Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Mitja Lapajne

Program za igranje taroka z uporabo drevesnega preiskovanja Monte-Carlo

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana 2011

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Izjava o avtorstvu diplomskega dela

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00011/2011
Datum: 04.11.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MITJA LAPAJNE**

Naslov: **PROGRAM ZA IGRANJE TAROKA Z UPORABO DREVESNEGA PREISKOVANJA MONTE-CARLO**

A PROGRAM FOR PLAYING TAROK USING MONTE-CARLO TREE SEARCH

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Preiskovanje dreves iger z metodo Monte-Carlo je novejši pristop k računalniškemu igranju iger. Naloga je izdelati program za igranje igre s kartami tarok z uporabo te metode preiskovanja drevesa možnih nadaljevanj igre. Program optimizirajte z eksperimentalnim spreminjanjem parametrov programa. Ocenite igralno moč programa z igranjem proti naključnemu igralcu in proti človeku.

Mentor:

akad. prof. dr. Ivan Bratko

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Spodaj podpisani Mitja Lapajne, z vpisno številko **63070190**, sem avtor diplomskega dela z naslovom:

Program za igranje taroka z uporabo drevesnega preiskovanja Monte-Carlo

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom akad. prof. dr. Ivana Bratka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 14. decembra 2011 Podpis avtorja:

Zahvala

Rad bi se zahvalil mentorju, akad. prof. dr. Ivanu Bratku, za trud, izkazano potrpežljivost, ter številne predloge in popravke med izdelavo diplomskega dela.

Posebna zahvala pa gre staršem, ki so mi študij omogočili, ter me tekom njega tudi vseskozi podpirali.

Kazalo

Povzetek

Drevesno preiskovanje Monte-Carlo je relativno nov algoritem, ki daje obetavne rezultate na področju splošnega igranja iger. V tem delu predstavimo implementacijo in rezultate aplikacije drevesnega preiskovanja Monte-Carlo za igro tarok. Drevesno preiskovanje Monte-Carlo smo prenesli na igro taroka, kot tudi na spremenjeno različico taroka s popolno informacijo, ter opravili več eksperimentov z različnimi parametri programa. Preverili smo, kako se algoritem obnese pri različnih vrednostih parametra C, in kako število simulacij ter dodatno hevristično znanje vplivata na kvaliteto igranja. Rezultati kažejo, da algoritem doseže raven priložnostnega igralca taroka in kaže potencial za nadaljnje izboljšave.

Ključne besede: metode Monte-Carlo, drevesno preiskovanje Monte-Carlo, tarok, igra s kartami

Abstract

Monte-Carlo tree search is a relatively new algorithm showing promising results for general game playing. This work presents an implementation and results of the Monte-Carlo tree search algorithm applied to the card game Tarok. We applied the Monte-Carlo tree search algorithm to the game of Tarok as well as to a modified version of Tarok with perfect information and conducted several experiments with different parameters of the program. We investigated how different values of parameter C, number of simulations per move and added heuristic knowledge impact the playing strength of the algorithm. The results suggest that the Monte-Carlo tree search algorithm is a viable option for computer Tarok game playing and demonstrates the playing strength of an occasional player as well as offers many potential opportunities for further improvement.

Keywords: Monte-Carlo methods, Monte-Carlo tree search, Tarok, card game

Poglavje 1

Uvod

1.1 Motivacija

Računalniško igranje iger od nekdaj privablja številne raziskovalce umetne inteligence. Trdo delo, novi pristopi in hiter razvoj strojne opreme so pripomogli k dejstvu, da je konec devetdesetih let prejšnjega stoletja računalnik v šahu že premagal takratnega svetovnega prvaka, Garija Kasparova[2]. Številni izzivi pa so še vedno ostali, predvsem računalniško igranje igre go, ter mnoge igre s kartami še vedno predstavljajo trd oreh za računalnik.

Leta 2006[13] pa se je pojavil nov, zanimiv pristop drevesnega preiskovanja Monte-Carlo (MCTS) za igro go, ki je dal dobre začetne rezultate, in sprožil pravo poplavo programov, ki temeljijo na temu algoritmu, in so čedalje boljši. Prav zaradi dobrih rezultatov na temu področju, bi radi preizkusili primernost tega algoritma še za igro tarok. V pričujočem delu začnemo s krajšim pregledom algoritmov, ki so vodili do algoritma drevesnega preiskovanja Monte-Carlo, nadaljujemo z obširno predstavitvijo omenjenega algoritma, nato pa opišemo še praktično rešitev za igro tarok ter analiziramo rezultate in določimo smernice za nadaljnje delo.

1.1.1 Cilji

Cilji tega dela so:

- Preučiti algoritem drevesnega preiskovanja Monte-Carlo ter algoritme na katerih ta pristop temelji.
- Implementirati algoritem za igro tarok in analizirati rezultate.

1.1.2 Raziskovalna vprašanja

Raziskovalna vprašanja, ki jih v tem delu preučujem, so:

- Ali drevesno preiskovanje Monte-Carlo prinese dobre rezultate tudi pri igri tarok ter kaže tak potencial, kot ga je pokazal za igro go?
- Ali izboljšave algoritma, ki so se izkazale pri igri go, dajo boljše rezultate tudi pri taroku? (So domensko neodvisne?)

1.1.3 Hipoteze

- Drevesno preiskovanje Monte-Carlo s popolnoma naključnimi simulacijami bo pri majhnemu številu simulacij dalo precej šibkega igralca. S povečevanjem števila simulacij se bo kvaliteta igranja izboljšala.
- Z izboljšavami, predvsem z določeno uporabo hevristik pri simulacijah, se bo kakovost igralca izboljšala.

1.2 Zgradba dela

- **Poglavje 1** predstavi zgradbo dela, motivacijo, raziskovalna vprašanja in hipoteze.
- **Poglavje 2** predstavi klasifikacijo iger in algoritem min-max.
- **Poglavje 3** predstavi splošen koncept metod Monte-Carlo, prejšnje delo na Monte-Carlo evaluacijah in podrobno opiše algoritem drevesnega preiskovanja Monte-Carlo.
- **Poglavje 5** predstavi zgodovino in pravila igre tarok ter prejšnje pristope k računalniškemu igranju taroka.
- **Poglavje 6** opiše praktično implementacijo algoritma za igro tarok ter predstavi dobljene rezultate.
- **Poglavje 7** interpretira dobljene rezultate in določi smernice za nadaljnje delo.

1.3 Prispevki

Za izvedbo dela sem preučil številne članke s področja drevesnega preiskovanja Monte-Carlo. Glavni prispevek mojega dela je implementacija algoritma drevesnega preiskovanja za igro tarok ter izvedba različnih eksperimentov pri igranju te igre. To je zanimivo predvsem, ker na področju iger s kartami ter iger z nepopolno informacijo še ni bilo izvedenih veliko raziskav s tem algoritmom. Izkaže se, da algoritem predstavlja obetavno rešitev za računalniško igranje taroka, in doseže raven priložnostnega igralca taroka.

Poglavje 2

Teoretično ozadje

V tem poglavju klasificiramo različne tipe iger ter preučimo algoritem minmax.

2.1 Tipi iger

Igre delimo na deterministične in stohastične[2], in na take s popolno in take z nepopolno informacijo. Tarok, na primer, je stohastična igra nepopolne informacije. Stohastična zaradi vpliva sreče (žreb kart) in igra nepopolne informacije vsled dejstva, da ne moremo videti nasprotnikovih kart.

- **Popolna informacija** ali **nepopolna informacija**. Pri igrah s popolno informacijo lahko vsak igralec vidi celotno stanje igre vseh ostalih igralcev. Vsi igralci imajo tako enako informacije. Primera takih iger sta šah in go, kjer so vse figure in pozicije le-teh vidne obema igralcema. Pri igrah z nepopolno informacijo nekateri deli igre niso vidni drugim igralcem. Tako pri taroku ne moremo videti kart ostalih igralcev.

- **Deterministična** ali **stohastična**. Naslednje stanje igre je pri determinističnih igrah določeno le z trenutnim stanjem. V nasprotju s tem pri stohastičnih igrah nastopajo stohastični dogodki, ki vplivajo na potek igre. Taki dogodki so met kocke, ali pa deljenje kart igralcem.

2.2 Drevo igre

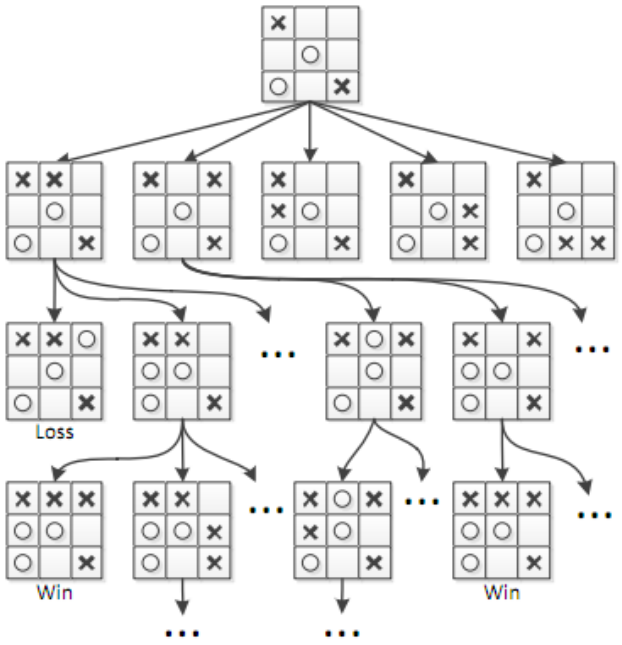
Eden izmed načinov predstavitve iger je v obliki drevesa igre[6]. Drevo igre je drevo, v katerem je stanje igre opisano z drevesom. Posamezna vozlišča drevesa predstavljajo trenutno pozicijo (stanje) igre, veje iz vsakega vozlišča pa možne poteze za to pozicijo.

2.3 Število vejitev in globina drevesa

Številu možnih vejitev na vsaki točki (vozlišču) drevesa rečemo faktor vejitve in je precej dober pokazatelj težavnosti obravnavane igre za računalnik[6]. Različne igre imajo tudi različne globine drevesa: največje število potez igre. V igri tri v vrsto vsak igralec izmenično polaga svoj simbol na igralno mesto. Ker je igralnih mest 9, je pri vsaki igri največ 9 potez. Drugače je pri šahu, kjer je potez lahko praktično neskončno. Drevo je pri taki igri lahko izjemno globoko, tudi če je faktor vejitve relativno majhen. Za računalnike so lažje igre z majhnim faktorjem vejitve in globokim drevesom, kot igre s plitvim drevesom in ogromnim faktorjem vejitve[6].

2.4 Pozicijska evaluacijska funkcija

Računalnik potezne igre igra tako, da pogleda katere poteze so mu na voljo, in izbere eno od njih. Da bi izbral eno od njih, mora vedeti, katere poteze so boljše od drugih. To znanje mu poda programer v obliki hevristične pozicijske evaluacijske funkcije[6]. Naloga pozicijske evaluacijeske funkcije je, da oceni trenutno stanje igre v določenem vozlišču. Če gre za končno stanje, je ta ocena kar končna vrednost igre. Precej težje pa je oceniti stanje v nekončnih pozicijah. Stanje ocenimo na podlagi določenih pravil in domenskega znanja o igri. Ena ključnih prednosti algoritma drevesnega preiskovanja Monte-Carlo, ki ga bomo spoznali v poznejših poglavjih je, da ne zahteva pozicijske evaluacijske funkcije.



Slika 2.1: Primer dela igralnega drevesa za igro 3 v vrsto.

Na sliki 2.1 je predstavljeno igralno drevo za igro 3 v vrsto. Z igralnimi drevesi lahko modeliramo katere poteze so možne v danem stanju, in katera nova stanja lahko dosežemo. Za grajenje drevesa igre, in nato določanje katero potezo odigrati obstaja več algoritmov, eden najbolj znanih izmed njih pa je algoritem min-max.

2.5 Algoritem min-max

Algoritem min-max je rekurzivni algoritem za iskanje optimalne strategije v igri z n-igralci, ponavadi igri z dvema igralcema[12]. Vsako stanje v igri je povezano z določeno vrednostjo igre. Ta vrednost je izračunana s pozicijsko evaluacijsko funkcijo, in kaže kako dobro bi bilo za igralca, da bi dosegel to pozicijo. Glavna ideja algoritma je, da predvideva, da oba igralca igrata optimalno. Ko je na vrsti MAX igralec (to smo mi) vedno izberemo potezo, ki bo nam prinesla največji donos. Prav tako, ko bo na vrsti MIN igralec (naš nasprotnik), predvidevamo, da bo izbral potezo, ki bo največ zmanjšala donos našega igralca MAX. Ko imamo vrednosti listov vozlišč rekurzivno izračunamo vrednosti stanj ostalih vozlišč[4].

Algoritem 1 int minmax(vozlisce) {

```
if jeList(vozlisce) then
  return oceni(vozlisce)
end if
if jeMaxVozlisce(vozlisce) then
  return največja vrednost vseh otrok vozlisca
end if
if jeMinVozlisce(vozlisce) then
  return najmanjša vrednost vseh otrok vozlisca
end if
}
```

Poglavje 3

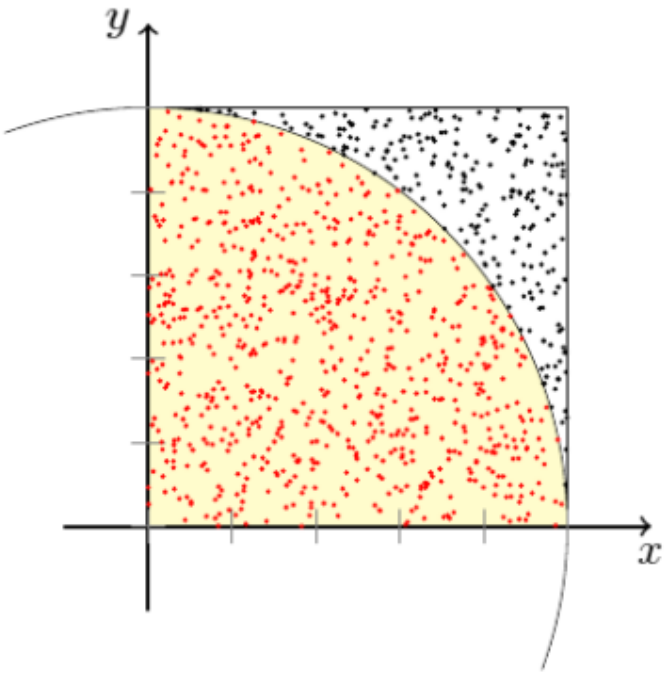
Metode Monte-Carlo

Metode Monte-Carlo spadajo v razred računalniških algoritmov, ki rezultat izračunajo na podlagi ponavljajočih naključnih porazdelitev[14]. Uporabljane so predvsem za računalniške simulacije fizičnih in matematičnih sistemov, ko je neizvedljivo izračunati točen rezultat z determinističnim algoritmom. Za nastanek se največkrat navaja leto 1944, ko so fiziki v Los Alamosu raziskovali kolikšno pot nevtroni prepotujejo skozi različne materiale. Problema ni bilo moč rešiti analitično, zato sta John Von Neumann in Stanislaw Ulam predlagala, da bi rešitev poskusili poiskati na podlagi računalniškega eksperimenta, temelječega na generiranju naključnih števil, ki ponazarjajo stanje sistema. To metodo so kasneje uporabljali tudi v sklopu projekta Manhattan, kjer so razvijali atomsko bombo, ter ji takrat tudi dali ime Monte-Carlo. Z vse hitrejšim razvojem računalnikov ter posledično izjemnim povečanjem računske moči te metode dobivajo še večji pomen, prav gotovo pa bo tako tudi v prihodnje.

3.1 Računanje števila π

Kot uvod v Monte-Carlo metode in oris koncepta teh metod si oglejmo primer izračuna števila π z metodo Monte-Carlo[2]. Vrednosti števila π lahko izračunamo takole: iz ravninske geometrije vemo, da je ploščina kroga z ra-

dijem R enaka πR^2. Ker je ploščina kvadrata, ki ima stranico dolgo R enaka R^2 je razmerje kroga z radijem R in kvadrata s stranico dolgo R enako π. To razmerje lahko izračunamo z naključnim polaganjem n točk znotraj kvadrata z včratnim krogom. Če preštejemo število točk, ki ležijo v krogu in to delimo s številom vseh točk, dobimo približek četrtine prej omenjenega razmerja, torej $\frac{\pi}{4}$.



Slika 3.1: Psevdonaključno porazdeljene točke znotraj kvadrata s stranico R. Razmerje med točkami znotraj četrtine krožnice, in vsemi točkami znotraj kvadrata je enako $\frac{\pi}{4}$.

3.2 Monte-Carlo Evaluacije

V začetku 90ih let se prvič pojavijo Monte-Carlo evaluacije (MCE) pri računalniškem igranju igre othello[1]. Ta pristop se uporablja in izboljšuje tudi v naslednjih letih, leta 2003 tako Bouzy in Helmstetter[8] razvijeta program za igranje igre

go, ki je tekmoval tudi na Računalniških olimpijadah leta 2003 in 2004[1].

Monte-Carlo evaluacije delujejo na sledeč način[1]: če označimo trenutno stanje v igri s P, potem izvajamo več simulacij od trenutnega stanja do konca igre. Poteze v simulaciji od trenutnega stanja do konca so za vse igralce izbrane naključno. Vsaka taka simulacija da nek rezultat, R_i, ki vsebuje rezultate vseh igralcev. Vrednost nekega stanja P pa je povprečje rezultatov vseh simulacij, torej $E_n(P) = \frac{1}{n} \sum Ri$.

3.3 Drevesno preiskovanje Monte-Carlo

Algoritem min-max je John von Neumann opisal že leta 1928[12]. Metode Monte-Carlo so bile uporabljane že v 40ih letih prejšnjega stoletja[14]. Šele leta 2006 pa Rémi Coulom[15] in drugi raziskovalci združijo te dve ideji v nov pristop k računalniškemu igranju igre go, in ga poimenujejo drevesno preiskovanje Monte-Carlo. Kocsis in Szepesvári[3] ta isti pristop, prav tako leta 2006, formalizirata v algoritem UCT. Od takrat je nastalo že več kot 150 raziskovalnih člankov na temo drevesnega preiskovanja Monte-Carlo, predlaganih je bilo več kot 50 različnih variant, izboljšav in optimizacij[15].

V tem poglavju natančneje predstavimo algoritem drevesnega preiskovanja.

3.3.1 Struktura drevesnega preiskovanja Monte-Carlo

MCTS je iskalna metoda najboljši-najprej, ki ne zahteva pozicijske evaluacijske funkcije in temelji na naključnem raziskovanju iskalnega prostora[1]. Glede na rezultate prejšnjih raziskovanj, algoritem postopoma v pomnilniku gradi igralno drevo, in postaja vse bolj natančen pri ocenjevanju najbolj obetavnih potez. Sestavljen je iz štirih faz, ki se ponavljajo dokler se algoritem izvaja.

- **Izbiranje vozlišča**. Začenši s korenskim vozliščem R, rekurzivno izberi optimalnega otroka trenutnega vozlišča, dokler ne prideš do vo-

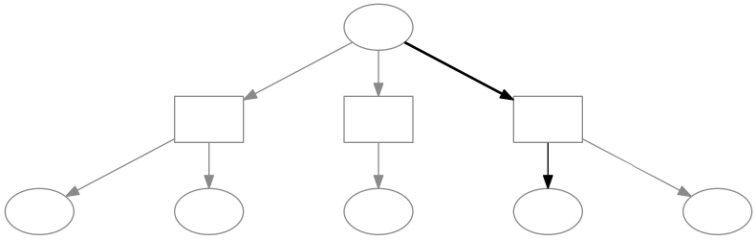
zlišča L, ki je list.

- **Razvoj vozlišča**. Če L ni končno vozlišče (konec igre), potem ustvari novo vozlišče.
- **Simulacija**. Simuliraj igro od stanja v novo narejenem vozlišču do konca igre.
- **Vzvratne vrednosti**. Z rezultatom simulacije posodobi vzvratne vrednosti vozlišč, ki si jih obiskal med simulacijo.

3.3.2 MCTS

V tem delu bomo pregledali strategije, med katerimi lahko izbiramo pri posameznih štirih korakih MCTSja: izbiranju vozlišča, razvijanju vozlišča, simulaciji in določanju vzvratnih vrednosti.

3.3.3 Izbiranje vozlišča



Slika 3.2: Dokler se ne pride do vozlišča, ki je list, se opravlja korake izbiranja. Na vsakem koraku se izbere glede na rezultat izbirne funkcije.

Korak izbiranja od korenskega vozlišča rekurzivno kliče izbirno funkcijo, ki izbere naslednje vozlišče glede na izbirno strategijo. Ustavi se ko pride do vozlišča, ki je list drevesa. Izbirna strategija nadzoruje ravnotežje med izkoriščevanjem in raziskovanjem. Na eni strani je dobro izbrati vozlišče, ki ima

dosedaj najboljše rezultate (izkoriščanje), po drugi strani pa morajo biti zaradi negotovosti evaluacije preizkušene tudi trenutno manj obetajoče poteze (raziskovanje)[1].

- **UCT.** Kot navaja Chaslot[1], predlagata Kocsis in Szepesvári[3] strategijo UCT (Upper Confidence bounds applied to Trees). Ta strategija je enostavna za implementacijo, in uporabljena v veliko programih. Deluje na sledeč način: Naj bo I množica vseh vozlišč dosegljivih iz trenutnega vozlišča p. UCT izbere otroka k vozlišča p, ki zadovolji formulo:

$$k \in argmax_{i \in I} \left(v_i + C \times \sqrt{\frac{ln(n_p)}{n_i}} \right) \tag{3.1}$$

  Pri čemer je v_i vrednost vozlišča i, n_p število obiskov starša vozlišča i, n_i število obiskov vozlišča i, C pa je konstanta, ki določa ravnotežje med raziskovanjem in izkoriščanjem.

- **UCB1-TUNED.** Gelly in Wang[9] predlagata uporabo UCT variante UCB1-TUNED. UCB1-TUNED izbere otroka k vozlišča p, ki zadovolji formulo:

$$k \in argmax_{i \in I} \left(v_i + C \times \sqrt{\frac{ln(n_p)}{n_i} \times min\{\frac{1}{4}, V_i(n_i)\}} \right) \tag{3.2}$$

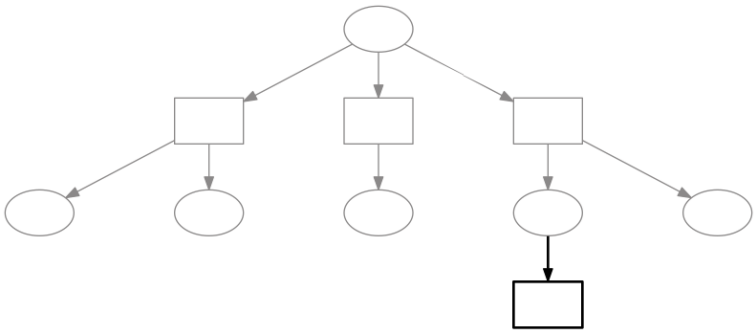
  kjer je

$$V_i(n_i) = (\frac{1}{n_i} \sum_{t=1}^{n_i} R_{i,t,j}^2 - v_i^2 + \sqrt{\frac{2ln(n_p)}{n_i}}) \tag{3.3}$$

  ocena zgornje meje variance v_i, $R_{i,t,j}$ je t^{ti} rezultat dobljen na vozlišču i za igralca j, v_i je vrednost vozlišča i, n_i je število obiskov vozlišča i, n_p število obiskov vozlišča p, C pa je konstanta, ki določa ravnotežje med raziskovanjem in izkoriščanjem.

Pomembna opazka na vse predstavljene izbirne strategije je, da so neodvisne od igre, in da ne uporabljajo domenskega znanja[1].

3.3.4 Razvijanje vozlišča



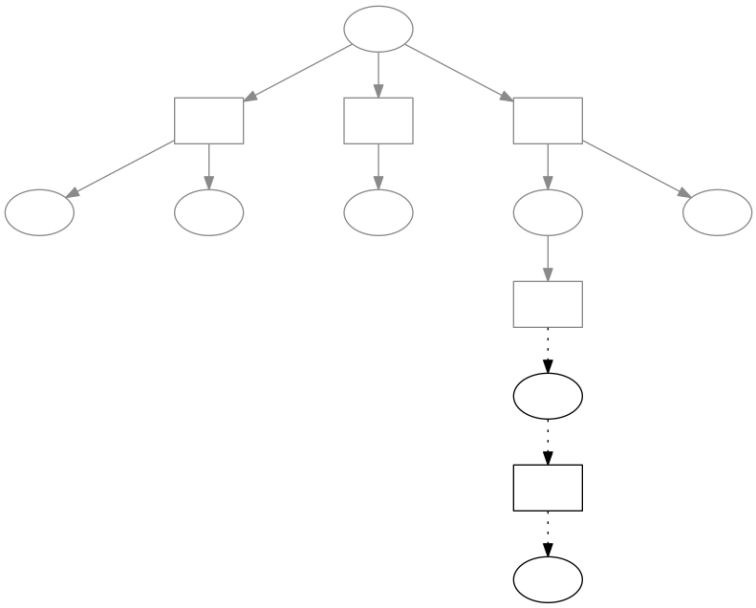
Slika 3.3: Po izboru vozlišča, se to razširi, v skladu s *strategijo razvijanja*.

Korak razvijanja doda vozlišče MCTS drevesu. Najbolj popularna strategija razvijanja je:

- Eno vozlišče je dodano za vsako simulirano igro. Vsako vozlišče ustreza prvi poziciji, ki še ni bila shranjena pri obhodu drevesa.

Ta strategija je preprosta in enostavna za implementacijo. Obstajajo še druge strategije, v splošnem pa je vpliv teh strategij na igranje majhen, strategija kreiranja enega vozlišča na eno simulacijo v večini primerov zadostuje[1].

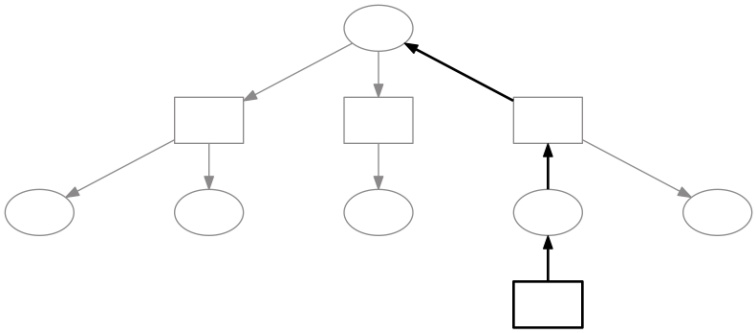
3.3.5 Simulacija



Slika 3.4: Po razvoju vozlišča, se simulira igra od trenutnega stanja do konca igre.

Simulacija izvrši igranje igre od trenutnega stanja do konca igre. Poteze vseh igralcev so izbrane v skladu s simulacijsko strategijo. Ta je lahko naključna, kjer so poteze vseh igralcev naključno izbrane (v skladu z pravili igre), lahko pa vsebuje tudi nekaj domenskega znanja o igri, in potez ne izbiramo naključno. Uporaba dobre simulacijske strategije se je izkazala za zelo pomembno pri kvaliteti igranja. Glavna ideja je, da igramo bolj zanimive poteze glede na neko hevristično znanje, ki je specifično za posamezno igro[1].

3.3.6 Vzvratne vrednosti



Slika 3.5: Po odigrani simulaciji in dobljenem rezultatu, se določi vzvratne vrednosti.

Rezultat simulacije določi vzvratne vrednosti vozliščem, ki smo jih obiskali med izbiranjem. Na primer, pri igri go je rezultat lahko štet pozitivno ($R_k = +1$) če je igra zmagana, in negativno ($R_k = -1$), če je igra izgubljena. Izenačenja so šteta kot $R_k = 0$. Pri taroku s tremi igralci je rezultat število točk igralca, ki je začel igro. Ta rezultat potem določi vzvratne vrednosti, pri čemer vozliščem, ki so od igralca, ki je začel igro ta rezultat prištejemo, vozliščem preostalih dveh igralcev pa prištejemo negiran rezultat. Strategija določanja vzvratnih vrednosti izračuna vrednost v_L vozlišča L. Najbolj popularna strategija določanja vzvratnih vrednosti je povprečje, ki izračuna povprečje rezultatov vseh simuliranih iger skozi to vozlišče, se pravi $v_L = (\sum_k R_k)/n_L$, pri čemer je R_k vrednost simulacije igre vozlišča L in n_L število obiskov vozlišča L. Obstajajo še druge vrste strategij določanja vzvratnih vrednosti[1]:

- **Max**. Strategija max določi vzvratne vrednosti na negamax način. Vrednost vozlišča p je največja vrednost njegovih otrok. Izkaže se, da max kot strategija določanja vzvratnih vrednosti pri MCTSju ne da dobrih rezultatov. Ko je število otrok vozlišča veliko in število simulacij majhno, vrednosti otrok niso zanesljive, zato namesto, da bi izbrali

najboljšega otroka, je preprosto izbran najbolj srečen otrok.

- **Mix** Coulom[5] kot alternativo maxu predlaga mix. Ta izračuna vrednost starša vozlišča p po enačbi:

$$v_p = \frac{v_{mean} \cdot w_{mean} + v_r \cdot n_r}{w_{mean} + n_r} \tag{3.4}$$

kjer je v_r vrednost poteze z največjim številom simulacij, n_r število odigranih simulacij. v_{mean} je povprečje vrednosti otrok vozlišča in w_{mean} teža tega povprečja. Ta strategija je bila uporabljena v programu *Crazy Stone*, ki je zmagal 9x9 Go turnir na Računalniški olimpijadi leta 2006[13]. Nova verzija tega programa pa uporablja kar povprečje, ki daje boljše rezultate.

3.3.7 Končna izbira poteze

Ko smo končali z dodajanjem novih vozlišč drevesu, je potrebno glede na rezultate simulacij izbrati katero potezo bomo odigrali. Za odločitev kateri otrok korenskega vozlišča (torej poteza, ki jo bomo odigrali) je najboljši, poznamo več načinov[1].

- **Max otrok**. Izberemo otroka z najvišjo vrednostjo.
- **Robusten otrok**. Izberemo otroka, ki je bil največkrat obiskan.
- **Max-robusten otrok**. Izberemo otroka, ki ima tako najvišjo vrednost, hkrati pa je bil tudi največkrat obiskan. Če tak ne obstaja je odigranih še več simulacij, dokler tak otrok ne obstaja.

3.4 Prednosti

MCTS ima v primerjavi s klasičnimi metodami drevesnega preiskovanja številne prednosti[15].

- **Ni hevrističen**. MCTS ne zahteva strateškega ali taktičnega znanja o domeni. Algoritem lahko učinkovito deluje zgolj z podatki o veljavnih potezah in končnih pogojih. Prav zaradi te lastnosti bi MCTS lahko imel potencial tudi za splošno računalniško igranje iger.

- **Asimetričen**. Drevo raste asimetrično, prilagaja se topologiji iskalnega prostora. Bolj zanimiva vozlišča so obiskana večkrat, algoritem porabi več časa v bolj relevantnih delih drevesa. Ta lastnost naredi MCTS primeren za igre z velikim faktorjem vejitve, kot je na primer go.

- **Poljubnost časa izvajanja**. Iskanje lahko kadarkoli ustavimo in dobimo trenutni približek za najboljšo potezo.

- **Eleganten**. Algoritem je enostaven za razumevanje in implementacijo.

3.5 Slabosti

Algoritem ima malo slabosti, so pa te lahko precej resne[15]:

- **Kakovost igranja**. Algoritmu (vsaj v osnovni obliki) celo pri srednje kompleksnih igrah lahko ne uspe najti dobrih potez v razumnem času. To je predvsem zaradi izjemne velikosti kombinatoričnega prostora možnih potez in dejstva, da ključna vozlišča niso bila obiskana dovoljkrat, da bi dali bolj zanesljivo oceno.

- **Hitrost**. Iskanje lahko vzame zelo veliko število ponovitev, kar je lahko problematično za bolj splošne aplikacije, ki so težavne za optimizacijo. Najboljše implementacije za igro go tako zahtevajo na miljone simulacij v povezavi z domensko specifičnimi optimizacijami in izboljšavami.
  Na srečo je algoritem z več tehnikami moč znatno optimizirati.

3.6 Izboljšave

Predlogov za izboljšavo algoritma je veliko. Lahko jih delimo na take, ki zahtevajo specifično znanje o domeni, in na take, ki tega ne zahtevajo[15].

- **Domensko znanje**. Specifično znanje o igri lahko izkoristimo tako, da že v drevesu izločimo slabše poteze (ali pa jih vsaj manj obravnavamo), ali pa ko simuliramo igro do konca favoriziramo določene poteze, za katere vemo, da so boljše. V tem primeru bodo simulacije igre do konca bolj realistične v primerjavi s popolnoma naključnimi simulacijami, in bo potrebno manj ponovitev za doseg bolj realističnih vrednosti.
- **Domensko neodvisne**. Neodvisne izboljšave lahko uporabimo povsod, ker niso vezane na točno določeno domeno. Večinoma gre za izboljšave na nivoju drevesa.

Poglavje 4

Tarok

Med najbolj razširjene in priljubljene družbene igre s kartami na Slovenskem gotovo sodi tarok. V več stoletnem razvoju se je igra tarok spreminjala in dobivala v posameznih deželah posebne načine oziroma različice. Od 14. stoletja, ko naj bi se po dosedanjih raziskavah s tarokom srečali v Evropi, lahko govorimo o tej igri tudi na slovenskem etničnem ozemlju, čeprav so nam za starejša obdobja na voljo le skromna pričevanja. Ves ta čas so poleg uveljavljenih, pogosto mednarodno sprejetih motivih zasnov nastajale tudi razne vrste posebnih tarokov, od unikatov do motivno opredeljenih v posebne tematike (motivi iz spolnega življenja, politično opredeljena motivika, humoristična, vojaška itd.). Posebna zvrst so tudi takoimenovani narodni taroki, ki z izborom motivov predstavljajo posamezne narode, njihovo kulturo, dediščino, posebnosti in znamenitosti, simbole itd. Mednje sodi tudi Slovenski tarok[16].

4.1 Pravila Taroka

4.1.1 Splošno

Karte za igranje taroka se razlikujejo od drugih kart. Komplet vsebuje 54 kart. 32 je barvnih: za srce, karo, pik in križ je po 8 kart, poleg tega je še 22 tarokov, opremljenih z rimskimi številkami od 1 do 21.

Rdeče barvne karte (karo in srce) imajo po 8 kart, ki si sledijo po vrednosti: kralj, dama, konj (imenovan tudi kaval), fant, as, dvojka, trojka in štirica. Zadnjim štirim kartam rečejo v kartaškem žargonu platelci. Tudi črne barvne karte(križ in pik) imajo po osem kart, ki si sledijo po vrednosti: kralj, dama, konj (kaval), fant, temu sledijo še desetka, devetka, osmica in sedmica, štiri prazne karte ali platelci.

Vsaka karta ima tudi svojo vrednost:

Karta	**Vrednost**
Kralj	5
Dama	4
Konj	3
Fant	2

Tabela 4.1: Tabela prikazuje vrednosti barvnih kart

Ostale barvne karte in taroki so brez vrednosti, z izjemo treh posebnih tarokov - škisa (tarok 22), monda (tarok 21) ter pagata (tarok 1). Ti trije so tudi vredni vsak po pet točk, skupaj pa jih imenujemo trula. Po koncu igre se prešteje karte igralcev. Šteje se tako, da se karte jemlje po tri, vrednosti posameznih kart pa se seštejejo. Od skupnega seštevka vrednosti treh kart se odšteje 2, če imajo vse tri karte vrednost vsaj 2, 1 če imata vsaj 2 karti vrednost vsaj 2, ali 0 če ima le ena karta vrednost vsaj 2. V primeru, da so vse tri karte brez vrednosti, imajo skupaj vrednost 1. Če na koncu štetja ostane ena karta ali dve, se odšteje ena. Za zmago je treba zbrati vsaj 35 točk.

4.1.2 Tarok za tri

Vsak igralec dobi 16 kart, preostalih 6 pa gre v talon. Karte v talonu so znane šele po koncu licitacije. Po deljenju se začne licitacija. Najprej prvi igralec napove svojo igro, nato drugi, in na koncu še tretji. Vsak naslednji

Igra	**Vrednost**
Klop	/
Tri	10
Dva	20
Ena	30
Berač	70

Tabela 4.2: Tabela prikazuje vrednosti posameznih iger

igralec lahko napove le višjo igro od prejšnjega. V primeru, da ne bo igral, reče dalje. Vkolikor vsi trije igralci ne igrajo, se igra klopa. Pri licitaciji zmaga igralec, ki napove najvišjo igro. Pri igri s tremi igralci je to igralec, ki igra proti ostalima dvema igralcema[10].

Po koncu licitiranja se skupaj razvrsti po tri, dve ali eno (odvisno od tipa igre) karto talona, ter se jih obrne. Igralec vzame eno, dve ali tri izbrane karte (odvisno za katero igro je licitral). Prav toliko kart tudi založi - odvzame iz svojega kupa kart ter jih položi na mizo. To stori tako, da ostali igralci ne vidijo, katere karte je igralec založil.

Prvi vrže karto igralec, ki je zmagal pri licitaciji. Igralec, ki začne rundo lahko vrže katerokoli karto, ki jo ima. Ostali igralci morajo vreči karto enake vrste (enake barve, če je barva, oziroma tarok, če je tarok). V primeru, da je nimajo, lahko vržejo tarok. V primeru, da tudi taroka nimajo, lahko vržejo katerokoli karto. Če so na mizi le barvne karte, pobere tista z največjo vrednostjo barve, ki je začela rundo. Vkolikor je na mizi tudi kakšen tarok, pobere najvišji tarok. Igralec, ki je pobral zadnjo rundo tudi začne naslednjo. To se ponavlja, dokler imajo vsi igralci še karte[10]. Če je igralec, ki je igral sam imel vsaj 35 točk, je igro dobil.

4.2 Obstoječi pristopi računalniškega igranja taroka

Prejšnje delo na računalniškem igranju taroka temelji predvsem na algoritmu min-max.

4.2.1 Silicijasti tarokist 1.0

Mitja Luštrek leta 2002 izdela program za igranje taroka Silicijasti tarokist[10], uporablja pa algoritem min-max s številnimi izboljšavami - alfa-beta iskanje, transpozicijska tabela, particijsko iskanje, zgodovinska hevristika ter še nekaj drugih specifičnih hevristik. Z omenjenimi izboljšavami mu uspe število vozlišč zmanjšati iz 1.598.924 na 8.667, kar znatno pripomore k hitrosti algoritma. Nepopolno informacijo je modeliral z Monte-Carlo vzorčenjem. Človeški igralci program ocenjujejo kot spodobnega, čeprav ne ravno vrhunskega nasprotnika[10].

4.2.2 Silicijasti tarokist 2.0

Leta 2003 Luštrek izdela še izboljšano verzijo Silcijastega tarokista[11], ki poleg prejšnjih izboljšav uporablja še iterativno poglabljanje. Število vozlišč mu uspe zmanjšati na 2.470. Rezultati naj bi bili podobni kot prej.

4.2.3 Licitacija

Na področju licitacije je bila leta 2006 izvedena raziskava[7], kjer avtorji primerjajo pristopa licitacije s simulacijo večih iger ter s pomočjo Bayesovih mrež pri igri taroka s štirimi igralci. Rezultati licitacije z Bayesovimi mreži so se izkazali za znatno boljše, hkrati pa je tovrsten pristop tudi časovno bolj učinkovit, saj simulacija velikega števila iger traja dolgo časa.

Poglavje 5

Implementacija MCTS za igro tarok

5.1 Izbira poteze in modeliranje nepopolne informacije

MCTS za gradnjo drevesa potrebuje popolno informacijo o stanju igre – za gradnjo drevesa bi torej morali vedeti, kakšne karte ima naš nasprotnik, kar pa pri taroku glede na pravila igre seveda ni mogoče. Ta problem smo rešili s še eno plastjo Monte-Carlo vzorčenja. Preden začnemo graditi drevo, vsem igralcem naključno porazdelimo karte, in na ta način preidemo v igro popolne informacije. Nato začnemo z običajno gradnjo drevesa. Takih porazdelitev nasprotnikovih kart naredimo čimveč, in za vsako porazdelitev zgradimo drevo. Po koncu gradnje vseh dreves, izberemo tisto potezo, ki se je največkrat izkazala za najboljšo.

Algoritem 2 karta MCTS(seznamKart) {

```
poteza ← ∅
for i = 0 →steviloDreves do
  MCTSdrevo ←generirajMCTSDrevo(stSimulacij, seznamKart)
  for v ∈otroci(MCTSDrevo.koren) do
    poteza[v] ←poteza[v] + v.stObiskov
  end for
end for
return izberiNajboljsoPotezo(poteza)
}
```

5.2 Gradnja MCTS drevesa

Ob začetku gradnje imamo le vozlišče, ki določa igralca, ki začne igro. Začnemo z rekurzivnim izvajanjem štirih korakov algoritma MCTS in postopnim grajenjem drevesa.

Spodnji algoritem kliče funkcijo *korakMCTS* dokler drevo nima zahtevanega števila vozlišč (simulacij). Vsak klic funkcije *korakMCTS* drevesu doda eno vozlišče.

Algoritem 3 MCTSDrevo generirajMCTSDrevo(stSimulacij, igra) {

```
drevo ←MCTSDrevo(igra)
for i = 0 →stSimulacij do
  drevo ←MCTSDrevo.korakMCTS()
end for
return drevo
}
```

Funkcija *korakMCTS* postopoma, na vsaki globini drevesa izbere vozlišče, dokler ne naleti na vozlišče, ki je list. Takrat se izbere primerno karto glede na trenutno stanje igre (glede na to, kateri igralec je na vrsti in katere karte so že na mizi - v skladu s pravili igre), ki je vsebovano v vozlišču. Zatem

se vozišče razvije, igro simulira do konca in ta rezultat posodobi vzvratne vrednosti po vseh vozliščih, ki smo jih obiskali med izbiranjem.

Algoritem 4 void korakMCTS() {

```
seznamVozlisc ← ∅
while (!jeList(vozlisce)) do
  vozlisce ← izberiVozlisce(vozlisce)
  seznamVozlisc.dodaj(vozlisce)
end while
karta ← izberiKarto(vozlisce)
novoVozlisce ← razvijVozlisce(vozlisce, karta)
rezultat ← simulirajVozlisce(vozlisce)
igralec ← vozlisce.igralec
for vozlisce ∈seznamVozlisc do
  posodobiVrednost(rezultat, igralec, vozlisce)
end for
}
```

5.2.1 Izbiranje

Za izbiranje najbolj obetavnega vozlišča se uporablja standardna UCT funkcija.

Algoritem 5 vozlisce izberiVozlisce(vozlisce) {

```
najvecjaVrednost ← 0
izbranoVozlisce ← ∅
for v ∈otroci(vozlisce) do
  if UCTVrednost(v) >najvecjaVrednost then
    najvecjaVrednost ← UCTVrednost(v)
    izbranoVozlisce ← v
  end if
end for
return izbranoVozlisce
}
```

Algoritem med otroki trenutnega vozlišča izbere tistega, ki ima največjo UCT vrednost, ki jo vrne funkcija *UCTVrednost*.

5.2.2 Razvijanje vozlišča

Pri razvijanju uporabljamo najbolj popularno strategijo, eno vozlišče na eno simulirano igro.

Algoritem 6 vozlisce razvijVozlisce(karta,vozlisce) {

```
stPoteze ← naslednjaStevilkaPoteze(vozlisce)
stRunde ← naslednjeSteviloRunde(vozlisce)
naslednjiIgralec ← naslednjiIgralec(vozlisce)
vozlisce.karte.dodaj(karta)
novoVozlisce ← kreirajNovoVozlisce(stPoteze, )
novoVozlisce.karte.izbrisi(karta)
return izbranoVozlisce
}
```

5.2.3 Simulacija

Simulacija igre od stanja v katerem se nahaja v vozlišču, kjer pričnemo s simulacijo, do konca igre, je naključna.

Algoritem 7 int simulirajVozlisce(vozlisce) {

```
kartNaIgralca ← 48/STEVILO_IGRALCEV
vseKarte ← vozlisce.karte
for i = 0 →kartNaIgralca - stRunde + 1 do
  odigrajEnoRundo(vseKarte, vozlisce)
  vrednost ←vrednost + vrednostZadnjeRunde(vseKarte)
end for
return vrednost
}
```

Funkcija *odigrajEnoRundo* vrže po eno karto za vsakega igralca, do zaključka runde. Karte tudi doda na seznam *vseKarte*. Odigramo toliko rund, kot jih je še ostalo do konca igre, nato izračunamo vrednost igre glede na seznam odigranih kart v polju *vseKarte*, in izračunano vrednost vrnemo kot izhod.

5.2.4 Vzvratne vrednosti

Z vrednostjo dobljeno pri simulaciji, določimo vzvratne vrednosti nazaj po obiskanih vozliščih. Vozliščem soigralcev igralca za katerega smo dobili rezultat pri simulaciji vrednost prištejemo, nasprotnikom pa vrednost odštejemo.

Algoritem 8 void posodobiVrednost(rezultat, igralec, vozlisce) {

```
if jeSoigralec(igralec, vozlisce) then
  vozlisce.vrednost ←vozlisce.vrednost + rezultat
else
  vozlisce.vrednost ←vozlisce.vrednost - rezultat
end if
}
```

5.2.5 Izbira poteze

Po generiranju vseh dreves imamo v razpršeni tabeli *poteza*, ki je tudi vhod v funkcijo *izberiNajboljsoPotezo* sešteto število obiskov vozlišč vseh možnih potez v vseh drevesih. Izberemo tisto potezo, ki ima skupno največje število obiskov.

Algoritem 9 karta izberiNajboljsoPotezo(poteza) {

```
najvecjaVrednost ← −∞
najboljsaKarta ← ∅
for v ∈poteza do
  if v.vrednost >najvecjaVrednost then
    najvecjaVrednost ← v.vrednost
    najboljsaKarta ← v.karte[v.karte.size − 1]
  end if
end for
return najboljsaKarta
}
```

5.3 Licitacija

Pri licitaciji smo odločili kar za preprosto hevristično funkcijo, ki glede na število tarokov ter vrednosti kart določi kako bo igralec licitiral.

- Če imaš vsaj 6 tarokov, vsaj enega kralja, in eno karto iz trule ali če imaš 10 tarokov, igraj tri.
- Če imaš vsaj 7 tarokov, vsaj dva kralja, in eno karto iz trule ali če imaš 10 tarokov, igraj dva.
- Če imaš vsaj 8 tarokov, vsaj dva kralja, in eno karto iz trule ali če imaš 10 tarokov, igraj ena.
- Drugače prepusti igro - naprej

Taka licitacija je zelo približna, saj ne upošteva niti višine tarokov. Vsekakor bi bilo smiselno implementirati ali rešitev z Bayesovimi mrežami, ki se je izkazala za učinkovito[7], ali pa poskusiti s simulacijo temelječo kar na algoritmu drevesnega preiskovanja Monte Carlo (vendar pa bi bilo to časovno zelo zahtevno).

5.4 Izbiranje kart iz talona in zalaganje

Tudi tukaj smo vključili hevristično znanje. Izbrana je tista karta, oziroma tisti par ali trojček kart, ki pri zalaganju omogoča, da se znebimo čimveč barv. V primeru enakosti izberemo tiste z večjim številom tarokov in vrednosti karti.

Poglavje 6

Rezultati

V tem poglavju predstavimo rezultate implementacije algoritma MCTS za tarok. Najprej primerjamo rezultate algoritma pri igri taroka s popolno informacijo (torej, ko vsi igralci poznajo karte vseh ostalih igralcev), nato pa še rezultate pri klasični igri taroka z nepopolno informacijo, kjer nasprotnikove karte modeliramo s pomočjo metode Monte-Carlo. Na koncu predstavimo še rezultate igranja v primerjavi s človekom. Rezultati so vedno od igralca, ki je zmagal pri licitiranju. Za izbiranje je bila vselej uporabljena funkcija UCT, strategija določanja vzvratnih vrednosti je zmeraj uporabljala metodo povprečja, končna poteza pa je bila vedno izbrana glede na največje število obiskov otroka korenskega vozlišča.

6.1 Rezultati pri igri popolne informacije

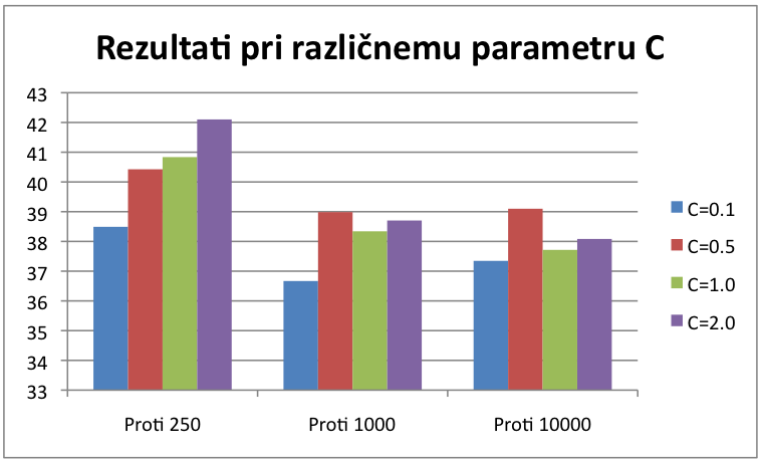
Simulacije v posameznih vozliščih so bile odigrane naključno. Igralec, ki zmaga v licitaciji, igra sam proti ostalima dvema igralcema. Število točk tega igralca primerjamo pri različnih parametrih. Za vsak parameter eksperimenta odigramo 100 partij.

6.1.1 Rezultati pri različnih vrednostih parametra C

Zanima nas kako parameter C vpliva na kvaliteto igranja pri enakem številu simulacij proti trem nasprotnikom z različnim številom simulacij in konstantim parametrom C. Konstanta C določa ravnotežje med raziskovanjem in izkoriščanjem. Partije so bile odigrane z igralcem s 5000 simulacijami. Parameter C se je spreminjal, medtem ko se je pri nasprotnikih spreminjalo število simulacij, C pa je bil konstanten, 0.5. Proti igralcemu z 250 simulacijami je razlika med rezultati precejšnja. Najbolje se je odrezal igralec s C = 2.0. Proti igralcu s 1000 simulacijami razlike niso bile tako velike, le C = 0.1 je bil spet precej slabši. Proti igralcu z 10000 simulacijami pa je bil najboljši igralec s C = 0.5.

C	Rezultat proti 250	Rezultat proti 1000	Rezultat proti 10000
0.1	38.49	36.66	37.34
0.5	40.42	38.98	39.09
1.0	40.83	38.34	37.71
2.0	42.12	38.70	38.08

Tabela 6.1: Tabela rezultatov pri različnih vrednostih parametra C



Slika 6.1: Graf rezultatov pri različnih vrednostih parametra C

Parametra C, ki bi se v vseh primerih izkazal za najboljšega torej nismo našli, je pa C = 0.5 najboljši tako proti igralcema s 1000 kot tudi igralcema z 10000 simulacijami na potezo.

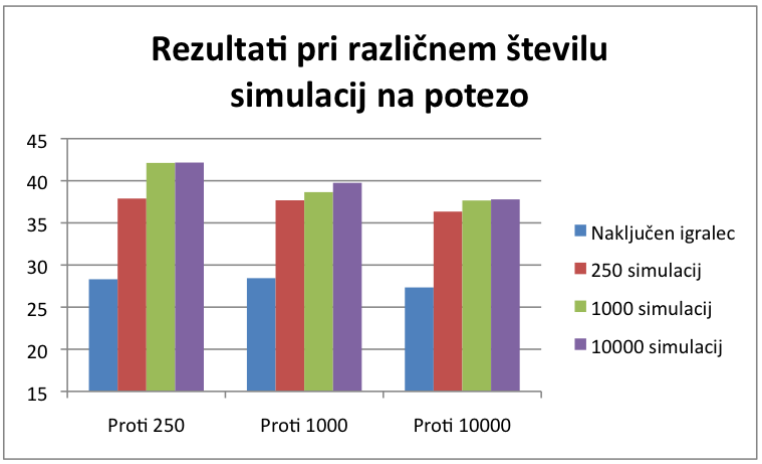
6.1.2 Rezultati pri različnih vrednostih števila simulacij za potezo

V tem eksperimentu preizkusimo kako število simulacij za posamezno potezo vpliva na kvaliteto igranja.

Število simulacij	**Proti 250**	**Proti 1000**	**Proti 10000**
Naključen	28.31	28.44	27.34
250	37.89	37.68	36.34
1000	42.12	38.65	37.66
10000	42.16	39.75	37.79

Tabela 6.2: Tabela rezultatov pri različnemu številu simulacij za potezo

Naključen igralec je proti vsem nasprotnikom večijo partij izgubil. Že igralec z 250 simulacijami na potezo je v povprečju zmagoval, prav tako pa tudi igralca s 1000 in 10000 simulacijami na potezo. Največja razlika med igralci je proti igralcema z številom simulacij 250. Opaziti je, da se razlika v številu točk med igralci z različnim številom simulacij manjša, ko igramo proti igralcema z večjim številom potez.



Slika 6.2: Graf rezultatov pri različnemu številu simulacij

6.1.3 Rezultati pri hevrističnih simulacijah

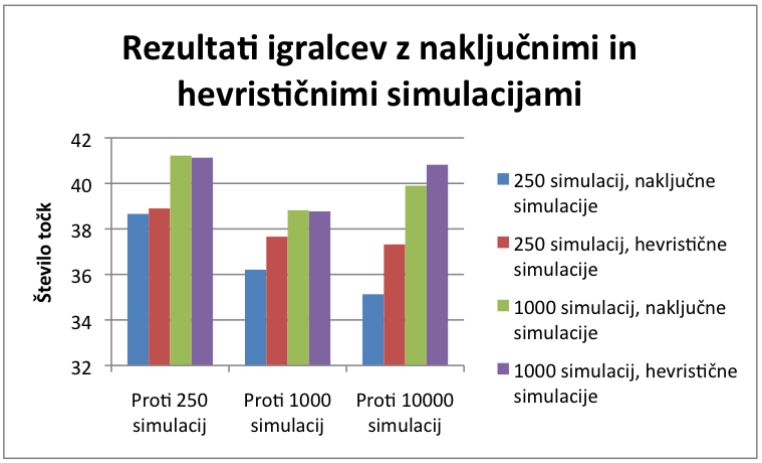
Pri simulaciji vozlišč smo v tem primeru dodali nekaj preprostih pravil.

- Če smo zadnji na potezi, in sta oba druga igralca dala barvno karto, mi pa moramo dati tarok, vrzi najmanjši tarok.
- Če smo zadnji na potezi in moramo vreči tarok, vrzi ali najmanjšega, ali pa najmanjšega, ki še pobere.
- Če je nekdo vrgel monda, in imamo škisa, ga vedno poberi.

Število simulacij in tip simulacij	**Proti 250**	**Proti 1000**	**Proti 10000**
250, naključne simulacije	38.66	36.21	35.13
250, hevristične simulacije	38.9	37.66	37.32
10000, naključne simulacije	41.22	38.82	39.9
10000, hevristične simulacije	41.13	38.77	40.82

Tabela 6.3: Tabela rezultatov pri različnemu številu in tipu simulacij

Opaziti je nekoliko boljše rezultate pri številu simulacij 250. Proti igralcema z 1000 simulacij na potezo je igralec z hevrističnimi simulacijami dobil 1.45 točke več. Pri igralcu z 10000 simulacijami na potezo ni opaziti razlike med naključnimi in hevrističnimi simulacijami. Rezultat je pričakovan, pri manjšem številu simulacij nam hevristične simulacije pomagajo, da nekoliko hitreje konvergiramo k rezultatu, medtem ko se pri večjem številu to ne pozna.



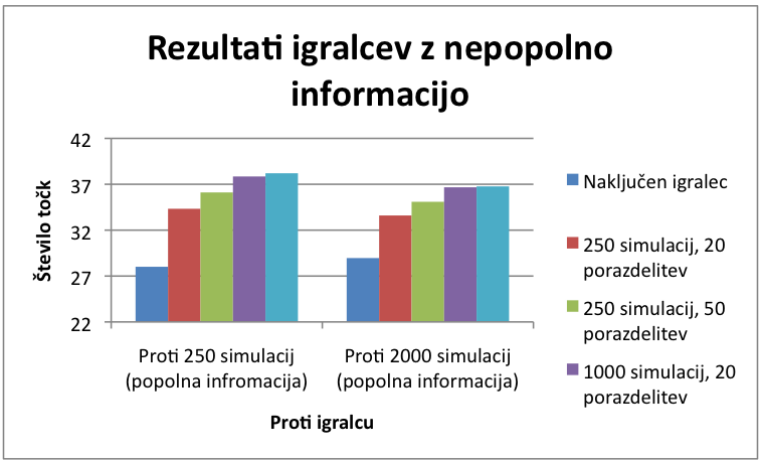
Slika 6.3: Graf rezultatov pri različnemu številu in tipih simulacij

6.2 Rezultati pri igri nepopolne informacije

V tem primeru smo izvajali eksperimente pri katerih naš igralec ni poznal nasprotnikovih kart. Nepoznavanje nasprotnikovih kart modeliramo s še eno plastjo Monte-Carlo vzorčenja. Nasprotnika igrata kar igro popolne informacije, s številom simulacij na 250 in 2000 ter parametrom C 0.5.

Število simulacij in porazdelitev	**Proti 250**	**Proti 2000**
Naključni igralec	28.01	28.96
250, 20 porazdelitev	34.33	33.6
250, 50 porazdelitev	36.11	35.11
1000, 20 porazdelitev	37.85	36.67
1000, 50 porazdelitev	38.20	36.77

Tabela 6.4: Tabela rezultatov pri različnemu številu simulacij ter porazdelitev



Slika 6.4: Graf rezultatov igralcev z nepopolno informacijo z različnim številom simulacij in porazdelitev

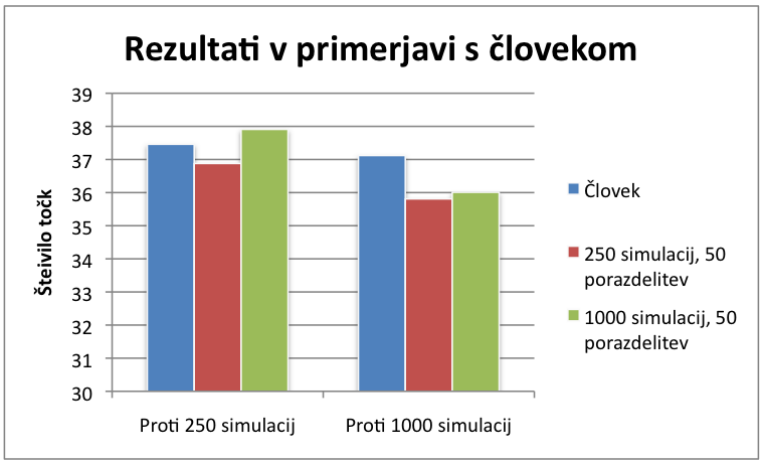
Vsi igralci so znatno boljši od naključnega, in razen tistega z 250 simulacijami in 20 porazdelitvami v povprečju zmagujejo proti obema nasprotnikoma.

6.3 Rezultati proti človeku

Zanimalo nas je tudi, kako dobro se program obnese v praksi - je konkurenčen človeškemu nasprotniku? Odigral sem 25 partij. Vse so bile odigrane proti dvema nasprotnikoma s popolno infromacijo, številom simulacij na 250 in 5000, in parametrom C 0.5. Istega igralca je nato odigral še program z 250 simulacijami na potezo in 50 porazdelitvami kart, ter z 1000 simulacijami na potezo in 50 porazdelitvami, proti istima dvema nasprotnikoma.

Število simulacij in porazdelitev	**Proti 250**	**Proti 2000**
Človek	37.46	37.12
250 simulacij, 50 porazdelitev	36.88	35.81
1000 simulacij, 50 porazdelitev	37.91	36.01

Tabela 6.5: Tabela rezultatov v primerjavi s človekom



Slika 6.5: Graf rezultatov v primerjavi s človekom

Izkaže se, da je algoritem MCTS z 250 simulacijami in 50 porazdelitvami malo slabši od človeka, medtem ko je tak z 1000 simulacijami in 50 porazdelitvami proti nasprotnikoma z 250 simulacijami boljši od človeka, proti nasprotnikoma z 2000 simulacijami pa slabši. Rezultati kažejo, da se algoritem MCTS lahko enakovredno kosa z priložnostnim človeškim igralcem taroka, v določenih primerih pa igra še bolje.

6.4 Primer odigrane partije

Igralec 1 je igralec z nepopolno informacijo igre, 250 simulacijami na potezo in 50 porazdelitvami možnih nasprotnikovih kart. Igralca 2 in 3 sta igralca s popolno informacijo igre z 250 simulacijami na potezo. Licitacijo je zmagal igralec 1. Po zalaganju je stanje sledeče:
Igralec 1: T22, T21, T20, T19, T18, T15, T14, T11, T7, T6, T5, T2, ♡K, ♡C, ♢K, ♠8.
Igralec 2: T17, T13, T4, T3, ♡D, ♡4, ♡3, ♡1, ♢J, ♢3, ♠C, ♠J, ♠10, ♣K, ♣8, ♣7
Igralec 3: T12, T10, T9, T8, T1, ♡J, ♡2, ♢1, ♢2, ♢4, ♣D, ♣C, ♣9, ♠7, ♠J, ♠9

Igro začne igralec 1. Glede na to, da ima veliko število visokih tarokov, nima pa pagata, ki je vreden 5 točk, se mu splača začeti z visokimi taroki ter upati na to, da bo s tem izsilil pagata.

Igralec 1: T22, igralec 2: T3, igralec 3: T9.
Igralec 1: ♡K, igralec 2: ♡4, igralec 3: ♡2.
Igralec 1: ♢K, igralec 2: ♢3, igralec 3: ♢1.
Igralec 1: T21, igralec 2: T4, igralec 3: T3: T8
Igralec 1: T20, igralec 2: T13, igralec 3: T10
Igralec 1: T18, igralec 2: T17, igralec 3: T12
Igralec 1: T15, igralec 2: ♡3, igralec 3: T1
Igralec 1: T14, igralec 2: ♠10, igralec 3: ♢4
Igralec 1: T7, igralec 2: ♡1, igralec 3: ♣C
Igralec 1: T19, igralec 2: ♠C, igralec 3: ♡J
Igralec 1: T6, igralec 2: ♢J, igralec 3: ♠7
Igralec 1: T11, igralec 2: ♣8, igralec 3: ♢2
Igralec 1: T2, igralec 2: ♣7, igralec 3: ♠9
Igralec 1: T5, igralec 2: ♣K, igralec 3: ♣9
Igralec 1: ♡C, igralec 2: ♡D, igralec 3: ♣D
Igralec 2: ♠K, igralec 3: ♠J, igralec 1: ♠8

Igralcu 1 tudi uspe izsiliti pagata, kar glede na veliko število visokih tarokov, tudi ni presenetljivo. Do predzadnje runde igralec 1 pobere vse karte. Predzadnjo rundo pobere igralec 2, s srčevo damo. Temu se, ker sta igralec 2 in 3 odigrala pravilno, in pustila prave karte za konec, tudi ne more izogniti. V zadnji rundi igralec 2 pobere s pikovim kraljem, to pa sta tudi edini dve rundi, ki sta jih pobrala igralec 2 in 3, in ki sta igralca 1 ločila od uspešne izvedbe valata.

Poglavje 7

Zaključek

7.1 Sklepne ugotovitve

Rezultati so dobri že pri majhnem številu simulacij (250), ter brez kakršnegakoli hevrističnega znanja, in kažejo zmožnost uporabe algoritma drevesnega preiskovanja tudi na področju iger s kartami, ter potrjujejo njegov potencial za splošno igranje iger. Naša hipoteza, da bo algoritem MCTS pri nizkem številu simulacij dal šibkega igralca se ne izkaže za povsem točno, saj že ob številu simulacij 250 igra precej dobro. Tudi druga hipoteza, da se bo z uporabo hevristik kvaliteta igranja znatno povečala se ni izkazala za resnično. Uporaba hevristik ni znatno vplivala na rezultate in povečala kvaliteto igranja, je pa dejstvo, da smo vključili le nekaj preprostih hevristik, in prav zavoljo tega bi vredno poskusiti še z vključevanjem še več domenskega znanja. Algoritem je upravičil pričakovanja in predstavlja smiselno rešitev za problem računalniškega igranja taroka.

7.2 Bodoče delo

Idej in možnosti za nadaljnje delo je veliko. Naj omenim le nekatere:

- **Optimizacija algoritma.** Z uporabo tehnik transpozicijske tabele in zgodovinske hevristike bi lahko precej zmanjšali iskalni prostor ter

posledično tudi število vozlišč. Na ta račun bi se zmanjšalo tudi število simulacij, kar bi pohitrilo algoritem.

- **Paralelizacija** Algoritem MCTS je precej enostavno paralelizirati. Paralelizacija je možna na nivoju lista, na nivoju korena, ali na nivoju drevesa. Tudi to bi precej pohitrilo izvajanje algoritma.

- **Eksperimentiranje**. Poleg pohitritev, ki bi jih prinesli optimizacija in paralelizacija algoritma, bi bilo smiselno izvesti tudi dodatne eksperimente. Ker je MCTS relativno nov algoritem predvsem na področju iger s kartami nepopolne informacije še ni bilo veliko raziskav, posledično tudi še niso povsem znani najboljši prijemi za to področje iger. Tukaj je možnosti res veliko - od še več eksperimentov z različnimi parametri, ki so že predstavljeni v tem delu, pa do eksperimentacije z različnimi selekcijskimi funkcijami in strategijami določanja vzvratnih vrednosti ter povečanju hevrističnega znanja.

- **Licitacija, zalaganje, napovedi**. V našem delu smo za licitiranje in zalaganje uporabljali kar ločeno hevristično funkcijo. Vsekakor bi bilo po optimizaciji algoritma smiselno poskusiti tudi z licitacijo in zalaganjem temelječim na obstoječem preiskovalnem algoritmu, pri čemer bi že pred samo igro simulirali možne variante zalaganja ter licitacije, in se nato odločili za tisto, ki bi dala najboljše rezultate. Enako velja za napovedi, ki niso bile vključene v igro. Tudi te bi lahko temeljile kar na rezultatih algoritma MCTS, ki bi že pred začetkom igre simuliral igro od začetka do konca za vse igralce.

Literatura

[1] G. Chaslot. *Monte-Carlo tree search*, str. 16-25. Dostopno na: `http://www.unimaas.nl/games/files/phd/Chaslot_thesis.pdf`

[2] T. van der Kleij. *Monte-Carlo Tree Search and Opponent Modeling through Player Clustering in no-limit Texas Hold'em Poker*, str. 2-3, 13-15. Dostopno na: `http://www.ai.rug.nl/~mwiering/Tom_van_der_Kleij_Thesis.pdf`

[3] L. Kocsis and C. Szepesvári. *Bandit based Monte-Carlo Planning*, 15th European Conference on Machine Learning (ECML 2006), str. 282-293, 2006.

[4] S. Russel. *Artificial Intelligence: A modern approach (3rd edition)*, New Jersey: Prentice Hall, 2009, poglavje 5.

[5] R. Coulom, *Efficient selectivity and backup operators in Monte-Carlo tree search,* Proceedings of the 5th International Conference on Computers and Games, str. 72–83, 2006

[6] I. Millington, J. Funge. *Artificial Intelligence for games, Second edition*, San Mateo, CA: Morgan Kaufmann, 2009, poglavje 8.

[7] D. Marinčič, M. Gams, M. Luštrek. *Knowledge vs. Simulation for Bidding in Tarok*, Informatica 30, str. 467-476. 2006. Dostopno na: `http://dis.ijs.si/domen/filez/Marincic_tarok_informatica.pdf`

[8] B. Bouzy, B. Helmstetter, *Monte-Carlo Go Developments*. Advances in Computer Games 10: Many Games, Many Challenges, str. 159–174, Kluwer Academic Publishers, Boston, MA, USA, 2003.

[9] S. Gelly in Y. Wang, *Exploration exploitation in Go: UCT for Monte-Carlo Go*, NIPS, 2006.

[10] M. Luštrek *Računalniško igranje iger s kartami*. Dostopno na: `http://dis.ijs.si/mitjal/documents/Racunalnisko_igranje_iger_s_kartami-BSc_thesis-02.pdf`

[11] M. Luštrek, M. Gams, I. Bratko. *A program for playing Tarok*, ICGA Journal, str. 190-197, September 2003.

[12] (2011) Min-Max algorithm. Dostopno na: `http://en.wikipedia.org/wiki/Minimax`

[13] (2011) Crazy Stone. Dostopno na: `http://en.wikipedia.org/wiki/Crazy_Stone_(software)`

[14] (2011) Monte-Carlo method. Dostopno na: `http://en.wikipedia.org/wiki/Monte_Carlo_method`

[15] (2011) MCTS.ai Dostopno na: `http://www.mcts.ai/?q=mcts`

[16] (2011) Slovenski tarok. Dostopno na: `http://www.slovenski-tarok.si/pravila.html`