Juho Vähäkangas

Vahvistusoppimisen soveltaminen pelitestauksessa

Tietotekniikan kandidaatintutkielma

8. toukokuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Juho Vähäkangas

Yhteystiedot: juho.e.vahakangas@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Vahvistusoppimisen soveltaminen pelitestauksessa

Title in English: Applying reinforcement learning in playtesting

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 19+0

Tiivistelmä: Tutkielma tarkastelee vahvistusoppimisen hyötyjä automaattisen pelitestauksen edistämisessä. Tarkastelussa keskitytään myös jo olemassa olevien mallien kartoittamiseen sekä niiden luomiin mahdollisuuksiin ja heikkouksiin.

Avainsanat: vahvistusoppiminen, pelitestaus

Abstract: The aim of this thesis is to survey the possibilities reinforcement learning can offer to game testing and what models have already been developed for it.

Keywords: reinforcement learning, game testing, playtesting

Kuviot

Sisällys

1 Johdanto

Viime vuosina pelien kompleksisuuden noustessa pelitestaukseen kulutetut resurssit ovat kasvaneet. Tämä on johtanut keskusteluun mahdollisesta pelitestauksen automatisoinnista. Tässä kirjallisuuskartoituksessa automaattista pelitestausta lähestytään koneoppimisen näkökulmasta. Pelitestauksessa huomiodaaan sekä pelaajan pelikokemuksen että pelin teknisen toimivuuden pysyminen tavoitellun kaltaisena. Automaattinen pelitestaus keskittyy enimmäkseen pelin teknisen toimivuuden varmistamiseen, sillä pelajaan pelikokemuksen testaaminen vaatii vielä ihmistestaajia (Albaghajati ja Ahmed 2023). Vaikka pelitestauksen automatisointi nopeuttaisi testausprosessia ja täten säästäisi pelituotannon resursseja (Albaghajati ja Ahmed 2023), peleille sopivan mallin löytäminen voi olla vaikeaa ja oman kehittäminen veisi vielä entistä enemmän resursseja (Rafi ym. 2012).

Pelitestausta tarkastellessa täytyy huomioida pelin kehittävän tahon koko sekä heidän julkaisumenetelmänsä. Esimerkiksi itsejulkaistujen pelien kehittäjistä 30% olivat maininneet testauksen menneen hyvin heidän kehitysprosessissaan, kun taas ulkopuolista julkaisijaa hyödyntäneistä kehittäjistä vain 19% kokivat näin. Yksi testaamisen epäonnistumisen syistä oli palautteen vähäinen tai vääränlainen hyödyntäminen (Washburn ym. 2016). Indiepelistudioilla pelitestauksessa tulee yleisesti vastaan budjetti, jonka takia testaus voi jäädä vähäiseksi ja sen tuottamaa palautetta ja dataa ei ehditä hyödyntää tarpeeksi kehittämisprosessin aikana (Mirza-Babaei, Moosajee ja Drenikow 2016). Pelitestauksen automatisointi indie-pelistudioille olisi tämän takia tärkeää, sillä testausprosessi nopeutuisi ja sen luoma data olisi helpommin analysoitavissa. Näille pienille studioille oman mallin kehittäminen ei kuitenkaan ole yleensä mahdollista resurssien puutteen takia, joten he joutuisivat käyttämään kolmannen osapuolen luomia ratkaisuja.

Vahvistusoppiminen on koneoppimisen osa-alue, johon kuuluvat algoritmit hyödyntävät saatavilla olevia toimintoja muuttaakseen ympäristön tilaa, päämääränään jokin ennalta-asetettu tavoite (Wiering ja Van Otterlo 2012). Vahvistusoppimista ollaan hyödynnetty jo usean erilaisen pelin, kuten lautapeli Go:n, Atari-pelikonsolin eri pelien (Mnih ym. 2013), StarCraft II (Vinyals ym. 2019), Dota 2 (*OpenAI Five*) sekä eri FPS-pelien (Lample ja Chaplot 2017), pelaamiseen. Näiden mallien ansiosta voidaan havaita vahvistusoppimisen sopivan pelaa-

misprosessin automatisointiin, joten sen hyödyntäminen pelitestauksessakin samankaltaisilla tuloksilla voi olla mahdollista.

Tämän kirjallisuuskartoituksen tarkoituksena on selvittää, miten vahvistusoppimista voidaan hyödyntää pelitestauksessa. Tämänhetkiset vahvistusoppimista hyödyntävät pelitestauksen mallit hoitavat useita pelitestauksen tavoitteita. Näihin kuuluvat esimerkiksi pelin sisäisten virheiden, eli bugien, etsintä, laadunvarmistus, uuden sisällön tuottaminen peliin sekä pelin vaikeustason määrittäminen.

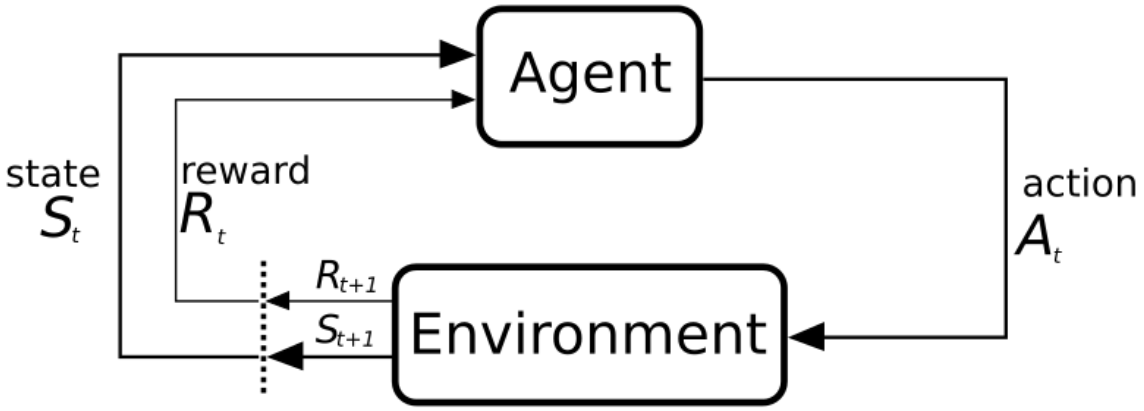
Automaattiseen pelitestaukseen löytyy useita muitakin menetelmiä koneoppimisen lisäksi, vaikkakin koneoppimista hyödyntävät mallit yrittävät yleensä täydentää kyseisten menetelmien ongelmia. Yleisin metodi on luoda pelaajadatan avulla skriptattuja agentteja, jotka matkivat pelaajan liikkeitä. Muihin metodeihin kuuluvat myös esimerkiksi Monte Carlo puuhaun (Holmgård ym. 2019) tai eri kuvantunnistusalgoritmien hyödyntämiset (Paduraru, Paduraru ja Stefanescu 2021).

Tutkielman luvussa 2 avataan kirjallisuuskartoituksessa esille tulleet keskeiset käsitteet yleisesti sekä tutkielman aiheen kontekstissa. Luvussa 3 käymme läpi vahvistusoppimisen soveltamisen pelitestauksessa. Tarkemmassa käsittelyssä on vahvistusoppimisen hyödyntäminen eri pelitestauksen vaiheissa sekä sen heikkouksien täydentäminen eri keinoin. Luvussa 4 tarkastelemme Zheng ym. 2019 kehittämää Wuji-kehystä, jota on hyödynnetty kolmessa jo julkaistussa pelissä. Tutkielman luvussa 5 käydään läpi kirjallisuuskartoituksessa esille tulleet johtopäätökset ja huomiot.

2 Käsitteet

2.1 Koneoppiminen ja vahvistusoppiminen

Koneoppimisen algoritmit perustuvat kerätyn datan perusteella muodostettuihin malleihin. Vahvistusoppiminen on taas koneoppimisen osa-alue, jossa algoritmit keräävät dataa ilman minkäänlaista ulkopuolista vaikutusta ja muodostavat keräämänsä datan perusteella malleja. Vahvistusoppimisen algoritmien päätöksenteot perustuvat muunmuassa Markovin päätösprosesseihin (Markov Decission Processes), jotka taas perustuvat järjestelmän nykyiseen tilaan ja saatavilla oleviin toimintoihin (Puterman 1990, Wiering ja Van Otterlo 2012).



Kuvio 1. Vahvistusoppimisalgoritmin toimintasykli (Wikipedia 2020)

Markovin päätösprosessin toimintasykliä kuvaa Kuvio 1. Prosessissa agentti vuorovaikuttaa ympäristön kanssa saatavilla olevilla toiminnoilla, etsiäkseen tilanmuutosten sekä tuotetun palkkion (engl. reward) avulla optimaalisen toiminnon kyseisessä ympäristön tilassa. Vahvistusoppimisen algoritmeissa muihin Markovin päätösprosesseihin eroavana tekijänä on vuorovaikutus ympäristön kanssa. Ainoana palautteena tällöin toimii skalaari, ei suuntaa antava, palkkio. Vahvistusoppimista hyödyntävät algoritmit tavoittelevat tämän palautteen maksimoimista pitkällä aikavälillä, joka toimii ympäristössä vaikuttavan agentin tavoitteena (engl. goal) (Wiering ja Van Otterlo 2012.)

Tavoitteen optimointi aiheuttaa joillekin pelitestauksessa vahvistusoppimista käyttäville kehyksille ongelman. Vahvistusoppimisen algoritmit keskittyessään tavoitteen optimointiin yrittävät hyödyntää mahdollisimman optimaalisia toimintoja. Näiden optimointiin agentin täytyy kuitenkin tutkia laajemmin saatavilla olevien toimintojen vaikutusta ympäristön tilaan. Tämä tutkiminen taas hidastaa prosessia, jolloin algoritmin tavoittelema optimaalinen suoritus heikentyy (Wiering ja Van Otterlo 2012). Tälle kompromissille laajan tutkimisen ja toimintojen hyödyntämisen välillä (engl. exploration-exploitation trade-off) esitetään yksi ratkaisu luvussa 4 käsitellyssä Wuji-kehyksessä.

2.2 Pelitestaus ja sen automatisointi

Pelitestauksesta puhuttaessa voidaan tarkoittaa kahta eri pelitestauksen osa-aluetta, pelikokemuksen ja pelisuunnittelun tai pelin teknisen toteutuksen testausta. Molemmat kyseisistä osa-alueista keskittyvät joihinkin pelin tiettyihin komponentteihin, kuten pelin ohjaimiin, pelattavuuteen, tekoälyyn tai fysiikkamoottoriin. Pelitestaus kokonaisuudessaan on koko pelin kehitysprosessin aikainen huomioonotettava prosessi, jonka unohdettaessa pelin laatu voi kärsiä huomattavasti (Washburn ym. 2016). Tavanomaisiin pelitestauksen tapoihin kuuluvat erilaisten syötekombinaatioiden testaaminen, eri toimintojen käyttö pelin eri tiloissa sekä ihmistestaajien pelikokemuksen tarkastelu.

Pelitestaus eroaa perinteisestä ohjelmistotestauksesta monin tavoin ja tämän takia pelitestauksen automatisointi ei ole yhtä helppoa tai joustavaa kuin ohjelmistotestaus. Murphy-Hill, Zimmermann ja Nagappan 2014 tuottamassa kyselyssä yksikkötestauksen ongelmakohdista nousi esille graafisen käyttöliittymän (engl. grapchical user interface, GUI) testaamisen hankaluus. Pelitestaus sisältää paljon pelaajan sekä pelin välistä vuorovaikutusta pelin graafisia käyttöliittymiä hyödyntämällä. Luvussa 4 käsittelemässämme Wuji-kehyksessä graafisten käyttöliittymien testaaminen oli jätetty kokonaan huomioimatta. Vertaamme pelitestauksen sekä ohjelmistotestauksen eroja pelisuunnittelun näkökulmasta luvussa 2.2.1 ja pelin teknisten toteutuksen näkökulmasta luvussa 2.2.2.

2.2.1 Pelisuunnittelu ja -kokemus

Pelisuunnittelua testatessa käydään läpi asioita, kuten pelin vaikeustasoa, ohjaimien intuitiivisuutta ja pelimekaniikkojen toimivuutta eri tilanteissa. Pelikokemuksen testaaminen on myös tärkeä osa testausprosessia, sillä pelaajan immersion säilyminen pitää pysyä pelaamisen aikana. Pelisuunnittelun sekä kokemuksen kannalta vahvistusoppimista on vaikea hyödyntää, sillä suurin osa testattavista osa-alueista perustuvat ihmisen tunteisiin ja ajatuksiin. Kuitenkin esimerkiksi pelin vaikeustasoa voidaan säädellä yhdistämällä syvä- sekä vahvistusoppimista ja jo olemassa olevaa pelaajadataa (Shin ym. 2020).

Suunnittelun näkökulmasta pelitestaus ja ohjelmistotestaus eroavat myös suurissa määrin. Perinteisen ohjelman suunnitelma pysyy alusta lähtien hyvin samankaltaisena, kun taas peliä kehittäessä suunnitelma voi muuttua hyvinkin paljon pelitestauksessa saadun palautteen mukaan.

2.2.2 Tekniset ongelmat sekä laadunvarmistus

Pelitestausta verrattaessa ohjelmistotestaukseen tulee vastaan vertailu niiden kompleksisuudesta. Pelitestauksessa täytyy testata peliä kokonaisuutena, sillä pelin komponentit yleensä tukeutuvat toisiinsa ja niiden testaaminen yksitellen voi olla hyvin haastavaa. Pelin teknisen toteutuksen testaus vaatii siis koko peliympäristön huomioimisen. Tämä ajaa siihen, että testaus tapahtuu suurimmilta osin peliä pelaamalla, jolloin testattava ympäristö on kompleksisuudeltaan paljon laajempi kuin perinteisessä ohjelmistotestauksessa.

Pelin kompleksisuuden lisäksi pelaajalle saatavilla oleva syöteavaruus ja sen kanssa vuorovaikuttaminen peliympäristössä luovat testaukselle mahdottoman laajan määrän testattavia järjestelmän tiloja . Näiden tilojen laaja testaus voi myös rajoittaa pelin pelattavuutta (Murphy-Hill, Zimmermann ja Nagappan 2014). Testaamattomien syötekombinaatioiden jättäminen mahdollistaa pelin sisäisten virheiden, bugien, jäämisen peliin, jolloin pelin laatu kärsii ja mahdollisen jälkituotannon aikainen testaaminen lisääntyisi. Luvussa 4 käsittelyssä olevaa Wuji-kehystä käytettiin kahteen jälkituotannnossa olevaan peliin, joista löydettiin muutamia uusia bugeja (Zheng ym. 2019).

Pelin laadunvarmistuksessa bugit jaetaan yleensä eri ryhmiin, sillä ne eroavat vaikutusalueil-

taan sekä laajuudeltaan todella paljon. Yleisimpiä jakoja ovat esimerkiksi pelin kaatumiset, pelin progression etenemisen esto sekä yleiset bugit, kuten virheet pelin grafiikoissa tai animaatioissa (Pfau, Smeddinck ja Malaka 2017; Zheng ym. 2019).

3 Vahvistusoppimisen soveltaminen pelitestauksessa

Vahvistusoppimista hyödyntävät algoritmit keskittyvät pelitestauksessa pelin tekniseen toteutukseen, eli sen pelimekaniikkojen toimivuuteen sekä bugien etsimiseen. Pelin muita osa-alueita, kuten pelikokemusta, viihdearvoa tai esteettömyyttä ei voi testata näillä keinoin juuri lainkaan ja näiden testaamiseen tarvitaan vielä tulevaisuudessakin ihmistestaajia. Tämän takia vahvistusoppimista hyödyntävät mallit eivät voisi vielä korvata tavanomaista pelitestausta täysin. (Ramadan ja Widyani 2013)

Pelikehityksessä keskeisenä osana ovat pelien tyypit eli genret sekä alustat, joille peli tullaan julkaisemaan. Pelien tyyppien monimuotoisuus ja niiden eroavaisuudet toisistaan tuovat oman haasteensa automaattisen pelitestauksen kehittämiseen. Kehyksiä on kehitetty jo sekä ensimmäisen persoonan ammuntapeleihin (Bergdahl ym. 2020) että pulmapeleihin (Shin ym. 2020), mutta suurin osa näistä kehitetyistä kehyksistä on luotu vain tietyn tyyppisille peleille. Zheng ym. 2019 kehittämä Wuji-kehys, jota käsitellään luvussa 4 tarkemmin, kykenee kolmen erilaisen pelityypin sekä alustan testaamiseen, joten tulevaisuudessa voisi olla mahdollista kehittää monipuolinen kehys, jonka tarpeista indie-pelistudioille käsiteltiin luvussa 1.

Nykypäivänä suurimmiksi pelialustoiksi ovat osoittautuneet mobiililaitteet, pelikonsolit sekä tietokoneet. Kaikki näistä kolmesta alustasta omaavat ainutlaatuisia syöttölaitteita. Käyttäjän syötteen ollessa keskuudessa pelatessa sekä pelitestauksessa, näiden syöttölaitteiden eroavuudet pitää huomioida monipuolisessa testauksessa, varsinkin jos peli tukee useaa niistä. Suurin ero syöttölaitteissa on esimerkiksi mobiililaitteiden näytön tai peliohjaimen ohjaussauvan analoginen signaali verrattuna näppäimistön digitaaliseen signaaliin.

3.1 Testaus eri kehityksen vaiheissa

Pelinkehitys prosessina koostuu monesta eri vaiheesta. Näihin kuuluvat esituotanto, tuotanto, testaaminen sekä julkaisun jälkeinen jälkituotanto ja ylläpito riippuen pelityypistä (Aleem, Capretz ja Ahmed 2016). Jälkituotanto on usein pelin julkaisun jälkeen yleisön palautteesta huolehtiminen ja mahdollinen pelin muokkaaminen niiden perusteella. Siihen myös kuuluu

”live-service”- tyyppisten pelien jatkuva sisällön tuottaminen, kuten uusien kenttien lisääminen (Shin ym. 2020, Ramadan ja Widyani 2013).

Yleisesti käytetyt vaiheet:

Esituotanto Luodaan prototyyppejä mahdollisista pelin ydinmekaniikoista ja testataan niiden toimivuutta toistensa kanssa. Testaus keskittyy pelikokemukseen eikä pelin tekniseen toteutukseen, sillä esituotannossa tuotetut pelimekaniikat eivät ole tarpeeksi hiottuja varmistakseen niiden teknisen toteutuksen pysymisen samana pelin kehityksen loppuun saakka.

Tuotanto Tuotantovaiheessa pelin ydinmekaniikat ollaan keksitty ja niiden toteutus aloitetaan. Mekaniikkoja luodessa aletaan testaamaan teknistä toteutusta ja bugien ilmetessä ne korjataan. Myös pelin kentät luodaan ja niitä testataan yksinkertaisin tavoin. Testaus tapahtuu kehityksen kanssa luonnollisesti samaan aikaan. Myös pelikokemuksen ja pelisuunnitellun testaus hoidetaan melkein loppuun tässä vaiheessa.

Pelitestaus Itse testausvaiheessa valmistaudutaan pelin julkaisuun, joten testauksen tavoitteena on laadunvarmistus. Vahvistusoppimista hyödynnettäisiin tässä vaiheessa eniten, sillä peli on kehitetty melkein kokonaan lopulliseen laajuuteensa.

Jälkituotanto Käydään läpi pelaajien palautteita ja korjataan mahdollisesti ilmenneitä bugeja. Myös riippuen pelin tyypistä voidaan aloittaa lisäsisällön tuottaminen, jolloin voidaan hyödyntää vahvistusoppimista pelin vaikeustason määrittämiseen ja testaamiseen (Shin ym. 2020). Uuden sisällön tuottaminen seuraa samaa kehityksen kulkua, joten vahvistusoppimista voidaan hyödyntää muissakin vaiheissa riippuen lisäsisällön laajuudesta.

3.2 Yhdistäminen muiden metodien kanssa

Zheng ym. 2019 käsittelevät yhtä vahvistusoppimisen huomattavimmista heikkouksista, joka ilmenee pelin bugien etsinnässä. Tämä heikkous on luvussa 2.1 käsittelemämme kompromissi pelin tilojen läpikäynnin ja optimaalisen suorittamisen välillä. Bugeja etsittäessä halutaan käydä läpi mahdollisimman laajasti pelin eri tilat, jossa vahvistusoppiminen yksinään suorituu heikosti. Kuviossa 2 näemme ainoastaan vahvistusoppimista sekä syväoppimista hyödyntävän malli suoriutuvan monimutkaisessa pelissä (NSH) kohtalaisesti. Kyseinen malli ei

kuitenkaan suoriudu lainkaan yhtä hyvin kuin Wuji-kehys, jossa hyödynnetään näiden lisäksi sekä evoluutioalgoritmeja (engl. evulotionary algorithms, EA), että monitavoiteoptimisaatiota (engl. multi-objective optimization, MOO).

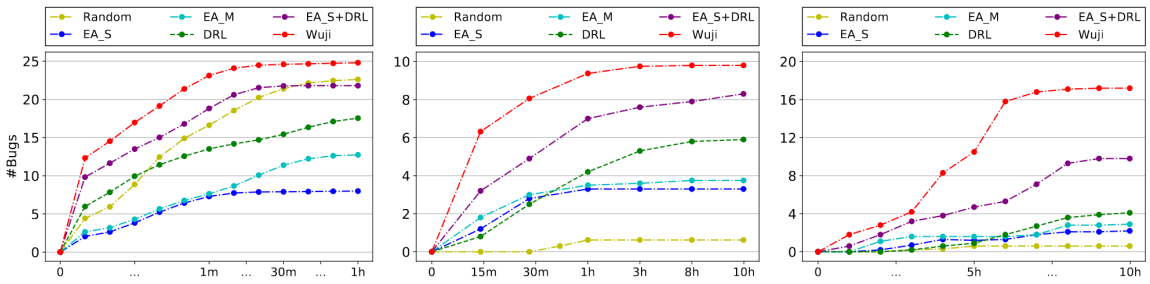
Evoluutio- ja monitavoiteoptimisaatioalgoritmien lisäksi vahvistusoppimisen heikkouksien täydentämiseen käytetään useita menetelmiä. Näitä ovat esimerkiksi aiemmin luvussa 3 sekä luvussa 4 mainitsemamme syväoppiminen (engl. deep learning) (Shin ym. 2020; Zheng ym. 2019) ja skriptattujen agenttien hyödyntäminen (Bergdahl ym. 2020). Myös hieman eri määritelmien mukaisia vahvistusoppimisen menetelmiä, kuten Q-oppimista (engl. Q-learning) tai käänteisvahvistusoppimista (engl. inverse reinforcement learning, IRL) ollaan hyödynnetty. Ferdous ym. 2023 hyödynsivät Q-oppimista kolmannen ulottuvuuden pelien testaamiseen ja Ariyurek, Betin-Can ja Surer 2021 käyttivät käänteisvahvistusoppimista yhdessä Monte Carlo -puuhaun (engl. Monte Carlo tree search, MCTS) kanssa täydentäkseen vahvistusoppimisen heikkoa järjestelmän tilojen läpikäyntiä.

4 Pelitestaus Wuji-kehyksellä

Zheng ym. 2019 kehittivät Wuji-kehyksen testaamaan pelejä niiden teknisen toteutuksen kannalta. Kehyksen tarkoitus on etsiä peleistä bugeja tutkimalla pelijärjestelmän tilat mahdollisimman laajasti. Kehys hyödyntää vahvistusoppimisen kanssa rinnalla syväoppimista, jota kutsutaan syvävahvistusoppimiseksi (engl. deep reinforcement learning, DRL). Syväoppimisen hyödyntäminen avaa kehykselle laajemman verkon luoda ”toiminto-tila”- pareja sekä ominaisuuksia oppimisen parantamiseksi.

Wujia testattiin kolmessa eri pelissä, joista yksi oli yksinkertainen kahden ulottuvuuden sokkelopeli. Kaksi muuta peliä olivat monimutkaisempia taistelupelejä, joista toinen oli kahdessa ja toinen kolmessa ulottuvuudessa pelattavia. Näiden pelivalintojen pelityyppien sekä ulottuvuuksien tärkeydestä mainitsimme tarkemmin luvussa 3. Peleiksi valittiin nämä kolme syöteavaruuden laajudeltaan huomattavissa määrin eroavaa peliä, jotta kehyksen suoriutumista voitiin verrata muihin metodeihin useassa mittakaavassa.

Vahvistus- ja syväoppimisen lisäksi Wuji hyödyntää useita eri algoritmeja täydentääkseen vahvistusoppimisen heikkouksia. DRL:n tarkoituksena on keskittyä suoritettavan tehtävän optimointiin ja palkkion maksimointiin. Lisäksi käytettyjä metodeja ovat evoluutio- ja monitavoiteoptimisaatioalgoritmit. Nämä käyvät läpi DRL:n heikkoudet, kuten järjestelmän tilojen sekä pelikentän suppean läpikäynnin.



(a) Block Maze (b) L10 (c) NSH

Kuvio 2. Wujin sekä sen yksittäisten osien vertailu keskenään (Zheng ym. 2019)

Kehyksen suoriutumista aiemmin mainituissa kolmessa pelissä voimme tarkastella kuviosta 2. Metodeja testattiin kyseisissä kolmessa pelissä luomalla uudelleen jo korjattuja bugeja, jotta kaikki menetelmät kykenisivät löytämään vertailtavan määrän bugeja. Huomattavaa kuviossa on Wujin suoriutuminen verrattuna muihin menetelmiin kompleksisuudeltaan eri tasoisissa peleissä. Verrattaessa ensimmäistä sekä viimeistä kaaviota näemmä Wujin ylläpitävän kohtalaisen suoriutumistason pelin kompleksisuuden noustessa, kun taas muut menetelmät eivät suoriudu läheskään yhtä hyvin.

5 Yhteenveto

Tutkielmassa käsittelimme vahvistusoppimisen mahdollisuuksia pelitestauksen automatisoinnissa kirjallisuuskartoituksen muodossa. Tavoitteena oli kartoittaa, millaisia malleja ja kehyksiä tähän tarkoitukseen ollaan jo kehitetty, mitä heikkouksia sekä vahvuuksia niissä on havaittavissa ja mitä mahdollisuuksia vahvistusoppimisen hyödyntämisellä on tulevaisuudessa.

Vahvistusoppimisen mahdollisuuksiin keskityimme luvussa 3. Tarkastelemalla olemassa olevia malleja havaittiin, että mallit eivät juurikaan käytä vahvistusoppimista itsessään ilman muiden metodien täydennystä. Tähän syynä oli luvussa 2.1 käsitelty kompromissi mahdollisimman laajan tilojen läpikäynnin ja optimaalisten toimintojen hyödyntämisen välillä. Pääsimme siihen lopputulokseen, että vaikka pelaaminen prosessina on hyvin samankaltainen tyypillisen Markovin päätösprosessin toimintasyklin kanssa, vahvistusoppimista hyödyntävät automaattisen pelitestauksen mallit keskittyvät pelin laadun varmistamiseen sen teknisen toteutuksen kannalta.

Luvussa 4 vertasimme muista malleista havaitsemiamme vahvistusoppimisen ominaisuuksia Wuji-kehykseen, joka kehitettiin etsimään virheitä pelin teknisestä toteutuksesta. Zheng ym. 2019 löysivät jo julkaistuista peleistä uusia bugeja, joita ihmistestaajat eivät olleet löytäneet pelin tuotantovaiheessa. Tämä toimi käytännön esimerkkinä vahvistusoppimisen hyödyistä pelitestauksen automatisoinnissa. Tutkielman havaintojen perusteella voimme nähdä, että ihmistestaajien korvaaminen vahvistusoppimista hyödyntävillä malleille on tietyissä tilanteissa mahdollista.

Lähteet

Albaghajati, Aghyad, ja Moataz Ahmed. 2023. “Video Game Automated Testing Approaches: An Assessment Framework”. *IEEE Transactions on Games* 15 (1): 81–94. https://doi.org/10.1109/TG.2020.3032796.

Aleem, Saiqa, Luiz Fernando Capretz ja Faheem Ahmed. 2016. “Critical Success Factors to Improve the Game Development Process from a Developer’s Perspective”. *Journal of Computer Science and Technology* 31, numero 5 (syyskuu): 925–950. ISSN: 1860-4749. https://doi.org/10.1007/s11390-016-1673-z.

Ariyurek, Sinan, Aysu Betin-Can ja Elif Surer. 2021. “Automated Video Game Testing Using Synthetic and Humanlike Agents”. *IEEE Transactions on Games* 13 (1): 50–67. https://doi.org/10.1109/TG.2019.2947597.

Bergdahl, Joakim, Camilo Gordillo, Konrad Tollmar ja Linus Gisslén. 2020. “Augmenting Automated Game Testing with Deep Reinforcement Learning”. Teoksessa *2020 IEEE Conference on Games (CoG),* 600–603. https://doi.org/10.1109/CoG47356.2020.9231552.

Ferdous, Raihana, Fitsum Kifetew, Davide Prandi ja Angelo Susi. 2023. “Towards Agent-Based Testing of 3D Games Using Reinforcement Learning”. Teoksessa *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering.* New York, NY, USA: Association for Computing Machinery. ISBN: 9781450394758. https://doi.org/10.1145/3551349.3560507.

Holmgård, Christoffer, Michael Cerny Green, Antonios Liapis ja Julian Togelius. 2019. “Automated Playtesting With Procedural Personas Through MCTS With Evolved Heuristics”. *IEEE Transactions on Games* 11 (4): 352–362. https://doi.org/10.1109/TG.2018.2808198.

Lample, Guillaume, ja Devendra Singh Chaplot. 2017. “Playing FPS games with deep reinforcement learning”. Teoksessa *Proceedings of the AAAI Conference on Artificial Intelligence,* nide 31. 1.

Mirza-Babaei, Pejman, Naeem Moosajee ja Brandon Drenikow. 2016. “Playtesting for Indie Studios”. Teoksessa *Proceedings of the 20th International Academic Mindtrek Conference,* 366–374. AcademicMindtrek ’16. Tampere, Finland: Association for Computing Machinery. ISBN: 9781450343671. https://doi.org/10.1145/2994310.2994364.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra ja Martin Riedmiller. 2013. “Playing atari with deep reinforcement learning”. *arXiv preprint arXiv:1312.5602.*

Murphy-Hill, Emerson, Thomas Zimmermann ja Nachiappan Nagappan. 2014. “Cowboys, Ankle Sprains, and Keepers of Quality: How is Video Game Development Different from Software Development?” Teoksessa *Proceedings of the 36th International Conference on Software Engineering,* 1–11. ICSE 2014. Hyderabad, India: Association for Computing Machinery. ISBN: 9781450327565. https://doi.org/10.1145/2568225.2568226.

OpenAI Five. https://openai.com/research/openai-five-defeats-dota-2-world-champions.

Paduraru, Ciprian, Miruna Paduraru ja Alin Stefanescu. 2021. “Automated game testing using computer vision methods”. Teoksessa *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW),* 65–72. https://doi.org/10.1109/ASEW52652.2021.00024.

Pfau, Johannes, Jan David Smeddinck ja Rainer Malaka. 2017. “Automated Game Testing with ICARUS: Intelligent Completion of Adventure Riddles via Unsupervised Solving”. Teoksessa *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play,* 153–164. CHI PLAY ’17 Extended Abstracts. Amsterdam, The Netherlands: Association for Computing Machinery. ISBN: 9781450351119. https://doi.org/10.1145/3130859.3131439.

Puterman, Martin L. 1990. “Chapter 8 Markov decision processes”. Teoksessa *Stochastic Models,* 2:331–434. Handbooks in Operations Research and Management Science. Elsevier. https://doi.org/https://doi.org/10.1016/S0927-0507(05)80172-0.

Rafi, Dudekula Mohammad, Katam Reddy Kiran Moses, Kai Petersen ja Mika V. Mäntylä. 2012. “Benefits and limitations of automated software testing: Systematic literature review and practitioner survey”. Teoksessa *2012 7th International Workshop on Automation of Software Test (AST),* 36–42. https://doi.org/10.1109/IWAST.2012.6228988.

Ramadan, Rido, ja Yani Widyani. 2013. “Game development life cycle guidelines”. Teoksessa *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS),* 95–100. https://doi.org/10.1109/ICACSIS.2013.6761558.

Shin, Yuchul, Jaewon Kim, Kyohoon Jin ja Young Bin Kim. 2020. “Playtesting in Match 3 Game Using Strategic Plays via Reinforcement Learning”. *IEEE Access* 8:51593–51600. https://doi.org/10.1109/ACCESS.2020.2980380.

Washburn, Michael, Pavithra Sathiyanarayanan, Meiyappan Nagappan, Thomas Zimmermann ja Christian Bird. 2016. “What Went Right and What Went Wrong: An Analysis of 155 Postmortems from Game Development”. Teoksessa *Proceedings of the 38th International Conference on Software Engineering Companion,* 280–289. ICSE ’16. Austin, Texas: Association for Computing Machinery. ISBN: 9781450342056. https://doi.org/10.1145/2889160.2889253.

Wiering, Marco A, ja Martijn Van Otterlo. 2012. “Reinforcement learning”. *Adaptation, learning, and optimization* 12 (3): 729.

Wikipedia. 2020. *Reinforcement learning cycle.* https://commons.wikimedia.org/wiki/File:Markov_diagram_v2.svg.

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev ym. 2019. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. *Nature* 575 (7782): 350–354.

Zheng, Yan, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen ja Changjie Fan. 2019. “Wuji: Automatic Online Combat Game Testing Using Evolutionary Deep Reinforcement Learning”. Teoksessa *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE),* 772–784. https://doi.org/10.1109/ASE.2019.00077.