Harri Huhta

Koneoppimisen hyödyntäminen Unity-peli-moottorissa

Tradenomi
Tietojenkäsittely
Syksy 2021

Tiivistelmä

Tekijä: Huhta Harri

Työn nimi: Koneoppimisen hyödyntäminen Unity-pelimoottorissa

Tutkintonimike: Tradenomi (AMK), tietojenkäsittely

Asiasanat: koneoppiminen, tekoäly, Unity

Työn avulla pyritään antamaan lukijalle tiivis tietopaketti koneoppimisesta ja sen hyödyntämisestä pelikehityksessä Unity-pelimoottorissa. Tarkoitus on antaa käytännön ohjeistusta koneoppimisen hyödyntämiseen soveltuvista työmenetelmistä ja työkaluista sekä riittävä teoriapohja koneoppimisen toiminnasta kokonaisuuden tukemiseksi.

Työn teoriapuoli painottuu koneoppimisen perusteiden tutkimiseen ja eri koneoppimisen menetelmien esittelyyn ja arviointiin pelikehityksen näkökulmasta. Työssä perehdytään erityisesti Unityn ML-Agents-kirjaston toimintaan, ja sen hyödyntämiseen pelin tekoälyn kehityksessä. Työssä tutkitaan ML-Agents-ympäristön toimintatapoja, oppimisagenttien ominaisuuksia ja yleisiä kompastuskiviä koneoppimisagenttien toteuttamisessa. Työssä käydään läpi ML-Agents-koneoppimisagenttien suunnittelemisen ja kehittämisen menetelmiä, jotka muodostavat yhtenäisen työprosessin koneoppimista hyödyntävien tekoälyagenttien valmistamisessa.

Teorian tutkimisen jälkeen seuraa käytännön osio, jossa tutkittuja periaatteita ja työkaluja sovelletaan yksinkertaisten pelidemojen luomiseen. Pelidemot havainnollistavat konkreettisemmin koneoppimisen käyttöön liittyvää työprosessia ja koneoppimisella saavutettavia tuloksia. Pelidemojen kehitysprosessi dokumentoidaan käyden läpi kunkin koneoppimisagentin ominaisuudet sekä agentin kehityksen tärkeimmät tavoitteet ja haasteet.

Agenttien kehityksessä kohdattujen haasteiden yhteydessä dokumentoidaan myös menetelmät, joilla ongelmat ratkaistiin. Erityistä huomiota kiinnitetään kehitettyjen koneoppimisagenttien havaintojen suhteuttamiseen ja normalisointiin, sillä niiden koetaan vaikuttavan merkittävästi agenttien oppimistahtiin ja opitun käytöksen laatuun. Työ antaa kattavan yleiskuvan koneoppimisagenttien toiminnasta ja tarjoaa käytännönläheisiä neuvoja koneoppimisagenttien kehittämisestä ML-Agents-ympäristössä.

KAMK • University of Applied Sciences

Abstract

Author: Huhta Harri

Title of the Publication: Utilizing Machine Learning in the Unity Game Engine

Degree Title: Bachelor of Business Administration, Business Information Technology

Keywords: machine learning, artifical intelligence, Unity

The goal of this thesis was to provide the reader with a compact collection of information about machine learning and utilizing machine learning in game development within the Unity game engine. The purpose was to give practical advice about suitable working methods and tools for utilizing machine learning, and a sufficient understanding of the theory of machine learning to support the big picture.

The theory section of this thesis focuses on studying the core principles of machine learning and evaluating various machine learning methods from the perspective of game development. The thesis delves especially into the workings of the Unity ML-Agents library, and how it can be utilized in developing artificial intelligence in video games. The thesis studies the working methods of the Unity ML-Agents environment, the properties of machine learning agents and common stumbling blocks in implementing machine learning agents. The thesis reviews methods for designing and developing machine learning agents, forming a coherent workflow for producing artificial intelligence agents that utilize machine learning.

The study of theory is followed by a section of practice, where the studied principles and tools are applied in creating simple game demos. The game demos demonstrate more concretely the working process involved in utilizing machine learning and the results machine learning can help achieve. The development process of the game demos is documented by going through the properties of each machine learning agent and by reviewing the main objectives and challenges involved in developing them.

Problems faced during the development of agents are documented along with methods used to solve them. Special attention was paid to making the agents' observations relative and normalized, because it was found to significantly affect the learning rate of the agents and the quality of learned behavior. The thesis provides a comprehensive overview of the functionality of machine learning agents and supplies practical advice for developing machine learning agents in the ML-Agents environment.

Sisällys

1 Johdanto

Tämä opinnäytetyö tutkii koneoppimisen hyödyntämistä pelinkehityksen näkökulmasta. Koneoppiminen on alati kasvava trendi teknologian aloilla, ja sen suosio on kasvanut rajusti viime vuosien varrella. Myös pelialalla koneoppimisen hyödyntämiseen on kohdistunut kasvavaa kiinnostusta, johtaen uusien koneoppimistyökalujen kehitykseen ja niiden integroimiseen pelimoottoreihin. Opinnäytetyön aihe on valittu vastaamaan sekä koneoppimisen kasvavaan suosioon, että uusien kehitystyökalujen tuomaan aiheen ajankohtaisuuteen.

Työ on päätetty toteuttaa Unity-pelimoottorilla, koska Unity on yksi suosituimmista pelimoottoreista ja pelimoottorin sisältämiä integroituja koneoppimistyökaluja kehitetään aktiivisesti. Työ keskittyy Unity-moottorin ML-Agents-koneoppimiskirjastoon, jonka tarkoitus on tarjota vahvasti pelimoottoriin integroitu ympäristö koneoppimisagenttien kehitykselle ja koulutukselle. Yksittäisen pelimoottorin koneoppimistyökaluihin keskittymisestä huolimatta opinnäytetyön on määrä käsitellä myös yleispäteviä koneoppimiseen ja koneoppimisagenttien suunnitteluun liittyviä seikkoja, joiden hyödyllisyys ei ole sidottu yksin valittuun kehitysympäristöön.

Työn päätehtävä on koota yleistä tietoa koneoppimisesta ja sen hyödyntämisestä Unity-pelimoottorissa. Kootun tiedon pohjalta toteutetaan koneoppimista hyödyntäviä tekoälyagentteja, joiden kehitysprosessin dokumentoimisella havainnollistetaan työprosessia ja tuloksia. Kehitettyjen demojen päätavoite on toimia yksinkertaisina esimerkkeinä koneoppimisen hyödyntämisestä Unity-pelimoottorissa, joten niiden kehityksessä keskitytään ensisijaisesti tekniseen toteutukseen graafisen ilmeen tai pelikokemuksen sijasta.

Opinnäytetyön on määrä selvittää muun muassa, kuinka koneoppimista hyödynnetään Unity-pelimoottorissa, mitä eroavaisuuksia Unityn ML-Agents-kirjaston tarjoamien koneoppimismenetelmien välillä on ja mitä seikkoja on otettava huomioon koneoppimisagentteja suunnitellessa.

2 Yleistä koneoppimisesta

Koneoppiminen on tekoälyn kouluttamista datan ja matemaattisten koulutusalgoritmien avulla suoran ohjeistuksen sijasta. Koneoppimiseen tarkoitetut koulutusalgoritmit erikoistuvat tunnistamaan niille syötetystä datasta johdonmukaisuuksia. Koulutuksen aikana löydettyjä johdonmukaisuuksia käytetään koostamaan koneoppimismalli, joka kykenee tuottamaan aiempiin havaintoihin perustuvia ennusteita. [1.] Koneoppimismalleja voidaan kehittää tuottamaan yhä tarkempia ennusteita kouluttamalla niitä lisää ja lisäämällä koulutuksessa käytettävän datan määrää [2]. Koneoppimista hyödyntävä tekoäly kykenee siis imitoimaan ihmisten tapaa muodostaa havainnoista kokemuksen myötä tarkentuvia syy-seuraussuhteita, joiden avulla ennusteiden tuottaminen on mahdollista [3]. Koneoppimista käyttävän tekoälyn poikkeuksellinen sopeutumiskyky on erityisen hyödyllinen tehtävissä, joissa käsiteltävän datan määrä, monimutkaisuus tai jatkuva muuttuminen tekevät perinteisiä ohjeistuksia noudattavan tekoälyn ohjelmoimisesta haasteellista [1].

Viime vuosina räjähdysmäisesti kasvaneesta suosiostaan huolimatta koneoppiminen on jo suhteellisen pitkään tutkittu ilmiö. Ensimmäinen elektroninen neuroverkkojen prototyyppi kehitettiin jo vuonna 1943 havainnollistamaan neurofysiologisti Warren McCullochin ja matemaatikko Walter Pittsin mallia neuronilogiikan toiminnasta. McCulloch ja Pitts pyrkivät esittämään pelkistetyn esimerkin siitä, kuinka aivot voisivat kyetä toteuttamaan monimutkaisia toimintoja käyttäen suurta määrää yksinkertaisia toisiinsa yhteydessä olevia soluja, neuroneja. [4.] McCullochin ja Pittsin mallin perustana toimi heidän pelkistetty neuronimallinsa, niin kutsuttu MCP-neuroni. MCP-neuroni kuvaa yksittäistä logiikkayksikköä, jolla on summattu syötearvo, syötteen luokitteleva raja-arvo ja kaksi mahdollista tulosarvoa. Tulosarvo riippuu siitä, ylittääkö syötearvo raja-arvon vai ei. McCullochin ja Pittsin neuronimallilla voidaan teoriassa rakentaa mikä tahansa tietokoneella suoritettava looginen päätelmä käyttäen ainoastaan joukkoja toisiinsa yhdistettyjä MCP-neuroneja. [5.] MCP-neuronit eivät yksin kuitenkaan voineet ”oppia” mitään, sillä ne ovat yksinkertaisia, pääosin muistittomia logiikkayksiköitä. McCullochin ja Pittsin malli kuitenkin pohjusti koneoppimisen kehittämistä määrittelemällä alustavan mallin neuronien ja neuroverkkojen logiikalle ja havainnollisti, kuinka loogisia tehtäviä voidaan asetella neuroverkoilla ratkottavaksi. [5.] Jo vuosikymmenien ajan lukuisat tutkijat ovat kehittäneet neuroverkkojen ideaan pohjautuvan teknologian tutkimusta. Vuonna 1949 Donald O. Hebb esitti teorian neuroverkkojen oppimiskyvystä kirjassaan ”The Organization of Behavior”. Hebbin mukaan läheisten neuronien välinen

yhteys vahvistuu niiden aktivoituessa yhdessä, ja heikkenee niiden aktivoituessa toisistaan erillään. [6.] Hebbin oppimismalli oli aikainen teoria neuroverkkojen ohjaamattomasta oppimisesta, ja periaate neuronien yhteyksien painoarvottamisesta toimi jatkossa tärkeänä osana neuroverkkojen oppimisalgoritmien kehityksessä [7].

Jo 1950-luvulla IBM:n tutkija Arthur Samuel onnistui kehittämään ensimmäisen itsestään oppivan tammipeliä pelaavan tekoälyn [2]. Ensimmäinen käytännön haasteita ratkova neuroverkko taas kehitettiin Stanfordin yliopistossa 1959, kun Bernard Widrow ja Marcian Hoff kehittivät ADALINE-järjestelmän, jota voitiin hyödyntää kaiun poistamisessa puhelinlinjoilla [6]. Vuosikymmenien kehitystyön aikana neuroverkkojen teoria ja käyttömenetelmät ovat kokeneet valtavia harppauksia, ja ajoittaisista takaiskuista huolimatta koneoppiminen on tehnyt paluun yhdeksi suurimmista teknologiatrendeistä. [6] [8] Nykypäivänä koneoppimista käytetään laajasti jokapäiväisen elämän tuotteiden ja palveluiden kehityksessä. Koneoppimista hyödynnetään tuotesuosituksissa, hinnoittelussa, puheentunnistuksessa, kasvojentunnistuksessa ja lukuisissa muissa käyttötarkoituksissa niin saumattomasti, ettei loppukäyttäjä välttämättä tiedä käyttävänsä koneoppimista hyödyntävää tuotetta tai palvelua. [3.] [9.] [10.]

2.1 Koneoppimisen alalajit

Koneoppiminen käsitteenä sisältää valtavan kirjon eri menetelmiä ja käyttötarkoituksia. Eri koneoppimistekniikoita on olemassa satoja, mutta kaikki ne noudattavat samaa yhdenmukaista toimintaperiaatetta: datassa olevien säännönmukaisuuksien ja riippuvuuksien etsimistä ja niiden käyttämistä ymmärryksen ja ennusteiden tuottamiseen. [11.] Koska koneoppiminen perustuu olemassa olevan datan säännönmukaisuuksiin, on koneoppimisen onnistumisen kannalta tärkeää, että säännönmukaisuutta tuetaan jo suunnitteluvaiheessa. Koneelle annettujen syötteiden ja siltä odotettujen vasteiden täytyy olla selkeästi määriteltävissä ja mahdollisimman yhdenmuotoisia. [12.] Kone ei tiedä syötteiden muotoa, joten jos syötteiden määrä tai muoto vaihtelee odottamattomasti, on koneen vaikea tehdä niistä yhdenmukaisia päätöksiä. Vastaavasti myös onnistumisen kriteerit on suunniteltava niin, että koneen saavuttamat lopputulokset voidaan asettaa selkeään paremmuusjärjestykseen [12].

Koneoppiminen voidaan pääosin jakaa kolmeen alalajiin – ohjattu oppiminen, ohjaamaton oppiminen ja vahvistusoppiminen [13, s. 23]. Ohjattu oppiminen tapahtuu käyttäen valmiiksi merkittyjä syötteitä ja vasteita, joiden perusteella algoritmi oppii tekemään ennusteita myös uusille

syötteille. Sekä syötteet että ulostulo eli vaste ovat ohjatussa oppimisessa merkittyjä. Näin ollen ulostulon muoto tiedetään valmiiksi, eikä sitä tarvitse erikseen tulkita. [14, s. 35.] Ohjattua oppimista voidaan käyttää luokittelemiseen tai regressioon. Luokittelussa algoritmilla on ennalta määritelty valikoima vaihtoehtoja, joista sen täytyy valita paras ehdokas syötteen vasteeksi. Regressiossa sen sijaan voi olla periaatteessa loputon määrä eri lopputuloksia ja vaste ennustetaan arvioimalla jatkuvan muuttujan arvo. [15.]

Ohjaamattomassa oppimisessa koulutusdataa tai syötteitä ei ole merkitty, vaan algoritmin tehtävä on etsiä syötteiden väliltä yhtäläisyyksiä ja eroja. [14, s. 35.] Ohjaamaton oppiminen voi pyrkiä joko assosiointiin tai ryhmittelyyn. Assosioinnissa samankaltaisten syötteiden välille muodostetaan yhteyksiä, kun taas ryhmittelyssä muodostetaan ryhmiä samankaltaisista syötteistä. [16.] Ohjaamattomassa oppimisessa lopputulokset siis kertovat yhdenkaltaisuuksista eri syötteiden välillä, mutta yhteyksien tulkinta jää avoimeksi [15].

Vahvistusoppimisessa taas ei ole ennalta määriteltyä koulutusdataa. Oppimisagentti kerää syötteitä ympäristöstään ja oppii kohti haluttua käytöstä kokeilemalla eri syötteiden ja vasteiden pareja, suosien niitä, jotka johtavat suurempaan palkkioon. [15.] Vahvistusoppimisen lopputuloksena on käytös, joka saavuttaa agentille määritellyssä ympäristössä suurimman mahdollisen palkkion. Näin ollen palkkioiden tarkalla suunnittelulla on suuri vaikutus vahvistusoppimisessa. [17.]

2.2 Koneoppiminen peleissä

Koneoppiminen ja sen hyödyntäminen ovat kokeneet valtavia kasvupyrähdyksiä viime vuosien aikana, ja tämä näkyy myös pelialalla [18]. Strategiapeleissä on aiemminkin hyödynnetty vahvistusoppimista tekoälyn toteuttamiseen, mutta kasvavissa määrin koneoppimisen valtavaa potentiaalia pyritään valjastamaan tekoälyn ohjaamiseen myös nopeatempoisissa AAA-peleissä [19][20][21]. EA:n SEED (Search for Extraordinary Experiences Division) on kehittänyt tekoälyn, joka opettelee pelaamaan Battlefield 1 -moninpeliä. SEED:n kehittämä tekoäly tekee peliympäristöstään reaaliaikaisia havaintoja käyttäen samaa näkökenttää kuin pelaajat ja osaa niiden perusteella suorittaa tehtäviä sekä taistella vastustajia vastaan. SEED:n teknisen ohjaajan Magnus Nordinin mukaan koneoppimista on lyhyellä aikavälillä määrä käyttää enimmäkseen DICE:n pelien laadunvarmistuksessa, mutta pidemmällä aikavälillä koneoppimisagenttien odotetaan ohjaavan myös lopputuotteen pelimaailmassa toimivia hahmoja. [19.]

Vaikka pelien hahmojen päätöksiä ohjaakin edelleen useimmiten perinteinen tekoäly koneoppivan agentin sijasta, koneoppimista hyödynnetään myös muilla tavoin pelien kehityksessä. Kasvavissa määrin pelinkehityksen työkalut hyödyntävät koneoppimista suurten datamäärien käsittelemiseksi ja aikaa vievien työprosessien automatisoimiseksi. Pelistudiona ja julkaisijana toimiva Ubisoft on kehittänyt useita koneoppimista hyödyntäviä menetelmiä pelien hahmoanimaatioiden kehitysprosessin parantelemiseksi. Ubisoftin kehittäjien mukaan koneoppimista hyödyntävät animaatiomenetelmät mahdollistavat hahmoanimaatioiden interpoloimisen luonnollisempien animaatioiden saavuttamiseksi tavalla, joka säästää resursseja vähentämällä sekä animaatioiden tekemiseen tarvittavaa työaikaa että tietokoneen muistin käyttöä. [22.] Aiemmat toteutukset alalla yleisistä kontekstitietoisista hahmoanimaation toistomenetelmistä vaativat toimiakseen kattavia animaatiotietokantoja, joiden muistinkäyttö kasvaa animaatioiden määrän myötä. Tämä muistinkäytön kasvu on noussut ongelmaksi AAA-peleissä, jotka voivat nykypäivänä sisältää kymmeniä tuhansia hahmoanimaatioita. [23.]

Koska koneoppiminen mahdollistaa suuren datamäärän ja huomattavan laskentavoiman käyttämisen aikaa vievien työtehtävien automatisoimiseksi, siitä on tullut myös merkittävä työkalu pelien huijauksenesto-ohjelmistojen arsenaalissa [24]. Moninpelien huijauksenestoon kehitys on perinteisesti ollut loputon juoksukilpailu huijausohjelmistojen kehittäjiä vastaan, mutta koneoppiminen voi auttaa yleispätevämmän ja mukautuvamman ratkaisun luomisessa [25]. Vuonna 2017 Valve implementoi Counter Strike Global Offensiveen syväoppimista hyödyntävän tekoälyn, jonka tarkoitus oli auttaa huijareiden tunnistamisessa riippumatta huijareiden käyttämästä huijausohjelmistosta [24]. Pelissä oli jo vuosia ollut käytössä Overwatch-raatijärjestelmä, jossa pelaajat auttoivat huijareiden kiinniotossa katsomalla pelattujen otteluiden uusintoja ja merkitsemällä niistä huijareita. Tämä raatijärjestelmä tarjosi antoisan tilaisuuden syväoppimisen implementoinnille, sillä sen avulla saatiin suuria määriä eriteltyä dataa sekä huijaavien että viattomien pelaajien pelityyleistä. [24.] Valve koulutti raatijärjestelmän datalla VacNet-nimisen tekoälyn, joka tunnistaa peleistä huijareita ja lähettää ne raadin tuomittaviksi. VacNetin implementoimisen myötä Overwatch-raatijärjestelmän tehokkuus huijareiden kiinniottamisessa moninkertaistui. [24.]

Koneoppimiskirjastoja on nykyään saatavilla paljon, mutta pelimoottorien puolella virallinen tuki ja integraatio koneoppimismenetelmille on kuitenkin usein vähäistä tai olematonta. Suurimmassa osassa suosituimmista pelimoottoreista koneoppimistyökalujen integraatio on joko kehitettävä itse tai asennettava kolmannen osapuolen lisäosilla. Pelimoottoreiden joukosta erottuu tässä tilanteessa eniten Unity, jolla on virallinen pelimoottorin kehittäjien luoma ML-Agents-kirjasto,

joka tarjoaa kattavan integraation ja monipuoliset työkalut koneoppimisagenttien kehitykseen ja koulutukseen suoraan pelimoottorin ympäristössä [17].

3 Unity ML-Agents

ML-Agents on Unityn virallinen avoimen lähdekoodin kokoelma työkaluja tekoälyn hyödyntämiseen Unity-pelimoottorissa. Sen tarkoitus on tarjota yksinkertainen rajapinta Pythonilla toimivaan koneoppimisympäristöön ja antaa pohja siihen yhteensopiville koneoppimisen agenteille. [26] ML-Agents käyttää suosittua Python-koneoppimiskirjastoa nimeltään TensorFlow, joka hoitaa varsinaisen koneoppimismallien kouluttamisen pelimoottorin tarjoamalla datalla. TensorFlow:n luomia malleja voi koulutuksen jälkeen tuoda takaisin Unityyn, ja niitä voidaan käyttää tekoälyn ohjaamiseen peleissä tai simulaatioissa. TensorFlow:n käyttö mahdollistaa myös koneoppimisen datan graafisen visualisoinnin TensorBoard-työkalulla. TensorBoard muodostaa TensorFlow:n keräämästä datasta graafeja ja helpottaa koneoppimisen etenemisen seuraamista ja opitun käytöksen pisteellistä arvioimista. [27.]

Unityn ML-Agents-ympäristössä koneoppivia tekoälyjä kehitetään toteuttamalla erilaisia Agent-luokan periviä skriptejä. Agent-luokka toimii rajapintana Unity-peliympäristön ja taustalla toimivan koneoppimisalgoritmin välillä. Se välittää peliympäristöstä tehdyt havainnot eteenpäin oppimisalgoritmille, ja toteuttaa algoritmilta saatuja päätöksiä vastaavat toimet. [17.] Kuvassa 1 esitetään koneoppimisagenttien kehittämisen kannalta oleellisimmat Agent-luokan toteuttamat funktiot.

Agent

```
public void RequestDecision();
public void AddReward(float increment);
public void SetReward(float reward);
public void EndEpisode();

public virtual void Initialize();
public virtual void OnEpisodeBegin();
public virtual void CollectObservations(VectorSensor sensor);
public virtual void Heuristic(float[] actionsOut);
public virtual void OnActionReceived(float[] vectorAction);
```

Kuva 1. Agent-luokan toiminnallisuutta hallinnoivia funktioita ja niiden vastaanottamia parametreja.

ML-Agents-kehikon tarjoamat koulutusmenetelmät voidaan jakaa pääosin kahteen kategoriaan: vahvistusoppiminen sekä imitaatio-oppiminen. Vahvistusoppiminen perustuu lukuisien yrityksien

ja erehdyksien kautta opittuun käytökseen, jota ohjataan antamalla reaaliaikaisia palkkioita halutun käytöksen saavuttamisesta. Koneoppimisalgoritmi käyttää palkkioita apuna päättämään, mitkä havaintojen perusteella tehdyt päätökset johtivat hyviin tuloksiin ja mitkä taas eivät. [17.] Vahvistusoppimisessa tekoäly pyrkii maksimoimaan saamansa palkkiot, käyttäen saadun palkkion määrää mittana opitun käytöksen laadusta. Näin ollen agenttien palkitsemisperusteet on valittava erityisen huolellisesti, sillä agentit ohjautuvat helposti kohti yksinkertaisinta tapaa maksimoida saatu palkkio. Huolimattomasti suunnitellut palkkiosäännöt voivat halutun käytöksen lisäksi palkita myös yksinkertaisempia odottamattomia käytöksiä ja ohjata oppimista väärään suuntaan. [17.] Vahvistusoppimisessa käytettäviä palkkioita suunnitellessa on myös tärkeää ottaa huomioon palkkioiden harvuuden luomat haasteet. Koska vahvistusoppiminen muistuttaa etenkin alkuvaiheessa lukuisia satunnaisia ja sokeita yrityksiä, liian haastavasti tai harvaan asetetut palkkiot voivat olla suuri este oppimisen edistymiselle. Jos koneoppimisagentti ei satunnaisilla kokeiluilla onnistu saavuttamaan vähäistäkään palkkiota, se ei kykene arvioimaan eri käytöksien laatua, eikä siksi kykene jalostamaan laadukkaampaa käytöstä, joka maksimoisi saadun palkkion määrän. Pienempien ja tiheämpään jaettujen palkkioiden antaminen voi mahdollistaa oppimisen nopeamman edistymisen ja parantaa mahdollisuuksia hienosäätää agentin toimintaa palkkiosääntöjä muokkaamalla. [17.]

Imitaatio-oppiminen puolestaan perustuu nimensä mukaisesti haluttua käytöstä esittävien mallisuoritusten imitoimiseen. Imitaatio-oppimisessa agentin koulutusta varten taltioidaan mallisuorituksia, joissa päätöksiä ohjaa agentin sijasta esimerkiksi ihmispelaaja. Näiden mallisuorituksien perusteella koneoppimisalgoritmi voi etsiä riippuvuussuhteita peliympäristöstä tehtyjen havaintojen ja ohjaajan tekemien päätöksien välillä sekä kouluttaa annettuja mallisuorituksia imitoivan käytöksen. [28.] Unityn ML-Agents-ympäristössä on saatavilla kaksi eri imitaatio-oppimisalgoritmia, BC (Behavioral Cloning) ja GAIL (Generative Adversial Imitiation Learning). Behavioral Cloning kouluttaa koneoppimisagentin kopioimaan mallisuoritusten toimia sellaisinaan, muttei osaa yleistää oppimaansa mallisuorituksien sisältämien tilanteiden ulkopuolelle. Näin ollen Behavioral Cloning sopii parhaiten koulutuksiin, joissa lähes kaikki olennaisimmat agentin olotilat käydään läpi mallisuorituksissa. Generative Adversial Imitiation Learning sen sijaan kykenee yleistämään oppimaansa monipuolisemmin. GAIL-algoritmi kouluttaa samanaikaisesti kahta vastakkaista osapuolta, koneoppimistehtävää suorittavaa agenttia sekä syrjijää, jonka tehtävä on oppia erottamaan agentin suoritukset mallisuorituksista. Agentti saa suorituksistaan sitä enemmän palkkiota mitä enemmän ne muistuttavat mallisuorituksia, mutta syrjijän oppiessa erottamaan agentin suoritukset mallisuorituksista onnistumisen kriteerit myös tiukkenevat jatkuvasti. [17.]

Haluttua käytöstä suoraan imitoimalla imitaatio-oppiva agentti voi usein saavuttaa haluttua käytöstä muistuttavan osaamistason nopeammin, sillä toisin kuin vahvistusoppiva agentti, sen oppiminen ei ole yhtä riippuvainen sattumanvaraisien päätöksien kokeilusta. Imitaatio-oppiminen voi siis etenkin koulutuksen alkuvaiheessa tuottaa nopeammin tulosta, mutta sillä on myös omat heikkoutensa. Koska imitaatio-oppiminen pohjautuu täysin annettuihin mallisuorituksiin, myös mallisuorituksissa ilmenevät epätäydellisyydet päätyvät helposti koulutettuun käytökseen. Mallisuorituksia pidetään lähtökohtaisesti täydellisinä esimerkkeinä halutusta käytöksestä, joten agentti pyrkii välttämään mallisuorituksen toimista poikkeamista huolimatta niiden mahdollisista negatiivisista seurauksista. Näin esimerkiksi mallisuorituksien aikana peliä ohjanneen ihmisen tekemät virheet tulkitaan haluttuna käytöksenä, ja imitaatio-oppiva agentti toistaa samat virheet. [17.] Jos rallipelin autoa ohjaavassa mallisuorituksessa pelaaja luisuu usein radalta pientareelle, kyseistä suoritusta imitoiva agentti voi oppia pyrkimään samaan, vaikka rallipelin tarkoituksen ymmärtävälle ihmiselle se olisikin ilmiselvä virhe. Tämä ongelma voi tehdä erityisen laadukkaiden mallisuoritusten keräämisestä vaikeata, sillä ihmisille virheiden tekeminen haastavissa tehtävissä on luonnollista, kun taas riittävän hyvän perinteisen tekoälyn ohjelmoiminen voi olla haastavaa. Toisaalta ajoittainen inhimillisten virheiden tekeminen voi tilanteesta riippuen myös lisätä tekoälyn käytöksen ihmismäisyyttä, mikä voi etenkin peleissä olla haluttu piirre [28]. Toinen imitaatio-oppimiselle tyypillinen ongelma on mallisuoritusten monipuolisuuden tai yleispätevyyden puute. Mallisuorituksia imitoimalla koneoppimisagentti oppii pääasiassa selvittämään sen saamien esimerkkien kaltaisia tilanteita. Mikäli mallisuoritukset ovat liian yksipuolisia, voi agentin käytöksen laatu vaihdella rajusti riippuen havaitun peliympäristön ja mallisuorituksien ympäristön samankaltaisuudesta. Ympäristöissä ja tilanteissa joiden kaltaisia mallisuorituksissa ei esiinny, imitaatio-oppiva agentti on siis usein heikoilla jäillä. [17.]

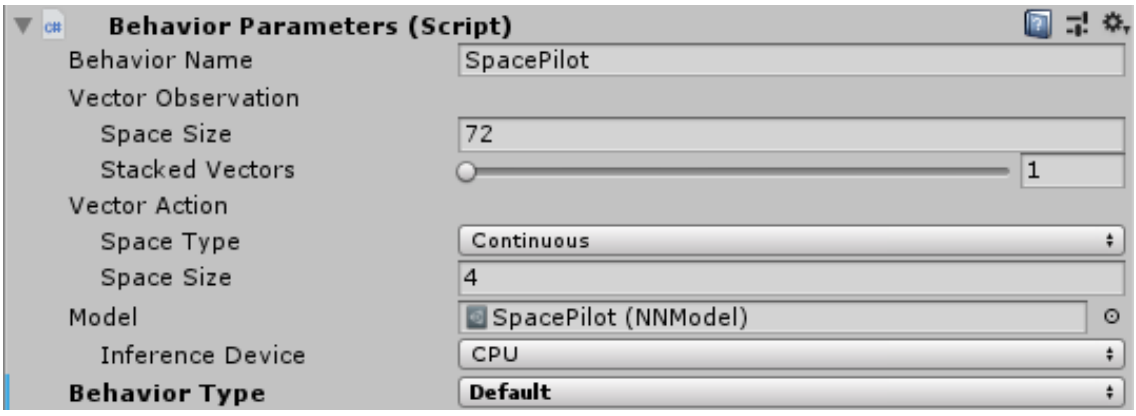
Koska sekä imitaatio-oppimisella että vahvistusoppimisella on omat ominaiset vahvuutensa ja heikkoutensa, voi olla kannattavaa pyrkiä hyödyntämään molempia. Unity ML-Agents mahdollistaa imitaatio- ja vahvistusoppimismetodien sekoittamisen toisiaan täydentäväksi kokonaisuudeksi. Kun imitaatio-oppimisalgoritmeja käytetään yhdessä vahvistusoppimisen kanssa, oppimisalgoritmi kerryttää tavoittelemaansa palkkiota samanaikaisesti sekä tavoitteiden saavuttamisesta jaetusta vahvistusoppimispalkkioista että imitaatio-oppimista varten taltioitujen mallisuoritusten noudattamisesta. [17.] Eri koneoppimismenetelmien antamien palkkioiden painoarvoja on mahdollista hienosäätää agentin konfigurointitiedostossa sopivan tasapainon ja optimaalisen lopputuloksen saavuttamiseksi. Sopiva yhdistelmä imitaatio- ja vahvistusoppimista voi olla tehokas tapa edistää oppimista etenkin ympäristöissä, joissa palkkiot ovat harvassa. Imitaatio-oppiminen

voi silloin tarjota nopeammin palkkiota saavuttavan perusosaamisen, jonka pohjalta vahvistusoppimisella voidaan hioa käytöstä paremmaksi. [17.]

3.1 Agentin valmistelu

Unityssa ML-Agents-agentin valmistelu tapahtuu Initialize- ja OnEpisodeBegin-metodeissa. Initialize on nimensä mukaan tarkoitettu alkuvalmisteluille, ja sitä kutsutaan vain kerran koko oppimisympäristön käynnistyessä. Siellä voi siis suorittaa kaikki valmistelut, joiden odotetaan pysyvän muuttumattomina koko oppimisprosessin ajan. OnEpisodeBegin puolestaan kutsutaan automaattisesti joka kerta, kun yksittäinen oppimisepisodi alkaa. Episodien alku on sopiva aika nollata kaikki harjoittelun aikana mahdollisesti muuttuneet arvot ja varmistaa, että jokainen episodi alkaa samankaltaisesta lähtöasetelmasta. [29.]

Peliympäristössä agenttien skripteille lisätään myös agentin yleisiä asetuksia hallinnoiva komponentti nimeltään ”Behavior Parameters”. Tämän komponentin arvoihin voidaan asettaa editorissa agentin koulutettavan käytöksen nimi, kerralla tehtävien havaintojen määrä, vastaanotettavien päätöksien tietotyyppi ja päätöksien määrä. Kuvassa 2 esimerkki agentille säädetystä ”Behavior Parameters”-komponentista.



Kuva 2. SpacePilot-käytökselle säädetty ”Behavior Parameters” -komponentti.

Mikäli agentin halutaan pyytävän päätöksiä automaattisesti pienin aikavälein reaaliaikaista toimintaa varten, voidaan lisätä myös ”Decision Requester” -komponentti, ja asettaa siitä haluttu aikaväli päätöksien pyytämiselle. [30.] Muussa tapauksessa agentin skriptin RequestDecision-funktiota on kutsuttava manuaalisesti joka kerta, kun agentin halutaan tekevän uuden päätöksen.

3.2 Havaintojen keräys

Agenttien havaintojen kerääminen toteutetaan Agent-luokan CollectObservations-metodissa. Kerättävät havainnot riippuvat aina oppimisympäristöstä. Mitä tietoa agentin tilasta ja ympäristöstä tarvitaan, jotta haluttu käytös voidaan toteuttaa? Esimerkiksi jos agentin tarkoitus on kävellä tien yli autoja väistellen, voidaan oleellisina tietoina pitää ainakin agentin ja autojen sijaintitietoja sekä autojen nopeuksia. Metodi saa parametrina viittauksen sensoriin, jolle kaikki oppimiselle oleelliset havainnot on syötettävä. Havainnot ovat aina todellisuudessa float-tyyppisiä, mutta sensori osaa muuttaa useimmat yleisimmät tyypit yksittäisiksi float-muuttujiksi automaattisesti, joten sille voidaan syöttää suoraan myös vektoreita, kokonaislukuja, kvaternioita ja totuusarvoja. [29.]

Unityn ML-Agents-dokumentaatio suosittelee kerättävien havaintojen normalisointia ennen niiden syöttämistä sensorille huomioiden, että normalisointi ei ole pakollista, mutta se voi auttaa nopeuttamaan koneoppimista. Normalisointi tarkoittaa tässä yhteydessä kaikkien havaintoarvojen skaalausta välille 0–1 tai -1–1. Havaintoarvoja skaalatessa olisi siis päätettävä ensin kullekin havainnolle mahdollisista arvoista suurin ja pienin arvo ja ilmoitettava jokainen havainto suhteellisena sen raja-arvoihin. Erityistä huomiota normalisointitapaan on kiinnitettävä lukujoukkoja kuten vektoreja normalisoidessa. Yleensä vektorien normalisoinnilla tarkoitetaan koko vektorin skaalausta niin, että sen kokonaispituus on 1, mutta havaintovektorien elementit tulisi normalisoida toisistaan erillisinä arvoina, jotka suhteutetaan kyseiselle havainnolle sopiviin raja-arvoihin. [29.]

3.3 Toimien toteuttaminen

Agentit vastaanottavat koneoppimismallin tekemät päätökset metodin ”OnActionReceived” välityksellä. Vastaanotettujen päätöksien tyyppi riippuu agentille valituista asetuksista. Asetuksella ”Continuous” agentti vastaanottaa päätökset liukuarvoina arvojen -1 ja 1 välillä. [29.] Liukuarvot sopivat erityisesti agenteille, joiden on tarkoitus tehdä hienovaraisempia päätöksiä, kuten liikkua tai pyöriä tarkkoja määriä. Asetuksella ”Discrete” päätökset ovat kokonaislukuja ja jokainen kokonaisluku vastaa eri päätöstä. Kokonaislukupäätökset sopivat paremmin tilanteeseen, jossa on ennalta tiedetty rajattu määrä vaihtoehtoja eikä tarvetta välimuodoille ole. Esimerkiksi auton kääntymistä ohjailevalle agentille on hyödyllistä saada hienovaraisempia vaihtoehtoja kuin vasen tai oikea, mutta yksinkertaisiin joko-tai-valintoihin kokonaisluvut ovat hyvä vaihtoehto. [17.]

Vastaanotettujen päätösten perusteella agentin on toteutettava päätöksiä vastaavat toimet. Päätökset vastaanotetaan nimettöminä taulukon osina, joten on päätettävä, mitkä päätöksistä vastaavat mitäkin tekoja. Päätösten järjestys voidaan päättää mielivaltaisesti, kunhan Agentin asetuksista säädetyt päätösvektorien koot täsmäävät tarpeisiin. Päätöksien toteuttamista kannattaa testata ajoissa jo ennen tekoälyn koulutusta käyttämällä agentin Heuristic-metodia, joka mahdollistaa päätöksien manuaalisen syöttämisen testailun aikana. Heuristic-metodi mahdollistaa siis agentin ohjailun esimerkiksi näppäimistön avulla, antamalla metodissa määritellyt sisääntulot päätöksinä OnActionReceived-metodille. Näin voidaan varmistua päätöksien ja toimien oikeanlaisesta toteutuksesta jo ennen tekoälyn kouluttamista. [29.]

3.4 Palkkiot

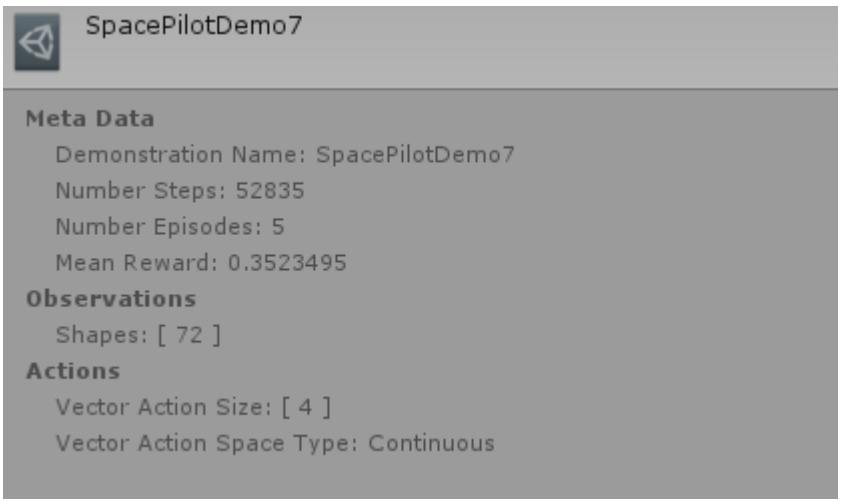
Palkkiot ovat vahvistusoppimisen aikana annettavia arvoja, jotka merkitsevät, milloin agentti on tehnyt jotain oikein. Vahvistusoppimisalgoritmi pyrkii maksimoimaan agentin saamat palkkiot kannustamalla palkkioihin johtavia päätöksiä. [17.] Palkkioita voidaan antaa agentille koska tahansa koulutuksen aikana käyttämällä agentin AddReward- tai SetReward-funktioita. Palkkiona annettu liukuluku voi olla positiivinen tai negatiivinen riippuen siitä, halutaanko nykyistä käytöstä kannustaa vai rangaista. Joissain tilanteissa on myös kannattavaa antaa agentille pientä jatkuvaa negatiivista palkkiota ajan kuluessa, jotta maksimoidakseen palkkion agentin tulee toimia myös mahdollisimman nopeasti. Palkkiot määritellään agentin päätösten välissä, joten jokaisen uuden päätöksen tullessa siihen asti kerrytetty palkkio tallennetaan ja nollataan. Päätösten välissä kerrytettyjen palkkioiden summan tulisi optimitilanteessa olla lukujen -1 ja 1 välillä, sillä Unityn dokumentaation mukaan suuremmat vaihteluvälit voivat johtaa epätasaisiin tuloksiin koulutuksessa. [29.] Jokaisen episodin jälkeen agentin kokonaiskunto arvioidaan sen kerryttämien palkkioiden perusteella. Agenttien koulutuksen edistymistä voi seurata tutkimalla agenttien keskimääräistä episodin aikana kerrytettyä palkkiota. [31.]

Ulkoisten palkkioiden eli ympäristön agentille tarjoamien palkkioiden lisäksi agentin oppimista on mahdollista ohjata agentin sisäisellä uteliaisuutta mallintavalla palkkiolla. Uteliaisuuspalkkio palkitsee agenttia uusien ja yllättävien tilanteiden kohtaamisesta, kannustaen agenttia oppimaan tutkimaan ympäristöään laajemmin. Uteliaisuuspalkkio on erityisen hyödyllinen vahvistusoppimisympäristöissä, joissa palkkiot ovat harvassa tai vaikeasti tavoitettavissa. Vahvistusoppiva koneoppimisagentti vaatii kehittyäkseen palkkioita, ja uteliaisuuspalkkiot voivat auttaa estämään agenttia jumittumasta tilanteissa, joissa sattumanvaraiset kokeilut eivät yksin todennäköisesti

johda ulkoisten palkkioiden saavuttamiseen. [17.] Uteliaisuuspalkkio voidaan ottaa käyttöön ML-Agents-ympäristössä lisäämällä halutun vahvistusoppimisagentin asetustiedostoon arvo ”use_curiosity: true” [32].

3.5 Mallisuoritukset

Mallisuoritukset ovat imitaatio-oppimisessa käytettäviä nauhoitettuja koneoppimisympäristön suorituksia, joiden tarkoitus on esittää haluttu käytös imitaatio-oppivalle agentille [17]. Mallisuorituksia voidaan nauhoittaa helpoiten lisäämällä koneoppimisagentille ML-Agents-kirjaston sisältämä ”DemonstrationRecorder”-skripti, asettamalla päälle skriptin Record-arvo, ja nimeämällä nauhoitettaville mallisuorituksille haluttu nimi ja kansio. DemonstrationRecorder nauhoittaa sitten automaattisesti koulutuksen aikana kyseisen agentin havaintoja, päätöksiä ja palkkioita mallisuorituksiksi. [29.] Mallisuorituksia nauhoitettaessa agentin toimia voidaan ohjata agentin aiemminkin mainitun Heuristic-metodin avulla. Agentin Heuristic-metodi mahdollistaa agentin päätösten manuaalisen määräämisen esimerkiksi pelaajan näppäinkosketusten perusteella. Toteutettuaan agentilleen oman manuaalisen ohjaustapansa Heuristic-metodissa, pelaaja voi asettaa mallisuorituksien nauhoituksen päälle, vaihtaa agentin ohjauksen Heuristic-tilaan, ja ohjata agenttia halutun käytöksen mukaisesti, kunnes sopiva määrä mallisuorituksia on nauhoitettu. Kun mallisuorituksien nauhoittamisen jälkeen pelisessio päätetään, mallisuoritukset tallentuvat ”DemonstrationRecorder”-skriptissä määriteltyyn “.demo”-päätteiseen tiedostoon. [29.] Kuva 3 esittää Unity-editorin näyttämät tiedot nauhoitetusta mallisuorituksesta.



Kuva 3. Tallennettu mallisuoritus Unity-editorissa.

Tallennettua mallisuoritusta voidaan hyödyntää koneoppimisagentin koulutuksessa ”Behavioral Cloning”- ja ”Generative Adversarial Imitation Learning”-imitaatio-oppimismenetelmillä. Mallisuorituksen käyttöönottamiseksi koneoppimisagentin asetustiedostoon on kirjattava tarvittavat lisäasetukset, jotka määrittelevät käytettävän imitaatiomenetelmän sekä osoittavat oikean mallisuoritustiedoston sijainnin. [32.] Valittavat imitaatio-oppimismenetelmät eivät sulje toisiaan pois, vaan molempia menetelmiä on mahdollista hyödyntää saman agentin koulutuksessa. Tällöin asetustiedostoon määritellään molempien imitaatio-oppimismenetelmien lisäasetukset. [17.] Kuva 4 havainnollistaa koneoppimisagentin asetustiedostoa, johon on lisätty imitaatio-oppimismenetelmien lisäasetukset.

```
behaviors:
  SpacePilot:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 2048
      learning_rate: 0.0003
      beta: 0.01
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        strength: 1.0
        gamma: 0.99
+     gail:
+       gamma: 0.99
+       strength: 0.5
+       encoding_size: 128
+       learning_rate: 0.0003
+       use_actions: true
+       use_vail: true
+       demo_path: E:/unity/ML/Machine Learning/Assets/Demos/SpacePilotDemo7.demo
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 64
    summary_freq: 60000
    threaded: true
+   behavioral_cloning:
+     demo_path: E:/unity/ML/Machine Learning/Assets/Demos/SpacePilotDemo7.demo
+     steps: 20000
+     strength: 0.5
+     samples_per_update: 0
```

Kuva 4. Esimerkki koneoppimisagentin asetustiedostosta. Kuvassa korostettuna lisäasetukset, jotka lisättiin nauhoitetun mallisuorituksen käyttämiseksi imitaatio-oppimisessa.

Määriteltyjen mallisuorituksen yhteensopivuus tarkistetaan, kun koulutus käynnistetään. Asetustiedostossa määriteltyjen nauhoitettujen mallisuoritusten täytyy olla nauhoitettu koneoppimis-

agentilla, jonka havainnot ja toimet ovat samassa muodossa. Mikäli koulutettavan agentin havaintojen määrä, päätöksien määrä tai päätöksien tyyppi ei vastaa asetustiedostossa määritellyn mallisuoritustiedoston sisältämiä arvoja, koulutus keskeytyy virheilmoitukseen. Peliympäristöön ja skripteihin voidaan tehdä muutoksia mallisuorituksen nauhoituksen ja koulutuksen välissä, mutta muutokset eivät saa vaikuttaa havaintojen määrään, päätösten määrään tai päätösten muotoon. Vanhoja mallisuorituksia käyttäessä on myös suositeltua välttää muutoksia, jotka vaikuttaisivat merkittävästi agentin tekemien havaintojen sisältöön. Vaikka koulutusprosessi suostuisikin käynnistymään, merkittävät muutokset agentin tekemissä havainnoissa voivat johtaa vialliseen imitaatioon. Siispä agentin havaintoihin tai päätöksiin merkittävästi vaikuttavien muutoksien jälkeen on parasta nauhoittaa uudet mallisuoritukset ennen seuraavan koulutuksen käynnistämistä.

3.6 Episodien hallinta

Jokainen koulutuskerta muodostuu useasta episodista, eli yksittäisistä suorituskerroista, joiden välissä agentit nollataan. Episodit voivat loppua automaattisesti, mikäli agentin asetuksissa asetettu suurin mahdollinen episodin kesto ”max step” saavutetaan.

Agenttien episodeja voi myös lopettaa kutsumalla agentin EndEpisode-metodia. Episodien manuaalinen lopetus kannattaa silloin, kun agentti on saavuttanut lopullisen maalinsa, tai jos agentti on pysyvästi epäonnistunut tehtävässään. Käytännössä episodin lopetus tarkoittaa nykyisen suorituksen tallennusta, ja uuden suorituksen aloittamista valitulle agentille. [29.]

3.7 Koulutus

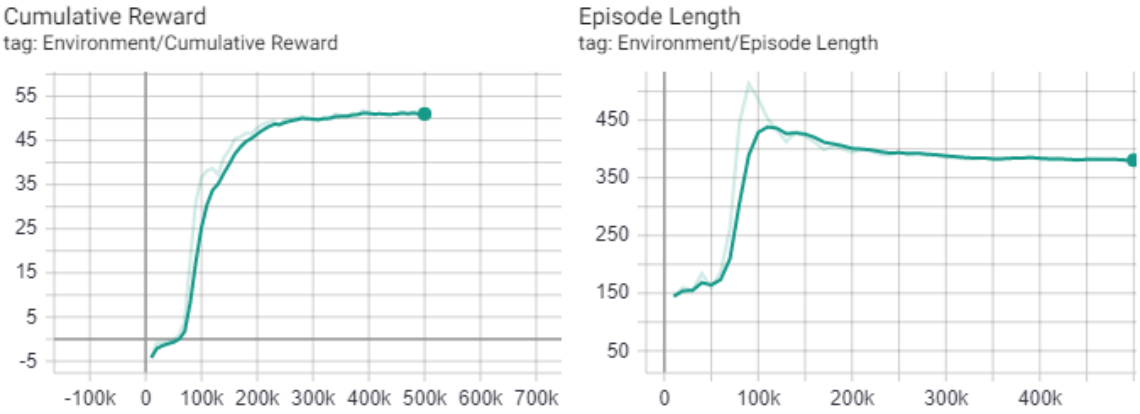
Ennen agentin koulutuksen aloittamista agentille on luotava koulutuksen parametrit määrittelevä YAML-asetustiedosto [30]. Yksinkertaisin tapa päästä alkuun on kopioida yksi ML-Agents-komentokehotetyökalun asennuskansiosta löytyvistä asetustiedostoista ja muokata sen sisältö nykyiselle agentille sopivaksi. Vähimmillään asetustiedostoon pitää muokata kentän ”behaviors” arvoksi sama käytöksen nimi kuin peliympäristössä olevassa ”Behavior Parameters” -komponentissa. Imitaatio-oppimista käyttäessä mallisuorituksiin viittaaviin ”demo_path”-kenttiin on asetettava uuden agentin mallisuoritusten sijainti. [32.] Tarkempia asetusten hienosäätöjä tehdessä

kannattaa ottaa mallia samankaltaisten ML-Agents-esimerkkiprojektien käyttämistä asetuksista, sekä ML-Agents-dokumentaatiosta [30].

Agentin valmistelun jälkeen koulutusympäristö voidaan käynnistää käyttäen Unityn ML-Agents-komentokehotetyökalua. Käynnistäminen tapahtuu käyttämällä komentoa ”mlagents-learn”, ja tarjoamalla parametreinä valitun agentin asetustiedosto sekä vapaavalintainen nimi, jolla nykyinen koulutuskerta tunnistetaan. Komentokehotetyökalu lataa valitut asetukset valmiiksi, ja odottaa kunnes ympäristö käynnistetään painamalla ”play”-nappia Unityssa. [33.] Kuvassa 5 esitetään esimerkki komentokehotteella käynnistetystä koulutussessiosta.

Kuva 5. Komentokehotteessa käynnistetty ML-Agents-koulutussessio.

Agenttien koulutuksen edistymistä on mahdollista seurata editorissa koulutuksen aikana, mutta agenttien saavuttamista palkkioista ja nykyisestä edistymisestä kerätään myös automaattisesti dataa, joka auttaa seuraamaan tilannetta objektiivisemmin. Näiden tilastojen esittämiseen voidaan käyttää ML-Agents-komentokehotetyökalun mukana tulevaa TensorBoard-työkalua. TensorBoard muodostaa kerätystä datasta graafeja ja esittää ne nettiselaimella avattavalla paikallisella HTML-sivulla. Komento ”tensorboard --logdir results --port 6006” käynnistää TensorBoard-sovelluksen, ja julkaisee datasta luodun HTML-sivun paikallisesti. Sovelluksen ollessa päällä, graafeja esittävä sivu voidaan avata navigoimalla nettiselaimella paikalliseen osoitteeseen ”localhost:6006”. [31.] Kuvassa 6 TensorBoardin automaattisesti luomia graafeja, jotka kertovat agentin koulutuksen etenemisestä.



Kuva 6. TensorBoard-graafeja agentin keräämästä palkkiosta ja episodien kestosta.

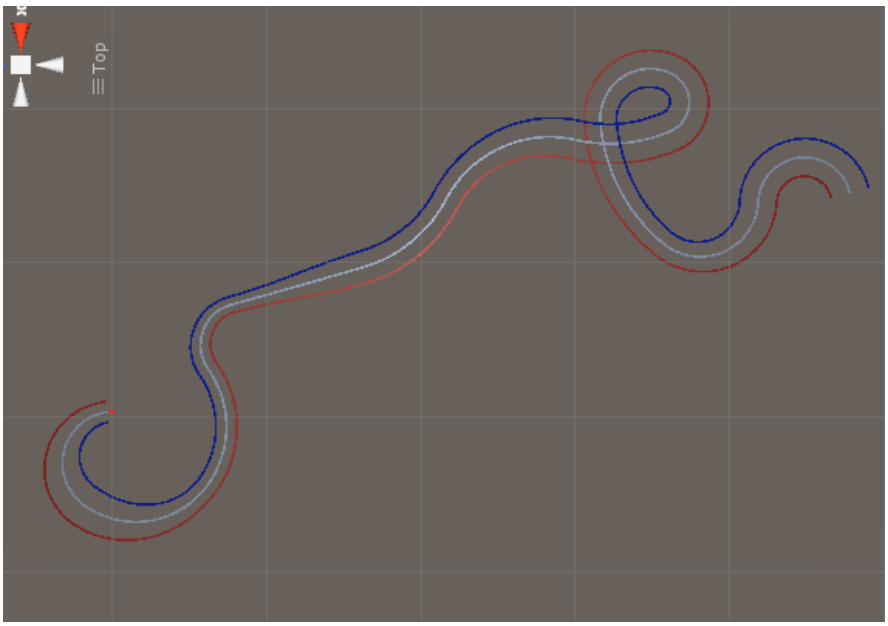
3.8 Koulutettujen mallien käyttö

Koulutetut mallit tallennetaan automaattisesti ML-Agents-komentokehotetyökalun ”results”-nimiseen kansioon. Jokaisen koulutuskerran data löytyy koulutuskerran nimen osoittamasta alikansiosta. Jotta koulutettua mallia voidaan käyttää Unityssa, on valitun koulutuskerran kansiosta kopioitava sitä vastaava ”.nn”-päätteinen tiedosto. [33.] Kyseinen tiedosto sisältää koulutetun mallin kokonaisuudessaan. Koulutettua mallia voidaan hyödyntää tuomalla sen tiedosto Unityyn ja asettamalla se halutun Agentin ”Model”-kenttään. [26.]

Agentit, joille on määritelty koulutettu malli, voivat seurata sen määräämiä ohjeita reaaliajassa koulutuksen ulkopuolella. [26.] Mikäli aiempaa mallia halutaan käyttää mahdollisessa jatkokoulutuksessa, voidaan koulutusta jatkaa aiemmasta pisteestä yksinkertaisesti lisäämällä koulutuksen käynnistävään ”ml-agents-learn”-komentoon parametri ”—initialize-from x”, jossa x korvataan aiemman koulutuskerran nimellä. [33.]

4 Pelidemo: vahvistusoppiminen

Koneoppimisagenttien kehitysprosessin havainnollistamiseksi kehitettiin rataa pitkin ajava autoagentti. Autoagentti on yksinkertainen ajoa ohjaava tekoäly, joka hyödyntää ajoradan pisteitä päättääkseen kääntymissuunnan ja ajonopeuden. Se on koulutettu Unityn ML-Agents-kirjaston tarjoamalla vahvistusoppimistyökaluilla. Autoagentin koulutuksen monipuolistamiseksi kehitettiin myös yhteensopiva ratageneraattori. Ratageneraattori auttaa varmistamaan, että agentti ei opi ajamaan vain yhtä rataa, vaan osaa reagoida monipuolisempiin tilanteisiin. Kuva 7 havainnollistaa ratageneraattorilla luotua satunnaista rataa.



Kuva 7. Ratageneraattorin luoma satunnainen rata.

4.1 Ympäristö

Agentin auto on toteutettu yksinkertaisena primitiiveistä koottuna mallina. Auton liikkeet on rajoitettu x- ja z-akseleille, eikä auton liikkeitä pyritä simuloimaan realistisesti, vaan tarkoitus on keskittyä ennemmin koneoppivan tekoälyn kouluttamiseen. Autolle on määritelty suurin mahdollinen vauhti, kiihtyvyys ja renkaiden kääntönopeus. Auton ohjaaminen tapahtuu kahden syötteen avulla: yksi kääntymiselle ja yksin vauhdin säätämiselle. Kääntösyöte säätää auton renkaiden kulmaa, mahdollistaen koko auton kääntämisen suhteessa auton nopeuteen ja akseliväliin.

Ajorata generoidaan proseduraalisesti jokaisen suorituskerran alussa. Ratageneraattori luo radan pienemmissä osuuksissa ja interpoloi osuuksien välit muodostaakseen yhtenäisen radan. Jokaiselle rataosuudelle arvotaan pituus, kääntymiskulma ja tien leveys, jonka avulla osuudet muodostetaan. Rata ei sisällä fyysisiä esteitä tai rajoja, joten osuuksien päällekkäisyys on sallittua. Ajoradan reitti syötetään autoagentille kulkujärjestyksessä, joten päällekkäisyydet eivät tässä tapauksessa myöskään haittaa agentin toimintaa. Koulutuksen aikana simuloidaan useita ajoagentteja samaan aikaan, jolloin jokaisella agentilla on myös oma proseduraalisesti generoitu ajoratansa.

4.2 Havainnot

Autoagentti havaitsee ajaessaan ajoratansa 10 seuraavaa välipistettä eli noin 10 autonmittaa eteenpäin. Ajoradan pisteiden sijainti havaitaan suhteellisina koordinaatteina suhteessa auton sijaintiin ja suuntaan. Suhteellisien koordinaattien käyttö auttaa autoagenttia oppimaan ja reagoimaan samalla tavalla riippumatta tien suunnasta koordinaatistossa. Lisäksi suhteellisten koordinaattien ansiosta auton suuntaa ei tarvitse erikseen havaita. Ajoradan pisteiden suhteellisen sijainnin lisäksi agentti havaitsee ainoastaan nykyisen vauhtinsa.

Havainnot normalisoidaan pyrkien noudattamaan ML-Agents-dokumentaation määräämiä käytänteitä. Ajoradan pisteiden suhteellinen sijainti normalisoidaan pisteiden arvioidun maksimietäisyyden perusteella. Maksimietäisyys arvioidaan kertomalla kerralla havaittujen ajoradan pisteiden maksimimäärä välipisteiden maksimietäisyydellä. Auton vauhti normalisoidaan yksinkertaisesti suhteessa auton suurimpaan mahdolliseen vauhtiin, jolloin havainto pysyy aina -1 ja 1 välillä.

4.3 Päätökset

Autoa ohjataan kahden päätösarvon avulla, jotka molemmat ovat liukuarvoja -1 ja 1 välillä. Toinen liukuarvoista säätää auton vauhtia, ja toinen määrää kuinka jyrkästi auton halutaan kääntyvän.

Kääntymispäätöksen liukuarvosta lasketaan sitä vastaava haluttu renkaiden kulma, ja renkaat kääntyvät kohti haluttua asentoa kääntönopeuden rajoittamana. Vastaavasti nopeutta ohjaavan päätöksen arvoa käytetään auton nopeuden säätämiseen autolle määritellyn kiihtyvyyden ja huippunopeuden rajoittamana.

4.4 Palkkiot

Agenttia palkitaan ajoradan suuntaan etenemisestä sekä ajoradan keskellä pysymisestä. Havaintojen keräyksen aikana auton nykyistä sijaintia käytetään pitämään kirjaa siitä, millä radan etapilla auto tällä hetkellä on. Nykyisen etapin sijaintia, suuntaa ja tien leveyttä hyödyntäen agentille lasketaan palkkio, jonka suuruus riippuu auton nopeudesta ja etäisyydestä tien reunoihin. Vastaavasti agenttia myös rangaistaan, mikäli se etenee tiellä väärään suuntaan. Mikäli agentti eksyy tieltä kokonaan, agentille lisätään suuri rangaistus ja suoritus lopetetaan automaattisesti. Tiellä pysymistä arvioidaan vertailemalla agentin sijaintia tieosuuden sijaintiin ja suuntaan. Jos agentin etäisyys tien sivusuunnassa on suurempi kuin tien leveys, agentin katsotaan ajaneen ulos radalta. Agentti saa myös jatkuvaa ajan myötä kertyvää automaattista rangaistusta, joka kannustaa agenttia suorittamaan tehtävänsä mahdollisimman nopeasti.

4.5 Ongelmat ja haasteet

Suurimpana haasteena autoagentin toteuttamisessa ilmeni havaintojen muodostaminen. Ensimmäisissä versioissa autoagentti havaitsi ajoradan etapit maailman koordinaateissa ja normalisoi sijainnit suhteessa koko radan kokoon. Vaikka maailman koordinaateista, auton omasta sijainnista sekä sunnasta onkin periaatteessa mahdollista päätellä ajoa ohjaavat päätökset, oli vanha asetelma liian hatara tuottamaan tehokkaita tuloksia ainakaan lyhyiden koulutusjaksojen aikana. Agentti oppi nopeasti kulkemaan eteenpäin, muttei vaikuttanut reagoivan ratojen muotoihin paljoa.

Myös palkkioiden määrittelyssä ilmeni haasteita autoagenttia suunnitellessa. Alun perin agentti sai palkkioita ainoastaan ajoradan etappien ohittamisesta, mutta se ei yksin vaikuttanut palkitsevan radalla pysymistä tarpeeksi. Myöhemmin palkkio annettiin edetyn matkan ja tien keskellä pysymisen perusteella, mutta koska palkkion suuruus kerrottiin tien keskellä pysymistä mittaavalla kertoimella, vastaava rangaistus väärään suuntaan ajamisesta tien reunoilla oli myös pienempi. Tämä johti odottamattomaan käytökseen, jossa agentti oppi ajamaan ympyrää tien halki. Agentti löysi siis yksinkertaisimman tavan maksimoida saavuttamansa palkkiot ajamalla ympyrää; jossa eteenpäin kuljetaan keskellä tietä; missä tien keskellä pysymisen tuottama palkkiokerroin on suurin; ja taaksepäin tien reunassa missä sama kerroin on rangaistukselle pienin. Kun odottamattoman käytöksen syy selvisi, palkkiokerroin lukittiin suurimpaan arvoonsa aina, kun agentti kulkee rataa väärään suuntaan. Kuvassa 8 on näyte lopullisesta palkkiokertoimen laskemisesta.

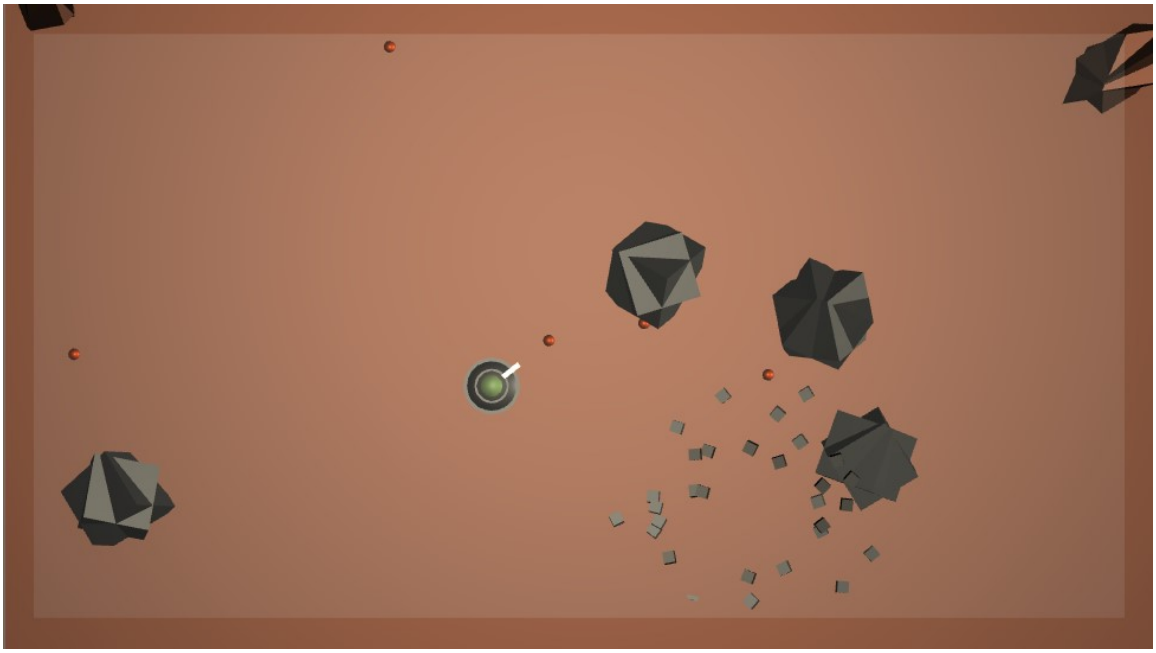
```
float relativeCenterDistance = carSideOffset.magnitude / (currentPoint.width/2f);
float relativeEdgeDistance = 1 - relativeCenterDistance;
float speedMultiplier = speed / maxSpeed;
float directionAngle = Vector3.Angle(transform.forward, currentPoint.direction);
float directionMultiplier = directionAngle > 90 ? -1 : 1;
if (directionMultiplier < 1)
{
    relativeEdgeDistance = Mathf.Max(1,relativeEdgeDistance);//max benalty when going wrong direction
}
Reward(Mathf.Pow(relativeEdgeDistance,4) * speedMultiplier * pointPassedReward * Time.fixedDeltaTime * directionMultiplier);
```

Kuva 8. Näyte palkkiokertoimen laskemisen koodista.

Autoagentin tehtäväksi asetettiin auton ohjaaminen satunnaisesti generoiduilla ajoradoilla. Palkkiosääntöjen ja havaintojen hienosäätämisen jälkeen agentin koulutus onnistui erinomaisesti. Autoagentin viimeisimmän version koulutuksessa agentti oppi jo noin viidessä minuutissa osaamistason, jonka kerryttämä keskimääräinen palkkio ylitti ihmispelaajan keräämän keskimääräisen palkkion määrän. Mahdollisena parannuskohteena autoagentin toteutuksessa voidaan pitää muun muassa sitä, ettei agentti kulje nopeinta mahdollista ajolinjaa, vaan pyrkii pysymään tien keskellä joka tilanteessa. Mikäli agentti haluttaisiin jatkokehittää ajamaan nopeinta ajolinjaa, olisi todennäköisimmin joko vähennettävä tien keskellä pysymisestä annetun palkkiokertoimen vaikutusta tai muutettava palkkiokerroin seuraamaan etäisyyttä optimaalisesta ajolinjasta tien keskikohdan sijaan. Toinen mahdollinen kehityskohde voisi olla agentin ajon tasaisuuden parantaminen. Agentti pysyy hyvin radallaan, mutta sen ohjaus on ihmisen ajotapaan verrattuna terävää, nykivää ja robottimaista. Agentin ajotapa ei aiheuta haittaa agentin nykyisen tehtävän suorittamisessa, mutta ihmismäisempi ajotapa voisi olla haluttu piirre esimerkiksi videopelin autoja ohjaavalle tekoälylle. Ajotavan tasaisuuden parantamista voisi todennäköisesti jatkokehittää joko kannustamalla tasaisempaa ajotyyliä uusilla vahvistusoppimispalkkioilla tai yksinkertaisimmillaan tasoittamalla agentilta saatuja ohjauspäätöksiä enemmän auton renkaita käännettäessä.

5 Pelidemo: imitaatio-oppiminen

Imitaatio-oppimisen havainnollistamiseksi suunniteltiin toinen demo käyttäen aiempaa monimutkaisempaa ja vaativampaa asetelmaa. Tällä kertaa koneoppimisagentin tehtävänä on ohjata avaruusalusta kaksiulotteisessa ympäristössä väistellen tai tuhoten kohti lentäviä meteoroideja. Pilottiagentin täytyy havaita ympäriltään läheisten meteoroidien sijainnit ja nopeudet sekä pyrkiä selviytymään ohjaamalla aluksensa pois meteoroidien lentoradalta ja tuhoamalla lähestyviä meteoroideja ampumalla niitä tykillään. Peli suunniteltiin vaativammaksi ja monimutkaisemmaksi kuin edellinen demo, jotta se toimisi myös esimerkkinä ympäristöstä, jossa agentin kouluttaminen yksin vahvistusoppimisen ja satunnaisten kokeilujen avulla voi olla vaikeaa. Kuva 9 esittää kuvankaappauksen avaruuspelin pelinäkymästä.



Kuva 9. Kuvankaappaus avaruuspelistä, jossa avaruusalus tuhoaa meteoroideja.

5.1 Ympäristö

Pelin ympäristönä toimii rajattu kaksiulotteinen alue avaruudessa. Agentti ohjaa pientä pyöreätä avaruusalusta, jonka liike on rajoitettu x- ja y-akseleille. Alukselle on määritelty kunto, suurin mahdollinen vauhti, kiihtyvyys, sekä aluksen tykin toimintaa määrittävät tulinopeus ja ammuksen vauhti. Aluksen nopeutta ohjataan kaksiulotteisella syötevektorilla, joka kuvaa aluksen haluttua

suuntaa ja vauhtia. Avaruusaluksen tykkiä taas ohjataan tykin suunnan määräävällä suunta-arvolla sekä totuusarvolla, joka määrittää, pyrkiikö tykki ampumaan. Avaruusaluksen tykin ammus tuhoaa ensimmäisen kohtaamansa meteoroidin. Vastaavasti avaruusalus tuhoutuu, mikäli yhteensä kolme meteoroidia osuu alukseen yhden pelisuorituksen aikana.

Meteoroideja lennätetään pelialueen halki satunnaistetuin nopeuksin, suunnin ja aikavälein. Peliin on suunniteltu ajan myötä nouseva vaikeustaso, joka kasvattaa meteoroidien lentonopeutta ja ilmestymistiheyttä pelin edetessä. Pelillä on ennalta määritellyt arvot helpoimmalle sekä vaikeimmalle mahdolliselle vaikeusasteelle, ja näiden arvojen välillä interpoloidaan kuluneen ajan perusteella vaikeusastetta hiljalleen kasvattaen. Peli on suunniteltu toimimaan omana irrallisena hierarkianaan niin, että yksittäisessä Unity-ympäristössä voi toimia samanaikaisesti useampia kopioita pelistä. Tämä mahdollistaa useamman agentin instanssin kouluttamisen samanaikaisesti, nopeuttaen agentin koulutusta.

5.2 Havainnot

Pilottiagentti havaitsee aluksensa ympäriltä näköpiirin sisällä lähimpien meteoroidien tai pelialueen reunojen etäisyyden aluksesta. Agentille on määritelty ympäristön havainnointiin käytettävien säteiden määrä ja pituus. Havaintosäteet jaetaan tasaisesti agentin ympärille niin, että ne osoittavat tasaisin välein ulospäin agentin avaruusaluksen keskikohdasta, havainnoiden aluksen ulkopuolella olevien esineiden etäisyyttä. Pilottiagentti havaitsee myös ympäriltä havaittujen esineiden nopeuden. Mikäli havaintosäde osuu meteoroidiin, havaituksi nopeudeksi asetetaan meteoroidin nopeus x- ja y-akseleilla. Muulloin havaittu nopeus jätetään nolla-arvoille.

Pilottiagentin tekemät havainnot normalisoidaan suositeltujen käytänteiden mukaisesti suhteuttamalla tehtyjen havaintojen arvot suurimpiin mahdollisiin havaintoarvoihin. Havaintosäteiden havaitsemat etäisyydet normalisoidaan 0:n ja 1:n välille suhteessa havaintosäteiden suurimpaan mahdolliseen pituuteen. Havaittujen meteoroidien nopeus normalisoidaan suhteessa suurimpaan sallittuun nopeuteen, jolloin meteoroidin kaksiulotteisen nopeusvektorin arvot sijoittuvat x- ja y-akseleilla -1:n ja 1:n välille.

5.3 Päätökset

Pilottiagentin tekemät päätökset ohjaavat avaruusaluksen liikettä, sekä avaruusaluksen tykin toimintaa. Agentin päätökset muodostuvat neljästä liukuarvosta -1:n ja 1:n välillä.

Ensimmäiset kaksi päätösarvoa kuvaavat avaruusaluksen kohdenopeutta x- ja y-akseleilla suhteessa aluksen huippunopeuteen. Agentin ohjaama avaruusalus pyrkii saavuttamaan agentin päättämän kohdenopeuden avaruusalukselle määritellyllä kiihtyvyydellä. Kolmas päätösarvo -1:n ja 1:n välillä on aluksen tykin suuntaa osoittava suhteutettu luku. Suhteutettu luku vastaa 0:n ja 360:n asteen välillä olevaa astelukua, jonka perusteella avaruusaluksen tykki kohdistetaan astelukua vastaavaan suuntaan kaksiulotteisessa tilassa. Neljäs ja viimeinen agentin tekemä päätös toimii tykin liipaisimena. Mikäli päätöksen arvo on suurempi kuin 0, tykki ampuu tulinopeuden salliessa. Muulloin tykki ei ammu, vaan odottaa tulikäskyä.

5.4 Palkkiot

Agentille määrätyn tehtävän monimutkaisuuden vuoksi ja imitaatio-oppimisen havainnollistamiseksi agentin palkkiot suunniteltiin mahdollisimman yksinkertaisiksi. Agentti saa ulkoista vahvistuspalkkiota vain selviytymänsä ajan perusteella. Jokaisella koulutusjakson väliaskeleella elossa olevaa agenttia palkitaan koulutusjakson suurimpaan mahdolliseen väliaskelten määrään suhteutetulla murtoluvulla niin, että kokonaisen koulutusjakson ajan selviytyvän agentin kerryttämä palkkio on yhteensä 1. Mikäli agentin avaruusalukseen osuu meteoroidi, agentti saa rangaistuksen, joka vastaa suuruudeltaan noin kahden sekunnin aikana kertyvää selviytymispalkkiota.

Selviytymispalkkion lisäksi agentti kerryttää automaattista sisäistä palkkiota imitaatio-oppimisalgoritmeilta. Agentin koulutuksessa on käytetty sekä BC (Behavioral Cloning) -menetelmää, että GAIL (Generative Adversarial Imitation Learning) -menetelmää. Imitaatio-oppimismenetelmiä hyödyntäessä agentti saa sisäistä palkkiota mallisuoritusten mukaisesta käytöksestä, kannustaen agenttia ottamaan mallia sille tarjotuista mallisuorituksista. Imitaatio-oppimisalgoritmien antamien palkkioiden on määrä auttaa agenttia selvittämään toimiva käytös nopeammin monimutkaisessa ja haastavassa ympäristössä, jossa pelkän selviytymisestä saadun palkkion perusteella oppiminen voisi olla haastavaa.

5.5 Mallisuoritukset

Imitaatio-oppimisen mahdollistamiseksi agentin tehtävästä nauhoitettiin pelaajan ohjaamia mallisuorituksia. Mallisuorituksissa pelaaja ohjaa avaruusalusta nuolinäppäimillä, tähtää hiiren osoittimella ja ampuu avaruusaluksen tykillä käyttäen vasenta hiiren painiketta. Pelaajan käyttämät syötteet muutetaan agentin ”Heuristic”-metodissa samaan muotoon kuin agentin tekemät päätökset, mahdollistaen agentin toimien ohjaamisen sekä havaintojen ja päätösten nauhoittamisen mallisuorituksiin.

Pelaaja pyrki mallisuoritusten aikana parhaaseen mahdolliseen tulokseen, mutta pelin nousevan vaikeustason vuoksi täydellisten suoritusten nauhoittaminen oli vaikeata. Pelaajan ohjaamien mallisuoritusten mediaanipalkkio oli noin 0,35, eli pelaaja selviytyi keskimäärin noin kolmasosan pisimmästä sallitusta peliajasta.

5.6 Ongelmat ja haasteet

Pilottiagentin koulutuksessa palkkioiden painoarvojen säätö osoittautui haastavaksi. Liian suuri painoarvo imitaatio-oppimisella johti käytökseen, joka muistutti haluttua käytöstä, muttei vaikuttanut oppivan koulutuksen edetessä enää pidemmälle. Liian suuri painoarvo vahvistusoppimisella taas johti hyvin satunnaisenomaiseen käytökseen, sillä pelkästä selviytymisestä annettava ulkoinen palkkio ei riittänyt kehittyneen käytöksen kouluttamiseksi ainakaan lyhyellä aikavälillä ja näin haastavassa tehtävässä.

Kouluttamisprosessia hidasti myös peliympäristön riippuvaisuus Unity-pelimoottorin fysiikoista. Yleensä koulutus tapahtuu nopeutetulla ajankululla, mutta fysiikkamoottoria hyödyntävissä tilanteissa liian suuri nopeutus voi johtaa epäluotettavaan simulaatioon. Nopeutuksen poistamiseksi pilottiagentin koulutussessiota käynnistäessä käytettiin lisäparametria ”--time-scale 1”, joka asettaa ajankulun takaisin pelimoottorin tavalliselle nopeudelle.

Pilottiagentin havainnointia päädyttiin muokkaamaan useaan otteeseen koulutuksen tuottaman käytöksen tason parantamiseksi. Havaintojen näköpiirin laajuutta kasvatettiin, sillä alun perin se ei riittänyt kattamaan kaikkia mallisuorituksissa pelaajan havaitsemia lähestyviä meteoroideja. Alun perin pilottiagentti ei myöskään havainnut meteoroidien nopeuksia tai pelialueen reunoja, mutta nämä koettiin oleelliseksi osaksi pelaajan tekemiä havaintoja, joten ne lisättiin myös agentin havaintoihin.

Koulutusprosessin reaaliaikainen seuraaminen Unity-editorissa osoittautui hyödylliseksi ohjelmointivirheiden ja muiden ongelmien löytämiseksi. Peli testattiin erikseen pelaajan ohjaamana, mutta koulutuksen aikana huomattiin vielä virheitä, joita ei pelaajan ohjatessa ollut vielä huomattu. Avaruusaluksen tykin tähtäys oli alun perin toteutettu tähtäämällä suoraan kaksiulotteisen päätösvektorin määräämää absoluuttista sijaintia kohti. Tämä johti tilanteeseen, jossa pelaajan testatessa peliä tykin tähtäys toimi oikein, mutta koneoppimisagenttien koulutuksessa tähtäys kohdistui usein joko lähelle peliympäristön nollapistettä tai kohti pelaajan mallisuorituksessa käytetyn peli-instanssin sijaintia. Virhe korjattiin muuttamalla tähtäystä kuvaava päätös yhdeksi liukuluvuksi, joka ohjaa tykin kulmaa Z-akselilla. Toinen ohjelmointivirhe huomattiin agentin liikuttua aivan pelialueen reunoilla. Tykin ammukset merkitään poistettavaksi niiden poistuessa pelialueelta, mutta agentin ampuessa ulospäin aivan pelialueen reunalta, ne eivät saapuneet pelialueelle alkuunkaan. Tämä johti tilanteeseen, jossa agentit saattoivat vahingoittaa ammuksillaan muiden samaan aikaan toimivien peli-instanssien meteoroideja tai avaruusaluksia, häiriten muiden agenttien koulutusprosessia.

Pilottiagentin tehtävä on ohjata avaruusalusta selviytyäkseen pelialueella mahdollisimman pitkään meteoroideja väistellen ja tuhoten. Kehityksessä ilmenneistä haasteista huolimatta pilottiagentti koulutettiin onnistuneesti suorittamaan tehtävänsä. Koulutettu pilottiagentti kykenee päättelemään havaitsemistaan meteoroideista uhkaavimmat yksilöt ja väistämään tai tuhoamaan avaruusalusta kohti kiitäviä meteoroideja. Pilottiagentin saavuttama osaamistaso on riittävä, mutta siinä on myös parantamisen varaa. Vaikka pilottiagentti selviääkin pelialueella suhteellisen kauan, ei agentti kuitenkaan saavuttanut mallisuorituksissa pelaajan esittämää osaamistasoa. Agentin keskimääräinen palkkio saavutti parhaimmillaan lähes puolet mallisuorituksien keskimääräisestä palkkiosta. Osaamistasoa voidaan todennäköisimmin jatkokehittää säätämällä agentin koulutusasetuksia sekä yksinkertaisesti kouluttamalla agenttia pidempään. Kokonaisuudessaan agentin kehitys onnistui kuitenkin hyvin, ja agentti oppi suorittamaan sille suunnitellun tehtävän.

6 Yhteenveto

Työssä tutkittiin koneoppimista, sen käyttöä pelialalla, ja erityisesti koneoppimisen hyödyntämistä Unity-pelimoottorissa. Työn tavoite oli kerätä tietoa koneoppimisesta ja sen hyödyntämisestä pelinkehityksessä. Työssä koottiin tietoa koneoppimisagenttien suunnittelun ja toteutuksen käytänteistä sekä yleispätevällä tasolla, että tarkemmin Unityn ML-Agents-ympäristöön perehtyen. Tutkimustavoitteille relevanttia tietoa kerättiin muun muassa aihealueen tutkimuksista, asiantuntijoiden esitelmistä ja Unityn ML-Agents-kirjaston dokumentaatiosta.

Tutkimuksessa kerätyn tiedon perusteella toteutettiin pelidemoja, jotka havainnollistavat koneoppivan tekoälyn kehitysprosessia Unity-pelimoottorissa. Agenttien toimintaperiaatteet sekä kehityksen aikana kohdatut haasteet dokumentoitiin kehitysprosessin havainnollistamiseksi, ja mahdollisten suunnitteluratkaisujen esittelemiseksi. Pelidemojen koneoppimisagenttien kehittämisessä erityisen tärkeäksi koettiin agentin havaintojen suhteutus ja normalisointi mahdollisimman yleistettävään muotoon. Vahvistusoppivalla agentilla erityistä huomiota kiinnitettiin myös agentin palkkiosääntöjen suunnitteluun halutun käytöksen kannustamiseksi. Imitaatio-oppivalla agentilla taas pyrittiin laadukkaiden mallisuorituksien käyttämiseen huomattavasti yksinkertaisempien vahvistuspalkkioiden tukemiseksi. Koneoppimisagenttien kehityksen jälkeen kunkin agentin toiminnallisuudesta arvioitiin hyvät ja huonot puolet, joiden perusteella agenteille suunniteltiin myös mahdollisia jatkokehitysmenetelmiä. Kokonaisuudessaan työ antaa perustavan kuvan koneoppimisagenttien suunnittelusta ja tarjoaa käytännönläheisiä huomioita agenttien kehityksestä Unityn ML-Agents-ympäristössä.

Lähteet

1 What is machine learning? Microsoft; Saatavilla: https://azure.microsoft.com/en-us/overview/what-is-machine-learning-platform/ Haettu: 20.9.2021

2 Hurwitz J, Kirsch D. Machine Learning For Dummies. John Wiley & Sons, Inc.; 2018. Saatavilla: https://www.ibm.com/downloads/cas/GB8ZMQZ3

3 Machine Learning. IBM Cloud Education; Saatavilla: https://www.ibm.com/cloud/learn/machine-learning Haettu: 20.9.2021

4 Marsalli M. McCulloch-Pitts Neurons. The Mind Project; Saatavilla: https://mind.ilstu.edu/curriculum/mcp_neurons/index.html Haettu: 20.9.2021

5 Marsalli M. McCulloch-Pitts Neurons: Introductory Level. Saatavilla: https://mind.ilstu.edu/curriculum/mcp_neurons/mcp_neuron1.html Haettu: 5.10.2021

6 Foote K. A Brief History of Machine Learning. Saatavilla: https://www.dataversity.net/a-brief-history-of-machine-learning/ Haettu: 20.9.2021

7 Widrow B, Youngsik Kim, Dookun Park. The Hebbian-LMS Learning Algorithm. IEEE; 2015. p. 37–53. Saatavilla: doi:10.1109/MCI.2015.2471216

8 Marr B. The 5 Biggest Technology Trends In 2022. Saatavilla: https://www.forbes.com/sites/bernardmarr/2021/09/27/the-5-biggest-technology-trends-in-2022/ Haettu: 5.10.2021

9 Zorotovich M, Donovan M. Current use cases for machine learning in retail and consumer goods. Microsoft; 2018. Saatavilla: https://azure.microsoft.com/en-us/blog/current-use-cases-for-machine-learning-in-retail-and-consumer-goods/?cdn=disable

10 Hauert S. Eight ways intelligent machines are already in your life. 2017; Saatavilla: https://www.bbc.com/news/uk-39657382 Haettu: 5.10.2021

11 Elements of AI. Helsingin Yliopisto; Saatavilla: https://course.elementsofai.com/fi/4/3 Haettu: 5.8.2020

12 Ahonen I. Koneoppimisen rajat ja mahdollisuudet. Saatavilla: https://www.vincit.fi/fi/koneoppimisen-rajat-ja-mahdollisuudet/ Haettu: 5.8.2020

13 Jaakkola T. Machine Learning. Saatavilla: https://dspace.mit.edu/bitstream/handle/1721.1/46320/6-867Fall-2002/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-867Machine-LearningFall2002/7AF7673F-EF49-4F65-8A67-B9B58860E8DD/0/lecture1.pdf Haettu: 5.8.2020

14 Sathya R, Abraham A. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. 2013;2(2). Saatavilla: doi:10.14569/IJARAI.2013.020206

15 Elements of AI. Helsingin Yliopisto; Saatavilla: https://course.elementsofai.com/fi/4/1 Haettu: 5.8.2020

16 Unsupervised Learning. IBM Cloud Education; Saatavilla: https://www.ibm.com/cloud/learn/unsupervised-learning Haettu: 5.8.2020

17 ML-Agents Toolkit Overview. Unity Technologies; Saatavilla: https://github.com/Unity-Technologies/ml-agents/blob/release_6_docs/docs/ML-Agents-Overview.md Haettu: 17.8.2020

18 Harmer J, Gisslen L, del Val J, Holst H, Bergdahl J, Olsson T, et al. Imitation Learning with Concurrent Actions in 3D Games. IEEE; 2018. p. 1–8. Saatavilla: doi:10.1109/CIG.2018.8490398

19 Teaching AI-agents to Play Battlefield. Electronic Arts; Saatavilla: https://www.ea.com/news/teaching-ai-agents-battlefield-1 Haettu: 17.8.2020

20 Robbins M. Neural Networks inSupreme Commander 2. Game Developers Conference, 2012 Saatavilla: http://twvideo01.ubm-us.net/o1/vault/gdc2012/slides/Summit_AI/Robbins_Michael_Off%20the%20Beaten.pdf Haettu: 17.8.2020

21 Matos X. Meet the computer that's learning to kill and the man who programmed the chaos. Engadget; Saatavilla: https://www.engadget.com/2014-06-06-meet-the-computer-thats-learning-to-kill-and-the-man-who-progra.html?guccounter=1 Haettu: 17.8.2020

22 Harvey F, Yurick M, Nowrouzezahrai D, Pal C. Robust motion in-betweening. ACM; 2020;39(4): 60:1-60:12. Saatavilla: doi:10.1145/3386569.3392480

23 Holden D, Kanoun O, Perepichka M, Popa T. Learned motion matching. ACM; 2020;39(4): 53:1-53:12. Saatavilla: doi:10.1145/3386569.3392440

24 McDonald J. Robocalypse Now: Using Deep Learning to Combat Cheating in Counter-Strike: Global Offensive. 2018. Saatavilla: https://www.gdcvault.com/play/1024994/Robocalypse-Now-Using-Deep-Learning Haettu: 5.10.2021

25 Prescott S. Valve wants to take a 'machine learning' approach to Counter-Strike anti-cheat. PC Gamer; 2017. Saatavilla: https://www.pcgamer.com/valve-wants-to-take-a-machine-learning-approach-to-counter-strike-anti-cheat/ Haettu: 5.10.2021

26 Unity Inference Engine. Unity Technologies; Saatavilla: https://github.com/Unity-Technologies/ml-agents/blob/release_6_docs/docs/Unity-Inference-Engine.md Haettu: 9.9.2020

27 Background: TensorFlow. Unity Technologies; Saatavilla: https://github.com/Unity-Technologies/ml-agents/blob/release_6_docs/docs/Background-TensorFlow.md Haettu: 9.9.2020

28 Nigretti, Alessia. Imitation Learning in Unity: The Workflow. Unity; 2018. Saatavilla: https://blog.unity.com/technology/imitation-learning-in-unity-the-workflow Haettu: 25.9.2020

29 Learning Environment Design Agents. Saatavilla: https://github.com/Unity-Technologies/ml-agents/blob/release_6_docs/docs/Learning-Environment-Design-Agents.md Haettu: 9.9.2020

30 Making a New Learning Environment. Unity Technologies; Saatavilla: https://github.com/Unity-Technologies/ml-agents/blob/release_6_docs/docs/Learning-Environment-Create-New.md Haettu: 10.10.2021

31 Using TensorBoard. Unity Technologies; Saatavilla: https://github.com/Unity-Technologies/ml-agents/blob/release_6_docs/docs/Using-Tensorboard.md Haettu: 9.9.2020

32 Training Configuration File. Unity Technologies; Saatavilla: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md Haettu: 5.10.2021

33 Training ML-Agents. Unity Technologies; Saatavilla: https://github.com/Unity-Technologies/ml-agents/blob/release_6_docs/docs/Training-ML-Agents.md Haettu: 9.9.2020