Ville Mäyrälä

Koneoppivan tekoälyn hyödyntäminen Unityn moninpelissä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

20.3.2025

Tiivistelmä

Tekijä:	Ville Mäyrälä
Otsikko:	Koneoppivan tekoälyn hyödyntäminen Unityn moninpelissä
Sivumäärä:	35 sivua
Aika:	20.3.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaaja:	Lehtori Antti Laiho

Insinöörityön tarkoituksena oli selvittää, pystyykö koneoppivan tekoälyn saamaan toimivaksi Unityn moninpeliin. Koneoppivan tekoälyn oli tarkoitus hyödyntää kolmea tasoa. Kehityksessä käytettiin Unityn ilmaisia laajennuksia ML-Agents sekä Photon Unity Network. ML-Agents mahdollisti koneoppivan tekoälyn tekemisen Unityn sisällä. Photon Unity Network puolestaan mahdollisti moninpeliosuuden tekemisen. Ilmainen versio Photonista mahdollistaa kahdenkymmenen pelaajan samanaikaisen pelaamisen ilman erillistä maksua.

Koneoppimista käytetään useisiin tarkoituksiin, esimerkiksi verkkosivuilla suosittelemaan sisältöä perustuen katseluhistoriaan tai itsestään ohjautuviin autoihin. Tällä hetkellä koneoppivaa tekoälyä koskevaa lakia ei ole paljon missään päin maailmaa ja ei ole tiedossa, onko lakeihin tulossa muutosta.

Sovelluksen kehitys aloitettiin aluksi tekemällä tekoäly, joka osasi löytää ja tuhota yhden pelaajaobjektin. Kehityksessä hyödynnettiin vahvistusoppimista sekä jäljitelmäoppimista. Kun tekoäly saatiin toimimaan, lisättiin tekoälylle ominaisuus tuhota useita pelaajaobjekteja. Tekoälyn toimittua yksinpelissä lisättiin peliin moninpeliominaisuus. Lopuksi saatiin tekoäly toimimaan moninpelissä, vaikkakin välillä tekoälyllä oli vaikeuksia löytää pelaajaobjekteja pelikentältä.

Sovelluksella on useita erilaisia jatkokehityskohteita. Niistä muutamat ovat pelin suoritustehoihin vaikuttavat parannukset sekä erilaiset pelin parannusmahdollisuudet. Suurin osa suoritustehoparannuksista vaatii tekoälyn koulutusta uudestaan. Sovelluksessa keskityttiin enemmän koodiin ja ominaisuuksiin mallien ja ulkonäön sijaan.

Avainsanat: koneoppiminen, tekoäly, vahvistusoppiminen, jäljitelmäoppiminen

---

Tämän opinnäytetyön alkuperä on tarkastettu Turnitin Originality Check -ohjelmalla.

Abstract

Author:	Ville Mäyrälä
Title:	Using machine learning AI In a multiplayer game in Unity
Number of pages:	35 pages
Date:	20 March 2025
Degree:	Bachelor of Engineering
Degree of Programme:	Information and Communication Technology
Professional Major:	Game Applications
Supervisor:	Antti Laiho, Senior Lecturer

The purpose of this final year project was to find out if it is possible to create a machine learning AI that works in a multiplayer game in Unity. The machine learning AI was using three dimensions. In the development, the free packages available in Unity were used, namely ML-Agent and Photon Unity Network. ML-Agents made it possible to create a machine learning AI in Unity. In turn, Photon Unity Network allows turning a game into a multiplayer game in Unity. The free version of the package allows 20 simultaneous players in a game without a separate payment.

Machine learning is used for many purposes. It is used, for example, on websites to recommend content based on the viewing history or in autonomous vehicles. At the moment, there is not much legislation in the world regarding the subject and currently it is uncertain if any changes are made to the laws.

The development was started with making an AI that could find and destroy one player object. Reinforcement learning and imitation learning were used in the development. When the AI was working, more functionality was added for the AI to destroy more player objects. When the AI was working correctly in a single-player game, a multiplayer was added to it. At the end, the AI was working in a multiplayer game. Sometimes the AI had problems in finding player objects in the play area.

The game has multiple areas for improvement, such as the performance of the game. Most of the performance improvements need AI to be trained again. The development focused on the code and game features rather than on models and the overall look of the game.

Keywords: machine learning, AI, imitation learning, reinforcement learning

Sisällys

1 Johdanto 1

2 Koneoppiva tekoäly 2

2.1 Koneoppivan tekoälyn historia 2
2.2 Koneoppivan tekoälyn käyttötarkoitukset 3
2.3 Koneoppivan tekoälyn tulevaisuus 4

3 Unity ja käytetyt laajennukset 5

3.1 Unity-pelimoottori 6
3.2 ML-Agents 7
3.3 Photon 12

4 Koneoppivan moninpelin kehitys 13

4.1 Sovelluksen tavoitteet 13
4.2 Sovelluksen työmenetelmät 14
4.3 Sovelluksen kehityksen eteneminen 15
4.4 Ongelmatilanteet sovelluksen kehityksessä 25

5 Koneoppivan moninpelin tuloksia 27

5.1 Jatkokehitys 27
5.2 Loppuanalyysi 29

6 Yhteenveto 31

Lähteet 33

1 Johdanto

Tässä Insinöörityössä ei keskitytä yleisesti tekoälyyn. Sen sijaan työssä keskitytään tarkemmin koneoppivaan tekoälyyn. Koneoppiva tekoäly toimii kouluttamalla itse itseään, kehittäjän antamien parametrien mukaisesti. Peleissä ei ole yleisesti käytetty koneoppivaa tekoälyä.

Insinöörityössä tutkitaan, toimiiko koneoppiva tekoäly pelitarkoituksessa ja miten koneoppivaa tekoälyä hyödynnetään moninpeleissä. Työssä käydään läpi koneoppivan tekoälyn historiaa. Insinöörityössä käydään läpi myös, kuinka käytetyt laajennukset saa ladattua sekä asennettua kehitysympäristöön. Työssä käydään yksityiskohtaisesti läpi, kuinka ML-Agents- ja Photon-laajennukset asennetaan Unityyn.

Insinööriprojektissa selvitetään, miten Unityssä onnistutaan tekemään koneoppiva tekoäly yksinpeliin ja moninpeliin sekä kuinka tekoälyä voidaan hyödyntää muissa projekteissa. On otettava huomioon, että käytettyihin laajennuksiin tulee todennäköisesti jatkuvasti päivityksiä ja korjauksia, minkä vuoksi projektin ratkaisut tai ongelmat eivät välttämättä ole myöhemmin olemassa.

Insinööriprojektin aiheeksi valikoitiin koneoppiva tekoäly kolmessa tasossa, sillä verkosta ei löytynyt esimerkkejä tai projekteja, joissa olisi hyödynnetty kolmea tasoa. Projektissa on tarkoituksena selvittää, toimisiko koneoppiva tekoäly nimenomaan käyttäen kolmea tasoa. Lisäksi verkosta ei löytynyt yhtään esimerkkiä, jossa olisi koneoppimista hyödynnetty moninpelissä. Tämän vuoksi projektiin lisättiin moninpeli.

Insinööriprojektissa käytetty koneoppiva tekoäly käyttää koulutuksessa satunnaisuutta, jonka vuoksi on mahdollista, ettei tekoälyn toimintaa pysty toistamaan. Tämän takia jokainen koulutuskerta vaihtelee hieman, vaikka mitään muutoksia ei tehtäisi. Lähtökohtaisesti kun tekoälyn koulutusta jatketaan tarpeeksi pitkään, pitäisi tulosten vakioitua koulutuskertojen välillä.

2 Koneoppiva tekoäly

2.1 Koneoppivan tekoälyn historia

Seuraavaksi esitellään koneoppivan tekoälyn merkityksellisiä keksintöjä. SRI eli Stanford Research Institute kehitti vuonna 1966 tekoälyn, joka ohjasi robottia nimeltä Shakey. Robotin komponentteihin kuului TV-kamera ja optinen etäisyyden mittari, joiden avulla Shakey pystyi tulkitsemaan ympäristöä. Komponentteihin kuului myös antennilinkki, jonka avulla robotin kanssa pystyttiin kommunikoimaan ja antamaan robotille ohjeita. Loput robotin komponentit olivat törmäystutkia, jotka olivat osana navigointijärjestelmää. Robotissa oli työntötanko, jonka avulla pystyttiin työntämään esineitä. Käyttäen mainittuja komponentteja Shakey pystyi löytämään tietyn pisteen seitsemän huoneen ympäristöstä, löytämään halutut laatikot ympäristöstä ja työntämään ne haluttuihin ryhmiin ohjeiden mukaisesti sekä varomaan erilaisia esteitä. Kaikki halutut toiminnot sille annettiin tekstimuodossa, ja se osasi itse päätellä, mitä sen tarvitsi tehdä. Shakey-robottiohjelman avulla on kehitetty esimerkiksi kaistanavustustekniikkaa autoihin ja navigointijärjestelmiä. Shakeyyn on viitattu usein itsestään-ohjautuvien autojen ja sotilaslennokkien esi-isänä. [1.]

Koneoppivan tekoälyn historiassa ei tapahtunut suuria muutoksia ennen kuin IBM kehitti Deep Blue -järjestelmän pelaamaan shakkia. Vuonna 1997 Deep Blue onnistui voittamaan hallitsevan maailmanmestarin Garry Kasparovin. Ottelu oli kuusi kierrosta, jonka Deep Blue voitti 3.5-2.5-tuloksella. Deep Blue koostui 32 prosessorista, ja se pystyi laskemaan 200 miljoonaa shakkisijoitusta sekunnissa. Voiton jälkeen Deep Blue siirrettiin Smithsonian museoon Washingtoniin. Järjestelmää käytettiin auttamaan shakkipelaajia harjoittelussa. Deep Blue inspiroi monia muita järjestelmiä, esimerkiksi taloudellisessa mallinnuksessa tai lääketieteessä. [2.] Vaikkakaan Deep Blueta ei suoraan luokitella tekoälyksi, on mahdollista verrata sen datan prosessointitapaa tekoälyn kanssa. Deep Blue käytti samoja algoritmeja, joita nykyään käytetään koneoppivien tekoälyjen kanssa, sillä algoritmeillä pystytään hyvin prosessoimaan suuria määriä dataa.

OpenAI lähetti tekoälyjä pelaamaan Dota 2 -turnaukseen ihmisiä vastaan vuonna 2018. Turnauksen pelaajat olivat ammattipelaajia. Tekoälyt voittivat kaikki muut ottelut paitsi finaalin. Finaalissa katsomo sai valita tekoälyjen käyttämät hahmot. Tämä asetti tekoälyt huomattavasti huonompaan tilanteeseen kuin aikaisemmin. Tekoälyt koulutettiin käyttäen hyödyksi koneoppivaa tekoälyä. Kehityksessä tekoälyt pelasivat toisiaan vastaan jatkuvasti. Ne pelasivat 180 vuoden edestä pelejä päivässä. [3.]

Vuonna 2022 OpenAI julkaisi ChatGPT-sovelluksen. ChatGPT on suurta kielimallia käyttävä koneoppiva tekoäly, jonka tarkoituksena on luoda ihmistyylisiä tekstivastauksia annettuihin kehotteisiin. Se on maksuton, toisin kuin monet muut tekoälyä käyttävät sisältöä tuottavat sovellukset. On otettava huomioon, että ChatGPT saattaa välillä antaa väärää tietoa, tuottaa vahingoittavia ohjeita tai puolueellista sisältöä. Tämä johtuu koulutuksessa käytetystä datasta. [4.] Tässä insinöörityössä ei keskitytä generatiiviseen tekoälyyn, vaan tarkemmin vahvistusoppimiseen ja menetelmiin, jotka hyödyntävät sitä.

2.2 Koneoppivan tekoälyn käyttötarkoitukset

Koneoppivaa tekoälyä käytetään moniin tarkoituksiin, joista tunnetuimmat ovat monien nettisivujen esittämät suositukset esimerkiksi Youtubessa ja Netflixissä. Nämä sivustot suosittelevat sisältöä perustuen käyttäjän katseluhistoriaan. Vaikeinta tekoälyn kehityksessä on se, miten hyödyntää kerättyä dataa, jotta saataisiin halutut tulokset. [5.]

Vahvistumisoppimista käytetään eniten peleissä. Sillä voidaan myös ratkaista ongelmia, jotka voidaan pelillistää, esimerkiksi pörssimarkkinoiden tai mainonnan optimointia. Robotti-imurit ovat hyvä esimerkki koneoppivasta tekoälystä. Ensin ne kiertävät tilan läpi ja tallentavat sen muistiin. Robotti-imurit hyödyntävät tätä tietoa jatkossa sekä tekevät tarvittavia muutoksia, jos uusia esteitä löytyy. [5.]

Koneoppimista hyödynnetään itsestään ohjautuvissa autoissa. Niiden liikenteen tulkinta on parantunut huomattavasti koneoppimisen lisäyksellä. Koneoppimista on käytetty hyödyksi tulkitsemaan erilaisten sensorien tuottamaa dataa. Vahvistusoppimista voidaan käyttää arvioimaan erilaisten sensoreiden epäonnistumista. On oletettavissa, että itsestään ohjautuvista autoista tulee parempia kuljettajia kuin ihmisistä, sillä ne näkevät koko ajan ympärilleen. On kuitenkin haasteellista, kun koneoppimisella ei voida välttämättä tuottaa aina samoja tuloksia. Riskinä itsestään ajaville autoille on erilaiset olosuhteet, jotka vaikuttavat niiden sensoreihin. Tällä hetkellä itsestään ohjautuvilla autoilta puuttuu myös oikeaa dataa liikenteestä, jonka vuoksi niihin joudutaan lisäämään synteettisesti luotua dataa. [6.]

Pelaamisessa on pitkään käytetty hyödyksi koneoppivaa tekoälyä. Sillä on selvitetty useisiin eri peleihin, mikä on teoreettisesti nopein aika läpäistä jokin taso tai jopa koko peli. Yleensä koneoppivaa tekoälyä ei ole rakennettu pelin sisään, vaan erillisellä sovelluksella peliin ohjataan tekoälyn näppäinpainallukset. On olemassa verkkosivuja, joissa eri ihmiset kilpailevat siitä, kuka onnistuu kehittämään tekoälyn, joka osaa läpäistä pelin nopeinten.

2.3 Koneoppivan tekoälyn tulevaisuus

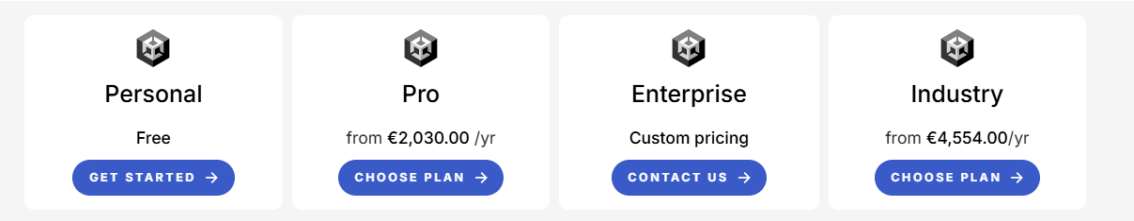
Tällä hetkellä kansainvälinen laki rajoittaa tekoälyn käyttöä todella vähän. Yritykset, jotka käyttävät tekoälyä tuotteissaan, eivät välttämättä kerro kuluttajille tekoälystä tai sen käyttämästä datasta. Tekoäly kehittyy nyt niin nopeasti, ettei lainsäädäntö pysy kehityksessä mukana. Nopeasti tehdyistä lakimuutoksista voi olla enemmän haittaa kuin hyötyä. [5.]

Koneoppiva tekoäly voi mahdollisesti olla kriittisenä osana edistämässä lääketeollisuutta, kehittämässä tarkempia malleja, käyttäen hyödyksi suuria määriä tällä hetkellä olevaa dataa. Koneoppivan tekoälyn avulla voi olla mahdollista ennustaa henkilön reaktiota tiettyihin lääkkeisiin tai lääkkeiden aineenvaihdunnallista roolia. Tekoälyn avulla voidaan helpottaa hoitojen muokkaamista sopivaksi jokaiselle henkilölle. [7.]

Tällä hetkellä monet yritykset haluavat lisätä tekoälyä tuotteisiinsa tai palveluihinsa. Tämän näkee helposti esimerkiksi eri verkkosivuilta löytyvästä keskustelurobotista. Vaikka tällä hetkellä tekoäly on laajassa käytössä yrityksillä, jää nähtäväksi, miten sille jatkossa käy.

3 Unity ja käytetyt laajennukset

Unityn pelimoottoriin on useita erilaisia hintoja vaihdellen käyttötarkoituksesta ja käyttäjästä. Esimerkiksi ilmaisessa versiossa on vähemmän ominaisuuksia, ja se mahdollistaa pelin tulot kahteensataantuhanteen dollariin saakka, jonka jälkeen pitää vaihtaa johonkin maksulliseen versioon kuten kuvasta 1 näkyy. [8.]



Kuva 1. Unityn tarjoama hinnasto eri paketeille. [8]

Unityn pelimoottori on suosittu pienten tai itsenäisten pelikehittäjien keskuudessa. Unity oli tekemässä muutosta maksumalliinsa vuonna 2024. Uusi malli olisi perustunut käyttöaikamaksuun tällä hetkellä voimassa olevan asiakaskohtaisen tilausmallin sijaan. Tämä muutos peruttiin nopeasti suuren negatiivisen palautteen takia. [9.]

Unityyn pystyy lataamaan paljon erilaisia paketteja tai laajennuksia, joista osa on ilmaisia kuten projektissa käytetyt Photon ja ML-Agents. Unityssä on sisäänrakennettu kauppapaikka, josta voi ladata joko ilmaisia tai maksullisia voimavaroja (assets). Nämä voivat olla esimerkiksi ohjelmakoodia, objekteja, ääniä ja efektejä.

3.1 Unity-pelimoottori

Unityn pelimoottorilla pystyy tekemään 2D- tai 3D-pelejä, joihin Unityn sisältä löytyvät omat työkalut ja komponentit. Unity tukee pelikehitystä useilla alustoilla, joita ovat mobiili-, konsoli-, työpöytä- ja verkkoalustat. [10.]

Unityn sisältä löytyy monenlaisia toimintoja, kuten animaatioiden tekeminen, välianimaatioiden (cutscenes) upottaminen, erilaisten äänilähteiden ja efektien tukeminen sekä erilaisten varjostimien (sharders) tekemisen. Unity tukee myös suoraan erilaisia syötteenantolaitteita (input), kuten hiiriä, näppäimistöjä, peliohjaimia sekä kosketusnäyttöjä. [10.]

Unity käyttää skenejä (scenes), joissa pelin eri ominaisuudet ovat. Peli voi koostua yhdestä skenestä, mutta usein pelit koostuvat monesta skenestä. Jokaisella skenellä voi olla oma ympäristö, hahmot, objektit ja jopa käyttöliittymä. Usein peleissä jokainen kartta on oma skenensä. [11.] Useiden skenejen käytössä on hyvä muistaa lisätä skenet pelin rakennukseen (build), sillä ne eivät automaattisesti sinne lisäänny. Skenejen käyttö auttaa helposti keventämään peliä osiin.

Unityn pelimoottorin tärkein ominaisuus on peliobjektit (gameObjects). Jokainen objekti pelissä on peliobjekti, hahmoista efekteihin. Itsessään peliobjektit eivät tee mitään. Niille tulee antaa erilaisia ominaisuuksia, minkä avulle ne toimivat halutulla tavalla. Peliobjektit pitävät sisällään komponentteja. Komponentit voivat olla esimerkiksi ohjelmakoodia, fysiikkaa, materiaaleja tai efektejä. Jokaisella peliobjektilla on transform-komponentti, joka sisältää objektin sijainnin skenessä, kääntymisasteen sekä mittakaavan. Transform-komponenttia ei ole mahdollista poistaa peliobjekteista. [12.]

Pelimoottori on riippuvainen alan standardoiduista käsitteistä, kuten .NET ja C#. Pelimoottori mahdollistaa kehittäjän käyttää erilaisia työkaluja, kuten Visual Studio, jonka pystyy lataamaan suoraan Unityn kanssa. Koodausmahdollisuudet ovat monipuoliset Unityssä. Nämä voivat olla esimerkiksi objektin liikuttaminen tai suurien monimutkaisten ympäristöjen kontrollointi. Oletuksena Unityssä on käytössä oliopohjainen-ohjelmointi, mutta Unityyn on myös lisätty Data-Oriented

Technology Stack, jota kutsutaan myös nimellä DOTS. [10.] Unityn ongelmana on pitkään ollut pelien suoritustehovaatimukset, joita parantamaan on kehitetty DOTS.

DOTS tuo kolme eri kokonaisuutta Unityyn, jotka ovat Entity Component System, mikä tunnetaan myös nimellä ECS, Burst Compiler ja C# Job System. Kokonaisuuksia voi käyttää erikseen, mutta on kuitenkin suositeltavaa käyttää kaikkia kolmea samanaikaisesti. ECS on dataorientoitu viitekehys, joka on yhteensopiva Unityn peliobjektien (gameObjects) kanssa. ECS mahdollistaa paremman suoritustehon ja suuremman kontrollin kokeneille kehittäjille kuin ilman ECS käyttöä. ECS auttaa kehittämään monimutkaisempia, dynaamisemmalla ympäristöllä olevia ja suoritustehoja vaativammille laitteille olevia pelejä kuin ilman ECS-laajennusta. ECS vaatii vähintään Unityn version 2022.3 tai paremman toimiakseen. Sen pystyy lataamaan Unityn pakettimanagerista. [13.]

Burst Compiler kääntää IL/.NET-tavukoodia natiiviksi koodiksi. Burst Compiler mahdollistaa koodin tarkan optimoinnin suorituskyvyn kriittisellä tasolla. C# Job System antaa kehittäjille mahdollisuuden hyväksikäyttää moniytimistä alustaa, jolloin koodia pystytään ajamaan rinnakkain nopeasti ja turvallisesti. Molemmat kokonaisuudet pystytään lataamaan käyttäen Unityn pakettimanageria. [14; 15.]

3.2 ML-Agents

ML-Agents on maksuton laajennus Unityyn. Sen pystyy lataamaan Unityn pakettimanagerista tai ML-Agentsin GitHubista. Unityn paketin lisäksi ML-Agents vaatii Pythonin latauksen laitteelle ja virtuaaliympäristön luomisen tekoälyn koulutusta varten.

ML-Agents mahdollistaa pelien ja simulaatioiden suorittamisen tekoälyjen kouluttamiseen Unityssä. Koulutus voidaan suorittaa vahvistusoppimisena, jäljitelmäoppimisena, neuroevoluutiona tai muilla metodeilla. Koulutettuja tekoälyjä voidaan hyödyntää useisiin käyttötarkoituksiin, esimerkiksi kontrolloimaan NPC

hahmoja tai automaatiotestaamaan pelin rakennettuja pelejä ja arvioimaan erilaisia suunnittelupäätöksiä ennen pelin julkaisua. [16.]

ML-Agents laajennus mahdollistaa minkä tahansa Unityn skenen muuttamisen koulutusympäristöksi. Paketti sisältää neljä tärkeintä kokonaisuutta. Ne ovat agenttien määrittäminen, käyttäytymisen määrittäminen, demonstraation nauhoittaminen ja koulutettujen käyttäytymisien upottaminen skeneihin. [16.]

Agentit ovat entiteettejä (entities), jotka luovat havaintoja sensoreilla, tekevät toimintoja ja saavat palkkiota ympäristöstä. Käyttäytyminen (behavior) määrittelee, miten agentin kuuluu toimia. Monta agenttia pystyy jakamaan saman käyttäytymisen, ja skenessä pystyy olemaan monta käyttäytymistä samanaikaisesti. Demonstraation nauhoittaminen näyttää agentin toimintaa Unityn editorilla. Demonstraatioita voi hyödyntää agentin käyttäytymisen koulutuksessa. Koulutettujen käyttäytymisten upottaminen mahdollistaa agentin oppimisen ja käyttäytymisen vaihtelua toiminnan välillä Unityn-moottorin sisällä. [16.]

ML-Agents on mahdollista ajaa headless-tilassa, jossa se ei tuota Unityn editorissa visuaalisia havaintoja agenteista. Tällä hetkellä koulutuksessa pelin nopeutta voidaan kasvattaa satakertaiseksi todelliseen aikaan verrattuna. Nopeutuksessa on otettava huomioon ML-Agentsin hyödyntämä työkalu Academy. Se huolehtii, että peli pysyy synkronisoituna koulutuksen ajan. Academy liikkuu käyttäen Unityn metodia FixedUpdate eikä Update, joka saattaa aiheuttaa synkronisointiongelmia, riippuen käytettävästä toteutuksesta. [16.]

Pythonin asentaminen

ML-Agentsin koulutusympäristö vaatii Pythonin asennuksen laitteelle, jossa tekoälyä koulutetaan. Python-version pitää olla 3.10.0 tai suurempi versio, mutta vaatimus voi vaihdella riippuen siitä, miten ML-Agents-laajennusta päivitetään. Pythonin eri versioita pystyy lataamaan heidän omilta nettisivuiltaan. Kun Pythonia asennetaan laitteelle, kannattaa Python samalla lisätä ympäristön muuttujiin asennusohjelman sisältä löytyvällä painikkeella Add Python to environment variables. Jos Pythonia ei lisätä asennusvaiheessa, pitää se itse käydä

lisäämässä myöhemmin ympäristön muuttujiin, jotta Pythonia pystytään käyttämään laitteella. [17.]

Virtuaaliympäristön luominen

Virtuaaliympäristön pystyy tekemään komennolla

py -m venv testenv

Tämä laitetaan suoraan komentoriville. Py kutsuu Pythonia. Py voi korvata myös sanalla python riippuen laitteesta. Tarkenne -m ilmoittaa Pythonille ohjelmakoodin olevan moduuli tai paketti. Venv luo virtuaaliympäristön nimeltä testenv. Testiympäristön saa aktiiviseksi ajamalla komennon

venv\Scripts\activate

Tässä venv on testiympäristölle asetettu nimi, ja se sisältää kansion Scripts, josta löytyy activate-ohjelmakoodi. Komennon jälkeen virtuaaliympäristö on päällä. Sen huomaa helposti komentoriviltä, koska se alkaa (venv) nimellä, kuten kuvasta 2 näkyy pelissä käytetty sample-env virtuaaliympäristö. [17.]

```
(sample-env) c:\>
```

Kuva 2. Komentokehotteessa oleva aktiivinen virtuaaliympäristö sample-env.

Ympäristön ollessa aktiivisena on suositeltavaa päivittää Pythonin oma pakettien lataustyökalu pip. Tätä käytetään lataamaan tarvittavat lisäykset, jotta koulutus koneoppivalle tekoälylle toimisi. Päivittäminen toimii komennolla

py -m pip install --upgrade pip

Komento lataa tarvittavat päivitykset, jos niitä löytyy. ML-Agents-laajennuksen voi sen jälkeen ladata virtuaaliympäristöön käyttäen komentoa

pip install mlagents

Tämän jälkeen täytyy ladata myös pytorch-laajennus, jota ML-Agents hyödyntää koneoppimisessa. Se onnistuu komennolla

pip install torch torchvision torchaudio

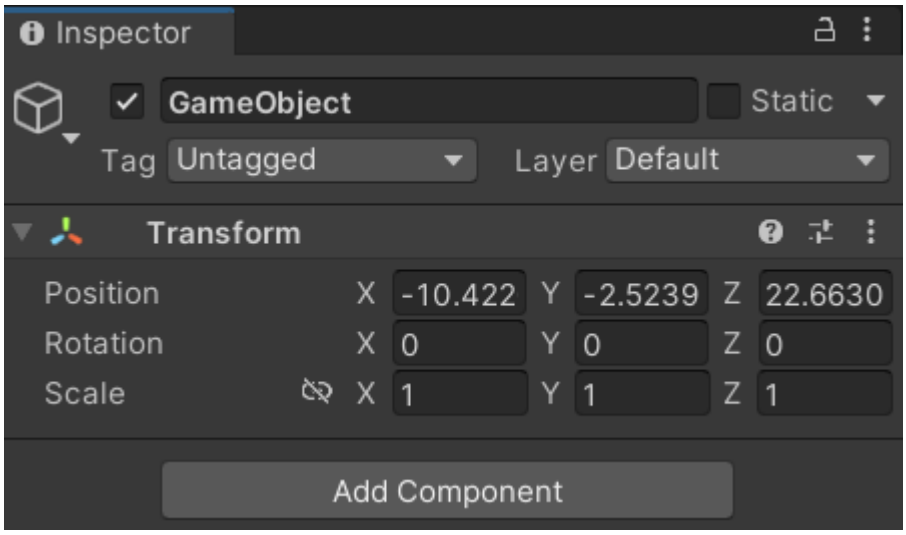
Torchvision antaa koneoppivalle tekoälylle mahdollisuuden hyödyntää näkemistä ja torchaudio mahdollistaa kuulemisen. Näiden avulla koneoppiva tekoäly pystyy tekemään päätöksiä ja toimintoja pelissä. Seuraavaksi testataan, onko ML-Agents asennettu ympäristöön oikein, komennolla

mlagents-learn –help

Jos komento ei ilmoita virheestä, on laajennukset ladattu oikein ja ML-Agents on käytettävissä kouluttamista varten. Jos virheitä ilmenee, on puuttuvat tai virheelliset paketit ladattava komentorivin ohjeiden mukaisesti. [17.]

Tarvittavien komponenttien ja koodin lisäys tekoäly objektille

Unityssä objektille, johon tahdotaan tekoäly lisätä, tarvitsee lisätä Behaviour Parameter-ohjelmakoodi. Se tulee ML-Agents-laajennuksen mukana. Lisäksi objektiin tarvitsee lisätä Decision Requester-ohjelmakoodi, jotta tekoäly tekee päätöksiä automaattisesti tietyn ajan välein. Vakiona päätöksien väli on viisi askelta. Jos ohjelmakoodia ei lisätä, täytyy päätökset manuaalisesti kutsua koodissa käyttäen RequestDecision-funktiota. [18.] Ne saa helposti lisättyä objektille käyttäen Unityn inspector-ikkunaa ja painamalla Add Component -nappia, kuten kuvasta 3 näkyy.



Kuva 3. Unityn inspector-ikkuna, jossa näkyy Add Component -nappi.

Tekoälyn koulutuksessa on tärkeää aina aloittaa koulutustilanteet samassa tilanteessa pienellä vaihtelulla, esimerkiksi siirtää tekoälyn aloituspaikkaa hieman. Tällöin tekoäly ei totu tekemään samoja liikkeitä, vaan ymmärtää toimintaansa, vaikkakin ympäristö vaihtuu. Tekoälyn koulutuksessa sille on määritetty suurin määrä askeleita, ennen kuin se aloittaa tilanteen uudestaan. Tekoäly kutsuu funktiota OnEpisodeBegin. Funktioon on hyvä laittaa tarvittavat koodit, jotta tilanteet alkavat halutulla tavalla. Kun tekoäly pääsee voittotilanteeseen, eli lopulliseen tilanteeseen, jota ollaan havittelemassa, täytyy kutsua funktiota EndEpisode. Tämä lopettaa kyseisen koulutustilanteen etuajassa ja aloittaa uuden tarvittaessa.

Vahvistusoppiminen sekä jäljitelmäoppiminen

Vahvistusoppimisella tarkoitetaan tekoälyn oppimista, jossa sille annetaan negatiivinen palkinto toiminnasta, joka ei ole haluttua. Mitä huonompi toiminta sitä suurempi pitää palkinnon määrän olla. Esimerkiksi jos tekoäly putoaa kokonaan pelikartalta, voidaan sille antaa -1. Kun taas jos tekoäly kääntyy sokkelossa väärään suuntaan, voidaan sille antaa -0.1. On myös tärkeää antaa tekoälylle positiivinen palkinto halutusta toiminnasta. Yleensä voittotilanteessa tekoälylle annetaan reilusti positiivista palkintoa, esimerkiksi verraten aikaisempiin

arvoihin, sille voidaan antaa 1. Vahvistumisoppimisessa voidaan pakinnon antamisen lisäksi asettaa palkinnon arvo suoraan, mutta tätä ei yleensä suositella käyttämään, sillä se ei mahdollista yhtä hyvää optimointia tekoälylle.

Jäljitelmäoppisella tarkoitetaan sellaista tekoälyn oppimistapaa, jossa tekoälylle annetaan valmiiksi tehty demo. Demo sisältää toimintaa, jota toivotaan tekoälyltä. Mieluusti demo on viety useaan otteeseen tilanteeseen, jossa tekoäly voittaa pelin. Tämä mahdollistaa tehokkaan optimoinnin, sillä usein koulutustilanteiden välillä on pieniä eroavaisuuksia. Jäljitelmäoppimista käytetään usein vahvistusoppimisen kanssa yhdessä. Tämän toteutuksen avulla koulutuskerran aikana tehdyt liikkeet eivät välttämättä ole tekoälyn itse tekemiä, vaan ne tulevat demosta. Tekoäly koittaa tällöin päätellä kumman liikkeitä ne ovat ja tehdä muutoksia sen mukaan.

Hyperparametrien käyttäminen

Tekoälylle tulee antaa käyttöön hyperparametrejä sisältävä tiedosto, jolla pystytään muokkaamaan erilaisia ominaisuuksia koulutuksesta. Tiedosto on yaml-tiedostopäätteinen. Hyperparametrejä ei ole pakko muokata tekoälyn koulutuksessa, mutta se on erittäin suositeltavaa. Hyperparametreilla voidaan muokata esimerkiksi sitä, kuinka monta kokemusta tehdään ennen kuin kokemuksia otetaan huomioon opetuksessa. Kokemukset ovat havaintoja, toimintoja ja palkintoja. Hyperparametreissä määritellään myös maksimimäärä askeleita, ennen kuin koulutus loppuu, ja normalisoidaanko koulutuksen tuloksia eli poistetaanko ääripäät tuloksista. Hyperparametrien avulla voidaan yhdistää eri koulutustyyppejä, esimerkiksi käytetäänkö vahvistumisoppimista 40 %:a koulutuskerroista ja 60 %:a jäljitelmäoppimista. [19.]

3.3 Photon

Photonista käytetään nimitystä PUN, joka on lyhenne sanoista Photon Unity Network. Photon on paketti Unityssä moninpeleille. Paketti mahdollistaa joustavan liittymisen huoneisiin, joissa objektit voidaan synkronisoida verkon

välityksellä. Yhteys pelaajien välillä on nopea ja luotettava. Yhteys liikkuu Photonin omien palvelimien läpi, minkä takia pelaajien ei tarvitse liittyä keskenään. [20.] Photon-paketin saa ladattua Unityn kauppapaikasta.

Photonilla on kaksi erilaista pakettia saatavilla PUN 2 Free ja PUN 2 Plus. PUN 2 Free on ilmainen, ja se sisältää useita demoja, valmiita ohjelmakoodeja, dokumentaatioita ja tukee enintään kahtakymmentä pelaajaa samanaikaisesti. Pun 2 Plus on maksullinen ja sisältää kaikki samat ominaisuudet kuin ilmainen versio, mutta se pystyy tukemaan sataa pelaajaa samanaikaisesti. [20.]

Objekteja voi instansioida (instantiate) peliin moninpeliobjekteiksi lisäämällä niihin Photonview-komponentin Unityssä. Photonview identifioi objektin ja sen omistajan. Pelaaja, joka omistaa objektin, päivittää kyseisen objektin muille pelaajille. Tyypillisesti objektit instansioidaan käyttäen PhotonNetwork.Instantiate-funktiota. Pelaajat voivat kutsua Photonin omia Remote Procedure Calls -funktioita. Nämä funktiot ovat erillisissä moninpeliobjekteissa, jolloin funktion sisältö tapahtuu samanaikaisesti kaikille pelaajille. Funktioita kutsutaan käyttäen photonview.RPC-komentoa. On myös mahdollista kutsua Photonin avulla omia tapahtumia käyttäen PhotonNetwork.RaiseEvent-funktiota. [20.]

4 Koneoppivan moninpelin kehitys

4.1 Sovelluksen tavoitteet

Sovelluksen tavoitteena on selvittää, onnistuuko koneoppivan tekoälyn tykkitorni seuraamaan ja tuhoamaan yhden tai useamman pelaajan pelissä. Kaikki verkosta löytyvät esimerkit Unityn koneoppivista tekoälyistä on tehty kahdessa tasossa. Sovelluksessa on kuitenkin tarkoituksena hyödyntää kolmea tasoa.

Sovellus koostuu aluksesta, jonka päällä on tykkitorni. Tykkitorni pystyy pyörimään sivuille, ylös ja alas sekä ampumaan. Tykkitorni on koneoppivalla tekoälyllä toimiva. Pelissä on myös aluksia, joita pelaajat voivat liikuttaa ja ohjata vapaasti.

4.2 Sovelluksen työmenetelmät

Sovelluksen kehityksessä on käytetty GitHub-versionhallintatyökalua. Työkalua käyttämällä on mahdollistettu sovelluksen vieminen ja päivittäminen helposti usean laitteen välillä. Pääasiassa työkalua käytettiin varmistamaan, että laitevian tapahtuessa sovellus ei tuhoutuisi ja että mahdollisten virheiden sattuessa pystyttäisiin palaamaan virhettä edeltävään tilaan.

Sovelluksen kehityksessä käytettiin Unityn pelimoottoria, sillä se on kehittäjälle tutuin. Unityn käyttäminen mahdollisti myös Phonton- ja ML-Agents-laajennusten hyödyntämisen kehityksessä, nämä olivat tuttuja kehittäjälle. Käytetty Unityn versio projektissa oli 2021.2.7f1, jonka vuoksi projektissa ei voitu käyttää hyödyksi Unityn uusia DOTS-laajennuksia parantamaan sovelluksen suorituste-hoja.

Unityn sisällä olevan laajennuksen ML-Agents pystyy lataamaan suoraan Unityn kautta tai ML-Agentsin GitHubista. ML-Agents vaatii toimiakseen Pythonin lataamisen ja virtuaalisen ympäristön luomisen laitteelle, jolla koneoppimisen koulutusta tehdään.

Moninpelin tekemiseen sovelluksessa hyödynnettiin Unityn kaupasta löytyvää ilmaista laajennusta Photon PUN 2. Laajennuksen käyttäminen edellytti kirjautumista Photonin nettisivulle ja käyttäjätilin luomista sinne. Kun käyttäjätili on saatu tehtyä nettisivulle, saa nettisivulta koodin, joka tulee lisätä Unityyn. Jos Photonin ikkuna Pun Wizard ei aukea automaattisesti Unityssä, on ikkuna mahdollista avata käyttäen Unityn Window-painiketta, kuten kuvasta 4 näkyy. Laajennus mahdollistaa moninpelin tekemisen, johon kaksikymmentä pelaajaa pystyy liittymään samanaikaisesti ilman erillistä maksua.

Kuva 4. Unityn Window-ikkunan sisältämä sijainti Pun Wizardille.

4.3 Sovelluksen kehityksen eteneminen

Sovelluksen kehitys aloitettiin tutustumalla ML-Agents-laajennukseen ja laittamalla projekti GitHubiin. Tämän jälkeen sovellukseen tehtiin tarvittavat objektit eli pelaajalle alus, ammus ja tekoälyn tykkitorni. Objektit ovat alustavia ja ne pystytään vaihtamaan helposti myöhemmin toisennäköisiksi, mutta niiden käyttäminen helpottaa kehitystä ja hahmottamista. Sovelluksen tarkoituksena on keskittyä koodiin ja toiminnallisuuksiin objektien ulkonäön sijaan.

Kehitys yhdellä pelaajaobjektilla

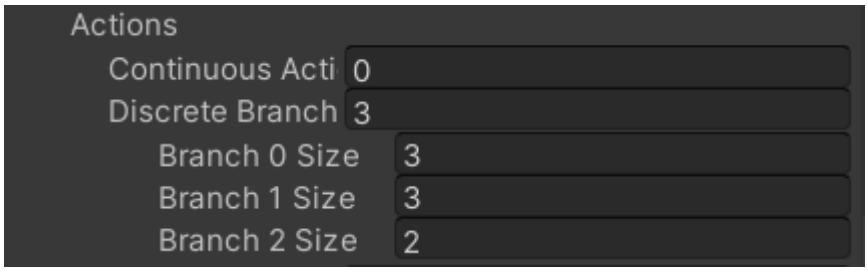
Aluksi koneoppivalle tekoälylle luotiin tilanne, jossa yksi pelaaja instansioitiin (instantiate) kentälle satunnaisesti tiettyjen koordinaattien välille ja tekoälyn täytyi löytää pelaaja siirtämällä tykkitornia ja osua pelaajaan. Aluksi X- ja Z-koordinaatit olivat negatiivisen viidenkymmenen ja viidenkymmenen välillä ja Y-koordinaatti oli nollan ja viidenkymmenen välillä. X- ja Z-koordinaatit vastaavat Unityssä sivuttaisakseleita ja Y-koordinaatti ylöspäin suuntaavaa akselia.

Tekoälylle annettiin kahdeksan havaintoa (observation), jotka olivat kohteen koordinaatit pelissä, tykkitornin koordinaatit sekä tykkitornin kierto sivuttais- ja pystyakselilla. Koordinaattien lisääminen vaatii kolme havaintoa, jonka vuoksi havaintoja on yhteensä kahdeksan. Koordinaatit pystytään kirjoittamaan yhdelle riville koodissa, mutta ne silti vievät kolme havaintoa. Niiden kirjoittaminen erikseen helpottaa hahmottamaan, kuinka monta havaintoa on yhteensä käytetty. Havainnot päivitetään tekoälylle aina Unityn oman Update-metodin avulla. Se ajetaan aina joka kuvalla (frame). Esimerkkikoodista 1 näkyy, kuinka tekoälyn kohteen koordinaatteihin on lisätty tarkistus, ettei tekoälyn kohde ole tyhjä, sillä ilman tarkistusta on riski, että koodirivit aiheuttavat virheen, joka lopettaa tekoälyn toiminnan.

```
//Agents location
 sensor.AddObservation(gameObject.transform.position.x);
 sensor.AddObservation(gameObject.transform.position.z);
 sensor.AddObservation(gameObject.transform.position.y);

 //Agents Base rotation
 sensor.AddObservation(turretbase.transform.localRotation.y);

 // Agents Arms rotation
 sensor.AddObservation(turretArms.transform.localRotation.x);

 if(target != null)
 {
     sensor.AddObservation(target.transform.position.x);
     sensor.AddObservation(target.transform.position.z);
     sensor.AddObservation(target.transform.position.y);

 }
```

Esimerkkikoodi 1. Agentin ”CollectObservations” funktio.

Tekoäly pystyy tekemään kolme erityyppistä toimintaa, kuten kuvasta 5 näkyy. Näitä toimintoja se voi tehdä samaan aikaan tai peräkkäin. Jokainen toiminto voi pitää useamman vaihtoehdon sisällään. Tämä näkyy kuvasta 5.



Kuva 5. Agentille asetetut toiminnot ja niiden määrät.

Tässä projektissa tekoälyllä on ensimmäisessä toiminnossa 3 vaihtoehtoa. Toiminnot ovat liikuttaa tykkitoria vasemmalle, oikealle tai ei mihinkään. Toisessa toiminnossa on myös kolme vaihtoehtoa. Ne ovat liikuttaa tykkitornia ylös, alas tai ei mihinkään. Viimeisessä toiminnossa on kaksi vaihtoehtoa. Ne ovat ampua tai ei ampua.

Koska tekoälyä koulutetaan vahvistusoppimisella, tarvitsee tekoälyltä vähentää palkintoa ajan kuluessa. Tässä projektissa se -1 / maksimiaskeleilla, kuten kuvasta 6 näkyy.

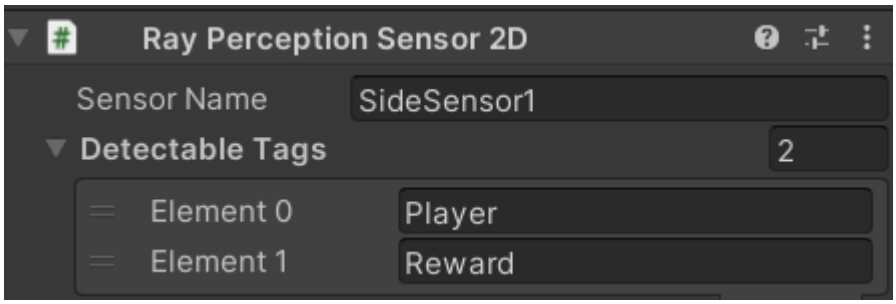
```
AddReward(-1f / MaxStep);
```

Kuva 6. Agentin palkintoa poistava funktio suhteutettuna suurimpaan askelmäärään.

Palkinnon vähentäminen mahdollistaa tekoälyn optimoinnin myöhemmin. Kun tekoälyn ammus onnistuu osumaan pelaajaobjektiin, lisätään palkintoa tekoälylle. Ammuksen osuessa pelaajaobjektiin lasketaan osuma voittotilanteeksi tekoälylle. On myös mahdollista asettaa palkintoa tekoälylle, mutta tämä ei mahdollista tekoälyn optimointia yhtä hyväksi kuin palkinnon lisääminen, sillä voittotilanteen lisäksi halutaan tekoälyn toiminta mahdollisimman hyväksi. Parhaimmassa tilanteessa tekoäly käyttää mahdollisimman vähän askeleita osuakseen pelaajaobjektiin. Jos voittotilanteessa asetetaan palkinnoksi plus yksi, saa tekoäly aina sen verran palkintoa riippumatta siitä, montako askelta voittotilanteeseen menee. Tässä tilanteessa tekoäly ei saa palautetta, että sen kuuluisi parantaa toimintaansa. Kun voittotilanteessa lisätään palkintoa plus yksi, ottaa tekoäly huomioon käytettyjen askeleiden vähennykset palkinnossa. Vuorostaan tilanne antaa palautetta tekoälylle, että parantaminen tekoälyn toiminnassa on mahdollista.

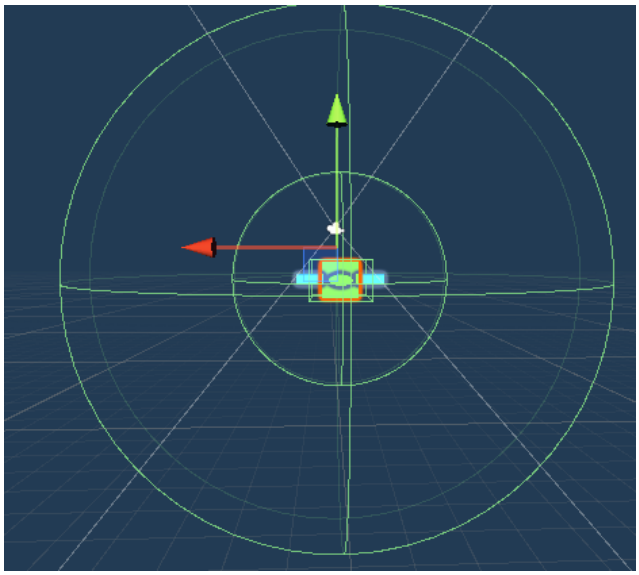
Tykkitorniin lisättiin myös useita ML-Agentsista löytyviä valmiita komponentteja, jotka ovat nimeltään Ray Perception Sensor 2D. Niiden avulla pystyttiin lisäämään säteensuuntauksia (raycast) pelaajan löytämiseksi pelialueelta. Koneoppiva tekoäly osaa löytää ja käyttää kyseisiä komponentteja itse, kunhan komponentit löytävät tekoälyn objektin hierarkiasta tai sen lapsien hierarkiasta. Komponentteja ei myöskään tarvitse lisätä tekoälyn havaintoihin. On tärkeää muuttaa ammuksien komponenteista tunniste (tag) joksikin muuksi. Tässä projektissa ne on merkitty nimellä Ammo. Seuraavaksi tulee Ray Perception Sensor 2D -komponenttiin lisätä Detectable Tags kohtaan pelaajan tunniste sekä ylimääräisten palkintojen tunniste. Kuvasta 7 näkyy, kuinka tarvittavat tunnisteet on lisätty komponenttiin. Tunnisteiden lisääminen on kannattavaa, ettei agentti tunnista ammuksia, sillä muuten ammukset estävät säteensuuntauksia

osumasta pelaajaan. Projektissa käytettiin 2D-sensoreita 3D-sensoreiden sijaan, sillä ne ovat muuten samat, mutta 3D mahdollistaa säteen liikuttamisen ylös- tai alaspäin alkamiskohdasta. Tässä projektissa säteet menevät suoraan, joten ei ole tarpeellista käyttää 3D-sensoreita.



Kuva 7. Säteensuuntauskomponenttiin lisätyt Detectable Tags.

Pelaajaobjektiin lisättiin viisi Sphere Collider -komponenttia, joihin tekoälyn ammuksen osuessa annetaan tekoälylle palkintoa. Collider-komponentit ovat kolmen ja kahdenkymmenen toimintasäteen väliltä. Collider-komponenteille tarvitsee antaa erillinen tunniste ja poistaa komponentti käytöstä, jos ammus osuu siihen. Tämä auttaa tekoälyä oppimaan nopeammin ampumaan pelaajaa päin. Lopullisessa versiossa pelaajaobjekteilla on vain kaksi Sphere Collider -komponenttia, sillä tekoäly ei oppinut löytämään pelaajaa Collider-komponenttien ollessa liian kaukana pelaajaobjektista. Pelaajaobjektilla on kahden Sphere Collider -komponentin lisäksi Box Collider -komponentti. Kun tekoälyn ammus osuu Box Collider -komponenttiin, poistetaan pelaajaobjekti ja asetetaan tekoälylle voittotilanne. Lopullisessa versiossa Sphere Collider -komponentit ovat ulkona pelaajaobjektista kolmen ja seitsemän yksikön verran, kuten kuvasta 8 näkyy.



Kuva 8. Pelaajaobjektin päällä olevat Collider-komponentit.

Tekoälyn kouluttaminen

Pystyäkseen kouluttamaan tekoälyä täytyy ladata Python laitteelle. Tämän lisäksi tulee luoda virtuaalinen ympäristö, jonka avulla tekoälyn kouluttaminen tapahtuu. Ympäristössä on mahdollista nopeuttaa ajan liikkumista pelissä jopa satakertaiseksi. Joka kerta, kun tekoälyä aletaan kouluttamaan, tarvitsee virtuaalinen ympäristö aktivoida. Virtuaaliympäristön saa aktiiviseksi avaamalla komentorivin ja ajamalla ympäristön tiedostoista löytyvän aktivointikoodin. Ympäristön nimi vaihtelee sen perusteella, mikä nimi on asetettu asennusvaiheessa. Tässä projektissa sen nimi on sample-env. Virtuaaliympäristön saa aktiiviseksi avaamalla komentorivillä kansion sample-env ja suorittamalla komennon

\Scripts\activate

Tämän jälkeen virtuaalinen ympäristö on valmiina tekoälyn koulutusta varten. Kun itse koulutus halutaan aloittaa, ajetaan koodirivi

mlagents-learn Config/ShooterAgent.yaml --run-id=Shooter001

Komennossa mlagents-learn aloittaa koulutuksen. Tämän jälkeen voidaan lisätä komento Config/ShooterAgent.yaml, joka on tekoälyn hyperparametrit. Niitä ei ole pakko muokata, mutta niiden vaihtelu nopeuttaa ja parantaa tekoälyn kehitystä. Tässä tapauksessa ne ovat Unityn sisällä Config-nimisessä kansiossa tiedostonimellä ShooterAgent. Tiedostot voidaan halutessa nimetä toisin. Komentojen jälkeen tuleva --run-id=Shooter001 on tekoälyä vastaava koulutustiedoston nimi, joka on tässä tapauksessa Shooter001. Sen voi myös haluttaessa nimetä uudelleen. Käytetyn nimen tulee olla eri jokaisella koulutuskerralla, muuten nimi aiheuttaa virheen.

Kun Unityssä käynnistetään peli, alkaa tekoäly automaattisesti kouluttamaan itseään virtuaaliympäristön avulla, kuten kuvasta 9 näkyy.

Kuva 9. Komentokehotteen antama ilmoitus mahdollisuudesta aloittaa koulutus.

Koulutuksen alettua pitää odottaa hetken aikaa, jotta saadaan selville, alkaako koneoppiva tekoäly oppimaan. Odotettava aika riippuu tekoälyn monimutkaisuudesta. Tässä projektissa aikaa kului noin neljäkymmentäviisi minuuttia. Oppimista pystyy seuraamaan komentoriveiltä Std of Reward -arvo ja Mean Reward -arvo olevien arvojen perusteella. Mean Reward -arvo on tekoälyn keräämät pisteet oppimiskerroksien aikana. Std of Reward -arvo on Mean Reward -arvo suhteutettuna eri koulutuskertoihin. Koneoppivan tekoälyn tuloksissa halutaan

Mean Reward -arvo mahdollisimman isoksi ja Std of Reward -arvo mahdollisimman pieneksi, jolloin jokainen pelikerta olisi samanlainen ja pelikerroissa ei olisi paljon eroja toiminnassa. Yleensä koulutuksen alussa molemmat arvot vaihtelevat paljon. Arvojen tasoittuessa huomataan, että kouluttaminen toimii. Tämän vuoksi oppimista pitää odottaa hetki, jotta saadaan selville oppiiko tekoäly. Tarvittaessa tekoälyyn tehdään muutoksia ja yritetään sen jälkeen uudestaan. Alussa, vaikka Std of Reward -arvo olisi suuri, saattaa tekoäly silti oppia, kunhan Mean reward -arvo kasvaa. Projektissa useasti Mean Reward -arvo kasvoi, mutta Std of Reward -arvo ei laskenut ollenkaan ja pysyi suurena. Kuvasta 10 näkyy, kuinka koulutuksen aloituksessa arvot heittelevät paljon. Arvojen heittelyjen vuoksi koulutuskertoja jouduttiin tekemään useita. Lopullinen versio tekoälystä vaati kuusikymmentäneljä koulutuskertaa.

```
[INFO] ShootPlayer. Step: 10000. Time Elapsed: 135.840 s. Mean Reward: 0.134. Std of Reward: 1.755. Training.
[INFO] ShootPlayer. Step: 20000. Time Elapsed: 262.870 s. Mean Reward: 0.552. Std of Reward: 1.919. Training.
[INFO] ShootPlayer. Step: 30000. Time Elapsed: 385.863 s. Mean Reward: -0.860. Std of Reward: 0.256. Training.
[INFO] ShootPlayer. Step: 40000. Time Elapsed: 509.917 s. Mean Reward: -0.696. Std of Reward: 0.224. Training.
[INFO] ShootPlayer. Step: 50000. Time Elapsed: 631.058 s. Mean Reward: -0.108. Std of Reward: 1.536. Training.
[INFO] ShootPlayer. Step: 60000. Time Elapsed: 759.098 s. Mean Reward: -0.124. Std of Reward: 1.706. Training.
```

Kuva 10. Agentin koulutuksen aloitus ja siitä tulleita arvoja.

Vahvistusoppimisen lisääminen

Kehityksessä hyödynnettiin vahvistusoppimisen lisäksi jäljitelmäoppimista. Pystyäkseen hyödyntämään jäljitelmäoppimista täytyy ensin lisätä tekoälyn koodiin funktio Heuristic. Tämän avulla pystytään tekoälyä liikuttamaan manuaalisesti. Esimerkkikoodista 2 näkyy, kuinka Heuristic-metodi tarvitsee kutsuunsa tekoälyn toimintalistan. Listan avulla voidaan hyödyntää samaa koodia kuin tekoäly käyttää. Esimerkkikoodista 2 näkyy myös, kuinka eri näppäinpainikkeisiin on lisätty eri toimintalistan osia, esimerkiksi A-painike siirtää tykkitornia vasempaan ja D-painike siirtää sitä oikeaan, kun taas W-painike siirtää tykkitornia ylös ja S-painike alas. Jotta tekoäly käyttäisi Heuristic-metodia normaalin koodin sijaan, tarvitsee objektin Behavior Parameters -komponenttiin käydä vaihtamassa Behavior Type Heuristic Only -tilaan komponentin valikkoa käyttäen.

```
public override void Heuristic(in ActionBuffers actionsOut)
    {
        ActionSegment<int> discreteActionsOut = actionsOut.Dis-
creteActions;

        if (Input.GetKey(KeyCode.A))
        {
            discreteActionsOut[0] = 1;
        }
        if (Input.GetKey(KeyCode.D))
        {
            discreteActionsOut[0] = 2;
        }
        if (Input.GetKey(KeyCode.W))
        {
            discreteActionsOut[1] = 1;
        }
        if (Input.GetKey(KeyCode.S))
        {
            discreteActionsOut[1] = 2;
        }
        if (Input.GetKeyDown(KeyCode.Space))
        {
            discreteActionsOut[2] = 1;
        }
    }
```

Esimerkkikoodi 2. Agentin Heurestic-metodi.

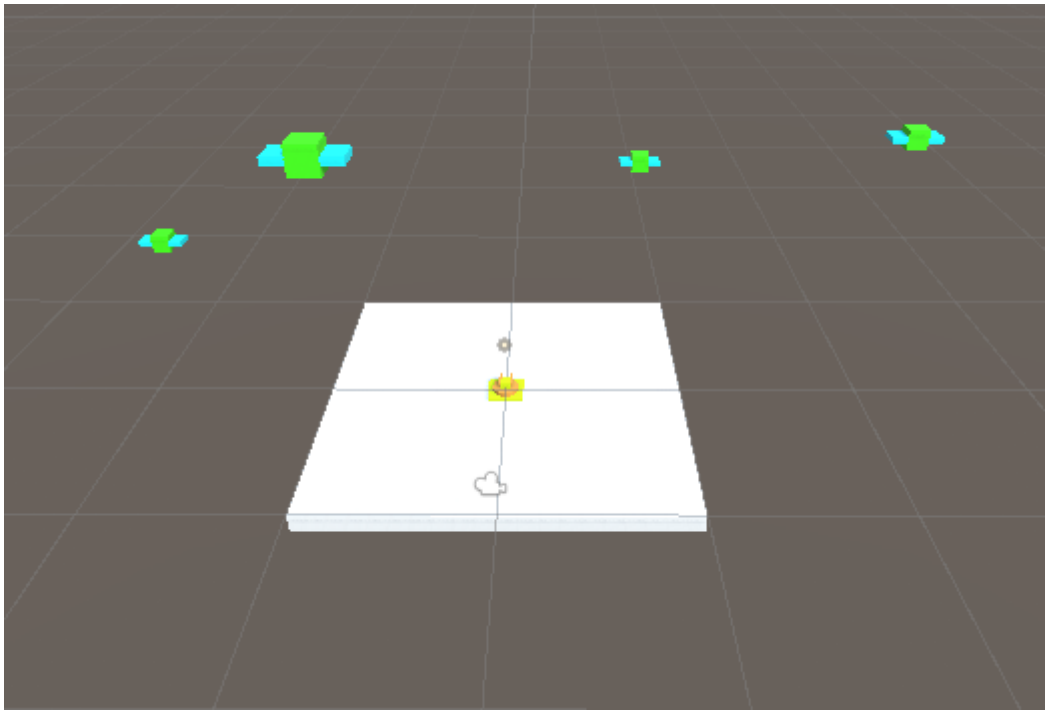
Pystyäkseen hyödyntämään jäljitelmäoppimista täytyy ensin nauhoittaa demo halutusta toiminnasta tekoälyn käytettäväksi. Demon nauhoittaminen onnistuu helposti Unityn sisältä käyttämällä Demonstration Recorder -komponenttia, joka löytyy ML-Agentsin-paketista. Demoa varten on tarpeellista tehdä Heuristic-metodi ja lisätä tarvittavat näppäinpainallukset. Tallenteen luominen onnistuu asettamalla komponentin Record-painikkeen päälle ja painamalla Unityn editorin päälle. Tämän jälkeen tarvitsee vain toimia tekoälyn halutulla tavalla, jotta tekoäly pystyisi hyödyntämään demoa mahdollisimman hyvin. Demo kannattaa viedä voittotilanteeseen useaan otteeseen. Demon saa valmiiksi pysäyttämällä editorin. Kuten esimerkkikoodista 3 näkyy, seuraavaksi täytyy demo lisätä hyperparametreihin ja asettaa sille painoarvo. Tässä projektissa käytettiin painotusta kymmenen ja kolmenkymmenen prosentin välillä. Kun siirryttiin yhdestä pelaajasta useampaan, luovuttiin jäljitelmäoppimisen käytöstä, sillä tekoäly alkoi luottamaan liikaa demoon itsenäisen oppimisen sijaan.

```
reward_signals:
  extrinsic:
    gamma: 0.88
    strength: 1.0
gail:
  strength: 0.12
    demo_path: Demo/ShootPlayerDemo3.demo
```

Esimerkkikoodi 3. Vahvistumisoppimisen ja jäljitelmäoppimisen painoarvot hyperparametreissä.

Kehitys monella pelaajaobjektilla

Kun tekoäly oli oppinut löytämään yhden pelaajan ja tuhoamaan sen, alkoi tekoälyn muokkaaminen, jotta tekoäly voisi toimia myös useamman pelaajan ollessa kentällä. Tekoälyä koulutettiin siten, että satunnaisesti vaihdeltiin pelaajien lukumäärää yhden ja viiden pelaajan välillä. Pelaajaobjektien sijainnit asetettiin myös satunnaisesti samanlailla kuin aikaisemmin. Kuvasta 11 näkyy, kuinka neljä pelaajaobjektia on satunnaisesti asetettu kentälle.



Kuva 11. Pelaajaobjektit satunnaisesti instansioituna Unityn skeneen.

Kun tekoälyllä alkoi olla ongelmia löytää pelaajaobjekteja, aloitussijaintien väliä pienennettiin. Lopulta X- ja Z-akseleiden arvot olivat negatiivisen kahdenkymmenen ja kahdenkymmenen välillä ja Y-akselin arvot olivat nollan ja kahdenkymmenen välillä.

Moninpelin lisääminen

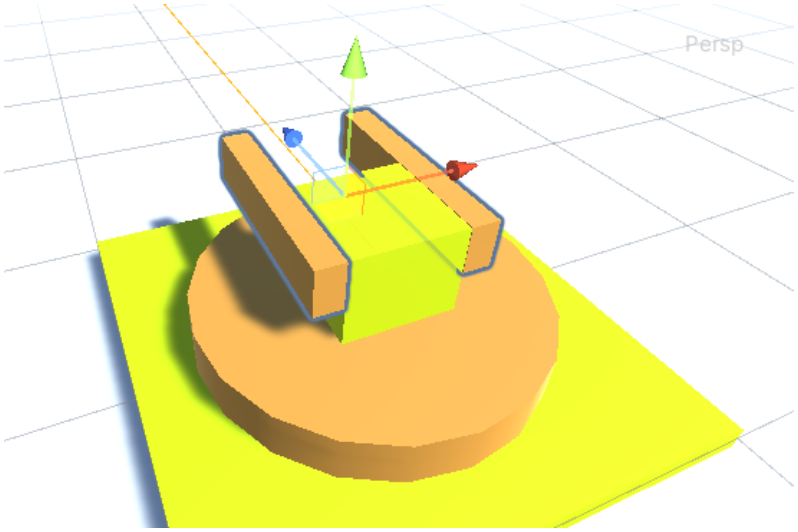
Tekoälyn toimittua yksinpelissä ryhdyttiin muokkaamaan projektista moninpeliä. Aluksi sovellukseen ladattiin Photon Pun 2 -laajennus, jonka avulla moninpeli toteutettiin. Kun laajennus oli saatu lisättyä, vaati se käymistä Photonin verkkosivuilla. Sieltä piti hakea koodi, joka yhdistettiin kyseisen sovelluksen Photonin asetuksiin. Näin peli osasi liittyä pilveen, jossa Photonin palvelimet ovat. Peliin tehtiin erillinen objekti, joka vastaa moninpelistä. Objektiin luotiin koodia, joka ajetaan heti, kun peli avautuu. Pelin aluksi liitytään palvelimelle, jos peli ei ole vielä liittynyt palvelimelle. Tämän jälkeen liitytään automaattisesti kyseisen palvelimen aulaan, jossa voidaan tehdä oma huone tai liittyä jo valmiiseen huoneeseen. Kun kaikki pelaajat ovat liittyneet huoneeseen, voidaan aloittaa peli. Peliin liittyminen on tehty lisäämälle jokaiselle pelaajaobjektille metodi, joka kutsuu PhotonView.RPC-metodia. Metodin avulla, kun pelaajia liittyy eri vaiheissa, saavat kaikki pelaajat tarvittavat tiedot muista pelaajaobjekteista. Kun kaikki pelaajat ovat valmiita, käynnistyy tekoäly ja pelaajien täytyy varoa tekoälyn ammuksia.

Pelaajaobjekteille täytyy lisätä erillistä koodia, jotta ne saatiin toimimaan. Ilman sitä, ei peli tietäisi, mitä pelaajaobjektia kukakin pelaaja ohjaa tai mitä kameraobjektia kuuluisi käyttää. Pelin alussa kaikkien pelaajaobjektien kamerat ovat pois päältä. Käyttämällä PhotonView.IsMine-metodia pystytään tarkistamaan, omistaako pelaaja tietyn pelaajaobjektin. Jos pelaaja omistaa objektin, voidaan sen kameraobjekti laittaa päälle. Tarkistus ja kameran päälle laittaminen asetettiin Unityn omaan Start-metodiin, joka ajetaan heti objektin tullessa aktiiviseksi. Pelaajan liike sijoitetaan Unityn omaan Update-metodiin, joka ajetaan joka kuvalla. On tärkeää muistaa laittaa Update-metodiin sama tarkistus PhotonView.IsMine-metodia käyttäen, jotta vain omaa pelaajaobjektia liikutetaan

annetulla syötteellä (input). PhotonView osaa automaattisesti määritellä pelaajien liikkeen sen jälkeen muille samassa huoneessa pelaaville pelaajille.

4.4 Ongelmatilanteet sovelluksen kehityksessä

Sovelluksen kehityksessä tuli vastaan monenlaisia ongelmia. Kun tekoälyn kehitys aloitettiin vain yhdellä pelaajalla, tekoäly vaikutti toimivan hyvin. Pelaajien määrän kasvaessa tekoäly ei osannut enää soveltaa toimintaansa. Se luotti liian vahvasti vain yhden pelaajan olevan pelissä. Suurin ongelma tuli esiin, kun tekoäly pystyi ampumaan aikaisintaan tietyn ajan kuluttua ja ammukset liikkuivat liian hitaasti. Usein tekoäly oli ehtinyt jo vaihtaa tähtäystä, kun ammus osui pelaajaobjektiin. Tämä vaikeutti kehitystä. Kyseinen ongelma korjattiin antamalla tekoälylle vähän palkintoa, jos se onnistui katsomaan pelaajaa päin. Tämä mahdollisti pelaajaan päin ampumisen heti, kun olosuhteet sen mahdollistivat. Korjauksessa onnistuttiin, koska koodiin lisättiin erillinen säteensuuntaus, joka lisäsi palkintoa tekoälylle. Kuvasta 12 näkyy tekoälylle lisätty yksittäinen säteensuuntaus. Se näkyy oranssina viivana, joka kulkee suoraan eteenpäin tykkitornista.



Kuva 12. Tekoälyyn lisätty erillinen säteensuuntaus.

Palkinnon lisäys tapahtui, jos säteensuuntaus tunnisti pelaajaobjektin samalla kuvalla. Lisäyksen myötä oli tarpeellista muokata ylimääräisille Collider-

komponenteille erillinen tunniste, jottei tämä erillinen säteensuuntaus havainnoisi valheellista tietoa ylimääräisistä collider-komponenteista.

Toinen suuri ongelma kehityksessä tuli esiin, kun tekoäly alkoi ampumaan kohtisuoraan ylös tai alas, eikä tehnyt enää muita toimintoja, vaan jäi jotenkin jumiin. Tekoälyä koitetiin korjata laittamalla erillinen näkymätön Collider-komponentti tekoälyn alle, joka antoi negatiivista palautetta ammusten siihen osuessa. Tämä ratkaisu ei toiminut, sillä se loi tilanteita, jossa tekoäly lakkasi ampumasta kokonaan tai se alkoi tähtäämään suoraan ylös ja lakkasi ampumasta. Kääntymisongelma korjaantui laittamalla rajoituksia tekoälyn kääntymiseen niin, ettei tykkitorni voinut kääntyä viittätoista astetta alemmas. Näin estettiin ammusten osumiset alustaan. Lisäksi rajoitettiin tykkitornin liikettä ylöspäin yhdeksänkymmenen asteen kulmaan. Esimerkkikoodista 4 näkyy, kuinka tekoälyn liikettä rajoitetaan. Koodissa on huomioitava, kun käytetään eulerAngles-arvoja, että ne eivät pysty menemään negatiivisen arvon puolelle. Tämän vuoksi koodi saattaa näyttää kummalliselta.

```
    private void MoveUp()
    {
        if (turretArms.transform.localRotation.eulerAngles.x < 20)
        {
            turretArms.GetComponent<Transform>().Rotate(-1, 0, 0);
        }
        else if (turretArms.transform.localRotation.eulerAngles.x >
270)
        {
            turretArms.GetComponent<Transform>().Rotate(-1, 0, 0);
        }
    }

    private void MoveDown()
    {
        if (turretArms.transform.localRotation.eulerAngles.x < 15)
        {
            turretArms.GetComponent<Transform>().Rotate(1, 0, 0);
        }
        else if (turretArms.transform.localRotation.eulerAngles.x >
260)
        {
            turretArms.GetComponent<Transform>().Rotate(1, 0, 0);
        }
    }
```

Esimerkkikoodi 4. Tykkitornin funktiot, jotka liikuttavat tykkitornia ylös ja alas.

Toimenpiteet auttoivat huomattavasti tekoälyn toimintaa ja oppimista. Myöhemmin tekoälyyn täytyi lisätä vielä säteensuuntauksia, jotta tekoäly löytäisi paremmin pelaajaobjekteja.

Ajoittain tekoälyllä on vieläkin ongelmia löytää pelaajaobjekteja kentältä. Vaikka sillä on paljon säteensuuntaimia, ne eivät aina osu pelaajaobjektiin. Näin tapahtuu varsinkin, jos pelaajaobjektit ovat kaukana tekoälystä. Tämä on yksi syy, miksi luotu tekoäly toimisi huomattavasti tehokkaammin 2D-tilanteessa kuin 3D-tilanteessa, koska kolmiulotteisessa pelissä kenttä, josta tekoäly etsii pelaajaobjekteja, on huomattavasti suurempi kuin 2D-tilanteessa.

Tekoälylle yritettiin lisätä ominaisuus, että se ei ampuisi koko ajan, mutta siinä ei lopullisesti onnistuttu. Aluksi ampumista yritettiin rajoittaa vähentämällä palkintoa joka kerta tekoälyn ampuessa. Tämä johti tilanteeseen, jossa tekoäly ei ampunut ollenkaan. Oli parempi lisätä ajallinen rajoite ammusten ampumiseen. Tällä onnistuttiin välttämään tekoälyn liiallinen ampuminen, vaikkakin ideaalisessa tilanteessa tekoäly ampuisi vain, jos se tietäisi tai arvioisi osuvansa pelaajaobjekteihin.

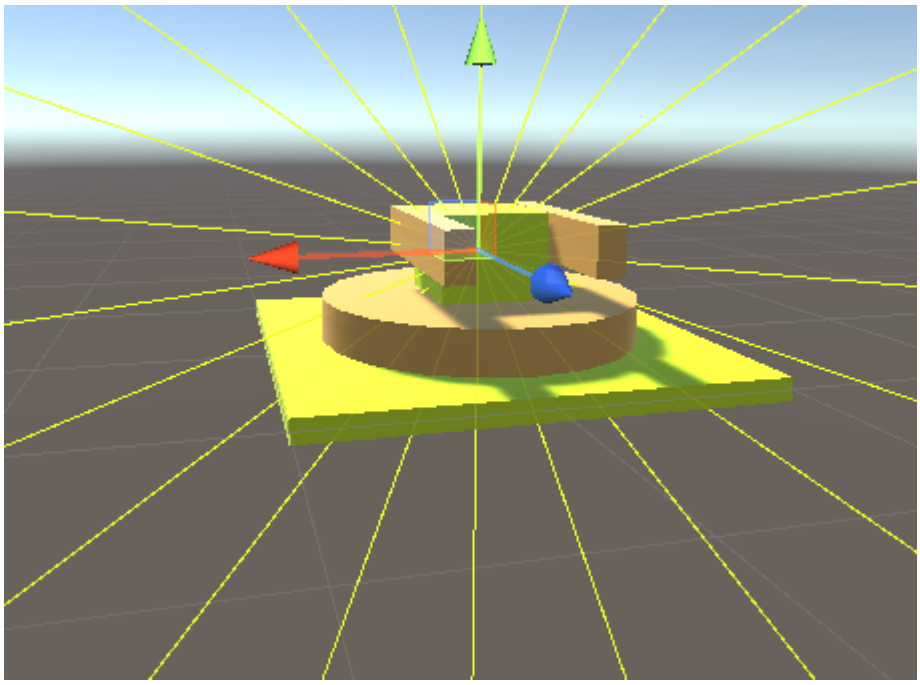
5 Koneoppivan moninpelin tuloksia

5.1 Jatkokehitys

Sovelluksessa on tällä hetkellä paljon erilaisia mahdollisuuksia jatkokehitykselle. Koodi ja ominaisuudet on rakennettu pelissä niin, että mallien vaihtaminen on helppoa eikä vaikuta toimintaan. Sovelluksessa on käytössä tällä hetkellä vain esimerkkejä malleista ja käyttöliittymästä. Tarkoituksena ei tässä projektissa ollutkaan tehdä valmista peliä, vaan testata, saadaanko koneoppiva tekoäly toimimaan Unityn moninpelissä. Ennen kehityksen aloittamista tarkoituksena oli tehdä useat ominaisuudet, jotka on mainittu tässä kappaleessa. Ajan puutteessa jouduttiin ylimääräisiä ominaisuuksia karsimaan.

Moninpeliosuus toimii pelissä, mutta sitä pystyy tulevaisuudessa parantamaan huomattavasti. Siihen voisi tehdä tykkitornin liikkeen varmistuksen, jotta tykkitorni varmasti liikkuu jokaisella pelaajalla samaan suuntaan samanaikaisesti. Lisäksi varmistaa, että tykkitorni ampuu aina samaan aikaan ja samaa pelaajaa kohti. Itse ammusten liikkeitä ei tarvitse tarkastaa, jos kaikki muut asiat on tarkastettu. Ammukset lentävät samaan suuntaan. Tällä hetkellä jokainen pelaaja joutuu poistamaan aloituskäyttöliittymän tätä varten tehdyllä nappulalla. Photonissa on mahdollisuus tehdä tapahtumia, joiden avulla yksi pelaaja pystyy poistamaan aloituskäyttöliittymän kaikilta pelaajilta kyseisessä pelissä.

Pelin ja tekoälyn optimointi luo ison osan jatkokehityksestä. Tekoälyllä on käytössään paljon erilaisia säteensuuntauksia, kuten kuvasta 13 näkyy. Tekoälyllä on todellisuudessa kuusikymmentä säteensuuntausta osoittamassa ympärilleen ja yksi eteenpäin. Säteensuuntausten poistaminen tai optimointi vaatii paljon testausta, mutta testauksen avulla saisi parhaiten vapautettua tehoja takaisin pelille. On myös mahdollista, että peliin täytyy vielä lisätä säteensuuntauksia. Välillä tekoälyllä on vaikeuksia löytää pelaajaa pelialueelta.



Kuva 13. Agentin pystysuunnassa olevat kaksikymmentä säteensuuntaajaa.

Toinen suuri kehityskohde tekoälylle on hyperparametrien optimointi. Niiden optimointia kannattaisi tehdä samaan aikaan muiden optimointien kanssa. Hyperparametrien optimointi vie paljon aikaa, koska niitä pitää manuaalisesti muokata ja katsoa, parantuuko tekoälyn toiminta. Tämän vuoksi niiden optimointi on erittäin hidasta.

Peliin voisi myös hyvin lisätä esteitä, tässä tilanteessa esimerkiksi asteroideja, joiden takana pelaajat voisivat olla suojassa koneoppivalta tekoälyltä ja sen ammuksilta. Tämä lisäys vaatii todennäköisesti tekoälyn uudelleen koulutuksen, jotta se osaisi vaihtaa pelaajakohdetta, jos pelaaja on suojassa. Lisäys voisi toimia pelkästään muokkaamalla säteensuuntauksia. Ilman muokkausta säteensuuntaukset menisivät esteiden läpi ja näkisivät pelaajaobjektin, vaikka se olisi suojassa. Tällä hetkellä koneoppivatekoäly ei silti osaisi vaihtaa kohdetta, joten se tarvitsisi kuitenkin uudelleen kouluttaa vaihtamaan kohdetta pelaajan ollessa piilossa. Samalla koneoppivalle tekoälylle voisi tehdä alustan, mikä liikkuisi joko automaattista reittiä tai erillisen tekoälyn avulla. Tämäkin vaatisi tekoälyn uudelleenkoulutuksen.

Peliin voisi myös laittaa usean tykkitornin, joita ohjaisivat omat tekoälyt. Ne voisivat olla osana samaa joukkuetta. Joukkueen avulla tykkitornit voisivat tehdä yhteistyötä tuhotakseen pelaajat. Kun usea tekoäly lisätään koulutukseen, tulee ottaa huomioon, että koulutuksesta tulee vielä vaikeampaa ja hitaampaa kuin tällä hetkellä. Tykkitorneille on myös mahdollista antaa omat hyperparametrit, eli jos haluttaisiin niiden oppivan eri tavalla tai eri arvoilla, on se mahdollista. Itse hyperparametrit sisältyisivät samaan yaml-päätteiseen tiedostoon, mutta ne olisivat merkattuina siellä erikseen.

5.2 Loppuanalyysi

Kuten on selvinnyt, koneoppivaa tekoälyä pystyy hyödyntämään moninpelissä. Kehityksen ja toiminnan tehokkuuden kannalta koneoppivan tekoälyn rakentaminen kannattaa tämän tyylisessä pelissä pitää kahdessa tasossa kolmen sijaan. Jos pelissä tarvitsee tekoälyä, jonka tulee olla kolmessa tasossa, olisi

suositeltavaa käyttää jotain muuta tekoälyn muotoa kuin koneoppivaa. Pelkästään jo kehitykseen vaadittava aika koneoppivan ja esimerkiksi tilakone-tekoälyn välillä kolmessa tasossa on suuri. Pelkän koneoppivan tekoälyn kehitykseen kului aikaa noin kolmesataa tuntia. Suurin osa ajasta kului koulutuksessa. Koulutuksessa testattiin, toimiiko tekoäly vai ei. Kun tehtiin samantyylinen tekoäly tilakoneena, kehitykseen meni noin kaksi tuntia. Kehityksessä testaaminenkin oli huomattavasti nopeampaa, sillä tekoälystä sai heti palautteen, jos se ei toiminut, toisin kuin koneoppivan tekoälyn kohdalla.

Vaikein asennusosuus projektin kehityksessä tuli koneoppimis-laajennuksen asennuksessa ja toimintaan saamisessa. Asennusohjeet laajennuksessa ovat hyvät, mutta ne ovat monimutkaiset, joten virheisiin on suuri mahdollisuus. Kun virtuaalisen ympäristön toimintaan saaminen vaatii useiden laajennusten latausta, monimutkaistaa se asennusta huomattavasti. Virtuaalisen ympäristön komennot tehdään komentoriviltä. Virheiden sattuessa on iso riski vahingoittaa järjestelmää. Laajennuksen käyttö oli aika helppoa ja siihen oli tehty valmiiksi useita hyödyllisiä komponentteja. Käyttöön ja soveltamiseen löytyy paljon materiaalia verkosta. Suurin osa materiaalista toimii vain siinä kyseisessä tapauksessa, mihin ne on rakennettu ja ovat hyvin yksinkertaisia. Mutta materiaalin tutkimisesta on mahdollista saada koodia, joita pystyy hyödyntämään omiin projekteihin.

Projektissa moninpeliosuuden tekeminen oli helppoa, sillä käytetyn laajennuksen asentaminen ja toimintakuntoon laittaminen on helppoa. Peliin monipelin luominen vaati hieman ohjelmointia. Ainoana rajoitteena moninpeliin on samanaikaisten pelaajien määrä, jos ei halua maksaa ylimääräisistä pelaajista. Pelin testaaminen oli vaikein osa moninpelin lisäämisessä. Tämä vaati pelin rakentamista, jotta siitä saatiin useampi instanssi moninpeliosan tarkistusta varten. Jos peli on suuri tai monimutkainen, saattaa pelin rakentamisessa kestää huomattavan kauan. Tässä projektissa se ei onneksi kestänyt kuin muutaman minuutin.

6 Yhteenveto

Koneoppivien tekoälyjen käyttö on yleistynyt viime aikoina, koneoppivan tekoälyn tulevaisuudesta ei ole varmaa tietoa. Mahdollisesti tekoälyä koskevaa lakeja tullaan kansainvälisesti lisäämään, sillä tällä hetkellä lakeja on erittäin vähän.

Unityn pelimoottoria on mahdollista käyttää ilmaiseksi kehittämään monenlaisia pelejä erilaisille alustoille. Unityyn löytyy myös paljon erilaisia laajennuksia ja paketteja, joita pystyy lataamaan heidän omasta kauppapaikastansa tai pakettimanagerista. Laajennuksista osa on ilmaisia ja osa maksullisia. Projektissa käytettiin ML-Agents-laajennusta, jonka avulla peliin tehtiin koneoppiva tekoäly, sekä Photon Pun 2 -laajennusta, jonka avulla peliin tehtiin moninpeliosuus. Molemmat käytetyt laajennukset olivat ilmaisia. Photon-laajennuksen avulla moninpeliä pystyy pelaamaan kaksikymmentä pelaajaa samanaikaisesti ilman erillistä maksua.

Projektin kehitys aloitettiin tekemällä koneoppiva tekoäly, joka osasi tuhota yksittäisen pelaajaobjektin. Kun se saatiin toimimaan, muutettiin peliä sisältämään useampia pelaajaobjekteja. Sekä peliä että tekoälyä täytyi muokata, jotta tekoäly onnistuisi tuhoamaan useamman pelaajaobjektin. Lopuksi, kun tekoäly saatiin toimimaan kunnolla useamman pelaajaobjektin pelissä, täytyi vielä tehdä moninpeliosuus. Moninpeliosuuden tekeminen oli suoraviivaista, eikä sen toimintaan saaminen vaatinut tekoälyn uudelleenkoulutusta tai muokkausta.

Projektin kehitys vaati lopullisesti tekoälyltä kuusikymmentäneljä koulutuskertaa. Yhteensä ajallisesti koulutus vaati noin kolmesataa tuntia. Kehityksessä tuli erilaisia ongelmia vastaan, joihin kuitenkin onnistuttiin löytämään ratkaisuja. Projektissa on useita mahdollisuuksia jatkokehitykselle sekä optimoinnille. Pelin tekoäly saatiin toimimaan moninpelissä, vaikkakin välillä se ei onnistu löytämään pelaajia pelialueelta. Virhe tapahtuu harvoin pelaamisen aikana, mutta se on kuitenkin otettava huomioon mahdollisessa jatkokehityksessä.

Insinöörityön päätavoite, eli toimisiko koneoppiva tekoäly pelitarkoituksessa sekä moninpelissä, saavutettiin kehityksen aikana. Kehitys oli sujuvaa, vaikkakin tekoälyn koulutus vaati huomattavan ajan kokonaiskehityksestä.

Lähteet

1. 75 Years of Innovation: Shakey the Robot. Verkkoainesto. SRI. <https://www.sri.com/press/story/75-years-of-innovation-shakey-the-robot/>. Luettu 22.1.2025.

2. Deep Blue. Verkkoaineisto. IBM. <https://www.ibm.com/history/deep-blue>. Luettu 22.1.2025.

3. Mccarthy, Sandy. 2018. Ranked Dota 2 player demolished by AI. Verkkoainesto. University Wire. <https://www.proquest.com/docview/2085297468?parentSessionId=lHREJNgDxQbV28O0JRdGPJJ9xUOgpWTerK%2FRy3UYx80%3D&pq-origsite=primo&accountid=11363&sourcetype=Wire%20Feeds>. Päivitetty 8.8.2018. Luettu 20.1.2025.

4. Eke, Damian Okaibedi. 2023. ChatGPT and the rise of generative AI: Threat to academic integrity. Verkkoainesto. Journal of responsible technology. <https://www.sciencedirect.com/science/article/pii/S2666659623000033?via%3Dihub>. Päivitetty 8.2023. Luettu 20.1.2025.

5. Rahman, Was. 2020. AI and Machine Learning. E-kirja. Ebook Central.

6. Pandharipande, Ashish; Cheng, Chih-Hong; Dauwels, Justin; Gurbuz, Sevgi Z.; Ibanex-Guzman, Javier; Li, Guofa; Piazzoni, Andrea; Wang, Pu & Santra, Avik. 2023. Sensing and Machine Learning for Automotive Perception: A Review. Verkkoaineisto. IEEE sensors journal. <https://ieeexplore-ieee-org.ezproxy.metropolia.fi/document/10089400>. Päivitetty 6.2023. Luettu 24.1.2025.

7. Kaiser, M.Shamim; Xie, Juanying & Rathore, Vijay Singh. 2024. Intelligent Strategies for ICT: Proceedings of ICTCS 2023, Volume 2. E-kirja. Springer eBooks.

8. Choose the plan that is right for you. Verkkoaineisto. Unity Technologies. <https://unity.com/products/compare-plans>. Luettu 18.1.2025.

9. Unity is canceling the runtime fee. Verkkoainesto. Unity Technologies. <https://unity.com/blog/unity-is-canceling-the-runtime-fee>. Luettu 19.3.2025.

10. Unity Engine. Verkkoaineisto. Unity Technologies. <https://unity.com/products/unity-engine> Luettu 17.1.2025.

11. Introduction to scenes. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/Manual/CreatingScenes.html>. Luettu 19.3.2025.

12. Introduction to GameObjects. Verkkoainesto. Unity Technologies. <https://docs.unity3d.com/Manual/CreatingScenes.html>. Luettu 19.3.2025.

13. Entities. Verkkoainesto. Unity Technologies. <https://docs.unity3d.com/Packages/com.unity.entities@1.3/manual/index.html>. Luettu 25.1.2025.

14. Unity’s Data-oriented Technology Stack (Dots). Verkkoainesto. Unity Technologies. <https://unity.com/dots>. Luettu 18.1.2025.

15. ECS for Unity. Verkkoainesto. Unity Technologies. <https://unity.com/ecs>. Luettu 18.1.2025.

16. ML-Agents. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/Packages/com.unity.ml-agents@3.0/manual/index.html>. Luettu 10.1.2025.

17. Setting up ML Agents for Unity in 5 Minutes. Verkkoaineisto. <https://www.youtube.com/watch?v=v1d7SinTzEY>. Luettu 17.1.2025.

18. Class DecisionRequester. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.DecisionRequester.html>. Luettu 10.1.2025.

19. ML-Agents Training. Verkkoaineisto. Github. <https://github.com/gzrjzcx/ML-agents/blob/master/docs/Training-PPO.md>. Luettu 10.1.2025.

20. Photon. Verkkoaineisto. Photon. <https://doc.photonengine.com/pun/current/getting-started/pun-intro>. Luettu 17.1.2025.