Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Neža Belej

Analiza preiskovalnih metod na primeru igre Scotland Yard

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: izr. prof. dr. Polona Oblak

Ljubljana 2015

Rezultati diplomskega dela so intelektualna lastnina avtorja. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Cilj diplomske naloge je preučiti metodo drevesnega preiskovanja Monte-Carlo in jo aplicirati na asimetrično namizno igro Scotland Yard. Preučiti je potrebno vse korake algoritma in se posvetiti predvsem tretji fazi, simulaciji. Simulacija naj se implementira na različne načine. Pri tem naj bo vsaj en način osnoven, kar pomeni, da se od trenutnega vozlišča simulacija potez izvaja popolnoma naključno. Ostali načini naj bodo naprednejši, kar pomeni, da bolje ponazarjajo dejansko igrano igro. Implementirajte avtomatsko igranje igre, pri čemer se detektivi premikajo s pomočjo različnih implementacij drevesnega preiskovanja Monte-Carlo, lopov pa se premika naključno ali pa pametno. Kombinacije testirajte na več odigranih igrah in rezultate primerjajte po številu zmag in po številu potez, potrebnih za zmago.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Neža Belej sem avtorica diplomskega dela z naslovom:

Analiza preiskovalnih metod na primeru igre Scotland Yard

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom izr. prof. dr. Polone Oblak,
- 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 29. avgusta 2015 Podpis avtorja:

Zahvaljujem se svoji mentorici, izr. prof. dr. Poloni Oblak, za pomoč pri izbiri tematike diplomskega dela, za vse hitro odgovorjene e-maile, ter za zabavno in sproščeno sodelovanje z obilico dragocenih nasvetov.
Nejc, hvala za vse, kar si storil zame. Za tvojo podporo, potrpežljivost in optimizem, ki mi dajejo pogum za uresničevanje ciljev.
Zahvaljujem se moji družini, ki mi je ta študij omogočila, še posebno Primožu za pomoč pri mojih prvih korakih programiranja.
V času študija sem spoznala nekaj posebnih ljudi in prijateljev, s katerimi je bil študij toliko lažji : Veronika, Manca, Matevž, Blaž, Alen in Matej, hvala tudi vam.

Moji mamici.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Računalniki in igre 1

1.2 Cilj 2

2 Igra Scotland Yard **5**

2.1 Opis igre 5

2.2 Priprava igre 7

2.3 Potek igre 8

2.4 Posebne poteze lopova 8

3 Drevesno preiskovanje Monte-Carlo **11**

3.1 Drevesno preiskovanje Monte-Carlo 11

3.2 Struktura MCTS 11

3.2.1 Izbira 12

3.2.2 Razširitev 13

3.2.3 Simulacija 13

3.2.4 Posodabljanje vozlišč 14

3.2.5 Minimalno število obiskov vozlišča 14

3.3 Uporaba MCTS na primeru igre Scotland Yard 15

3.3.1 Problematika apliciranja MCTS na igro Scotland Yard 15

3.3.2 Določanje konstant C in r 19
3.3.3 Predstavitev različnih metod simulacije 20

4 Implementacija **25**
4.1 Programski jezik in okolje 25
4.2 Optimizacijski problemi 25
4.3 Metoda za iskanje najmanjše razdalje 26
4.4 Izboljšave in olajšave 28
4.5 Implementacija vozlišča 29

5 Rezultati **31**
5.1 Delež zmag detektivov in povprečna zaporedna številka zmagovalne poteze 31
5.2 Časovna analiza tipov simulacij 34
5.3 Obrazložitev 35
5.4 Vpeljava konstante T 37

6 Sklep **39**

Seznam uporabljenih kratic

kratica	angleško	slovensko
MCTS	Monte Carlo Tree Search	drevesno preiskovanje Monte Carlo
BFS	Breadth First Search	preiskovanje v širino
AI	Artificial Intelligence	umetna inteligenca
UCT	Upper Confidence bounds for Trees	zgornja meja zaupanja za drevesa

Povzetek

V diplomskem delu se seznanimo s področjem umetne inteligence, ki se ukvarja z raziskovanjem namiznih iger in iskanjem njihovih programskih rešitev. Preučimo algoritem drevesnega preiskovanja Monte-Carlo in ga poskušamo čim bolj spretno prenesti na znano namizno igro Scotland Yard, pri čemer upoštevamo nasvete Nijssena in Winandsa [9]. Osredotočimo se predvsem na tretjo fazo algoritma, simulacijo, katero se odločimo implementirati na tri različne načine (od manj naprednih do bolj naprednih), te načine pa želimo kasneje med seboj primerjati. Poskušamo ugotoviti, do kakšne mere se napredna izvedba simulacije obrestuje v nasprotju s časovno manj potratnimi metodami. Ker želimo izvesti avtomatsko preverjanje iger, implementiramo tudi samo igro, v kateri detektivi igrajo po prej omenjenem algoritmu, lopov pa se premika na dva načina - naključno in pametno. Vseh teh šest kombinacij želimo avtomatsko testirati na večjem številu iger in rezultate primerjati ter jih razložiti.

Ključne besede: drevesno preiskovanje Monte-Carlo, Scotland Yard, namizne igre, umetna inteligenca.

Abstract

In the thesis we learn about the field of artificial intelligence that investigates board games and their program-based solutions. We examine Monte-Carlo tree search algorithm and transfer it to well-known board game Scotland Yard, considering advices from Nijssen and Winands [9]. We focus mainly on the third phase of the algorithm, playout, and decide to implement it in three different ways (from less to more advanced techniques). We compare these three aproaches. We compare the win rates and computation time of simple and advanced methods. We also implement the game to the purpose of automated testing. In this game, detectives play by Monte-Carlo tree search algorithm and Mister X plays in two different ways - random and advanced. We want to test all of these six combinations on a large number of games, compare the results and explain them.

Keywords: Monte-Carlo tree search, Scotland Yard, board games, artificial intelligence.

Poglavje 1

Uvod

1.1 Računalniki in igre

Namizne igre so bile izumljene že v času prvih civilizacij. Ena najstarejših namiznih iger se imenuje Senet, igrali pa so jo Egipčani okoli leta 3500 pr. n. št. Natančnih navodil za to igro se ne ve, vendar je gotovo, da je igra bila igrana še 3000 let kasneje. Nekatere izmed starodavnih iger se igrajo še danes, ena izmed takih je na primer kitajska namizna igra Go, ki je stara približno 4000 let [6].
Skozi stoletja se je število takih iger množično povečalo, od pojavitve računalnikov pa je v umetni inteligenci preiskovanje iger postalo eno najpomembnejših področij. Tako dandanes že velika večina računalnikov zmore precej prepričljivo premagati človeka pri igranju iger, implementiranih s pomočjo naprednih algoritmov. Precej uspeha je bilo na področju uravnoteženih iger za dva igralca, kot so na primer šah, dama in Go. Večji izziv je razviti igro z več igralci (angl. multiplayer game), kjer so metode za prej omenjene igre le pogojno uporabne. Igre za več igralcev lahko razvrstimo v dve kategoriji [6]:

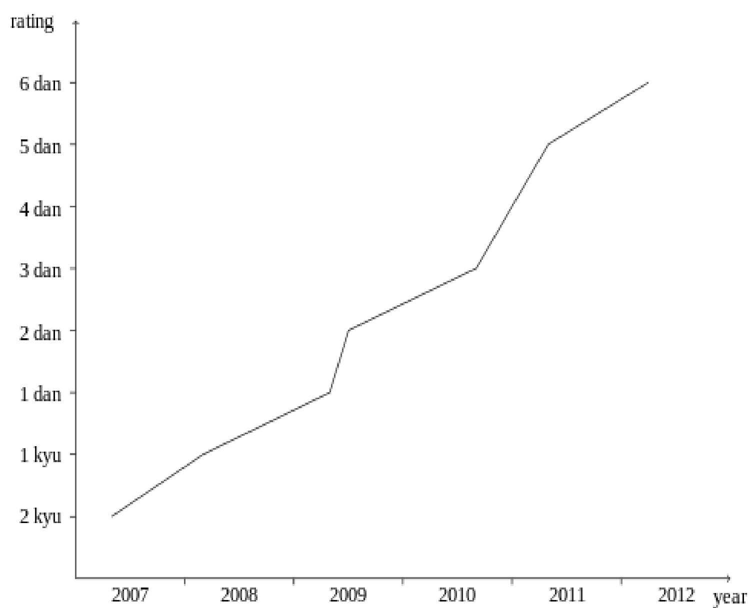
- deterministične igre s popolnimi informacijami,
- igre z nepopolnimi informacijami (angl. hide-and-seek games).

Slika 1.1: Novejša različica starodavne egipčanske namizne igre Senet (vir slike: http://www.unclesgames.com/index-store.php).

Naše zanimanje je vzbudila popularna igra Scotland Yard, ki sodi med igre z nepopolnimi informacijami. Poimenovana je po znani britanski policiji in je, tako kot pove že ime, namenjena vsem navdušencem nad logičnimi namiznimi igrami. Sama igra je predvsem zelo napeta in razgibana, zanimiva pa je tudi komunikacija med detektivi, ki morajo med sabo sodelovati pri iskanju osebe X (lopov).

1.2 Cilj

Naš cilj pri izdelavi diplomske naloge je poiskati programsko rešitev za igralce v vlogi detektivov pri igri Scotland Yard. Za preiskovanje smo si izbrali algoritem drevesnega preiskovanja Monte-Carlo, saj v mnogih igrah ta metoda prinaša odlične rezultate. Želimo si implementirati različne taktike detektivov, od manj do bolj naprednih metod, ki bi jih primerjali med seboj na podlagi večjega števila igranih iger proti nasprotniku (lopovu).
Za testiranje teh metod potrebujemo veliko množico igranih iger, zato mo-



Slika 1.2: Rezultati najboljših Go programov vse od leta 2007. Od tega leta vsi najboljši programi uporabljajo MCTS [5]. Rang **kyu** se nanaša na šibkejše nasprotnike, rang **dan** pa na močnejše. Graf nam pove, da računalnik v letu 2012 zmore premagati že zelo močnega nasprotnika.

ramo omogočiti avtomatizacijo igranja. To vodi k implementaciji potez lopova, ki bi igral čim bolj podobno človeškemu igranju. Naše metode bomo zato najprej testirali proti lopovu, ki igra povsem naključno, kasneje pa še proti lopovu, ki se premika pametno z manjšo možnostjo naključnega premika.

Poglavje 2

Igra Scotland Yard

Opis in pravila igre so povzeta po [4].

2.1 Opis igre

Scotland Yard je namizna igra, v kateri skupina petih detektivov sodeluje pri iskanju osebe "Mister X"(v nadaljevanju lopov). Na plošči, ki prikazuje zemljevid Londona, je izrisanih 199 lokacij, po katerih se lahko premikajo igralci. Te lokacije se med seboj povezujejo s tremi prometnimi povezavami: taksi, avtobus in podzemna železnica. Vsaka lokacija je postaja za eno ali več prometnih sredstev. Glede na barvo postaje vidimo, katero prevozno sredstvo se lahko tu ustavi. Lokacija detektivov je vedno znana, medtem ko je točna lokacija lopova znana le vsako peto potezo. Detektivom sta znani dve vrsti informacij:

- Lokacija zadnje znane pojavitve lopova.
- Prevozna sredstva, ki jih je lopov uporabil.

Na podlagi teh informacij se detektivi lahko pametno premikajo po plošči in s tem ujamejo lopova ali pa izločijo možne lokacije lopova.
Igra je dobila ime po sedežu znamenite londonske policije. Proizvajalec igre je Ravensburger, leta 1983 pa je prejela nagrado Spiel des Jahres (igra leta).

Slika 2.1: Del igralne plošče igre Scotland Yard.

2.2 Priprava igre

Na začetku igre si igralci morajo razdeliti vloge. Eden izmed njih mora prevzeti vlogo lopova, ostali so detektivi. Če je detektivov manj kot pet, je priporočljivo, da en igralec igra več detektivov. Število igralcev mora tako biti od dva do šest.

Pred začetkom igre vsak detektiv dobi :

- 10 taksi vozovnic,
- 8 avtobusnih vozovnic,
- 4 vozovnice za podzemno železnico,
- figurico.

Lopov dobi:

- 4 vozovnice za taksi,
- 3 avtobusne vozovnice,
- 3 vozovnice za podzemno železnico,
- 2 vozovnici za dvojni premik,
- toliko črnih vozovnic, kot je detektivov,
- kapa s ščitnikom (kot zaščita pred pogledi),
- tabelo premikov, kot jo lahko vidimo na sliki 2.2,
- figurico.

Igralci izmed 18 možnih začetnih mest izvlečejo svojo začetno lokacijo. Detektivi svoje figurice postavijo na pripadajoče lokacije.

Slika 2.2: Tablica premikov lopova.

2.3 Potek igre

Prvo potezo naredi lopov. To naredi tako, da v tabelo premikov napiše številko polja, na katerega se bo premaknil. Ta zapis pokrije z vozovnico (taksi / avtobus / podzemna železnica / skrita poteza). Sledi premik detektivov. Po premiku porabljeno vozovnico podarijo lopovu. V tretji potezi mora lopov po premiku svojo figurico položiti na trenutno lokacijo na igralni plošči. Naslednjič ponovi to v osmi potezi, kasneje pa še v 13., 18. in zadnji - 24. potezi. Lopov zmaga, če ga detektivi po 24 potezah ne ulovijo (lopov pride do zadnjega polja v tabeli premikov) oziroma, ko detektivom zmanjka vozovnic za nadaljni premik. Če kakšen detektiv med igro pride na lokacijo lopova, mora ta to sporočiti, in zmaga je v rokah detektivov.

2.4 Posebne poteze lopova

Obstajata dve vrsti posebnih vozovnic, ki jih sme uporabiti le lopov:

- Dvojna poteza

Lopov ima dvakrat v igri možnost, da se premakne za dve postaji hkrati. Pri tem vpiše obe postaji v tabelo premikov in na obe položi ustrezni vozovnici. Karto "2X" mora ob tem dati na stran. Če s prvo od dveh potez pride do polja, kjer se mora pokazati, se tako vmes pokaže in se nato še enkrat premakne.

- Črna karta

  Črna karta se lahko uporabi na dva načina. Prvi je, da lopov skrije uporabljeno prevozno sredstvo. Tako detektivi nimajo informacije, s katerim prevoznim sredstvom se je lopov premaknil. Ob tem je vredno povedati, da je seveda nesmiselno to vozovnico uporabiti v potezah, kjer se lopov mora pokazati, saj s tem detektivom ne skrije nobene informacije. Na igralni plošči je nekaj povezav tudi po reki Temzi. Po teh povezavah se sme premikati le lopov, in sicer le s črno karto. To pa je tudi drugi način, kako uporabiti črno karto. Detektivi seveda ne vedo, katerega od načinov je uporabil lopov.

Poglavje 3

Drevesno preiskovanje Monte-Carlo

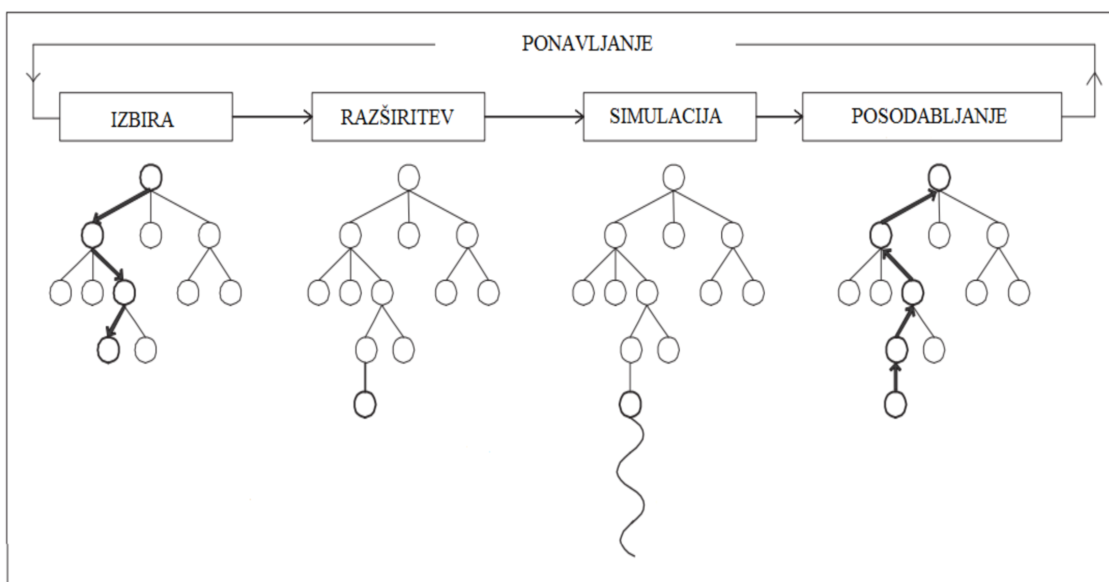
3.1 Drevesno preiskovanje Monte-Carlo

Drevesno preiskovanje Monte-Carlo (MCTS) je nadgradnja naključne metode Monte-Carlo, ki temelji na ponavljajočem se naključnem vzorčenju za pridobivanje numeričnih rezultatov.
Pri MCTS gre za naključno preiskovanje iskalnega prostora, pri čemer ves čas gradimo drevo in v njem shranjujemo rezultate raziskovanja. Na začetku so poteze izbrane povsem naključno, sčasoma pa se, s pomočjo sproti zgrajenega preiskovalnega drevesa, vse bolj izbirajo poteze z večjo verjetnostjo uspešnih rezultatov [3].

3.2 Struktura MCTS

Metoda MCTS je sestavljena iz štirih glavnih korakov:

- Izbira vozlišča,
- razširitev vozlišča,
- simulacija igre iz pravkar razvitega vozlišča in



Slika 3.1: Shema drevesnega preiskovanja Monte-Carlo iz članka [3].

- posodabljanje vozlišč.

Vse štiri faze so za boljšo predstavo ilustrirane na sliki 3.1, vsaka posebej pa je opisana v spodnjih podpoglavjih.

3.2.1 Izbira

Izbira je prva faza drevesnega preiskovanja Monte-Carlo. V tem koraku potujemo od korena drevesa vse do vozlišča, ki še ni bilo v celoti raziskano, kar pomeni, da vsebuje otroke, ki še niso bili priključeni drevesu. V vsakem nivoju drevesa se tako izbere eden izmed otrok trenutnega vozlišča.
Pri izbiri vozlišča sta pomembna predvsem dva pojma: **izkoriščanje** (angl. exploitation) in **raziskovanje** (angl. exploration). Pri izkoriščanju gre za izbiro do sedaj najbolj obetavnih vozlišč, raziskovanje pa skrbi za obiskovanje manj obetavnih vozlišč [3].
Izbere se potomec z največjim številom točk v_i. Pri tem vsako vozlišče točkujemo po formuli UCT (angl. Upper Confidence bounds applied to

Trees) [7].:

$$v_i = \frac{s_i}{n_i} + C \cdot \sqrt{\frac{\ln(n_p)}{n_i}}, \quad (3.1)$$

v kateri s_i ponazarja vsoto točk vozlišča i, pri čemer je zmaga točkovana z eno točko, poraz pa z nič točkami. Spremenljivki n_i in n_p ponazarjata število obiskov otroka i in starša p. C je konstanta, ki uravnava razmerje med zgoraj omenjenima pojmoma, raziskovanjem in izkoriščanjem.
Omenimo še, kaj se zgodi, če je $n_i = 0$. V tem primeru lahko trdimo, da je $v_i = \infty$, kar v praksi pomeni, da s formulo dosežemo, da bo doslej neobiskano vozlišče zagotovo obiskano.
Konstanta C naj bi bila v teoriji enaka $\sqrt{2}$, vendar je v praksi ponavadi najdena empirično in je odvisna od samega problema.

3.2.2 Razširitev

V drugi fazi se k drevesu doda novo vozlišče, razširjeno iz nazadnje izbranega vozlišča iz prejšnje faze. Otrok, dodan v tej fazi, je izbran naključno.

3.2.3 Simulacija

Simulacija (ang. playout) je najbolj kompleksen korak v MCTS. Bistvo simulacije je, da se od razširjenega vozlišča iz prejšnje faze samodejno izvajajo poteze vse do konca igre. Te poteze so lahko popolnoma naključne, lahko pa upoštevajo razne *strategije igranja* [3]. Te strategije pogosto težijo k čim bolj realistični simulaciji igre. Vodijo k bolj zanesljivim in kvalitetnejšim rezultatom, a so pogosto precej bolj računsko zahtevne, kar pomeni, da dovoljujejo manj iteracij MCTS-ja v istem časovnem obdobju, kot naključne poteze [14].
Če povzamemo: če je strategija igranja preveč *stohastična* (izbira poteze skorajda naključno), je rezultat lahko precej šibek. Če je strategija preveč *deterministična* (poteze so izbrane vedno na enak način; preveč je *izkoriščanja*),

je *raziskovanje* lahko premalo pogosto [3].
Zato je dobro izbrati neko enostavnejšo hevristično funkcijo, ki ohranja ravnotežje med zgornjima pojmoma. Sturtevant v svojem članku predlaga strategijo *ϵ-požrešne simulacije* [11], kar pomeni, da algoritem izbere najboljšo potezo z verjetnostjo $1 - \epsilon$. V nasprotnem primeru je izbrana naključna poteza.
Ob koncu simulacije simulirana igra dobi nek rezultat: zmaga ali poraz.

3.2.4 Posodabljanje vozlišč

V zadnji fazi algoritma MCTS se rezultat simulirane igre prenese nazaj po drevesu vse od vozlišča, razširjenega v drugi, razširitveni fazi, do korena drevesa. Pot poteka preko vozlišč, izbranih v prvi fazi (faza izbiranja). Tukaj se posodabljajo vsi potrebni podatki za izračun vrednosti v_i v (3.1): obiskanost vozlišča (vsem vozliščem na poti se prišteje 1), ter podatek o zmagi oziroma porazu (1 / 0).

3.2.5 Minimalno število obiskov vozlišča

Na tem mestu pojasnimo pojem **minimalnega števila obiskov vozlišča** [15]. Kot smo že omenili, se pri izbiranju vozlišča rekurzivno pomikamo od korena drevesa navzdol po vejah, izbranih s formulo UCT, vse dokler ne pridemo do vozlišča, ki je bilo obiskano manj kot T-krat. Ko pridemo do takega vozlišča, prenehamo z izbiranjem in iz vozlišča začnemo izvajati simulacijo. V osnovnem algoritmu MCTS je $T = 1$, pogosto pa se pojavi eksperimentalno določanje praga T.

Vsi štirje koraki drevesnega preiskovanja Monte-Carlo se ponovijo n-krat (na primer 10000-krat), ali pa so časovno omejeni [7]. Ob koncu procesa se ponavadi izbere otroka z največjim številom obiskov [2]. Ta način, imenovan *robusten otrok (ang.: robust child)*, smo uporabili tudi mi, saj smo se ravnali po podatkih iz članka, ki se je ukvarjal s problematiko algoritma MCTS na igri

Scotland Yard. Obstajajo še tri tehnike izbiranja: izbira otroka z največjim številom točk (*največji otrok, ang.: max child*), izbira otroka z največjim številom obiskov in največjim številom točk (*ang.: robust-max child*), ali pa izbira *varnega otroka (ang.: secure child)*, t.i. otroka, ki maksimizira najnižjo mejo zaupanja [3].

3.3 Uporaba MCTS na primeru igre Scotland Yard

3.3.1 Problematika apliciranja MCTS na igro Scotland Yard

Zakaj nam implementacija drevesnega preiskovanja na igro Scotland Yard predstavlja izziv, pojasnijo njene tri pomembne lastnosti [9]:

- Igra vsebuje **nepopolno informacijo** za del igralcev. V našem primeru se to nanaša na detektive, ki večji del igre ne vedo točne lokacije lopova. Detektivi ves čas izvajajo javne poteze, medtem ko je lokacija lopova jasna samo v 3., 8., 13., 18. in 24. krogu igre.

- Igra vsebuje **koalicijo (sodelovanje) petih detektivov**, kar zahteva večji razmislek, kako v tem primeru učinkovito implementirati MCTS. Tukaj se pojavi tudi vprašanje, kako točkovati zmago detektivov, saj lopova lahko ujame le en detektiv, zmaga pa pripada vsem detektivom.

- Igra je **asimetrična**, kar se tiče njenih ciljev. To pomeni, da nasprotni igralci nimajo istih ciljev. Lopov se namreč želi izogniti detektivom vse do konca igre, detektivi pa želijo ujeti lopova.

Po dobrem razmisleku in po prebranih člankih [8] in [9] za zgoraj opisane probleme predlagamo naslednje rešitve:

- Rešitev za problem nepopolne informacije:

1. Najbolj očitna stvar, ki nam pade na pamet ob problemu nejasne lokacije lopova, je uporaba informacij o vozovnicah, ki jih je uporabil lopov. Na tak način lahko ves čas vzdržujemo seznam možnih lokacij lopova.

   Spodaj je predstavljena psevdokoda izračuna seznama možnih lokacij lopova (algoritem 1). Seznam možnih lokacij je na tak način

Algorithm 1 Izračun možnih lokacij lopova [9]

```
1: procedure MrXLocation
2:     K ← PossibleLocations
3:     PossibleLocations ← 0
4:     if currentRound ∈ (3, 8, 13, 18, 24) then
5:         L ← location(hider)
6:     else
7:         for all p ∈ K do
8:             T ← targets(p,ticket)
9:             PossibleLocations ← PossibleLocations ∪ (T \SeekersLocations)
10:        end for
11:    end if
12: return PossibleLocations
13: end procedure
```

   posodobljen po vsaki potezi lopova, glede na uporabo njegove porabljene vozovnice. Pri tem uporablja za osnovo prejšnji seznam možnih lokacij lopova (kam se je lahko premaknil od prejšnjih možnih lokacij), pri čemer se lopov ne more premakniti na lokacijo detektivov (vrstica 9 v algoritmu 1). Metoda *target(p, ticket)* vrne seznam lokacij, ki so dostopne iz lokacije p z vozovnico *ticket*. Ko se detektiv premakne na eno izmed možnih lokacij, lopova pa ne ujame, se ta lokacija umakne z našega seznama.

2. Za soočanje z nepopolnimi informacijami bomo uporabili tudi princip **determinizacije**.

V tretji fazi drevesnega preiskovanja, simulaciji, je potrebno v samodejno igrani igri premikati detektive po nekem principu - tu lahko po mnenju Nijssana [9] uporabimo dve hevristiki:

(a) minimizacija vsote števila potez, ki bi jih moral opraviti detektiv, da bi prišel do posamičnih možnih lokacij,

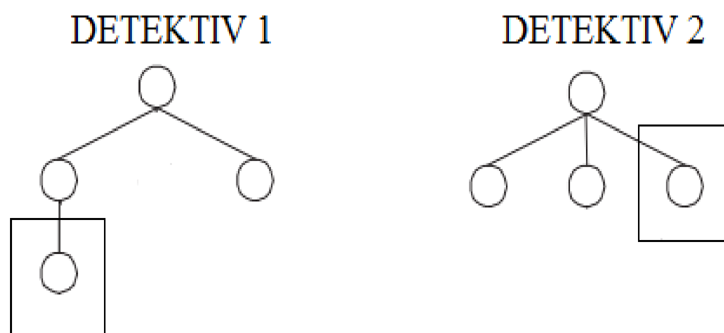
(b) minimizacija števila potez, ki bi jih moral opraviti detektiv, da prispe do predpostavljene lokacije lopova.

Po nasvetu iz članka [9] se odločimo za uporabo druge točke, ki na prvi pogled izgleda tudi precej manj časovno potratna. Pojem determinizacija pomeni predpostaviti lokacijo lopova glede na neko domensko znanje.
Na primer, če ima lopov možnost, da se premakne za dve mesti ali pa za eno mesto stran od najbližjega detektiva, bo bolj verjetno izbral tisto, ki je za dve mesti oddaljeno. Na tak način si torej v vsaki potezi simulacije izberemo eno izmed možnih lokacij lopova. Možne lokacije lopova dobimo po algoritmu 1. Za determinizacijo lokacije lopova uporabimo verjetnosti iz naslednje tabele, pridobljene iz članka [8]:

Tabela 3.1: Kategorije oddaljenosti lokacije lopova

Kategorija	1	2	3	4	5
a	2454	9735	4047	1109	344
n	12523	14502	7491	2890	1756

V tabeli 3.1 je razvidno obnašanje lopova po 1000-ih igranih igrah. Število n ponazarja število situacij, ko bi se lopov lahko premaknil na lokacijo, ki je za dano kategorijo oddaljena od najbližjega detektiva. Število a ponazarja število situacij, ko se je lopov dejansko premaknil na lokacijo iz dane kategorije.



Slika 3.2: Reševanje problema petih dreves

Vsako lokacijo iz seznama možnih lokacij lopova opremimo s kategorijo (razdalja do najbližjega detektiva), potem pa z verjetnostmi, pridobljenimi iz tabele 3.1 predpostavimo lokacijo lopova v tej potezi simulacije.
Detektivi se potem premikajo proti tej lokaciji.

- Rešitev za problem koalicije petih detektivov:

  1. Odločimo se, da bomo za vsakega detektiva zgradili svoje drevo, saj sklepamo, da bi drevo, ki kot vozlišča vsebuje kombinacije detektivov, bilo preveč časovno potratno.
     Slika 3.2 prikazuje problem, ki se pojavi pri vzporedni gradnji petih dreves. Pogosto se zgodi, da drevesa posameznih detektivov v prvi fazi izberejo vozlišča na različnih nivojih kot drugi detektivi. Slika 3.2 bi v praksi pomenila, da je prvi detektiv odigral dve potezi, drugi detektiv pa samo eno. Zato predlagamo naslednjo strategijo: Po zaključeni drugi fazi drevesnega preiskovanja Monte-Carlo poiščemo drevo detektiva, ki ima pravkar razvito vozlišče v najnižjem nivoju. Naj bo ta nivo poimenovan *minLevel*. Iz izbranih poti preostalih detektivov izberemo vozlišča, ki ležijo na nivoju *minLevel*. Na tak način dobimo iz izbranih poti vseh detektivov vozlišča na istem nivoju, iz katerih se bo izvajala na-

daljna simulacija igre. Tudi posodabljanje vozlišč se dogaja le od vozlišč, iz katerih se je izvajala simulacija.

2. Ob zmagi detektivov se lahko pojavita dve različni situaciji: detektivi obkolijo lopova, da se ne more nikamor premakniti, druga možnost pa je, da eden izmed detektivov stopi na mesto lopova. Ob tej situaciji se pojavi vprašanje, kako točkovanje zmag vpliva na delovanje algoritma. V članku [9] je predlagana rešitev **zmanjšanja koalicije** (coalition reduction). Gre za taktiko, ki detektivu, ki dejansko ujame lopova, nameni 1 točko, medtem ko ostali detektivi dobijo $1 - r$ točk, pri čemer je $r \in [0, 1]$.
   Paziti je potrebno, da r ne sme biti prevelik, saj detektivi postanejo s tem preveč "individualni", res pa je tudi, da lahko nekaj detektivov primerno obkoli lopova, na njegovo mesto pa stopi le en - torej nima le ta detektiv zaslug. Po drugi strani pa, če je r premajhen, lahko dobijo detektivi, ki so lahko precej oddaljeni od lopova, preveč zaslug. Zato je treba izbrati primeren r, ki uravnava ti dve skrajnosti.

- Rešitev za problem asimetrične naravnanosti igre:
  Za rešitev tega problema je potrebno uporabiti domensko znanje igre, in ga pravilno integrirati v algoritem MCTS. Za razliko od simetričnih iger, je potrebno spisati drugačne metode za nasprotujoče si igralce. Odločili smo se, da se bomo posvetili pisanju algoritmov za vlogo detektivov, saj se nam zaradi problematike nepopolne informacije zdi to večji izziv.

3.3.2 Določanje konstant C in r

Spomnimo se, kaj pomenita konstanti C in r. Konstanta C je omenjena v podpoglavju 3.2.1, in sicer v formuli (3.1). Gre za konstanto, ki uravnava ravnotežje med raziskovanjem in izkoriščanjem. Teoretično je enaka $\sqrt{2}$, vendar je v praksi skoraj vedno določena empirično.

Konstanta r je opisana v prejšnjem razdelkui. Gre za odbitek točke tistih detektivov, ki kljub zmagi niso bili zaslužni za ujem lopova.

Testirali smo različno obnašanje detektivov za različne kombinacije konstant C in r v dveh različnih situacijah: ko so detektivi blizu lopova, in ko so detektivi daleč stran od lopova. Testirali smo kombinacije naslednjih vrednosti konstant:

- C: 0.5, 1, 1.5, 2, 2.5, 3
- r: 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1

Vsako kombinacijo smo preizkusili 30-krat. Izbrali smo kombinacijo $C = 1.0$ in $r = 0.7$, saj smo iz rezultatov ocenili, da se detektivi pri tej kombinaciji še najbolje premikajo v različnih situacijah postavitve igralcev. Pri odločanju smo takoj zavrgli rezultate, kjer se detektiv, ki je za eno mesto oddaljen od lopova, ne pomakne na njegovo mesto in ostale situacije, kjer je očitno, da premikanje ni optimalno. Tukaj še omenimo, da je bilo v rezultatih razvidno, da se ista kombinacija lahko obnaša različno v situaciji, ko so detektivi bližje ali daleč stran od lopova. Sicer se izračunavi optimalne kombinacije nismo podrobneje posvetili, saj v diplomskem delu nameravamo predvsem pri enakih konstantah C in r testirati različne metode simulacije.

3.3.3 Predstavitev različnih metod simulacije

Simulacija je tretji korak MCTS, opisan v poglavju 3.2.3. Kot smo že omenili, gre za samodejno izbiranje potez vse do konca igre. Poteze so lahko naključne ali delno naključne. Primerna strategija simulacije lahko bistveno izboljša rezultate igre [1]. Strategijo izberemo na podlagi **hevristik**, ki jih najdemo v domenskem znanju specifične igre. Hevristike so ocene, ki izboljšajo učinkovitost raziskovanja [10].
Odločili smo se implementirati tri vrste simulacij, ki jih bomo kasneje testirali na večjem številu iger in primerjali med seboj:

1. **Naključno premikanje detektivov in naključno premikanje lopova**
   Gre za najbolj preprosto in osnovno obliko simulacije v MCTS. Lopov se premika popolnoma naključno, onemogočeno mu je le stopiti na trenutno lokacijo detektiva. Detektivi se premikajo popolnoma naključno, s tem da dva detektiva ne smeta stopiti na isto mesto v neki potezi.

2. **Neodvisno približevanje detektivov in pametno igranje lopova**
   V tej strategiji smo razvili nekoliko pametnejše obnašanje lopova in detektivov.

   (a) Lopov igra po naslednjem algoritmu:
   Najprej za vse možne naslednje pozicije lopova izračunamo razdalje do vseh detektivov. Tako dobimo tabelo spodnje oblike:

Tabela 3.2: Oddaljenost naslednjih možnih lokacij lopova do posameznih detektivov.

Pozicija lopova	D1	D2	D3	D4	D5
27	1	3	4	2	2
34	2	2	2	1	1
11	13	2	1	5	6

Razlaga tabele za ta primer: če se lopov premakne na pozicijo 27, bo za eno mesto oddaljen od prvega detektiva, za tri mesta od drugega, za štiri mesta od tretjega itd. Če se premakne na pozicijo 34, bo za eno mesto oddaljen od petega detektiva in podobno. V celotni tabeli poiščemo minimum. Nato v tabeli 3.2 poiščemo vrstico, kjer se ta minimum najmanjkrat pojavi. Vrstice s to frekvenco minimuma pustimo, ostale odstranimo iz tabele. V našem primeru tako lahko odstranimo drugo vrstico. Če imajo vse vrstice enako frekvenco minimuma, se nobena vrstica ne izbriše.
Po preureditvi tabele minimum povečamo za ena in ponavljamo

postopek. V našem primeru je v drugem koraku minimum enak dva. Zbrišemo prvo vrstico tabele, saj je frekvenca števila dva v tej vrstici enaka dva (v zadnji vrstici pa je frekvenca enaka ena). Ko v tabeli ostane samo še ena vrstica, postopek končamo. V našem primeru je izbrana lokacija 11. V primeru, da sta dve vrstici popolnoma enaki, s postopkom seveda ne moremo priti le do ene vrstice. Zato podamo še drugi pogoj za izstop iz rekurzije: če je naš minimum večji ali enak 5, postopek tudi lahko zaključimo, saj ob takih razdaljah lopov ni v nevarnosti.

Na zgoraj opisan način se lopov optimalno odmakne od detektivov. Ker pa mora biti simulacija delno naključna, poskrbimo, da obstaja 10% možnosti za naključen premik lopova.

(b) Detektivi igrajo po naslednjem algoritmu:

Vsak detektiv se premakne najbližje predpostavljeni lokaciji lopova, upoštevajoč svoj nabor vozovnic in dejstva, da se ne sme premakniti na mesto drugega detektiva.

Tukaj obstaja 20% možnost naključne poteze detektivov.

3. **Sodelovanje detektivov in pametno igranje lopova**

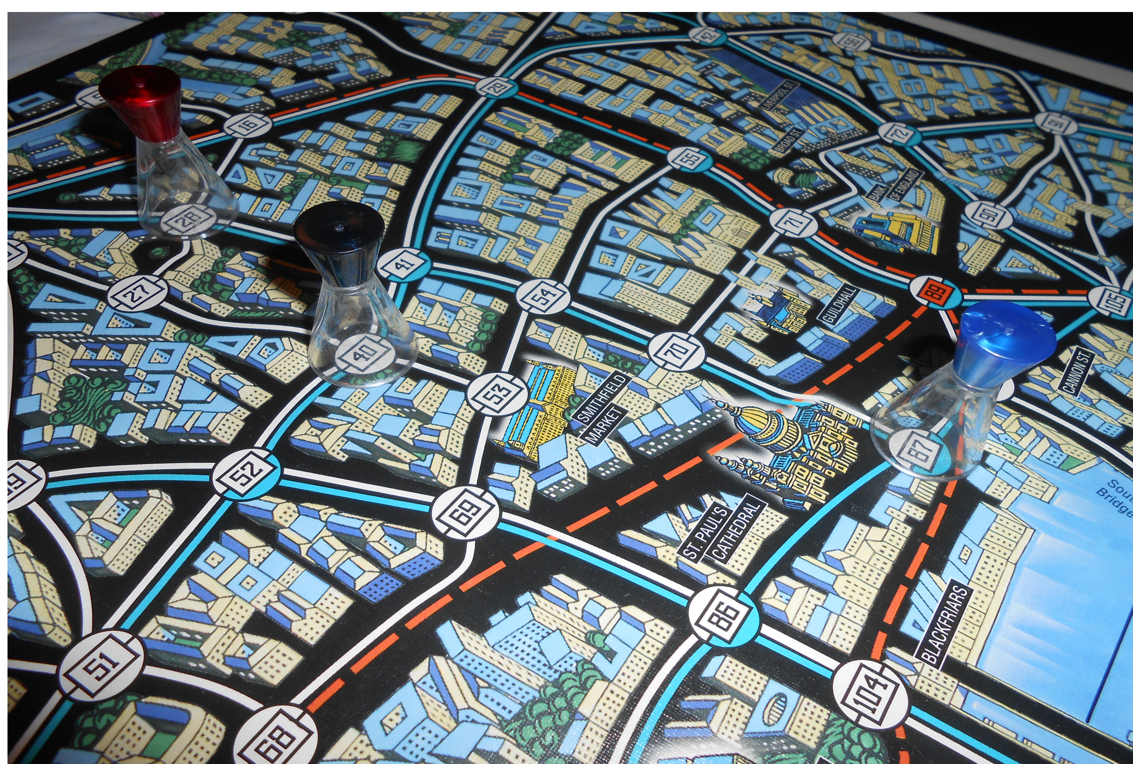
   (a) Lopov igra po naslednjem algoritmu:

   Algoritem je enak opisanemu algoritmu lopova v (2.a).

   (b) Detektivi igrajo po naslednjem algoritmu:

   Algoritem 2.b ima naslednjo pomanjkljivost: poglejmo si sliko 3.3. Črna figurica predstavlja lopova, rdeča in modra pa dva izmed detektivov. Denimo, da je prvi na vrsti rdeči detektiv. Če se premakne na pozicijo 41 ali pa na pozicijo 27, bo le za 1 mesto oddaljen od pozicije lopova. Denimo, da se premakne na mesto 41. Zdaj pride na vrsto drugi, modri detektiv. Njegova najbližja pozicija za ujem lopova bi bila lokacija 41, vendar je to pozicijo zasedel že rdeči detektiv. Zato se mora premakniti na drugo najmanjšo razdaljo do lopova. Tako **skupna minimalna razdalja do lopova**

ni najmanjša. Zato uvedemo naslednji algoritem:
Preverjamo vse možne kombinacije naslednjih mest detektivov, upoštevajoč njihove vozovnice. Izberemo kombinacijo, ki ima skupno minimalno razdaljo najmanjšo. Na tak način rešimo zgoraj opisani problem.
Tudi tukaj obstaja 20% možnost naključne poteze detektivov.



Slika 3.3: Primer pozicije dveh detektivov in lopova (črna figurica).

Poglavje 4

Implementacija

4.1 Programski jezik in okolje

Programerski del diplomskega dela smo napisali v programskem jeziku Java, saj ta programski jezik najbolje poznamo. Iz istega razloga smo kot razvojno okolje uporabili Eclipse.

4.2 Optimizacijski problemi

V okviru programerskega dela smo naleteli na nekaj situacij, kjer bi naivna implementacija vzela enostavno preveč časa za obsežno testiranje. Opisali bomo problem, ki se zgodi pri implementaciji metode simulacije, ki je opisana v 3. poglavju, v razdelku 3.3.3. (metoda za sodelovanje detektivov).
Pri tej metodi preverjamo vse možne kombinacije naslednjih mest detektivov, upoštevajoč njihove vozovnice. Na tak način najdemo rešitev, ki prinaša minimalno skupno razdaljo do lopova. Problem, ki se tukaj pojavlja, je velika časovna zahtevnost. Če ima vsak izmed petih detektivov v povprečju 5 možnih lokacij za premik, pomeni, da moramo preveriti $5^5 = 3125$ možnih kombinacij. Pri prometnih vozliščih (tista z vsemi vozovnicami) je lahko tudi do 13 možnih naslednjih mest, tako da se ta številka lahko tudi močno poveča. Ker faze MCTS-ja iteriramo tudi po 10000-krat, bi to pomenilo večminutno

izvajanje MCTS-ja.
Zato predlagamo naslednjo optimizacijo:
najprej za vsakega detektiva poiščemo lokacijo, na katero se lahko premakne v naslednji potezi in ki je najbližja dani poziciji lopova. Potem gremo za vsak par detektivov preverjati, če je njuna izbrana lokacija enaka. Če je, poiščemo optimalni premik le za ta par detektivov. To storimo s pomočjo kombinacij naslednjih lokacij trenutnih mest.
Opisana metoda je ena izmed možnosti, kako rešiti problem, ki se zgodi ob neodvisnem premikanju detektivov, obenem pa je precej manj časovno potratna od naivne implementacije, kjer preverjamo čisto vsako možno kombinacijo naslednjih lokacij detektivov.

4.3 Metoda za iskanje najmanjše razdalje

Večkrat smo pri implementaciji naleteli na problem, ko je bilo treba izračunati najmanjšo razdaljo med dvema igralcema. Primeri:

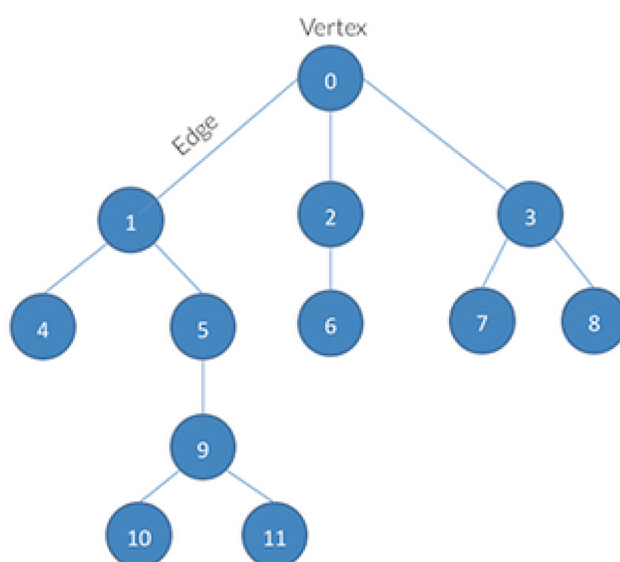
- Detektiv se mora v simulaciji premikati najbližje domnevni poziciji lopova.
- Pametni lopov se premika tako, da maksimizira minimalno razdaljo do najbližjega detektiva.
- Uporaba pri *determinizaciji* lokacije lopova, ko so mesta z določeno razdaljo verjetnostno ovrednotena.

Za iskanje najmanjše razdalje smo uporabili **preiskovanje v širino** (angl. Breadth First Search). Gre za algoritem, ki vedno najde optimalno pot med dvema vozliščema v grafu. Vrstni red preiskovanja v širino je razviden iz slike 4.1, spodaj pa je spisana tudi psevdokoda algoritma.
Časovna zahtevnost algoritma je je $O(bd)$, pri čemer b pomeni razvejitveni faktor grafa (število naslednikov vsakega vozlišča), d pa predstavlja globino drevesa (grafa). Prostorska zahtevnost je enaka številu vozlišč na zadnjem

Algorithm 2 Algoritem BFS (preiskovanje v širino) po [13].

```
1: procedure BreadthFirstSearch(graph, sourceVertex)
2:     connected[]                    ▷ Tabela, ki hrani povezave oče-otrok.
3:     V ← sourceVertex
4:     Queue Q
5:     Q.add(V)
6:     A[]                            ▷ Tabela, ki hrani že obiskana vozlišča.
7:     while Q not empty do
8:         Q.remove(v)
9:         for child C connected to parent V do
10:            if C not checked then
11:                connected[C] ← v
12:                C ← checked
13:                Q.add(C)
14:            end if
15:        end for
16:    end while
17: end procedure
```

nivoju drevesa: $O(b^d)$ [13].



Slika 4.1: Vrstni red preiskovanja vozlišč pri algoritmu preiskovanja v širino.

4.4 Izboljšave in olajšave

V tem podpoglavju bomo opisali tri modifikacijske tehnike, ki smo jih uporabili pri implementaciji:

- Zaradi naključne narave drevesnega preiskovanja Monte-Carlo lahko pride do slabših rezultatov v primeru, ko so si detektivi zelo blizu. Zato upoštevamo nasvet Teytauda [12], ki uvaja pojem **odločitvenih potez**. Gre za to, da v primeru, ko ima igralec na voljo zmagovalno potezo, to potezo tudi izvede, čeprav se morda MCTS ni odločil za to potezo.

  To pravilo smo upoštevali na tak način, da smo detektiva vedno pre-

maknili na eno izmed možnih mest lopova, če je to bilo mogoče. S tem smo lahko ali ujeli lopova, ali pa skrajšali seznam njegovih možnih lokacij.

- Na začetku igre seznam možnih lokacij lopova vsebuje 13 mest. Ko se lopov nekam premakne, se ta seznam bistveno podaljša. Opazili smo, da je algoritem MCTS zaradi dolžine tega seznama v prvih dveh potezah nesmiseln in časovno preveč potraten, zato ga v prvih dveh potezah ne izvajamo. MCTS je smiseln šele po tretji potezi, ko se lopov pokaže.
  V realnosti je v prvih dveh potezah cilj detektivov, da pridejo na čim bolj mobilna mesta, torej na lokacije z vsemi možnimi povezavami, kar se tiče vozovnic. Zato za vse začetne lokacije že prej smiselno nastavimo prvi dve potezi detektivov, pri čemer pazimo, da v posamezni potezi nobena lokacija ne sme biti enaka drugi.

- V tem poglavju naj še omenimo, da v implementacijo nismo vključili skritih in dvojnih vozovnic lopova. Tako smo se odločili zato, ker implementacija teh vozovnic ne vpliva bistveno na primerjavo strategij v simulaciji. Dejstvo je tudi, da bi uporaba teh vozovnic lahko bistveno podaljšala sezname možnih lokacij lopova, in s tem precej podaljšala tudi čas igranja.
  Zato smo se odločili, da posebnih vozovnic ne implementiramo.

4.5 Implementacija vozlišča

Vozlišče (in posledično drevo) smo implementirali tako, da smo ustvarili razred *Node*, ki hrani naslednje informacije o vozlišču drevesa:

- zaporedno število vozlišča,
- seznam otrok vozlišča,
- število točk vozlišča,

```
public class Node {
    private int nodeNumber;
    private ArrayList<Node> next;
    private double score;
    private int visits;
    int[] tickets;

    public Node(int nodeNumber_)

    void setNodeNumber(int n)
    void setNext(ArrayList<Node> n)
    void setNext(Node n)
    void setScore(double n)
    void setVisits(int n)
    int getNodeNumber()
    ArrayList<Node> getNext()
    double getScore()
    int getVisits()
}
```

Slika 4.2: Vmesnik razreda *Node*.

- število obiskov,
- stanje vozovnic.

Na tem mestu se spomnimo, da drevo gradimo za vsakega detektiva posamezno in za vsako potezo znova. *Koren* drevesa bo torej trenutna lokacija detektiva. *Število točk* in *število obiskov* vozlišča sta vsoti, ki se seštevata znotraj enega MCTS procesa. Ker v naši implementaciji proces MCTS obsega 10000 iteracij, pomeni, da bo vsota števila obiskov vseh otrok korena vedno enako 10000.

Ker s premiki v globino drevesa ponazarjamo pot detektiva v prihodnosti, je nesmiselno iti preveč v globino, saj ima detektiv na voljo omejeno število vozovnic. Zato vsako vozlišče hrani tudi podatek o tem, kakšno je njegovo stanje vozovnic v nekem vozlišču.

Poglavje 5

Rezultati

Po implementaciji vseh zgoraj opisanih metod smo zagnali 250 iger za vsako od spodaj opisanih kombinacij igranja. Podrobnosti posameznih metod simulacije so opisane v 3.3.3. Pri odigranih igrah so nas zanimale predvsem naslednje informacije:

- Delež zmag detektivov,
- povprečna zaporedna številka zmagovalne poteze,
- čas igranja.

5.1 Delež zmag detektivov in povprečna zaporedna številka zmagovalne poteze

Sledijo opisi posameznih tipov iger in dobljeni rezultati za dani tip igre.

- Prvi tip igre: Detektivi, ki igrajo po MCTS, pri čemer uporabljajo simulacijo tipa *naključno premikanje detektivov - naključno premikanje lopova* proti avtomatskemu igralcu lopova, ki igra povsem naključno.

- Drugi tip igre: Detektivi, ki igrajo po MCTS, pri čemer uporabljajo simulacijo tipa *naključno premikanje detektivov - naključno premika-*

nje lopova proti avtomatskemu igralcu lopova, ki igra pametno na tak način, da rekurzivno maksimizira minimalno razdaljo do najbližjega detektiva, pri čemer obstaja 10% možnosti naključnega premika.

- Tretji tip igre: Detektivi, ki igrajo po MCTS, pri čemer uporabljajo simulacijo tipa *samostojno približevanje detektivov - pametno igranje lopova* proti avtomatskemu igralcu lopova, ki igra povsem naključno.

- Četrti tip igre: Detektivi, ki igrajo po MCTS, pri čemer uporabljajo simulacijo tipa *samostojno približevanje detektivov - pametno igranje lopova* proti avtomatskemu igralcu lopova, ki igra pametno na tak način, da rekurzivno maksimizira minimalno razdaljo do najbližjega detektiva, pri čemer obstaja 10% možnosti naključnega premika.

- Peti tip igre: Detektivi, ki igrajo po MCTS, pri čemer uporabljajo simulacijo tipa *sodelovanje detektivov - pametno igranje lopova* proti avtomatskemu igralcu lopova, ki igra povsem naključno.

- Šesti tip igre: Detektivi, ki igrajo po MCTS, pri čemer uporabljajo simulacijo tipa *sodelovanje detektivov - pametno igranje lopova* proti avtomatskemu igralcu lopova, ki igra pametno na tak način, da rekurzivno maksimizira minimalno razdaljo do najbližjega detektiva, pri čemer obstaja 10% možnosti naključnega premika.

Tabela 5.1: Delež zmag detektivov in povprečna zaporedna številka zmagovalne poteze.

Tip igre	Delež zmag (%)	Delež porazov (%)	Povprečna zmagovalna poteza
1.	100	0	6,77
2.	71	29	10,48
3.	100	0	6,36
4.	81	19	9,82
5.	100	0	5,91
6.	80	20	10,01

Naš algoritem igranja detektivov proti naključno igranemu lopovu igra zelo dobro pri vseh treh tipih simulacije, a že tukaj je razvidno, da bolj napreden tip simulacije prinaša boljše rezultate. Vidimo namreč, da detektivi v povprečju ulovijo lopova v 6,77. potezi pri igri z naključno simulacijo (prvi tip igre); pri naprednejši obliki simulacije s samostojnim premikanjem detektivov in pametnim premikanjem lopova (tretji tip igre) pa detektivi v povprečju ulovijo lopova v 6,36. potezi. Pri naši najnaprednejši obliki s koalicijo detektivov in pametnim premikanjem lopova (peti tip igre) detektivi v povprečju zmagajo že pred šesto potezo.

Kot smo predvidevali, so rezultati za igranje proti pametnemu lopovu bolj raznoliki. Potrdimo svoje razmišljanje, ko vidimo, da je bolj napreden tip simulacije bolj uspešen kot povsem običajen tip z naključnimi premiki. Slednji nam prinaša namreč samo 71-odstotno zmago.
Vidimo, da sta rezultata naprednejših tipov (četrti in šesti tip) uspešnejša za približno 10%.

Iz tabele je jasno razvidno tudi, da se naprednejša tipa simulacij po uspešnosti ne razlikujeta bistveno. Prav tako ni večje razlike v zaporedni številki zmagovalne poteze. Zanimivo je, da tip igre z individualnim pristopom detektivov

prinaša celo nekoliko boljše rezultate. Menimo, da se to zgodi iz sledečih razlogov: Scotland Yard je igra z nepopolno informacijo. Zaradi tega je v naši implementaciji med drugim prisotne veliko naključnosti. Spomnimo se, da je bil eden izmed načinov, kako reševati problem nepopolne informacije, metoda determinizacije. Pri tej metodi si v simulaciji naš program naključno izbere eno izmed možnih potez lopova. Metoda, s katero se detektivi premikajo proti lokaciji lopova, je lahko še tako napredna, a če predvidena lokacija ni enaka dejanski, metoda ne bo uspešna.
Za nadaljne raziskovanje smo opravili nekaj časovnih testov, saj nas zanima, katero metodo je bolje uporabiti iz časovnega vidika.

5.2 Časovna analiza tipov simulacij

Za časovne meritve smo zagnali večje število iger, pri čemer smo vsako potezo detektivov izračunali na vse tri načine drevesnega preiskovanja Monte-Carlo. Vsak proces MCTS za vsako potezo smo merili programsko in pretečen čas izpisovali v konzolo, kot kaže slika 5.1. Tako smo dobili raznolike procese MCTS-ja z različno dolgimi možnimi seznami lokacij lopova. V posameznem procesu algoritma MCTS smo izvedli 10000 iteracij.

```
Round 8
X's new location: 76 Ticket: 1
X coulds: [76]
Time Smart Detectives Playout: 62
Time Individual Detectives Playout: 23
Time Random Playout: 6
Detectives: 157 36 24 65 13
```

Slika 5.1: Izpis v konzolo pri merjenju časa.

Po stotih potezah detektivov smo izračunali povprečen čas razmišljanja detektivov v eni potezi. Prišli smo do naslednjih ugotovitev:

- Povprečen čas razmišljanja pri tipu simulacije *naključno premikanje detektivov - naključno premikanje lopova* je 45.58 sekund.
- Povprečen čas razmišljanja pri tipu simulacije *samostojno približevanje detektivov - pametno igranje lopova* je 148.54 sekund.
- Povprečen čas razmišljanja pri tipu simulacije *sodelovanje detektivov - pametno igranje lopova* je 270.7 sekund.

Ker se časovne meritve od procesa do procesa precej razlikujejo, je potrebno omeniti še naslednje informacije: V veliki večini potez je bila časovna zahtevnost simulacije z uporabo 3. tipa največja, časovna zahtevnost simulacije z uporabo 1. tipa pa najmanjša. V krajših procesih MCTS-ja sta bila 1. in 2. tip simulacije zelo podobna, medtem ko je v zahtevnejših procesih (takih z daljšim seznamom možnih lokacij lopova) prišla do izraza večja razlika, kot je vidno tudi iz rezultatov meritev.

Meritve smo izvajali na računalniku s procesorjem z naslednjimi specifikacijami: *Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz 2.00 GHz.*

5.3 Obrazložitev

Glede na tabelo 5.1 ugotovimo, da se uporaba naprednejših metod simulacije do neke mere obrestuje. Pri osnovni simulaciji, ki vsebuje popolnoma naključne poteze lopova in detektivov, so rezultati dokazano slabši kot pri naprednejših oblikah simulacije. A naša metoda detektivov, ki se individualno premikajo bližje lopovu prinaša zelo podobne rezultate kot pametnejša metoda detektivov (tretji tip simulacije), pa vendar porabi približno polovico manj časa.
Iz tega lahko sklepamo, da je v našem primeru uporaba tretjega tipa simulacije neučinkovita. Razlog za to leži v naključni naravi algoritma MCTS ter v nepopolnosti informacije v naši igri, saj si v simulaciji pomagamo z determinizacijo, kar pomeni, da se premikamo v smeri predvidene lokacije

```
Start (X): 197
Start (detectives): 198 53 34 174 112
X coulds: [13, 26, 29, 50, 91, 94, 103, 117, 132, 138, 141, 155, 197]

Round 1
X's new location: 195 Ticket: 1
X coulds: [4, 6, 15, 16, 17, 23, 24, 27, 37, 38, 39, 41]
Detectives: 186 54 47 161 100

Round 2
X's new location: 182 Ticket: 1
X coulds: [3, 5, 7, 12, 13, 14, 15, 16, 21, 22, 23, 24, 25, 26]
Detectives: 185 55 46 160 82

Round 3
X's new location: 195 Ticket: 1
X coulds: [195]
Detectives: 153 54 78 161 101

Round 4
X's new location: 182 Ticket: 1
X coulds: [182, 194, 197]
Detectives: 180 70 79 128 82

Round 5
X's new location: 195 Ticket: 1
X coulds: [181, 183, 184, 192, 193, 195, 196]
Detectives: 181 71 111 89 140

Round 6
X's new location: 197 Ticket: 1
X coulds: [166, 167, 168, 169, 180, 182, 183, 184, 185, 190, 191, 194, 196, 197]
Detectives: 166 89 163 128 153

Round 7
X's new location: 195 Ticket: 1
X coulds: [155, 156, 165, 167, 168, 169, 170, 178, 179, 181, 182, 183, 184]
Detectives: 181 67 191 185 167

Round 8
X's new location: 197 Ticket: 1
X coulds: [197]
Detectives: 182 23 179 184 155

Round 9
X's new location: 195 Ticket: 1
X coulds: [195, 196]
Detectives: 195 13 165 196 168

Mr.X found. Detectives won.
```

Slika 5.2: Primer zapisa v konzolo pri igranju igre.

lopova. Ko je seznam možnih lokacij lopova zelo dolg, je lahko ta metoda zelo netočna. Sklepamo, da bi se zelo napreden tip simulacije obrestoval v igrah s popolno informacijo, kjer bi naša simulacija temeljila na resničnih informacijah, ne na predvidenih. Če imamo na voljo več časa, se namesto za zelo napreden tip simulacije raje odločimo za več iteracij algoritma MCTS s simulacijo z neodvisnim (individualnim) premikanjem detektiva.

Denimo, da imamo na razpolago čas t za izvajanje enega procesa MCTS. Po naših rezultatih bi se lahko v tem času izvedlo n iteracij algoritma MCTS z drugim tipom simulacije (individualni detektivi) ter približno $\frac{n}{2}$ iteracij algoritma MCTS s tretjim tipom simulacije. Ker smo ob našem delu ugotovili, da tipa simulacije prinašata podobne rezultate pri istem številu iteracij, se raje odločimo za uporabo drugega tipa simulacije, saj nam več iteracij algoritma MCTS prinaša bolj točne rezultate.

Iz tega bi lahko sklepali, da je bolje uporabiti neko pametnejšo metodo simulacije, ki pa ni preveč časovno potratna. Bolje je, da namesto preveč naprednih metod izvedemo več iteracij drevesnega preiskovanja Monte-Carlo in tako dobimo še bolj natančne rezultate.

5.4 Vpeljava konstante T

Kot zanimivost smo opravili še nekaj testov, kjer smo uporabili prag T, opisan v razdelku 3.2.5. Konstanto T smo nastavili na 50, opravili pa smo 100 iger drugega tipa (*detektivi, ki igrajo po MCTS, pri čemer uporabljajo simulacijo tipa naključno premikanje detektivov - naključno premikanje lopova proti avtomatskemu igralcu lopova, ki igra pametno.*). Brez konstante T je bila uspešnost detektivov 71% (tabela 5.1).
Zaporedna številka zmagovalne poteze se z uvedbo konstante T ni pretirano izboljšala: prej je bila enaka 10,48, pri igrah z uvedbo konstanto T pa je bila enaka 10,41. Veliko večji razkorak opazimo pri deležu zmag pri igrah z upo-

rabljeno konstanto T. Delež zmag se namreč dvigne na kar 86%. Sklepamo, da se to zgodi zaradi večje natančnosti, ki jo prinaša konstanta T. Kot smo že omenili, lahko le ena izvedba simulacije iz nekega vozlišča prinaša zelo nenatančne rezultate. Razlog za to leži v nepopolnosti informacij igre. Z določanjem minimalnega praga poskrbimo, da se iz nekega vozlišča mora izvesti določeno število simulacij, preden informacije o tem vozlišču lahko začnemo uporabljati v enačbi UCT v fazi izbiranja vozlišča. Dosežemo, da se o obetavnosti nekega vozlišča ne odločimo le na podlagi enega rezultata. Na tak način poskrbimo za kvalitetnenjše informacije, ki jih UCT uporablja.

Poglavje 6

Sklep

V diplomskem delu smo preučili algoritem drevesnega preiskovanja Monte-Carlo in ga uspešno aplicirali na namizno igro Scotland Yard. Poleg tega nam je uspelo raziskati pomen naprednejših metod simulacije v drevesnem preiskovanju Monte-Carlo. Ugotovili smo, da je dobro izbrati enostavnejšo hevristiko, ki ni preveč časovno požrešna. Če imamo na voljo več časa za izvajanje procesa MCTS, potem se raje odločimo za večje število iteracij kot pa za uporabo naprednejših metod v simulaciji.

Kljub uspešno raziskanem področju smo ob izdelavi diplomskega dela naleteli na precej možnosti za izboljšavo in nadgradnjo izdelanega.

Za avtomatskega igralca lopova proti naši MCTS tehniki smo uporabili podobno metodo, kot za premike lopova v simulaciji.
Iz tega lahko sklepamo, da bi bil ob simulaciji, ki ustreza igranju človeka, naš algoritem zelo uspešen tudi za igranje proti človeku. Tako bi bil naš naslednji izziv osredotočiti se predvsem na ponazoritev realne igre v simulaciji. To bi lahko glede na prebrano literaturo [8] dosegli na takšen način, da bi zbirali informacije o premikanju različnih igralcev in te informacije kasneje uporabili tako, da so nekatere pozicije lopova bolj verjetne kot druge. Gradili bi torej bazo podatkov, podobno tabeli 3.1, potem pa bi to tabelo uporabili v metodi

determinizacije.

Naslednja stvar, v kateri vidimo priložnost za še boljše rezultate, je iskanje primernejše konstante C pri enačbi UCT in konstante r pri *koaliciji detektivov*. V taki vrsti igre obstaja možnost, da je v določenih situacijah neka konstanta C boljša kot v drugih situacijah (npr. detektivi v okolici lopova ali pa detektivi oddaljeni od lopova). Prav tako velja tudi za konstanto r. Zanimivo bi bilo napisati ustrezno funkcijo, ki bi glede na situacijo izbrala ustrezni vrednosti teh dveh konstant.

Možnost izboljšav leži tudi v testiranju alternativnega premikanja v simulaciji. Detektive bi lahko poskusili premikati tudi po načelu minimizacije vsote števila potez, ki bi jih moral opraviti detektiv, da bi prišel do posamičnih lokacij. Sklepamo, da bi bil ta način precej časovno zahteven.

Literatura

[1] Guillaume Chaslot. Monte-carlo tree search. *Maastricht: Universiteit Maastricht*, 2010.

[2] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*, pages 72–83. Springer, 2007.

[3] Mark H. M. Winands Jos W. H. M. Uiterwijk Guillaume Chaslot, H. Jaap Van Den Herik and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

[4] Projekt Team III. *Scotland Yard*. Ravensburger Spieleverlag, 1998.

[5] Sensei's Library. Historical kgs ratings of a few bots, 2007. [http://senseis.xmp.net/?KGSBotRatings; obiskano 15. avgust 2015].

[6] Pim J. A. M. Nijssen. *Monte-Carlo Tree Search for Multi-player games*. PhD thesis, 2013.

[7] Pim J. A. M. Nijssen and Mark H. M. Winands. Enhancements for multi-player monte-carlo tree search. In *Computers and Games*, pages 238–249. Springer, 2011.

[8] Pim J. A. M. Nijssen and Mark H. M. Winands. Monte-carlo tree search for the game of scotland yard. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 158–165. IEEE, 2011.

[9] Pim J. A. M. Nijssen and Mark 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, 2012.

[10] Robin. Heuristic search, 2009. [http://intelligence.worldofcomputing.net/ai-search/heuristic-search.html; obiskano 15. avgust 2015].

[11] Nathan R. Sturtevant. An analysis of uct in multi-player games. In *Computers and Games*, pages 37–49. Springer, 2008.

[12] Fabien Teytaud and Olivier Teytaud. On the huge benefit of decisive moves in monte-carlo tree search algorithms. In *IEEE Conference on Computational Intelligence and Games*, 2010.

[13] Alexander Useche. Search algorithms: Breadth first search, 2014. [http://www.alexanderuseche.com/search-algorithms-breadth-first-search/; obiskano 15. avgust 2015].

[14] Mark H. M. Winands and Yngvi Bjornsson. $\alpha\beta$-based play-outs in monte-carlo tree search. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 110–117. IEEE, 2011.

[15] Kazuki Yoshizoe, Akihiro Kishimoto, Tomoyuki Kaneko, Haruhiro Yoshimoto, and Yutaka Ishikawa. Scalable distributed monte-carlo tree search. In *Fourth Annual Symposium on Combinatorial Search*, 2011.