Univerza v Ljubljani

Fakulteta za racunalniˇ ˇstvo in informatiko

Jernej Habjan

**Uˇcenje realno-ˇcasovne strateˇske igre z uporabo globokega spodbujevalnega**

**uˇcenja**

DIPLOMSKO DELO

UNIVERZITETNI STUDIJSKI PROGRAMˇ

PRVE STOPNJE

RACUNALNIˇ STVO IN INFORMATIKAˇ

Mentor: doc. dr. Matej Guid

Somentor: prof. dr. Branko Sterˇ

Ljubljana, 2018

Copyright. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za raˇcunalniˇstvo in informatiko Univerze v Ljubljani. Za objavo in koriˇsˇcenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za raˇcunalniˇstvo in informatiko ter mentorja.

*Besedilo je oblikovano z urejevalnikom besedil LATEX.*

Fakulteta za raˇcunalniˇstvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Besedilo teme diplomskega delaˇstudent prepiˇse izˇstudijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj priˇcakuje od kandidatovega diplomskega dela. Kaj so cilji, kakˇsne metode uporabiti, morda bo zapisal tudi kljuˇcno literaturo.

*Zahvaljujem se mentorju doc. dr. Mateju Guidu in somentorju prof. dr. Branku Steru, prijateljem in druˇzini, ki so mi pomagali pri pisanju diplomskeˇ naloge.*

**Kazalo**

**Povzetek**

**Abstract**

1. **Uvod 1**
2. **Realno-ˇcasovne strateˇske igre 5**

[2.1 Strategija 7](#_Toc42497)

[2.2 Taktika 7](#_Toc42498)

[2.3 Abstrakcija prostora 8](#_Toc42499)

[**3 Predstavitev algoritma Alpha Zero** **13**](#_Toc42500)

[3.1 Zgodovina 13](#_Toc42501)

[3.2 Potek uˇcenja 15](#_Toc42502)

[**4 Opis pravil igre** **19**](#_Toc42503)

[4.1 Stanje igre 20](#_Toc42504)

[4.2 Akcije 21](#_Toc42505)

[4.3 Kodiranja 25](#_Toc42506)

[4.4 Konec igre 26](#_Toc42507)

[**5 Uˇcenje modela** **35**](#_Toc42508)

[5.1 Zgradba nevronske mreˇze 35](#_Toc42509)

[5.2 Predstavitev parametrov 36](#_Toc42510)

[5.3 Postopno uˇcenje 37](#_Toc42511)

[**6 Vizualizacije** **41**](#_Toc42512)

[6.1 Pygame 41](#_Toc42513)

[6.2 Unreal Engine 4 43](#_Toc42514)

[**7 Rezultati** **47**](#_Toc42515)

[7.1 Uˇcenje z ustavitveno funkcijo s ˇcasovno omejitvijo 47](#_Toc42516)

[7.2 Poveˇcevanje ˇcasovne omejitve 51](#_Toc42517)

[7.3 Sprememba konfiguracij zlata 51](#_Toc42518)

[7.4 Zmanjˇsevanje velikosti ˇsahovnice 54](#_Toc42519)

[7.5 Vizualizacija rezultatov v pogonu Unreal Engine 56](#_Toc42520)

[**8 Diskusija** **57**](#_Toc42521)

[**9 Zakljuˇcek** **59**](#_Toc42522)

[**Literatura** **62**](#_Toc42523)

**Seznam uporabljenih kratic**

|  |  |  |
| --- | --- | --- |
| **kratica** | **angleˇsko** | **slovensko** |
| **MCTS** | Monte Carlo tree search | drevesno preiskovanje Monte  Carlo |
| **UE4** | game engine Unreal Engine 4 | celostni pogon Unreal Engine  4 |
| **RTS** | real-time strategy | realno-ˇcasovna strateˇska |
| **One hot** | one hot | kodiranje z eno enico v zapisu vsakega stanja |
| **JSON** | JavaScript Object Notation | notacija za oznaˇcevanje Java-  Script objektov |

**Povzetek**

**Naslov:** Uˇcenje realno-ˇcasovne strateˇske igre z uporabo globokega spodbujevalnega uˇcenja

**Avtor:** Jernej Habjan

Z obstojeˇcim Alpha Zero algoritmom smo implementirali uˇcenje in priporoˇcanje akcij v realno-ˇcasovni strateˇski igri. Pregledali smo krajˇso zgodovino globokega spodbujevalnega uˇcenja na igrah in povzeli zakaj je pristop samostojnega uˇcenja najprimernejˇsi. Za strateˇsko igro smo doloˇcili figure in njihove akcije in zakodirali kompleksno stanje igre s kodirnikom. Prav tako smo doloˇcili ustavitvene pogoje pri igri, ki nima konˇcnega ˇstevila potez na podlagi zmanjˇsanja ˇzivljenjskih toˇck figur. Rezultate smo prikazali s Python modulom Pygame in v celostnem pogonu Unreal Engine 4. V obeh vizualizacijah lahko igramo proti nauˇcenem modelu, ali pa opazujemo, kako se dva raˇcunalniˇska nasprotnika bojujeta med sabo. Na koncu smo ˇse pregledali rezultate in povzeli uˇcinek uˇcenja algoritma.

**Kljuˇcne besede:** Alpha Zero, realno-ˇcasovna strateˇska igra, Unreal Engine.

**Abstract**

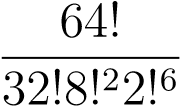
**Title:** Learning a real-time strategy game with deep reinforcement learning **Author:** Jernej Habjan

With the existing Alpha Zero algorithm we have implemented the learning and recommendation of actions in a real-time strategy game. We examined the shorter history of deep reinforcement learning in games and summarized why the self-learning approach is best suited. For a strategic game we defined the characters and their actions and coded the complex state of the game with the encoder. We have also determined the stop conditions of the game, which has no number of moves due to damage to the pieces. The results were displayed with the Python Pygame module and the integrated drive of the Unreal Engine 4. In both visualizations we can play against the learned model, or we can observe two computer opponents fighting against each other. In the end, we also reviewed the results and summarized the learning effect of the algorithm.

**Keywords:** Alpha Zero, real-time strategy game, Unreal Engine.

**Poglavje 1**

**Uvod**

Razvijanje inteligentnega agenta v realno-ˇcasovnih strateˇskih (ang. real-time strategy; RTS) igrah je problem, s katerim se mora sooˇciti veˇcina razvijalcev teh iger. Agentove akcije so ~~pa~~ pogosto predvidljive, saj se ˇcloveˇski igralec nauˇci njihovih naˇcinov delovanja in jih tako laˇzje premaga. Ce pustimoˇ agentu, da sam opravlja akcije nekontrolirano, bo le te akcije izvajal nakljuˇcno, ki so ~~pa veˇcino ˇcasa~~ slabˇse kot vnaprej opisana taktika. Lahko pa agentu podamo hevristiko, po kateri se mora ravnati in ta bo poskuˇsal izvesti ˇcim boljˇso akcijo, vendar bo za njen izraˇcun porabil predolgo ˇcasa, saj bo moral pregledati cel preiskovalni prostor, ki je pri realno-ˇcasovnih strateˇskih igrah ~~zna biti~~ prevelik. Na primer 10 figur v igri, kjer ima vsaka 5 moˇznih potez, se razveji na moˇzen faktor 510 ≈ 10 milijonov moˇznih akcij. Za igro StarCraft je ocenjenih moˇznih vsaj 101685 moˇznih akcij, kjer je za ˇsah ponavadi ocenjeno med 1040 in 1050, natanˇcneje  oziroma pribliˇzno 1043 [13] ter 10171 pri igro Go [6]. Za primerjavo lahko kot zanimivost vzamemo ˇstevilo atomov v opazovanem vesolju, ki obsega med 1078 in 1082 [1].

Preiskovanje prostora z grobo silo torej odpade. Ostanejo nam ~~potem~~ hevristiˇcni algoritmi ~~kot na~~ (primer Alpha-Beta rezanje ali drevesno preiskovanje Monte Carlo (ang. Monte Carlo tree search; MCTS)). Seveda pa moramo vpoštevati, da algoritem Alpha-Beta rezanje deluje dobro samo pod pogoji, da obstaja zanesljiva evaluacijska funkcija in da ima igra majhen vejitveni prostor, kar pa je lastnost

1

*2* Jernej Habjan

veliko klasiˇcnih namiznih iger kot Go in video iger. Zato se je v takˇsnih primerih bolje odloˇciti za algoritem MCTS [2]. Ta pa ima pomanjkljivost, da si stanj igre ne zapomni skozi iteracij veˇc iger, kjer bi lahko to vrednost stanja uporabil za bolj natanˇcen izraˇcun naslednjih stanj.

Za pomnjenje stanj so primerne globoke nevronske mreˇze, ki ~~pa~~ z uˇcenjem ugotovijo zakonitosti v uˇcni mnoˇzici in skozi mnogo iteracij izboljˇsajo svojo napoved doloˇcenega izhoda ob doloˇcenem vhodu. To je ~~pa~~ toˇcno to, kar potrebuje MCTS kot zaˇcetno stanje, iz katerega laˇzje izraˇcunamo najboljˇso akcijo.

Da pa nevronska mreˇza dobi dovolj vhodnih podatkov za uˇcenje, ~~pa~~ moramo realizirati algoritem, ki bo igral proti drugem raˇcunalniˇskem nasprotniku. S tem bo algoritem izgradil dovolj veliko uˇcno mnoˇzico z rezultati zmag oz. porazov teh iger. Ob tem izhodu nevronska mreˇza posodobi svojo napoved akcij za doloˇceno stanje igre.

To je glavna ideja o implementaciji algoritma, ki jo ~~pa~~ vsebuje algoritem Alpha Zero, ki smo ga uporabili v naši diplomski nalogi. Algoritem se nauˇci igranja raznih namiznih iger, ~~kot tudi naˇse~~  ~~in~~ realno-ˇcasovne strateˇske igre z igranjem mnogo iger sam proti sebi (boljˇsa razliˇcica algoritma napreduje v naslednjo iteracijo). Ko je model nevronske mreˇze nauˇcen, ga lahko uporabimo, da nam priporoˇci akcijo v doloˇcenem stanju. S tem lahko implementiramo raˇcunalniˇskega igralca, ki pridobiva akcije od nauˇcenega modela in jih izvrˇsuje, kot tudi priporoˇcilni sistem za akcije, ki jih prikazujemo ˇcloveˇskemu igralcu. Algoritem nam priporoˇci akcijo in ne tipa strategije (za to bi potrebovali še bolj abstrakten pogled na igro) ~~katerega naj izberemo, kar bi potrebovalo ˇse bolj abstrakten pogled na igro.~~

O strateˇskih igrah, njihovih abstrakcijah in zakaj so tako zanimive za raziskovanje umetne inteligence bomo veˇc spoznali v poglavju 2. Za tem si bomo podrobneje pregledali sestavo Alpha Zero algoritma in zakaj je primeren za naˇso realno-ˇcasovno strateˇsko igro v poglavju 3. Ko bomo imeli sestavljen algoritem uˇcenja, bomo zanj sestavili RTS igro v poglavju 4 in izpostavili, kaj so glavne teˇzave pri teh igrah v zvezi z njihovim uˇcenjem. Sestavljen algoritem bomo nauˇcili na igri v poglavju 5, kjer bomo pregledali razne parametre pri uˇcenju in nauˇcen model potem preizkusili z vizualizacijo Diplomska naloga *3*

v Python modulu Pygame in celostnem pogonu Unreal Engine 4 v poglavju 6. Rezultate uˇcenja bomo potem ˇse ocenili in ugotovili, katera vrsta uˇcnih parametrov nam je podala najboljˇsi rezultat v poglavju 7. V poglavju 8 pa bomo rezultate pregledali s ˇsirˇse perspektive, omenili bomo tudi moˇzne nadaljnje pristope in izboljˇsave, ki bi pomagale pri uˇcenju algoritma. Pregledali bomo tudi koristnost aplikacije nauˇcenega modela v pogon kot je Unreal Engine in kaj so njegove omejitve. Na koncu bomo ˇse naredili kratek pregled doseˇzkov in prispevkov diplomske naloge v poglavju 9.

tole bi js čist ist drugač nardila. Js bi napisala tkle.

V naši diplomski nalogi, vam bomo skozi strukturiran pristop v posameznih točkah opisali še:

- zanimivosti raziskovanja umetne iteligence in stateške igre z abstrakcijami (poglavje 2),

- sestava alfa bet in zakaj je pomemben za našo diplomsko nalogo (poglavje 3)…..

*4* Jernej Habjan

**Poglavje 2**

**Realno-ˇcasovne strateˇske igre**

Realno-ˇcasovne strateˇske igre so ˇzanr strateˇskih iger, kjer igralec nadzoruje mnoˇzico figur, in poskuˇsa premagati nasprotnika z izgradnjo ekonomije, izboljˇsavo raznih tehnolog in urjenjem primernih vojaˇskih figur, ki dodajo dodano vrednost konˇcnem cilju poraza nasprotnega igralca in s tem zmagi igre. Igre ~~pa~~ se odvijajo v realnem ˇcasu, kar pomeni da se stanje igre lahko spremeni veˇckrat na sekundo, ~~kar~~ s tem pa prisili igralca k stalnem fokusu na igro. Primer RTS igre sta igra Age of Empires II(Ensemble Studios) in StarCraft(Blizzard Entertainment, THQ) 2.1.

Razlike realno-ˇcasovnih iger v primerjavi s tradicionalnimi namiznimi igrami je dobro opisal Santiago Ontan˜on v raziskovalnem delu Pregled uˇcenja umetne inteligence na realno-ˇcasovnih strateˇskih igrah in tekmovanja v igri StarCraft [7]. Te razlike pa so naslednje:

* izvajanje akcij v istem ˇcasovnem intervalu, ki ~~pa~~ lahko trajajo veˇc ˇcasovnih intervalov,
* odloˇcitev akcij v krajˇsem ˇcasovnem obdobju, saj za razliko od ˇsaha, kjer ima igralec na voljo veˇc minut za izbiro poteze, se v igri kot npr. StarCraft stanje igre zamenja 24-krat na sekundo,
* igre so lahko le vidne na podroˇcjih kjer je igralec ˇze raziskal in imajo vpogled igralˇceve figure,

5

* veˇcina iger ni deterministiˇcnih, kjer imajo akcije moˇznost uspeha,
* kompleksnost raziskovalnega prostora je veliko veˇcji.

Zaradi teh razlik so nastopili razni izzivi:

* planiranje: Realno-ˇcasovne strateˇske igre imajo obiˇcajno veˇcji raziskovalni prostor, kar pa prepreˇci globje raziskovanje stanja iger. Kot bomo pozneje pregledali, se igre zato abstrahirajo na veˇc nivojev. Viˇsji kot je nivo, bolj dolgoroˇcni so cilji, kot na primer gradnja ekonomije, na niˇzjem nivoju je pa premik posamezne figure ipd.,
* uˇcenje: Uˇcenje igranja igre lahko poteka na naˇcin predhodnega uˇcenja, ki uporablja posnetke ˇze odigranih iger in na naˇcin uˇcenja v igri, ki uporablja po veˇcini spodbujevalno uˇcenje in modeliranje nasprotnika,
* negotovost: Negotovost nastane zaradi nevidnosti nasprotnikovih figur in njegovih potez v vsakem trenutku. Prav tako pa v danem trenutku ne moremo doloˇciti, katero akcijo bo nasprotnik izvedel, zato zgradimo drevo, ki nam pove najveˇcjo verjetnost izbrane akcije,
* prostorsko in ˇcasovno razumevanje: Prostorsko razumevanje je usmerjeno k postavljanju stavb in pozicijo vojske za obrambo in napad, med tem ko je ˇcasovno razumevanje usmerjeno k ugotavljanju ˇcasovne primernosti izdelave doloˇcenih figur, ki so primerne za izboljˇsavo igralˇceve ekonomije, tehnoloˇskega drevesa ali ˇcas napada ipd.,
* izkoriˇsˇcanje znanja domen: V tradicionalnih igrah kot npr. ˇsah se lahko zanaˇsamo na dobre ovrednotenske funkcije kot na primer algoritem alpha-beta rezanje in tabele za opis stanja konca igre, v realnoˇcasovnih strateˇskih igrah pa ˇse ni jasno kako lahko raˇcunalniˇski nasprotniki uporabijo domenska znanja iz posnetkov iger, zato se razvijalci tovrstnih iger bolj osredotoˇcajo na izgradnjo veˇc taktik, med katerimi se raˇcunalniˇski nasprotnik odloˇca.
* razdelitev nalog 2.2: Veˇcje in zahtevnejˇse naloge so razdeljene na manjˇse, ki pa jih uvrˇsˇcamo v veˇc nivojev glede na abstrakcijo.
  + strategija je najviˇsji nivo abstrakcije, ki zajema okrog 3 minutna planiranja in vse figure ki jih igralec nadzoruje,
  + taktika, ki je implementacija trenutne strategije (pozicija vojske, hiˇs - 30 sekundno planiranje),
  + reakcijska kontrola, (ki je implementacija taktike, ki je osredotoˇcena na posamezno figuro,)
  + analiza terena, (ki se osredotoˇca na strnjena obmoˇcja in na viˇsinsko prednost,)
  + pridobivanje znanja, s katerim pridobivamo informacije o taktiki nasprotnika.

## 2.1 Strategija

V strateˇskih igrah je velikokrat uporabljen pristop direktnega kodiranja strategije, ki uporabljajo avtomate konˇcnih stanj, kjer lahko razbijemo delovanje na veˇc stanj kot so napadanje, nabiranje surovin, popravilo itd. in hitro menjavanje med njimi. Direktno kodiranje prinese dobre priˇcakovane rezultate, vendar se lahko igralec nauˇci strategije in ga tako agenta hitro porazi.

Planirani pristopi ponujajo veˇcjo prilagodljivost kot direktno kodirani.

## 2.2 Taktika

Taktika spada pod neposrednejˇsi nadzor figur kakor strategija in je bolj osredotoˇcena na kontrolo doloˇcenih toˇck na mapi, zmagi posameznih bitk in iskanje oˇzin, kjer je nasprotnik ˇsibkejˇsi. Taktika temelji na analizi terena, ki ga lahko razbijemo na kompozicijo oˇzin.

## 2.3 Abstrakcija prostora

Razbiranje strategije in taktike je za algoritme umetne inteligence teˇzji (težji od česa), saj potrebuje viˇsji nivo abstrakcije prostora, figur in akcij, kot za izbiro posameznih nizkonivojskih akcij.

Za primer abstrakcije prostora vzemimo raˇcunalniˇsko realno-ˇcasovno igro, kjer bi lahko premik figure spremljali vsak prikazan piksel na prikazovalniku, vendar je primernejše da gledamo na premik figur kot izid na viˇsjih ravneh [11]. Posamezne figure vojaˇskih enot lahko zdruˇzujemo v veˇcje skupine, katere potem obravnavamo in upravljamo kot en osebek. Te skupine vojaˇskih enot je pa potrebno pravilno postavljene na razne strateˇske toˇcke, ki nam zagotavljajo znatno prednost pred nasprotnikovikom. V ta namen lahko razgradimo zemljevid igre glede na oˇzine prestope iz viˇsjega na niˇzji del terena. Na ta naˇcin lahko postavimo enote glede na ta razbiti teren [10]. V naˇsem delu smo zemljevid razbili kar na kvadratno mreˇzo 8x8 ali 6x6, saj ne vsebuje nobenih oˇzin in nedostopnih mest. Imamo pa majhno ˇstevilo posameznih enot, ki pa lahko v raˇcunalniˇsko igro preslikamo kot skupek enot, kot smo to opisali zgoraj.

Abstrakcija prostora poteka teˇzje ˇce igra vsebuje dejavnike negotovosti, kot na primer prekrivanje zemljevida z meglo (ang. Fog of war) kjer ne vidimo nasprotnikovih figur in potez v vsakem trenutku. Napoved nasprotnikovih potez je tako veliko teˇzja, tako da vsi algoritmi s tako negotovostjo ne delujejo. Mi smo se za diplomsko nalogo odloˇcili, da imata oba raˇcunalniˇska agenta popoln vpogled na stanje igre in nasprotnikove akcije.

Prav tako lahko v igri kot npr StarCraft posamezno vojaˇsko enoto premaknemo na poljubno koordinato na zemljevidu, pod pogojem da je ta koordinata dosegljiva z vidika algoritma za iskanje poti. Te koordinate premika lahko segajo tudi v neznane dele zemljevida, ki pa so lahko ˇze zasedene z nasprotnikovo arhitekturo, zato lahko premikanje poenostavimo na sosednja polja. Za napad nasprotnikovikovih enot, nabiranje zlatnikov ipd. lahko omejimo na izvajanje akcij znotraj doloˇcenega polja na najbliˇzjo figuro. Na primer vojaˇska enota lahko napade najbliˇzjo nasprotnikovo enoto znotraj polja, v katerem je ta naˇsa vojaˇska enota. V naˇsem primeru, kot bomo pojasnili pri poglavju opis pravil igre 4, ne moremo postaviti veˇc figur na isto polje, zato so akcije kot napad, nabiranje zlatnikov omejene na sosednja polja.



Slika 2.1: Na zgornjih dveh slikah sta predstavljeni igra StarCraft (zgoraj) in Age of Empires II (spodaj). Obe igri sta prikazani ˇse v zaˇcetnem stanju, kjer sta na obeh slikah vidni glavni hiˇsi, delavci, viri surovin(minerali, rude zlata, drevesa ipd.). Pri igri Age of Empires II je viden zemljevid, kjer zasenˇcenost predstavlja neraziskan del. Izvajanje doloˇcenih akcij poteka veˇc ˇcasa, saj na primer delavci pri igri StarCraft vraˇcajo minerale veˇc ˇcasovnih enot, pri igri Age of Empiress II pa se gradi delavec veˇc ˇcasovnih enot.

povprečni

igralčeve namere načrtovani čas

~3 min

~30 sec

|  |
| --- |
| Strategija |

|  |
| --- |
| Taktike |

Reakcijska kontrola ~1 sec



delna opazovanja

direktno znanje

Časovno razmišljanje

Prostorsko sklepanje

Slika 2.2: Slika prikazuje razdelitev nalog glede na ˇcas reakcije in abstrakcije nalog. Prikazani so nivoji odloˇcanja glede na ˇcasovni razpon, kjer se negotovost akcij poveˇcuje s poveˇcevanjem ciljnega ˇcasa, za kar moramo igro pravilno abstrahirati na razliˇcne taktike in strategije.

*12* Jernej Habjan

**Poglavje 3**

# Predstavitev algoritma Alpha Zero

## 3.1 Zgodovina

Igranje iger je popularno podroˇcje znotraj vede o umetni inteligenci. Desetletja je bil za raziskovalce s podroˇcja umetne inteligence razviti program za igranje ˇsaha, dandanes pa so najboljˇsi algoritmi za igranje ˇsaha nepremagljivi za celo svetovnega prvaka. Ti algoritmi temeljijo na preiskovanju prostora veˇc milijonov ˇsahovskih pozicij in metodah, ki temeljijo na pravilih. Za razliko od teh programov je bil eden izmed prvih programov programski pogon Checkers (Samuel 2000), ki se je nauˇcil igranja z metodami samo-igranja in strojnega uˇcenja in ne npr. na metodah ki temeljijo na pravilih.

Pri teh igrah je faktor vejanja akcij ˇse relativno majhen in je laˇzje oceniti konˇcno pozicijo iz danega stanja [12]. Reˇceno je bilo, da igre kot npr. Go, ki imajo toliko veˇcji faktor vejanja 10171 v primerjavi sˇsahom, ki pa ima 1047, ne bo moˇzno ugotoviti vrednost konˇcnega stanja ˇse nekaj desetletij. Algoritem AlphaGo [8] pa je naredil preboj s tem da uporablja metodo globokega spodbujevalnega uˇcenja in algoritem drevesno preiskovanje Monte Carlo. Oktobra 2016 je premagal profesionalnega Go igralca na podlagi uˇcenja na domenskem znanju iger, ki so jih odigrali eksperti. Te sistemi so temeljili na predznanju.

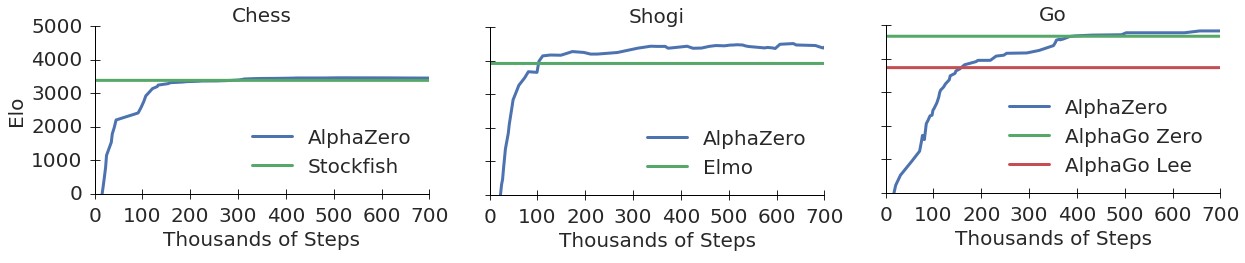
13

*14* Jernej Habjan

ekspertov za uˇcenje in evaluacijo modela, kar pomeni, da ob igranju novih iger posnemajo katere akcije so eksperti izvajali ob doloˇceni situaciji.

Njegovo delovanje pa lahko opiˇsemo v naslednjih korakih [3]:

* miselno odigraj igro z raziskovanjem neodkritega, pri ˇcemer upoˇstevaj nasprotnikove akcije,
* pri sooˇcenju z neznano pozicijo, oceni njeno vrednost in popravi ocene pozicij ki so vodile do trenutne pozicije,
* po prenehanju razmiˇsljanja odigraj potezo ki je najbolj obetavna,
* po koncu igre preglej vse pozicije kjer si se za pozicije narobe odloˇcil in jih popravi.



Slika 3.1: Predstavitev igranja iger z nauˇcenim algoritmom Alpha Zero z 700,000 iteracijami in ˇcas za napoved akcije je 1 sekunda. Na levi sliki je prikaz igranja ˇsaha proti programu Stockfish iz leta 2016, na srednji sliki igranje proti programu Elmo iz leta 2017 v igri Shogi, na desni pa igranje igre Go proti proramoma AlphaGo Lee in AlphaGo Zero.

Leto za tem, je bil razvit algoritem AlphaGo Zero [9], ki opisuje pristop k uˇcenju brez domenskega znanja ekspertov, ampak uporablja metodo samo-igranja 3.1. Novi model je prav tako premagal AlphaGo algoritem, kar predstavlja odliˇcne rezultate z vidika, da AlphaGo Zero ne potrebuje ˇcloveˇsko usmerjanje pri uˇcenju. Raˇcunalniki se lahko tako nauˇcijo reˇsevanje problema brez ˇcloveˇskih ekspertov, ki pogostejˇse delajo napake zaradi utrujenosti ali Diplomska naloga *15*

povrˇsnosti in nimajo takojˇsnega vpogleda na celotno zbirko iger, kot na to imajo raˇcunalniki.

Za tem je bil razvit algoritem Alpha Zero, ki vzame ideje AlphaGo Zero kot temelj, ampak je model generaliziran za poljubne igre, kot na primer ˇsah, Shogi, Go, kjer algoritem potrebuje samo pravila igre, ta pa se uˇci na podlagi globokih nevronskih mreˇzam in tabula rasa algoritmom za spodbujevalno uˇcenje. Zaradi te generalizacije algoritma, lahko algoritem apliciramo na naˇso RTS igro, katero moramo sprvaˇse definirati da je kompatibilna z algoritmom. Alpha Zero je drugaˇcen od AlphaGo Zero v tem, da so igre pri algoritmu AlphaGo Zero nastale z igranjem vseh posameznih modelov prejˇsnjih iteracij, in se je potem moˇc modela izraˇcunala proti najboljˇsim igralcem, medtem ko Alpha Zero samo hrani eno nevronsko mreˇzo, ki se stalno posodablja, namesto da ˇcaka iteracijo da se konˇca.

## 3.2 Potek uˇcenja

Alpha Zero se uˇci verjetnosti in ocenitve konˇcnega stanja izkljuˇcno z igranjem proti samemu sebi. Te potem uporabi pri preiskovanju z glavno namensko metodo drevesnim preiskovanjem Monte Carlo, da raziˇsˇce drevo stanj za akcijo. Drevo preiˇsˇce prostor in vrne verjetnost zmage pri izbiri doloˇcene akcije iz trenutnega stanja imenovano Pi in oceno konˇcnega stanja iz trenutnega stanja v, ki zavzema vrednosti -1 ali 1 (Poraz, zmaga). Alpha Zero izvede veˇc serij igranja iger proti svojim nasprotnikom, ki predstavlja zdajˇsnji najboljˇsi model igranja. Rezultat igranja igre je lahko -1 za poraz, +1 za zmago in 0 za neodloˇceno. Po vsaki seriji uˇcenja, se izvede proces igranja iger dveh nauˇcenih modelov, kjer oba igrata drug proti drugemu nekaj iger, in se na to doloˇci zmagovalen model, ki sedaj postane najboljˇsi model, ˇce je razlika v ˇstevilu zmag veˇcja za nek faktor. V naˇsem primeru je bil ta faktor 60%.

Ta pogoj izbiranja modelov smo morali ˇse izboljˇsati, saj ob upoˇstevanju samo ˇstevila zmag in porazov se lahko algoritem prekomerno prilagodi na izenaˇcevanje iger. Ravno to se nam je pripetilo pri uˇcenju na konfiguraciji *16* Jernej Habjan

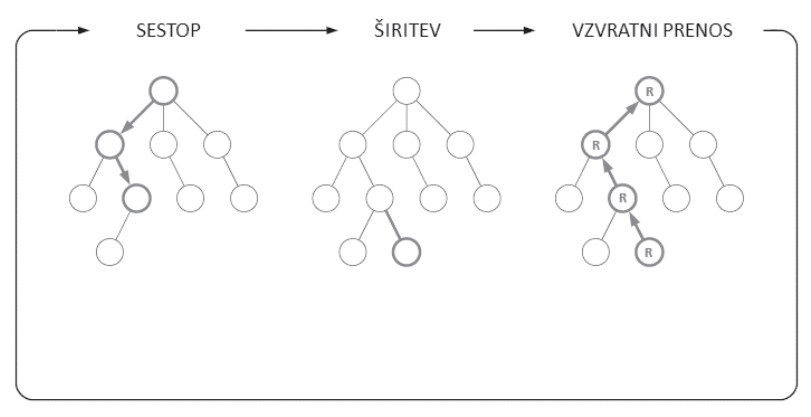
igre, kjer smo razporedili polja zlata na robove ˇsahovnice 7.3. Da smo ta problem odpravili, smo pri upoˇstevanju izbire novega modela dodali tudi neodloˇcene izide kakor slabe rezultate.

if float(nwins) / (pwins + nwins + draws) < updateThreshold:

# reject new model else:

# accept new model

Parametri nevronske mreˇze so za tem popravljeni, da minimizirajo napako med napoved stanja nevronske mreˇze in dejanskim rezultatom igre in da maksimizirajo podobnost napovedjo potez nevronske mreˇze z dejanskimi vrednostnimi akcij, ki jih je vrnil MCTS. Oziroma parametri se nastavijo z gradientnim spustom na funkcijo izgube, ki seˇsteje napako srednjega korena (mean-squared error) in preˇcne entropije (cross entropy). Nevronska mreˇza sprejme uˇcne mnoˇzice stanja iger in vrne ravni vektor napovedi akcij v trenutnem stanju in napoved zmage.



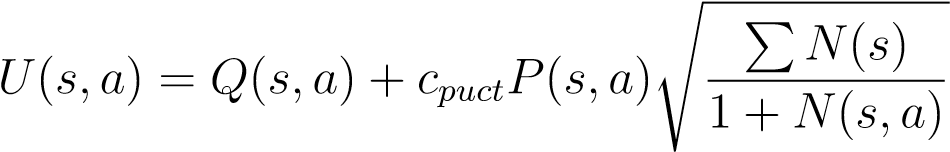
Slika 3.2: Na sliki je prikazano delovanje algoritma MCTS pri izvedbi algoritma Alpha Zero, ki ga uporabljamo v diplomski nalogi. Ta MCTS ne uporablja postopka odigravanja stanj do konca igre, ampak izvede doloˇceno ˇstevilo iteracij iskanja z raziskovalno funkcijo 3.1.

Uporaba algoritma MCTS je v tem algoritmu drugaˇcna kakor v sploˇsni Diplomska naloga *17*

uporabi. Stevilo iteracij je namenjeno biti veliko manjˇsi, kakor v njegoviˇ klasiˇcni uporabi, kjer je ˇstevilo iteracij veˇc sto tisoˇc. MCTS pripomore k izboljˇsavi napovedi stanja, ki ga vrne nevronska mreˇza z raziskovanjem prostora. Algoritem ne uporablja simulacij za pridobitev konˇcnega stanja igre, napoved stanja, ki ga vrne nevronska mreˇza samo izboljˇsa.

Vozliˇsˇce, ki ˇse ni bilo obiskano, se vzpostavi s napovedjo nevronske mreˇze in za tem vzvratno propagira napoved stanja. Ce je vozliˇsˇce konˇcno sta-ˇ nje igre, vzvratno propagira konˇcno stanje. MCTS v tem primeru prejme par sto iteracij (v naˇsem primeru 30 - 50) in ne veˇc tisoˇc, kot jih izvajajo drugi algoritmi. V sklopu diplomske naloge govorimo o MCTS iskanjih in ne odigravanjem, saj jih ta ne vpeljuje 3.2.

**Izrek 3.1** *formula po kateri raˇcuna verjetnost zmage pri doloˇceni akciji v algoritmu MCTS. Priˇcakovana vrednost akcije je doloˇcena z Q(s, a), ki predstavlja priˇcakovano nagrado ob izbiri akcije v danem stanju igre, kateri je priˇsteta napoved nagrade P(s, a) ob izbiri akcije v danem stanju, ki ga vrne nevronska mreˇza, pomnoˇzeno s raziskovalnim faktorjem in korenom ˇstevila vseh obiskov stanja igre v primerjavi s ˇstevili obiskov v danem vozliˇsˇcu.*

 (3.1)

Glavno uˇcenje algoritma poteka z igranjem iger, ki pa se za razliko od MCTS-ja odigrajo do konca in se dodajo v seznam uˇcnih primerov. Konec vsake iteracije igranja epizod iger se nevronska mreˇza uˇci na podlagi teh uˇcnih primerov. Za tem preveri moˇc novo nauˇcenega modela z igranjem proti starejˇsi razliˇcici modela in se shrani novi model samo ˇce je boljˇsi od starejˇsega za doloˇcen odstotek.

*18* Jernej Habjan

**Poglavje 4**

# Opis pravil igre

Igro smo opisali po Surag Nairjevi predlogi za Alpha Zero, ki je na voljo na repozitoriju Github (https://github.com/suragnair/alpha-zero-general). Igra je dodana kot modul, ki vsebuje definicijo igre in njena pravila, igralce, vizualizacijo in izgradnjo modela.

Igra je doloˇcena s kvadratno mreˇzo 8 x 8, kjer polje lahko vsebuje najveˇc eno figuro. Ostale igre, ki so napisane za to razliˇcico Alpha Zero izvedbe, kot na primer ˇstiri v vrsto, gobang, othello, tri v vrsto, vsebujejo ˇcrno-bele figure. Zakodirane so lahko z eno ˇstevilko: -1 za igralca -1, +1 za igralca 1 ali 0, ˇce je polje prazno. Pri teh igrah je dimenzija kodiranja 2-dimenzionalna, kjer dimenzije predstavljajo viˇsino in ˇsirino igralne ploˇsˇce. Pri RTS igrah pa moramo vedeti poleg igralca, komur ta figura pripada, tudi stanje te figure, na primer trenutno zdravje in njen tip. Zato je prostor kodiranja 3-dimenzionalen, kjer je tretja dimenzija zakodirano stanje figure. Ce biˇ dovolili, da na posamezno polje spada veˇc figur, se dimenzija ponovno poveˇca za 1.

Ob prvem poizkusu opisa igre smo se zapledli v ne-numeriˇcen prikaz igre, ki pa je zelo spremenil Surag-Nairjevo implementacijo algoritma Alpha Zero. Implementacija igre se je uˇcila zelo poˇcasi, saj je bilo preverjanje akcij poˇcasnejˇse od sedajˇsne implementacije. Algoritem je na voljo na naslednjem repozitoriju:

19

https://github.com/JernejHabjan/TD2020-Object-AlphaZero.

## 4.1 Stanje igre

V tem razdelku bomo opisali zapis posamezne figure, njihove akcije in kaj naredijo in tip kodiranja stanja igre, ki ga potem sprejme nevronska mreˇza.

Igro smo opisali tako, da je ˇcim bolj skladna s samim algoritmom Alpha Zero, kot tudi da je njena aplikacija dovolj primerljiva z obstojeˇcimi strateˇskimi igrami kot npr. StarCraft. S tem v mislih, smo opisali nekaj preprostih pravil, ki jih ta igra upoˇsteva:

* figure se ne poˇzirajo: Vojaˇske figure ne napadejo drugih figur tako, da ˇce je akcija napad moˇzna, se postavijo na polje nasprotnikove figure in s tem prepiˇsejo nasprotnikovikovo figuro in s tem jo uniˇci 4.1. Prav tako imajo vse figure doloˇceno zdravje, ki ga vojaˇske figure v veˇcih korakih zmanjˇsajo z napadom,
* ena figura na polje: S tem ni moˇzno blokiranje figur, s tem da se figura postavi na polje zlata in blokira nasprotnikovo figuro da jih nabere,
* igralec zgubi, ˇce zgubi vse figure - veˇc o ustavitvenih pogojih spodaj v razdelku 4.4,
* nabiranje zlatnikov je enkratna operacija, ki poteka podobno kot pri igri StarCraft, kjer mora figura pristopiti do polja zlata, zlatnike pobrati in jih za tem vrniti v glavno hiˇso. Nabiranje zlatnikov v tem primeru ne poteka tako, da se figura pomakne do polja zlata in s tem priˇcne avtomatiˇcno pridobivati zlatnike, brez da bi jih vraˇcal na odlagaliˇsˇce.

Sprva moramo opisati figure, ki bodo imele doloˇceno vlogo v igri. Nabor figur je majhen, saj noˇcemo, da preiskovalni prostor postane prehitro prevelik.

Na posameznem polju je lahko najveˇc ena figura, tako da igralec ne more blokirati surovin zlata nasprotnemu igralcu, ˇce to surovino ne obkoli v celoti.

|  |  |
| --- | --- |
| Ime figure | Opis |
| polje zlata | vir surovin, ki predstavljajo denar v igri, s katerim lahko igralec gradi nove stavbe in uri nove figure. Vir zlata je neomejen in ne mora biti uniˇcen |
| delavec | figura namenjena gradnji hiˇs in nabiranju zlata |
| vojasˇnica | stavba namenjena urjenju vojaˇskih figur |
| vojak | figura namenjena napadanju nasprotnikovih figur |
| glavna hiˇsa | stavba namenjena urjenju delavcev in vraˇcanju surovin zlata |

Tabela 4.1: Doloˇcitev figur in njihovih namenov v igri. Definirali smo samo 5 figur, med katerimi je polje zlata nevtralna, saj je igralec ne more nadzorovati.

Realizirali smo atribute figur. Pomembno je, da so te atributi numeriˇcni, da lahko podamo stanje igre kot N-dimenzionalen vektor, ki ga nevronska mreˇza lahko sprejme in se iz teh numeriˇcnih podatkov uˇci. Prav tako je pomembno, da ima vsako polje na ˇsahovnici enako ˇstevilo atributov, tudi ˇce je to polje prazno. Vsako prazno polje ima vanj vpisan atribut ˇcas igranja, ki je sploˇsen za celo igro, vsa ostala polja pa imajo vrednost 0.

Poseben primer je figura polje zlata, ki ne pripada nobenemu igralcu v veˇcini RTS igrah. V tem primeru pa smo podali vsakemu igralcu svoje polje zlata, da je igra simetriˇcna in nevronska mreˇza ne interpretira prazno polje igralca kot prazno polje. Prav tako se figuri polje zlata ne spreminja atribut zdravja, saj jo ne moremo poˇskodovati. Zlata je neomejeno in ko igralec odloˇzi zlatnike v glavno hiˇso, se vsem figuram tega igralca nastavijo zlatniki na novo dobljeno vrednost. Prav tako se pri izgradnji nove stavbe ali urjenju figure ˇstevilo zlatnikov zmanjˇsa za vse figure tega igralca.

## 4.2 Akcije

Prav tako moramo opisati akcije 4.2, ki jih te figure lahko izvajajo. Vsaka figura ne more izvajati vseh akcij, kot na primer stavbe se ne morejo premikati,

|  |  |
| --- | --- |
| Ime kodirnega  polja | Opis |
| ime igralca | doloˇca igralca, h kateremu ta figura pripada. Igralec lahko nadzoruje samo svoje figure, izvaja akcije na svojih figurah in napada nasprotnikovikove figure |
| tip figure | atribut predstavlja numeriˇcno predstavitev tipa figure kot na primer polje zlata, delavec ipd. Stanje igre potrebuje zapise tipov figur na poljih, da program ve, katere akcije tem figuram pripadajo |
| trenutno zdravje | koliko zdravja ima trenutna figura. Zdravje se lahko poveˇcuje do nekega maksimuma z akcijo zdravi in zniˇzuje z napadom figure |
| nosi zlatnike | poseben atribut za delavce, ki predstavlja vrednost 1, ˇce figura nosi zlatnike in 0, ˇce ga ne nosi. To se upoˇsteva pri nabiranju in vraˇcanju zlata, kjer se ti dve akcije ne zgodita v roku ene poteze, ampak se mora stanje prenaˇsati skozi veˇc potez |
| denar | trenutna koliˇcina zbranega denarja za posameznega igralca. To polje se ob spremembi koliˇcine denarja spremeni v vseh figurah tega igralca |
| cˇas igranja | to polje predstavlja koliko potez se je v trenutni igri ˇze izvedlo. Atribut je prisoten v vseh poljih in se spremeni v vseh poljih  ˇsahovnice, ko se izvede nova akcija |

Tabela 4.2: Definicija on opis kodirnikov stanja igre, s katerimi je predstavljeno vsako polje na ˇsahovnici. Veˇc o kodiranju teh polj si bomo pogledali v sekciji kodiranja 4.3.

same figure kot delavec in vojak pa ne morejo uriti novih figur 4.4.

Sprva smo doloˇcili nekatere izmed zgornjih akcij, na naˇcin izbire prvega mesta med sosednjimi polji, ki ustreza akciji. Naslednje akcije so se izvajale po zaporedju 4.3:

* naberi zlatnike,
* vrni zlatnike,
* napadi,
* delavec,
* vojak,
* vojaˇsnica, • glavna hiˇsa,
* zdravi.

koordinate = [(x - 1, y + 1), (x, y + 1),

(x + 1, y + 1),

(x - 1, y),

(x + 1, y),

(x - 1, y - 1),

(x, y - 1),

(x + 1, y - 1)]

for sosednji\_x, sosednji\_y in koordinate:

# preveri pogoj akcije if (pogoj == ok): return sosednji\_x, sosednji\_y

Ko je doseˇzeno prvo prazno polje v tem zaporedju, se tam izgradi nova hiˇsa ali izuri nova figura. Ko je prva nasprotnikova figura izbrana v tem

|  |  |
| --- | --- |
| Ime akcije | Opis |
| premiki (4 smeri) | figuri vojak in delavec se lahko premakneta na sosednje polje na ˇsahovnici, ˇce je to mesto prazno |
| naberi zlatnike | delavec lahko nabere zlatnike ˇce je v neposredni bliˇzini figure polje zlata. Zlatnike za tem drˇzi pri sebi, pri katerem se nastavi zastavica nosi zlatnike |
| vrni zlatnike | delavec vrne zlatnike, ki jih drˇzi pri sebi v glavno hiˇso, na kar se igralcu priˇsteje denar |
| napadi (4 smeri) | vojak lahko napade nasprotnikovo figuro, ˇce je ta v neposredni bliˇzini in jo rani za doloˇcen faktor. Ceˇ figuri ne preostane veˇc ˇzivljenjskih toˇck je eliminirana s ˇsahovnice, kot je tudi igralec, ˇce je bila eliminirana njegova zadnja figura |
| izuri delavca (4 smeri) | glavna hiˇsa lahko izuri novo figuro delavec, ˇce ima dovolj denarja, na kar se igralcu odˇsteje denar |
| izuri vojaka (4 smeri) | vojaˇsnica lahko izuri novo figuro vojak, ˇce ima dovolj denarja, na kar se igralcu odˇsteje denar |
| izgradi vojaˇsnico  (4 smeri) | delavec lahko izgradi vojaˇsnico na prazno mesto zraven njega, na kar se igralcu odˇsteje denar |
| izgradi glavno hisˇo (4 smeri) | delavec lahko izgradi glavno hiˇso na prazno mesto zraven njega, na kar se igralcu odˇsteje denar |
| zdravi (4 smeri) | figura lahko zdravi sosednjo prijateljsko figuro, ˇce ta nima polnega ˇzivljenja, na kar se igralcu odˇsteje denar |

Tabela 4.3: Opis akcij, njihovih dejanj in pogojev, ki morajo biti izpolnjeni, da se akcija lahko izvrˇsi. Nekatere akcije imajo 4 smeri izvajanja, kar pomeni, da bo akcija npr. izuri delavca dol povzroˇcila, da se izuri delavec na juˇzni strani glavne hiˇse.

|  |  |  |  |
| --- | --- | --- | --- |
| Ime figure | Akcije | Zdravje | Strosˇek izdelave |
| polje zlata | / | 10 | 0 |
| delavec | smeri premikanja, vojaˇsnica, glavna hiˇsa, naberi in vrni zlatnike, zdravi | 10 | 1 |
| vojasˇnica | vojak, zdravi | 10 | 4 |
| vojak | smeri premikanja, napad,  zdravi | 20 | 2 |
| glavna hiˇsa | delavec, zdravi | 30 | 7 |

Tabela 4.4: Tu so ˇse opisane figure z njihovimi akcijami in nastavljenimi atributi.

zaporedju, je napadena. Rezultat tega je gradnja hiˇs in urjenja figur v spodnji levi kot ˇsahovnice, saj so izbrana prva polja v zaporedju, kot na primer x-1, y+1, in ˇsirjenje proti zgornjim desnim kotom, ko so vsa ostala polja zasedena, oziroma tam ni nasprotnikovih figur.

Algoritem smo popravili tako, da smo spremenili te akcije (razen naberi zlatnike in vrni zlatnike), da so posamezne akcije za vsako izmed ˇstirih sosednjih polj. Za vpeljavo posameznih akcij smo se odloˇcili, ker ne moramo izbrati polja po zgornjem zaporedju tako, da bi bilo za oba igralca enako. Sedaj je igra scela uravnoveˇsena glede na igralca.

Popravek za to bi bilo doloˇcitev zgornjih navedenih akcij za vsako izmed teh polj, kar bi se prevedlo v veliko veˇcji prostor akcij.

## 4.3 Kodiranja

Doloˇciti smo ˇse zaˇcetno stanje vsake igre, kjer sta igralca postavljena v sredino mreˇze z njihovima glavnima hiˇsama, zraven njiju pa ima vsak igralec svoje polje zlata. Vsakemu igralcu se doda na zaˇcetku doloˇcena koliˇcina denarja za izgradnjo zaˇcetnih delavcev. V naˇsem primeru je bilo to 1, tako da je lahko izgradil samo enega delavca.

Sedaj pa smo potrebovali zakodirati to stanje igre, v numeriˇcni prikaz, ki ga bo nevronska mreˇza lahko interpretirala. To stanje lahko zakodiramo z desetiˇskim kodiranjem, vendar obstaja moˇznost, da nevronska mreˇza sloni proti boljˇsim obravnavanjem pozitivnih ˇstevil za igralca +1, kot za igralca -1. Ravno iz tega razloga obstaja kodiranje z enico v zapisu vsakega stanja (ang. one-hot encoding), ki spremeni desetiˇska ˇstevila v binarni vektor 4.3.2.

Akciji naberi in vrni zlatnike, ostanetaˇse vedno po 1 akcijo, ker za delavca ni razlika, iz katerega sosednjega polja zlata vzame zlatnike, kot ni razlike pri vraˇcanju njih.

### 4.3.1 Desetiˇsko

Pri desetiˇskim kodiranjem, predstavimo vsak atribut figure z eno desetiˇsko

ˇstevilko. Ker imamo figure s 6 atributi, lahko stanje zakodirane igre predstavimo z dimenzijami ˇsirina x viˇsina x 6.

Igralec predstavlja ˇstevilko -1 za igralca -1, 1 za igralca 1 in 0 za prazno polje.

### 4.3.2 One-hot

Dimenzija zakodiranega prostora je tako 8 x 8 x 22, kar je 3.6-krat ˇstevil, ki kodira posamezno stanje igre. To zna oteˇziti uˇcenje nevronske mreˇze, ker ima s tem kodiranjem veˇc ˇstevil, pri katerih mora ugotoviti primernost posameznega ˇstevila.

## 4.4 Konec igre

Konec igre se izvede pod doloˇcenimi pogoji:

* igralec nima za izvesti veˇc nobene moˇzne akcije,

|  |  |  |
| --- | --- | --- |
| Ime kodirnega  polja | sˇt kodirnih bitov | Opis |
| ime igralca | 2 | figura na polju je lahko predstavljena s 2 biti zaradi treh razliˇcnih moˇznosti: 00 predstavlja prazno polje, 01 predstavlja igralca 1 in 10 igralca -1 |
| tip figure | 3 | predstaviti moramo 5 razliˇcnih figur, kar lahko zakodiramo z najmanj 3 biti |
| trenutno zdravje | 5 | nekatere figure imajo veliko ˇzivljenjskih toˇck (npr. glavna hiˇsa 30), za kar moramo uporabiti 5 bitov. Uporabili smo veˇcje ˇstevilo ˇzivljenjskih toˇck, tako da lahko uspeˇsno deluje ranjujoˇca funkcija 4.4, da figure ne eliminira prehitro |
| nosi zlatnike | 1 | zastavica ki se postavi na 1, ko figura delavec nosi zlatnike, drugaˇce pa je postavljena na 0 |
| denar | 5 | za ta kodirnik smo uporabili veˇcje ˇstevilo, saj pustimo da igralec gradi ekonomijo in shranjuje denar, da ga potem lahko na hitro zapravi na figurah, ko ga ima dovolj za njihovo izgradnjo. To lahko privede v zanimive taktike hranjenja denarja in za tem hitro izgradnjo vojaˇskih enot za napad nasprotnikovih figur |
| cˇas igranja | 11 | 211 = 2048, kar pusti igralcu dovolj ˇcasa da odkriva nove poteze, ampak ga dovolj hitro omeji, da se konˇca igra in zaˇcne nova |

Tabela 4.5: Predstavitev ˇstevilo kodirnih atributov za posamezni kodirnik in pojasnitev odloˇcitve za ˇstevilo kodirnih bitov.

* vse figure igralca so uniˇcene,
* ko se izteˇce ˇcas.

Doloˇciti smo morali sintentiˇcen konec igre, saj se lahko igra zacikla tako, da se na primer delavci premikajo v ciklu in se tako igra nikoli ne konˇca. Hoteli smo prioritizirati aktivne igralce, ki nabirajo zlatnike in imajo veˇc enot kakor nasprotni igralec.

### 4.4.1 Reˇsevanje problema neskonˇcnega ˇstevila potez

Cakanje, da se ˇcas igre izteˇce, je problematiˇcno, saj uˇcenje modela poteka zeloˇ poˇcasi, ˇse posebej ko MCTS raziskuje prostor. Za to smo razvili funkcijo, ki prisili model k izvajanju akcij v zgodnem ˇcasu igre, drugaˇce se figuram preveˇc zaˇcnejo zmanjˇsevati ˇzivljenjske toˇcne in so zato eliminirane s ˇsahovnice.

Vidimo, da je krivulja zmanjˇsanja ˇzivljenjskih toˇck veliko bolj stroga in se v igri zaˇcenja ˇze zelo zgodaj. To je zato, ker ˇzelimo hitro odpraviti nedejavnih igralcev in tiste, ki zbirajo zlatnike in pridobivajo nove figure. Vidimo tudi y osi od 0 do 64, kar je najveˇcje ˇstevilo igralcev za enega igralca, tako da bo pri pribliˇzno 2000 korakih vsaka figura dobil smrtno poˇskodbo, zato ˇcasovni potek nikoli ni doseˇzen.

Figure lahko tudi uporabijo akcijo zdravljenja, s ˇcimer poveˇcajo trenutno zdravje doloˇcene figure do najveˇc njenega maksimuma.

Z zgoraj doloˇceno funkcijo je bil problem doloˇciti pravˇsnjo stopnjo koliˇcine zmanjˇsanja ˇzivljenjskih toˇck figur in izloˇcevanje neaktivnih igralcev dovolj zgodaj v igri, in sicer problem z balansiranjem koliˇcine in stroˇskom zdravljenja. Ce so bili stroˇski dovolj nizki, so igralci stalno samo zdravili figureˇ in konˇcali igro pri pribliˇzno tisoˇc potezah, kar je pregloboko za normalno igro. Za to je bilo potrebno poviˇsati stroˇsek zdravljenja, kar je pa privedlo do hitrejˇsega nenadnega umiranja figur, saj igralci niso imeli dovolj kovancev za zdravljenje, na kar je bilo potrebno poviˇsati koliˇcino vrnjenih kovancev iz figure zlato.

### 4.4.2 Ustavitveni pogoj

Ustavitveni pogoj deluje tako, da se na ˇstevilu doloˇcenih potez igra preprosto prekini in oceni zmagovalca po eni izmed spodaj navedenih formul. Igra se prekine po na primer 100 ali 200 potezah, ˇce si do takrat igralca med sabo ˇse nista uniˇcila figur. V spodnjih treh enaˇcbah sta oznaˇcena igralec 1 z oznako p1 in igralec 2 z oznako p2.

**Izrek 4.1** *Prvi: igralec 1 zmaga, ˇce ima veˇc denarja kot igralec 2*

*p*1*.zlatniki > p*2*.zlatniki* (4.1)

Prvi izrek je dober ustavitveni pogoj, za testiranje igralcev pri nabiranju zlatnikov, kjer zmaga preprosto tisti, ki jih nabere veˇc.

**Izrek 4.2** *Drugi: igralec 1 zmaga, ko je seˇstevek zdravja vseh figur igralca 1 je veˇcji od seˇstevka zdravja vseh figur igralca 2*

#### X X

*p*1*.figure.zdravje > p*2*.figure.zdravje* (4.2)

Ce je pogoj za zmago veˇcje ˇstevilo ˇzivljenja svojih figur kakor nasprotni-ˇ kovih hkrati pomeni, da lahko igralec nabira veˇc zlatnikov in z njimi gradi nove hiˇse in uri nove enote, kar zagotavlja za igralca veˇcjo skupno vsoto ˇzivljenja figur in hkrati zagotavlja teˇzo k urjenju vojaˇskih enot z namenom, da nasprotnikovim enotam zmanjˇsa ˇstevilo ˇzivljenjskih toˇck.

**Izrek 4.3** *Tretji: igralec 1 zmaga, ko je seˇstevek zdravja vseh figur igralca 1 plus njegov denar je veˇcji od seˇstevka zdravja vseh figur igralca 2 plus njegov denar*

#### X X

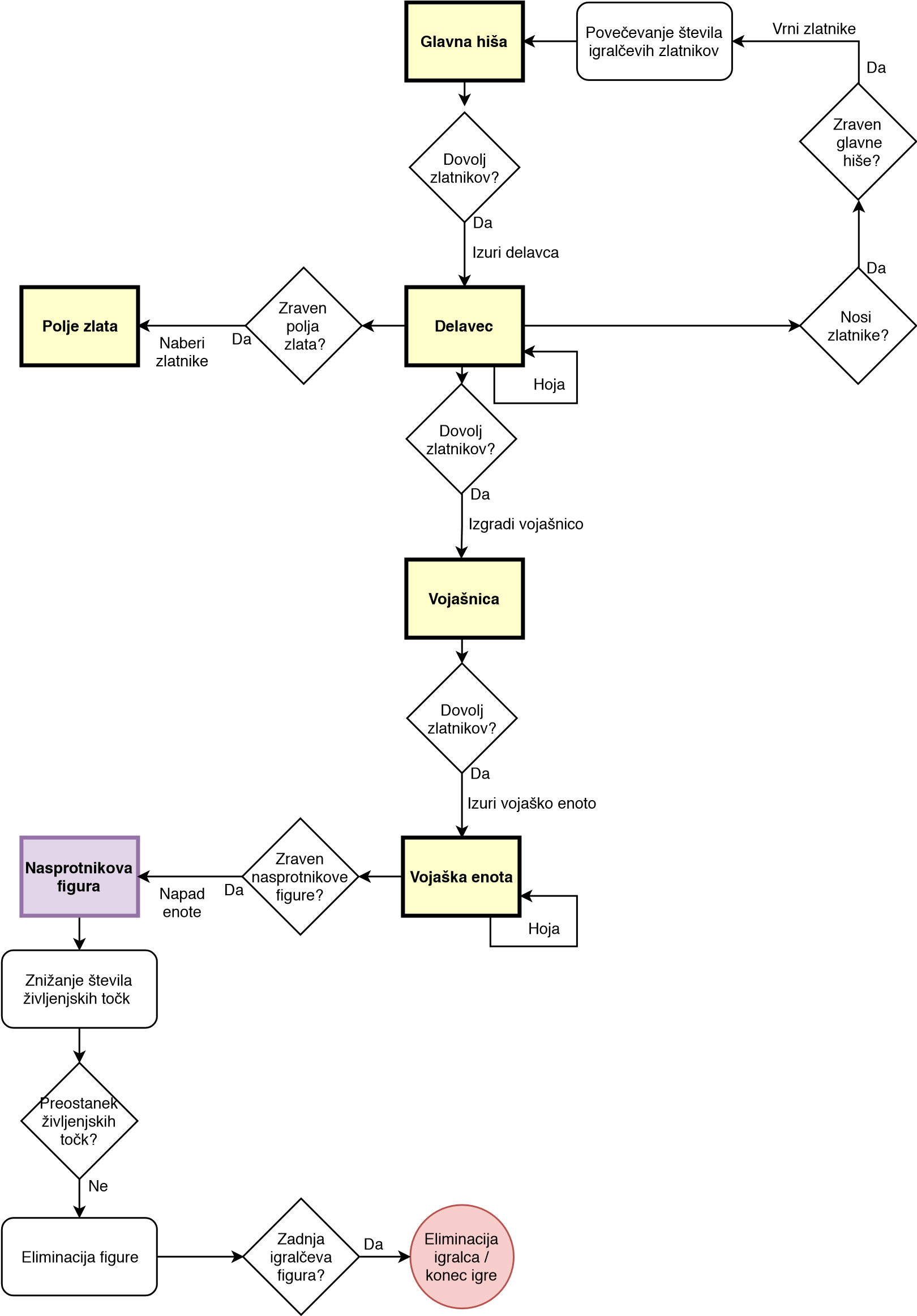
*p*1*.figure.zdravje* + *p*1*.zlatniki > p*2*.figure.zdravje* + *p*2*.zlatniki*

(4.3)

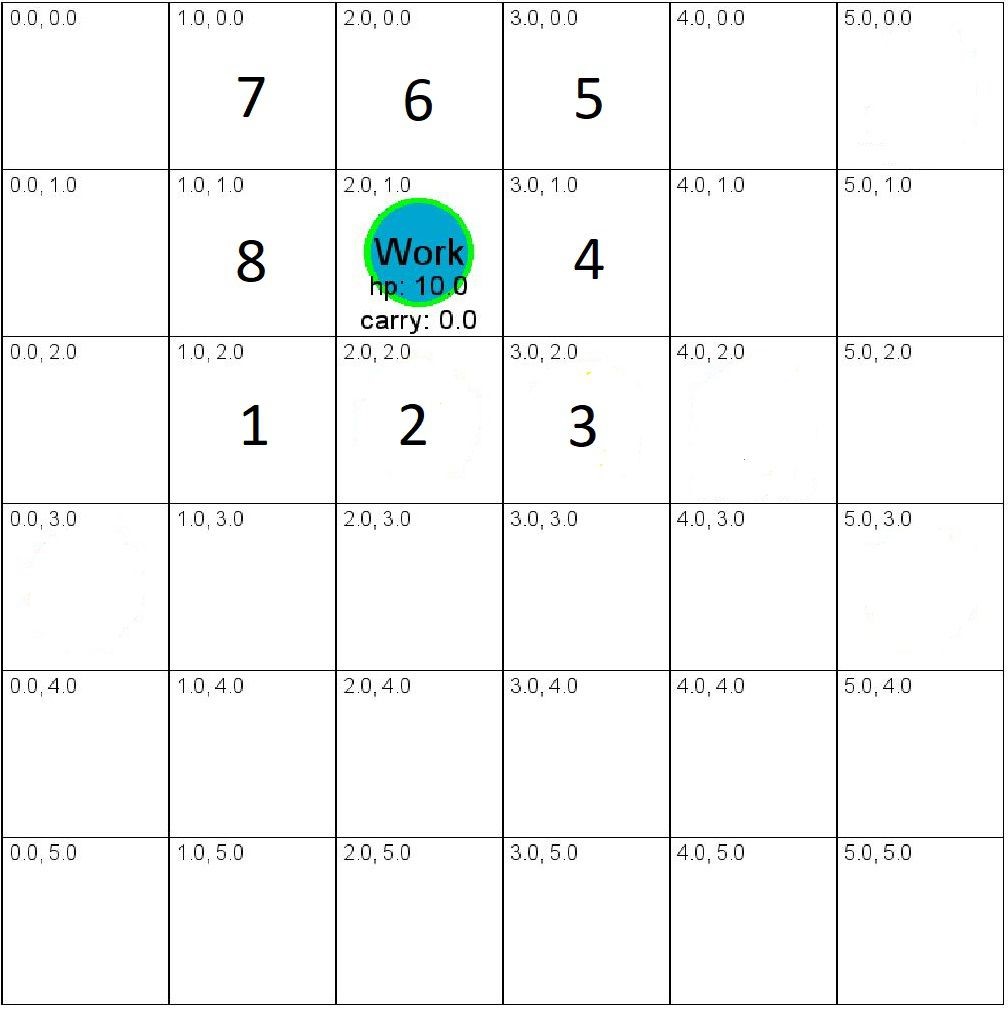
K drugemu izreku smo ˇse pripeli trenutno ˇstevilo shranjenih zlatnikov igralca, kar dodatno doprinaˇsa motivacijo igralca k nabiranju novih zlatnikov. V veˇcini uˇcnih primerov, kot smo to opisali v poglavju rezultati 7, smo uporabljali tretji ustavitveni pogoj 4.3, saj zdruˇzuje tako ˇzivljenjske toˇcke enot, kot tudi denarja. Vendar imajo figure obiˇcajno veliko veˇc ˇzivljenjskih toˇck kolikor so vredne denarja, tako da je izgradnja nove figure za igralca primernejˇsa, kot pa da bi denar shranjeval.



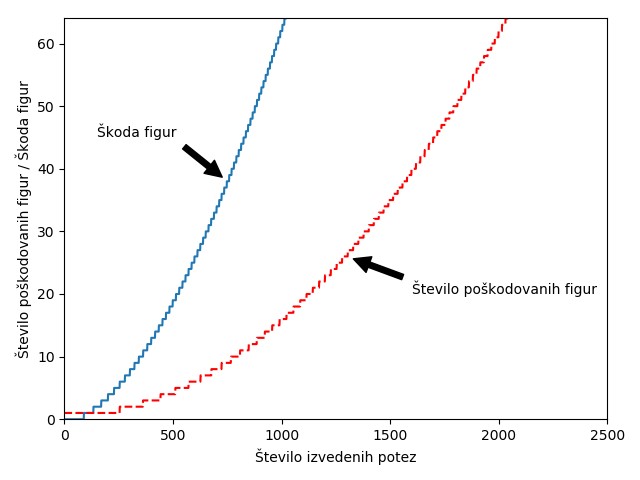
Slika 4.1: Na zgornji sliki je prikaz poˇziranja figur v igri ˇsah, kjer se figura kmet lahko premakne na eno izmed dveh diagonalnih polj in s tem uniˇci trenutno nasprotnikovo figuro na tem polju. Na spodnji sliki pa je prikaz vojskovanja delavca z volkom v igri Age of Empires II, kjer se figurama odˇsteva ˇstevilo ˇzivljenjskih toˇck med napadanjem.



Slika 4.2: Slika prikazuje diagram poteka moˇznih akcij in njihovih rezultatov ob doloˇcenih pogojih.



Slika 4.3: Na sliki je s ˇstevilkami oznaˇceno zaporedje, v katerem se izvedejo zgoraj navedene akcije. Sprva se vzpostavijo sosednje koordinate, za tem pa se po zaporedju od prve do zadnje koordinate preverja veljavnost polj. Ko je doseˇzeno prvo veljavno polje, se izbere to polje kot primerno.



Slika 4.4: Na grafu sta narisani dve funkciji, ki doloˇcata zmanjˇsanje

ˇzivljenjskih toˇck se obravnava v doloˇceni figuri v danem ˇcasu igre(modra) in koliko igralcev je bilo poˇskodovanih v trenutnem ˇcasovnem okviru(rdeˇca).

**Poglavje 5**

# Uˇcenje modela

Do sedaj smo definirali delovanje algoritma Alpha Zero in njegovih komponent, opredelili pravila igre in pri tem pazili da sta med seboj kompatibilna. Ena izmed glavnih komponent uˇcnega postopka pa je seveda nevronska mreˇza, ki hrani moˇci povezav doloˇcenih akcij ob doloˇcenem stanju igre. V tem poglavju bomo sprva predstavili zgradbo nevronske mreˇze iz tehniˇcnega vidika, nato se bomo posvetili prestavitvi parametrov, ki so nastavljivi pri postopku uˇcenja in od njih je odvisno, koliko ˇcasa in na kakˇsen naˇcin se bo naˇs model uˇcil ob nastavljeni konfiguraciji igre. Proti koncu poglavja se bomo ˇse poglobili v moˇzno tehniko postopnega uˇcenja modela, kjer inkrementalno dodajamo zahtevnejˇse konfiguracije in pravila igre.

## 5.1 Zgradba nevronske mreˇze

V sklopu te diplomske naloge se nismo podajali v spreminjanje zgradbe nevronske mreˇze, temveˇc smo vzeli ˇze izgrajeno nevronsko mreˇzo, primerno za uˇcenje igre Othello. To ni najprimernejˇsi pristop, kar je mogoˇce tudi poslabˇsal zmoˇznost in hitrost uˇcenja modela, o ˇcemer smo veˇc prediskutirali v poglavju 8.

Uporabili smo modul Keras znotraj TensorFlow knjiˇznice, za implementacijo modela nevronske mreˇze. Programska koda za izgradnjo modela je laˇzje

35

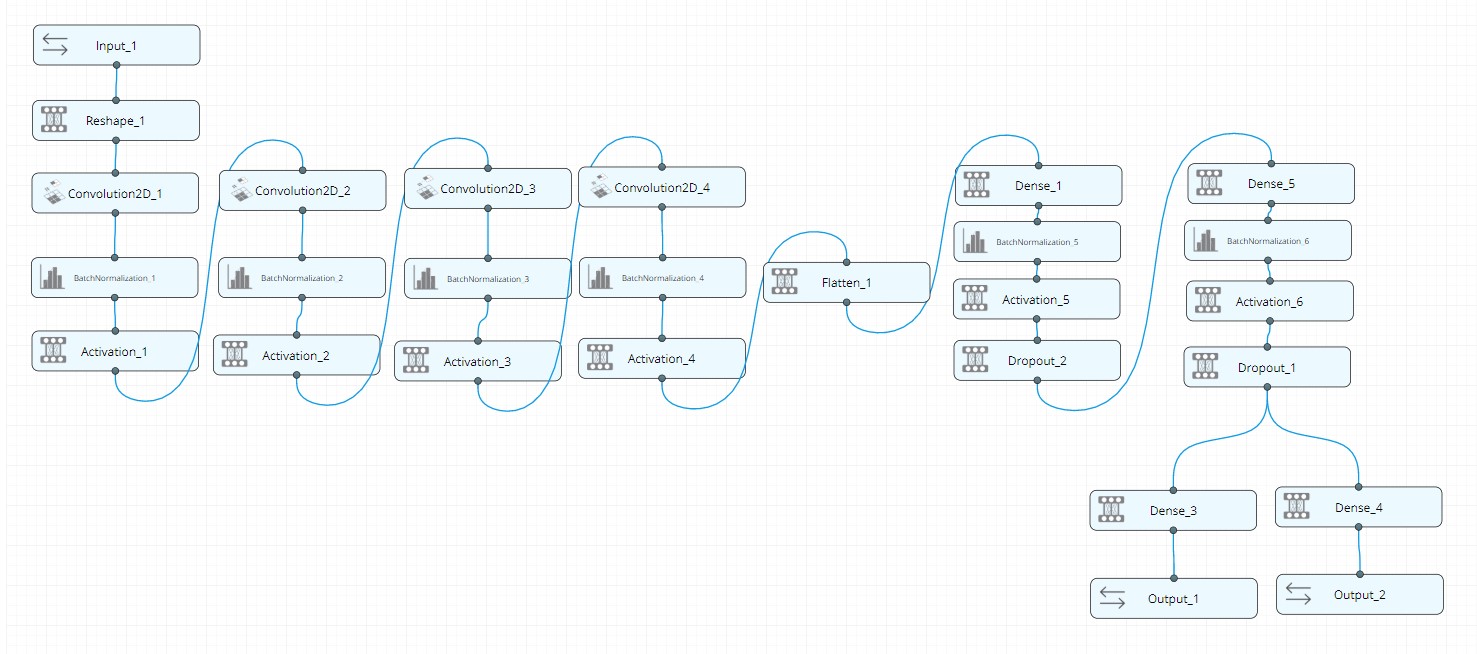
berljiva v modulu Keras kakor v TensorFlow, zato smo se zanj tudi odloˇcili. Ker pa je Keras impementiran znotraj TensorFlow knjiˇznice od verzije 1.9 izdani leta 2017, ni bilo veˇcjih teˇzav z inˇstalacijo te knjiˇznice na odjemalˇcevem raˇcunalniku z vtiˇcnikomm tensorflow-ue4.

Model za vhod vzame uˇcne mnoˇzice stanja iger dimenzij ˇsirina x viˇsina x ˇstevilo kodirnikov, kar je v naˇsem primeru 8 x 8 x 6. Potem gre ta uˇcna mnoˇzica skozi 4 konvolucijske nivoje, kjer je velikost filtra 3. Prva dva konvolicijska nivoja imata oblogo niˇcel okrog matrike, tako da se velikost konvolucijskega nivoja ne zmanjˇsa, druga dva pa tega obloge nimata, kar zniˇza velikost nivoja iz 8 na 4. tako da je izhod zadnjega konvolucijskega nivoja dimenzije velikost serije x (ˇsirina - 4) x (viˇsina - 4) x ˇstevilo kanalov Vsaka izmed konvolicijskih nivojev se normalizira z ”normalizacijo serije”, ki normalizira aktivacijo prejˇsnjega nivoja ob vsaki seriji, ti pa se spustijo skozi relu aktivacijsko funkcijo. Za tem se izhod zadnjega nivoja normalizirane konvolucije izravna v 1-dimenzionalni vektor in se poda dvema polno povezanima nivojema, ki sta ponovno normalizirana z ”normalizacijo serije”. Tedva nivoja sta potem ponovno spuˇsˇcena skozi aktivacijsko funkcijo relu podana v Dropout funkcijo, ki prepreˇci prekomerno prileganje.

Za tem je izgrajen polno povezan nivo Pi, ki ima toliko ˇstevilo izhodov, koliko je moˇzno ˇstevilo akcij v igri za vsako celico, ki ima aktivacijsko funkcijo softmax, prav tako je pa izgrajen polno povezan nivo V, ki ima en izhod, ki predstavlja zmago ali poraz z tanh aktivacijsko funkcijo. Za izhod Pi se nastavi funkcija izgube kategoriˇcna preˇcna entropija, za izhod V pa srednja napaka korena (mean-squared error).

## 5.2 Predstavitev parametrov

Uˇcni algoritem uporablja mnoˇzico parametrov za uˇcenje. Ti predstavljajo razliˇcno ˇstevilo iteracij igranja iger (iteracije in epizode igranja), kot tudi iteracij uˇcenja nevronske mreˇze (epohi). Predstavljajo tudi same nastavitve raziskovanja algoritma MCTS (Cpuct) in ˇstevilo primerjav, kateri nauˇcen Diplomska naloga *37*



Slika 5.1: Model nevronske mreˇze, uporabljen za uˇcenje igre. Izgrajen je iz

ˇstirih konvolucijskih nivojev, med katerimi je izhod normaliziran in spuˇsˇcen skozi relu aktivacijsko funkcijo. Nato se poda izhod zadnjega konvolucijskega nivoja v dva polno povezana nivoja, za katerima se doloˇci izhod V in Pi, ki predstavljata napoved zmage (V) in napovedi verjetnosti akcij (Pi). Slika je bila izdelana z uporabo aplikacije Deep Cognition na voljo na https://deepcognition.ai/.

model je boljˇsi.

V tabeli opisa parametrov 5.1 so predstavljeni najpomembnejˇsi parametri, ki smo jih spreminjali ob uˇcenju modela na naˇsi RTS igri.

## 5.3 Postopno uˇcenje

Uˇcenje te igre je zapleteno zaradi pogojev konca igre. Algoritem izvaja igranje igre dokler ne doleti do ustavitvenega pogoja, ta pa pri RTS igri lahko ni nikoli doseˇzen, saj se lahko vojaˇska enota stalno premika v istem krogu, kakor bi se lahko trdnjava vedno premikala samo po dveh istih poljih. Cikel akcij se lahko reˇsi z uporabo ˇcasovnih omejitev, kjer se igranje igre ustavi ko se izteˇce ˇcas, vendar to povzroˇca ˇse povrˇsnejˇsi pribliˇzek ocene stanja igre, ki pa vpliva na uˇcenje nevronske mreˇze. Zaradi kompleksnejˇsega ustavitvenega po-

|  |  |  |
| --- | --- | --- |
| Ime parametra | Privzeta vrednost | Opis |
| Cpuct | 1 | parameter drevesnega raziskovanja, ki vpliva direktno na raziskovanje MCTS algoritma. Veˇcja kot je vrednost parametra, bolj bo MCTS dajal prednost neraziskanim vozliˇsˇci. V naˇsem primeru smo izbrali vrednost 1 in s tem nismo prispevali k dodatni uteˇzi k raziskovanju. |
| sˇtevilo iteracij | 30 - 60 | predstavlja ˇstevilo uˇcenj algoritma na odigranih igrah in ˇstevilo izbire boljˇsega modela. Parameter tudi predstavlja, kolikokrat se bo igra odigrala v celoti, kjer bo konec predstavljal konˇcni pogoj oziroma eliminacija nasprotnika. |
| sˇtevilo epizod | 8 | nam zagotovi pridobitev dovolj velikega nabora odigranih iger, nad katerimi se potem algoritem uˇci. Vsako epizodo se odigra ˇstevilo iger, kolikor je nastavljeno s parametrom ˇstevilo iteracij. |
| sˇtevilo MCTS iskanj | 30 - 50 | predstavlja ˇstevilo raziskanih vozliˇsˇc v iteraciji igre. MCTS iskanja se ne izvrˇsijo do konca igre, ampak do neraziskanega vozliˇsˇca oziroma ˇce je raziskano vozliˇsˇce konec igre. Prav tako bodimo pozorni na majhno ˇstevilo MCTS iskanj (30 - 50), za razliko od tradicionalnih MCTS algoritmov, ki na primer v igri ˇsah odigrajo veˇc 10-tisoˇc iteracij [5]. |

Tabela 5.1: Tabela 1/2 o opisu parametrov.

Diplomska naloga *39*

|  |  |  |
| --- | --- | --- |
| Ime parametra | Privzeta vrednost | Opis |
| sˇtevilo primerjanj modelov | 10 | kolikokrat se bosta trenutni model k se uˇci in njegova prejˇsna razliˇcica pomerila med sabo, da se ohrani boljˇsi. Nauˇcena modela se med sabo pomerita z igranjem iger od zaˇcetka do konca, kjer je rezultat zmaga nekoga izmed modelov, oziroma neodloˇceno. |
| sˇtevilo iteracij ucˇnih primerov | 8 | nam zagotavlja, da ohranjamo novejˇse uˇcne primere in starejˇse zavrˇzemo. Vsako epizodo se doda nova zbirka iger ter odstrani najstarejˇsa, ˇce ˇstevilo shranjenih epizod presega ta parameter. To nam zagotavlja dovolj sveˇzo uˇcno mnoˇzico, nad katero se nevronska mreˇza uˇci. V naˇsem primeru je bil nastavljen na manjˇso vrednost (8), saj kodiranja stanj naˇse igre zasedejo veliko pomnilnika. |
| epohi | 100 | Stevilo iteracij uˇcenja nevronske mreˇze skozi uˇcne pri-ˇ mere. |

Tabela 5.2: Tabela 2/2 o opisu najpomembnejˇsih uˇcnih parametrov, uporabljenih pri uˇcenju algoritma Alpha Zero. Pri parametrih so zapisane tudi njihove okvirne oziroma najveˇckrat uporabljene konfiguracijska ˇstevila.

goja lahko model hitro prekomerno prilagodi na napaˇcno igranje igre. Dober primer je ustavitveni pogoj z vsoto zdravja igralˇcevih enot in njegovih trenutnih zlatnikov 4.3, kjer se igralca ne nauˇcita pravilno napadati nasprotnikovih enot, da zniˇzujeta nasprotnikove ˇzivljenjske toˇcke, ampak konstantno nabirata nove zlatnike in gradita in urita nove enote. Ker igralca nista nauˇcena zakljuˇciti igro z eliminacijo nasprotnikovih enot.

Iz tega razloga, se nam je porodila uˇcna ideja s postopnim uˇcenjem modela. Ideja govori o spreminjanju pravil iger in nastavitev parametrov med uˇcenjem. S tem bi lahko v zaˇcetnih fazah uˇcenja prioritiziral na primer nabiranje zlatnikov in s tem nauˇcil algoritem uspeˇsnega in hitrega nabiranja, kot na primer ustavitveni pogoj 1 4.1. Za tem bi ta nauˇceni model nagradil na tak naˇcin, da bi mu spremenil ustavitveni pogoj na 2 4.2 in h konfiguraciji akcij dodal moˇznost gradnje stavb, s ˇcimer bi zdajˇsnji model, ki dobro nabira zlatnike z nadaljnjim uˇcenjem ˇse nadgradil da poleg nabiranja ˇse gradi hiˇse in s tem poveˇcuje ˇstevilo ˇzivljenjskih toˇck enot, ki jih igralec obvladuje. Ta model bi pa v tretji fazi nadgradil ˇse z akcijami urjenja vojaˇskih enot in napadom drugega igralca.

Poskusili smo nauˇciti model po zgoraj opisanem postopku, vendar uˇcenje ni potekalo uspeˇsno. Nevronska mreˇza je vzpostavila zaˇcetna stanja vrednosti vozliˇsˇc akcij po vnaprej nauˇcenih uteˇzeh pridobljenih iz nauˇcenega modela, ki pa v fazi 2 ali 3 ni vseboval novo dodanih akcij kot gradnja stavb oziroma napadanje. Zaradi majhnega ocenjenega stanja novih akcij, so te manjkrat obiskane in zato manj raziskane. Algoritem zaradi sprememb pravil igre deluje zelo nakljuˇcno, saj nauˇcene uteˇzi delujejo v nasprotju igralˇcevih ˇzelja. MCTS pri tem iskanju ne pripomore veliko, saj so neraziskana stanja igre vzpostavljena glede na uteˇzi iz nevronske mreˇze, stanje pa se popravi ko doseˇze konˇcno stanje, kar se pa zgodi redko.

**Poglavje 6**

# Vizualizacije

## 6.1 Pygame

S Python knjiˇznico Pygame smo izdelali vizualizacijo, ki je primerna za pregled igre med samim razvijanjem. Sahovnica je oznaˇcena s ˇcrtami, medˇ katerimi so s krogi izrisane figure, kjer njihove barve predstavljajo svoj tip figure in obroba krogca igralca -1 ali +1. V krogcih je tudi napisano zdravje za to figuro in zastavica, ali delavec prenaˇsa zlato. Zgoraj je izpisano, koliko denarja ima posamezen igralec in koliko potez sta igralca ˇze odigrala. Prav tako so izpisane vse moˇzne akcije, ki jih igralec lahko izvrˇsi z doloˇceno figuro.

Igralec lahko nadzoruje svoje figure s tipkovnico in miˇsko. Uporabnik mora najprej izbrati figuro z levim miˇskinim klikom in potem izbrati doloˇceno akcijo, ki je izpisana na zaslonu. Uporabnik lahko spremeni figuro, tako da jo odznaˇci s klikom desnega miˇskinega gumba na prazno mesto.

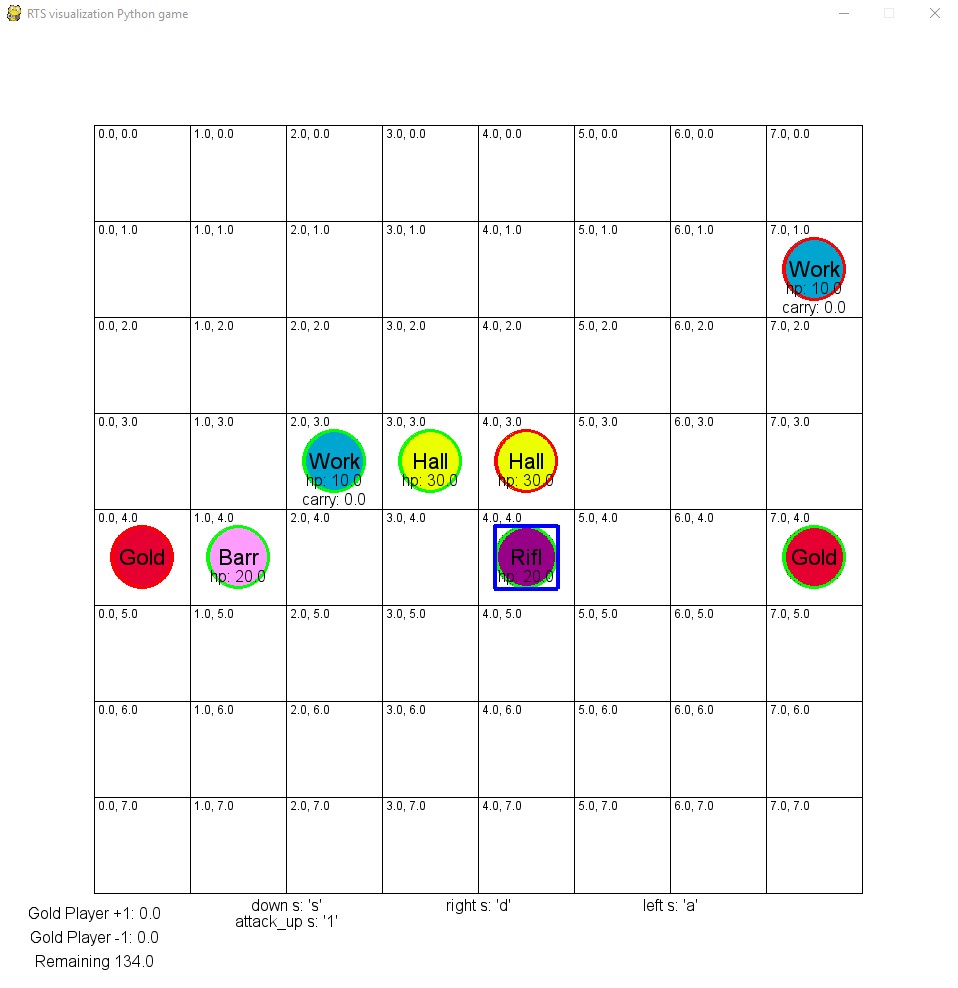
* premikanje: igralec lahko premakne delavce in vojake za 1 kvadratek v vseh 4 smereh ˇce so prazni s klikom na eno od 4 mest,
* napadanje: z izbrano vojaˇsko figuro lahko uporabnik napade nasprotnikove figure, ki so v dosegu,
* zbiranje in vraˇcanje sredstev: z izbranim delavcem lahko uporabnik porabi sredstva, tako da klikne desno miˇskino tipko na polje zlata, ˇce

41

je v dosegu. To velja tudi za vraˇcanje sredstev, vendar mora biti delavec v bliˇzini glavne hiˇse,

* gradnja: Za gradbene figure in zgradbe mora uporabnik uporabiti eno od bliˇznjic na tipkovnici.

Prav tako lahko uporabnik igra igro s pisanjem akcij v konzolo.



Slika 6.1: Na zgornji sliki ˇcloveˇski igralec igra izgrajeno strateˇsko igro proti raˇcunalniˇskim nasprotnikom.

Diplomska naloga *43*

## 6.2 Unreal Engine 4

Celostni pogon Unreal Engine 4 (ang. game engine Unreal Engine 4; UE4) je odprto-kodni program podjetja Epic Games, ki je namenjen hitri izdelavi raˇcunalniˇskih iger. Obstajajo ˇse drugi celostni pogoni kot je na primer Unity. Unreal Engine 4 omogoˇca hitro ustvarjanje iger s pomoˇcjo posebnih diagramov (ang. blueprint) in hkrati podpira programski jezik C++, ki ga uporabimo za hitro izvedbo velikega ˇstevila matematiˇcnih izrazov [4].



Slika 6.2: Zgornja slika predstavlja igro izdelano v Unreal Engine 4 z uporabniˇskim vmesnikom. Na sliki je vidna glavna hiˇsa, vojaˇsnica, vojak in delavec, kot tudi polje zlata na levi. Igralec lahko upravlja z vmesnikom za poˇsiljanje proˇsenj za predloge potez, ki je nakazan v vnosnim poljem in gumbi na desni strani slike.

V temu programu smo oblikovali vizualizacijo igre, ki nam je omogoˇcilo bolj moderen in realistiˇcen prikaz realno-ˇcasovne strateˇske igre, saj je vizualiziran seveda v 3-D in ne 2-D kot v Pygame.

V igri lahko izvajamo akcije preko uporabniˇskega vmesnika, kot tudi z bliˇznjicami na tipkovnici. Kompleksnejˇsi uporabniˇski vmesnik nam pa zagotavlja veˇc funkcionalnosti kakor igra izdelana v Pygame. Z njim lahko izbiramo veˇc figur ter jih grupiramo v skupine. Figuram postavljamo veˇc zaporednih akcij, ki jih ta izvrˇsuje v tem vrstnem redu. Cloveˇski igralec imaˇ tudi vpogled na manjˇsi zemljevid, prikazan v levem spodnjem kotu, na katerem vidi nasprotnikove in svoje figure. Nekaj je pa tudi kozmetiˇcnih funkcij kot na primer prekrivanje zemljevida z meglo (ang. Fog of war), dnevnonoˇcni cikel, animacije za vojskovanje, premikanje, nastavitve hitrosti ˇcasa ipd. Igra ima tudi implementirana izvajalna drevesa, kjer na primer agentu naroˇcimo nabiranje zlatnikov, ta pa jih sam vraˇca v najbliˇzje odlagaliˇsˇce zlatnikov. Prav tako so ta drevesa za napadanje, tudi ko je agent nedejaven, da se prosto premika okrog, iˇsˇce stavbe itd. Igra ima implementirane tudi razne zvoˇcne efekte, uˇcinke delcev za prikaz zmanjˇsanja ˇzivljenjskih toˇck nasprotnikovim enotam. V igri lahko tudi stanje igre shranimo na disk in ga pozneje naloˇzimo, da igro lahko igramo naprej. Ob tem zapisovanju igre smo ugotovili pravi postopek, kako zajeti igro v Unreal Engine in ga zapisati v doloˇcen format, ki ga potem lahko laˇzje prenaˇsamo. To nam je koristilo tudi pri kodiranju igre v notacijo za oznaˇcevanje JavaScript objektov (ang. JavaScript Object Notation; JSON), da smo jo lahko poslali nauˇcenem modelu v Python skripto. Igra podpira tudi preprosto analitiko za primerjavo denarja med igralci, tipi figur ipd.

Igra je zasnovana tudi tako, da se lahko dva igralca med sabo pomerita preko mreˇze s spletnim podsistemom Steam ali preko lokalne mreˇze, preko katerega se lahko tudi komunicirata. Oba igralca v tem primeru na svoji lokalnem raˇcunalniku poganjata nauˇcena modela in od njega zahtevata priporoˇcila akcij. Ker pa lahko igralca izbereta veˇc razliˇcnih map, na katerih bosta igrala, bi bilo potrebno za vsako izmed teh map izgraditi svoj model, saj imajo lahko mape drugaˇcne dimenzije v ˇsirini in viˇsini. Prav tako z zdajˇsnim algoritmom niso prokriti primeri, da doloˇceno polje ne bi bilo dosegljivo (voda, skalovje), tako da mape morajo biti kvadratne in vsa polja so dosegljiva. Nekatere izmed zgoraj navedenih funkcij je izdelal Nick Pruehs v vtiˇcniku ue4-rts (https://github.com/npruehs/ue4-rts), kot recimo nekej izvajalnih dreves, zemljevid, prekrivanje zemljevida z meglo.

Potrebno je bilo preslikati akcije in figure v urejevalnik Unreal Engine, Diplomska naloga *45*

da se tam figure primerno premikajo in izvajajo akcije. Potrebno je bilo (mapirati) animacije, efekte, da premikanje in napadanje zgleda dokaj realistiˇcno. Ko igralca priˇcneta z igranjem igre, se naloˇzi TensorFlow model, katerega bosta igralca uporabljala za pridobivanje akcij. Prav tako se inicializira MCTS. Za laˇzjo komunikacijo z Python modulom in vraˇcanje povratnih klicov v engine smo uporabili vtiˇcnik tensorflow-ue4 (Jan Kaniewski https://github.com/getnamo/tensorflow-ue4), ki nadgradi vtiˇcnik UnrealEnginePython(20tab https://github.com/20tab/UnrealEnginePython) s TensorFlow komponento. Ta komponenta se avtomatiˇcno naloˇzi na odjemalˇcevem raˇcunalniku in zagotavlja, da lahko ta uporablja vse funkcionalnosti TensorFlowa.

Ko igralec ali raˇcunalniˇski nasprotnik poda zahtevo za pridobitev akcije, se prvo pridobi vse podatke o igri in se jih mapira v JSON zapis, da se ga potem poˇslje Python skripti. V tem JSON zapisu so zapisane figure s kodirniki (x, y, igralec ,tip figure, zdravje, nosi zlatnike ,zlatniki ,preostal ˇcas). Potem se seveda asinhrono poˇslje Python skripti ta JSON zapis, ta pa izgradi novo ˇsahovnico s figurami na podlagi prejetih zakodiranih figur. Skripti se poda tudi ime igralca, ki zahteva akcijo. Skripta za tem pokliˇce funkcijo za pridobitev verjetnosti akcij, ki izvede doloˇceno ˇstevilo MCTS iteracij in izbere tisto z najveˇcjo verjetnostjo. Python skripta pa vrne koordinati x in y ter akcijo, ki se potem ta izvede v celostnem pogonu s preslikanimi svojimi akcijami za gor, dol, napad, naberi ipd. Potrebno je bilo tudi paziti z orientacijo koordinatnega sistema, saj je bil v Python igri drugaˇce orientiran kot v pogonu Unreal Engine. Potrebno ga je bilo obrniti za -90◦ v osi Z (pogon Unreal Engine 4: +x gor, +y desno, Python igra: +x desno, +y dol)

Ta predstavitev omogoˇca tudi igranje dveh ˇcloveˇskih igralcev enega proti drugemu preko internetne mreˇze, kjer vsak igralec pridobiva priporoˇcene ukaze iz modela. Cloveˇski igralec lahko igra tudi proti raˇcunalniˇskim igral-ˇ cem, ki pa vsake 0.5 sekund zahteva za novo najboljˇso akcijo. Lahko si pa ogledamo dva raˇcunalniˇska igralca igrati drug proti drugemu.

### 6.2.1 Prenos stanja igre

Za vsakega od igralcev, se vzpostavi svoja komponenta za pridobivanje akcij, ki naloˇzi model da je pripravljen na pridobivanje napovedi. V trenutku lahko samo eden od igralcev pridobi napoved, saj pride do konfliktov, ˇce se na primer oba igralca odloˇcita figuro premakniti na isto polje. Ko igralec poˇslje proˇsnjo za napoved, zraven ˇse poˇslje svoje stanje igre, in kateri igralec je tisti, ki poˇsilja proˇsnjo. Na to algoritem nastavi trenutno igro na poslano in izbere najprimernejˇso akcijo. Po izvedbi izbrane akcije, igra poˇcaka ˇse doloˇcen ˇcas, da se akcija izvede do konca, za tem pa lahko ta postopek ponovi ˇse drugi igralec.

Cloveˇski igralec lahko akcijo pridobi kadarkoli, a mora poˇcakati da seˇ trenutna proˇsnja za napoved konˇca. Za njim se postavi v ˇcakalno vrsto tudi raˇcunalniˇski nasprotnik.

**Poglavje 7**

# Rezultati

V temu poglavju bomo predstavili veˇc konfiguracij uˇcenja in njihovih rezultatov v vizualizaciji Pygame. Nauˇcene modele, ki so med seboj kompatibilni, bomo med seboj primerjali in izpostavili, zakaj je zmagovalna konfiguracija boljˇsa od poraˇzene. Za tem bomo nauˇcen model povezali z pogonom Unreal Engine, kjer bosta dva raˇcunalniˇska nasprotnika igrala igro drug proti drugemu ter primer, kjer ˇcloveˇski igralec igra proti raˇcunalniˇskem igralcu ki uporablja nauˇcen model za izbiro akcij.

Pri samem uˇcenju smo uporabili TensorFlow 1.9.0, Python 3.6, za vizualizacijo pa Pygame 1.9.4 in Unreal Engine 4.20.

Za uˇcenje smo na voljo podali vse moˇzne akcije.

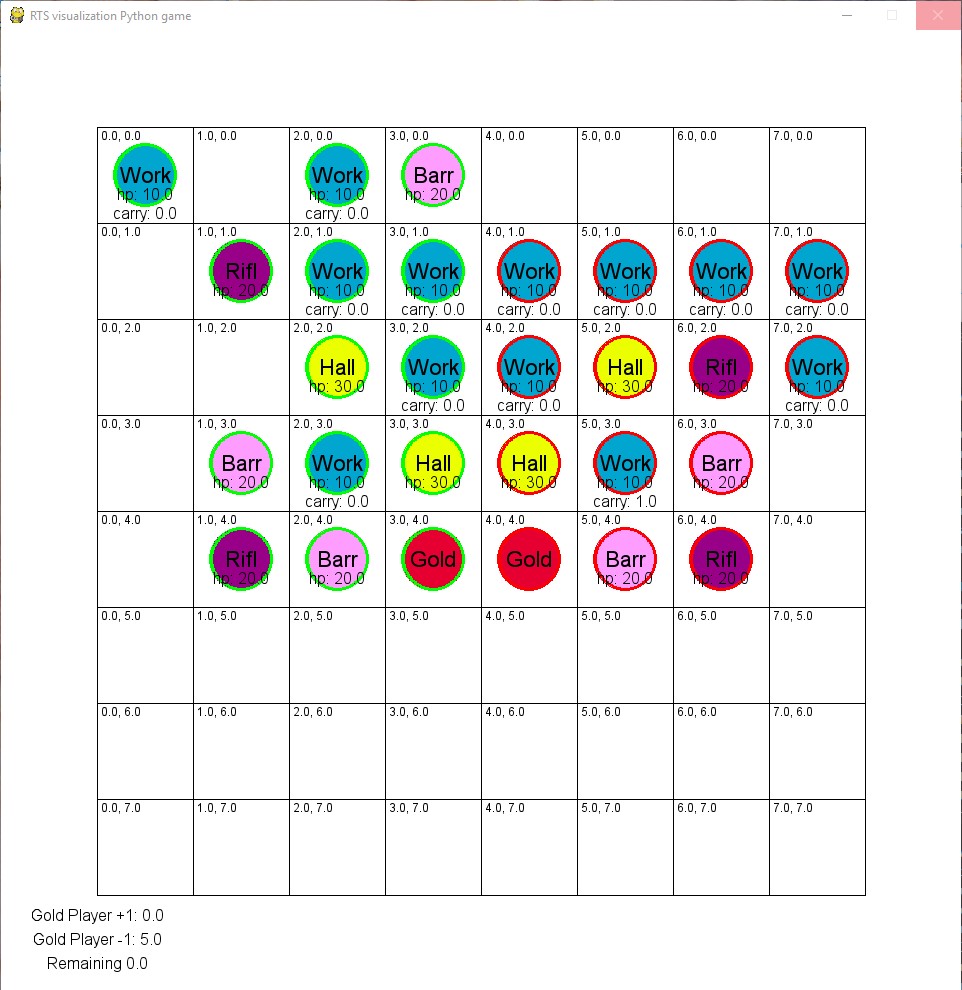
## 7.1 Uˇcenje z ustavitveno funkcijo s ˇcasovno omejitvijo

V prvi fazi smo poskusili uˇcenje modela z manjˇso ˇcasovno omejitvijo (100 potez) ter dovolj velikim ˇstevilom iteracij tako da algoritem potrebuje kar nekaj ˇcasa, da dokonˇca z izvajanjem uˇcenja.

47

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tip konfiguracije | 7.1 | 7.2 | 7.3 | 7.4 |
| cˇasovna omejitev | 100 | 200 | 200 | 200 / fn |
| iteracije | 40 | 20 | 30 + 30 | 20 |
| epizod | 8 | 8 | 8 | 8 |
| MCTS iskanj | 50 | 50 | 30 | 50 |
| primerjave | 20 | 20 | 20 | 20 |
| zgodovina uˇcnih primerov | 8 | 8 | 8 | 8 |
| epohi | 100 | 100 | 100 | 100 |
| zacˇetni zlatniki | 20 | 20 | 1 | 1 |
| povecˇevanje zlatnikov | 5 | 5 | 1 | 1 |
| zmanjsˇanje zˇivljenjskih tocˇk | 20 | 20 | 20 | 20 |
| kolicˇina zdravljenja | 20 | 20 | 20 | 20 |
| strosˇki zdravljenja | 5 | 5 | 5 | 5 |
| sˇtevilo polj | 8 x 8 | 8 x 8 | 8 x 8 | 6 x 6 |

Tabela 7.1: V tej tabeli so napisane konfiguracije definicij igre in uˇcenja za spodaj opisane primere uˇcenja modela.



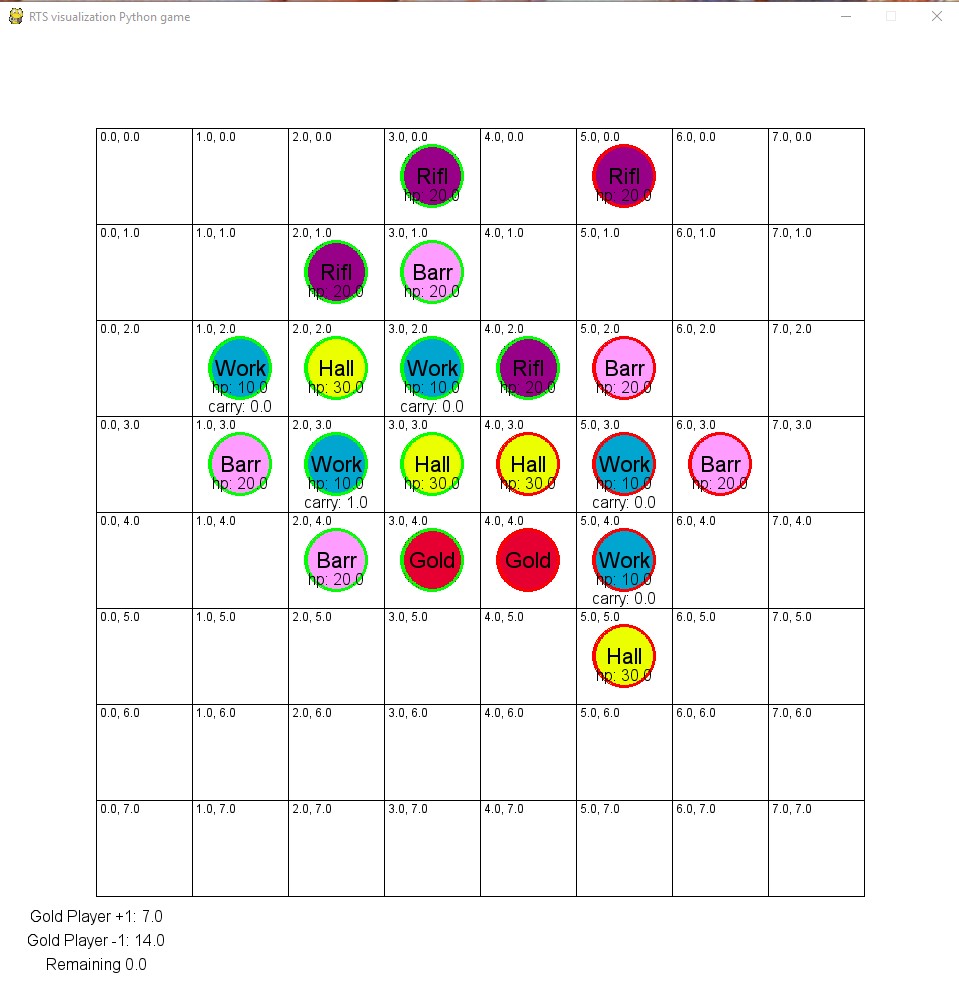
Slika 7.1: Vizualizacija nauˇcenega modela dveh raˇcunalniˇskih nasprotnikov po izteku ˇcasovne omejitve v Pygame. Igralca sta se najbolj osredotoˇcila na nabiranje zlatnikov in mnoˇziˇcni izgradnji cenejˇsih figur. Uporabljen model je nauˇcen z numeriˇcnim kodirnikom. Uˇcenje je potekalo 1 dan in 4 ure.

### 7.1.1 Numeriˇcno

Numeriˇcni kodirnik se je bolj osredotoˇcil na izdelavo delavcev, ki so najcenejˇse figure. Kar pomeni, da takoj ko je igralec imel dovolj denarja za izdelavo figure, jo je izdelal. Igralca sta s svojimi figurami dobro nabirala zlatnike, s tem da sta glavna hiˇsa in polje zlata neposredno drug ob drugem.

### 7.1.2 Kodiranje z enico v zapisu vsakega stanja

Z numeriˇcnim kodiranjem se je algoritem osredotoˇcil bolj na izdelavo vojaˇskih figur in nabiranju samih zlatnikov. Delavca sta stalno nabirala zlatnike in jih vraˇcala v glavno hiˇso. Nabiranje in vraˇcanje zlatnikov je v tem primeru veliko bolj konsistentno kakor z uporabno numeriˇcnega kodirnika. Igralca



Slika 7.2: Vizualizacija nauˇcenega modela dveh raˇcunalniˇskih nasprotnikov po izteku ˇcasovne omejitve v Pygame. Igralca sta se bolj osredotoˇcala na izgradnjo vojaˇskih figur ter napadanju sovraˇznih enot. Uporabljen model je nauˇcen z one-hot kodirnikom. Uˇcenje je potekalo 1 dan in 8 ur.

sta pozornost tudi deloma posveˇcala napadanju sovraˇznik figur, vendar ne do toˇcke, kjer bi z napadanjem sovraˇznega igralca eliminirali.

### 7.1.3 Primerjava

Prav tako smo primerjal numeriˇcno kodiranje proti one-hot kodiranju na enakih konfiguracijah modelov ter igre.

Algoritem, nauˇcen z one-hot kodiranjem ne more delovati pri primerjanju dveh modelov z numeriˇcnim kodirnikom in obratno. Model uˇcen z one-hot kodiranjem se lahko obiˇcajno pomerja samo proti drugemu modelu, ki je bil prav tako nauˇcen s to konfiguracijo. Enako velja za numeriˇcni kodirnik.

Modele z razliˇcnimi konfiguracijami je med sabo teˇzko primerjati, ne moreta igralca imeti razliˇcnih nastavitev za model, razen ˇce jih eksplicitno prepiˇsemo. Primerjanje modelov smo prilagodili tako, da vsak izmed igralcev lahko uporablja drugaˇcno konfiguracijo uˇcnih parametrov 5.2. Za popolno primerjanje dveh razliˇcno nauˇcenih modelov bi potrebovali tudi spremembe parametrov konfiguracije igre 4, da bi lahko vsak izmed modelov deloval s poljubno konfiguracijo pravil igre, kot tudi tip ustavitvene funkcije ipd. Ta sprememba bi preveˇc posegla v Surag Nairjevo razliˇcico algoritma Alpha Zero, kar hoˇcemo ˇcim bolj ohraniti.

Izkazalo se je da one-hot kodiranje prinaˇsa boljˇse rezultate po ocenitvi modelov z igranjem 20 iger drug proti drugemu, kjer je bil rezultat 20:0 za model zakodiran z one-hot naˇcinom.

bi bilo potrebno ˇse predelati, tako da se lahko posameznem igralcu doda konfiguracijo pravil igre in uˇcnih konfiguracij, kot tudi tip ustavitvene funkcije ipd. S tem bi lahko primerjali dva popolnoma razliˇcna modela na isti konfiguraciji igre (npr ˇsahovnica 8 x 8).

## 7.2 Poveˇcevanje ˇcasovne omejitve

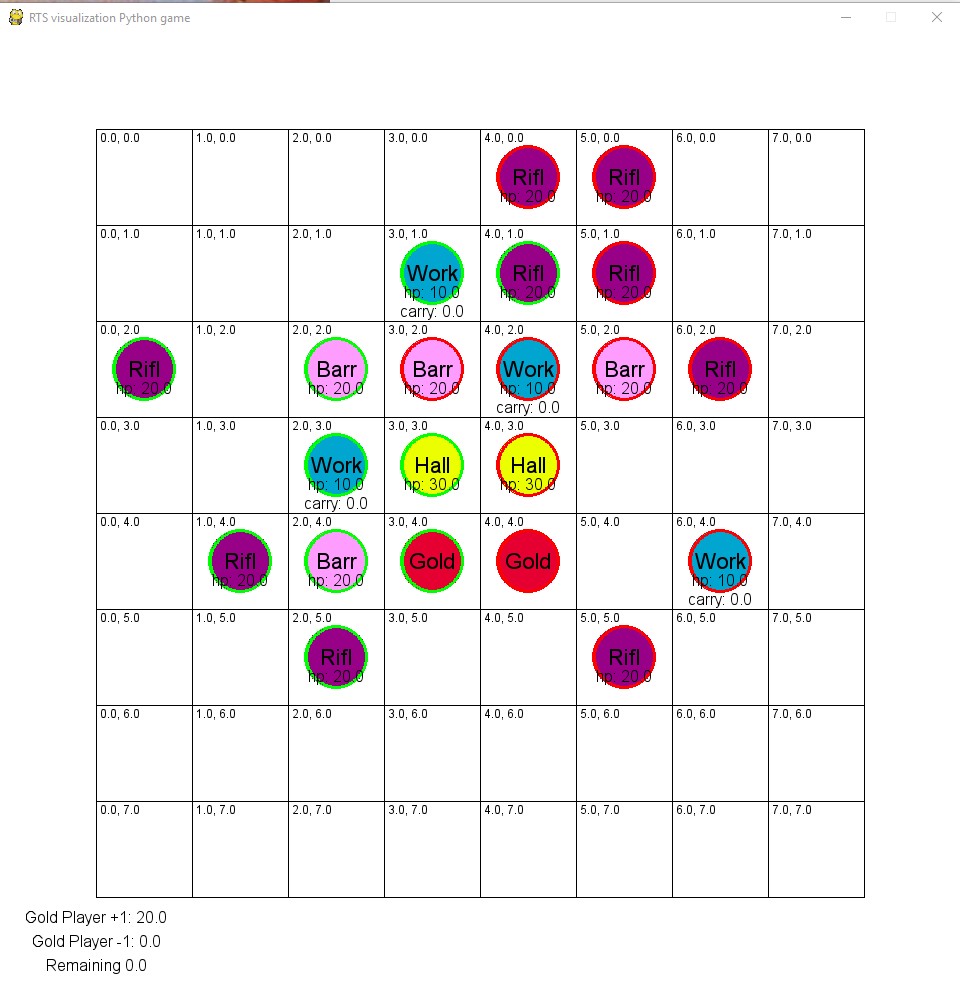
V tem koraku smo izbral kodirnih one-hot, ker se je v primerjavi pri prejˇsnji konfiguraciji obnesel boljˇse.

V tem primeru smo poveˇcali ˇcasovno omejitev na 200 in zniˇzali ˇstevilo iteracij na 20. Poveˇcali smo ˇcasovno omejitev, saj so se pri prejˇsni nastavitvi 100 igre prehitro zakljuˇcevale. Zniˇzali pa smo ˇstevilo iteracija na 20, saj bodo posamezne igre trajale dlje, in smo ˇzeleli pribliˇzno isto ˇcasovno dolˇzino uˇcenja, kot pri prejˇsnjem primeru, da lahko modela vsaj vizualno primerjamo.

Algoritem se je bolj osredotoˇcil na izdelavo vojaˇskih figur in vojskovanja kot v prvi iteraciji uˇcenja z manjˇso ˇcasovno omejitvijo

## 7.3 Sprememba konfiguracij zlata

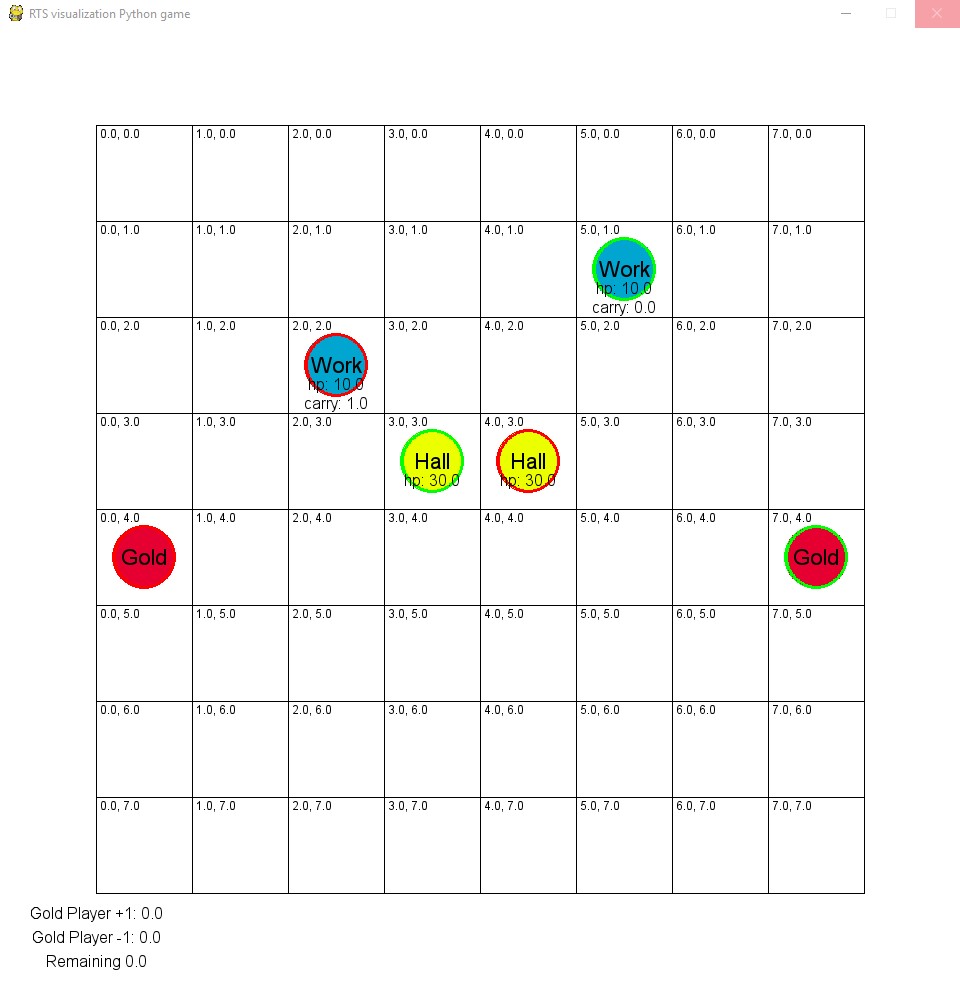
Polji zlata smo pomaknil iz sredine na levi in desni rob, tako da se morajo delavci pomakniti do roba, nabrati zlatnike in jih vrniti nazaj v glavno hiˇso,



Slika 7.3: Vizualizacija nauˇcenega modela v Pygame z poveˇcano ˇcasovno omejitvijo na 200 in zniˇzano ˇstevilo iteracij na 20.

ki je pa ˇse vedno na sredini mreˇze. Prav tako smo spremenili nastavitev, koliko zaˇcetnih zlatnikov imata igralca iz 20 na 1, kar dovoli izgradnjo enega delavca na zaˇcetku igre. Prav tako smo zmanjˇsali koliˇcino vrnjenega denarja iz 5 na 1, kar bi zagotavljalo, da morata igralca izbrati mnogo pravih sekvenc pomikanja do polja zlata, pridobiti zlatnike in jih vrniti v glavno hiˇso, preden bi lahko izgradili novo enoto.

Model smo gradili 48 ur v dveh delih po 30 iteracij. Prvi del uˇcenja je potekal 24 ur in ni vraˇcala nobenih koristnih rezultatov, saj sta se izurjena delavca samo sprehajala po ˇsahovnici, brez da bi nabirala zlatnike. Po nadaljevanju uˇcenja obstojeˇcega modela, ki je potekal prav tako toˇcno 24 ur smo pridobili boljˇse rezutate, ki pa ˇse vedno niso dobri. Izurjena delavca sta tako kot prej skoraj nakljuˇcno hodila po ˇsahovnici, s tem da je kdo izmed njiju izvedel akcijo naberi zlatnike. Zlatnikov potem ni vrnil do skoraj konca igre s ˇcasovno omejitvijo 200. Po vrnitvi zlatnikov je igralec takoj za tem



Slika 7.4: Vizualizacija nauˇcenega modela v Pygame z polji zlata na robovih in glavnimi hiˇsami v sredini. Delavci morajo hoditi daljˇso razdaljo do polj zlata da iz njih naberejo zlatnike, ki jih za tem morajo vrniti v glavno hiˇso sredi ˇsahovnice.

izdelal dodatno enoto, kar je prineslo dovolj prednosti za zmago. Zlatnike vrne v glavno hiˇso proti koncu iteracije igre. Mogoˇce algoritem ˇcaka na konec igre, da preseneti nasprotnika, vendar bolj verjetno je pa da proti koncu igre MCTS zaˇcne bolj delovati, saj vraˇca prave vrednosti stanja igre, ki pa so konˇcna stanja. V tem primeru se je iz delovanja jasno razbralo, da MCTS ne vraˇca primernih rezultatov oziroma ne izboljˇsa uteˇzi modela dovolj dobro. Dober primer je ta, da delavec hodi okrog glavne hiˇse z nabranimi zlatniki, vendar jih ne vrne v glavno hiˇso, kar bi poveˇcalo njegov seˇstevek toˇck in se s tem postavil v prednost pred nasprotnikom.

S tem inkrementalnim uˇcenjem modela v veˇcih korakih smo prikazali, koliko poˇcasi uˇcenje te igre poteka in hkrati dokazali da se model izboljˇsuje. Poˇcasnost uˇcenja je predvsem zaradi velikega ˇstevila akcij, ki se lahko na ˇsahovnici pripetijo na vsakem polju. Vseh moˇznih akcij je 30, kar privede v 8 x 8 x 30 = 1920 ˇstevil, ki jih prejme nevronska mreˇza kot vhodni nivo. Prav tako je zaradi ˇstevila akcij poˇcasno preverjanje katere izmed njih so veljavne, kar upoˇcasni iteriranje stanj iger.

Tako kot smo opisali v sekciji poteku uˇcenja 3.2, smo v tem primeru naleteli na teˇzavo prekomernega prileganja, saj algoritem ni pravilno prepoznaval neodloˇcene izide. Neodloˇceni izidi niso bili kaznovani s strani primerjanja dveh modelov, pri katerih so se primerjale samo zmage in porazi novejˇsega modela proti starejˇsem. Izidov pa je bilo skozi uˇcenje vedno veˇc neodloˇcenih, pri katerem je pri 40. iteraciji uˇcenja ostali samo ˇse neodloˇceni izidi, z redko zmago katerega izmed modelov in rezulta tega je bilo nakljuˇcna hoja delavcev, s ˇcimer so dosegli nov neodloˇcen izid. Popravek izbire modela je prinaˇsal boljˇse rezultate, vendar je izbira novejˇsega modela veliko teˇzja, saj dobra algoritma velikokrat doseˇzeta neodloˇcen izid, ˇse posebej v zaˇcetnih stopnjah igre, kjer imata oba igralca malo zlatnikov in figur.

## 7.4 Zmanjˇsevanje velikosti ˇsahovnice

V tej konfiguraciji uˇcenja pa smo zmanjˇsali velikost ˇsahovnice iz 8 x 8 na 6 x 6. Prav tako smo zmanjˇsali ˇstevilo iteracij iz 30 na 20, saj bi teoretiˇcno bilo potrebnih manj iteracij za uˇcenje manjˇse ˇsahovnice.

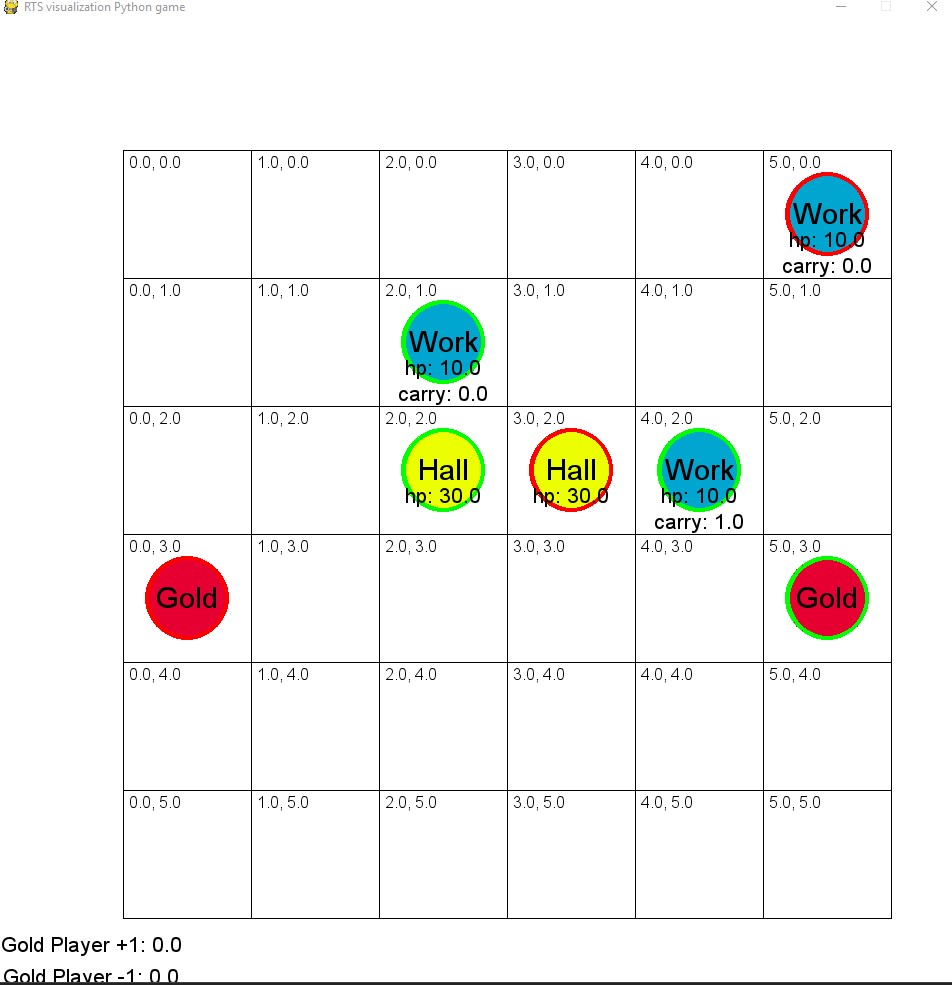
Zmanjˇsevanje ˇsahovnice je privedlo do priˇcakovanih rezultatov. Igralca sta obˇcasno nabirala zlatnike, saj je polje zlatnikov v bliˇzini glavne hiˇse, tako da se delavec postavi med glavno hiˇso in zlatnike in jih nabira brez da bi se moral premakniti. Zaradi nabranih zlatnikov potem igralca izdelata nove delavce.

Stevilo nabranih zlatnikov in izdelanih delavcev je ˇse vedno majhno.ˇ

### 7.4.1 Ustavitvena funkcija zmanjˇsanja ˇzivljenjskih toˇck

V tem primeru smo pa uporabili ustavitveno funkcijo, ki zmanjˇsuje ˇstevilo ˇzivljenjskih toˇck figuram, opisano v sekciji 4.4.1. Po doloˇcenem ˇstevilu potez (v naˇsem primeru 90) funkcija priˇcne izmeniˇcno zmanjˇsevati ˇzivljenjske toˇcke igralˇcevih enot.

Uporaba funkcije v primerjavi z ustavitvenim ˇcasom je proizvedla slabˇse rezultate. Igralca nabereta manj zlatnikov in poslediˇcno izdelata manj figur.



Slika 7.5: Zgornja slika prikazuje stanje igre pri 70. potezi, ki je nauˇcena z uporabo ustavitvene funkcije zmanjˇsanja ˇzivljenjskih toˇck. Igralca sta nabirala zlatnike, in izurila nova delavca, vendar je postopek potekal slabˇse kakor brez te ustavitvene funkcije.

Ta naˇcin uˇcenja modela ni primeren, saj ni popolnoma kompatibilen z obiˇcajnimi RTS igrami, saj se igralˇcevim enotam ˇzivljenjske toˇcke ne odˇstevajo zaradi neznanega razloga. Moˇzno bi bilo ta naˇcin uˇcenja aplicirati v igre, kjer nekatere izmed figur lahko prejmejo zniˇzanje ˇzivljenjskih toˇck zaradi naravnih dejavnikov, kot npr. mraz, strupen plin ipd.

## 7.5 Vizualizacija rezultatov v pogonu Unreal Engine

Proti koncu smo rezultateˇse vizualizirali v pogonu Unreal Engine 4. Izbiranje potez deluje izmenjujoˇce z zamikom pol sekunde, da se akcije konˇcajo, preden se poˇslje nov zahtevek z novim kodiranjem stanja igre.



Slika 7.6: Zgornja slika predstavlja igranje igre dveh raˇcunalniˇskih nasprotnikov enega proti drugemu. V sredini vidimo glavni hiˇsi in polja zlata, okrog pa vojaˇsnice in delavce.

Igranje proti raˇcunalniˇskem nasprotniku je z uporabo tega algoritma moˇzno, vendar je nasprotnik prelahek, da bi bil primeren za njegovo aplikacijo v modernejˇse strateˇske igre.

**Poglavje 8**

# Diskusija

Kot smo v poglavju 7 ugotovili, uˇcenje modela poteka zelo poˇcasi, vendar se model uspeˇsno uˇci. Zasnova opisa igre in njenega enkodiranja je prava, ker uˇcenje poteka uspeˇsno. Poteka pa poˇcasi zaradi ustavitvenega pogoja in ocenitvenih funkcij, ki ugotovi zmagovalca ob tem ustavitvenem pogoju, ki mogoˇce ni pravi zmagovalec ob koncu igre. Prav tako MCTS naredi zelo majhno ˇstevilo iskanj in ne simulira igre do konca oziroma doloˇcene globine, kjer bi ugotovil stanje igre ter to stanje vrnil nazaj, ampak stanje igre pridobi od nauˇcenega modela, ki velikokrat ni pravo.

V sklopu te diplomske naloge se prav tako nismo podajali v spreminjanje zgradbe nevronske mreˇze, temveˇc smo vzeli ˇze izgrajeno nevronsko mreˇzo, primerno za uˇcenje igre Othello. Mogoˇce bi ravno sprememba strukture nevronske mreˇze privedla do boljˇsih rezultatov, saj je matrika stanja igre veliko globja, kakor pri igri Othello.

Pri opisu igre je veliko problemov povzroˇcalo uravnoveˇsanje parametrov igre, kot na primer ˇstevilo vrnjenih zlatnikov, koliˇcina in cena zdravljenja ipd. Dober primer neuravnoveˇsene konfiguracije igre sta bila prav koliˇcina in cena zdravljenja, kjer je stalno samo nabiral zlatnike in zdravil svoje figure, tako da je dosegal vedno daljˇse ˇstevilo narejenih potez. Zaradi tega je uˇcenje potekalo zelo poˇcasi, saj so posamezne igre trajale predolgo da so se zakljuˇcile.

Najveˇc problemov pa je pri opisu igre povzroˇcal ustavitveni pogoj. Ocenitev

57

stanja igre ni najboljˇsa, saj je pridobljena po preprosti formuli, ki pa seveda ne vkljuˇcuje vse dejavnike igre.

Algoritem se da dobro aplicirati v pogon Unreal Engine z nekaj vtiˇcniki, opisanimi v razdelku 6.2. Veˇc igralcev lahko na istem raˇcunalniku zahteva priporoˇcila akcij. Ce pa igra poteka preko mreˇze, pa vsak igralec na svojemˇ raˇcunalniku poganja algoritem, ki ne ovira delovanje drugih TensorFlow sej na raˇcunalnikih drugih igralcev. Sama igra, napisana v Pythonu, na kateri je bil nauˇcen model se pa ne preslika direktno v dejansko igro napisano v Unreal Engine. v tej igri se akcije ne zgodijo instantno, saj vojaˇske enote in delavci potrebujejo nekaj ˇcasa da se premaknejo na drugo lokacijo, izvedejo akcijo kot na primer nabiranje zlatnikov, ki tudi ni instantna. Zaradi trajanja akcij, asinhronosti pridobivanja priporoˇcila od Python modula se lahko zgodi, da stanje igre ni veˇc takˇsno, kot smo ga poslali v Python skripto, in bi bila priporoˇcena akcija z asinhrono skripto drugaˇcna. V nekaterih primerih se ob takih pogojih dve figuri premakneta na isto polje, oziroma izgradi hiˇsa na polju, kjer je trenutno enota. Reˇsitev za to bi bila vpeljava zahtevanja priporoˇcil akcij ko so akcije zakljuˇcene, ter nezmoˇznost izvajanja akcij v ˇcasu od zahtevka priporoˇcila do vrnjenega rezultata. Ta reˇsitev pa ni primerna, saj bi bila primerna za strateˇske igre, vendar ne za podkategorijo realno-ˇcasovnih strateˇskih iger.

Nauˇcen model prav tako vraˇca samo 1 akcijo, ki pa ne more vkljuˇcevati veˇcjo skupino figur, kot na primer vseh vojaˇskih enot, da se premaknejo proti nasprotnikoviku za napad. Veˇcino teh akcij pa so tudi omejene na sosednja polja, kot na primer pomik gor, dol, napad gor ipd., kar tudi ni primerna aplikacija v dejansko igro, kjer se lahko figure premikajo v poljubnih dolˇzinah in smereh.

**Poglavje 9**

# Zakljuˇcek

V diplomski nalogi smo povzeli kaj realno-ˇcasovna strateˇska igra je, da smo jo lahko uspeˇsno tudi implementirali. Pregledali smo njihove nivoje nadzorovanja in abstrakcije in s kakˇsnega zornega kota na njih gledajo nevronske mreˇze. Za tem smo se podali v raziskovanje algoritmov za uˇcenje te strateˇske igre in smo naleteli na algoritem Alpha Zero. Na hitro smo pregledali njegovo zgodovino in korake k samostojnem algoritmu, ki je primeren za reˇsevanje poljubne igre z metodami samouˇcenja. Ta algoritem smo potem ˇse podrobneje pregledali, da smo njegov proces uˇcenja in igranja iger razumeli, da smo lahko za tem opisali svojo strateˇsko igro v Pythonu. Doloˇcili smo pravila igre in glavne cilje, ki jih strateˇska igra mora imeti. Pri tem smo morali paziti da smo se drˇzali okvira Surag-Nairjeve implementacije algoritma Alpha Zero, da smo lahko zanj pripravili svojo strateˇsko igro, ki je kompatibilna z njegovim algoritmom. Za tem smo doloˇcili akcije ki jih figure lahko izvajajo in jih abstrahirali, tako da so za algoritem nedvoumne in hitre. Da pa smo nevronski mreˇzi lahko podali stanje igre, smo ga morali pravilno zakodirati. Izbrali smo desetiˇski in one-hot naˇcin kodiranja, med katerima se je one-hot izkazal uspeˇsnejˇsi, saj ne prioritizira veˇcjih zakodiranih ˇstevil kot boljˇsih. Za tem smo ugotovili pravˇsnji naˇcin evalvacije stanja igre in njen ustavitveni pogoj. Doloˇcili smo ustavitveno ranjujoˇco funkcijo, ki dovoljuje bolj aktivnim igralcem daljˇse igranje, vendar jih kaznuje iz razlogov**, ki** sami strateˇski igri

59

niso naravni. Drugi pristop ustavitvenega pogoja je bil ˇcasovni iztek, pri katerem se je po doloˇcenem ˇstevilu potez presodilo, kateri igralec je zmagovalen po doloˇcenem kriteriju. Ko smo imeli opisano igro, smo morali ˇse ugotoviti primerne nastavitve uporabiti pri samem uˇcenju igre in izbrati parametre. Za tem smo pripravili vizualizacijo igre v Pythonu z modulom Pygame, ki prikaˇze igro v preprosti ˇsahovnici in figure s krogci. V pogonu Unreal Engine 4 pa smo pripravili bolj kompleksno strateˇsko igro, ki je boljˇsa predstavitev dejanske realno-ˇcasovne strateˇske igre. V tej igri poˇsiljamo zahtevke za akcije preko vtiˇcnika v python skripto, v katero podamo trenutno stanje igre, nazaj pa dobimo priporoˇceno akcijo. Zahtevke lahko poˇsilja raˇcunalniˇski nasprotnik ali ˇcloveˇski igralec, ko v najboljˇso akcijo ni prepriˇcan. Za tem smo se posvetili predvsem uˇcenju modelov z razliˇcnimi konfiguracijami ter jih vizualizirali v Pygame in Unreal Engine. Ugotovili smo, da je uˇcenje poˇcasnejˇse zaradi kompleksnosti igre, vendar da uspeˇsno poteka. Diplomska naloga je dober prispeve k Surag-Nairjevim igram za Alpha Zero, ki razˇsiri preproste igre ˇcrno-belih figur v figure veˇcih atributov. V to igro smo pripeljali tudi ˇcasovne kompleksnosti in jo razˇsirili z vizualizacijo v Pygame in Unreal Engine 4.

Igra izdelana v Ue4 in Alpha Zero General algoritem s Pygame igro sta na voljo na naslednjih repozitorijih:

* igra izdelana v Unreal Engine: https://github.com/JernejHabjan/TrumpDefense2020,
* Alpha Zero General algoritem z izdelano igro v Pygame: https://github.com/JernejHabjan/alpha-zero-general Diplomska naloga *61*

*62* Jernej Habjan

# Literatura

1. How many atoms are there in the universe? [https:// www.universetoday.com/36302/atoms-in-the-universe/.](https://www.universetoday.com/36302/atoms-in-the-universe/) Accessed:

2018-12-03.

1. Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008. [3] Matej Guid. Sah in znanost.ˇ *Sahovska miselˇ* , (1):58, 2018.
2. Marko Kladnik. Primerjava igralnih pogonov unity in unreal engine. Diplomska naloga, Fakulteta za elektrotehniko in raˇcunalniˇstvo, Univerza v Ljubljani, 2015.
3. Andraˇz Kohne. Monte-carlo tree search in chess endgames. Dodiplomska disertacija, Fakulteta za raˇcunalniˇstvo in informatiko, Univerza v Ljubljani, 2012.
4. Santiago Ontan´on. Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research*, 58:665–702, 2017.
5. Santiago Ontan˜on, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, and David Churchill. A survey of real-time strategy game ai research and competition in starcraft. IEEE Transactions on Computational Intelligence and AI in games, IEEE Computational Intelligence Society, 2013.

63

*64* Jernej Habjan

1. David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.
2. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by selfplay with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.
3. Alberto Uriarte and Santiago Ontan˜o´n. Game-tree search over high-level game states in rts games. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.
4. Alberto Uriarte and Santiago Ontan˜on. Automatic learning of combat models for rts games. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
5. Wikipedia. AlphaGo — Wikipedia, the free encyclopedia. [http:// en.wikipedia.org/w/index.php?title=AlphaGo&oldid=863856157,](http://en.wikipedia.org/w/index.php?title=AlphaGo&oldid=863856157) 2018. [Online; accessed 20-October-2018].
6. Wikipedia. Shannon number — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=Shannon% 20number&oldid=869989473,](http://en.wikipedia.org/w/index.php?title=Shannon%20number&oldid=869989473) 2018. [Online; accessed 03-December-

2018].