Timo Visuri

KONEOPPIMINEN UNITY-PELIMOOTTORISSA

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutus
Helmikuu 2023

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Helmikuu 2023	**Tekijä/tekijät** Timo Visuri
Koulutus Tieto- ja viestintätekniikan koulutus		☒ AMK ☐ YAMK
Työn nimi KONEOPPIMINEN UNITY-PELIMOOTTORISSA		
Työn ohjaaja Sari Lipsanen		**Sivumäärä** 29

Insinöörityön päätarkoituksena oli tutkia koneoppimisen soveltuvuutta pelikehityksessä. Työssä koneoppimisen soveltuvuutta tutkittiin erityisesti pienen peliprojektin näkykulmasta. Lisäksi työssä selvitettiin yleisesti koneoppimisen perusteita. Tavoitteena oli luoda tekoälylle haastava tilanne, jossa koneoppimisen rajoja voitiin testata helposti.

Tekoälyn koulutusta varten luotiin 2D-peli Unity-pelimoottorilla. Koneoppimiseen käytettiin Unity ML-Agents-työkalupakettia. Tekoälyn oppimistuloksien seuraamiseen käytettiin TensorBoard-työkalua.

Työtä varten kehitettiin pieni avaruuslentelypeli. Pelissä tarkoituksena oli voittaa vastustaja tuhoamalla vastustajan alus. Koulutus tapahtui käyttämällä self-play-koulutusmenetelmää, jossa tekoäly harjoittelee omaa klooniansa vastaan.

Lopputuloksena todettiin, että ML-Agents-työkalun käyttö oli todella helppo aloittaa ja sen avulla pystyi nopeasti toteuttamaan yksinkertaisen tekoälyn. Täysin toimivaa tekoälyä ei kuitenkaan saatu valmiiksi, mutta toisaalta peliin suunnitellulle tekoälylle asetettiin melko korkeat vaatimukset, jotta pystyttiin selvittämään, mihin koneoppiminen pystyy. Koneoppimisen hyödyt ja ongelmat tulivat työssä hyvin esille. Koneoppimisen käyttö pienissä peleissä vaikutti varteenotettavalta vaihtoehdolta, jos pelin tekoäly oli suunniteltu hyvin.

Asiasanat
Koneoppiminen, tekoäly, Unity

ABSTRACT

Centria University of Applied Sciences	**Date** February 2023	**Author** Timo Visuri
Degree programme Information and Communications Technology		
Name of thesis MACHINE LEARNING IN UNITY GAME ENGINE		
Centria supervisor Sari Lipsanen		**Pages** 29

The main purpose of the thesis was to investigate the applicability of machine learning in game development. In particular, the suitability of machine learning was investigated from the perspective of a small game project. In addition, the thesis explained the basics of machine learning in general. The aim was to create a challenging situation for AI where the limits of machine learning could be easily tested.

To train the AI, a 2D game was created using the Unity game engine. The Unity ML-Agents toolkit was used for machine learning. The TensorBoard tool was used to monitor the AI's learning results.

A small spaceflight game was developed for this work. The goal of the game was to defeat the opponent by destroying the opponent's ship. Training was done using the self-play training method, where the AI trains against its own clone.

The end result was that the ML-Agents toolkit was really easy to get started with and allowed for a quick implementation of a simple AI. However, it was not possible to complete a fully functional AI, but on the other hand, the AI designed for the game was set quite high standards in order to see what machine learning could do. The benefits and problems of machine learning were well illustrated in the work. The use of machine learning in small games seemed to be a viable option if the game's AI was well designed.

Key words
Artificial Intelligence, machine Learning, Unity

KUVAT

1 JOHDANTO

Viime vuosien aikana termeistä tekoäly ja koneoppiminen on tullut muotisanoja. Vaikka tekoäly ja koneoppiminen liittyvät erottamattomasti toisiinsa ja niillä on samankaltaisia ominaisuuksia, ne eivät kuitenkaan ole sama asia; pikemminkin koneoppiminen on tekoälyn osa-alue. Tässä opinnäytetyössä perehdytään koneoppimiseen pelikehityksen näkökulmasta.

Opinnäytetyössä tutustutaan Unity Technologiesin kehittämän ML-Agents Toolkitin käyttöön. Työn tarkoituksena on testata, kuinka helppoa ML-agentsin käyttäminen on luomalla yksinkertainen peli. Työssä perehdytään ML-agentsin soveltuvuuteen pelihahmon tekoälyn luomisessa ja parhaisiin käytäntöihin. Tavoitteena on myös selvittää, kuinka nopeasti tekoälyn opettaminen tapahtuu. Tekoälyn opettamista varten tehdään bullet hell-tyyppinen peli, jossa tekoäly oppii pelaamalla itseään vastaan.

Aluksi työssä perehdytään koneoppimiseen yleisellä tasolla ja tutustutaan koneoppimisen eri muotoihin. Työssä käydään myös tarkemmin läpi, miten ML-agents toimii ja mitkä ovat sen tärkeimmät komponentit. Opinnäytetyön keskeisenä osana on käydä läpi, miten koneoppiminen toteutetaan konkreettisesti Unity-pelimoottorissa. Lopussa käsitellään työn tuloksia, haasteita ja työn aikana esille tulleita ongelmia. Tuloksissa saadaan selville koneoppimisen soveltuminen luotuun peliympäristöön.

2 KONEOPPIMINEN

Viime vuosina koneoppimisen edistysaskeleet ovat mahdollistaneet läpimurtoja muun muassa kuvien tunnistamisessa, tekstin kääntämisessä, puheen tunnistamisessa ja pelien pelaamisessa. Kun koneoppiminen kehittyy, se muuttaa tapaa, jolla pelejä kehitetään, miten tekstuurit ja 3D-mallit luodaan ja miten ei-pelattavat hahmot ohjelmoidaan.

Suurin osa nykyisistä peleistä käyttää tekoälyratkaisuja, jotka ovat käsin ohjelmoituja ja koostuvat tilakoneista, joissa saattaa olla jopa tuhansia sääntöjä. Niiden ylläpito ja testaus on hankalaa ja hidasta. Koneoppiminen pystyy ymmärtämään raakadataa, ilman että asiantuntijan on määriteltävä, miten dataa tulkitaan. Yleinen esimerkki, johon koneoppimista voidaan käyttää, on kuvien luokittelu sen mukaan, mitä kuvissa näkyy. Koneoppiminen sekä erityisesti syväoppiminen tarvitsee oppimista varten kuvat ja tiedon, mitä kuva esittää. Algoritmille annetaan palautetta oikeista ja vääristä päätöksistä, ja sen kautta se oppii tunnistamaan kuvat tarkasti, myös silloin kun kuvien sisältöä ei tiedetä ennalta. Tämä automaattinen oppiminen auttaa yksinkertaistamaan ja nopeuttamaan pelien luomisprosessia. (d'Archimbaud.)

Koneoppimista voidaan soveltaa tietokoneen ohjaamien pelihahmojen käyttäytymiseen. Vahvistusoppimista voidaan käyttää kouluttamaan tekoäly arvioimaan peliympäristössä suoritettavien toimintojen painoarvoa. Kun tekoäly on koulutettu, se pystyy tekemään toimintoja, joista saa eniten painoarvoa, ilman että sitä tarvitsee erikseen ohjelmoida, miten toimia eri tilanteissa.

2.1 Koneoppimisen eri tyypit

Koneoppiminen jaetaan eri osa-alueisiin riippuen siitä, millaisia ongelmia on tarkoitus ratkaista. Yleisimmät koneoppimismallit ovat ohjaamaton koneoppiminen, ohjattu koneoppiminen ja vahvistusoppiminen. On kuitenkin olemassa myös eri menetelmien yhdistelmiä, kuten puoliohjattu koneoppiminen. (Brownlee 2019.)

2.2 Ohjaamaton koneoppiminen

Ohjaamatonta koneoppimista käytetään yleensä silloin, kun on paljon merkitsemätöntä dataa ja se halutaan jakaa ryhmiin. Ohjaamaton oppiminen pystyy löytämään datasta rakenteita, joita ihminen ei

välttämättä huomaa. Ohjaamatonta oppimista käytetään esimerkiksi verkkokauppojen mainonnassa, jolloin jonkin tuotteen ostajalle voidaan suositella tuotteita, sen mukaan millaiseen käyttäjäryhmään asiakas kuuluu. Yhtenä esimerkkinä voidaan ajatella jonkin videopelin pelaajia. Jaetaan pelaajat kahteen ryhmään, sen mukaan kuinka sitoutuneita he ovat peliin. Voimme ensin määritellä pelaajille attribuutit, kuten pelatut tunnit, oikealla rahalla pelissä tehdyt ostokset ja läpäistyjen pelikenttien määrä. Tämän jälkeen ohjaamattomalle algoritmille annetaan luotu data, ja määritellään, että pelaajat jaetaan kahteen ryhmään. Tässä tapauksessa lopputulos olisi, että kaikki pelaajat on jaettu kahteen ryhmään, joista toisessa on sitoutuneet pelaajat ja toisessa sitoutumattomat. (Background: Machine Learning.)

Tässä esimerkissä ohjaamattomalle koneoppimiselle ei annettu erikseen esimerkkejä, mitkä pelaajat tulisi laittaa sitoutuneiden ryhmään ja mitkä sitoutumattomien ryhmään. Määriteltiin vain attribuutit ja luotettiin siihen, että algoritmi löysi nämä kaksi ryhmää itsestään.

2.3 Ohjattu koneoppiminen

Ohjatussa oppimisessa datalla on jokin tunniste, jolla voidaan luokitella sen sisältö. Algoritmille kerrotaan datasta oikea lopputulos. Otetaan esimerkiksi automaattinen kuvantunnistusohjelma. Ennen asiantuntijat joutuivat kirjoittamaan käsin suodattimia, jotka poimivat piirteitä kuvista ja näiden avulla kuvista voitiin tunnistaa sisältävätkö ne esimerkiksi kissoja vai koiria. Ohjatun oppimisen algoritmille voidaan syöttää kuvia, jotka on luokiteltu kuvassa esiintyvän asian nimen mukaan. Kun algoritmi opettelee tunnistamaan ennalta annettuja kuvia, sille annetaan palautetta oikeista ja vääristä päätöksistä, ja sen kautta se oppii tunnistamaan kuvat tarkasti myös silloin, kun kuvien sisältöä ei tiedetä ennalta. Ohjatun oppimisen huono puoli on, että data pitää luokitella etukäteen ja se voi olla työlästä suurien datamäärien kohdalla. (Elements of AI.)

2.4 Vahvistusoppiminen

Vahvistusoppiminen on tämän opinnäytetyön kannalta tärkein koneoppimisen tyyli. Vahvistusoppimista voidaan käyttää esimerkiksi itseohjautuvan robotin kouluttamiseen. Robotille voidaan antaa tehtäväksi navigoida labyrintissä ja löytää ulos. Robotti saa dataa ympäristöstä kameran ja kosketusanturin avulla, prosessoi tiedon ja tekee toiminnon, kuten kääntyy tai ajaa eteenpäin. Toisin sanoen robotti tekee toistuvasti valintoja ympäristössä, saadun informaation perusteella. Vahvistusoppimisen tavoitteena on oppia, miten toimia kun robotin sensorit saavat havaintoja ympäristöstä ja tehdä sitten oikea toiminto. Oikeista toiminnoista robotti saa positiivisen palkinnon. Vääristä toiminnoista robotti saa negatiivisen palkinnon. Robotti voi esimerkiksi saada positiivisen palkinnon, jos pääsee labyrintistä ulos

ja negatiivisen jos se törmää seinään tai labyrintin selvittämiseen menee liikaa aikaa. Robotille ei siis kerrota mikä toiminto sen tulisi milloinkin tehdä, vaan sen tulee itse havaita, mistä toiminnoista se saa eniten palkintoa. Tilanteet mistä ja kuinka suuria palkintoja robotti saa, pitää määritellä harkiten. Jos robotti saa negatiivisen palkinnon aina seinään törmätessään, se saattaa oppia pysymään pelkästään paikallaan, koska silloin se minimoi negatiiviset palkinnot. Jos robotille annetaan negatiivisia palkintoja jokaisesta labyrintissä vietetystä sekunnista, sen on järkevämpää yrittää labyrintistä ulos, vaikka se törmäilisikin seiniin. Vahvistusoppimisessa yksittäisillä toiminnoilla ei ole suurta merkitystä kokonaisuuteen nähden. (Background: Machine Learning.)

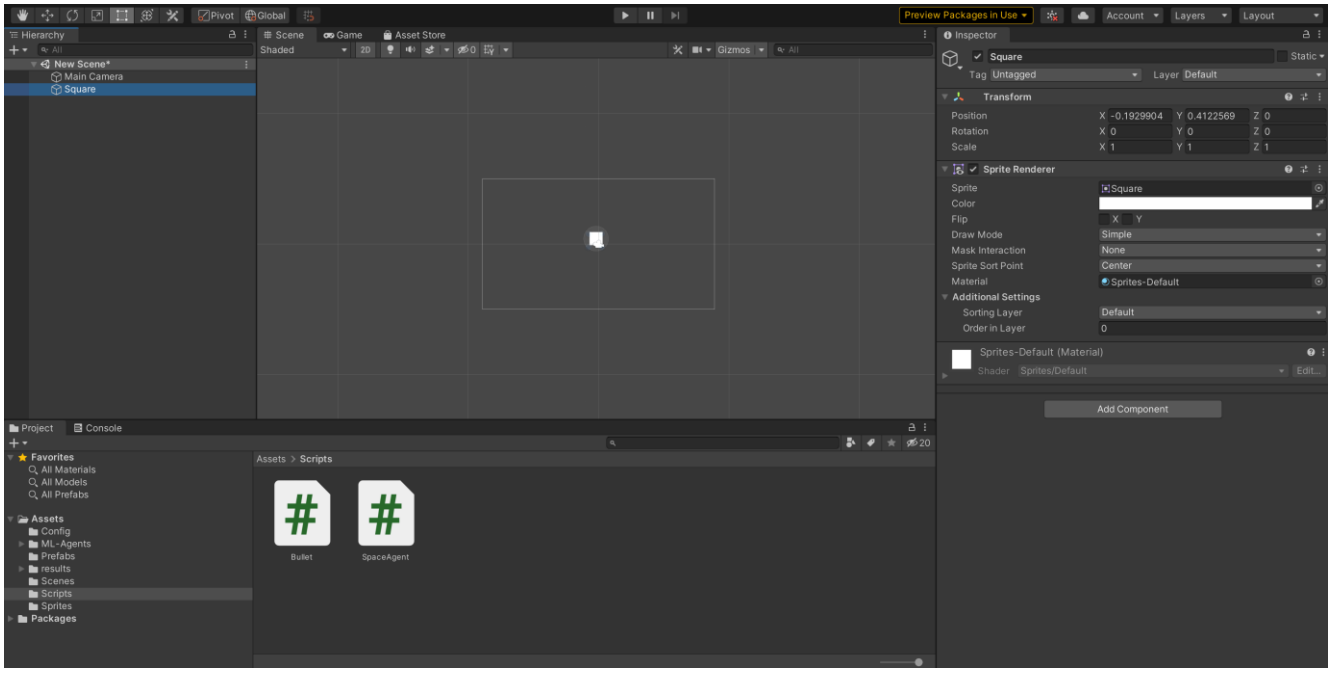
2.5 Imitaatio-oppiminen

Imitaatio-oppiminen perustuu mallisuoritusten imitoimiseen. Imitaatio-oppimisessa ihmisen toiminnot nauhoitetaan ja siitä saatua dataa käytetään pohjana tekoälyn koulutuksessa. (Nigretti 2018.)

3 UNITY JA ML-AGENTS

Unity on Unity Technologiesin kehittämä ilmainen monialustainen pelimoottori ja se julkaistiin kesäkuussa 2005. Unity toimi alun perin yksinään Mac OS X-käyttöliittymässä. Unitylla voidaan kehittää yli 20 eri alustalle, kuten Windowsille, Mac OS:lle, konsoleille ja mobiililaitteille. Unity on yksi käytetyimmistä pelimoottoreista maailmassa. Unitylla voi tehdä erilaisia pelejä hyvin laajasti, aina yksinkertaisista 2d-peleistä realistisiin 3d-peleihin, sekä AR- ja VR-peleihin. Unity on erityisesti suosittu iOS- ja Android-pelien kehityksessä. Unitya käytetään myös muuhun kuin pelikehitykseen, muun muassa erilaisiin visualisointeihin. (Jerga 2021.)

Unity tukee C#-ohjelmointikieltä. Unity tukee myös useita muita alustoja ja sovelluksia kuten GitHub, Visual studio ja MonoDevelop. Unitysta on saatavilla ilmainen versio sekä useampi erilaisia ammattilaisille suunnattuja versiota, joiden hinta vaihtelee. Unity on hyvin aloittelijaystävällinen ja sillä on helppo aloittaa pelikehitys. Verkosta löytyy paljon oppaita Unityn käyttöön. (Jerga 2021.)



KUVA 1. Unityn oletusnäkymä.

Unity editorin oletusnäkymä on jaettu viiteen osaan (KUVA 1). Hierarchy on luettelo Scenen sisältämistä peliobjekteista. Project Browser sisältää kaikki projektissa olevat tiedostot. Inspectorissa lisätään

ja hallitaan peliobjektin komponentteja, sekä muutetaan muuttujien arvoja. Scene näyttää peliscenen ympäristön. Game-ikkuna näyttää pelikameran näkymän.

3.1 ML-Agents

ML-Agent on lisäosa Unity-pelimoottoriin, lisäosan ensimmäinen versio julkaistiin 19.9.2017. Se tarjoaa tutkijoille ja pelintekijöille mahdollisuuden luoda monimutkaisia ympäristöjä ja kouluttaa älykkäitä agentteja. (ML-Agents Toolkit Overview.)

ML-Agents Toolkit on avoimen lähdekoodin projekti, joka käyttää pythonia ja TensorFlowia (avoimen lähdekoodin matematiikkakirjasto), ja sen on kehittänyt pääasiassa Unity Technologies. Toolkit toimii vuorovaikutuksessa Unity Editorin kanssa Python API:n kautta ja sisältää SDK:n, jossa on kaikki ydintoiminnot koulutusympäristön luontiin Unityssa. Toolkitin tavoite on tarjota alusta koneoppimisen hyödyntämiseen virtuaaliympäristöissä. Verrattuna reaalimaailmaan, virtuaaliympäristöjen etuna on skaalautuvat koulutusprosessit, joita voidaan suorittaa rinnakkain sekä nopeutetusti. Toolkitin mukana tulee laaja valikoima työkaluja, joita käytetään älykkäiden agenttien kouluttamiseen. Koneoppimisalgoritmien tulos tallennetaan neuroverkkomalliin, joka voidaan liittää Unityyn. Toolkitin tarjoamat koneoppimisalgoritmit ovat käyttövalmiita, eikä ylimääräistä python-ohjelmointia tarvita. ML-Agents Toolkit on rakennettu PyTorch-kirjaston päälle. Toolkit mahdollistaa minkä tahansa Unity-projektin muuttamisen oppimisympäristöksi, ja sen liittämisen koneoppimisalgoritmeihin. (ML-Agents Toolkit Overview.)

ML-Agents SDK koostuu kolmesta keskeisestä kokonaisuudesta: sensorit, agentit ja akatemia. Agentti on luokka, jolla peliobjekti määritellään agentiksi. Agentti kerää havaintoja sensoreiden avulla, vastaanottaa palkintoja ja tekee toimintoja. Havaintoja agentti kerää siihen liitettyjen erilaisten sensoreiden avulla. Sensorit voivat kerätä tietoa esimerkiksi renderöidyistä kuvista, raycastien tuloksista tai pituusvektoreilla. Lopuksi agentit saavat palkkioita palkitsemisfunktion kautta, jota käytetään oppimissignaalina halutun käyttäytymisen aikaansaanniksi. Akatemia vastaa agenttien instantioinnista ja hallinnasta sekä simulaation aikana otettujen steppien määrän seuraamisesta. (ML-Agents Toolkit Overview.)

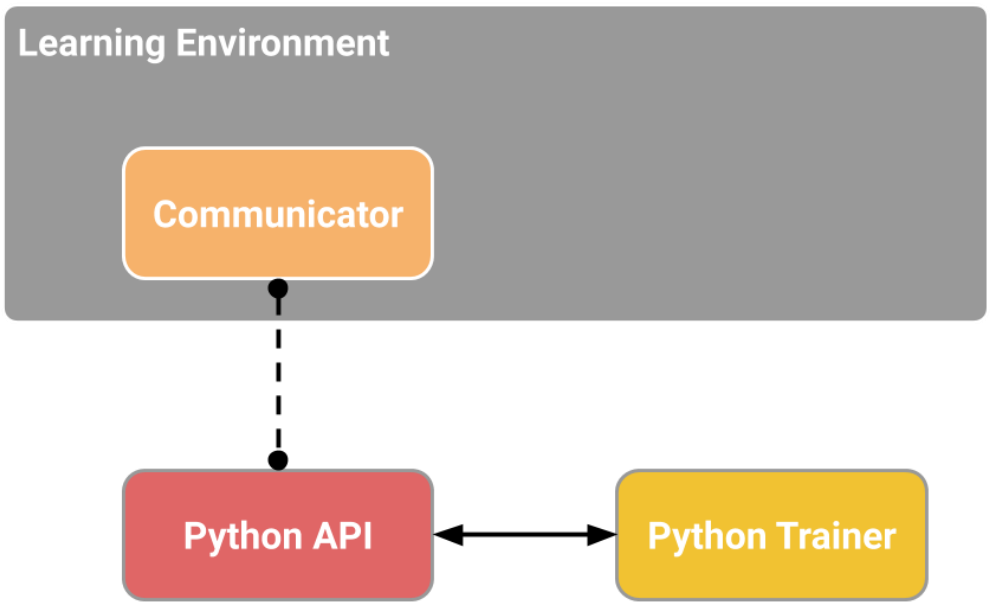
ML-Agents Toolkit sisältää neljä korkean tason komponenttia (KUVA 2).

1. **Oppimisympäristö** sisältää Unity Scenen ja agentit. ML-Agents Toolkittiin sisältyy ML-Agents Unity SDK, jonka avulla voidaan muuttaa mikä tahansa Unity Scene oppimisympäristöksi määrittelemällä agentit ja niiden käyttäytyminen. Scene tarjoaa ympäristön, jossa agentit havainnoivat, suorittavat toimintoja ja oppivat. Se miten scenestä luodaan oppimisympäristö, riippuu oppimistavoitteista. (ML-Agents Toolkit Overview.)

2. **Python Low-Level API** sisältää matalan tason Python-rajapinnan oppimisympäristön vuorovaikutukseen ja manipulointiin. Toisin kuin oppimisympäristö, Python API ei ole osa Unityä, vaan se kommunikoi Unityn kanssa ulkopuolelta käsin. Python API sisältyy omaan mlagents_envs Python-pakettiin, ja Python-harjoitteluprosessi käyttää sitä kommunikointiin akatemian kanssa ja sen ohjaamiseen koulutuksen aikana. API:n avulla voi myös käyttää Unitya simulointimoottorina omille koneoppimisalgoritmeille. (ML-Agents Toolkit Overview.)

3. **Ulkoinen kommunikaattori** yhdistää oppimisympäristön Pythonin matalan tason API:n kanssa. (ML-Agents Toolkit Overview.)

4. **Python Trainers** sisältää kaikki koneoppimisalgoritmit, jotka mahdollistavat agenttien kouluttamisen. Algoritmit on toteutettu Python-kielellä ja ne ovat osa omaa mlagents Python-pakettia. Python-kouluttajat käyttävät rajapintana ainoastaan Pythonin matalan tason API:a. (ML-Agents Toolkit Overview.)



KUVA 2. Yksinkertainen kaavio ML-Agents toolkitistä. (ML-Agents Toolkit Overview)

3.1.1 Agent-luokka

Agentti on Unityn Game Object, joka koostuu pääasiassa Agent-luokasta perityistä scripteistä ja agentille lisättävästä Behavior Parameters -komponentista. (KUVA 3.)

```
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;

public class SpaceAgent : Agent
{
    public override void Initialize(){ }
    public override void Heuristic(in ActionBuffers actionsOut) { }
    public override void OnEpisodeBegin() { }
    public override void CollectObservations(VectorSensor sensor) { }
    public override void OnActionReceived(ActionBuffers actions) { }
}
```

KUVA 3. Agent-luokan funktioita.

- **Initialize()**-metodia käytetään agentin valmisteluun, sitä kutsutaan vain kerran koulutuksen käynnistyessä ja se vastaa käytännössä Unityn MonoBeahavior luokasta periytyvää Awake()- ja Start()-funktioita.

- **Heuristic()** Kun Agentin käyttäytymisparametreissa Behavior Type on asetettu arvoksi Heuristic Only, Agentti käyttää Heuristic()-metodia Agentin toimintojen kutsumiseen, sen sijaan että koulutusalgoritmi päättäisi mitä toimintoja tehdään. Heuristic()-metodin avulla voidaan ohjata agenttia manuaalisesti, jolloin agentin toimintaa voidaan testata ennen koneoppimista. (Agents)

- **OnEpisodeBegin()** -metodia kutsutaan automaattisesti jokaisen oppimisepisodin alussa, myös ensimmäisellä kerralla kun koulutus käynnistyy. Tässä metodissa voidaan nollata oppimisympäristö takaisin lähtöasemiin. (Agents)

- **CollectObservations()** Kutsutaan jokaisen päätösaskeleen aikana. Tämä on yksi tapa kerätä havaintoja ympäristöstä. (Agents)

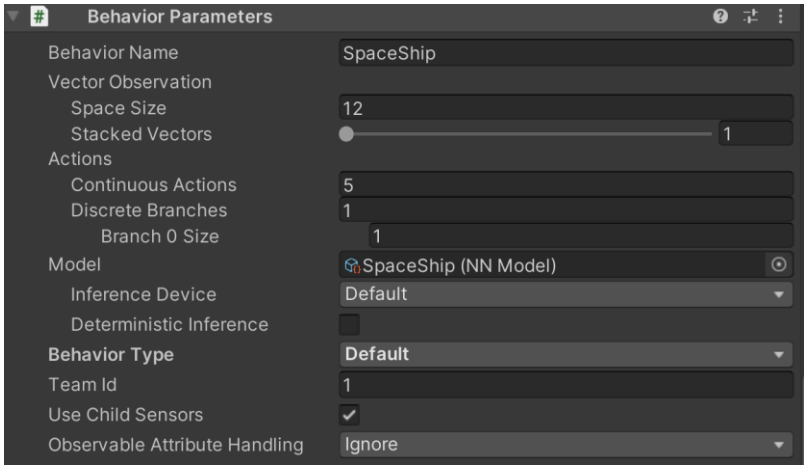
- **OnActionReceived()** -metodissa suoritetaan agentin toiminnat. Kutsutaan joka kerta kun agentti saa koneoppimismallilta toiminta käskyn. Tämä on myös yleinen metodi, jossa agentille annetaan palkintoja sen tekemien toimintojen perusteella. (Agents)

3.1.2 Episodit

Episodi on koulutuksessa tapahtuva agentin yksittäinen suoritus. Koulutus muodostuu useista episodeista, ja yleensä episodit ovat hyvin lyhyitä. Episodin aikana agentti tekee päätöksiä ja saa palkintoja. Jokaisen episodin jälkeen agentti nollataan. Episodi voidaan lopettaa EndEpisode-metodikutsulla. Episodi kannattaa yleensä lopettaa heti kun agentti on saavuttanut tavoitteen tai epäonnistunut siinä. Joskus episodit saattavat venyä liian pitkiksi ja selkeää tilannetta episodin manuaaliselle lopettamiselle ei synny, silloin episodit voidaan asettaa päättymään automaattisesti. Automaattinen episodin lopetus otetaan käyttöön agentin asetuksista, rajoittamalla max step -arvoa, joka määrittää episodin maksimikeston. Kun episodi loppuu, se tallennetaan ja aloitetaan uusi episodi. (Class Agent.)

3.1.3 Behavior Parameters

Behavior Parameters on komponentti, joka lisätään agentille (KUVA 4). Behavior Parameters sisältää parametrit, jotka määräävät, minkä menettelytavan agentti vastaanottaa.



KUVA 4. Behavior Parameters -komponentti.

- **Behavior Name**. Käyttäytymisen tunniste. Agentit, joilla on sama käyttäytymisen nimi, oppivat saman menettelytavan.

- **Space Size**. Agentin vektorihavainnon pituus.

- **Stacked Vectors**. Aiempien vektorihavaintojen määrä, jotka pinotaan ja joita käytetään päätöksenteossa.

- **Continuous Actions**. Samanaikaisten jatkuvien toimintojen määrä, joita agentti voi tehdä.

- **Model**. Neuroverkkomalli (saatu harjoittelun jälkeen).

- **Inference Device**. Käytetäänkö mallin suorittamiseen päätelmän aikana CPU:ta vai GPU:ta.

- **Behavior Type**. Määrittää, suorittaako agentti harjoittelun, inferencen vai käyttääkö se Heuristic()-metodia

- **Team ID**. Määrittää agentin joukkueen self-play-opetuksessa

- **Use Child Sensors**. Käytetäänkö kaikkia sensorikomponentteja, jotka on kiinnitetty agentin child GameObjecteihin

- **Max Step.** Agenttikohtainen askelten enimmäismäärä. Kun tämä määrä on saavutettu, agentti nollataan.

3.1.4 Havaintojen keräys

ML-Agents tarjoaa erilaisia tapoja kerätä havaintoja, joista yleisimmin käytetyt ovat Agent-luokan CollectObsevations()-metodi ja erilaiset sensorikomponentit, kuten RayPerceptionSensor, GridSensor ja CameraSensor. Tässä työssä käytettiin CollectObsevations()-metodia joka toimii parhaiten numeraalisten ja ei-visuaalisten havaintojen seuraamiseen. Metodilla on parametri, joka viittaa vektorisensoriin, jolle havainnot syötetään. Funktion toteutuksen on kutsuttava VectorSensor.AddObservation-metodia vektorihavaintojen lisäämiseksi. VectorSensor.AddObservation-metodilla on useita ylikuormituksia, joten sille voidaan syöttää erilaisia datatyypejä, kuten kokonaislukuja, totuusarvoja, vektoreita ja kvaternioita. Jotta koulutus tuottaa varmasti parhaan tuloksen on suositeltavaa, että havaintoarvot normalisoidaan ennen kuin ne syötetään sensorille. Havaintojen normalisointi tapahtuu skaalaamalla ne välille {0, 1} tai {-1, +1}. Vektorin normalisointi tapahtuu skaalaamalla jokainen vektorin arvo (x, y, z) erikseen, eli ei voida käyttää Vector3.Normalize()-metodia tai Vector3.normalized-ominaisuutta.

Havaintojen keräyksessä oleellista on mitä agentille halutaan opettaa ja mitä havaintoja agentti tarvitsee oppiakseen halutun asian. Jos agentti ei saa riittäviä ja oikeita tietoja, agentti saattaa oppia huonosti tai ei ollenkaan. Havaintoja miettiessä on hyvä miettiä mitä tietoja ihminen tarvitsisi ongelman ratkaisuun. Esimerkiksi jos agentti halutaan opettaa ajamaan autoa liikenteessä, oleellisia havaintoja olisivat agentin oma sijainti, nopeus ja renkaiden kulma. Lisäksi agentin pitäisi havainnoida ympäristöstä ainakin muiden autojen sijainti ja nopeus, sekä esteiden sijainti. Havaintoina syötettävät arvot on syytä testata ennen agentin kouluttamista, jotta ne varmasti ovat sitä mitä halutaan, sillä virheitä havainnoissa ei välttämättä huomaa ennen kuin koulutuksen myöhemmissä vaiheissa. (Agents.)

3.1.5 Toiminnot

Koneoppimismalli antaa ohjeistuksen toimista, jotka agentti toteuttaa. Toimintoja on olemassa kahdenlaisia, jatkuvia ”Continuous” ja erillisiä ”Discrate”. Jatkuvat toiminnot ovat liukuarvoja ja ne sopivat tilanteisiin, joissa tarvitaan tarkempaa kontrollia, kuten eri nopeuksilla liikkuminen ja kääntyminen. Erilliset toiminnot ovat kokonaislukuja sisältävä taulukko ja niitä käytetään yleensä vähän kuin totuusarvoja, eli jos on ennalta tiedettyjä vaihtoehtoja, eikä tarvita välimuotoja. Esimerkiksi jos agentin pitää hypätä tai ampua, voidaan käyttää discrate-toimintoa, kun taas auton nopeuden muuttaminen käyttäisi continuos-toimintoa. Toiminnot välitetään agentille parametrina, kun akatemia kutsuu agentin OnActionReceived()-funktiota. OnActionReceived()-funktiossa määritellään kaikki toiminnot joita agentin on mahdollista suorittaa. Koulutusalgoritmi ei tiedä mitä toimintoja agentilla on, tai mitä eri arvot tarkoittavat. Koulutusalgoritmi kokeilee erilaisia toimintaluettelon arvoja ja vertailee niiden tuottamien palkintojen arvoa koulutuksen aikana. Jos esimerkiksi agentti halutaan liikkumaan 2d-ulottuvuudessa, voidaan käyttää joko jatkuvia tai erillisiä toimintoja. Jatkuvaa toimintoa käytettäessä agentin toimintojen koko on kaksi, yksi liukuluku kummallekin akselille (x, y). Erillistä toimintoa käytettäessä olisi yksi haara, jonka koko olisi 4 (0, 1, 2, 3), yksi jokaiselle suunnalle. (Agents.)

Kun jatkuvaa toimintoa käytettäessä PPO-koulutusalgoritmi välittää agentille liukuluvun se oletuksena skaalaa luvun välille [-1, 1], mutta on hyvä käytäntö skaalata arvot manuaalisesti myös koodissa, sillä jotkut kolmannen osapuolen algoritmit eivät sitä välttämättä automaattisesti tee (KUVA 5). (Agents.)

```
controlSignal.x = Mathf.Clamp(actions.ContinuousActions[0], -1f, 1f);
controlSignal.y = Mathf.Clamp(actions.ContinuousActions[1], -1f, 1f);
```

KUVA 5. Arvojen skaalaus.

Toiminnot on hyvä testata ennen koulutusta, käyttämällä Agentin Heuristic()-metodia, jonka avulla toimintoja voidaan kutsua esimerkiksi näppäimistöä tai peliohjainta käyttämällä. Heuristic-metodin avulla agentilla voidaan pelata ja kokeilla että kaikki toiminnot tekevät varmasti mitä niiden kuuluu tehdä ja agentti saa oikeat palkinnot toiminnoista. Vahvistusoppimisessa koulutusalgoritmi lähettää aluksi satunnaisesti arvoja OnActionReceived()-funktiolle. Koulutuksen aikana agentti saa kuitenkin palkintoja toimintojen ja tulosten perusteella, jolloin se pyrkii lähettämään arvoja, jotka tuottavat ajan kuluessa parhaan palkkion. (Agents.)

3.1.6 Palkinnot

Vahvistusoppimisessa palkinnot ovat signaaleja, joita annetaan agentille sen toimintojen tai tulosten perusteella, niiden avulla agentti tietää onko se tehnyt jotain oikein tai väärin. Palkintosignaalit voivat olla positiivisia ja negatiivisia, riippuen siitä halutaanko agenttia palkita vai rankaista. Palkkiot kannustavat agenttia tekemään toimintoja, joista se saa eniten palkintosignaaleja. Yleensä agenttia kannattaa palkinta ennemmin tulosten kuin toimintojen perusteella. Vahvistumisalgoritmi optimoi agentin toiminnot siten että se saisi ajan mittaan suurimman mahdollisen kumulatiivisen palkkion. Palkintoja annetaan AddReward- ja SetReward-funktioilla jokaisen agentin tekemän päätöksen välissä, jonka jälkeen palkinnot tallennetaan ja nollataan. Jos AddReward() käytetään useasti yhden päätöksen aikana, annetut palkkiot lasketaan yhteen sen arvioimiseksi, kuinka hyvä edellinen päätös oli. SetReward() kumoaa kaikki agentille edellisen päätöksen jälkeen annetut palkkiot ja asettaa uuden palkkion. Jokaisen erillisen palkintosignaalin tulisi olla väliltä [-1, 1], sitä suuremmat tai pienemmät arvot voivat johtaa epävakaaseen koulutukseen. (Agents.)

3.1.7 Imitaatio-oppiminen

Imitaatio-oppimista käytettäessä agentti saa automaattisesti palkintoja, sen mukaan miten hyvin se imitoi mallisuorituksen mukaista käytöstä. (Nigretti 2018.)

3.1.8 Self Play

Self-play on koulutus, jossa kaksi tai useampi agentti tekee toimia samanaikaisesti, mutta kumpikin maksimoi oman palkkionsa. Self-play käyttää agentin nykyistä ja aiempaa versiota itsestään vastustajana. Tämä tarjoaa luonnollisesti kehittyvän vastustajan, jota vastaan agentti voi vähitellen kehittyä

perinteisten vahvistusoppimis-algoritmien avulla. Pelaamalla aina vain vahvempaa versiota itsestään vastaan, agentti löytää uusia ja parempia strategioita. Täysin koulutettua agenttia voidaan käyttää kilpailijana edistyneille ihmispelaajille. Samaa tapaa on käytetty monien tunnettujenkin tekoälyjen opettamisessa, kuten AlphaGo ja OpenAI Five. (Cohen 2020.)

3.1.9 Kurssioppiminen

Agentin koulutuksessa voidaan käyttää kurssioppimista. Kurssioppimisessa koulutus tapahtuu asteittain, siten että haastetta kasvatetaan vähitellen niin, että agentti on aina optimaalisesti haastettu. Kurssioppimista voidaan verrata tapaan, miten ihmisetkin tyypillisesti oppivat. Lapsille opetetaan koulussa ensin numerot, sitten helppoja laskutoimituksia ja myöhemmin siirrytään vaikeampiin tehtäviin. Aikaisemmin opitut taidot muodostavat pohjan tuleville oppitunneille. Koneoppimisessa voidaan käyttää samaa tekniikka. Esimerkkinä voidaan ajatella, vaikka itseohjautuvan auton opettamista, ensin voidaan opettaa auto liikkumaan yhdestä pisteestä toiseen. Kun auto osaa liikkua voidaan nostaa haastetta ja asettaa reitille esteitä. (Introducing ML-Agents Toolkit v0.2.)

3.2 Hyperparametrit

Kaikki oppimisalgoritmien hyperparametriasetukset määritetään koulutukseen YAML-tiedostossa. Tiedosto sisältää kaikkien ympäristön agenttien käyttäytymiskonfiguraatiot. Konfiguraatiotiedosto ei ole osa itse ympäristöä, ja se on välitettävä python-kouluttajille koulutuksen alkaessa. Konfiguraatiot sisältävät myös sen, mitä algoritmeja koulutuksessa käytetään. (Training Configuration File.)

- **trainer_type** (oletus: ppo) Kouluttajan tyyppi, jota käytetään: ppo, sac tai poca.

- **summary_freq** (oletus: 50000) Kokemusten määrä, joka kerätään ennen koulutusstatistiikan generointia ja näyttämistä. Määrittelee diagrammin tarkkuuden Tensorboardissa.

- **time_horizon** (oletus: 64) Askelten määrä, joka kerätään ennen kuin ne lisätään kokemuspuskuriin. Jos askelten määrä tulee täyteen ennen koulutusjakson loppua, yritetään agentin nykyisestä tilasta ennustaa palautteen määrä. Tilanteissa, joissa palkintoja annetaan usein tai koulutusjakso on kohtuuttoman pitkä, pienempi arvo voi olla parempi. Jos agentti ei pääse

tavoitteeseen riittävässä ajassa, koulutusjakso alkaa alusta. Tällä voidaan välttää tilanne, jossa agentti on jäänyt looppiin, eikä koulutus etene. Arvon pitäisi olla niin suuri, että kaikki tärkeimmät agentin toiminnot saadaan tallennettua. Tyypillinen arvo on välillä 32–2 048

- **max_steps** (oletus: 500 000) Askelten kokonaismäärä, joka pitää saavuttaa ympäristössä ennen kuin koulutus lopetetaan. Jos ympäristössä on useita agentteja, joilla on sama käyttäytymisnimi, niiden suorittamat askeleet lasketaan yhteen. Tyypillinen arvo on välillä 5e5 - 1e7

- **learning_rate** (oletus: 3e-4) Oppimisnopeus gradienttimenetelmälle. Tätä arvoa tulisi laskea, jos koulutus on epävakaata ja palkkio ei nouse johdonmukaisesti. Tyypillinen arvo on välillä 1e-5 - 1e-3

- **batch_size** Kokemusten lukumäärä jokaisella gradienttimenetelmän iteraatiolla. Arvon tulisi olla aina useita kertoja pienempi kuin buffer_size-hyperparametri. Jos agentti käyttää jatkuvaa toimintatilaa, arvon tulisi olla tuhansissa, erillisessä toimintatilassa kymmenissä. Tyypillinen arvo on välillä: (Jatkuva - PPO): 512–5 120; (Jatkuva - SAC): 128–1 024; (Erillinen, PPO ja SAC): 32–512

- **buffer_size** (oletus: 10 240 jos PPO ja 50 000 jos SAC)
  PPO: Kokemusten lukumäärä, joka kerätään ennen menettelytavan päivittämistä. Vastaa siitä kuinka paljon kokemuksia pitäisi kerätä ennen kuin oppiminen tapahtuu ja oppimismallin käytäntöä päivitetään. Arvon tulisi olla useita kertoja suurempi kuin batch_size-hyperparametri. Yleensä suurempi arvo tarkoittaa vakaampaa oppimista. Tyypillinen arvo on välillä: PPO: 2 048–409 600; SAC: 50 000–1 000 000

- **beta** (oletus = 5.0e-3) Entropian vahvuus. Tekee oppimismallista satunnaisemman, eli agentti tekee toimintoja laajemmalla skaalalla. Suurempi arvo tarkoittaa, että on useampia satunnaisia toimintoja. Arvo tulisi asettaa sellaiseksi, että entropia laskee hitaasti, samalla kun palkkio nousee. Tämä on mitattavissa TensorBoardissa. Tyypillinen arvo on välillä 1e-4 - 1e-2

- **epsilon** (oletus = 0.2) Vaikuttaa siihen, miten nopeasti oppimismalli/menettelytapa(”policy”) voi kehittyä koulutuksen aikana. Tarkoittaa hyväksyttyä eroavaisuuden raja-arvoa vanhan ja uudeen menettelytavan välillä gradienttimenetelmän päivityksen yhteydessä. Pienempi arvo

tuottaa vakaampia päivityksiä, mutta hidastaa oppimisprosessia.
Tyypillinen arvo on välillä 0.1–0.3

- **lambd** (oletus = 0.95) Kertoo, kuinka paljon agentti luottaa tulevaisuuden palkintoarvioon. Pienempi arvo tarkoittaa, että agentti luottaa enemmän nykyiseen arvioon, kun taas suurempi arvo tarkoittaa, että agentti luottaa enemmän peliympäristöstä saatuun palautteeseen. Oikein säädetty arvo johtaa vakaampaan oppimisprosessiin. Tyypillinen arvo on välillä 0.9–0.95

- **num_epoch** (oletus = 3) Kuinka monesti kokemuspuskuri käydään läpi, kun gradienttia optimoidaan. Mitä isompi batch_size-arvo on, sitä isommaksi tämä voidaan asettaa. Pienempi arvo johtaa vakaampiin päivityksiin, mutta hidastaa oppimista. Tyypillinen arvo on välillä 3–10

- **num_layers** (oletus = 2) Piilotettujen kerrosten määrä neuroverkossa. Yksinkertaisissa ongelmissa, pienempi määrä kerroksia johtaa todennäköisemmin nopeampaan ja tehokkaampaan oppimiseen. Monimutkaisissa ongelmissa voi tarvita useampaa kerrosta. Tyypillinen arvo on välillä 1–3

- **hidden_units** (oletus = 128) Neuronien määrä kerroksissa. Yksinkertaisissa ongelmissa arvon kannattaa olla pieni. Tyypillinen arvo on välillä 32–512

3.3 TensorFlow

TensorFlow on Googlen kehittämä ja alun perin vuonna 2015 yleisölle julkaistu avoimen lähdekoodin kirjasto numeeriseen laskentaan ja laajamittaiseen koneoppimiseen. TensorFlow niputtaa yhteen joukon koneoppimisen ja syväoppimisen malleja ja algoritmeja (eli neuroverkkoja). TensorFlow käyttää Pythonia tai JavaScriptiä tarjotakseen käyttöliittymän sovellusten tekemiseen. TensorFlow avulla Unity pystyy luomaan neuroverkon ja tallentamaan siitä TensorFlow-mallin, joka on .nn-tiedosto. Malli tallennetaan Unityyn ja sitä voidaan sen jälkeen käyttää tekoälynä pelissä. (Yegulalp 2020.)

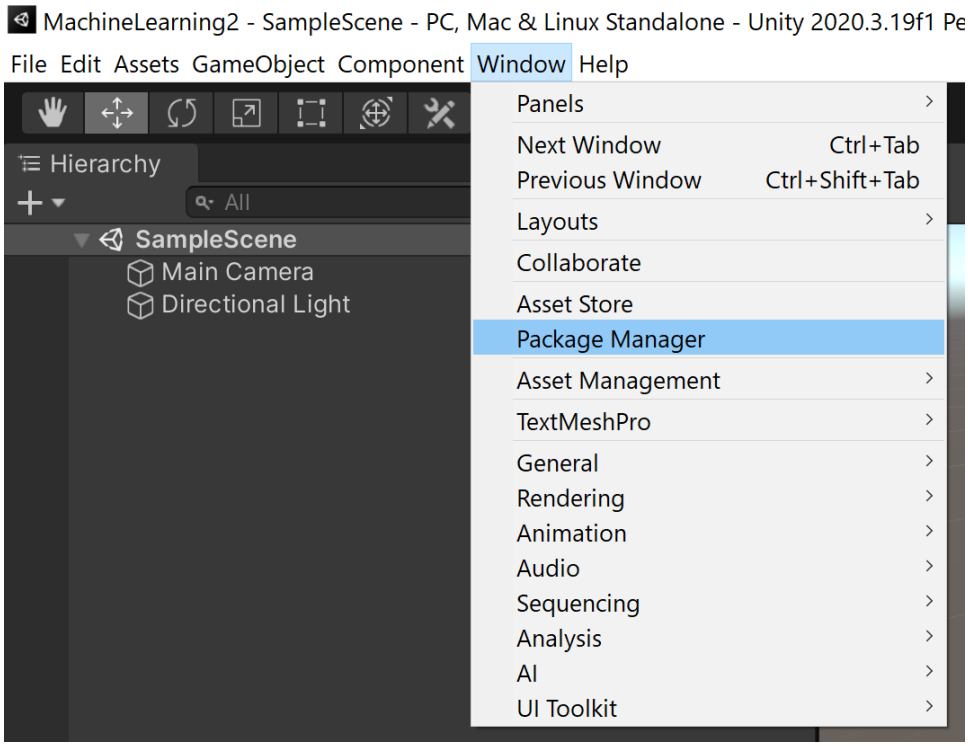
3.4 TensorBoard

Yksi osa mallien kouluttamista TensorFlow'lla on hyperparametrien arvojen asettaminen. Näiden hyperparametrien oikeiden arvojen löytäminen voi vaatia useita iteraatioita. Tätä varten on olemassa TensorFlow'n visualisointityökalu nimeltä TensorBoard. Se mahdollistaa tiettyjen attribuuttien kuten palkintojen visualisoinnin koulutuksen aikana, mikä on hyödyllistä optimaalisten arvojen asettamisessa Unity-ympäristöön. (Background: PyTorch.)

4 ML-AGENTSIN KÄYTTÖÖNOTTO JA ASENNUS

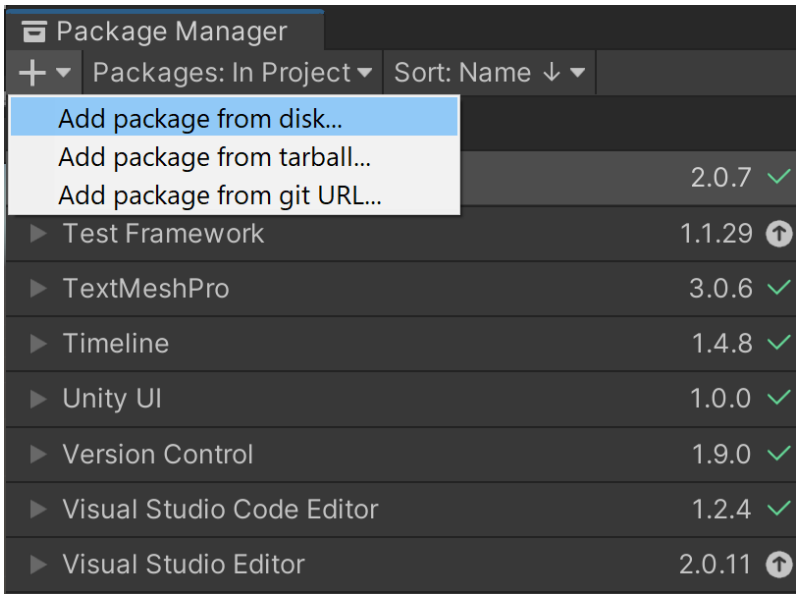
ML-Agentsin asennus on yksinkertaista. Unityltä löytyy GitHub repository, jossa on Unity projekti, joka sisältää ML-Agents toolkitin, sekä dokumentaatiot ML-Agentsin käyttöön. Jotta projekti saadaan käyntiin, ensimmäinen asia on ladata GitHubista ML-Agents Toolkit.

Kun ML-Agents Toolkit on ladattu, se liitetään projektiin package managerin kautta (KUVA 6).



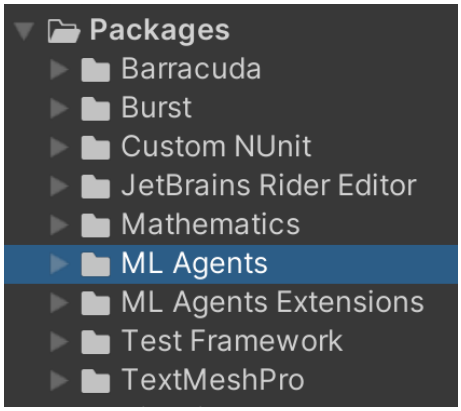
KUVA 6. Toolkit liitetään projektiin.

Valitaan “Add package from disk...” Etsitään package.json-tiedosto joka löytyy GitHubista ladatun toolkitin kansiosta com.unity.ml-agents (KUVA 7).



KUVA 7. Paketin lisäys.

Projektista löytyy nyt ML-Agents paketti(KUVA 8).

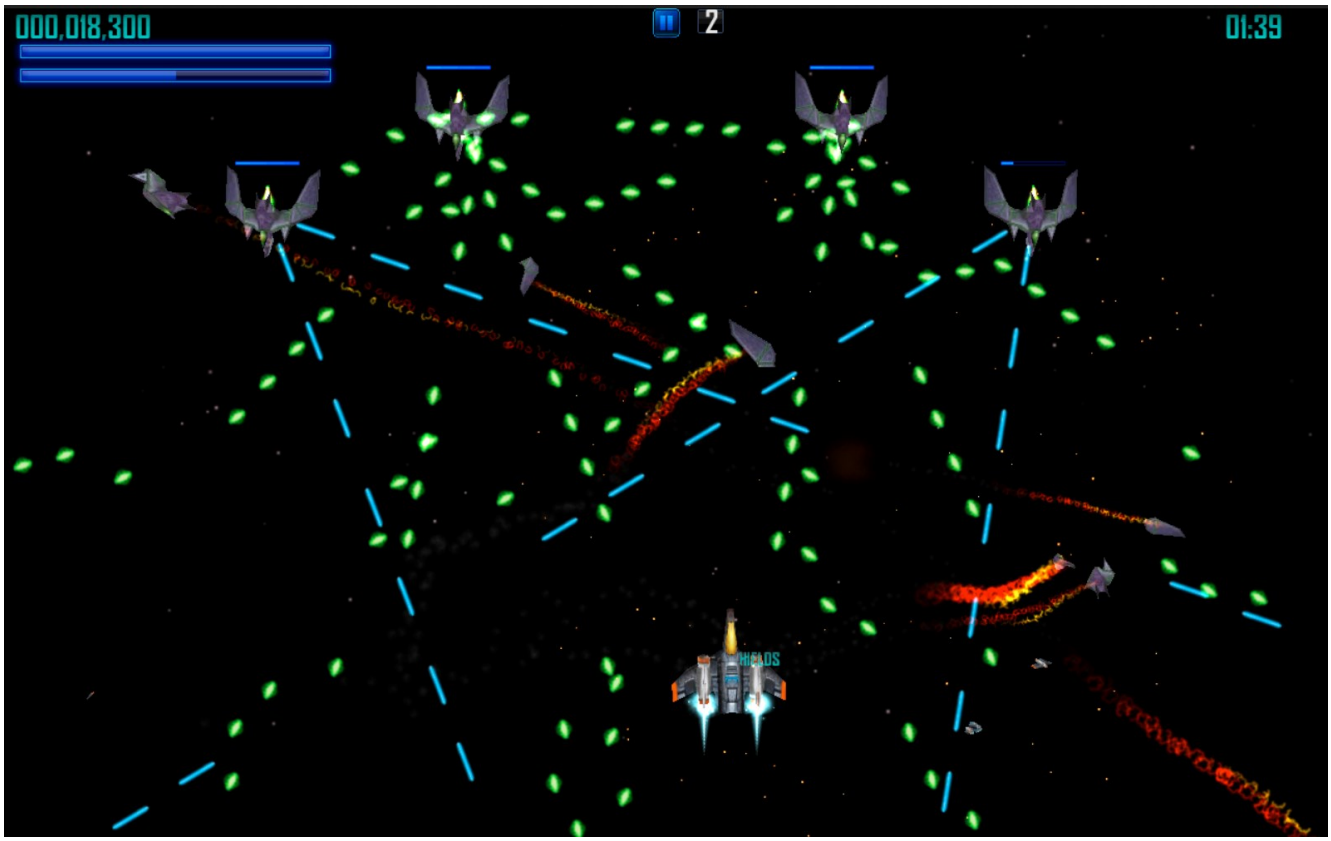


KUVA 8. Pakettikansio.

Lisäksi asennetaan Python ja tarvittavat liitännäiset.

5 TESTIYMPÄRISTÖN TOTEUTUS

Työ alkoi tarvittavien työkalujen asentamisella. Kun työkalut oli asennettu, niiden toimivuutta testattiin mukana tulleiden valmiiden pelidemojen avulla. Kun kaikki toimi, oman pelidemon suunnittelu alkoi. Tarkoituksena oli kehittää yksinkertainen pelidemo, joka olisi haastava tekoälylle. Testiympäristöksi valittiin bullet hell -tyyppinen 2d-avaruusräiskintäpeli. Bullet hell -pelit ovat tyyliltään pelejä, joissa pelaaja usein väistelee nimen mukaisesti kymmeniä tai jopa satoja ammuksia (KUVA 9). Bullet hell -pelit vaativat pelaajalta nopeaa reagointia ja pelikentän tarkkaa seuraamista. (Wikipedia.) Pelin vaikeustaso kasvaa sitä mukaa mitä enemmän väisteltävää ruudulla on. Tästä lähtökohdasta ajatellen peli tarjoaa tekoälylle paljon opittavaa ja vaikeustasoa on helppo skaalata vaikeammaksi sitä mukaa kun tekoäly oppii. Pelin ulkoasulla ei tekoälyn kannalta ole merkitystä, joten pelissä voitiin käyttää yksinkertaista 2d-grafiikkaa. Ulkoasun tuli olla kuitenkin selkeän näköistä, jotta tekoälyn toimintaa olisi helppo seurata.



KUVA 9. Esimerkki bullet hell peligenrestä. (indiedb).

5.1 Pelimekaniikka

Pelimekaniikka pidettiin yksinkertaisena, jotta tarpeetonta koodausta olisi mahdollisimman vähän ja pystyttiin keskittymään suurelta osin koneoppimisen kehittämiseen. Pelissä tarkoituksena oli tuhota vastustajan alus ennen kuin vastustaja tuhoaa pelaajan. Peliä pystyy ohjaamaan hiiren ja näppäimistön avulla tai vaihtoehtoisesti peliohjaimella. Avaruusalusta voidaan ohjata vapaasti mihin tahansa ilmansuuntaan 2d-vektorilla. Avaruusaluksen suuntaa voidaan myös kääntää 360 asteen säteellä. Ampuminen toimii nappia painamalla ja tähtäys tapahtuu avaruusaluksen suuntaa kääntämällä. Ampumisnopeus oli rajoitettu, mutta kuitenkin niin, että ammuksia pystyi ampumaan useita sekunnissa. Voittava alus ratkaistiin tuhoamalla toinen alus. Tuhoaminen vaatii tietyn määrän osumia vastustaja-alukseen. Tarvittavien osumien määrää vaihteli, opetustilanteen mukaan. Kun voittaja on selvillä, peli alkaa alusta ja pelaajien sijainnit ja osumapisteet nollataan. Yksittäisen pelin kesto pidettiin mahdollisimman nopeana koneoppimista varten, jotta opetustoistoja saatiin mahdollisimman paljon.

5.2 Ulkoasu ja opetusympäristö

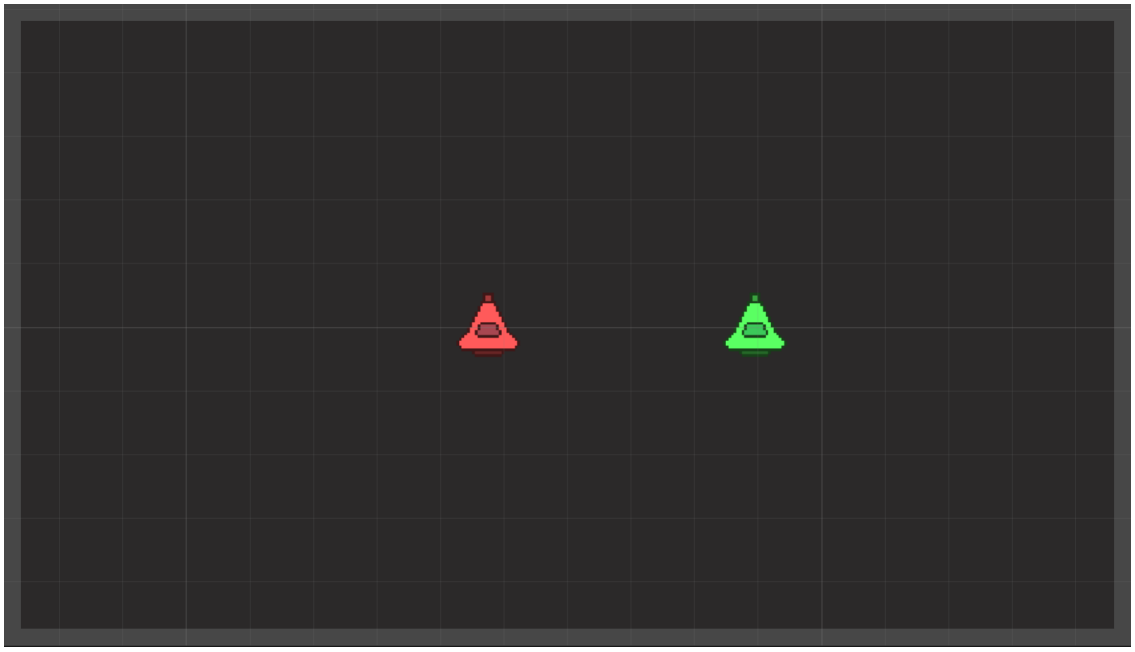
Peliä varten piirrettiin yksinkertaiset 2d-spritet avaruusaluksista, jotka toimivat koneoppimisagentteina. Lisäksi piirrettiin vielä ammukset, joita agentit ampuvat. (KUVA 10.) Muuta grafiikka olivat numerot ruudun ylälaidassa, jotka kertoivat pelaajien pisteet, sekä pelaajien ottamat osumat.

KUVA 10. Alukset ja ammukset.

Pelaajilla oli eri väriset alukset ja väreihin vastaavat ammukset. Värejä pystyi vaihtamaan lennosta, joten uusia pelaajia pystyi helposti lisäämään, ja ne erottuivat toisistaan värien avulla. Tämä auttoi pelin

skaalautuvuudessa, kun pelaaja määriä ja erilaisia opetustilanteita pystyi nopeasti vaihtamaan halutunlaiseksi.

Pelialueena toimi yksivärinen ruudullinen kenttä, joka täytti koko kuvaruudun. Reunoilla on ulkoseinät, jotka toimivat kentän rajoina ja estävät aluksia menemästä kentän ulkopuolelle. Ulkoseinät oli toteutettu niin että ne luodaan pelin käynnistyessä aina kameran kuva-alueen reunoille, jolloin pelialueen kokoa pystyttiin muuttamaan helposti ja nopeasti. (KUVA 11.)



KUVA 11. Pelialue.

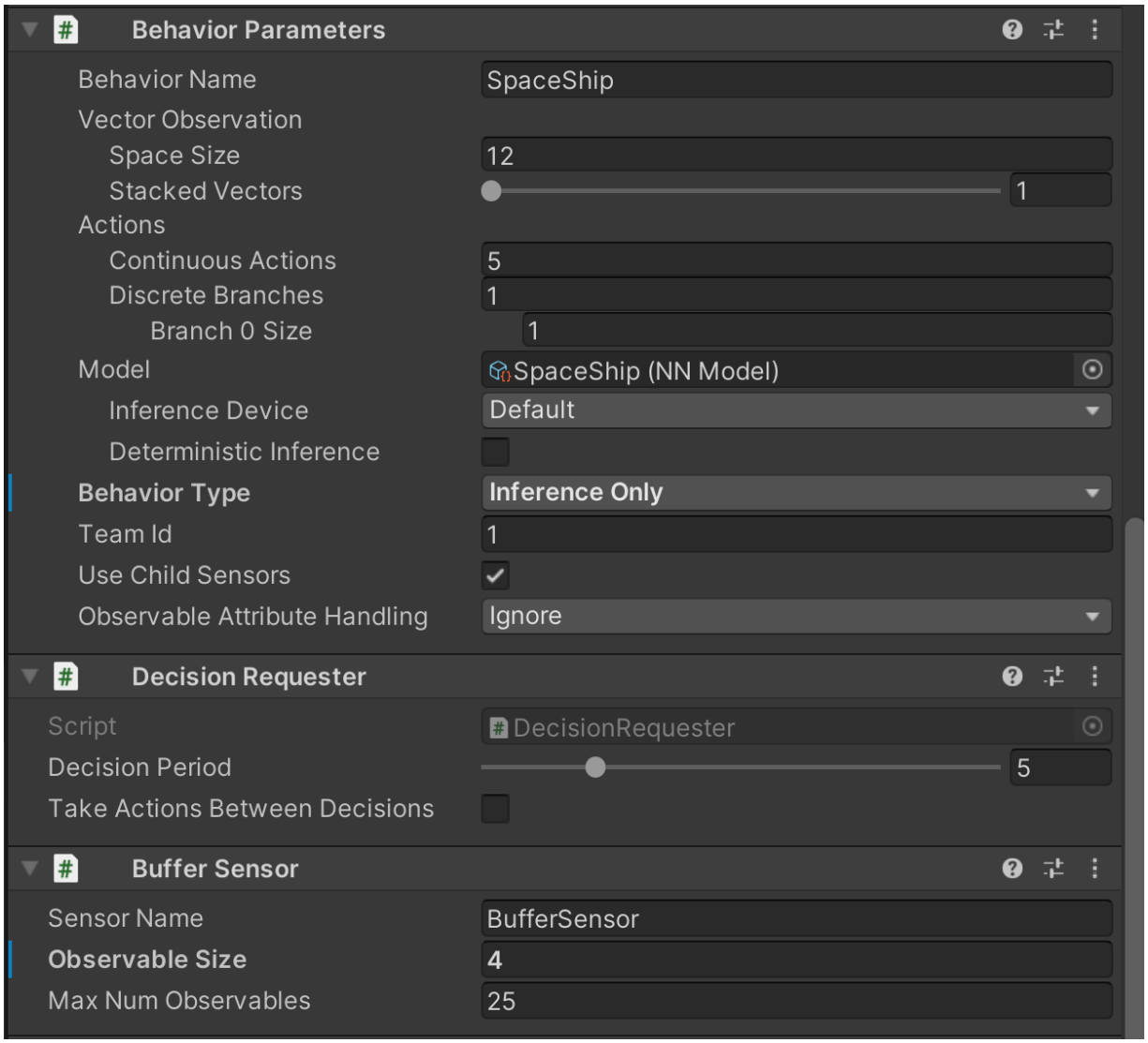
Oppimisen tehostamista varten peliympäristö kopioitiin useita kertoja. Kopioimalla peli, saadaan useita yhtäaikaisia pelejä toimimaan rinnakkain, jolloin myös koulutuksen toistot ja oppimisnopeus saadaan moninkertaistettua. (KUVA 12.)

KUVA 12. Pelialue kopioituna.

5.3 Agentti

Agentti on tekoälyn ohjaama pelaaja. Agentille on määritelty kolme toimintoa: liikkuminen x ja y akselilla, kääntyminen ja ampuminen. Aluksen liikkuminen on toteutettu 2D-Rigidbodylla, eli se toimii pelimoottorin omia fysiikoita käyttäen. Alusta ohjataan rigidbodyn Addforce() funktiolla joka kiihdyttää alusta annettujen x- ja y-arvojen mukaan. Addforce()-funktiolla kiihtyminen tapahtuu realistisesti fysiikoiden mukaan, esimerkiksi kun kiihtyminen loppuu aluksella kestää hetken aikaa että se pysähtyy. Fysiikoiden käyttäminen tuo tekoälylle lisähaasteen aluksen ohjauksessa. Kääntyminen tapahtuu myös x- ja y-arvoja muuttamalla. Ampuessaan alus luo ammuksen ja ammus on määritelty liikkumaan aina ammuksen oman x-akselin suuntaisesti.

Agent-peliobjekti käyttää Sprite Renderer-, Polygon Collider- ja Rigidbody 2D -komponentteja, joiden avulla pelimoottori renderöi aluksen ja hoitaa aluksen fysiikat. Näiden avulla saadaan pelihahmolle normaalit tarvittavat toiminnot. Lisäksi agentille tulee koneoppimiseen tarvittavat Behavior Parameters-, Decision Requester- ja Buffer Sensor -komponentit, sekä scripti joka on peritty Agent-luokasta ja sisältää Agent-luokan metodit. (KUVA 13.)



KUVA 13. Komponentteja.

5.3.1 Havainnot

Agentille määriteltiin kaikki havainnot, joita se tarvitsi toimintojen toteuttamiseen. Agentin täytyi pystyä seuraamaan omaa sijaintia, nopeutta ja rotaatiota. Lisäksi agentti seuraa vastustajan sijaintia ja nopeutta. Näillä tiedoilla agentti pystyy vertaamaan omaa sijaintiaan vastustajan sijaintiin ja tähtäämään vastustajaa kohti. Agentti myös etsii kaikki pelikentällä olevat vastustajan ammukset ja seuraa niiden sijaintia ja liikesuuntaa.

Havainnot on syytä suunnitella ja testata huolellisesti koska niistä johtuvia ongelmia ei välttämättä huomaa ennen kuin koulutus on edennyt riittävän pitkälle. Alkuun agentille ei ollut annettu tietoa

vastustajan nopeudesta ja meni pitkään, että ongelma tuli ilmi. Vaikka aluksi näytti siltä, että opetus sujui hyvin ja agentti oppi ampumaan vastustajaa kohti, myöhemmin ilmeni ongelmia agentin tähtäyksessä. Ongelma johtui ohjelmointivirheestä, jossa vastustajan nopeutta ei havainnoitu.

Havaintojen keräys tapahtuu Agent-luokan CollectObservations-metodissa (KUVA 14). Oppimisalgoritmi saa agentin, sekä ammusten sijainnin, nopeuden, suunnan vector2-datana.

```
public override void CollectObservations(VectorSensor sensor)
{
    sensor.AddObservation(Target.transform.localPosition);
    sensor.AddObservation(this.transform.localPosition);
    sensor.AddObservation(this.transform.rotation);
    sensor.AddObservation(rb.velocity.x);
    sensor.AddObservation(rb.velocity.y);

    var bullets = transform.parent.GetComponentsInChildren<Bullet>();
    int numBulletAdded = 0;
    foreach (Bullet b in bullets)
    {
        if (numBulletAdded >= 25)
        {
            break;
        }
        if (b.playerID == playerID)
        {
            continue;
        }

        float[] bulletObservation = new float[]{
            (b.transform.position.x - transform.position.x) / 10f,
            (b.transform.position.y - transform.position.y) / 10f,
            b.transform.up.x,
        };

        numBulletAdded += 1;
        m_BufferSensor.AppendObservation(bulletObservation);
    };
}
```

KUVA 14. CollectObservations-metodi.

5.3.2 Päätökset

Päätökset, joita agentti pystyy tekemään ovat nopeuden muuttaminen x- ja y-akselilla, rotaatio oman akselinsa ympäri sekä ampuminen.

Agentit vastaanottavat oppimismallin tekemät päätökset OnActionReceived-metodin välityksellä (KUVA 15). Oppimismalli antaa agentille x- ja y-signaalit, jotka on rajattu liukuarvoiksi -1 ja 1 välille. Tämän jälkeen nopeus kerrotaan halutulla arvolla, jolloin saadaan alukselle haluttu huippunopeus. Aluksen kääntymisen laskemiseen käytetään pelimoottorin tarjoamaa matematiikkakirjastoa.

```
public override void OnActionReceived(ActionBuffers actions)
{
    controlSignal = Vector2.zero;
    controlSignal.x = Mathf.Clamp(actions.ContinuousActions[0], -1f, 1f);
    controlSignal.y = Mathf.Clamp(actions.ContinuousActions[1], -1f, 1f);
    controlSignal2.x = Mathf.Clamp(actions.ContinuousActions[2], -1f, 1f);
    controlSignal2.y = Mathf.Clamp(actions.ContinuousActions[3], -1f, 1f);
    rb.AddForce(controlSignal * forceMultiplier);

    float angle = Mathf.Atan2(controlSignal2.x, -controlSignal2.y) * Mathf.Rad2Deg;
    transform.rotation = Quaternion.Euler(new Vector3(0, 0, angle));
```

KUVA 15. OnActionReceived.metodi.

5.3.3 Palkinnot

Tavoitteet agentille olivat, että se liikkuisi aktiivisesti, välttäisi kentän ulkoreunoja, ampuisi tarkasti vastustajaa ja väistäisi vastustajan ammuksia. Palkitseminen toteutettiin siten että agentti saa positiivisia palkintoja aina kun sen ampuma ammus osuu toiseen agenttiin. Jotta agentti ymmärtäisi, että sen täytyy tähdätä ja ampua vastustajaa kohti, sen täytyy saada riittävästi palkintoja osumista. Aluksi agentti ei kuitenkaan tiedä mitä sen pitäisi tehdä, joten sille annettiin koulutuksen alussa pienempiä palkintoja pelkästä ampumisesta. Kun agenttia palkitaan ampumisesta, se alkaa ampua useammin ja mahdollisuudet vahingossa vastustajaan osumisesta kasvavat, jolloin agentti saa osumispalkintoja. Kun agentti on oppinut tähtäämään riittävän hyvin, pelkästä ampumisesta saatavat palkinnot poistetaan. Agentille annettiin pieniä palkintoja myös silloin kun sen nopeus oli riittävän suuri, tämän

tarkoituksena oli saada agentti liikkumaan nopeammin, jotta se oppisi paremmin väistelemään ja jotta pelissä olisi nopeampi tempo. (KUVA 16.)

```
AddReward(-0.0002f);

if ((transform.localPosition - Target.transform.localPosition).magnitude < 3)
{
    AddReward(-0.002f);
}

bulletTimer += 1;
if (bulletTimer > 15)
{
    if (actions.DiscreteActions[0] == 1)
    {

        Fire();
        AddReward(0.001f);<
        bulletTimer = 0;
    }
}

if (rb.velocity.magnitude > 0.6f)
{
    AddReward(0.001f);
}
```

KUVA 16. Palkitseminen.

Positiivisten palkintojen lisäksi agentille annettiin myös negatiivisia palkintoja. Agenttia palkittiin negatiivisesti, jos siihen osui toisen agentin ampuma ammus, tai jos agentti törmäsi pelialueen reunaan. Lisäksi agentti sai jatkuvaa pientä negatiivista palkintoa riippumatta mitä pelissä tapahtui, tämän tarkoituksena oli saada agentti tekemään aktiivisesti päätöksiä ja saattamaan peli loppuun mahdollisimman nopeasti. Ilman tällaista negatiivista palkitsemista agentti saattaa oppia toimimaan passiivisesti. Esimerkiksi koulutuksen alussa, kun agentti kokeilee liikkumista ja törmää toistuvasti seiniin, sitä palkitaan negatiivisesti, tällöin agentti saattaa oppia, että paikallaan pysyminen on sille parempi vaihtoehto.

Suurimman palkinnon agentti saa, kun se voittaa vastustajan. Häviöstä agentti saa ison negatiivisen palkinnon.

5.4 Koulutusprosessi

Koulutus toteutettiin vahvistusoppimista ja Self-play-opetusta käyttäen. Self-play-opetus toteutetaan niin että agentti harjoittelee pelaamaan itseään vastaan. Koulutuksessa käytettiin kuvassa 17 nähtäviä hyperparametreja.

```
behaviors:
  SpaceShip:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0002
      beta: 0.003
      epsilon: 0.15
      lambd: 0.93
      num_epoch: 4
      learning_rate_schedule: constant
    network_settings:
      normalize: true
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.96
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 80000000
    time_horizon: 1000
    summary_freq: 20000
    self_play:
      window: 10
      play_against_latest_model_ratio: 0.5
      save_steps: 20000
      swap_steps: 10000
      team_change: 100000
```

KUVA 17. Käytetyt hyperparametrit.

5.5 Ongelmat

Tämän insinöörityön yhdeksi suurimmiksi ongelmiksi muodostui koulutuksen pitkä kesto, sekä toimivien hyperparametrien ja palkintojen painoarvojen löytäminen. Agenttien koulutuksessa saattoi kestää useita tunteja tai jopa päiviä ennen kuin saatiin tuloksia ja pystyttiin todentamaan hyperparametrien toimivuus. Usein koulutuksen jälkeen ilmeni monia ongelmia, joita ei osannut ennalta odottaa. Ongelmien korjaaminen vaati myös yleensä koulutuksen uudelleen aloittamista. Ongelmaksi koettiin myös tilanne, kun pelin toiminnallisuuksia haluttiin muuttaa kesken pelikehityksen, jouduttiin myös tekoäly opettamaan uudelleen muuttuneeseen tilanteeseen.

Koneoppimisalgoritmin hyperparametrien ja agenttien palkkioiden painoarvojen säätäminen koettiin haastavaksi. Jokainen tekoäly on yksilöllinen ja vaatii omanlaiset hyperparametrit, eikä valmiita hyperparametriasetuksia ole saatavilla, jolloin kaikki parametrit täytyy säätää sopiviksi käytännössä omien kokeilujen pohjalta.

Työtä varten toteutettu peli käytti Unity-pelimoottorin omaa fysiikkamoottoria, joka hidasti koulutusprosessia. Parhaassa tapauksessa koulutus tapahtuu pelimoottorissa nopeutetulla ajankululla, mutta fysiikkaan pohjautuvassa pelissä liian suuri ajankulun nopeuttaminen johtaa epätarkkaan fysiikan laskentaan, jolloin fysiikan simulointi ei toteudu joka kerta samalla tavalla. Ilman fysiikkamoottorin käyttöä, koulutusprosessi voidaan toteuttaa jopa kymmeniä kertoja nopeammin.

5.6 Tulokset

Tekoälyn koulutuksessa pääsi nopeasti alkuun, mutta täysin toimivaa siitä ei saatu. Tekoälylle oli nopea opettaa yksinkertaisia toiminnallisuuksia, mutta kaikkien toiminnallisuuksien yhdistäminen yhdeksi toimivaksi kokonaisuudeksi ei onnistunut. Agentti oppi kyllä tähtäämään ja väistämään ammuksia, mutta ei riittävän hyvin, että sitä olisi voinut käyttää sellaisenaan valmiissa pelissä. Hyperparametrien oikeat säädöt ja agenttien palkitseminen osoittautuivat kriittisiksi.

6 POHDINTA

Koneoppimista on käytetty jo pitkään, mutta avoimen lähdekoodin kirjastojen ja algoritmien myötä sen hyödyntäminen pienemmissä projekteissa on mahdollistunut vasta viime vuosina. Nykyään on olemassa monia avoimen lähdekoodin kirjastoja ja työkaluja, jotka on suhteellisen helppo ottaa käyttöön pienemmissäkin projekteissa. Koneoppimisesta löytyy nykyään paljon teoriatietoa, mutta tekoäly on usein yksilöllinen ja sen toteuttamiseen liittyvät ongelmat on ratkaistava itse.

Insinöörityön tarkoituksena oli tutkia koneoppimisen käyttöä pienissä peliprojekteissa. Työtä varten kehitettiin peli, jonka pääprioriteetti oli tekoälyn opetuksessa, eikä pelin muilla ominaisuuksilla ollut isoa merkitystä. Tarkoituksena oli luoda tilanne, jossa tekoälyn toteuttaminen olisi vaikeaa perinteisin menetelmin ja jossa koneoppimisen hyödyt tulisivat hyvin esiin.

Koneoppimisen käyttöön ottamiseen löytyi hyvin tietoa, mutta monimutkaisemmat ongelmat täytyi ratkaista käytännössä yrityksen ja erehdyksen kautta. Isoimmaksi ongelmaksi työssä muodostuivat tekoälyn koulutuksen pitkät harjoitusajat. Tekoälyn ongelmat ilmenivät usein vasta kun koulutus oli edennyt riittävän pitkälle ja virheiden korjaaminen vaati usein koulutuksen uudelleen aloittamista. Tekoälylle haluttujen toiminnallisuuksien muuttaminen jälkikäteen koettiin olevan melko vaikeaa, koska muutosten onnistuminen pystyttiin todentamaan vasta useita tunteja kestävän koulutuksen jälkeen. Hyvin suunniteltu ja toteutettu koulutus kuitenkin tuotti hyviä tuloksia kohtuullisen nopeasti. Koneoppiminen vaikutti varteenotettavalta vaihtoehdolta tilanteissa, joissa tekoälylle asetetut tavoitteet olivat sopivat.

Valmiiseen peliin koneoppimista hyödyntävän tekoälyn lisääminen on suhteellisen helppoa, koska pelin mekaniikat ja säännöt ovat jo valmiina. Pelikehityksessä kuitenkin usein tarvittaisiin tekoälyä jo kehityksen aikana, jotta pelimekaniikoita voidaan testata ja muokata tarpeen mukaan. Koska tekoälyn koulutusprosessissa saattaa kestää useita päiviä, ja koska vastustajan tekoäly on keskeinen asia pelissä, tekoälyä on vaikea kouluttaa keskeneräiseen peliin, jonka säännöt eivät ole vielä lopullisia, ja lopullisia sääntöjä on vaikea todeta toimiviksi ilman toimivaa tekoälyä. Mitä paremmin pelin pystyy ennalta suunnittelemaan, sitä helpompaa on koneoppimisen hyödyntäminen pelissä. Työkalujen parantuessa ja niiden käytön yleistyessä koneoppimisen hyödyntäminen peleissä luultavasti tulee riittävän helpoksi, mahdollistaen koneoppimisen käytön pienissäkin peleissä.

LÄHTEET

Elements of AI. Koneoppimisen lajit. Saatavissa: https://course.elementsofai.com/fi/4/1. Viitattu 16.2.2023.

Unity-Technologies. Agents. Saatavissa: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Design-Agents.md. Viitattu 16.2.2023.

Unity-Technologies. ML-Agents Toolkit Overview. https://github.com/Unity-Technologies/ml-agents/blob/release_6_docs/docs/ML-Agents-Overview.md. Viitattu 16.2.2023.

Unity-Technologies. Installation. Saatavissa: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Installation.md. Viitattu 16.2.2023.

Unity-Technologies. Background: Machine Learning. Saatavissa: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Background-Machine-Learning.md. Viitattu 16.2.2023.

Unity-Technologies. Background: PyTorch. Saatavissa: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Background-PyTorch.md. Viitattu 16.2.2023.

Unity-Technologies. Using TensorBoard to Observe Training. Saatavissa: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Using-Tensorboard.md. Viitattu 16.2.2023.

Unity-Technologies. Introducing: Unity Machine Learning Agents Toolkit. Saatavissa: https://blog.unity.com/technology/introducing-unity-machine-learning-agents. Viitattu 16.2.2023

Unity-Technologies. Class Agent. Saatavissa: https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.Agent.html Viitattu 12.4.2023.

Unity-Technologies. Introducing ML-Agents Toolkit v0.2: Curriculum Learning, new environments, and more. Saatavissa: https://blog.unity.com/community/introducing-ml-agents-v0-2-curriculum-learning-new-environments-and-more. Viitattu 12.4.2023.

Unity-Technologies. Training Configuration File. Saatavissa: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md#ppo-specific-configurations. Viitattu 12.4.2023.

Cohen, A. 2020. Training intelligent adversaries using self-play with ML-Agents. Saatavissa: https://blog.unity.com/technology/training-intelligent-adversaries-using-self-play-with-ml-agents. Viitattu 12.4.2023.

Wikipedia. Shoot 'em up. Saatavissa: https://en.wikipedia.org/wiki/Shoot_%27em_up. Viitattu 16.2.2023.

Yegulalp, S. 2020. What is TensorFlow? The machine learning library explained. Saatavissa: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html. Viitattu 16.2.2023.

D'Archimbaud, E. Programming Image Classification with Machine Learning: Why and How? Saatavissa: https://kili-technology.com/data-labeling/computer-vision/image-annotation/programming-image-classification-with-machine-learning. Viitattu 12.4.2023.

Brownlee, J. 2019. 14 Different Types of Learning in Machine Learning. Saatavissa: https://machinelearningmastery.com/types-of-learning-in-machine-learning/. Viitattu 12.4.2023.

Nigretti, A. 2018. Imitation Learning in Unity: The Workflow. Saatavissa: https://blog.unity.com/technology/imitation-learning-in-unity-the-workflow. Viitattu 12.4.2023.

Jerga, F. 2021. What Is The Unity Game Engine- All You Need To Know. Saatavissa: https://medium.com/eincode/what-is-the-unity-game-engine-all-you-need-to-know-d4ce77a1b7d2. Viitattu 12.4.2023.

Indiedb. Saatavissa: https://www.indiedb.com/games/guardian-earth-hd/images/bullet-hell Viitattu 12.4.2023