Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jernej Habjan

Učenje realno-časovne strateške igre z uporabo globokega spodbujevalnega učenja

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matej Guid
Somentor: prof. dr. Branko Šter

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode uporabiti, morda bo zapisal tudi ključno literaturo.

Zahvaljujem se mentorju doc. dr. Mateju Guidu in somentorju prof. dr. Branku Šteru, prijateljem in družini, ki so mi pomagali pri pisanju diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
MCTS	Monte Carlo tree search	Monte-Carlo drevesno preiskovanje
UE4	game engine Unreal Engine 4	celostni pogon Unreal Engine 4
RTS	real-time strategy	realno-časovna strateška

Povzetek

Naslov: Učenje realno-časovne strateške igre z uporabo globokega spodbujevalnega učenja

Avtor: Jernej Habjan

Z obstoječim Alpha Zero algoritmom smo implementirali učenje in priporočanje akcij v realno-časovni strateški igri. Pregledali smo krajšo zgodovino globokega spodbujevalnega učenja na igrah in povzeli zakaj je pristop samostojnega učenja najprimernejši. Za strateško igro smo definirali figure in njihove akcije in zakodirali kompleksno stanje igre s kodirnikom. Prav tako smo definirali ustavitvene pogoje pri igri, ki nima končnega števila potez na podlagi poškodovanja figur. Rezultate smo prikazali s Python modulom Pygame in v celostnem pogonu Unreal Engine 4. V obeh vizualizacijah lahko igramo proti naučenem modelu, ali pa opazujemo, kako se dva računalniška nasprotnika bojujeta med sabo. Na koncu smo še pregledali rezultate in povzeli učinek učenja algoritma.

Ključne besede: Alpha Zero, realno-časovna strateška igra, Unreal Engine.

Abstract

Title: Teaching of real-time strategy game using deep reinforcement learning

Author: Jernej Habjan

With the existing Alpha Zero algorithm, we implemented learning and recommending actions in a real-time strategy game. We examined the shorter history of deep stimulating learning in games and summarized why the self-learning approach is most appropriate. For a strategic game, we defined the figures and their actions and encoded the complex state of the game with the encoder. We also defined the stopping conditions of the game, which has no final number of moves based on damage to the figures. The results were displayed with the Python Pygame module and the Unreal Engine 4 integrated drive. In both visualizations we can play against the learned model, or we can observe how two computer opponents are fighting each other. In the end, we have 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-časovnih oziroma RTS igrah je problem, s katerim se mora soočiti večina razvijalcev teh iger, agentove akcije so pa pogosto predvidljive, saj se človeški igralec nauči njihovih načinov delovanja in jih tako lažje premaga. Če pustimo agentu, da sam opravlja akcije nekontrolirano, bo izvajal naključne akcije, ki so pa slabše kot vnaprej definirana taktika. Če pa agentu podamo hevristiko, po kateri se mora ravnati, bo poskušal izvesti čim boljšo akcijo, vendar bo za njen izračun porabil predolgo časa, saj bo moral preiskati cel preiskovalni prostor, ki pa pri realno-časovnih strateških igrah zna biti prevelik. Na primer 10 enot v igri, kjer ima vsaka 5 možnih potez, se razveji na možen faktor $5^{10} \approx 10$ milijonov možnih akcij. Za igro StarCraft je ocenjenih možnih vsaj 10^{1685} možnih akcij, kjer je za šah 10^{47} in 10^{171} za igro Go [3].

Preiskovanje prostora z grobo silo torej odpade. Ostanejo nam potem hevristični algoritmi, kot so Alpha-Beta rezanje ali Monte-Carlo drevesno preiskovanje oziroma MCTS. Ampak Alpha-Beta deluje dobro samo pod pogoji, da obstaja zanesljiva evaluacijska funkcija in da ima igra majhen vejitveni prostor, kar je pa lastnost veliko klasičnih namiznih iger kot Go in video iger. Zato se je bolje v takih primerih odločiti za MCTS [1]. MCTS pa ima pomanjkljivost, da si stanj igre ne zapomni skozi več iger, kjer bi lahko to vrednost stanja uporabil za bolj natančen izračun naslednjih stanj.

Za memorizacijo stanj pa pridejo v upoštev globoke nevronske mreže, ki pa z učenjem ugotovijo zakonitosti v učni množici in skozi mnogo iteracij izboljšajo svojo predikcijo določenega izhoda ob določenem vhodu. To je pa točno to, kar potrebuje MCTS kot začetno stanje, iz katerega lažje izračuna najboljšo akcijo.

Da pa nevronska mreža dobi dovolj vhodnih podatkov za učenje, pa moramo realizirati algoritem, ki bo igral proti drugem računalniškem nasprotniku, in pridobil rezultat, ali je to igro zmagal, ali zgubil. Ob tem izhodu nevronska mreža nagradi svoje predikcije ob določenem stanju, ali pa jih kaznuje.

To je glavna ideja o implementaciji algoritma, ki jo pa vsebuje algoritem Alpha Zero, ki smo ga uporabili v tej diplomski nalogi. Algoritem se nauči igranja igre z igranjem iger sam proti sebi, kjer boljša različica algoritma napreduje v naslednji krog. Ko je model nevronske mreže naučen, ga lahko uporabimo, da nam priporoči akcijo v določenem stanju. Tako lahko implementiramo računalniškega igralca, ki pridobiva akcije od naučenega modela in jih izvršuje, kot tudi priporočilni sistem za akcije, ki jih prikazujemo človeškemu igralcu. Algoritem nam priporoči akcijo in ne tipa strategije, katerega naj izberemo, kar bi potrebovalo še bolj abstrakten pogled na igro.

O strateških igrah, njihovih abstrakcijah in zakaj so tako zanimive za raziskovanje umetne inteligence bomo več spoznali v poglavju 2. V poglavju 3 bomo podrobneje pregledali sestavo Alpha Zero algoritma in zakaj je primeren za našo RTS igro. Ko bomo imeli sestavljen algoritem, bomo zanj sestavili RTS igro v poglavju 4 in izpostavili, kaj so glavne težave pri takih igrah. Sestavljen algoritem bomo naučili na igri v poglavju 5, kjer bomo pregledali razne parametre pri učenju in naučen model potem preizkusili z vizualizacijo v Python modulu Pygame in celostnem pogonu Unreal Engine v poglavju 6. Rezulate učenja bomo potem še ocenili in ugotovili, katera vrsta učnih parametrov nam je podala najboljši rezultat v poglavju 7 in zaključili ugotovitve v poglavju 8.

Poglavje 2

Realno-časovne strateške igre

Realno-časovne strateške igre oziroma RTS igre so žanr strateških iger, kjer igralec nadzoruje množico figur, in poskuša premagati nasprotnika z izgradnjo ekonomije, izboljšavo tehnologije in urjenjem primernih vojaških enot, ki dodajo dodano vrednost k končni zmagi igre. Primer RTS igre je na primer Age of Empires II ali igra StarCraft.

Izzivi realno-časovnih iger so naslednji:

- Upravljanje z viri
- Izbira akcij ob nevednosti
- Prostorsko in časovno razmišljanje
- Sodelovanje med večimi agenti
- Modeliranje nasprotnika in učenje
- Nesporno načrtovanje v realnem času

Zdajšni izzivi:

- Planiranje: Planiranje v realno-časovni igri je vidno kot več nivojev abstrahiranega stanja igre. Višji kot je nivo, bolj dolgoročni so cilji, kot naprimer gradnja ekonomije, na nižjem nivoju je pa premik posamezne enote ipd.

- Učenje: Predhodno učenje, ki uporablja posnetke že odigranih iger, Učenje v igri, ki uporablja po večini spodbujevalno učenje in modeliranje nasprotnika. Učenje med igrami

- Negotovost: Negotovost nastane zaradi nevidnosti nasprotnika in njegovih potez v vsakem trenutku. Prav tako pa ne vemo akcij, ki jih bo nasprotnik izvedel, zato zgradimo drevo, ki nam pove kaj je najverjetneje da bo nasprotnik naredil.

- Prostorsko in časovno razumevanje: Prostorsko razumevanje je usmerjeno k postavljanju stavb in pozicijo vojske za obrambo in napad. Časovno razumevanje je pa usmerjeno k ugotavljanju, kdaj je primerna izdelava hiš za ekonomijo in kdaj pa za napad.

- Izkoriščanje znanja domen: Izkoriščanje znanje botov. StarCraft je kompleksen, in to ostaja še odprt problem

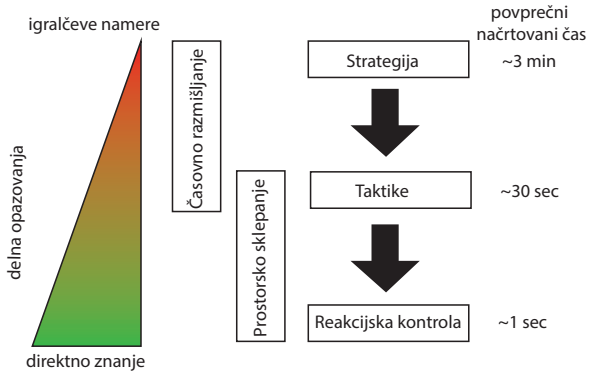
- Razdelitev nalog **??**: Strategija, ki je najvišja abstrakcija (3 min planiranje) Taktika, ki je implementacija trenutne strategije (pozicija vojske, hiš - 30 sec planiranje) Reakcijska kontrola, ki je implementacija taktike, ki je osredotočena na posamezno enoto Analiza terena, ki se osredotoča na strnjena območja in na višinsko prednost Pridobivanje znanja, s katerim pridobivamo informacije o taktiki nasprotnika [4].

  Pogosto razdelimo odločanje na dva dela:

  - Micro, kjer kontroliramo enote posamezno
  - Macro, kjer se osredotočimo na ekonomijo in izdelavo enot

2.1 Strategija

V strateških igrah je velikokrat uporabljen pristop direktnega kodiranja strategije, ki uporabljajo avtomate končnih stanj, kjer lahko razbijemo delovanje na več stanj kot so napadanje, nabiranje surovin, popravilo itd. in hitro



Slika 2.1: Razdelitev nalog glede na čas reakcije in abstrakcije nalog.

menjavanje med njimi. Direktno kodiranje prinese dobre pričakovane rezultate, vendar se lahko igralec nauči strategije in ga tako agenta hitro porazi. Planirani pristopi ponujajo večjo prilagodljivost kot direktno kodirani.

2.2 Taktika

Taktika spada pod direktnejši nadzor enot kakor strategija in je bolj osredotočena na kontrolo določenih točk na mapi, zmagi posameznih bitk in iskanje ožin, kjer je nasprotnik šibkejši. Taktika temelji na analizi terena, ki ga lahko razbijemo na kompozicijo ožin.

2.3 Abstrakcija prostora

Razbiranje strategije in taktike je za algoritme umetne inteligence težji, saj potrebuje višji nivo abstrakcije prostora, figur in akcij, kot za izbiro posameznih nizkonivojskih akcij.

Prav tako problem nastane zaradi negotovosti, kjer ne vidimo nasprotnikovih enot in potez v vsakem trenutku. Predikcija nasprotnikovih potez je tako veliko težja, tako da vsi algoritmi s tako negotovostjo ne delujejo. Mi smo se za diplomsko nalogo odločili, da imata oba računalniška agenta

popoln vpogled na stanje igre in nasprotnikove akcije.

Poglavje 3

Predstavitev algoritma Alpha Zero

3.1 Zgodovina

Igranje iger je popularno področje znotraj vede o umetni inteligenci. Eden izmed prvih programov je bil programski pogon Checkers (Samuel 2000), ki se je naučil igranja z medotami samo-igranja in strojnega učenja in ne z metodami, ki temeljijo na pravilih. Leta 2002 je Deep Blue premagal človeškega profesinalnega igralca šaha z nadčloveško sposobnostjo igranja. Pri teh igrah je faktor vejanja akcij še realativno majhen in je lažje oceniti končno pozicijo iz danega stanja. Rečeno je bilo, da igre kot npr Go, ki imajo toliko večji faktor vejanja 10^{171} v primerjavi s šahom, ki pa ima 10^{47}, ne bo možno ugotoviti vrednost končnega stanja še nekaj desetletij.

Ampak algoritem AlphaGo [5] je naredil preboj,s tem da uporablja metodo globokega spodbujevalnega učenja in algoritem Monte-Carlo drevesno preiskovanje. Oktobra 2016 je premagal profesionalnega Go igralca na podlagi učenjna na domenskem znanju iger, ki so bile odigrane od ekspertov.Te sistemi so temeljili na predznanju ekspertov za učenje in evaluacijo modela.

Leto za tem, je bil razvit algoritem AlphaGo Zero [6], ki opisuje pristop k učenju brez domenskega znanja ekspertov, ampak uporablja metodo samo-

igranja. Novi model je prav tako premagal AlphaGo algoritem, kar predstavlja odlične rezultate z vidika, da AlphaGo Zero ne potrebuje človeško usmerjanje pri učenju.

Računalniki se lahko tako naučijo reševanje problema brez človeških ekspertov, ki delajo napake in nimajo takojšnega vpogleda na celotno učno množico, kot to imajo računalniki.

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 šah, Shogi, Go, kjer algoritem potrebuje samo pravila igre, ta pa se uči z globokimi nevronskimi mrežami in tabula rasa algoritmom za spodbujevalno učenje. Zaradi te generalizacije algoritma, lahko algoritem apliciramo na našo RTS igro, kjer moramo definirati pravila igre. AlphaZero je drugačen od AlphaGo Zero tako, da AlphaZero vrača rezultate, ki so lahko drugačni od zguba, poraz, kot tudi neodločeno. Prav tako se razlika pojavi v tem, da so se igre pri algoritmu AlphaGo Zero zgenerirale iz vsej prejšnih iteracij, in se je potem moč modela izračunala proti najboljšim igralcem, medtem ko AlphaZero samo hrani eno nevronsko mrežo, ki se stalno posodablja, namesto da čaga iteracijo da se konča.

3.2 Potek učenja

AlphaZero se uči veretnosti in ocenitve končnega stanja izključno z igranjem proti samemu sebi. Te potem uporabi pri preiskovanju z glavno namensko metodo Monte-Carlo drevesnim preiskovanjem, da razišče drevo stanj za akcijo. Drevo preišče prostor in vrne verjetnost zmage pri izbiri določene akcije iz trenutnega stanja imenovano Pi in oceno končnega stanja iz trenutnega stanja v, ki zavzema vrednosti -1 ali 1 (Poraz, zmaga). AlphaZero izvede več serij igranja iger proti svojim nasprotnikom, ki predstavlja zdajšni najboljši model igranja. Rezultat igranja igre je lahko -1 za poraz, +1 za zmago in 0 za neodločeno. Po vsaki seriji učenja, se izvede proces igranja Arena, kjer oba naučena modela igrata drug proti drugemu nekaj iger, in se na to določi zma-

govalen model, ki sedaj postane najboljši model, če je razlika v številu zmag večja za nek faktor. V našem primeru je bil ta faktor 60Parametri nevronske mreže so za tem popravljeni, da minimizirajo napako med predikcijo stanja nevronske mreže in dejanskim rezultatom igre in da maksimizirajo podobnost predikcijo potez nevronske mreže z dejanskimi vrednostnimi akcij, ki jih je vrnil MCTS. Oziroma parametri se nastavijo z gradientnim spustom na funkcijo izgube, ki sešteje napako srednega korena (mean-squared error) in prečne entropije (cross entropy).Nevronska mreža sprejme učne množice stanja iger in vrne ravni vektor predikcije akcij v trenutnem stanju in predikcijo zmage.

Poglavje 4

Definiranje pravil igre

Igro smo definirali po Surag Nairjevi predlogi za Alpha Zero, ki je na voljo na (portalu?) Github (Insert reference here). Igra je dodana kot modul, ki vsebuje definicijo igre in njena pravila, igralce, vizualizacijo in izgradnjo modela

Igra je definirana v kvadratni mreži 8x8, kjer polje lahko vsebuje največ eno figuro. Ostale igre, ki so napisane za to različico Alpha Zero izvedbe, kot na primer štiri v vrsto, gobang, othello, tri v vrsto, vsebujejo črno-bele figure. Zakodirane so lahko z eno številko: -1 za igralca -1, +1 za igralca +1 ali 0, če je polje prazno. Pri teh igrah je dimenzija kodiranja 2-dimenzionalna, kjer dimenzije predstavljajo višino in širino igralne plošče. Pri rts igrah pa moramo vedeti poleg igralca, komur ta figura pripada, tudi stanje te figure, na primer trenutno zdravje in tip figure. Zato je prostor kodiranja 3-dimenzionalen, kjer je tretja dimenzija zakodirano stanje figure. Če bi dovolili, da na posamezno polje spada več figur, se dimenzija ponovno poveča za 1.

4.1 Definiranje stanja igre

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

Sprva moramo definirati figure, ki bodo imele določeno vlogo v igri. Nabor figur je majhen, saj nočemo, da preiskovalni prostor postane prehitro prevelik.

- Zlato - Vir surovin, ki predstavljajo denar v igri, s katerim lahko igralec gradi nove stavbe in uri nove enote
- Delavec - Figura namenjena gradnji hiš in nabiranju zlata
- Vojašnica - Stavba namenjena urjenju vojaških enot
- Vojak - Figura namenjena napadanju sovražnikovih enot
- Glavna hiša - Stavba namenjena urjenju delavcev in vračanju surovin zlata.

Realizirali smo atribute figur. Pomembno je, da so te atributi numerični, da lahko podamo stanje igre kot N-dimenzionalen vektor, ki ga nevronska mreža lahko sprejme in se iz teh numeričnih podakov uči. Prav tako je pomembno, da ima vsako polje na šahovnici enako število atributov, tudi če je to polje prazno. Vsako prazno polje ima vanj vpisan atribut čas igranja, ki je splošen za celo igro, vsa ostala polja pa imajo vrednost 0.

- Ime igralca: Določa igralca, h kateremu ta figura pripada. Igralec lahko nadzoruje samo svoje figure, izvaja akcije na svojih enotah in napada sovražnikove enote.
- Tip figure: Atribut predstavlja numerično predstavitev tipa figure kot naprimer zlato, 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 povečuje do nekega maksimuma z akcijo zdravi in znižuje z napadom enote
- Nosi zlato: Poseben atribut za delavce, ki predstavlja vrednost 1, če figura nosi zlato in 0, če ga ne nosi. To se upošteva pri nabiranju in

vračanju zlata, kjer se ti dve akcije ne zgodita v roku ene poteze, ampak se mora stanje prenašati skozi več potez

- Denar: Trenutna količina zbranega denarja za posameznega igralca. To polje se ob spremembi količine denarja spremeni v vseh figurah tega igralca

- Čas igranja: To polje predstavlja koliko potez se je v trenutni igri že izvedlo. Atribut je prisoten v vseh poljih in se spremeni v vseh poljih šahovnice, ko se izvede nova akcija

Poseben primer je figura Zlato, ki ne pripada nobenemu igralcu v večini RTS igrah. V tem primeru pa sem podal vsakemu igralcu svoje polje zlata, da je igra simetrična in nevronska mreža ne interpretira prazno polje igralca kot prazno polje. Prav tako se figuri zlato ne spreminja atribut zdravja, saj jo ne moremo poškodovati.

4.2 Akcije

Prav tako moramo definirati akcije, ki jih te figure lahko izvajajo. Vsaka figura ne more izvajati vseh akcij, kot naprimer stavbe se ne morejo premikati, same enote kot delavec in vojak pa ne morejo uriti novih enot 4.1.

- Nedejaven - Če igralec izbere akcijo nedejavnosti, se ne izvede nobena akcija in se poteza ta igralca zaključi

- Premik gor - Enoti vojak in delavec se lahko premakneta za eno polje proti zgornjemu robu šahovnice če je to mesto prazno

- Premik dol - Enoti vojak in delavec se lahko premakneta za eno polje proti spodnjemu robu šahovnice če je to mesto prazno

- Premik levo - Enoti vojak in delavec se lahko premakneta za eno polje proti levem robu šahovnice če je to mesto prazno

- Premik desno - Enoti vojak in delavec se lahko premakneta za eno polje proti desnem robu šahovnice če je to mesto prazno
- Naberi zlato - Delavec lahko nabere zlato če je v neposredni bližini enote zlato in jih za trenuten čas drži pri sebi
- Vrni zlato - Delavec vrne zlato, ki jih drži pri sebi v glavno hišo, na kar se igralcu prišteje denar
- Napadi - Vojak lahko napade sovražno enoto, če je ta v neposredni bližini in jo rani za določen faktor
- Izuri delavca - Glavna hiša lahko izuri novo enoto delavec, če ima dovolj denarja, na kar se igralcu odšteje denar
- Izuri vojaka - Vojašnica lahko izuri novo enoto vojak, če ima dovolj denarja, na kar se igralcu odšteje denar
- Izgradi vojašnico - Delavec lahko izgradi vojašnico na prazno mesto zraven njega, na kar se igralcu odšteje denar
- Izgradi glavno hišo - Delavec lahko izgradi glavno hišo na prazno mesto zraven njega, na kar se igralcu odšteje denar

Naslednje akcije se izvedejo po nekem zaporedju:

- naberi zlato
- vrni zlato
- napadi
- delavec
- vojak,
- vojašnica
- glavna hiša

```
coords = [(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 n_x, n_y in coords:
    # check action condition or execute action
```

Ko je doseženo prvo prazno polje v tem zaporedju, se tam izgradi nova hiša ali izuri nova enota. Ko je prva sovražna enota izbrana v tem zaporedju, je napadena. Rezultat tega je gradnja hiš in urjenja enot v spodnji levi kot šahovnice, saj so izbrana prva polja v zaporedju, kot naprimer x-1, y+1, in širjenje proti zgornjim desnim kotom, ko so vsa ostala polja zasedena, oziroma tam ni sovražnih enot.

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

4.3 Kodiranja

Definirali smo še začetno stanje vsake igre, kjer sta igralca postavljena v sredino mreže z njihovima glavnima hišama, zraven njiju pa ima vsak igralec svoje polje zlata. Vsakemu igralcu se doda na začetku določena količina denarja za izgradnjo začetnih delavcev. V našem primeru je bilo to 1, tako da je lahko izgradil samo enega delavca.

Sedaj pa potrebujemo zakodirati to stanje igre, v numerični prikaz, ki ga bo nevronska mreža lahko interpretirala. To stanje lahko zakodiramo z desetiškim kodiranjem, vendar obstja možnost, da nevronska mreža sloni proti boljšim obravnavanjem pozitivnih števil za igralca +1, kot za igralca

Ime figure	Akcije	Zdravje	Strošek izdelave
Zlato	/	10	0
Delavec	gor, dol levo desno, vojašnica, glavna hiša, naberi zlato, vrni zlato, zdravi	10	1
Vojašnica	vojak, zdravi	10	4
Vojak	gor, dol, levo, desno, napad, zdravi	20	2
Glavna hiša	delavec, zdravi	30	7

Tabela 4.1: Tu so še opisane figure z njihovimi akcijami in nastavljenimi atributi

-1. Ravno iz tega razloga obstaja kodiranje One hot, ki spremeni desetiška števila v binarni vektor.

4.3.1 Desetiško kodiranje

Pri desetiškim kodiranjem, predstavimo vsak atribut figure z eno desetiško številko. Ker imamo figure s 6 atributi, lahko stanje zakodirane igre predstavimo z dimenzijami širina x višina x 6.

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

4.3.2 One Hot kodiranje

- Ime igralca: 2 bita zaradi treh različnih možnosti: 00 predstavlja prazno polje, 01 predstavlja igralca 1 in 10 igralca -1,
- Tip figure: 3 biti, saj imamo 5 različnih figur,
- Trenutno zdravje figure: 5 bitov saj hočemo predstaviti večjo številko, zaradi odštevanja zdravja z ranjujočo funkcijo 4.1,

- Nosi zlato: 1 bit, kjer vrednost lahko zajema vrednost nosi - 1 ali ne nosi - 0,
- Denar: 5 bitov, kjer pustimo da igralec gradi ekonomijo in shranjuje denar, da ga potem lahko na hitro zapravi na enotah, ko ga ima dovolj za njihovo izgradnjo.
- Čas igranja: $2^{11} = 2048$, kar pusti igralcu dovolj časa da odkriva nove poteze, ampak ga dovolj hitro omeji, da se konča igra in začne nova

Dimenzija zakodiranega prostora je tako 8x8x22

4.4 Konec igre

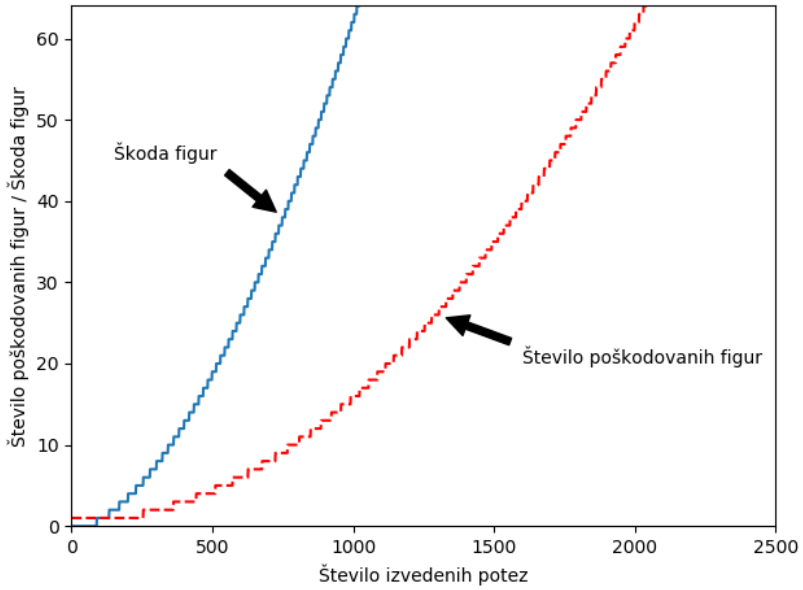
Konec igre se izvede pod določenimi pogoji:

- Igralec nima za izvesti več nobene možne akcije
- Vse figure igralca so uničene
- Ko se čas izteče

4.4.1 Reševanje problema neskončnega števila potez

Čakanje, da se čas igre izteče, je problematično, saj učenje modela poteka zelo počasi, še posebej ko MCTS raziskuje prostor. Za to smo razvili funkcijo, ki prisili model k izvajanju akcij v zgodnem času igre, drugače figure začnejo pridobivati preveč škode in so zato emilinirane s šahovnice.

Vidimo, da je krivulja škode veliko bolj stroga in se v igri začenja že zelo zgodaj. To je zato, ker želimo hitro odpraviti nedejavnih igralcev in prioritiziramo tiste, ki zbirajo minerale in pridobivajo nove fugzre. Vidimo tudi y osi od 0 do 64, kar je največje število igralcev za enega igralca, tako da bo pri približno 2000 korakih vsaka figura dobil smrtno poškodbo, zato časovni potek nikoli ni dosežen.



Slika 4.1: Na grafu sta narisani dve funkciji, ki določata koliko škode se obravnava v določeni enoti v danem času igre(modra) in koliko igralcev je bilo poškodovanih v trenutnem časovnem okviru(rdeča).

Figure lahko tudi uporabijo akcijo zdravljenja, s čimer povečajo trenutno zdravje določene figure do največ njenega maksimuma.

((((Opiši following problem: problem je biv balancat krivuljo ter heal ko sem nastavu heal cost na 1 in heal na 10, se je player nauču samo healat in pršu do runde 1000, potem sm ga pa ustavu Nastavu sm še return mineral amount to 5 al neki, da ma vč možnosti za delat stvari, ker drgač je sam healov ai))))

Poglavje 5

Učenje modela

Učenje te igre je zapleteno zaradi pogojev konca igre. Algoritem pričakuje, da se igra konča z uporabo simulacij MCTS, vendar se pa lahko igra zacikla če igralec večkrat ponovil isto potezo, na kar Python javi napako zaradi prevelike globine rekurzije. To se lahko reši z uporabo časovnih omejitev, kjer se simulacija ustavi ko se izteče čas, vendar lahko povzroči netočno MCTS drevo, ker vozlišča niso pravilno ovrednotena med povratnim propagiranjem. Potrebno je najti ustrezno končno stanje ali spremeniti vir, da izključite časovne omejitve, ker ne vrnejo najboljših rezultatov.

Prav tako se nam je porodila učna ideja o postopnim učenjem modela. Najprej bi začeli učiti model na preprostem končnem pogojem kot naprimer številu izdelanih delavcev. Ko bi model uspešno ustvarjal delavce, bi pogoj spremenili o naprimer nabiranju zlata, tako da bi model že vedel o gradnji delavcev, kar bi nadgradil še z nabiranjem zlata. Težava se pojavi zaradi kodiranja stanja, ki ni istih dimenzij kot prejšno stanje, kjer smo imeli drugi ustavitveni pogoj z drugačnim številom akcij pri figurah. Možna učna ideja Ideja je, da se s postopnim ucnim modelom spremeni stanje konca igre. Najprej začnite učiti model na preprost način končne igre, kot so izdelava delavcev in ko model uspešno ustvarja delavce, dodajte še en pogoj poleg tega že izvedenega modela. Možna težava se lahko pojavi zaradi velikosti modela

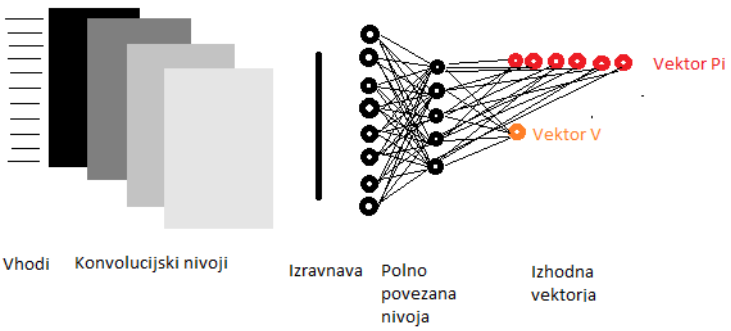
5.1 Zgradba modela nevronske mreže

Uporabili smo Keras modul znotraj TensorFlow knjižnice, za implementacijo modela nevronske mreže.

Model za vhod vzame učne množice stanja iger dimenzije širina, višina, število kodirnikov. Potem gre ta učna množica skozi 4 konvolucijske nivoje, kjer je aktivator relu, tako da je izhod zadnjega konvolucijskega nivoja dimenzije velikost serije x (širina-4) x (višina-4) x število kanalov

Za tem se izhod konvolucije izravna v 1-dimenzionalni vektor in se poda dvema polno povezanima nivojema z aktivacijo relu in in Dropout funkcijo, ki prepreči prekomerno prileganje. Vsi zgoraj navedeni nivoji so normalizirani z BatchNormalization.

Za tem je izgrajen polno povezan nivo Pi, ki ima toliko število izhodov, koliko je možno število 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 kategorična prečna entropija, za izhod V pa srednja napaka korena (mean-squared error),



Slika 5.1: Predstavitev izgrajenega modela

Izrek 5.1 *formula po kateri računa verjetnost zmage pri določeni akciji v algoritmu MCTS*

$$R(s,a) = Q(s,a) + cpuctP(s,a)\sqrt{\frac{\sum b * N(s,b)}{N(s,a)}} \tag{5.1}$$

5.2 Izbira parametrov

Cpuct je parameter drevesnega raziskovanja. V našem primeru je bil nastavljen na vrednost 1.

Pri učenju smo uporabili

Opišeš parameter mcts sims in cpuct, arenacompare, numiters, numeps,

Poglavje 6

Vizualizacije

6.1 Pygame

To je preprosta vizualizacija s Python knjižnico Pygame, za pregled igre med samim razvijanjem. Šahovnica je označena s črtami, med katerimi so s krogi izrisane figure, kjer njihove barve predstavljajo svoj tip enote in obroba krogca igralca -1 ali +1. V krogcih je tudi napisano zdravje za to enoto in zastavica, ali delavec prenaša zlato. Zgoraj je izpisano, koliko denarja ima posamezen igralec in koliko potez sta igralca že odigrala. Prav tako so izpisane vse možne akcije, ki jih igralec lahko izvrši z določeno figuro.

Igralec lahko nadzoruje svoje figure s tipkovnico in miško. Uporabnik mora najprej izbrati figuro z levim miškinim klikom in potem izbrati določeno akcijo, ki je izpisana na zaslonu. Uporabnik lahko spremeni figuro, tako da jo odznači s klikom desnega miškinega gumba na prazno mesto.

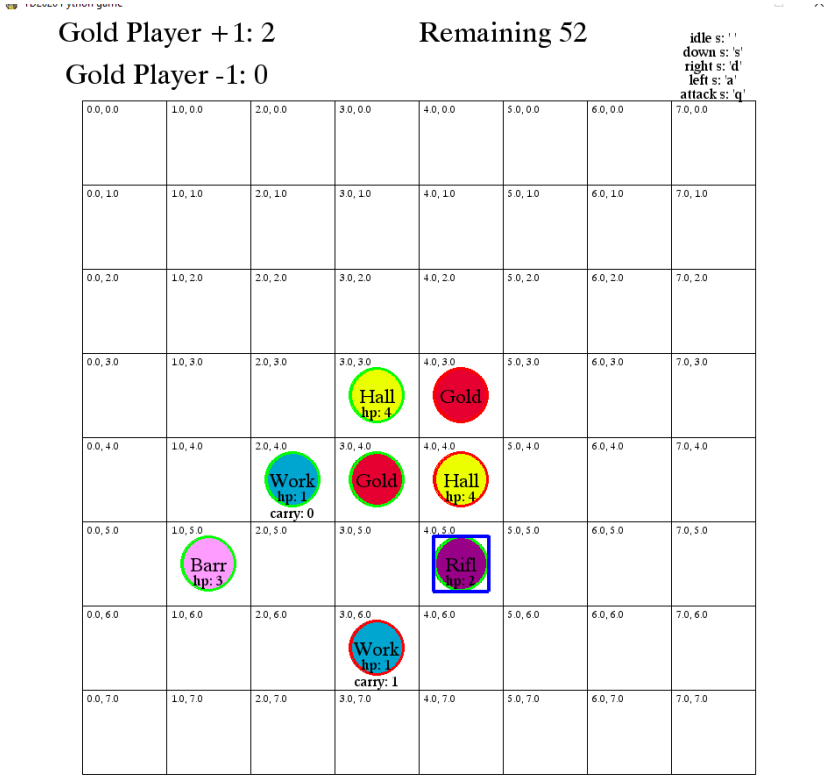
- Premikanje: igralec lahko premakne delavce in vojake za 1 kvadratek v vseh 4 smereh če so prazni s klikom na eno od 4 mest.
- Napadanje: z izbrano vojaško enoto lahko uporabnik napade sovražne enote, ki so v dosegu.
- Zbiranje in vračanje sredstev: z izbranim delavcem lahko uporabnik porabi sredstva, tako da klikne desno miškino tipko na polje zlata, če

je v dosegu. To velja tudi za vračanje sredstev, vendar mora biti delavec v bližini mestne hiše.

- Gradnja: Za gradbene enote in zgradbe mora uporabnik uporabiti eno od bližnjic na tipkovnici.

- Nedejaven: igralec lahko z izbranim igralcem pritisne presledek za mirovanje.

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



Slika 6.1: Na zgornji sliki človeški igralec igra izgrajeno strateško igro proti računalniškim nasprotnikom.

6.2 Unreal Engine 4

Unreal Engine 4 je odprtokodni program podjetja Epic Games, ki je namenjen hitri izdelavi računalniških iger. Obstajajo še drugi celostni pogoni kot je Unity.
Razliko med tema pogonoma je dobro predstavil Marko Kladnik [2].
Unreal Engine 4 omogoča hitro ustvarjanje iger s pomočjo posebnih diagramov (angl. blueprint) in hkrati podpira programski jezik C++, ki ga uporabimo za hitro izvedbo velikega števila matematičnih izrazov.

Potrebno je bilo preslikati akcije in figure v urejevalnik Unreal Engine, da se tam enote primerno premikajo in izvajajo akcije. Potrebno je bilo (mapirati) animacije, efekte, da premikanje in napadanje zgleda dokaj realistično.

Ta predstavitev omogoča tudi igranje dveh človeških igralcev enega proti drugemu preko internetne mreže, kjer vsak igralec pridobiva priporočene ukaze iz modela. Človeški igralec lahko igra tudi proti računalniškim igralcem, ki pa vsaki 2 sekundi zahteva za novo najboljšo akcijo. Lahko si pa ogledamo dva računalniška igralca igrati drug proti drugemu.

6.2.1 Prenos stanja igre

napišeš kko encodaš game pa ga pošleš pythonu, tm pa dobiš best action pa pol nazaj...

Slika 6.2: Zgornja slika predstavlja igranje igre dveh računalniških nasprotnikov enega proti drugemu.

Poglavje 7

Rezultati

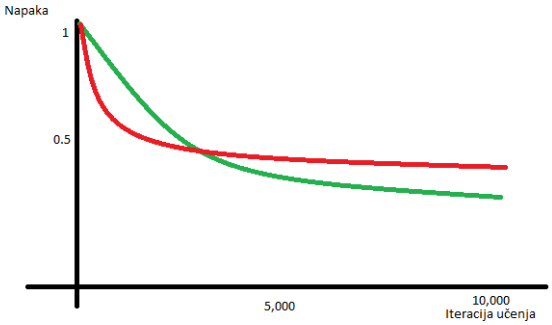
Opišeš parametre in čas učenja Opišeš rezultate pri različnih nastavitvah parametru igre - recimo bolj stroga krivulja, manj stroga, več denarja per mine, cenejši heal... Pa s kermu encodingom so bli bolši razultati Pa napišeš primerjava proti naključnim igralcem in greedy- kjer ima greedy player kriterij za število življenja maximize

7.1 Izbira ustavitvene funkcije

Pri izbiri funkcije za timeout smo zelo počasi prišli do dobrih rezultatov. Pri izbiri ranjujoče funkcije pa smo dobili naslednje rezultate.

7.2 Izbira kodiranja

Pri uporabi desetiškega kodiranja smo dobili naslednje rezultate. Pri One Hot kodiranju pa smo dobili naslednje rezultate.



Slika 7.1: Zgornja slika predstavlja vzorec, kako bo izgledal graf prikaza podatkov pri ustavitvenih funkcijah.

Poglavje 8

Zaključek

V zaključku obrazložim ali se algoritem splača vpeljati za moderno RTS igro in kje so vsi problemi. Opišeš probleme kot so v balanciranju igre, da se nevronska mreža lahko prav nauči in z ugotavljanjem konca igre.

Igra in AlphaZero algoritem je na voljo na naslednjih repozitorijih:
https://github.com/JernejHabjan/TrumpDefense2020
in
https://github.com/JernejHabjan/alpha-zero-general

Literatura

[1] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[2] Marko Kladnik. Primerjava igralnih pogonov unity in unreal engine. Diplomska naloga, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani, 2015.

[3] Santiago Ontanón. Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research*, 58:665–702, 2017.

[4] Santiago Ontañon, 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.

[5] 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.

[6] 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 self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.