Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jernej Habjan

Hevristični algoritmi v RTS igri TD2020

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matej Guid

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 celi družini pri podpori pri študiju, sostanovalcem v Ljubljani in vsem kolegom, ki so mi priskočili na pomoč. Prav tako se zahvaljujem mentorju doc. dr. Mateju Guidu za vodenje.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Realno-časovne strateške igre **3**

2.1 Izdelava strateške igre 3

2.2 Trump Defense 2020 4

2.3 Obstoječa dela na področju umetne inteligence v realno-časovnih igrah 6

2.4 Odprta vprašanja o umetni inteligenci v realno-časovnih igrah 8

3 Algoritmi **9**

3.1 Monte-Carlo drevesno preiskovanje 9

4 Implementacija algoritmov **11**

4.1 implementacija naključnosten algoritma 11

4.2 implementacija MCTS 11

4.3 sprememba preiskovalnega parametra pri ocenjevanju 11

5 Ovrednotenje rezultatov **13**

5.1 Simulacija igre računalnika proti računalniku 13

5.2 Igranje računalnika proti človeku 13

6 Zaključek **15**

Literatura **17**

Seznam uporabljenih kratic

kratica	angleško	slovensko
MCTS	Monte Carlo tree search	Monte-Carlo drevesno preiskovanje
UCB	upper confidence bound	zgornja meja zaupanja
RAVE	rapid action value estimation	hitra ocena učinka
CMAB	combinatorial multi-armed bandit	kombinatorni več-rokovni bandit
UE4	game engine Unreal Engine 4	celostni pogon Unreal Engine 4
RTS	real-time strategy	realno-časovna strateška

Povzetek

Naslov: Hevristični algoritmi v RTS igri TD2020

Avtor: Jernej Habjan

V okolju UE4 smo razvili algoritem, ki namesto klasičnega programiranja umetne inteligence v RTS igrah uporablja hevrističen algoritem,ki sam ugotovi, katera je optimalna odločitev v določeni situaciji in jo za tem izvede, pri katerem je problem velik preiskovalni prostor, ki se pojavlja v RTS igrah. Pri igri Trump Defense 2020, narejeni v UE4 smo uporabili algoritem Monte-Carlo drevesno preiskovanje, ki deluje na principu hevurističnega preiskovanja prostora, kjer imamo prostor stanj (graf, drevo), množico dosegljivih stanj in povezave med stanji. Najbolj uspešno se je izkazal CMAB izpeljanka algoritma MCTS, ki je dajal poudarek na računanju vrednosti akcije proti številu simulacij.

Ključne besede: hevristika, Monte-Carlo drevesno preiskovanje, CMAB.

Abstract

Title: Heuristic algoritms in RTS game TD2020

Author: Jernej Habjan

In the UE4 environment, we developed an algorithm that uses a heuristics instead of classically programmed artificial intelligence, which itself determine the optimal decision in a given situation and performs it, with the problem being a large investigative space that appears in RTS games. In the Trump Defense 2020 game, made in UE4, we used the Monte Carlo tree search algorithm to investigate trees, which operates on the principle of heuristic space exploration, where we have a space of states (graph, tree), a multitude of accessible states and connections between states. The most successful was the CMAB implementation of the MCTS algorithm, which emphasized calculating the value of the action against the number of simulations.

Keywords: heuristics, Monte Carlo tree search, CMAB.

Poglavje 1

Uvod

Računalnik, ki igra proti človeškem nasprotniku lahko sprogramiramo na klasičen način pogoj - akcija. Tako so narejene klasične realno-časovno strateške igre, vendar gre veliko časa za implementacijo posameznih sovražnikovih napadov in nabiranju virov. Lahko pa razvijemo algoritem, ki sam ugotovi, katera je optimalna odločitev v določeni situaciji in jo za tem izvede. Na preprostih problemih z malo odločitvami in kratkimi igrami kot je igra križci in krožci, lahko izberemo preprostejše algoritme kot je algoritem Minimax, kjer igralec hoče povečati svojo možnost zmage, nasprotnik pa mu hoče to možnost zmanjšati. Obstajajo tudi algoritmi, ki se naučijo igre iz učne množice, ki predstavlja nekaj iger, pri tem pa računalnik ugotovi zakonitosti igre in način igranja. Taki algoritmi so nevronske mreže ali globoke nevronske mreže, ki pa so zelo računsko potratni, vendar vračajo izjemne rezultate. Lahko pa uporabimo algoritem Monte-Carlo drevesno preiskovanje, ki deluje na principu hevurističnega preiskovanja prostora, kjer imamo prostor stanj (graf, drevo), množico dosegljivih stanj in povezave med stanji. Algoritem bom uporabil na svoji igri narejeni v celostnem pogonu Unreal Engine 4 imenovani Trump Defense 2020.

Razvijanje inteligentnega agenta v realno-časovnih 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 delova-

nja 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 boljše akcije, vendar bo to trajalo zelo dolgo, saj bo moral preiskati cel preiskovalni prostor, ki pa pri realno-časovnih strateških igrah zna biti zelo velik. Da bi agent lahko poiskal cel prostor, bi ga morali zelo abstraktirati, vendar še takrat se moramo posluževati algoritmov kot je Monte-Carlo drevesno preiskovanje, ki uporablja naključnost, s katero se pomika skozi preiskovalni prostor. S takim algoritmom lahko dosežemo inteligentnega agenta, ki se prilagaja nasprotnikovim akcijam, vsako igro izbira nov način igranja in deluje dovolj hitro brez daljšega učenja na superračunalnikih, kot to potrebujejo nevronske mreže.

Poglavje 2

Realno-časovne strateške igre

Njihov cilj je gradnja ekonomije, vojske in z njo premagati nasprotnika z porušenjem njihovih hiš. Razlike realno-časovnih strateških iger od šaha so naslednje: Akcije se izvršujejo hkrati in niso potezne Poteze lahko trajajo dlje časa(hoja do hiše) Igra je "realno-časovna" kar pomeni, da se mora v vsakem trenutku igralec odločiti za potezo, za katero lahko pri šahu razmišlja dlje časa (vsakih 42 milisekund) Večina iger je le delno vidnih, kjer vidimo nasprotnikove objekte samo ob določenih pogojih Prostor preiskovanja akcij je veliko večja.

2.1 Izdelava strateške igre

Igro moram izdelati v celostnem pogonu Unreal Engine 4. Igra bo realno-časovna, kar pomeni, da jo moram dobro optimizirati, če hočem poganjati hevristične preiskovalne algoritme. V tem okolju bom lažje zasnoval grafično prezentacijo algoritma.

2.1.1 Celostni pogon 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 [**?**].
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.
Pogon podpira odločitvena drevesa, v katera lahko sprogramiramo računalnikovo delovanje ob določenih pogojih, vendar je v igri Trump Defense 2020 narejena inteligenca računalnika s hevrističnim algoritmom Monte-Carlo drevesno preiskovanje.
Izdelati moram poganjalca algoritma, ki bo izbral določen algoritem in z njim igral proti nasprotniku ali proti človeškem igralcu.
Prav tako moram izdelati osebke in njihove akcije (npr. postavi hišo). Te osebke bom pa hranil v stanju igre, ki pa ga algoritem uporablja, ki je abstrakcija za to, kateri osebki in akcije so trenutno na voljo.

2.2 Trump Defense 2020

Opis igre in problema: Realno-časovna igra, kjer imamo na voljo 2 rase - Humans in Aliens. Svet je razdeljen na 2 mreži, kjer je mreža za gradnjo in postavljanje stavb velika 32 x 32 kvadratkov, mreža za hojo pa 8 x 8 kvadratkov. Tipične velikosti mape so od 64 x 64 do 256 x 256 kvadratkov. Kontroliramo lahko do 200 enot na igralca, kjer imamo za vsako raso približno 30 - 35 različnih enot in stavb. Tu je razvidna kompleksnost igre, ki je s preprostim preiskovanjem prostora ne premagamo.
Predpostavke:

- Da je okrog enote največ 16 sovražnih enot,
- enota se lahko premakne na sosednih 8 kvadratkov, čeprav se enota v realnosti lahko premakne kamorkoli na mapi.
- Delavec lahko postavi hišo samo na trenutno lokacijo (njegova lokacija)

2.2.1 Izzivi realno-časovnih iger

Zgodnje raziskave realno-časovnih iger so ugotovile naslednje izzive:

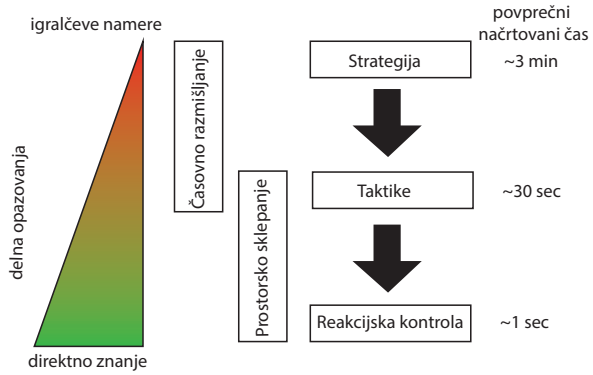
- Upravljanje z viri
- Izbira akcij ob nevednosti
- Prostorsko in časovno razmišljanje
- Sodelovanje med večimi agenti
- Modeliranje nasprotnika in učenje
- Nesporno načrtovanje v realnem času

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 naredil, 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 2.1: 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.



Slika 2.1: Razdelitev nalog.

Pogosto razdelimo odločanje na dva dela:

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

2.3 Obstoječa dela na področju umetne inteligence v realno-časovnih igrah

Izzivi realno-časovnih iger in obstoječa dela so povzetek članka iz leta 2013 A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft [1]

2.3.1 Strategija

To je še odprt problem, ki je v strateških igrah velikokrat uporabljen princip "hard-coded" ali direktnega kodiranja strategije, ki uporabljajo avtomate končnih stanj, kjer lahko razbijemo delovanje na več stanj kot so napadanje, nabiranje surovin, popravilo itd. in hitro menjavanje med njimi. Direktno kodiranje je prinesel dobre uspeške, vendar se lahko igralec nauči strategije in ga tako agenta hitro porazi. Planirani pristopi ponujajo večjo prilagodljivost kot direktno kodirani. Prav tako so izvajali nadzorovano učenje na podlagi podatkovnega rudarjenja na označenih StarCraft posnetkih, s katerih so se naučili sekvenco postavitve stavb. Prav tako je pa tu problem, da je StarCraft le delno viden, kjer le del algoritmov deluje ob nevednosti nasprotnikovih akcij.

2.3.2 Taktika

Razdeljena je na naslednje sklope:

- Analiza terena, kjer je Perkins apliciral Voronoi dekompozicijo na mapo, ki jo razbije na manjše delčke glede na pozicijo ožin. Hale je predstavil mapo z 2D geometrijsko navigacijsko mrežo
- Odločanje, kjer so bili uporabljeni različni pristopi kot strojno učenje in iskalna drevesa, skriti markovski modeli itd. Nekateri so uporabili spodbujevalno učenje in nevronske mreže, ki zagotovi ponovljivost taktike. Pri uporabi preiskovalnega drevesa je potrebno zagotoviti dovolj veliko abstrakcijo igre.

2.3.3 Reakcijski nadzor

Se osredotoča na nadzor enot, da jim povečamo izkoristek v kompleksnih bitkah na večnivojskem terenu. Raziskovalci so izdelali vplivne karte, uporabili A* za izogib pastem, uporaba navigacije za izogibanje nasprotnikovega ognja itd. Pri tehnikah, ki temeljijo na raziskavi terena je problem veliko število

parametrov, ki so ga nekateri razreševali z uporabo spodbujevalnega učenja, vendar lahko model hitro konvergira k lokalnem maksimumu. Pri raziskovalnih drevesih je Chirchill predstavil varacijo alpha-beta iskanja, ki deluje za hkratne gibe in za akcije, ki trajajo dlje časa.

2.3.4 Celostni pristopi

Področje ni dovolj raziskano, vendar pristopi, ki problem razbijejo na manjše dele v praksi vračajo boljše rezultate. Celostni pristopi, izvedeni z Monte-Carlo drevesnim preiskovanjem so bili izvedeni na manjših realno-časovnih strateških igrah.

2.4 Odprta vprašanja o umetni inteligenci v realno-časovnih igrah

Zaenkrat nimamo agenta, ki bi ugotavljal svoje taktike, kot naprimer spuščanje enot v sovražnikovo bazo na podlagi nasprotnikove strategije.

- Učenje in prilagajanje Prilagajanje na nasprotnikovo strategijo, učenje te strategije. Trenutni agenti se odločajo o strategiji glede na direktno kodiranih pogojih. Učenje na podlagi izkušenj. Učenje iz posnetkov.
- Planiranje Nadzorno načrtovanje pod omejitvami v realnem času. Razvite so metode za manjše strateške igre. Nadzorno načrtovanje pod nevednostjo zaradi delno vidne mape
- Integracija Integracija modulov za obravnavanje posameznih delov igre
- Znanje o domeni Znamo uporabiti določena znanja kot so zaporedje postavljanja stavb, ampak ne znamo še zakodirati strategije za ostale tipe iger.

Poglavje 3

Algoritmi

Ko bom imel izdelano ogrodje strateške igre, se bom lahko posvetil algoritmom. Preiskal bom naslednje algoritme:

3.1 Monte-Carlo drevesno preiskovanje

Implementacija algoritma v UE4 v Python programskem jeziku preko vtičnika Unreal Engine Python, ki bo poenostavila berljivost algoritma in lažjo implementacijo odprtokodnih rešitev različnih variacij MCTS algoritma, kot tudi nevronske mreže za uteženo izbiranje akcij.

3.1.1 UCB1, CAB, Naive MCTS

Izmed MCTS algoritmov bom prvo implementiral UCB1 algoritem.

$$R(s,a) = Q(s,a) + c\sqrt{\frac{\ln s}{N(s,a)}} \tag{3.1}$$

UCB1 se od ostalih algoritmov razlikuje z ocenjevalno funkcijo, ki je pa pri UCB1 najlažja. Prav tako bom implementiral izpeljanke algoritmov CAB in Naiven MCTS, ki mi bosta pomagala primerjati algoritme med sabo in izbiro najboljše rešitve.

3.1.2 Nevronske mreže

Prav tako se bo za uteženo izbiro akcij v vozliščih implementirala variacija nevronske mreže.

Prav tako bom moral hkrati opisati, zakaj sem se za kateri algoritem odločil.

Poglavje 4

Implementacija algoritmov

Ko bom imel algoritem dokončan, ga bom lahko implementiral v igro.

4.1 implementacija naključnosten algoritma

Sprva bom implementiral naključnosten algoritem, ki bo naključno izbiral akcije dveh igralcev. To mi bo dovolilo da preverim, ali se vse akcije uspešno izvajajo in da se stanje igre uspešno prenaša iz cikla v cikel. Prav tako mi bo omogočilo pripravo Python okolja, v katerega bom moral pridobiti vse podatke o igri, ki so potrebni za obdelavo.

4.2 implementacija MCTS

Nadgradnja naključnostnega algoritma bo osnoven MCTS, ki bo uporabljal UCB1 ocenjevalno funkcijo za izbiro vozlišča.

4.3 sprememba preiskovalnega parametra pri ocenjevanju

Tu bom še spremenil razne ocenjevalne funkcije algoritmov, spremenil preiskovalni parameter, ki določa utež preiskovanja prostora proti direktnemu

odločanju. Prav tako bom tu implementiral še nevronske mreže, ki bodo določale utež za izbiro vozlišča.

Poglavje 5

Ovrednotenje rezultatov

5.1 Simulacija igre računalnika proti računalniku

Rezultate bom ovrednotil tako, da bom različne algoritme poganjal drug proti drugemu in si hkrati beležil število zmag. Ko bo število iger dovolj veliko, da bo rezultat stabilen bom postopek ponovil proti drugemu algoritmu.

	UCB1	CAB	Naive MCTS	Opis
UCB1	22%	22%	22%	Opis1.
CAB	22%	22%	22%	Opis2.
Naive MCTS	22%	22%	22%	Opis3.

5.2 Igranje računalnika proti človeku

Nakoncu bom poskusil algoritem testirati proti znancem, ki bodo pa podali subjektivno oceno o težavnosti algoritma in število zmag algoritma proti njim. Tako bom izvedel, ali je algoritem koristen proti človeškim igralcem in ali se lahko človeški igralec nauči taktike, ki deluje najbolje proti algoritmu, če ta vedno izbira podobne akcije.

Pri simulaciji računalnika proti računalniku, se bom osredotočil na rezultate kot so naprimer povprečno število simulacij pri MCTS algoritmu in

konsistentnost ukazov. Pri igranju proti človeku bom pa analiziral nekaj odločitvenih dreves ki jih je računalnik zgeneriral in ocenil odločitve na podlagi mojega predznanja igre.

	UCB1	CAB	Naive MCTS	Opis
Oseba1	22%	22%	22%	Opis1.
Oseba2	22%	22%	22%	Opis2.
Rezultati	22%	22%	22%	

Poglavje 6

Zaključek

V zaključku bom povzel rezultate računalnika proti računalniku, kot tudi uspešnost algoritmov proti človeškim igralcem. Povzel bom koristnost hevrističnih algoritmov na realno-časovnih strateških igrah in vpliv abstrakcije prostora.

Literatura

[1] 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.