Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

David Penca

Strojno učenje obnašanja inteligentnih agentov v računalniških igrah

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Zoran Bosnić

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Kandidat naj v diplomski nalogi uporabi igralni pogon Unreal Engine 4 in z metodami strojnega učenja naj implementira igralca v spopadu, ki se je zmožen učiti igralne strategije. V diplomskem delu naj predstavi sodobne pristope, ki jih za to nalogo uporabljajo proizvajalci podobnih iger. Implementacije naj se loti z vsaj dvema metodama s področja strojnega učenja (npr. s spodbujevanim učenjem in z genetskimi algoritmi). V zaključku naj poda primerjavo uspešnosti učenja s predlaganimi pristopi.

Rad bi se zahvalil svojemu mentorju izr. prof. dr. Zoranu Bosniću za strokovno svetovanje pri izdelavi diplomskega dela.

Iskreno se zahvaljujem tudi mami in očetu za podporo med študijem.

Kazalo

Povzetek

Naslov: Strojno učenje obnašanja inteligentnih agentov v računalniških igrah

V diplomskem delu predstavljamo pristop programiranja igralcev v večigralskih spletnih igrah, ki temelji na metodah strojnega učenja. Pokazati želimo, da lahko posameznim likom določimo poteze, ki jih lahko izvajajo, jim podamo informacije o njihovem okolju in jih prepustimo, da si na podlagi bojev s človeškimi igralci ustvarijo igralno taktiko. Pristopi, ki temeljijo na sprotnem strojnem učenju taktik likov, lahko zmanjšajo čas, porabljen za programiranje, hkrati pa omogočajo prilagajanje nasprotnikov taktikam igralcev brez dodatnega dela programerjev. Tako dobimo igralce, ki se čez čas izboljšujejo in so robustni na izkoriščanje uveljavljenih taktik s strani človeškega igralca. Osredotočili smo se na spodbujevano učenje in na evolucijske algoritme, saj sta oba pristopa primerna za sisteme, ki se učijo na podlagi številnih interakcij s človeškimi nasprotniki. Naše rešitve smo implementirali v igralnem pogonu Unreal Engine 4.

Ključne besede: strojno učenje, Q-učenje, genetski algoritmi, računalniške igre, inteligentni agenti.

Abstract

Title: Machine learning of character behavior in computer games

In our thesis we present an approach for programming enemy characters in online multiplayer games that is based on machine learning algorithms. We wish to demonstrate, that it is possible to specify the available actions for specific characters, implement sensing of their environment and let them learn the tactics on their own, by fighting human players. Approaches based on machine learning have the potential to reduce the time needed for programming as well as enable the characters to adapt to current player tactics, without any additional programming. By using such programming methods we are able to create characters which get better over time and are not vulnerable to exploitation of established tactics by the players. We have focused mainly on reinforcement learning and evolutionary algorithms, because both approaches are suitable for use in systems that learn from numerous interactions with human players. We have implemented our prototype in the Unreal Engine 4 game engine.

Keywords: machine learning, Q-learning, genetic algorithms, computer games, inteligent agents.

Poglavje 1

Uvod in motivacija

Kljub številnim tehnološkim napredkom v računalništvu je del razvoja računalniških iger od njihovega samega začetka ostal nespremenjen. Govorimo o programiranju obnašanja računalniško nadzorovanih likov v računalniških igrah. V diplomskem delu smo se osredotočili na like, ki se borijo proti človeškemu igralcu.

V spletnih računalniških igrah, v katerih igralci medsebojno tekmujejo, kdo bo hitreje premagal računalniško kontroliranega sovražnika, se pogosto pojavijo taktike, ki izkoriščajo napake v vedenju nasprotnika. Takšne napake najpogosteje nastanejo zaradi ročnega programiranja vedenja likov. Ker si igralci na forumih medsebojno delijo informacije, se hitro zgodi, da velika večina igralcev nasprotnika premaga z isto strategijo, saj tako hitreje in varneje pridejo do nagrade. Naše mnenje je, da takšno slepo sledenje optimalni strategiji zmanjša zabavo med igranjem in izniči občutek dosežka, ko igralec uspešno premaga nasprotnika. Enako mnenje imajo tudi razvijalci iger, zato svoje igre posodabljajo z željo, da bi odpravili takšne napake. Ti popravki pogosto pridejo prepozno ali pa nasprotnika naredijo skoraj nepremagljivega in tako pokvarijo ravnovesje v igri.

V diplomskem delu želimo ustvariti prototip sistema za obnašanje računalniško

kontroliranih likov, ki bi bil robusten na izkoriščanje optimalnih strategij. To nameravamo doseči tako, da implementiramo sistem za obnašanje likov na način, ki bo omogočal prilagajanje aktualnim strategijam igralcev brez posredovanja programerja.

V sklopu diplomskega dela želimo implementirati sistem, ki omogoča strojno učenje strategij nasprotnikov v računalniških igrah med igro proti človeškemu igralcu. Stremimo k sistemu, ki bo odporen proti izkoriščanju optimalnih strategij, nagrajeval pa bo izvirne strategije s strani igralcev. Zadane cilje skušamo doseči z uporabo algoritmov strojnega učenja na način, ki bo omogočal učenje inteligentnih agentov na posplošenih podatkih, zbranih v številnih bojih s človeškimi igralci. Prav zaradi posplošitve učnih podatkov skozi veliko število iger smo dosegli želene cilje. Če veliko število igralcev izkorišča optimalno strategijo, se agent hitro prilagodi in nauči premagati to strategijo. V primerih, ko igralcem uspe premagati agenta s strategijo, ki je zgolj njihova in je drugi igralci ne poznajo, pa se ji agent zaradi posplošitve učnih podatkov ne more prilagoditi. Naš sistem kaznuje igralce, ki izkoriščajo uveljavljene taktike, in nagrajuje tiste, ki uporabljajo lastne strategije.

Rezultat našega diplomskega dela je prototip tridimenzionalne računalniške igre, narejen v igralnem pogonu Unreal Engine 4. Igra ponazarja spopad med identičnima igralcema, ki imata na voljo 9 potez. V igri sta implementirana dva strojno učena lika in en ročno programirani lik, ki služi učenju in ocenjevanju uspešnosti sistemov, naučenih z uporabo metod strojnega učenja. Za učenje inteligentnih agentov smo uporabili metodo spodbujevanega učenja, natančnejše Q-učenje in metodo učenja z genetskim algoritmom.

Verjamemo, da bo naše diplomsko delo v pomoč razvijalcem iger, ki želijo v svoje igre vključiti strojno učene inteligentne like, ki se bodo učili na podlagi številnih bojev z več igralci. Prav tako pa bo koristno tistim, ki želijo uporabiti strojno učenje za reševanje problemov v stohastičnem okolju.

Poglavje 2

Pregled področja

2.1 Strojno učenje v računalniških igrah

Trenutno je na trgu veliko iger, ki uporabljajo like, naučene z metodami strojnega učenja. Večinoma gre za probleme s popolno informacijo in z izmeničnimi potezami dveh igralcev. Spremenljivke, pomembne za strojno učenje, so v takšnih problemih diskretne, kar močno poenostavi učenje, ki je v takšnih igrah implementirano kot preiskovanje prostora stanj med nasprotnikoma. Ker imamo v problemu, ki ga skušamo rešiti v diplomskem delu, opravka z zveznimi spremenljivkami in ker bo naš inteligentni agent deloval v stohastičnem okolju, takšne metode strojnega učenja za rešitev našega problema niso primerne.

Čeprav so dokaj redke, obstajajo tudi računalniške igre, ki uporabljajo za iskanje strategije v igrah s stohastičnim okoljem strojno učenje. Pogosto so omenjena okolja zelo podobna okolju, v katerem bo deloval naš inteligentni agent. Implementacije strojnega učenja v igrah s stohastičnim okoljem najpogosteje delimo na dve skupini, glede na to, kdaj se v igri izvaja učenje:

- implementacije, pri katerih do strojnega učenja pride pred izidom igre,
- implementacije, v katerih do strojnega učenja pride med samim igranjem igre.

2.1.1 Učenje pred izidom igre

Strojno učenje je možno med nastajanjem igre uporabiti za učenje strategij računalniško kontroliranih likov. Omenjeni pristop v današnjih igrah s stohastičnim okoljem ni pogost, saj ročno programiranje likov zagotavlja bolj konsistentne rezultate, hkrati pa je dosti bolj preprosto.

Čedalje pogosteje pa se strojno učenje uporablja pri generiranju računalniško ustvarjenih okolij. Pod generiranjem okolij vključujemo generiranje terena, postavitev vodnih teles kot tudi raznega rastlinja, živali in drugih lastnosti okolja, kot so razni predmeti, stavbe in liki. Učenje generiranja okolja za igro Super Mario Bros podrobneje predstavijo Volz in sod. [13], v članku avtorji opišejo postopek generiranja novih stopenj za igro iz nabora ročno ustvarjenih stopenj. Sistem deluje z dvema nevronskima mrežama, ki tekmujeta med seboj in se tako medsebojno izboljšujeta; ena izmed mrež naključno generira možne stopnje, druga pa na podlagi podrobnosti z naborom veljavnih stopenj ocenjuje njihovo primernost. Enak postopek so uporabili Giacomello, Lanzi in Loiacono [7], ki so na podlagi originalne igre uspešno generirali nove naključne stopnje za igro DOOM.

2.1.2 Učenje med igro

Implementacije vedenja nasprotnikov v računalniških igrah, ki med igro uporabljajo strojno učenje, da svojo taktiko prilagodijo taktiki igralca, so v računalniških igrah zelo redke. Razvijalci iger se jih izogibajo, saj so v svoji nalogi preveč učinkovite, preveč dobro se znajo prilagoditi nasprotniku in tako hitro postanejo za igralca nepremagljiv izziv. Kako dobri znajo biti strojno naučeni agenti, je najbolje pokazalo podjetje OpenAI, ko je leta 2017 njihov inteligentni agent v igri Dota 2 premagal najboljše svetovne igralce [1]. Dota 2 velja za eno izmed najkompleksnejših računalniških iger. Inteligentni agent podjetja OpenAI se je naučil igrati, tako da je nenehno igral proti svoji identični kopiji in se tako postopoma izboljševal. Veliko bolj kot

za učenje obnašanja nasprotnikov v računalniških igrah se strojno učenje uporablja za reševanje zahtevnih problemov, ki jih ljudje pogosto ne moremo rešiti. Na področju računalniških iger so takšni problemi iskanja optimalne igralne strategije. Namesto človeškega igralca lahko v igro postavimo inteligentnega agenta, ki se uči s pomočjo strojnega učenja, takšen inteligentni agent se pogosto lahko zelo približa optimalni igralni strategiji.

2.2 Sorodna dela s področja strojnega učenja v računalniških igrah

V temu poglavju so našteta in na kratko povzeta sorodna dela na področju stojnega učenja v računalniških igrah.

2.2.1 Učenje z evolucijskimi nevronskimi mrežami

Stanley, Bryant in Miikkulainen [10] predstavijo metodo za razvijanje nevronskih mrež za upravljanje inteligentnih agentov v računalniški igri NERO. Gre za sistem, ki se uči med igro z genetskimi algoritmi. Sistem temelji na prilagodljivem sistemu kodiranja povezav v nevronskih mrežah, kar omogoča učenje uteži z genetskimi algoritmi. V igri posamezne inteligentne agente kontrolirajo nevronske mreže, ki se v življenjskem ciklu agenta ne spreminjajo. Agenti so razdeljeni v vrste glede na medsebojno podobnost, sistem skuša držati stalno število vrst, saj tako zagotavlja inovacije v novih članih populacije. Po koncu življenjskega cikla agentov oziroma po zaključenem ciklu učenja se ovrednoti ustreznost posameznih agentov in se na podlagi ustreznosti agentov določi ustreznost vrste. Na podlagi ustreznosti se izbere starševske vrste za nove generacije. Sistem je namenjen za igro NERO, ki pripada relativno novi vrsti računalniških iger, zgrajenih na podlagi strojnega učenja. Igralci so postavljeni v vlogo trenerja, ki mora naučiti strojno učene agente izpolniti zadano nalogo.

2.2.2 Strojno učenje v prvoosebnih strelskih igrah

Prvoosebne strelske igre spadajo med najtežja okolja za strojno učene agente. Pogosto tudi najboljši človeški igralci ne znajo pojasniti strategije, ki jo uporabljajo. Prav problematiko strojnega učenja inteligentnih agentov v prvoosebnih strelskih igrah raziskuje Geisler [6]. V članku avtor opiše postopek, s katerim je bil naučen sistem za premikanje inteligentnega agenta v prvoosebni strelski igri Soldier of Fortune 2 [2]. Inteligentni agent se je učil na podlagi velikega nabora informacij, pridobljenih iz iger ekspertnega igralca. Za metodo strojnega učenja so bile izbrane nevronske mreže, učenje pa je potekalo v obliki odkrivanja uteži povezav med posameznimi nevroni v nevronskih mrežah. Izkazalo se je, da so kompleksne odločitve človeškega igralca zelo težko posnemali z eno nevronsko mrežo. Veliko boljše podatke so dobili pri učenju z ansambli. Najuspešnejši so bili z metodo boosting [5], pri posnemanju kompleksnega človeškega vedenja pri igranju igre so dosegli 95-odstotno klasifikacijsko točnost.

2.2.3 Motivirano spodbujevano učenje

Eno izmed priljubljenih zvrsti računalniških iger predstavljajo igre, v katerih igralci nimajo točno določenih pogojev zmage in poraza. Namesto da jih igra usmerja k nekim določenim ciljem, si morajo igralci v takšnih igrah cilje ustvariti sami. Takšne igre se pogosto odvijajo v obširnih svetovih, v katerih imajo igralci veliko možnosti spreminjati okolje in dodajati nove predmete ter računalniško kontrolirane like v igro. Problem učenja obnašanja inteligentnih likov v takšnih računalniških igrah opisujeta Merrick in Maher [8]. V članku avtorja opišeta sistem za učenje obnašanja inteligentnih agentov, ki omogoča prilagajanje obnašanja inteligentnih agentov objektom v njihovem okolju. Sistem temelji na motiviranem spodbujevanem učenju, ki podpira večciljno učenje v dinamičnem svetu, hkrati pa spodbuja ciklično obnašanje agenta. Zaradi večciljnosti in spreminjajočega sveta so se avtorji odločili za opis igralnega sveta z množico markovskih procesov odločanja. Takšen

opis sveta omogoča dodajanje novih ciljev in spreminjanje obstoječih ciljev brez vpliva na preostale cilje. Zaradi inovativnega opisa igralnega sveta so avtorji morali implementirati tudi lasten sistem nagrajevanja agentov in izbiranja stanj. Gre za klasične sisteme spodbujevanega učenja, ki so prilagojeni večciljnim sistemom. Menimo, da ima sistem, opisan v omenjenem članku, zelo velik potencial za nadaljnji razvoj na področju inteligentnih agentov, katerih obnašanje je odvisno od predmetov v njihovem okolju.

Poglavje 3

Uporabljena metodologija

Za razvoj prototipa računalniške igre smo uporabili igralni pogon Unreal Engine 4 [3], saj je eden izmed najbolj naprednih prosto dostopnih igralnih pogonov. Programski jezik, ki ga bomo uporabili, je C++. Za samo strojno učenje smo uporabili metode spodbujevanega učenja in evolucijske algoritme, ki smo jih v celoti implementirali v programskem jeziku C++ in jih vključili v igralni pogon, tako smo zagotovili interakcijo med inteligentnimi agenti in igralnim prostorom.

3.1 Unreal Engine 4

Unreal Engine 4 je igralni pogon v lasti podjetja Epic Games, med izdelavo diplomskega dela je to najzmogljivejši prosto dostopni igralni pogon. Unreal Engine 4 razvijalcem predstavlja celoten skup razvojnih orodij, potrebnih za izdelavo iger. Pri razvijanju iger z omenjenim igralnim pogonom razvijalci programirajo v poljubnem urejevalniku besedil in v jeziku C++. Poleg vseh funkcionalnosti jezika C++ imajo programerji na voljo še knjižnice, vključene v igralni pogon. Te imajo zelo širok nabor funkcij, tipov in programskih vmesnikov, ki so ključni za uspešen razvoj iger. V pogon je vključenih 14 ločenih urejevalnikov, ki omogočajo razvijalcem nadzor nad posameznimi deli igre.

3.2 Uporabljene metode strojnega učenja

Strojnega učenja strategij računalniško nadzorovanih likov smo se lotili na dva načina: Odločili smo se za spodbujevano učenje in učenje na podlagi evolucijskih algoritmov. Kot pristop k spodbujevanemu učenju smo izbrali Q-učenje, kot metodo evolucijskih algoritmov pa genetske algoritme. Omenjena pristopa smo izbrali zaradi njune primernosti za implementacijo v sistemih, v katerih poteka učenje na velikem številu že zbranih podatkov. Izbrani metodi predstavljata popolnoma različna pristopa reševanja istega problema, obe skušata odkriti optimalno strategijo za boj proti ročno sprogramiranemu agentu. Podrobnejši opis Q-učenja in genetskih algoritmov in opis implementacije omenjenih metod na našem problemu sledi v nadaljnjih poglavjih.

3.2.1 Q-učenje

Q-učenje je metoda spodbujevanega učenja, ki jo je prvi predstavil Christopher John Cornish Hellaby Watkins v svoji doktorski disertaciji [14]. Q-učenje temelji na Q-funkciji, ki za vsako stanje in izbrano potezo vrne nagrado. Ta funkcija predstavlja celotno pridobljeno znanje nekega inteligentnega agenta.

Q-funkcijo najpogosteje predstavimo v obliki Q-tabele, gre za dvodimenzionalno tabelo pričakovanih nagrad za izbrane poteze v vseh možnih stanjih. Pri Q-učenju je postopek učenja predstavljen s posodabljanjem Q-tabele. Učenje začnemo tako, da Q-tabelo inicializiramo na neko izbrano vrednost, po navadi je to vrednost 0. Po inicializaciji agent v vsakem koraku izbere potezo, za katero dobi nagrado, in posodobi Q-tabelo. Pri izbiranju poteze lahko agent izbere enega izmed dveh pristopov. V nekem stanju lahko agent izbere naključno potezo, temu pravimo raziskovanje. Poleg raziskovanja se agent lahko odloči za izkoriščanje, ki se zgodi, ko agent na podlagi Q-tabele izbere najboljšo potezo za trenutno stanje. Učenje je pametno začeti z zelo veliko stopnjo raziskovanja in med učenjem Q-tabele počasi zmanjševati sto-

pnjo raziskovanja ter povečevati stopnjo izkoriščanja. Za posodabljanje Q-tabele uporabljamo Bellmanovo enačbo 3.2.1.

$$\boldsymbol{New}\mathbf{Q(s,a)} = \mathbf{Q(s,a)} + \alpha[\mathbf{R(s,a)} + \beta\boldsymbol{max}\mathbf{Q(s',a')} - \mathbf{Q(s,a)}],$$

kjer **Q(s,a)** predstavlja vrednost Q-tabele v stanju **s** in potezi **a**, **R(s,a)** predstavlja nagrado za izbrano potezo **a** v stanju **s**, $\boldsymbol{max}$**Q(s',a')** pa predstavlja maksimalno pričakovano nagrado za stanje **s'**. Grška črka α predstavlja stopnjo učenja, črka β pa faktor zniževanja. Kot je razvidno iz enačbe 3.2.1, stopnja učenja določa, kako hitro novi podatki prepisujejo stare, faktor zniževanja pa določa pomembnost prihodnjih nagrad, torej nam pove, v koliko korakih želi algoritem maksimizirati prejeto nagrado.

Po končanem učenju lahko s pridobljenim znanjem za vsako stanje določimo najboljšo potezo glede na pričakovano nagrado. Tako inteligentnemu agentu določimo optimalno strategijo.

3.2.2 Genetski algoritmi

Genetski algoritem je hevristična metoda strojnega učenja, ki spada pod evolucijske algoritme, to so algoritmi, ki skušajo posnemati evolucijske procese, ki se dogajajo v naravi. Zametke današnjih evolucijskih algoritmov je leta 1951 opisal Alan Turing [11]. Tako kot vsi evolucijski algoritmi tudi genetski algoritmi temeljijo na načelu naravne selekcije, ki pravi, da bodo najbolj prilagojeni osebki z največjo verjetnostjo prenesli svoje lastnosti na naslednjo generacijo.

Učenje z metodo genetskih algoritmov se začne z začetno populacijo, gre za skupek naključno generiranih rešitev problema. Vsak osebek v populaciji je zakodirana rešitev problema, najpogosteje rešitve kodiramo binarno. Za dani problem definiramo uspešnostno funkcijo, ki lahko za vsakega izračuna ustreznost pri reševanju problema. Ko imamo enkrat definirano začetno populacijo in uspešnostno funkcijo, lahko začnemo z učenjem. Najprej vsakemu

izmed osebkov v začetni populacij ocenimo uspešnost z uporabo uspešnostne funkcije. Nato iz populacije izberemo osebke, ki bodo služili kot starši naslednji generaciji, pri izbiri staršev imajo uspešnejši osebki večjo verjetnost, da bodo izbrani. Po izbiri staršev določimo točko križanja, to je točka, kjer se v potomcih zamenjata genetska zapisa staršev, v določenih implementacijah lahko izberemo tudi več točk križanja. Po križanju pride na vrsto mutiranje, v stopnji mutiranja vsak gen potomcev z neko majhno verjetnostjo mutiramo v drugega. Prav mutacije nam omogočajo raziskovanje celotnega prostora rešitev. Ko končamo z mutiranjem, smo dobili začetno populacijo za novo stopnjo učenja. Omenjene procese ponavljamo, dokler rešitve ne začnejo konvergirati.

Poglavje 4

Implementacija inteligentnega igralnega agenta

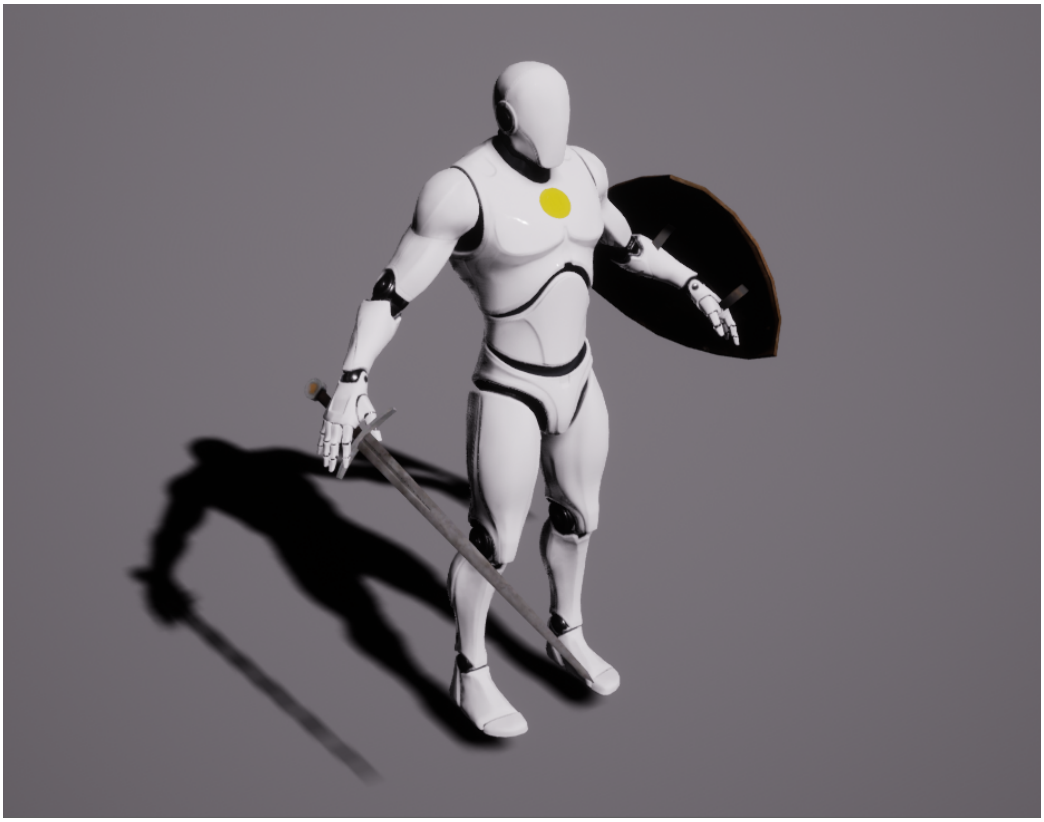
Za razvoj sistema za strojno učenje smo morali definirati svet, v katerem bodo delovali naši inteligentni agenti. Odločili smo se za tridimenzionalno računalniško igro, v kateri se igralec in nasprotnik spopadeta v dvoboju.

4.1 Cilj igre

V igri lika tekmujeta med seboj. Cilj igre je premagati nasprotnika tako, da ga z napadi poškodujemo in mu izčrpamo življenjske točke. Boj se lahko konča z zmago enega izmed likov ali z izenačenjem. Izenačenje se zgodi, če oba igralca hkrati ostaneta brez življenjskih točk.

4.2 Opis lika

Tako igralec, kot nasprotnik upravljata identičnega lika, oba imata na voljo iste poteze, premikata se z enako hitrostjo in imata ob začetku igre isto število življenjskih točk. Liki so opremljeni z mečem, s katerim lahko poškodujejo nasprotnika. Kako je lik implementiran v našem prototipu, je razvidno s slike 4.1.



Slika 4.1: Posnetek lika v igralnem okolju

4.3 Ročno programirani lik

Za učenje strategije strojno učenih likov smo ustvarili ročno programiranega agenta, ki ima na voljo iste poteze kot agenti, katerih strategijo smo določili s strojnim učenjem, vendar smo njegovo taktiko določili sami. Več ročno programiranih agentov smo pomerili med seboj, za učenje in testiranje algoritmov strojnega učenja pa smo izbrali najuspešnejšega. Za takšnega se je izkazal lik, ki se nasprotniku počasi približuje, ko pa zazna, da je v dosegu, začne napadati. Če zazna napad nasprotnika, se izmakne, smer izmikanja je vsakič naključno določena.

4.4 Inteligentni agent na podlagi Q-učenja

4.4.1 Začetna hipoteza

Predvidevamo, da se bo inteligentni agent, učen z metodo Q-učenja, uspešno naučil premagati najboljšega nasprotnika, ki nam ga je uspelo ročno sprogramirati. Učenje bomo obravnavali kot uspešno, če bo inteligentni agent po koncu učenja premagal ročno programiranega agenta v vsaj devetdeset odstotkov iger.

4.4.2 Implementacija

Sistem, ki s Q-učenjem skuša odkriti optimalno taktiko, smo v sklopu svoje igre implementirali v obliki posebnega razreda, katerega nalogi sta učenje optimalne strategije inteligentnega agenta in kontroliranje lika v računalniški igri. Sistem kot vhod dobi enega izmed desetih možnih stanj, opisanih v tabeli 4.1, in nagrado za zadnjo izbrano potezo, kot izhod pa poda izbrano potezo; možne poteze so predstavljene v tabeli 4.2. Tabela 4.3 predstavlja sistem za nagrajevanje inteligentnega agenta ob koncu vsake poteze.

Stanje	**Obrazložitev**
stanje 0	Agent je daleč od nasprotnika, nasprotnik ne napada
stanje 1	Agent je daleč od nasprotnika, nasprotnik napada
stanje 2	Agent je blizu nasprotniku, nasprotnik ne napada
stanje 3	Agent je blizu nasprotniku, nasprotnik začenja napad
stanje 4	Agent je blizu nasprotniku, nasprotnik napada
stanje 5	Agent je blizu nasprotniku, nasprotnik končuje napad
stanje 6	Agent je srednje blizu nasprotniku, nasprotnik ne napada
stanje 7	Agent je srednje blizu nasprotniku, nasprotnik začenja napad
stanje 8	Agent je srednje blizu nasprotniku, nasprotnik napada
stanje 8	Agent je srednje blizu nasprotniku, nasprotnik končuje napad

Tabela 4.1: Stanja, v katerih se lahko znajde agent

Poteza	**Obrazložitev**
Premik v levo	Lik se premakne v levo s polno hitrostjo
Premik v desno	Lik se premakne v desno s polno hitrostjo
Napad	Lik napade; če že napada, izvede drugi napad
Izmik nazaj	Lik se izmakne nazaj
Izmik naprej	Lik se izmakne naprej
Izmik v desno	Lik se izmakne v desno
Izmik v levo	Lik se izmakne v levo
Premik naprej	Lik se premakne naprej, proti nasprotniku
Premik nazaj	Lik se premakne nazaj, stran od nasprotnika

Tabela 4.2: Poteze lika v igri

Nagrada	**Obrazložitev**
nagrada 0	Med agentom in nasprotnikom ni bilo kontakta
nagrada 1	Agent je ranil nasprotnika
nagrada -1	Agent je bil ranjen s strani nasportnika
nagrada 10	Agent je zmagal igro
nagrada -10	Agent je izgubil igro
nagrada -50	Agent se je oddaljil predaleč od nasprotnika

Tabela 4.3: Sistem za nagrajevanje inteligentnega agenta

V našem sistemu smo Q-tabelo implementirali kot dvodimenzionalno tabelo decimalnih števil. Po vsaki končani igri se Q-tabela shrani na disk, pri začetku nove igre pa se Q-tabela iz diska naloži v delovni pomnilnik računalnika, kar nam omogoča, da učenje izvajamo skozi več zaporednih iger.

Ker mora naš inteligentni agent delovati v stohastičnem okolju, kjer ne moremo natančno vedeti, v katerem stanju se bo inteligentni agent znašel po izvedbi neke akcije, poleg Q-tabele posodabljamo še tabelo prehodov med stanji. Gre za tridimenzionalno tabelo, ki za vsako stanje in izbrano akcijo

hrani verjetnosti, da bo agent pristal v enem izmed možnih stanj. Tako za posodabljanje Q-tabele dobimo naslednjo Bellmanovo enačbo:

$$\boldsymbol{New}\mathbf{Q(s,a)} = \mathbf{Q(s,a)} + \alpha[\mathbf{R(s,a)} + \Sigma(\mathbf{P(s')}*\beta\boldsymbol{max}\mathbf{Q(s',a')}) - \mathbf{Q(s,a)}]$$

Kjer **P(s')** predstavlja verjetnost, da bo inteligentni agent pristal v stanju **s'**.

Po vsaki akciji naš sistem posodobi Q-tabelo in tabelo prehodov med stanji. Če je igra končana, torej da je eden izmed igralcev izgubil, ali če je prišlo do izenačenja, agent po posodabljanju tabel shrani obe tabeli na disk, prekine trenutno igro in zažene novo. Poenostavljeno delovanje sistema za učenje inteligentnega agenta z uporabo Q-učenja lahko enostavno predstavimo s psevdokodo:

```
begin
load Q-table
load StateTransitionTable
while current game is not finished
    if current action is finished
        update Q-table(Action, PreviousState)
        update StateTransitionTable(Action, PreviousState, State)
        choose a new value for Action
decrease LearningRate and ExplorationRate
save Q-table
save StateTransitionTable
restart the program
```

4.4.3 Rezultati

V našem sistemu za Q-učenje je Q-tabela začela konvergirati po petdesetih igrah, vendar smo zaradi doslednosti učenje nadaljevali. Po petsto igrah smo učenje zaključili in začeli testiranje inteligentnega agenta. Testiranje je bilo podobno učenju, vendar nismo več posodabljali Q-tabele in tabele prehoda

stanj, temveč smo zgolj šteli, v koliko igrah je inteligentni agent zmagal. Od tisoč iger je inteligentni agent zmagal v 996 primerih, izgubil v 4, izenačenj pa ni bilo. Po koncu učenja je naš inteligentni agent dosegel uspešnost, večjo od 99,6 %.

Poleg osupljive uspešnosti inteligentnega agenta smo opazili tudi zanimivo podrobnost pri njegovem obnašanju. Agent v boju proti nasprotniku praktično ni ponavljal potez, torej se skoraj nikoli ni zgodilo, da bi agent večkrat zapored ponovil isto potezo. Verjamemo, da agent potez ni ponavljal, saj mu je to omogočilo, da prekine nekoristne dele animacij in tako manj časa ostane izpostavljen nasprotniku.

Prekinjanje animacij smo prikazali s pomočjo slik, zajetih med igro. Na slikah črni igralec predstavlja inteligentnega agenta, učenega s pomočjo Q-učenja, beli igralec pa ročno programiranega agenta, rdeče črte označujejo pot po kateri je potoval meč v času napada in služijo zgolj za lažjo vizualizacijo. Na sliki 4.2 se intelignetni agent uspešno izmakne napadu nasprotnika, slika 4.3 pa zajema dogajanje neposredno po izmiku. Takoj po izmiku nazaj inteligentni agent izvede izmik v levo, ta dodatni izmik pa mu omogoči, da predčasno prekine animacijo prvega izmika in se na ta način izogne čakanju na konec prve animacije. Poleg zmanjšanja časa ranljivosti, zaradi prekinitve nekoristnega dela prve animacije, pa ima izvedeno zaporedje akcij še dodatno prednost, da agenta postavi na ugoden položaj za napad. Podobno taktiko prekinjanja animacij uporabljajo tudi človeški igralci v računalniških igrah.

Po uspešnem izmiku in prekinitvi animacije inteligentni agent počaka, da sovražnik napade drugič, napadu se izmakne ter napade nasprotnika, ko je ta najbolj ranljiv. Tudi če je prvi napad uspešen, se agent pogosto umakne na varno ter zopet čaka na priložnost. Po vsakem napadu obstaja okno časa, ko nasprotnik ne napada več, vendar se še vedno predvaja animacija napada, ravno to kratko okno se je naučil izkoriščati naš inteligentni agent.

Slika 4.2: Uspešen izmik inteligentnega agenta

Slika 4.3: Izmik v levo, ki prekine trenutno animacijo

4.5 Inteligentni agent na podlagi genetskega algoritma

4.5.1 Začetna hipoteza

Predvidevamo, da se bo inteligentni agent, učen z metodo genetskih algoritmov, uspešno naučil premagati najboljšega ročno programiranega nasprotnika. Učenje bomo obravnavali kot uspešno, če bo inteligentni agent po koncu učenja premagal ročno programiranega agenta v vsaj devetdeset odstotkov iger.

4.5.2 Implementacija

Tako kot sistem, ki se uči s Q-učenjem, smo tudi sistem, ki se uči na podlagi genetskega algoritma, v sklopu svoje igre implementirali v posebnem razredu. Sistem skuša odkriti vektor potez, ki definira strategijo inteligentnega agenta, na podlagi katere lahko uspešno premaga ročno programiranega nasprotnika.

Za kodiranje strategije inteligentnega agenta smo izbrali enodimenzionalni vektor, dolžine 10. Vsaka celica v vektorju predstavlja eno izmed možnih stanj, v katerih se lahko znajde agent. V vsaki celici vektorja je naravno število od 0 do 8, ki predstavlja akcijo, ki jo bo inteligentni agent izbral v tem stanju.

Ker naš inteligentni agent deluje v stohastičnem okolju in ker ne poznamo rešitve problema, ustreznost osebkov v populaciji računamo tako, da agenta pustimo, da se z ocenjevano taktiko spopade z nasprotnikom, in seštevamo nagrade, ki jih je pridobil med bojem. Za vsako ocenjevano taktiko smo sešteli nagrade v treh zaporednih igrah. Uporabili smo enak sistem nagrajevanja kot pri Q-učenju, omenjeni sistem je predstavljen v tabeli 4.3.

Po oceni uspešnosti vseh primerkov populacije za starša izberemo najuspešnejša osebka in s križanjem generiramo potomce. Pri vsakem križanju naključno izberemo mesto križanja in iz staršev ustvarimo dva potomca, ki sta sestavljena iz genetskih zapisov staršev, zamenjanih v točki križanja. Nad obema potomcema izvedemo mutacijo, tako da vsak gen v genskem zapisu potomcev z neko majhno verjetnostjo nadomestimo z naključnim genom, predstavljenim s celim številom od 0 do 8. Celotno populacijo imamo shranjeno v dvodimenzionalni tabeli, prav tako shranjujemo tudi podatke o uspešnosti posameznih osebkov. Te podatke na začetku vsake igre naložimo v delovni pomnilnik, med igro jih posodabljamo in jih ob koncu igre spet shranimo na disk.

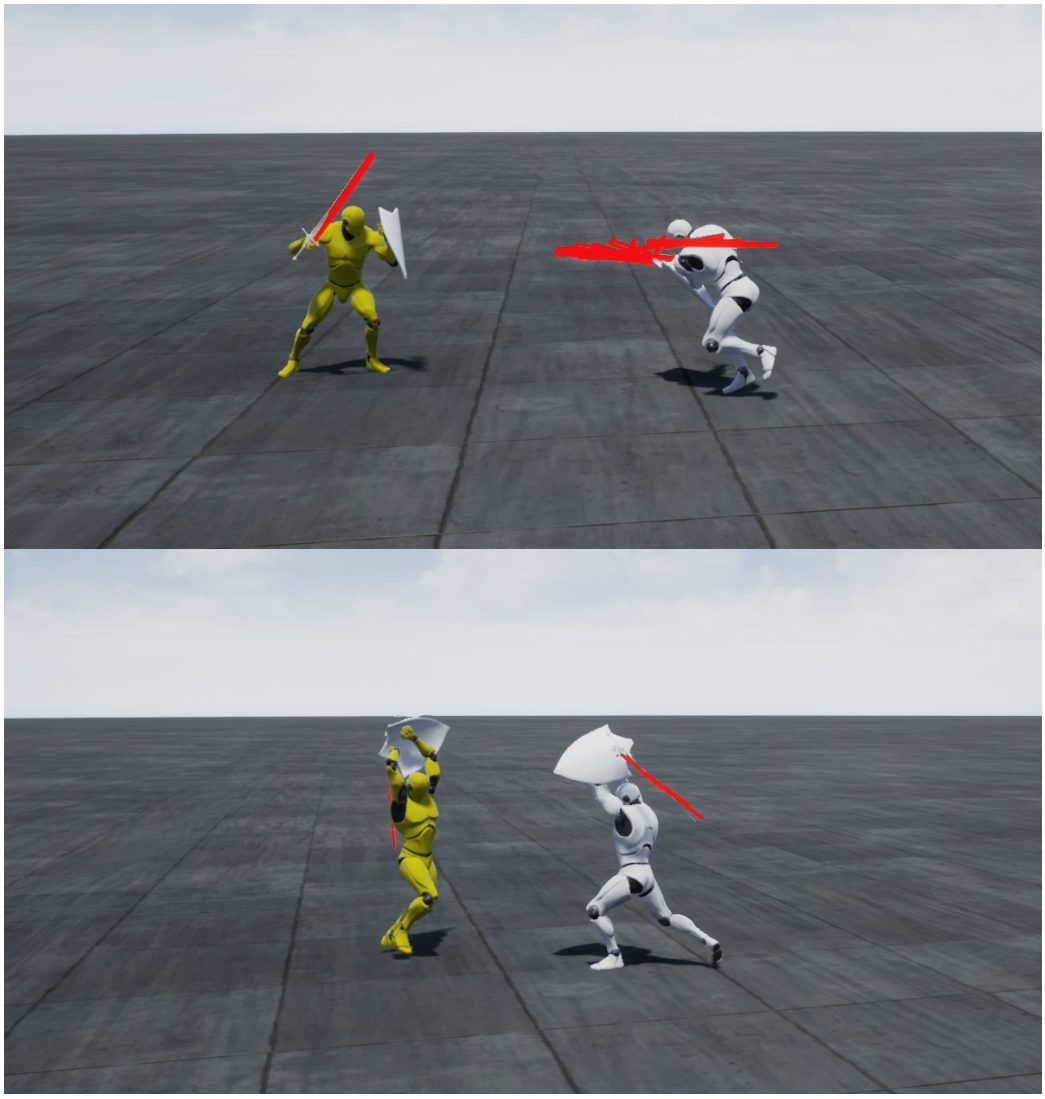
4.5.3 Rezultati

Učenje sistema na podlagi genetskih algoritmov smo ustavili po tristo generacijah. Najuspešnejši osebek zadnje generacije smo tako kot pri učenju z genetskimi algoritmi testirali s tisoč igrami proti ročno programiranemu nasprotniku. Izkazalo se je, da smo bili tudi pri učenju na podlagi genetskih algoritmov uspešni, saj je od tisoč iger inteligentni agent zmagal v 983 primerih, 17 iger je izgubil, izenačenj ni bilo.

Inteligentni agent, učen z genetskimi algoritmi, se skuša izmakniti vsem napadom nasprotnika. Za razliko od agenta, učenega s Q-učenjem pa agent, učen z genetskimi algoritmi, ne čaka vedno na optimalno priložnost za napad, ki se pojavi po izmiku, temveč skuša izkoristiti vsako priložnost. Če je prvi napad uspešen, agent nadaljuje z napadom. S to taktiko je agent zmagal v več kot 98 % primerov.

Obnašanje inteligentnega agenta lahko opazujemo na slikah 4.4 in 4.5, kjer rumeni igralec predstavlja inteligentnega agenta, učenega z genetskimi algoritmi, beli igralec ročno programiranega agenta, rdeče črte pa označujejo pot, po kateri je potoval meč v času napada, in služijo zgolj za lažjo vizualizacijo.

Slika 4.4: Izmik v levo in začetek izmika nazaj



Slika 4.5: Konec animacije izmika nazaj in začetek napada

Slika 4.4 prikazuje inteligentnega agenta ki se izmika napadom nasprotnika. Po izmiku nazaj se agent odloči napasti. Kot je razvidno s slike 4.5, agent kljub uspešnemu izmikanju ni prekinil svoje animacije izmika. Ker je za predvajanje nepotrebnega dela animacije izmika izgubil čas, je imel nasprotnik dovolj časa, da tudi sam začne napad. Inteligentni agent je začel svoj napad pred nasprotnikom, zato ima rahlo večjo verjetnost za zmago v tem obračunu. Kljub končni zmagi inteligentnega agenta je jasno, da je bil dvo-

boj zelo blizu. Sliki lepo ponazarjata taktiko inteligentnega agenta, učenega z genetskimi algoritmi, saj omenjeni agent poskuša izkoristiti tudi zelo majhne šibkosti nasprotnika in na ta način priti do zmage, ravno zaradi tega pa je pogosto kaznovan in sam prejme poškodbe.

4.6 Ugotovitve

Tako pristop s Q-učenjem kot tudi pristop z uporabo genetskega algoritma sta podala dobro rešitev zastavljenega problema. Oba sta dosegla bistveno večjo uspešnost, kot smo pričakovali. Oba sta prepustila nasprotniku prvo potezo, nato pa izkoristila čas, ko je nasprotnik že nehal napadati, vendar se je še vedno predvajala animacija napada. Ena izmed večjih razlik med pristopoma je, da pristop z genetskimi algoritmi napade že takoj po prvem napadu nasprotnika, pristop s Q-učenjem pa čaka na nasprotnikov drugi napad. Menimo, da je pristop inteligentnega agenta, učenega s Q-učenjem, boljši, saj je animacija pri drugem napadu rahlo daljša, posledično pa je tudi nasprotnik dlje časa ranljiv. Če je inteligentni agent na podlagi genetskega algoritma nasprotnika ranil, je izkoristil to ranljivost nasprotnika in nadaljeval z napadom. Agent, učen s Q-učenjem, pa se je pogosto kljub uspešnemu napadu odmaknil na varno razdaljo od nasprotnika.

Razliki se pojavita med napadom in v strategiji igranja, ko je nasprotnik že ranjen. Sistem, učen z metodo Q-učenja, igra varneje, čaka na optimalno priložnost, da napade, vendar zato potrebuje več časa za zmago. Sistem, učen z metodo genetskih algoritmov, pa je bolj agresiven in skuša izkoristiti vsako šibkost nasprotnika, zato tudi sam pogosteje prejme poškodbe, vendar pa igro zaključi veliko hitreje.

Poglavje 5

Zaključek

Menimo, da smo v diplomskem delu uspešno rešili zadano nalogo. Ustvarili smo dva ločena sistema za strojno učenje likov v računalniških igrah, ki predstavljata boljšo alternativo ročnemu programiranju. Problema smo se lotili tako s spodbujevanim učenjem kot z evolucijskimi algoritmi, z obema sistemoma smo prišli do zadovoljivih rešitev. Za nadaljnji razvoj dela predlagamo tudi implementacijo z nevronskimi mrežami [4]. Trenutno moramo agentu za vsako možno kombinacijo podatkov o igralnem okolju določiti svoje stanje, temu se lahko izognemo z uporabo nevronskih mrež, ki nam omogočajo, da agentu podamo zgolj ločene informacije o okolju in pustimo, da si z ustvarjanjem povezav med informacijami agent sam oblikuje trenutno predstavo sveta.

Verjamemo, da bi boljše rezultate dobili, če bi inteligentnemu agentu podali več informacij, torej če bi informacije iz okolja zakodirali v več kot zgolj 10 stanj. Menimo, da bi več informacij potrebovali o igralnem okolju, saj inteligentni agent trenutno ne dobi informacij o okolju, vendar zgolj o nasprotniku in njegovi relativni lokaciji glede na agenta. Podatki o okolju bi inteligentnemu agentu omogočili boljše manevriranje po igralnem prostoru in izkoriščanje lastnosti okolja za boj proti nasprotniku. Pristop, ki bi agentu omogočil integracijo lastnosti okolja v svojo taktiko, bi bil za agenta zelo

koristen, vendar pa bi tak pristop močno podaljšal čas, potreben za učenje.

Poleg učenja na podlagi večjega števila informacij iz okolja bi bil zanimiv pristop k problemu z uporabo zveznih spremenljivk. Trenutno vse zvezne spremenljivke v naših sistemih diskretiziramo ročno. Možno je razviti sistem, ki bi deloval z uporabo zveznih spremenljivk, kot temelj za razvoj takšnega sistema predlagamo metode, razložene v članku [12]. V omenjenem članku avtorji predstavijo algoritem, ki z uporabo odločitvenih dreves [9] poišče primerno diskretizacijo zveznega prostora za uporabo v strojnem učenju.

Oba algoritma, implementirana v sklopu našega diplomskega dela, se da enostavno razširiti na večuporabniške sisteme, zato verjamemo, da smo metode strojnega učenja, s katerimi smo rešili problem, ustrezno izbrali. Zaradi dobrih rezultatov in zanemarljive računske zahtevnosti implementiranih sistemov ima naša naloga zelo velik potencial pri strojnem učenju v sodobnih spletnih računalniških igrah, hkrati pa pušča veliko možnosti za nadaljnje razvijanje.

Literatura

[1] More on Dota 2. https://blog.openai.com/more-on-dota-2/. Dostopano: 2018-08-21.

[2] Soldiers of Fortune 2. https://en.wikipedia.org/wiki/Soldier_of_Fortune_II:_Double_Helix. Dostopano: 2018-08-21.

[3] Unreal Engine 4. https://www.unrealengine.com/en-US/what-is-unreal-engine-4. Dostopano: 2018-08-21.

[4] Ajith Abraham. Artificial neural networks. *handbook of measuring system design*, 2005.

[5] Yoav Freund and Robert E Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the ninth annual conference on Computational learning theory*, pages 325–332. ACM, 1996.

[6] Ben Geisler. Integrated machine learning for behavior modeling in video games. In *Challenges in game artificial intelligence: papers from the 2004 AAAI workshop. AAAI Press, Menlo Park*, pages 54–62, 2004.

[7] Edoardo Giacomello, Pier Luca Lanzi, and Daniele Loiacono. DOOM level generation using generative adversarial networks. *CoRR*, abs/1804.09154, 2018.

[8] Kathryn Elizabeth Merrick and Mary Lou Maher. Motivated reinforcement learning for adaptive characters in open-ended simulation games.

In *Proceedings of the international conference on Advances in computer entertainment technology*, pages 127–134. ACM, 2007.

[9] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[10] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. Evolving neural network agents in the NERO video game. *Proceedings of the IEEE*, pages 182–189, 2005.

[11] Alan M Turing. Intelligent machinery, a heretical theory. *Philosophia Mathematica*, 4(3):256–260, 1996.

[12] William TB Uther and Manuela M Veloso. Tree based discretization for continuous state space reinforcement learning. In *Aaai/iaai*, pages 769–774, 1998.

[13] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, and Sebastian Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. *CoRR*, abs/1805.00728, 2018.

[14] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.