c73154fc> Luettu 15.11.2019.

4 Statt, Nick. 2019. OpenAI's Dota 2 AI steamrolls world champion e-sports team with back-to-back victories. Verkkoaineisto. <https://www.theerge.com/2019/4/13/18309459/openai-five-dota-2-finals-ai-bot-competition-og-e-sports-the-international-champion> Luettu 2.9.2019.

5 Lorberfeld, Audrey. Machine Learning Algorithms In Layman's Terms, Part 1. Verkkoaineisto. <https://towardsdatascience.com/machine-learning-algorithms-in-laymans-terms-part-1-d0368d769a7b> Luettu 15.11.2019.

6 Wilson, Aidan. A Brief Introduction to Supervised Learning. Verkkoaineisto. <https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590> Luettu 15.11.2019.

7 Seif, George. An easy introduction to unsupervised learning with 4 basic techniques. Verkkoaineisto. <https://towardsdatascience.com/an-easy-introduction-to-unsupervised-learning-with-4-basic-techniques-da7fbf0c3adf> Luettu 15.11.2019.

8 Kapoor, Sanyam. Policy Gradients in a Nutshell. Verkkoaineisto. <https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d> Luettu 2.9.2019.

9 Agrawal, Samarth. Hyperparameters in Deep Learning. Verkkoaineisto. <https://towardsdatascience.com/hyperparameters-in-deep-learning-927f7b2084dd> Luettu 15.11.2019.

10 Richárd, Nagyfi. The differences between Artificial and Biological Neural Networks. Verkkoaineisto. <https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7> Luettu 4.9.2019.

11 Haritou, Maria. Typical structure of a feed-forward multilayer network. Verkkoaineisto. <https://www.researchgate.net/figure/Typical-structure-of-a-feed-forward-multilayer-neural-network_fig1_291339457> Luettu 5.9.2019.

12 Violante, Andre. Simple Reinfrocement Learning: Q-learning. Verkkoaineisto. <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56> Luettu 16.11.2019.

13 Unity ML-Agents: what it is, how it works, how to use it. Verkkoaineisto. Unity-Technologies <https://unity3d.com/how-to/unity-machine-learning-agents> Luettu 7.9.2019.

14 Creating an Academy. Verkkoaineisto. Unity-Technologies <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design-Academy.md> Luettu 16.11.2019.

15 Training with Proximal Policy Optimization. Unity-Technologies. Verkkoaineisto. <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md> Luettu 16.11.2019.

16 Training with Curriculum Learning. Verkkoaineisto. Unity-Technologies <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Curriculum-Learning.md> Luettu 7.9.2019.

17 Convolutional Neural Networks for Visual Recognition. Verkkoaineisto. <http://cs231n.github.io/neural-networks-1/> Luettu 2.10.2019.

18 Narverkar, Sammit; Sinapov, Jivko; Leonetti, Matteo & Stone, Peter. Source Task Creation for Curriculum Learning. Verkkoaineisto. Department of Computer Science, University of Texas at Austin.<https://www.cs.utexas.edu/~sanmit/papers/AAMAS16-Narvekar.pdf> Luettu 3.10.2019.

19 Bengio, Yoshua; Louradour, Jérôme; Collobert, Ronan & Weston, Jason. 2009. Curriculum Learning. Verkkoaineisto. <https://ronan.collobert.com/pub/matos/2009_curriculum_icml.pdf> Luettu 3.10.2019.

20 Training with Imitation Learning. Verkkoaineisto. Unity-Technologies. https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Imtation-Learning.md Luettu 4.1.2019.