Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Ilenič

Drevesno preiskovanje Monte Carlo pri namizni igri Scotland Yard

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Branko Šter

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Implementirajte algoritem drevesnega preiskovanja Monte Carlo in ga prilagodite namizni igri Scotland Yard, ki spada med igre z nepopolno informacijo. Igro Scotland Yard tudi implementirajte. Opišite obstoječe pristope k dani problematiki. Preizkusite delovanje algoritma proti nekaj različnim hevrističnim igralcem.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Nejc Ilenič sem avtor diplomskega dela z naslovom:

Drevesno preiskovanje Monte Carlo pri namizni igri Scotland Yard

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Branka Štera,
- 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 na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 1. septembra 2015 Podpis avtorja:

Zahvaljujem se prof. dr. Branku Šteru za usmerjanje, napotke, popravke in razlage med pisanjem diplomskega dela. Posebna zahvala gre vsej moji družini in bližnjim, ki so me podpirali in spodbujali čez celoten študij.

Kazalo

Seznam uporabljenih kratic

kratica	**angleško**	**slovensko**
$\alpha\beta$	Alpha-Beta pruning	Alpha-Beta rezanje
MCTS	Monte Carlo Tree Search	Drevesno preiskovanje Monte Carlo
UCT	Upper Confidence Bounds for Trees	Zgornja meja zaupanja pri drevesih
PSPACE	Polynomial amount of space	Polinomska količina prostora

Povzetek

Drevesno preiskovanje Monte Carlo zaradi uspeha pri računalniški igri *Go* postaja vse bolj uveljavljena metoda odločanja v različnih domenah. Za zelo uspešno se je izkazala pri igrah s popolno informacijo za enega, dva ali več igralcev, pri igrah, kjer igralcem v danem trenutku ni na voljo vsa informacija, pa je za večjo učinkovitost potrebno uvesti domensko specifične izboljšave.

V diplomskem delu so opisani in empirično preizkušeni obstoječi pristopi k problematiki uporabe drevesnega preiskovanja v namizni igri Scotland Yard. Izkazalo se je, da hevristična izbira možne lokacije igralca s popolno informacijo v največji meri vpliva na uspeh igralcev z nepopolno informacijo. Po poskusih se je MCTS igralec z vsemi izboljšavami izkazal kot konkurenčen nasprotnik človeškemu igralcu.

Ključne besede: drevesno preiskovanje, Monte Carlo, nepopolna informacija, Scotland Yard, odločanje, zgornja meja zaupanja pri drevesih, umetna inteligenca.

Abstract

Because of its success in the computer game of *Go*, Monte Carlo Tree Search is becoming a progressively popular decision making algorithm in various domains. It has proven its strengths in singleplayer and multiplayer games with perfect information, however domain specific improvements must be introduced in games with imperfect information.

In thesis existing approaches to the problem of applying the tree search to the Scotland Yard board game are described and empiricaly tested. It has turned out that heuristic selection of the possible location of the hider has the most impact on seekers performance. After testing, the MCTS player with all improvements has proven itself as a competitive opponent against the human player.

Keywords: tree search, Monte Carlo, imperfect information, Scotland Yard, decision making, Upper Confidence Bound for trees, artificial intelligence.

Poglavje 1

Uvod

Drevesno preiskovanje Monte Carlo je metoda za iskanje optimalnih potez s pomočjo naključnega vzorčenja in gradnje drevesa glede na rezultate. Spada v kategorijo iskanj, kjer se prostor preiskuje s pomočjo razširitve najobetavnejših vozlišč, izbranih glede na določena pravila (angl. best-first search), torej v primerjavi z bolj tradicionalnimi $\alpha\beta$-iskanji potrebuje zelo malo domenskega znanja. MCTS lahko uporabimo v katerikoli končni igri, ki jo lahko modeliramo kot stanje-akcija [5, 6, 9, 8].

Kljub zgoraj omenjenim prednostim se verjetnost izbire optimalne poteze poveča s prilagoditvijo algoritma izbrani domeni. To še posebej velja za domene z nepopolno informacijo, kjer agenti v danem trenutku ne poznajo vseh podatkov o trenutnem stanju. Primer take domene je namizna igra Scotland Yard, kjer skupina sodelujočih išče igralca, ki razkrije svojo lokacijo le po vnaprej določenih potezah.

1.1 Opis problematike

Za ustrezno delovanje algoritma morajo igralci z nepopolno informacijo glede na znane informacije oceniti trenutno stanje igre. K boljšim končnim rezultatom pripomorejo tudi drugi, s tem nepovezani procesi odločanja ter ustrezne nastavtitve parametrov MCTS iskanja. Ker so vsa ta znanja specifična za

izbrano domeno, je za uspešno evalvacijo delovanja algoritma nujno le ta poiskati, preizkusiti in jih oceniti.

1.2 Cilji naloge

Cilj naloge je opisati in implementirati različne obstoječe pristope k dani problematiki, ki jih predlagata Nijssen in Winands v članku [8]. Poleg tega je potrebno tudi empirično preizkusiti delovanje algoritma s simulacijami iger različnih hevrističnih igralcev ter delovanje algoritma proti človeškemu igralcu. Pridobljeni rezultati morajo biti na koncu predstavljeni in ocenjeni.

1.3 Struktura dela

- v 2. poglavju so opisani posamezni koraki algoritma drevesno preiskovanje Monte Carlo
- v 3. poglavju so opisana pravila namizne igra Scotland Yard
- v 4. poglavju so opisani obstoječi pristopi k problematiki vpeljave preiskovanja v igro z nepopolno informacijo Scotland Yard
- v 5. poglavju sta opisani implementaciji neodvisne knjižnice MCTS in igre Scotland Yard v Javi
- v 6. poglavju je predstavljena analiza rezultatov iger različnih hevrističnih igralcev

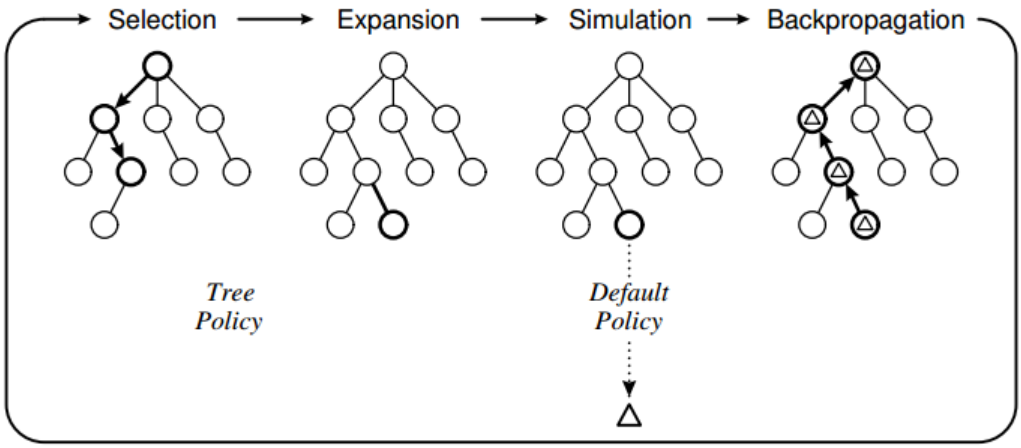
Poglavje 2

Drevesno preiskovanje Monte Carlo

Drevesno preiskovanje Monte Carlo je hevristično iskanje optimalnih potez, ki se je za zelo uspešno izkazalo v procesih odločanja, predvsem pri igranju iger. Tudi pri reševanju zahtevnih problemov, kjer so bile druge preiskovalne metode neuspešne, so rezultati pozitivni. Sloni na naključnem iskanju v preiskovalnem prostoru in za delovanje ne potrebuje sprotne funkcije vrednotenja, kar pomeni, da ni potrebnega nič oziroma zelo malo znanja iz izbrane domene. MCTS igralci v simulaciji izbirajo psevdo-naključne poteze, rezultati simulacij pa se uporabijo v fazi gradnje drevesa, ki predstavlja različna možna stanja igre. Simulacija ene igre ponudi malo znanja, po drugi strani pa se iz dovolj velike množice takih simulacij lahko oblikuje dobra strategija [5, 6, 9, 1].

2.1 Algoritem

Algoritem v osnovi iterativno gradi drevo, omejitev, na primer število iteracij pa je določena vnaprej. Vozlišča v drevesu predstavljajo stanja domene preiskovanja, povezave med vozlišči pa predstavljajo akcije do novih možnih stanj iz prejšnjega stanja. Vsaka iteracija se začne v korenskem vozlišču,



Slika 1: Iteracija algoritma MCTS

od koder poteka izbiranje poti do najugodnejšega razširljivega vozlišča, ki se mu doda nov otrok. Iz novo dodanega vozlišča se nato odvije simulacija. Rezultat simulacije se na koncu shrani v vsa vozlišča na izbrani poti in se uporablja pri naslednjih izbirah novega najugodnejšega vozlišča. Vsaka iteracija običajno torej predstavlja dodajanje enega novega vozlišča. Ob dosegu vnaprej določene omejitve se iskanje ustavi in algoritem vrne najbolj obetavno akcijo iz korenskega vozlišča. Zaradi narave algoritma je mogoče iskanje ustaviti pred dosegom omejitve, pomeni pa večje število iteracij večjo verjetnost izbire optimalne poteze [5, 9, 6].

Vsaka iteracija je razdeljena na štiri faze, ki so uvrščene v dve strategiji in si sledijo po naslednjem vrstnem redu. Fazi *izbira vozlišča* in *razširjanje vozlišča* spadata v *drevesno strategijo*, faza *simulacije* spada v *privzeto strategijo* in nazadnja faza *posodobitve rezultatov*. V praksi sta fazi *razširjanja vozlišča* in *simulacije* pogosto zamenjani. Posamezne faze so opisane v nadaljevanju.

2.1.1 Izbira vozlišča

Izbira vozlišča se vedno začne v korenskem vozlišču in se nato rekurzivno ponavlja na vseh nižjih nivojih, dokler ni najdeno ustrezno vozlišče za razširjanje.

Vozlišče je razširljivo, če ne predstavlja končnega stanja v domeni in ni že popolnoma razširjeno, kar pomeni, da so iz tega stanja možne še nepreizkušene akcije. Cilj je poiskati najbolj ugodno vozlišče za preiskovanje s pomočjo zbrane informacije iz prejšnjih iteracij. Na vsakem nivoju se naslednja akcija izbere na način, ki išče ravnotežje med izbiranjem trenutno najbolj obetavnih vozlišč in vozlišč, ki se lahko v prihodnosti izkažejo za obetavna. Ta način izbora imenujemo izkoriščanje obetavnih vozlišč proti raziskovanju novih vozlišč (angl. exploration vs. exploitation). Tovrstna izbira je implementirana s pomočjo zgornje meje zaupanja pri drevesih (angl. UCT) in je bolj podrobno opisana v podpoglavju 2.2 [5, 9].

2.1.2 Razširjanje vozlišča

Razširjanje vozlišča je običajno dodajanje enega (lahko tudi več) otroka vozlišču izbranemu v prejšnji fazi. Predstavlja stanje, ki do tega trenutka še ni bilo shranjeno v drevo, in navadno se ga izbere naključno glede na možne akcije iz izbranega vozlišča [5, 6].

2.1.3 Simulacija

Cilj simulacije je iz novo dodanega vozlišča odigrati igro do končnega stanja in oceniti izid, ki se nato uporabi pri naslednjih izbirah. Strategija igranja igre lahko naslednje akcije izbira uniformno-psevdo-naključno, vendar se izkaže, da so v tem primeru rezultati pogosto precej slabi in končna akcija posledično ni optimalna. Se pa rezultati močno izboljšajo, če strategija igranja izbira med akcijami, katerih verjetnosti so na primeren način obtežene. Za boljše delovanje algoritma so torej potrebna hevristična znanja, ki so specifična za domeno preiskovanja [5, 6].

2.1.4 Posodobitev rezultatov

Izid simulacije je potrebno na koncu ovrednotiti in rezultat shraniti za uporabo pri naslednjih izbirah. Vrednotenje poteka na vsakem nivoju drevesa,

rezultat pa se shrani v vsa vozlišča, ki so bila obiskana med izbiro vozlišča, vključno s korenskim vozliščem. Pri igrah z dvema igralcema se agent, ki ocenjuje končno stanje, menja izmenično, v splošnem pa stanje vedno ocenjuje trenutni agent prejšnjega nivoja v drevesu. Vsako vozlišče ima torej shranjeno informacijo o skupnem seštevku rezultatov iger, poleg tega pa tudi informacijo o številu odigranih iger na tem vozlišču. Preprost primer rezultatov je 1 točka za zmago, -1 točka za poraz in 0 točk za neodločen izid. Na vsakem nivoju se skupnemu rezultatu prišteje rezultat simulacije in poveča število odigranih iger. Ob koncu algoritma se najbolj obetavna akcija iz korenskega vozlišča navadno izbere glede na otroka, ki je bil med celotnim iskanjem največkrat obiskan [5, 6].

2.2 Zgornja meja zaupanja pri drevesih

Najbolj priljubljen algoritem v družini drevesnih Monte Carlo metod je algoritem UCT. Uporaba UCT v drevesni strategiji je bila empirično preizkušena in predlagana s strani Kocsis-a in Szepesvári-ja. Izbira otroka v vozlišču se prevede na neodvisne večroki bandit probleme (angl. multi-armed bandit problem), kjer je teoretična vrednost vsakega vozlišča pričakovana nagrada aproksimirana s strani Monte Carlo simulacij. Uspeh MCTS-ja je v največji meri rezultat tovrstne drevesne strategije. Dokazano je, da se ob približevanju števila iteracij neskončnosti verjetnost končne izbire akcije, ki ni optimalna, približuje ničli. UCT torej omogoča MCTS-ju, da konvergira k minimaks drevesu in je zato optimalen [7, 5, 2, 3].

$$UCT = v_j + 2C \times \sqrt{\frac{2 \ln n}{n_j}}$$

Formula UCT je preprosta in učinkovita. S seštevanjem prvega in drugega dela enačbe uravnoveša iskanje med obetavnimi vozlišči in raziskovanjem novih vozlišč. V enačbi v_j predstavlja skupno vrednost vozlišča j, n_j je število obiskov vozlišča j, n pa število obiskov starša vozlišča j. C je konstanta

oziroma nastavljiv parameter preiskovanja. Ob vsakem obisku vozlišča se imenovalec raziskovalnega dela enačbe poveča, torej se prispevek tega dela pomanjša. Na enak način se ob obisku neposrednega sorodnika vozlišča števec v ulomku poveča in s tem se poveča raziskovalni del vseh ostalih otrok. Posledica tega je, da imajo vsa vozlišča možnost preiskovanja in da bodo v dovolj dolgem času obiskana tudi tista z najslabšimi rezultati. Ob različnih parametrih preiskovanja bo algoritem torej v večji ali manjši meri preizkušal različne scenarije v domeni. Vozlišča, ki še niso bila obiskana, imajo privzeto neskončno UCT vrednost, s čimer zagotovimo, da je vsako vozlišče obiskano vsaj enkrat. Ob koncu algoritma lahko najobetavnejšo akcijo izberemo z isto formulo, kjer ima parameter C vrednost 0 in UCT vrednost predstavlja skupen seštevek nagrad vozlišča [5].

Poglavje 3

Namizna igra Scotland Yard

3.1 Opis igre

Scotland Yard je namizna igra skrivalnic, v kateri skupina sodelujočih detektivov poizkuša ujeti vnaprej določenega posameznika. Igralci se premikajo po igralni plošči - grafu, ki predstavlja zemljevid Londona. Izdelali so jo Manfred Burggraf, Dorothy Garrels, Wolf Hörmann, Fritz Ifland, Werner Scheerer in Werner Schlegel leta 1983, poimenovana pa je po središču londonske mestne policije Scotland Yard. V istem letu je prejela tudi nagrado za igro leta (angl. Game of the Year award). Kot namizna igra je bila izdana v dveh verzijah, prva s strani podjetja Ravensburger, druga pa s strani podjetja Milton Bradley. Podobno sta bila za igro razvita in izdana dva različna računalniška programa. Prvi program, razvit s strani podjetja Ravensburger Interactive Media GmbH, je bil razvit za operacijski sistem Windows, drugi program, razvit s strani podjetja DTP Young Entertainment GmbH & Co. KG, pa je bil razvit za ročno konzolo Nintendo DS [4, 8].

Graf, po katerem se gibljejo igralci, je neusmerjen in utežen, sama igra pa spada v kategorijo iger z nepopolno informacijo in je asimetrična ter PSPACE-popolna [4, 8].

3.2 Pravila igre

Igro lahko igra največ 6 igralcev: skupina petih detektivov in posameznik, ki se imenuje gospod X. Ker imajo detektivi skupen cilj ujeti Gospoda X-a, lahko Scotland Yard smatramo tudi kot igro za dva igralca. Vsi sodelujoči se premikajo po grafu, ki je sestavljen iz vozlišč označenih od 1 do 199, vozlišča pa so med seboj povezana z različnimi povezavami. Povezave se razlikujejo glede na dolžino premika in igralec mora skladno s tipom povezave za premik plačati z ustrezno kartico. Plačane kartice detektivov za vsak premik prejme Gospod X. Po najkrajših povezavah je možen premik s taksijem, po srednje dolgih povezavah premik z avtobusom, po najdaljših pa premik s podzemno železnico. Obstajajo še povezave, po katerih se je možno premikati s čolnom, vendar je uporaba le teh omejena na Gospoda X-a in na plačilo z univerzalno kartico.

Igra se začne z naključno postavitvijo igralcev na eno od 18 možnih začetnih polj, vsak pa dobi tudi začetne kartice za prevozna sredstva. Natančneje vsak od detektivov dobi 10 kartic za taksi, 8 kartic za avtobus in 4 kartice za podzemno železnico. Gospod X dobi 4 kartice za taksi, 3 kartice za avtobus in 3 kartice za podzemno železnico. Dodatno ima slednji na voljo še 5 univerzalnih kartic za katerokoli prevozno sredstvo in 2 kartici za dvojno potezo. Igralci odigrajo poteze v zaporednem vrstnem redu, v vsakem krogu vsak torej opravi natančno eno potezo, razen v primeru dvojne poteze Gospoda X-a. Lokacije detektivov so vsem vidne skozi celotno igro, iskani igralec pa se pokaže zgolj v 3, 8, 13, 18 in 24 krogu. Kljub temu mora po vsaki potezi detektivom razkriti nazadnje uporabljeno kartico prevoznega sredstva. Zmaga detektivov je ujetje Gospoda X-a, kar dosežejo s premikom enega od iskalcev na lokacijo iskanega. Zaradi števila kart, ki jih detektivi prejmejo na začetku, se igra odvija najdlje do 24 kroga. Po tem krogu se lahko premika le še iskani, kar pa predstavlja zmago za slednjega [8].

Poglavje 4

Obstoječi pristopi k dani problematiki

Ker je uporaba MCTS omejena na igre s popolno informacijo, so za uporabo pri namizni igri Scotland Yard potrebne določene prilagoditve. Prav tako lahko z vpeljavo znanja iz domene močno izboljšamo delovanje algoritma. V tem poglavju so opisane prilagoditve in izboljšave, ki jih v članku [8] predlagata J. (Pim) A. M. Nijssen in Mark H. M. Winands. Prvi dve podpoglavji opisujeta, kako se izogniti težavam zaradi nepopolne informacije detektivov, v nadaljevanju pa so predlagane ostale, s tem nepovezane izboljšave [8].

4.1 Omejevanje možnih lokacij iskanega

Problem nepopolne informacije rešimo s pretvorbo stanja igre v navidezno stanje, kjer je igralcem na voljo popolna informacija. V Scotland Yard-u lahko to storimo s postavitvijo Gospoda X-a na eno od prostih lokacij na igralni površini.

Ker morajo torej detektivi pred začetkom vsakega iskanja ugibati, kje se iskani nahaja, je smiselno možne lokacije na nek način omejiti. Naivni pristop bi bila na primer naključna izbira proste lokacije, vendar bi bilo iskanje v tem primeru neučinkovito. Ker po vsaki odigrani potezi Gospod

X razkrije uporabljeno prevozno sredstvo, lahko s pomočjo tega podatka in glede na prejšnje možne lokacije ter trenutne lokacije detektivov, nove možne lokacije natančno določimo. Po potezi detektiva se v primeru nadaljevanja igre njegova lokacija iz seznama umakne. Na začetku igre seznam lokacij vsebuje vse proste začetne lokacije, po vsakem krogu, ko se iskani pokaže, pa vsebuje le njegovo dejansko lokacijo. Na ta način močno omejimo število kandidatov za izbiro [8].

4.2 Hevristična izbira možne lokacije iskanega

Naslednji korak pretvorbe stanja igre v navidezno stanje s popolno informacijo je izbira ene izmed možnih lokacij, določenih v prejšnjem podpoglavju. S pomočjo vpeljave domenskega znanja lahko kandidate utežimo in po principu rulete izberemo najbolj verjetnega. V primeru, ko ima iskani na voljo premik na lokacijo, ki bi v naslednjem krogu skoraj zagotovo pomenila konec igre, obstaja velika verjetnost, da se ji bo poizkušal izogniti. Nijssen in Winands predlagata kategorizacijo in s tem utežitev lokacij na nekaj različnih načinov. Pri kategorizaciji glede na oddaljenost lokacije do najbližjega detektiva uvrstimo kandidate z oddaljenostjo ena v 1. kategorijo, kandidate z oddaljenostjo dva v 2. kategorijo in podobno vse do lokacij z oddaljenostjo pet ali več, ki jih uvrstimo v 5. kategorijo. Enako velja za kategorizacijo glede na povprečno oddaljenost do vseh detektivov. Pri kategorizaciji glede na vrsto prevoznega sredstva pa so kandidati uvrščeni v ustrezne kategorije glede na število povezav z različnimi prevoznimi sredstvi. Tukaj so možne štiri različne kategorije, in sicer prva za lokacije s taksi povezavami, druga za povezave s taksi in avtobus povezavami, tretja za povezave s taksi, avtobus in podzemnimi povezavami ter četrta za lokacije s povezavami s čolnom.

Vsaka kategorija c je nato utežena z vrednostjo $\frac{a_c}{n_c}$, kjer n_c predstavlja število ponovitev, ko je vsaj ena možna lokacija pripadala kategoriji c, a_c pa število ponovitev, ko je dejanska lokacija pripadala kategoriji c. Ustrezne ponovitve a in n lahko dobimo s simulacijo zadostnega števila iger. Detektivi

po vsaki odigrani potezi s pomočjo rulete določijo kategorijo za naključno izbiro lokacije, slednjo pa nato uporabijo pri samem iskanju [8].

4.3 Vpeljava domenskega znanja v privzeto strategijo

Z vpeljavo domensko-specifičnega znanja v fazo simulacije lahko močno izboljšamo končni rezultat iskanja. Nijssen in Winands predlagata uporabo ϵ-požrešne privzete strategije. V tej fazi se naslednja akcija trenutnega agenta izbere naključno z verjetnostjo ϵ, v nasprotnem primeru pa je izbira najboljše akcije hevristična. Gospod X z verjetnostjo 1-ϵ izbira poteze, ki maksimizirajo razdaljo do najbližjega detektiva. Če je teh potez več izbere tisto, ki maksimizira število možnih lokacij. Detektivi z verjetnostjo 1-ϵ izbirajo poteze, ki minimizirajo vsoto razdalj do vseh možnih lokacij iskanega. Če je potez, ki ustrezajo omenjenim kriterijem več, vsi igralci eno izmed teh izberejo naključno [8].

4.4 Učinkovitejše sodelovanje detektivov

Učinkovitejše sodelovanje detektivov dosežemo s prilagoditvijo faze posodobitve rezultatov. Ker lahko Scotland Yard smatramo kot igro dveh igralcev, se v primeru zmage detektivov v fazi shranjevanja rezultata simulacije vsem dodeli vrednost 1. Posledično se detektivi v preveliki meri zanašajo na ostale detektive in si ne prizadevajo dovolj za ujetje Gospoda X-a. Če v fazi simulacije iskanega najde ocenjujoči detektiv, se za shranjevanje vrne vrednost 1, v nasprotnem primeru pa se vrne vrednost 1-r, kjer je $r = [0, 1]$. Če je r premajhen, so detektivi preveč neprizadevni za ujetje Gospoda X-a, če je r prevelik, pa so preveč sebični in ne sodelujejo dovolj z ostalimi detektivi [8].

4.5 Učinkovitejša uporaba kartic Gospoda X-a

Ker ima Gospod X omejeno število univerzalnih kartic in kartic za dvojne poteze, je smiselno implementirati logiko za učinkovitejšo uporabo le teh. Z nekaj enostavnimi pogoji lahko preprečimo nepotrebno zapravljanje in s tem boljše preiskovanje iskanega. Univerzalna kartica naj se ne uporabi v prvih dveh krogih in v krogu, ko Gospod X razkrije svojo lokacijo, ter v situacijah, ko so na voljo le taksi povezave. V prvih dveh krogih je negotovost glede lokacije že sama po sebi dovolj velika, v ostalih dveh primerih pa z uporabo univerzalne kartice negotovosti ne povečamo. Podobno se uporaba kartice za dvojno potezo uporabi zgolj v situacijah, ko je povprečna razdalja vseh detektivov do iskanega dovolj nizka. Meja za uporabo dvojne kartice s je nastavljiv parameter [8].

Poglavje 5

Implementacija knjižnice MCTS in igre Scotland Yard

Razvoj programske opreme za potrebe poskusov je v osnovi razdeljen na dva dela. Prvi del obsega implementacijo neodvisne knjižnice drevesnega preiskovanja Monte Carlo v Javi, drugi del pa je implementacija namizne igre Scotland Yard v Javi. Za odločanje pri izbiri potez igralcev v igri je uporabljena MCTS knjižnica. Posamezne enote kode so testirane s pomočjo odprtokodne javanske knjižnice *JUnit4*, dostopne na naslovu https://github.com/junit-team/junit. V nadaljevanju so opisani najpomembnejši deli razvoja obeh komponent.

5.1 Implementacija MCTS

MCTS knjižnica je neodvisna, samo-vsebujoča komponenta in je posledično lahko uporabljena za potrebe preiskovanja v katerikoli domeni tipa stanje-akcija. Sestavljena je iz generičnih javanskih razredov in vmesnikov, kar omogoča uporabo različnih domensko specifičnih izboljšav. Poleg tega v inicializaciji in pri preiskovanju sprejme različne nastavljive parametre, ki vplivajo na delovanje algoritma in omogočajo prilagoditev preiskovanja različnim domenam.

Pri dodajanju vozlišč se za vsakega otroka naredi globok klon stanja. S tem se izognemo kvarjenju stanj višjih nivojev v drevesu med preiskovanjem. Za potrebe kloniranja sem uporabil odprtokodno javansko knjižnico *Java cloning library*, dostopno na naslovu https://github.com/kostaskougios/cloning. V fazi izbire vozlišča je uporabljena formula UCT, predstavljena v podpoglavju 2.2, neskončna UCT vrednost neobiskanih otrok vozlišča pa je dosežena s takojšnjo simulacijo igre iz novo dodanega otroka.

Uporaba knjižnice MCTS je prikazana v naslednjem odrezku kode. Najpomembnejši vidik neodvisnosti knjižnice sta generična vmesnika *MctsDomainAgent* in *MctsDomainState*, ki omogočata domensko specifične implementacije igralcev in stanj.

```
// implementacija vmesnika MctsDomainAgent
public class Player implements MctsDomainAgent<State> {...}

// implementacija vmesnika MctsDomainState
public class State implements MctsDomainState<Action, Player> {...}

// inicializacija preiskovanja
Mcts<State, Action, Player> mcts =
    Mcts.initializeIterations(NUMBER_OF_ITERATIONS);

// uporaba preiskovanja
Action mostPromisingAction = mcts.uctSearchWithExploration(state,
    explorationParameter);
```

Odrezek 1: Uporaba knjižnice MCTS

5.2 Implementacija igre Scotland Yard

Razreda *State* in *Player* namizne igre Scotland Yard implementirata vmesnika, prikazana v odrezku 1. Za potrebe poskusov so napisani statični razredi za različne simulacije iger v privzeti strategiji in za ostale izboljšave.

Tako lahko kreiramo različno kvalitetne igralce s poljubnimi izboljšavami, opisanimi v poglavju 4. Naključni igralec na primer uporablja izbiro naključnih potez, medtem ko pristranski igralec poteze izbira s pomočjo znanja, implementiranega v enem od statičnih razredov.

Zaradi nepopolne informacije detektivov so v razred stanja igre vpeljana pravila, ki različnim tipom igralcev vrnejo ustrezne informacije. Če je na primer stanje igre v preiskovanju in detektiv zahteva določeno informacijo o Gospodu X-u, morajo vrnjeni podatki vsebovati zgolj informacije o možni lokaciji iskanega. Primer takega pravila je prikazan v naslednjem odrezku kode.

```
public boolean seekerWon(Seeker seeker) {
  if (inSearchFromSeekersPov())
    return
        playersOnBoard.seekerOnHidersMostProbablePosition(seeker);
  else
    return playersOnBoard.seekerOnHidersActualPosition(seeker);
}
```

Odrezek 2: Preverjanje zmage specifičnega detektiva glede na tip igralca, ki zahteva informacijo

Ker so za implementacijo nekaterih izboljšav, omenjenih v poglavju 4, potrebne informacije o minimalni razdalji med dvema lokacijama, je med simulacijo potrebno te razdalje izračunati pred vsako potezo. Zaradi števila vozlišč in povezav na grafu igralne površine je čas računanja razdalj dolg. Posledično se močno poveča tudi čas izvajanja preiskovanja. Problemu se izognemo s predhodno shranitvijo razdalj med vsemi vozlišči in tako veliko hitrejšimi dostopi do podatkov med samim preiskovanjem. Za izračune je bil uporabljen algoritem Dijkstra, ki poišče najkrajše razdalje med vozlišči v grafu.

Poglavje 6

Poskusi in rezultati

Poskusi so razdeljeni v štiri različne sklope. V vsakem podpoglavju so prikazani in opisani rezultati iger različnih hevrističnih igralcev. Program se je izvajal na procesorju 1.7 GHz Intel® Core™ i5-3317U, čas izvajanja ene igre pa traja med 2 in 11 minutami. Zaradi omejenih procesorskih zmogljivosti je število iger enega poskusa omejeno na 20 in posledično so rezultati manj zanesljivi.

Naključni igralec vse poteze izbira psevdo-naključno, MCTS igralci pa s pomočjo drevesnega preiskovanja Monte Carlo. Osnovni MCTS igralec v fazi simulacije uporablja psevdo-naključno privzeto strategijo, požrešni MCTS igralec pa ϵ-požrešno privzeto strategijo. Ker se je izkazalo, da so detektivi Gospodu X-u konkurenčni zgolj z uporabo hevristične izbire možne lokacije iskanega, opisane v podpoglavju 4.2, le ta ni predmet poskusov. Uporabljajo jo vsi detektivi, razen naključnih. MCTS igralci optimalne poteze preiskujejo z 10000 iteracijami s parametrom preiskovanja 2 za detektive in 0.2 za Gospoda X-a. Povečevanje števila iteracij izboljša rezultate detektivov v večji meri kot pri Gospodu X-u [8].

Konkretno detektivi pri hevristični izbiri možne lokacije iskanega uporabljajo kategorizacijo glede na oddaljenost lokacije do najbližjega detektiva. Pri ϵ-požrešni simulaciji Gospod X izbira akcije, ki maksimizirajo razdaljo do najbližjega detektiva, detektivi pa izbirajo akcije, ki minimizirajo razdaljo

do izbrane možne lokacije iskanega.

V prvem podpoglavju so opisani rezultati poskusov iger naključnih in MCTS igralcev brez ostalih izboljšav, v drugem podpoglavju je predmet raziskave učinkovitejše sodelovanje detektivov, v tretjem pa učinkovitejša uporaba kartic Gospoda X-a. V četrtem podpoglavju so primerjani igralci z vsemi izboljšavami proti človeškem igralcu. Podatki v tabelah prikazujejo uspešnost detektivov proti iskanemu.

6.1 Primerjava naključnih in MCTS igralcev

Tabela 1 prikazuje uspešnost naključnih in MCTS detektivov proti naključnemu in MCTS Gospodu X-u. Pričakovano naključni igralec vedno izgubi proti MCTS igralcu, ne glede na vlogo. Zanimiv je podatek, da naključni detektivi zgolj v 30% zmagajo proti naključnemu Gospodu X-u, po drugi strani pa zgolj osnovni MCTS detektivi izgubijo proti požrešnemu MCTS iskanemu. Vsi ostali MCTS iskalci so v drugih primerih bolj učinkoviti kot Gospod X. Iz rezultatov je razvidna prednost (petih) detektivov, saj je igra Scotland Yard asimetrične narave. Pričakovano so vsi požrešni MCTS igralci bolj učinkoviti od osnovnih igralcev istega tipa [8].

detektivi/Gospod X	`naključni`	`osnovni MCTS`	`požrešni MCTS`
`naključni`	30%	0%	0%
`osnovni MCTS`	100%	60%	40%
`požrešni MCTS`	100%	65%	55%

Tabela 1: Primerjava naključnih in MCTS igralcev brez ostalih izboljšav

6.2 Učinkovitejše sodelovanje detektivov

V tabeli 2 zgolj detektivi uporabljajo učinkovitejše sodelovanje, iskani pa so enaki kot v tabeli 1. Uporabljena vrednost parametra r je 0.25. Strategija

v vseh primerih poveča učinkovitost iskalcev, je pa razlika bolj očitna pri požrešnih MCTS igralcih. Pri osnovnih MCTS detektivih se je uspešnost v povprečju povečala za 12.5%, pri požrešnih MCTS detektivih pa za 27.5%.

detektivi/Gospod X	`osnovni MCTS`	`požrešni MCTS`
`osnovni MCTS`	70%	55%
`požrešni MCTS`	90%	85%

Tabela 2: Primerjava MCTS iskanih z MCTS detektivi, ki uporabljajo učinkovitejše sodelovanje

6.3 Učinkovitejša uporaba kartic Gospoda X-a

V tabeli 3 zgolj Gospod X uporablja učinkovitejšo uporabo kartic, detektivi pa so enaki kot v tabeli 1. Uporabljena vrednost parametra s je 3. Podobno kot v prejšnjem primeru izboljšava v vseh primerih poveča učinkovitost Gospoda X-a, vendar razlika pri uporabi osnovnega in požrešnega igralca ni tako očitna. Osnovni igralec v povprečju zmanjša učinkovitost detektivov za 32.5%, požrešni pa za 30%. V primerjavi z učinkovitejšim sodelovanjem detektivov da učinkovitejša uporaba kartic za Gospoda X-a boljše rezultate. Rezultati v tabeli 2 kažejo v povprečju 20% povečanje učinkovitosti detektivov, medtem ko rezultati v tabeli 3 kažejo v povprečju 31.25% povečanje učinkovitosti iskanega. Gospod X v vseh primerih premaga detektive.

detektivi/Gospod X	`osnovni MCTS`	`požrešni MCTS`
`osnovni MCTS`	20%	15%
`požrešni MCTS`	40%	20%

Tabela 3: Primerjava MCTS detektivov z MCTS Gospodom X-om, ki uporablja učinkovitejšo uporabo kartic

6.4 Požrešni MCTS igralec proti človeškemu igralcu

Požrešni MCTS igralci z vsemi izboljšavami se izkažejo kot enakovreden nasprotnik človeškemu igralcu. Tako detektivi kot iskani pri preiskovanju uporabljajo učinkovitejše sodelovanje detektivov in učinkovitejšo uporabo kartic Gospoda X-a. Kljub rezultatom poskusov so zaradi asimetrične narave igre proti človeškemu igralcu detektivi precej bolj uspešni kot Gospod X in se izkažejo za težke nasprotnike. Iskani je v večini odigranih iger proti povprečnemu človeškemu igralcu izgubil, ujet pa je bil največkrat med 11 in 13 krogom. Detektivi so proti povprečnemu človeškemu igralcu zmagali večkrat, kot so izgubili.

Poglavje 7

Zaključek

V diplomskem delu je implementirana neodvisna knjižnica za drevesno preiskovanje Monte Carlo in namizna igra Scotland Yard. Preizkušeno je delovanje algoritma v igri z nepopolno informacijo in raziskane so različne prilagoditve iskanja za izbrano domeno. Pri uporabi knjižnice so bili implementirani obstoječi pristopi k dani problematiki, ki jih v članku [8] predlagata J. (Pim) A. M. Nijssen in Mark H. M. Winands. Učinkovitost različnih pristopov je bila preizkušena s pomočjo večjega števila simuliranih iger med različnimi hevrističnimi igralci. Rezultati so pokazali, da je namizna igra Scotland Yard asimetrična in da so detektivi kljub nepopolni informaciji v večini primerov bolj učinkoviti od Gospoda X-a. Za najučinkovitejšo izboljšavo se je izkazala hevristična izbira možne lokacije iskanega. Detektiv z vsemi predlaganimi izboljšavami se je izkazal kot težak nasprotnik proti povprečnemu človeškemu igralcu. Rezultati poskusov kažejo, da lahko drevesno preiskovanje Monte Carlo uspešno prilagodimo in vpeljemo v domeno z nepopolno informacijo, delovanje algoritma pa dodatno izboljšamo s pomočjo domensko-specifičnih znanj.

Literatura

[1] Monte Carlo Tree Search research hub. `http://mcts.ai/`. Dostopano: 15. 6. 2015.

[2] Sensei's Library - Monte Carlo Tree Search. `http://senseis.xmp.net/?MonteCarloTreeSearch`. Dostopano: 17. 6. 2015.

[3] Sensei's Library - UCT. `http://senseis.xmp.net/?UCT`. Dostopano: 17. 6. 2015.

[4] Wikipedia - Scotland Yard board game. `https://en.wikipedia.org/wiki/Scotland_Yard_(board_game)`. Dostopano: 11. 8. 2015.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, March 2012.

[6] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. In Christian Darken and Michael Mateas, editors, *AIIDE*. The AAAI Press, 2008.

[7] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

[8] P. Nijssen and M. H. M. Winands. Monte carlo tree search for the hide-and-seek game scotland yard. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(4):282–294, Dec 2012.

[9] M. H. Winands, Y. Björnsson, and T. Saito. Monte-carlo tree search solver. In *Proceedings of the 6th International Conference on Computers and Games*, CG '08, pages 25–36, Berlin, Heidelberg, 2008. Springer-Verlag.