Emil Keränen

Vertaileva tutkimus koneoppimisen hyödyntämisestä videopelien reitinhaussa

Tietotekniikan pro gradu -tutkielma

19. toukokuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Emil Keränen

Yhteystiedot: emil.a.keranen@student.jyu.fi

Ohjaaja: Tommi Kärkkäinen

Työn nimi: Vertaileva tutkimus koneoppimisen hyödyntämisestä videopelien reitinhaussa

Title in English: Comparative study of utilizing machine learning in video games' pathfinding

Työ: Pro gradu -tutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 21+0

Tiivistelmä: TODO: tiivistelmä suomeksi

Avainsanat: koneoppiminen, videopeli, reitinhaku, syvä vahvistusoppiminen

Abstract: TODO: In english

Keywords: machine learning, video game, pathfinding, deep reinforcement learning, Soft Actor Critic, Machine Learning Agents, Unity

Kuviot

Sisällys

1 Johdanto

- Tutkimuksen kohteena koneoppimisen tehostama reitinhaku videopeleissä ja sen vertaaminen heuristiseen A*-algoritmiin.

- Yleisesti reitinhaulla tarkoitetaan alku- ja loppupisteen välisen reitin selvittämistä. Useimmiten tarkoituksena on löytää lyhin reitti väistellen samalla matkan varrella olevia esteitä.

- Reitinhakua tarvitaan videopelien lisäksi myös mm. robotiikassa.

- Videopeleissä reitinhaku ilmenee pääasiassa tekoälyagenttien suorittamana toimintana, joten tässä tutkimuksessa keskitytään agentteihin. Näitä agentteja kutsutaan myös ei-pelaaja-hahmoiksi (engl. non-player-character, NPC).

- Ei-pelaaja-hahmot ja itseasiassa videopelien reitinhaku vertautuvat hyvin robotiikkaan ja robottien reitinhakuun.

- Sekä robotiikassa että videopeleissä toiminta-alue voi muuttua hyvinkin paljon reaaliajassa, jolloin reitinhaun täytyy sopeutua muutoksiin nopeasti. Käytetyt ratkaisut sen sijaan voivat vaihdella näiden kahden osa-alueen välillä: robotiikassa tarkkuus ja turvallisuus nousevat tärkeimmiksi ominaisuuksiksi ja vastaavasti videopeleissä nopeus määrittää reitinhaun "hyvyyden".

- Reitinhaku on aina ollut vaativa ongelma videopeleissä, mutta nykyään reitinhaun ongelmallisuus voidaan useimmissa tapauksissa sivuuttaa laatimalla heuristinen ratkaisu A*-algoritmin avulla.

- Koneoppiminen mahdollistaa aiemman kokemuksen hyödyntämisen myöhemmässä toiminnassa. Agentteja voidaan kouluttaa harjoitteludatan avulla, jolloin ne oppivat toimimaan tuntemattomissa tilanteissa. Koneoppimisen ansiosta reitinhaku-agentti voidaan opettaa toimimaan vaativissa ja dynaamisissa pelialueissa, joissa muuttuvat esteet ja alueen labyrinttimäisyys heikentävät A*-algoritmin toimintaa.

- Tutkimuksen ideana on käyttää Unity-pelimoottorille luotuja koneoppimisagentteja (engl. Unity Machine Learning Agents) ja opettaa niitä erilaisten pelialueiden avulla. Opettamisen

jälkeen agentteja testataan oikeilla pelialueilla ja verrataan tuloksia A*-algoritmilla saatuihin tuloksiin.

- ML-agents perustuu PyTorch-kirjastoon ja mahdollistaa vahvistusoppimisen hyödyntämisen.

- Tensorboard-lisäosan avulla voidaan visualisoida palkkioiden keskiarvot ja opetuksen edistyminen opetuksen aikana.

- Pelialueiden on tarkoitus olla monimutkaisia ja dynaamisia, koska A*-algoritmi suoriutuu yksinkertaisista reitinhakutehtävistä moitteettomasti.

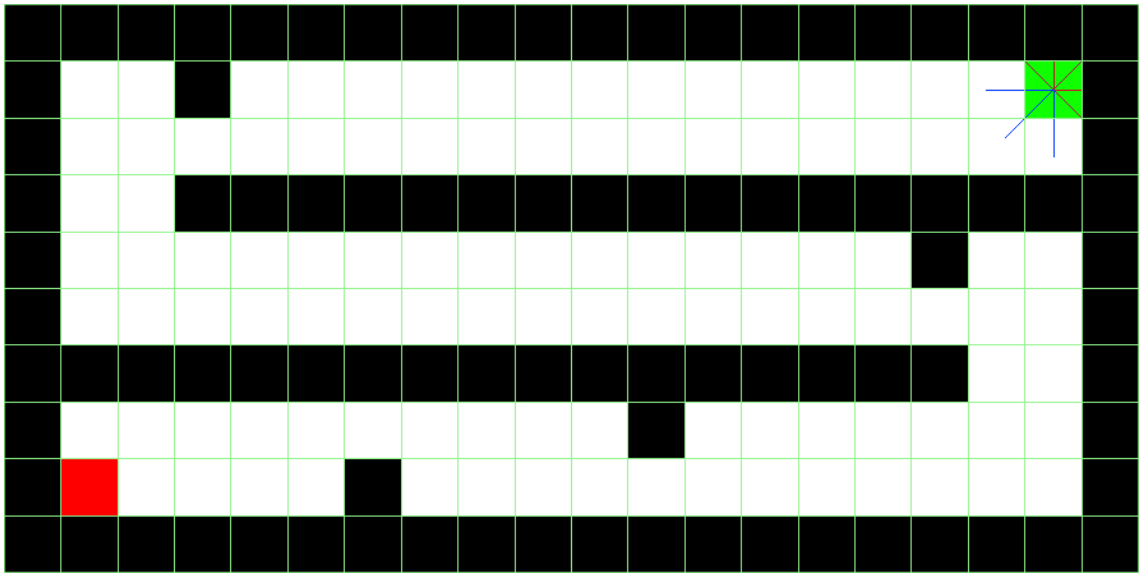
2 Reitinhaku videopeleissä

Reitinhaulla tarkoitetaan yksinkertaisimmillaan reitin selvittämistä kahden pisteen välillä. Se on yksi videopelien tekoälyn ja myös robotiikan tunnetuimmista ja haastavimmista ongelmista, jota on tutkittu jo muutaman vuosikymmenen ajan (Cui ja Shi 2011; Abd Algfoor, Sunar ja Kolivand 2015). Reitinhakua esiintyy monissa eri peligenreissä, kuten roolipeleissä ja reaaliaikaisissa strategiapeleissä, joissa ei-pelaaja-hahmoja (non-player-character, NPC) määrätään liikkumaan ennaltamäärättyyn tai pelaajan määräämään sijaintiin väistellen samalla vastaantulevia esteitä (Cui ja Shi 2011).

2.1 Pelialueen esitystavat

Yleisin alueen esittämistapa on ruudukko (engl. grid), joka sisältää yksittäisiä ruutuja (engl. tile). Ruudut merkitään ennalta vapaiksi tai esteiksi. Vapaista ruuduista muodostetaan verkko, jolloin jokainen vapaa ruutu vastaa yhtä verkon solmuista. Vierekkäisiä solmuja yhdistävät linkit, joita pitkin reitinhaku tapahtuu. Solmujen vierekkäisyys voi tarkoittaa horisontaalista ja vertikaalista vierekkäisyyttä (neljä suuntaa) tai näiden lisäksi myös diagonaalista vierekkäisyyttä (kahdeksan suuntaa). (Botea ym. 2013; Abd Algfoor, Sunar ja Kolivand 2015). Kuvio 1 havainnollistaa yksinkertaista ruudukkoaluetta, jossa liikkuminen tapahtuu neljään suuntaan.

Vähemmän käytettyjä alueen esitystapoja ovat kolmiointi ja kuusikulmiointi (oikea termi kuusikulmioinnista?) (Abd Algfoor, Sunar ja Kolivand 2015). Alueen kolmiointi ja siihen perustuvat TA*- ja TRA*-algoritmi ovat kuitenkin osoittautuneet moninkertaisesti nopeammiksi suurissa pelialueissa verrattuna A*-algoritmiin (Demyen ja Buro 2006). Kuusikulmioihin perustuvat alueet ja reitinhakualgoritmit ovat myös tuottaneet lupaavia tuloksia muun muassa robotiikan tutkimuksessa (Abd Algfoor, Sunar ja Kolivand 2015).



Kuvio 1. Ruudukko, jossa mustat ruudut ovat esteitä ja valkoiset ruudut vapaita. Vihreä ruutu on aloitusruutu ja punainen ruutu on maaliruutu. Vihreästä ruudusta lähtevät viivat ovat mahdollisia etenemissuuntia.

2.2 Reitinhakualgoritmit

Graafin lyhyimmän polun ongelmaa on tutkittu jo vuosikymmenten ajan. Vanhimmat ja tunnetuimmat algoritmit, Dijkstran algoritmi (Dijkstra ym. 1959) ja A*-algoritmi (Hart, Nilsson ja Raphael 1968), esiteltiin jo 50- ja 60-luvuilla. A*-algoritmi on selvästi tunnetumpi videopelien ja robottien reitinhaussa nopeutensa ansiosta (Cui ja Shi 2011; Abd Algfoor, Sunar ja Kolivand 2015; Botea ym. 2013). Tässä tutkimuksessa keskitytään tarkemmin A*-algoritmiin ja sen variaatioihin.

2.2.1 A*-algoritmi

A*-algoritmi on hyvin tunnettu paras-ensin -reitinhakualgoritmi (termi kuntoon?), joka hyödyntää heuristista arviointifunktiota lyhimmän reitin etsimiseen (Cui ja Shi 2011; Duchoň ym. 2014).

Algoritmin toiminta tapahtuu seuraavasti: jokainen aloitussolmun vierekkäinen solmu arvioidaan kaavan f(n)=h(n)+g(n) mukaisesti, jossa n on solmu, h(n) on heuristinen etäisyys solmusta n maalisolmuun ja g(n) on todellinen etäisyys aloitussolmusta solmuun n. Näis-

tä solmuista matalimman f(n)-arvon solmu käsitellään seuraavaksi, jolloin kyseisen solmun vierekkäisten solmujen f(n)-arvot lasketaan. Tämä prosessi jatkuu, kunnes maalisolmu saavutetaan. (oikeat matemaattiset merkinnät). Heuristiikan ollessa nolla A*-algoritmista tulee Dijkstran algoritmi.

A*-algoritmilla on kolme esitettyä ominaisuutta (Hart, Nilsson ja Raphael 1968). A*-algoritmi löytää reitin, jos sellainen on olemassa. Reitti on optimaalinen, jos heuristiikka on luvallinen(admissible?) eli arvioitu etäisyys on lyhyempi tai yhtä suuri kuin todellinen etäisyys. Lopuksi mikään muu algoritmi samalla heuristiikalla ei käy läpi vähemmän solmuja kuin A*-algoritmi eli A* käyttää heuristiikkaa tehokkaimmalla mahdollisella tavalla. (Hart, Nilsson ja Raphael 1968; Cui ja Shi 2011). Luvallisia heuristiikkoja ovat solmujen vierekkäisyydestä riippuen Euklidinen etäisyys, Manhattan-etäisyys, Chebyshev-etäisyys ja Octile-etäisyys (Duchoň ym. 2014; Botea ym. 2013). Manhattan-etäisyyttä käytetään pääasiassa neljän suunnan ja Octile- sekä Chebyshev-etäisyyttä kahdeksan suunnan vierekkäisyyksissä (Botea ym. 2013). Euklidista etäisyyttä voidaan käyttää tilanteessa, jossa

2.2.2 A*-algoritmin variaatiot

A*-algoritmin heikkous on huono skaalautuvuus suurien alueiden reitinhakuun. Esimerkiksi kaksiulotteisessa ja esteettömässä 1000x1000 ruudukkoalueessa solmuja on miljoona, jolloin A*-algoritmin muistinkäyttö ja nopeus koituvat ongelmaksi. (Cui ja Shi 2011; Duchoň ym. 2014) Tämä ongelma esiintyy videopelien lisäksi etenkin robotiikassa, jonka seurauksena A*-algoritmi on inspiroinut tutkijoita monien eri variaatioiden kehittämiseen, kuten Theta*-, Phi*- ja HPA*-algoritmit (Duchoň ym. 2014).

2.3 Reitinhaun ongelmat

Reitinhaun on oltava videopeleissä nopeaa ja laskennallisesti tehokasta. Lisäksi reittien on näytettävä realistiselta pelaajalle. Yhden agentin staattisen ruudukkoalueen reitinhakuongelma on ratkaistavissa nopeasti ja tehokkaasti heuristisella A*-algoritmilla, mutta nykyään videopeleissä reitinhakuongelmat ovat monimutkaisempia. Reitinhakuongelmat pitävät sisällään esimerkiksi useamman agentin samanaikaista reitinhakua ja reaaliajassa muuttuvien

alueiden reitinhakua. Nykytutkimus keskittyykin pääasiassa monimutkaisiin reitinhakuongelmiin.

Useamman agentin samanaikaisessa reitinhaussa alueella on useampi kuin yksi agentti ja jokaisella niistä on oma aloitus- ja lopetuspisteensä. Jos jokaisen agentin reitinhakuun sovelletaan A*-algoritmia, on vaadittu laskennallinen teho eksponentiaalinen agenttien lukumäärän suhteen O(b potenssiin k) (tähän merkintätapa ja lähde). Laskennallisen tehon lisäksi jokaisen agentin täytyy tarvittaessa väistää toisia agentteja ja mahdollisesti muita esteitä. A*-algoritmi osoittautuu riittämättömäksi ongelman ratkaisuun.

3 Koneoppiminen

Soft Actor-Critic on Haarnojan ym. kehittämä syvä vahvistusoppimis algoritmi (Haarnoja ym. 2018).

3.1 Neuroverkot

3.2 Vahvistusoppiminen ja syvä vahvistusoppiminen

3.3 Soft-Actor Critic -algoritmi

4 Unity

Unity on Unity Technologiesin kehittämä pelinkehitysalusta, joka sisältää oman renderöinti- ja fysiikkamoottorin sekä Unity Editor -nimisen graafisen käyttöliittymän (Juliani ym. 2018). Unityllä on mahdollista kehittää perinteisten 3D- ja 2D-pelien lisäksi myös esimerkiksi VR-pelejä tietokoneille, mobiililaitteille ja pelikonsoleille. Unitystä onkin vuosien mittaan tullut yksi tunnetuimmista pelinkehitysalustoista, jonka parissa työskentelee kuukausittain jopa 1.5 miljoonaa aktiivista käyttäjää ("Unity" 2022).

Viime vuosina Unityä on käytetty simulointialustana tekoälytutkimuksen parissa. Unity mahdollistaa lähes mielivaltaisten tilanteiden ja ympäristöjen simuloinnin 2D ruudukkokartoista monimutkaisiin pulmanratkaisutehtäviin, joka on sen suurimpia vahvuuksia simulointialustana. Kehitystyö ja prototypointi ovat Unityllä myös erityisen nopeaa. (Juliani ym. 2018).

4.1 Unityn hierarkia

Tässä luvussa käsitellään Unityn hierarkiaa. Aliluvuissa käydään läpi hierarkian osat ylimmästä lähtien.

4.1.1 Unity-projekti

Unityn hierarkian ylin osa on projekti, jonka luomisesta kehitystyö aina alkaa. Unityssä on mahdollista luoda projekti valmiista pohjista, joita ovat esimerkiksi 2D-, 3D- ja VR-pohjat. Pohjien avulla projekteihin saa lisättyä suoraan suositeltavat, parhaita käytäntöjä mukailevat asetukset.

Unity-projekteja voidaan hallinnoida ja avata erillisellä Unity Hub -sovelluksella. Unity Hub kertoo muun muassa mitä Unityn versiota projekti tukee. Projekteja voi tarvittaessa siirtää (engl. migrate) toimimaan uusimmilla Unityn versioilla, mutta siirto voi aiheuttaa toiminnallisuuksien muutoksia tai virheitä projektissa.

4.1.2 Näkymät

Projektista seuraavana hierarkiassa on näkymät (engl. scene). Näkymät toimivat työskentelyalustoina projektissa. Projekti sisältää aina yhden tai useamman näkymän, koska ilman niitä mitään ei pysty luomaan. Tavallisesti yksittäinen näkymä kuvaa aina yhtä tasoa pelissä, ja siirryttäessä toiselle tasolle Unity pystyy lataamaan ajon aikana uuden skenen. Näkymän lataaminen voi tosin viedä aikaa, joten lataus peitetään useimmiten latausruuduilla, jotka voivat myös olla omia, yksinkertaisia näkymiä. Yksinkertaisimmillaan peli voi kuitenkin sisältää vain yhden näkymän joka muokkautuu ja jota muokataan ajon aikana.

Projektin luonnin jälkeen Unity lisää siihen automaattisesti aloitusnäkymän, joka sisältää kameran ja valonlähteen. Tätä näkymää voi lähteä muokkaamaan lisäämällä siihen erilaisia objekteja, esimerkiksi maata ja erilaisia geometrisia muotoja.

4.1.3 Peliobjektit ja Prefabit

Peliobjektit (engl. GameObject) ovat tärkein osa Unityn pelinkehitysprosessia, koska kaikki peliin luotavat objektit ovat taustaltaan peliobjekteja. Peliobjektit eivät itsessään tee mitään tai näytä miltään, vaan ne toimivat säiliöinä komponenteille. Peliobjekteja voidaan järjestellä vanhempi-lapsi -periaatteella. Lapsiobjektit liikkuvat vanhemman mukana, jolloin niitä ei tarvitse liikutella näkymässä erikseen.

Prefabit ovat peliobjektien valmiita malleja, joita luodaan peliobjektien tavoin. Prefabit vähentävät toistuvaa työtä peliobjektien luomisen yhteydessä. Prefabeille voidaan lisätä komponentteja ja lapsiobjekteja kuten peliobjekteille. Kun prefabi lisätään näkymään, sen komponentteja ja arvoja voidaan muuttaa tarvittaessa ilman, että alkuperäisen prefabin arvot ja komponentit muuttuvat.

4.1.4 Komponentit

Komponentit antavat peliobjekteille ominaisuuksia ja toiminnallisuuksia kuten muoto, pinta, kuvio, väri tai fysiikka. Komponentteja voi olla rajattomasti, mutta peliobjektilla on luonnin jälkeen vähintään Transform-komponentti, joka määrittää peliobjektin sijainnin, suunnan ja

koon. Transform-komponenttia ei voi poistaa peliobjektilta. Komponentit mahdollistavat peliobjekteille myös erilaisia toiminnallisuuksia skriptien kautta. Usein komponenteilla on arvoja, joita voi muokata asetusnapin, pudotusvalikon tai tekstikentän avulla. Komponentit voivat sisältää myös viittauksia muihin peliobjekteihin, tiedostoihin tai assetteihin(käännös?). Esimerkiksi Sprite Renderer -komponenttiin voidaan lisätä viittaus kuvatiedostoon, jolloin Unity renderöi peliobjektin kohdalle lisätyn kuvan.

4.1.5 Skriptit

Ohjelmoinnin merkitys Unityn käytössä tulee skripteistä. Skriptit ovat ohjelmakooditiedostoja, joita voi lisätä peliobjektiin komponentin tavoin, jos valmiit komponentit eivät riitä toiminnallisuuksiltaan. Skripteissä voi esimerkiksi määritellä ominaisuuksia ja arvoja, joita voi muokata komponenttilistauksessa tai ajon aikana. Unity tukee tällä hetkellä vain C-# -ohjelmointikieltä, mutta ennen myös Javascriptiin pohjautuvaa UnityScript-ohjelmointikieltä.

Unity tarjoaa skripteihin MonoBehaviour-pohjaluokan, joka mahdollistaa pelinkehityksen tärkeimmät osat eli Start()-aloitusfunktion ja Update()-päivitysfunktion. Start()-funktio ajetaan ennen yhtäkään päivitysfunktiota, joten siinä voidaan määrittää ja alustaa tarvittavat alkuarvot. Update()-funktio ajetaan joka ruudunpäivityskerralla, joten siihen sijoitetaan usein pelilogiikka ja mahdollisesti fyysiset toiminnallisuudet.

Skriptien avulla voidaan tehdä lähes kaikki samat asiat kuin Unity Editorissa. Peliobjekteja voidaan etsiä tagien tai nimien kautta ja niille voidaan lisätä ja poistaa komponentteja ajon aikana. Käyttämättömät peliobjektit voidaan tuhota tai asettaa epäaktiivisiksi jos niitä ei tarvita.

4.2 Machine Learning Agents

Machine Learning Agents on Unitylle kehitetty ilmainen koneoppimispaketti, joka mahdollistaa Unity Editorilla luotujen simulaatioympäristöjen ja Python APIn välisen vuorovaikutuksen. ML-Agents SDK (Software Development Kit) tarjoaa kaikki toiminnallisuudet ja skriptit toimivan koneoppimisympäristön luomiseen. (Juliani ym. 2018).

4.2.1 ML-Agents SDK

ML-Agents SDK sisältää kolme ydinosaa: sensorit, agentit ja akatemia. Agentti-komponentti voidaan lisätä suoraan Unityn peliobjektille, jolloin se pystyy keräämään havaintoja, suorittamaan toimintoja ja vastaanottamaan palkkioita. Sensorit mahdollistavat havaintojen keräämisen eri tavoin. Akatemia ylläpitää tietoa simulaation askelmäärästä ja ympäristön parametreista sekä ohjaa agenttien toimintaa.

Agentin käytäntö määritellään Behavior Name -nimikkeen avulla. Eri agenteilla voi olla sama käytäntö, jolloin agentit käyttävät kyseistä käytäntöä päätöksentekoon ja jakavat harjoitteludatan keskenään. Myös useiden erilaisten agenttien toiminta voidaan mahdollistaa erinimisillä käytännöillä.

4.2.2 Python API ja PyTorch

Python APIa käytetään Unityllä tehdyn simulaatioympäristön ja koneoppimissilmukan käsittelyyn. APIn avulla tekijän ei tarvitse itse olla suoraan yhteydessä Pythonin koneoppimiskouluttajaan, vaan API tarjoaa helppokäyttöiset, valmiit menetelmät koneoppimissilmukan luomiseen. Tarkemmin APIsta ja sen toiminnasta voi lukea dokumentaatiosta (tähän linkki sivun alalaitaan tulevasta docsista?).

PyTorch on avoimen lähdekoodin koneoppimiskehys, johon pohjautuen Unity ML-agents -paketin koneoppimiseen liittyvät toteutukset on tehty (Unity ML agents docs). PyTorch sisältää kaikki syvän koneoppimisen perusosat datan kanssa työskentelystä ja koneoppimismallin luomisesta mallin parametrien optimointiin ja oppimismallien tallentamiseen (PyTorch docs).

5 Tutkimuksen empiirinen osuus

Tässä kappaleessa käsitellään tutkimuksen empiiristä osuutta. Tutkimus toteutettiin empiirisenä vertailevana tutkimuksena. Vertailun kohteena olivat Unityn ML-agentin suorittama reitinhaku ja heuristiseen A*-algoritmiin perustuva reitinhaku. Kappaleessa 5.1 kuvaillaan tutkimusta yleisellä tasolla ja esitellään tutkimuksessa käytetyt työkalut. Kappaleessa 5.2 käydään läpi simulaatioympäristöt ja koneoppimiseen liittyvät konfiguraatiot. Lopuksi kappaleessa 5.3 käsitellään tulosten mittaamista ja käytettyjä suureita.

5.1 Tutkimuksen kuvaus

Tutkimuksen simulaatioalustana käytettiin Unityä, koska sen käyttö oli ennestään tuttua ja se soveltuu hyvin erilaisten tilanteiden simuloimiseen. Unityllä luotiin yksinkertaisia ruudukkoalueita, jonne sijoitettiin esteruutuja ja kävelykelpoisia ruutuja. Alueelle määrättiin alku- ja maaliruudut sattumanvaraisesti. Reitinhaun vertailua varten luotiin A*-algoritmi ja opetettiin koneoppimisagentti. Koneoppimisagentti käyttää syvää vahvistettua oppimista.

5.2 Tutkimusasetelma / Konfiguraatio

Koneoppimisagentin toiminta riippuu vahvasti sen asetuksista.

- AddObservation()-funktion avulla agentti kerää tietoa omasta tilastaan ympäristössä. Tässä tutkimuksessa agentin täytyy kerätä tietoa sen omasta sijainnista (x ja y), mahdollisesti etäisyydestä maaliin (toimisiko esim. laskea manhattan-etäisyys?) ja maalin sijainnista (x ja y). Suositeltavaa kuitenkin on, että agentti keräisi tietoa vain näkemistään asioista eli tässä tapauksessa vain omasta sijainnistaan. Näiden havaintojen lisäksi käytetään RayPerceptionSensor2D-komponenttia, joka kerää ympäröivästä alueesta tietoa säteiden avulla. Tässä projektissa agentista lähetetään 12 sädettä, jotka yltävät vain yhden ruudun verran eteenpäin. Säteet pystyvät tunnistamaan vastaantulevia objekteja, joita ovat tässä tapauksessa esteet ja maaliruutu. Näiden havaintojen avulla agentti voi toimia älykkäämmin.

- Actions, toiminnat, yksinkertaisen pelialueen takia käytetään Discrete Actions ja viittä eri

vaihtoehtoa (ylös, alas, oikealle, vasemmalle, paikallaan). Jos pelialue olisi monimutkaisempi, voidaan käyttää Continuous Actions (suunta + nopeus).

- Rewards, palkinnot, agentti saa positiivisen palkkion kun saapuu maaliin (+1). Agentti saa negatiivisen palkkion jokaisen liikkeen tai aikamääreen jälkeen (-0.01). Jos agentti törmäilee seiniin, eli yrittää liikkua suuntaan jossa on este, agentille annetaan negatiivinen palkkio (-0.001).

-

5.3 Tulosten mittaaminen

- Reitinhakutehtävästä otetaan talteen aika (sekuntia) ja onnistuiko reitinhaku vai ei (onnistui/epäonnistui).

6 Tulokset ja johtopäätökset

7 Yhteenveto

Lähteet

Abd Algfoor, Zeyad, Mohd Shahrizal Sunar ja Hoshang Kolivand. 2015. “A comprehensive study on pathfinding techniques for robotics and video games”. *International Journal of Computer Games Technology* 2015.

Botea, Adi, Bruno Bouzy, Michael Buro, Christian Bauckhage ja Dana Nau. 2013. “Pathfinding in games”. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Cui, Xiao, ja Hao Shi. 2011. “A*-based pathfinding in modern computer games”. *International Journal of Computer Science and Network Security* 11 (1): 125–130.

Demyen, Douglas, ja Michael Buro. 2006. “Efficient triangulation-based pathfinding”. Teoksessa *Aaai,* 6:942–947.

Dijkstra, Edsger W, ym. 1959. “A note on two problems in connexion with graphs”. *Numerische mathematik* 1 (1): 269–271.

Duchoň, František, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico ja Ladislav Jurišica. 2014. “Path planning with modified a star algorithm for a mobile robot”. *Procedia Engineering* 96:59–69.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel ja Sergey Levine. 2018. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. Teoksessa *International conference on machine learning,* 1861–1870. PMLR.

Hart, Peter E, Nils J Nilsson ja Bertram Raphael. 1968. “A formal basis for the heuristic determination of minimum cost paths”. *IEEE transactions on Systems Science and Cybernetics* 4 (2): 100–107.

Juliani, Arthur, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar ym. 2018. “Unity: A general platform for intelligent agents”. *arXiv preprint arXiv:1809.02627.*

“Unity”. 2022. Viitattu 4. huhtikuuta 2022. https://unity.com/.

8 Liitteet

- Kuvat tai mallinnokset pelialueesta.

- Tensorboardin kuvaajat mm. agentin palkkioiden kehityksestä.