Univerza v Ljubljani
Fakulteta za matematiko in fiziko
Fakulteta za računalništvo in informatiko
Računalništvo in matematika - 2. stopnja

Vito Janko

Razvoj programa za igranje 1-2-3 šaha

magistrsko delo

Mentor: doc. dr. Matej Guid

Ljubljana, 2015

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Vito Janko sem avtor magistrskega dela z naslovom:

Razvoj programa za igranje 1-2-3 šaha

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom doc. dr. Mateja Guida,
- so elektronska oblika magistrskega dela, naslov (slovenski, angleški), povzetek (slovenski, angleški) ter ključne besede (slovenske, angleške) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela.

V Ljubljani, 31. avgusta 2015 Podpis avtorja:

Zahvala

Svojemu mentorju dr. Mateju Guidu se zahvaljujem za odlično predlagano temo ter za vso pomoč in nasvete, brez katerih naloge ne bi tako uspešno opravil.

Zahvalil bi se rad tudi družini in prijateljem, ki so me tekom študija podpirali in spodbujali.

Vito Janko, 2015

Kazalo

Program dela

Pred sabo imamo začetno pozicijo na šahovnici, figure pa se premikajo po enakih pravilih kot pri običajnem igranju šaha. Z eno pomembno razliko: beli najprej odigra eno potezo, črni odgovori z dvema potezama, beli nato izvede tri poteze... in tako dalje, vse do matiranja nasprotnika (ali pa do remija). Ta različica šaha se v angleščini imenuje Progressive chess, mi pa ga imenujmo kar "1-2-3 šah". Za to igro je značilna kombinatorična eksplozija možnosti, kjer sodobne metode in algoritmi s področja igranja iger brez dodatnih izboljšav in namenskih hevristik najverjetneje ne bodo obrodili sadov. Z vidika umetne inteligence predstavlja zanimiv izziv: kako implementirati program, ki bo za človeka tako rekoč nepremagljiv v tej igri?

Kandidatova naloga je razviti program za igranje omenjene igre. Program naj sledi splošno priporočeni strategiji za to igro: (1) poiščimo mat, če je le-ta izvedljiv v danem številu potez, (2) v primeru, da mata ni, poiščimo zaporedje čim bolj koristnih potez, (3) preverimo, ali načrtovano zaporedje potez preprečuje nasprotnikov mat (sicer poiščemo drugo). Program naj temelji na hevrističnem preiskovanju. Naloga vključuje oblikovanje ustreznih hevristik in prilagoditev obstoječih algoritmov zahtevam te igre. Kakovost razvitih hevristik in algoritmov eksperimentalno ovrednotite. Implementirajte tudi grafični vmesnik, ki bo človeškim igralcem omogočil igranje "1-2-3 šaha"in jim pomagal pri analizi odigranih partij.

V Ljubljani, 31. avgusta 2015 Podpis mentorja:

Povzetek

V tem delu predstavimo program za igranje igre 1-2-3 šah (ang. *Progressive chess*). Pri tej šahovski različici beli začne z eno potezo, črni nadaljuje z dvema, beli nato odigra tri in tako naprej. Zaporedne poteze lahko igralec odigra na ogromno načinov in ta kombinatorična eksplozija možnosti predstavlja velik izziv za klasične metode, namenjene igranju tovrstnih iger. Naš program sledi splošno priporočeni strategiji za to igro: (1) poišče mat, če je le-ta izvedljiv v danem številu potez, (2) v primeru, da mata ni, poišče potencialno dobra zaporedja potez, (3) odigra najbolje ocenjeno zaporedje potez, ki hkrati preprečuje nasprotnikov mat.

Za iskanje matov (in sorodno, preprečevanje le-teh) smo osnovali pet različnih hevristik in preiskusili dva preiskovalna algoritma, nato pa vse skupaj preiskusili na avtomatsko pridobljeni množici testnih šahovskih pozicij. Za iskanje dobrih zaporedij potez smo razvili in predstavili drug nabor hevristik in empirično pokazali njihovo koristnost. Uporabili smo kombinacijo algoritmov A* in Min-Max, kar nam je omogočilo spopad z ogromno kombinatorično zahtevnostjo domene. Pripravili smo otvoritveno knjižico in osnovali namenske hevristike za končnice. Razvili smo program z grafičnim vmesnikom, ki med drugim omogoča igranje proti računalniškemu nasprotniku ter iskanje matov iz poljubne vnešene pozicije.

Ključne besede

šah, 1-2-3 šah, hevristično preiskovanje, hevristike, algoritem A, algoritem Min-Max, iskanje matov, kombinatorična zahtevnost*

Math. Subj. Class. (2010): 68T20

Abstract

We present the design of a computer program for playing Progressive chess. In this game, rather than just making one move per turn, players play progressively longer series of moves. Combinatorical complexity generated by many sequential moves presents a difficult challenge for classic search algoritms. Our program follows the generally recommended strategy for this game, which consists of three phases: (1) looking for possibilities to checkmate the opponent, (2) playing generally good moves when no checkmate can be found, and (3) preventing checkmates from the opponent.

For the task of finding (and preventing) checkmates we considered two versions of the A* algorithm, and developed five different heuristics for guiding the search. An automatically generated set of chess positions was used to evaluate the efficiency of checkmate search. For the second phase of the game we developed another set of heuristics, and combined the A* algorithm with Min-Max search, in order to fight the combinatorical complexity. We constructed an opening book, and designed specialised heuristics for playing Progressive chess endgames.

We developed an application with a graphical user interface that enables human players to play Progressive chess against the computer, and is able to search for checkmates in any arbitrarily given chess position.

Keywords

chess, Progressive chess, heuristic search, heuristics, A algorithm, Min-Max search, checkmate search, combinatorial complexity.*

Poglavje 1

Uvod

Šahovske različice so družina strateških namiznih iger, ki so povezane z, navdahnjene po ali podobne klasični igri šaha. 1-2-3 šah (angl. *Progressive chess*) je različica šaha, kjer beli začne z eno potezo, nato črni odigra dve, beli nadaljuje s tremi in tako naprej. V svojem krogu imata igralca tako na voljo vedno daljšo sekvenco potez.

Gre za eno bolj popularnih šahovskih različic [1]; zanjo je bilo v zadnjih petdesetih letih organiziranih na stotine turnirjev [2]. O sami igri je bilo napisanih tudi nekaj knjig [3, 4, 5, 6].

Cilj tega magistrskega dela je bil razviti in predstaviti program z grafičnim vmesnikom, ki bi to različico znal dostojno igrati in v njej konkurirati človeškemu igralcu. Programa za igranje 1-2-3 šaha doslej ni razvil še nihče. Med razvojem programa smo preiskusili več algoritmov in hevristik za preiskovanje ter jih primerjali med seboj. Glede na testiranja smo najuspešnejšo kombinacijo implementirali v zadnji različici programa, ki je prosto dostopna na spletu [7].

Več zaporednih potez v tej igri omogoča igralcu ogromno možnosti kako odigrati svoj *krog* (vse zaporedne poteze enega igralca tvorijo njegov krog). Ta razvejanost privede do ogromne kombinatorične eksplozije vseh možnosti in predstavlja največji izziv pri razvoju tega programa. Prevelika razvejanost je onemogočila uporabo klasičnih pristopov za igranje tovrstnih iger ter nas

spodbudila, da jih nekoliko preoblikujemo in prilagodimo tej domeni.

Druga ovira je splošno pomanjkanje znanja o tej igri, saj je ta v primerjavi s klasičnim šahom še relativno neraziskana. To dejstvo oteži izbiranje dobrih hevristik za preiskovanje, zato bo njihovo iskanje imelo velik poudarek v tem delu.

Upamo, da bo naše delo na tej temi ponovno oživelo zanimanje za to igro ter predstavilo težavno novo domeno, v kateri bodo lahko tudi drugi raziskovalci testirali svoje algoritme, pristope in ideje.

1.1 Pregled magistrskega dela

Najprej si bomo igro podrobneje ogledali. V poglavju 2.1 bomo navedli vsa pravila te različice in predstavili nekaj njenih lastnosti. Predstavili bomo tudi nekaj primerov iz poteka same igre, v upanju da bodo bralcu ilustrirali izzive, ki se pri njej pojavljajo.

Nato bomo v poglavju 3 predstavili naš program in njegove funkcionalnosti ter okvirno začrtali delovanje računalniškega igralca. V splošnem je njegovo igranje sestavljeno iz dveh podproblemov; iskanja matov in iskanja sicer najboljših potez. Oba podproblema si bomo natančneje ogledali v kasnejših poglavjih.

Več zaporednih potez omogoča veliko različnih matnih vzorcev in obstoj matov v presenetljivo veliko šahovskih pozicijah. Učinkovito iskanje matov zato postane ključnega pomena za igranje te različice in ta podproblem lahko vzamemo za samosvoj raziskovalni problem. Ogledali si ga bomo v poglavju 4, kjer bomo predstavili predlagane rešitve, jih testirali in primerjali med sabo. Različne rešitve bomo preiskusili tudi v sami igri in opazovali, kako kakovost iskalca matov vpliva na kakovost igre.

Za spopad z razvejanostjo posameznih krogov smo v poglavju 5 predlagali več algoritmov za preiskovanje. Predlagali smo tudi več hevristik ter njihovo medsebojno kombinacijo. Različne možnosti smo nato s pomočjo iger, kjer je program igral sam s sabo (ang. *self-play*), primerjali med seboj.

V poglavju 6 predstavimo še otvoritve in končnice te igre. Podobno kot pri navadnem šahu lahko v teh delih igre uporabimo zanje specifična znanja in z njimi izboljšamo kvaliteto igre. Pri otvoritvah uporabimo vnaprej pripravljeno otvoritveno knjižico, pri končnicah pa namenske hevristike.

V dodatku podamo še nekaj zanimivih implementacijskih podrobnosti, kot je implementacija šahovnice z bitnimi polji (ang. *Bitboards*) ter za šah specifične zgoščevalne funkcije.

1.2 Predhodna objava

Del tega magistrskega dela je bil objavljen v naslednjem znanstvenem članku:

Janko Vito, Guid Matej. Development of a Program for Playing Progressive Chess. The 14th International Conference on Advances in Computer Games (ACG 2015). Julij, 2015.

Rezultati v tem delu so nadgradnja in razširitev omenjene študije.

Poglavje 2

Predstavitev igre 1-2-3 šah

V tem poglavju predstavimo igro 1-2-3 šah, njena pravila, lastnosti in primere iz same igre. Upamo, da bo podrobnejše razumevanje igre bralcu omogočilo boljše razumevanje preostanka tega dela.

2.1 Pravila igre

V tej igri, namesto da oba igralca odigrata eno potezo v svojem krogu, se sekvenca zaporednih potez povečuje. Beli začne z eno potezo, nato črni odigra dve, nato spet beli odigra tri in tako naprej. Pravila te različice šaha so sicer enaka običajnim, vendar vključujejo naslednje izjeme:

- šah je mogoče dati le v zadnji potezi v svojem krogu,
- igralec v nobeni svoji potezi ne sme premakniti svojega kralja v šah,
- kralj se mora šahu umakniti v prvi potezi svojega kroga; če tega ne more narediti, je matiran (izgubi igro),
- če igralec kadarkoli v svojem krogu nima legalne poteze, potem se igra zaključi z remijem (neodločen rezultat),
- *En passant* pravilo lahko igralec izkoristi le v prvi potezi svojega kroga.

Obstajata dve glavni različici 1-2-3 šaha: italijanska in škotska. Prva je bila bolj raziskana in zanjo je bila narejena tudi večja baza odigranih iger (imenovana "PRBASE"). V italjanski verziji (za katero smo se v tem delu odločili) je šah lahko narejen samo v zadnji potezi svojega kroga. Poudarimo, da če se lahko v neki poziciji izognemo šahu le tako, da šah damo nasprotniku, potem je tak način neveljaven in je igra izgubljena (primer na sliki 4.4). V škotski verziji lahko šah damo v poljubni potezi, vendar s to potezo zaključimo svoj krog. Pokazano je bilo, da razlika v različicah redko naredi razliko v rezultatu igre, saj predčasen šah navadno pripelje do poraza za igralca, ki šahira [8].

2.2 Lastnosti igre

1-2-3 šah si deli veliko lastnosti z običajnim šahom. Je igra za dva igralca, ki je končna (*finite*), zaporedna (*sequential*), deterministična (*deterministic*), s popolno informacijo (*perfect information*) in ničelnim izidom (*zero-sum*). Časovno-prostorska kompleksnost igre (definirana kot število stanj igre, ki jih lahko dosežemo z legalno igro) je podobna navadnemu šahu, kjer je ocenjena na okoli 10^{46} [9].

Glavna razlika leži v ekstremno velikem faktorju razvejanosti. Na primer, v sedmem krogu lahko igralec odigra poljubnih sedem zaporednih potez, kar mu da približno 20^7 možnih načinov kako odigrati svoj krog.

V drugi različici šaha, imenovani "Arimaa", ima igralec "le"4 poteze na krog in ocenjeno povprečno razvejanost okoli 16.000 [10]. V tej igri so ljudje do sedaj še vedno prevladali v letnemu "Arimaa challenge"tekmovanju in prav razvejanost (in posledična kombinatorična zahtevnost preiskovanja) naj bi bila glavna ovira pri razvoju uspešnega programa za igranje te različice [11].

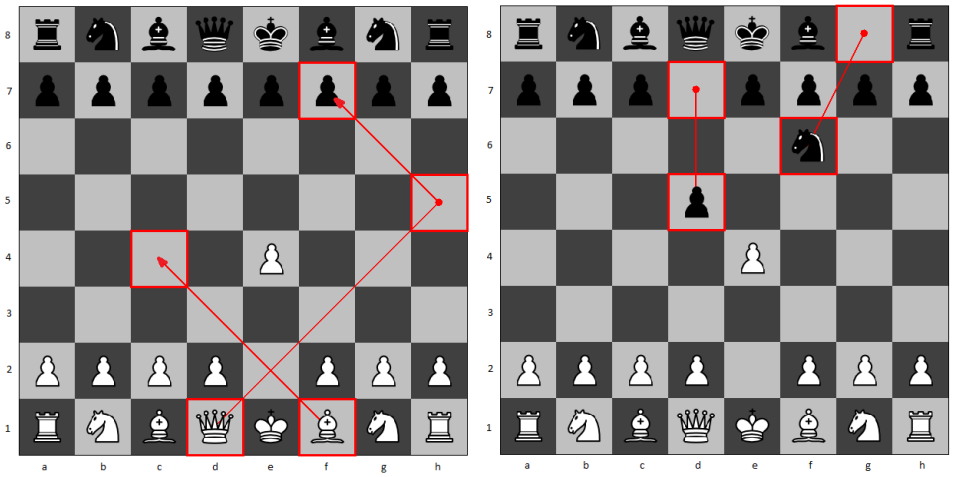
1-2-3 šah s še bistveno večjo razvejanostjo predstavlja novo, težavno domeno, kjer standardni algoritmi za igranje šaha in podobnih iger ne delujejo.

2.3 Potek igre

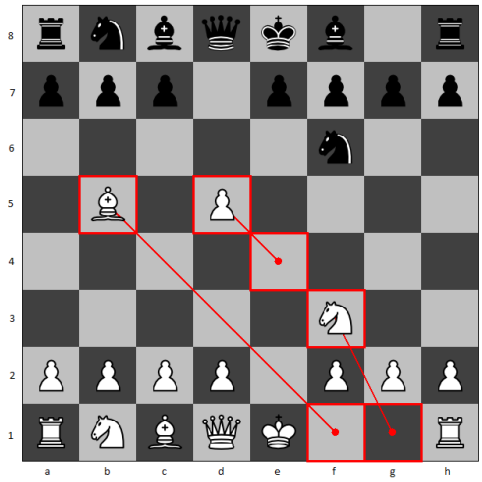
V tem poglavju podamo nekaj primerov pozicij iz igre in nekaj razmislekov o odločitvah narejenih v njih.

Strategija igre za oba igralca je lahko povzeta na naslednji način. Najprej poišči potencialni mat; če ga ne najdeš, poskušaj preprečiti nasprotnikov mat v naslednjem krogu. Nato poskušaj vzeti nasprotnikove najmočnejše figure in kar se da obvarovati svoje.

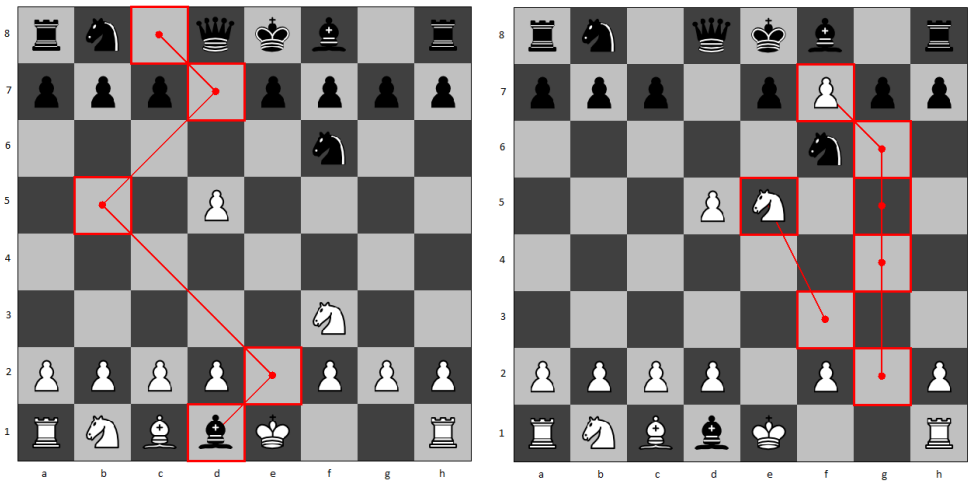
Iskanje matov je najpomembnejša naloga igralca v tej igri, saj se zaradi velikega števila zaporednih potez potencialni mati pojavljajo v mnogih pozicijah. V poglavju 4.4 bomo pokazali, da lahko samo s povečanjem sposobnosti iskalca matov drastično povečamo pričakovano število zmaganih iger. Primeri, kako hitro lahko pridemo do matne pozicije, so ilustrirani na slikah 2.1, 2.2, 2.3, 2.4 in 2.5.



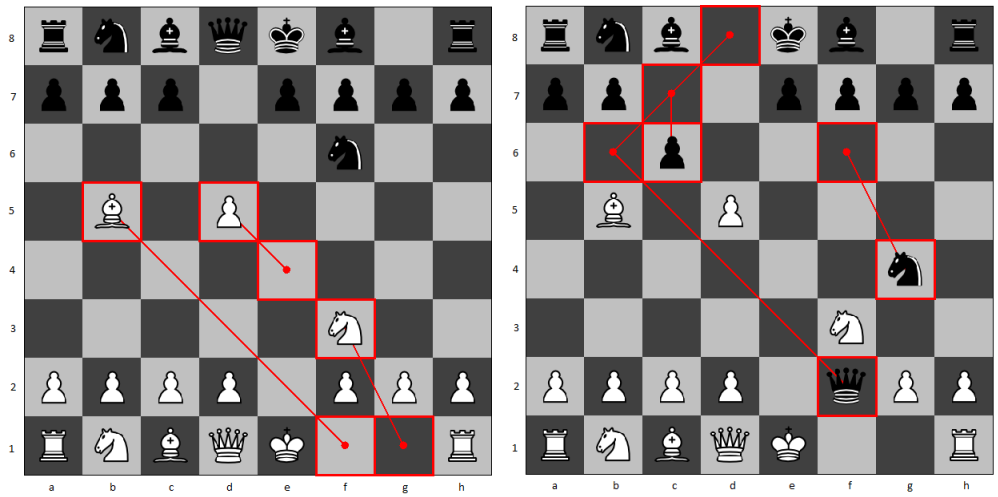
Slika 2.1: Beli je otvoril igro s potezo 1.e2-e4. V naslednjem krogu belega bo ta imel 3 poteze in matno grožnjo 3.Lf1-c4 Dd1-h5, Dh5-f7#, prikazano na levi. Črni ta mat ustavi z igranjem 2.d2-d4, Sg8-f6. Črni kmet na polju d4 blokira diagonalo lovca in prej opisan mat onemogoči. Opomba: nasprotnikovega kralja se nikoli ne da vzeti, saj bi to povzročilo nelegalen šah sredi sekvence (beli ne more igrati 3.Dd1-h5 Dh5-f7+ Df7-e8).



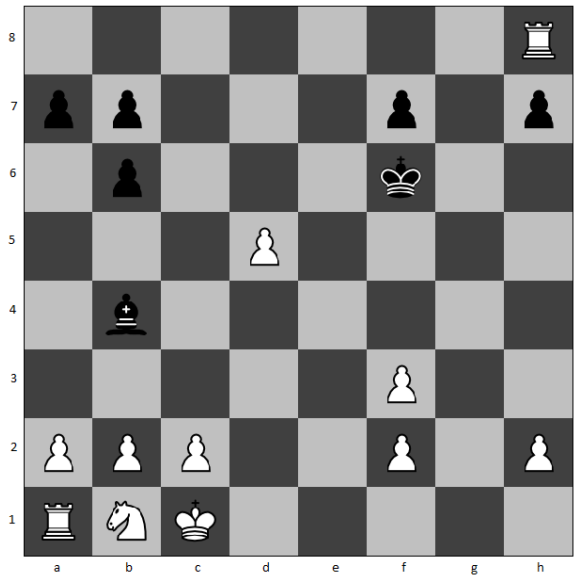
Slika 2.2: Beli nadaljuje igro iz slike 2.1 in odigra sekvenco 3.Sg1-f3 e4xd5 Lf1-b5+. Z njo razvije dve figuri, vzame črnega kmeta in v zadnji potezi da šah. Tak šah je koristen, saj mora sedaj črni prvo potezo svojega kroga porabiti za blokiranje diagonale med nasprotnim lovcem in svojim kraljem.



Slika 2.3: Črni nadaljuje igro iz slike 2.2 in odigra slabo sekvenco 4.Lc8-d7 Ld7-b5 Lb5-e2 Le2-d1, ki sicer izgleda obetavna: vzame tako lovca kot kraljico (prikazano na levi), vendar lahko beli nanjo odgovori z matom 5.Sf3-e5 g2-g4 g4-g5 g5-g6 g6xf7# (prikazano na desni).



Slika 2.4: Črni je v poziciji iz slike 2.2 (prikazana na levi) imel boljšo alternativo. Igra lahko kar zmagovalno sekvenco: 4.c7-c6 Sf6-g4 Dd8-b6 Db6-f2# (prikazano na desni). Opomnimo, da je črni sekvenco moral začeti s c6, saj se je šahu treba izogniti s prvo potezo sekvence.

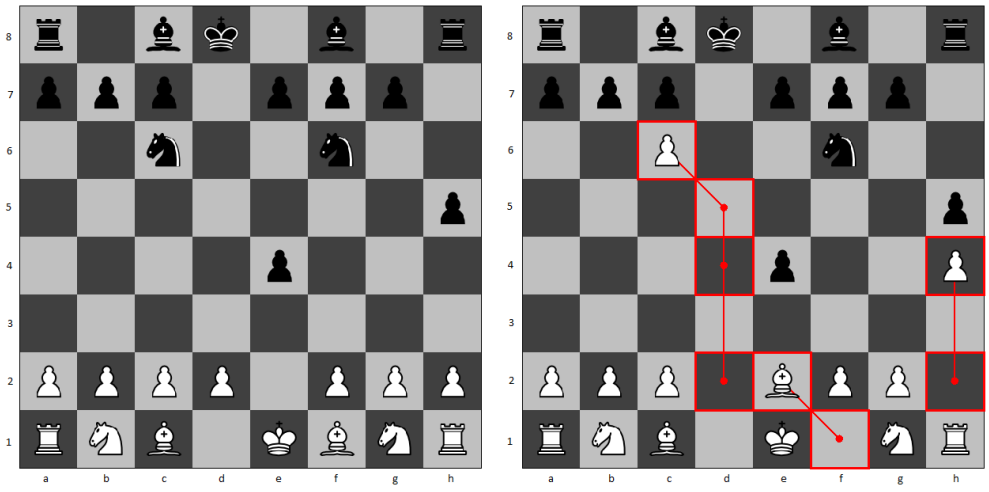


Slika 2.5: Črni na potezi, mat v 8. Rešitev na voljo na sliki 3.2.

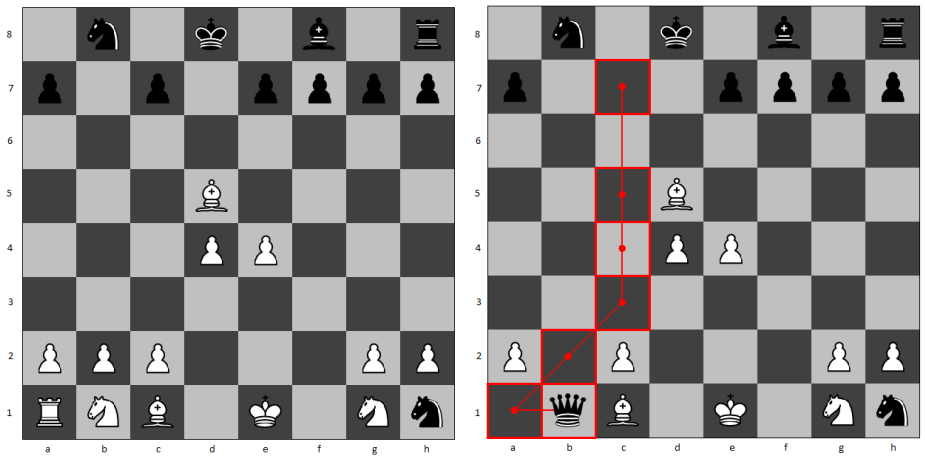
Prikazane pozicije so ponazorile nujnost dobrih zmožnosti iskanja matov že od otvoritve dalje. Vsak krog je potrebno preračunati, ali lahko matiramo, ter kako preprečiti nasprotnikov mat.

Kadar se igra ne konča z nenadnim matom, se navadno prelevi v čim hitrejše jemanje materiala. Zaporedne poteze omogočajo jemanje več figur na krog, tudi če so nekatere izmed njih branjene. Že na sliki 2.5 je opaziti, da je šahovnica relativno prazna, kljub temu da je minilo le sedem krogov. Dodatno omenimo, da kmetje lahko promovirajo v petih potezah (če njihova pot ni blokirana) in lahko igralec enostavno pride do nove kraljice (primer na sliki 2.7).

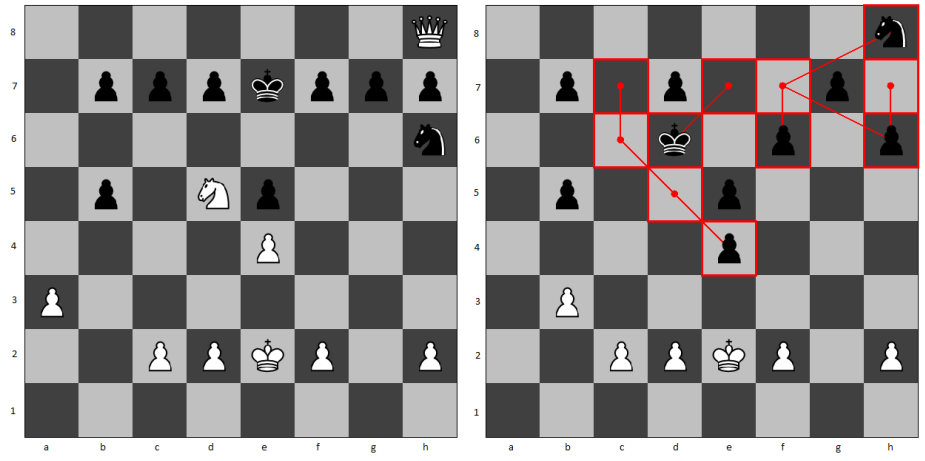
Igra se navadno konča, ko eni strani zmanjka materiala (primer na sliki 2.8), nakar druga stran na primer promovira v kraljico ter z njo v nekaj krogih matira. Vzorec matiranja s kraljico ali kakšno drugo kombinacijo figur je opisan v poglavju 6.2.



Slika 2.6: Beli je na vrsti s petimi potezami. Mata se v tej poziciji ne da narediti. Beli odigra 5.d2-d4 d4-d5 d5xc6... S tem vzame črnega skakača, in svojega kmeta spremeni v večjo grožnjo. Nadaljuje s potezo h2-h4, kar prepreči promocijo črnega kmeta na polju h5. Krog konča s potezo Lf1-e2, s tem zaščiti svojega kralja pred nasprotnikovim matom.



Slika 2.7: Slika prikazuje pogost vzorec, kjer lahko igralec v svojem krogu pripelje kmeta čez celotno šahovnico in ga tako promovira v kraljico. Pri tem vzame še nekaj nasprotnikovih figur.

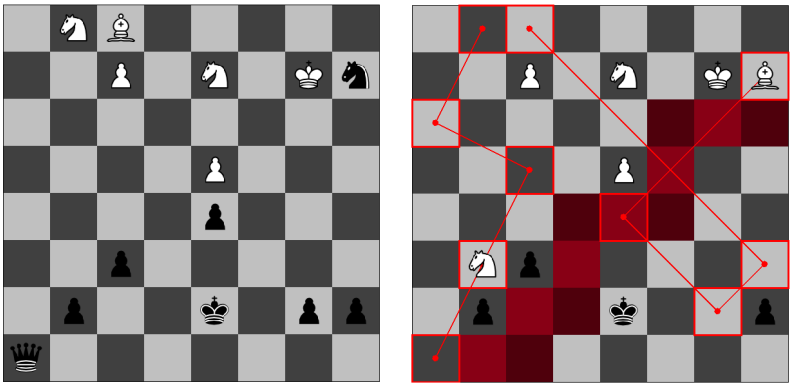


Slika 2.8: Črni v poziciji na levi lahko igra sekvenco osmih potez. Na prvi pogled izgleda, da se je znašel v težavah, saj je v šahu, beli pa ima še dodatno kraljico, vendar sta oba problema enostavno rešljiva. Odigra 8.Ke7-Kd6 f7-f6 Sh6-f7 Sf7-h8 c7-c6 c6xd5 d5xe4 h7-h6 (prikazano na desni). Beli je sedaj izgubljen; nima nobene figure in promovirati ne more nobenega kmeta. Potezi h7-h6 in Ke7-Kd6 sta bili ključni pri ustavljanju možnih promocij belega.

Tabela 2.1: Zapis celotne igre

Poteze
1.d2-d4
2.d7-d5 c7-c6
3.Lc1-g5 Lg5xe7 Le7xd8
4.Lc8-f5 Lf5xc2 Lc2xd1 Ke8xd8
5.Ke1xd1 e2-e4 e4xd5 Kd1-c2 b2-b4
6.c6xd5 Sb8-c6 Kd8-d7 h7-h5 Ta8-c8 Sc6xb4+
7.Kc2-b2 Sb1-d2 Ta1-c1 Tc1xc8 Tc8xf8 Tf8xg8 Tg8xh8
8.h5-h4 h4-h3 h3xg2 g2xh1=D Dh1xh2 Dh2xh8 f7-f6 Sb4xa2
9.Sg1-h3 Sh3-f4 Sf4-g6 Lf1-a6 La6xb7 Lb7xd5 Ld5xa2 Sg6xh8 Sh8-g6

Tabela 2.1 prikazuje še primer igre iz poskusnega dvoboja s FIDE mojstrom šaha, sicer pa tudi izkušenim igralcem igre 1-2-3 šah. Program je zmagal vseh šest iger v dvoboju, kjer je program imel na voljo 60 sekund na potezo, igralec pa 10 minut za celotno partijo. Končna pozicija, kjer se je črni (igralec) predal, je še posebej zanimiva in je prikazana na sliki 2.9.



Slika 2.9: Beli je v poziciji na levi lahko igral sekvenco devetih potez. V svoji sekvenci je uspel ustaviti vse promocije in bistveno omejil črnega kralja. Rdeče pobarvana polja so napadena s strani figur belega igralca. Vidimo, da ti napadi preprečujejo kralju, da bi se približal katerikoli beli figuri.

Poglavje 3

Opis programa

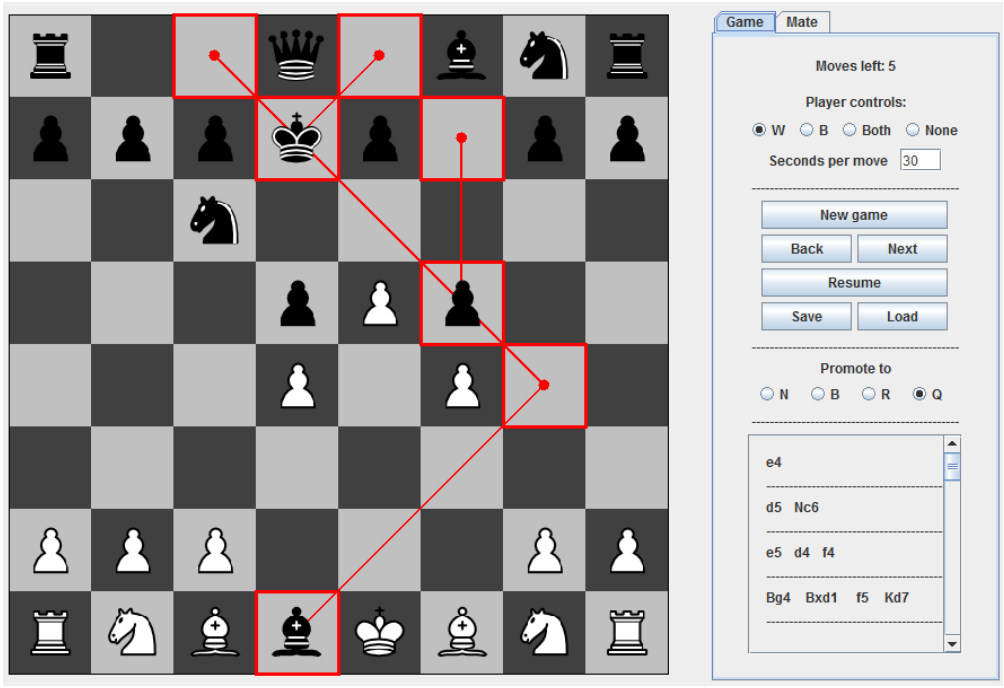
Osrednji cilj te magistrske naloge je bil razvoj programa za igranja 1-2-3 šaha. V tem poglavju naš program predstavimo, naštejemo njegove funkcionalnosti in opišemo njegovo delovanje. Njegova zadnja različica, ki implementira najuspešnejše v tem magistrskem delu predstavljene pristope, je na voljo na spletu[7].

3.1 Grafični vmesnik

Program uporablja italijansko verzijo pravil (za podrobnosti glej poglavje 2.1) in ima naslednje funkcionalnosti:

- igranje proti računalniku ali drugemu človeškemu nasprotniku,
- gledanje igre računalnika proti samemu sebi,
- iskanje matov v dani poziciji,
- vnos poljubne pozicije na šahovnico,
- shranjevanje igre,
- nalaganje in gledanje shranjenih pozicij.

Grafični vmesnik je prikazan na sliki 3.1 in sliki 3.2.



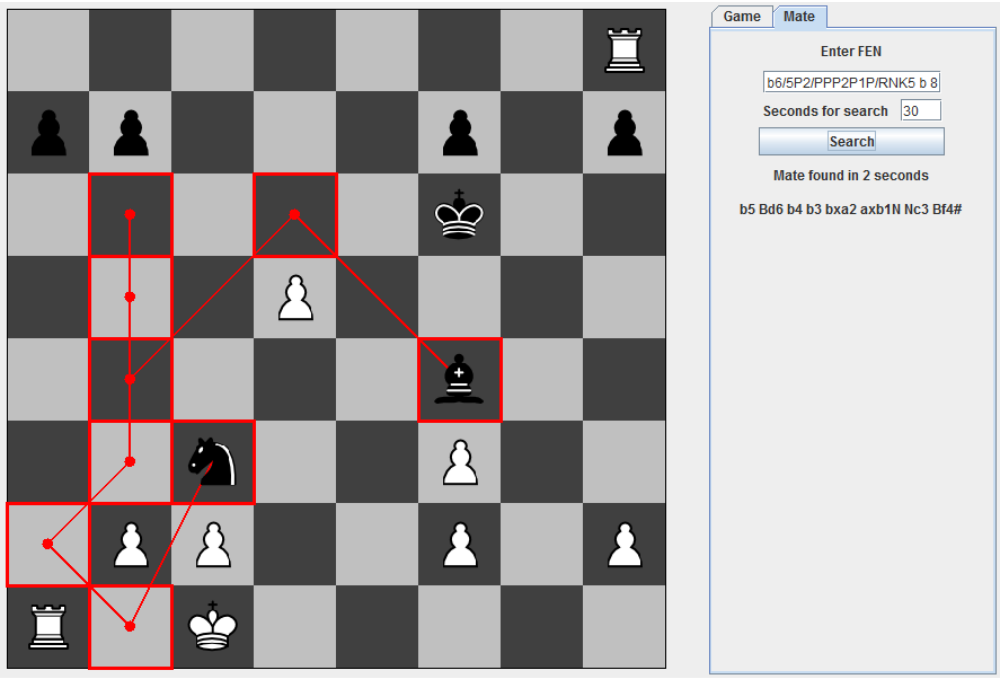
Slika 3.1: Grafični vmesnik našega programa za igranje 1-2-3 šaha. Na levi je trenutna pozicija, z rdečo so označene poteze zadnjega kroga. Na desni so prikazani status in zgodovina igre ter gumbi za spreminjanje njenega poteka: nova igra, shranjevanje in nalaganje shranjene igre, razveljavitev poteze ter sprememba barve figur, ki jih nadzoruje človeški igralec.

3.2 Shema delovanja računalniškega igralca

Pri odločanju o svoji potezi mora računalniški igralec preiskovati prostor svojih možnosti: katere vse sekvence potez ima na voljo, možne odgovore nasprotnika itd... Visoka stopnja razvejanosti posameznega kroga predstavlja veliko oviro za klasične pristope preiskovanja (več o tem v poglavju 5.2), zato smo preiskovanje morali prilagoditi specifikam te domene. Izračunavanje potez v njegovem krogu poteka v treh fazah:

1. Iskanje mata v dani poziciji.

2. Iskanje nekaj najboljših sekvenc potez, če mata nismo našli.



Slika 3.2: Grafični vmesnik našega programa za igranje 1-2-3 šaha. Omogočena sta vnos pozicije v FEN [12] notaciji in iskanje mata na njej. Najden mat (8.Lb4-d6 b6-b5 b5-b4 b4-b3 b3xa2 a2xb1=S Sb1-c3 Ld6-f4#) je prikazan na šahovnici. Ta mat je rešitev za pozicijo iz slike 2.5.

3. Preprečevanja mata nasprotniku.

Taka ločitev preiskovanja se precej ujema s človeškim igranjem te igre. V tej igri so namreč napadalne poteze tipično nekoristne, razen seveda če vodijo do mata. Iskanje najboljših potez je zato zelo odvisno od obstoja mata v dani poziciji ter je v fazi iskanja matov realizirano z drugačnim algoritmom in z drugimi hevristikami, kot v fazi iskanja najboljših sekvenc potez v primeru, ko mata ni oziroma ga nismo našli.

3.2.1 Prva faza: iskanje matov

V prvi fazi poskušamo najti mat v trenutni poziciji, kar za razliko od klasičnega šaha ni enostaven problem. Pri tej igri (z izjemo prvih nekaj krogov, kjer so

sekvence krajše) ni izvedljivo, da bi v realnih časovnih omejitvah generirali vse sekvence, ki jih imamo v svojem krogu na voljo. Mata torej ne moremo najti kar z izčrpnim preiskovanjem vseh možnosti in se moramo zato zateči k njihovemu hevrističnemu preiskovanju. Z reševanjem tega problema se ukvarja poglavje 4. V primeru, da mat uspešno najdemo, najdene poteze odigramo in zaključimo preiskovanje. Sicer nadaljujemo z drugo fazo.

3.2.2 Druga faza: iskanje najboljše sekvence

V drugi fazi poskusimo najti določeno število kar se da dobrih sekvenc potez, pri čemer mata eksplicitno ne iščemo (morebitni mati bi morali biti najdeni že v prvi fazi). S tem, kaj določa dobro sekvenco potez ter kako jih poiščemo, se ukvarja poglavje 5. Najdene dobre sekvence razvrstimo glede na njihovo hevristično oceno in jih uporabimo v tretji fazi.

3.2.3 Tretja faza: preprečevanje matov

V tretji fazi vzamemo najboljšo izmed najdenih sekvenc in preostali dani čas za krog porabimo za preverjanje, ali nasprotnik lahko odgovori z matom. Uporablja se isti iskalnik matov, kakor v prvi fazi. V primeru, da je mat najden, vzamemo drugo najboljšo sekvenco in jo preverimo. Če tudi ta dovoli nasprotnikov mat, vzamemo naslednjo in tako naprej. Naš iskalnik mata lahko mat potrdi, ne more pa ga ovreči, razen z izčrpnim preiskovanjem vseh možnosti (kar je v večini praktičnih primerov neizvedljivo). Ta problem prepuščamo nadaljnjemu delu. Program ob izteku časa odigra najboljše ocenjeno sekvenco v kateri ni našel mata. Ta način omogoča, da potencialno dolgotrajno preprečevanje matov opravimo s preiskovanjem najmanjšega možnega števila pozicij.

Poglavje 4

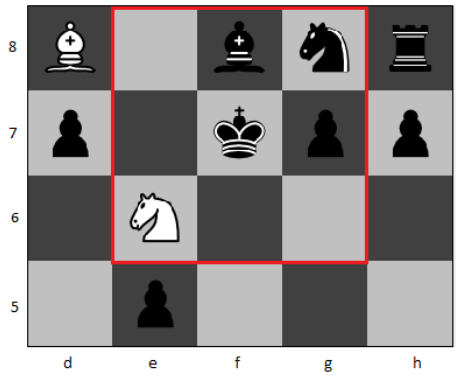
Iskanje matov

To poglavje se ukvarja s problemom iskanja matov, najverjetneje najpomembnejšim delom te igre. Nanj lahko gledamo kot problem preiskovanja z enim igralcem (ang. *single-agent search*), saj v tem delu nimamo interakcije z nasprotnikom. Testirali bomo sposobnost algoritma, da v dani poziciji najde mat. Alternativno bi si lahko zadali nalogo najti vse mate v poziciji, ali definitivno določiti, da mat ne obstaja brez izčrpnega preiskovanja vseh možnosti, vendar naša izbira bolje opiše potrebe igre. Poskusili bomo dve različici A* algoritma in pet različnih hevristik za usmerjenje iskanja.

4.1 Matne hevristike

Potreben (ampak ne zadosten) pogoj za mat je pokritost vseh polj okoli kralja, vključno s kraljem samim. V nadaljevanju bomo ta polja imenovali *matni kvadrat*. Pokritost pomeni, da na tem polju že stoji kaka nasprotnikova figura, ali pa da to polje napada naša figura (polje mora napadati, ne le stati na njemu). Primer je prikazan na sliki 4.1. Zaradi tega opažanja se vse naše hevristike ubadajo s tem, kako ta kvadrat najhitreje pokriti. Drugi pogoj za mat je, da figura, ki napada kralja, ne more biti vzeta ali blokirana. Ugotavljanje tega je računsko zahtevno za hevristiko, ki naj bo čim hitrejša, in zato ta pogoj v hevristikah ni upoštevan.

Opomba: nekatere spodaj naštete hevristike je treba minimizirati, denimo *Manhattansko*, nekatere pa maksimizirati, denimo *Pokritost*; za samo implementacijo to ni pomembno, saj lahko eno pomnožimo z (-1) in bosta obe delovali v isti smeri.



Slika 4.1: Z rdečo je označen *matni kvadrat*. V njem so z lovcem pokrita polja e7 in f6 ter s črnimi figurami polja f8, g8, g7.

4.1.1 Manhattanska razdalja

Prva hevristika, poimenovana *Manhattan*, šteje vsoto vseh Manhattanskih razdalj med svojimi figurami do nasprotnikovega matnega kvadrata.

$$h(x) = \sum_{f \in \text{figure}} \text{Manhattanska razdalja}(f, \text{matni kvadrat})$$

Pod *figure*, tukaj in povsod v nadaljevanju, štejemo naše figure brez kmetov. Na primeru 4.1 bi ta razdalja bila:

$$h(x) = 0 \text{ (za skakača) } + 1 \text{ (za lovca) } + \text{ razdalje ostalih figur izven slike}$$

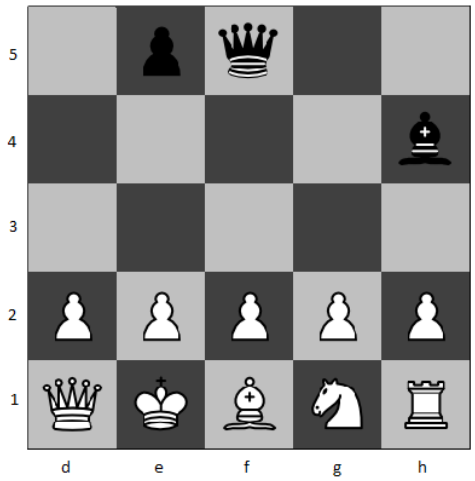
Gre za hevristiko, ki je relativno enostavno izračunljiva, in vodi iskanje v pozicije, kjer so naše figure čim bližje nasprotnikovemu kralju. Seveda hevristika ignorira dejansko premikanje figur - te se ne obnašajo Manhattansko.

4.1.2 Pokritost matnega kvadrata

Naslednja hevristika *Pokritost* šteje, koliko polj v matnem kvadratu je že pokritih. Na primeru 4.1 je to število 5. Je relativno hitra za izračun in daje zelo dobre ocene, koliko blizu matu smo. Vprašanje je, ali je smiselno šteti pokritost dvojno, če je polje napadeno dvakrat (in večkrat za vsak napad) ter ali šteti nasprotnikove figure v kvadratu kot pokrivajoče ali ne. Izkazalo se je, da se večkratno pokritje in ignoriranje nasprotnikovih figur pri štetju pokritosti izplača, saj omogoča lažje najti pogoste mate oblike vidne na sliki 4.2.

$$h(x) = \sum_{p \in \text{matni kvadrat}} \text{število naših figur, ki pokriva polje } p$$

V praksi se je ta hevristika izkazala za daleč najboljšo in je bila uporabljena v končni verziji našega šahovskega programa.



Slika 4.2: Črni je med preiskovanjem naletel na to pozicijo, kjer bo imel priložnost matiranja v naslednji potezi na polju f2. Želeli bi, da hevristika to pozicijo opiše kot zelo obetavno. V ta namen naj hevristka prepozna napada tako lovca kot kraljice za koristna, kljub temu da pokrivata isto polje, in kljub temu, da je to polje že zasedeno z belim kmetom.

4.1.3 Figure in Polja

Ideja naslednjih dveh hevristik je posodobiti prejšnji hevristiki z dejanskim vzorcem premikanja figur. Definirajmo razdaljo d:

$$d(\text{figura}, \text{polje}) = \text{pot} + \text{ovire}$$

kjer je *pot* število legalnih potez, ki bi jih ta figura potrebovala do napada polja pri prazni šahovnici in *ovire* število figur na trenutni šahovnici na tej poti.

Intuitivno, za vsako figuro na poti potrebujemo eno potezo, da jo umaknemo (če je naša) ali vzamemo (če je nasprotnikova), zato dodaten člen *ovire*. Ta razdalja je še vedno relativno hitra za izračun (čeprav precej počasnejša od Manhattnove), vendar daje bolj realno oceno o tem, koliko časa neka figura potrebuje za pot do nekega polja v matnem kvadratu.

Figure

Hevristika *Figure* vzame vsoto vseh razdalj vseh figur do nasprotnikovega matnega kvadrata. Podobno kot *Manhattan* poskuša kralja preplaviti z množico figur.

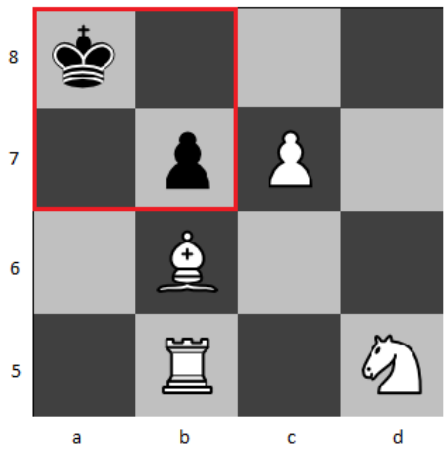
$$h(x) = \sum_{f \in \text{figure}} d(f, \text{matni kvadrat})$$

Polja

Hevristika *Polja* za vsako polje v matnem kvadratu vzame minimalno razdaljo med njim in vsemi našimi figurami. Nato se minimalne razdalje vseh polj seštejejo. Podobno kot *Pokritost* hevristika spodbuja, da je vsakemu polju blizu vsaj ena naša figura.

$$h(x) = \sum_{p \in \text{matni kvadrat}} \min_{f \in \text{figure}} d(f, p)$$

Primer izračuna obeh hevristik je na sliki 4.3.



Slika 4.3: Matni kvadrat je označen z rdečo. Vrednost hevristike *Figure* je 0 (lovec že napada kvadrat) + 0 (kmet že napada kvadrat) +1 (trdnjava napada kvadrat na prazni šahovnici, vendar je vmes lovec) + 2 (skakač potrebuje en premik do napada, ena figura je vmes) = 3. Vrednost hevristike *Polja* je 0 (a7 je že napadeno) + 0 (b8 je že napadeno) + 1 + 1 (trdnjava ima razdaljo 1 do b7 in do a8) = 2.

4.1.4 Izhodišče

Pri testiranju smo uporabili tudi hevristiko *Izhodišče*, ki poziciji vedno predpiše vrednost, ki je enaka številu do sedaj uporabljenih potez. To sprevrže uporabljen algoritem A* v preiskovanje v globino (ang. *Depth-first search*), saj bo vedno prioritiziral razvoj najglobljega vozlišča. Ta hevristika služi kot merilo, ki ga morajo ostale preseči.

4.1.5 Dodatne izboljšave

V tem podpoglavju navedemo dve možni izboljšavi za hevristike. Ti izboljšavi lahko uporabimo na katerikoli izmed naštetih hevristik. Obe sta se izkazali za zelo uporabni in enostavni za izračun, zato sta bili uporabljeni skupaj s hevristiko *Pokritost* v končni verziji programa.

Promoviranje kmetov

Prva izmed njih, *Promocija*, temelji na opažanju, da se veliko kasnejših matov začne s promocijo enega izmed kmetov, največkrat v kraljico, in potem uporabo te figure pri samem matu. Ta izboljšava nagrajuje dodaten material in oddaljenost kmetov do promocijskega polja. Izmed vseh kmetov vzame oddaljenost najbližjega, ki lahko legalno promovira (ni blokiran s figuro, pot ne povzroči šaha nasprotnemu kralju, in v preostanku kroga je na voljo še dovolj potez za pot do promocije).

Vezava na kralja

Omenili smo že, da je nelegalno dati šah v prvi potezi (dovoljen je šele v zadnji potezi v danem krogu), kar nam omogoča nastaviti pozicije, kjer bi se šahu dalo izogniti le s tako nelegalno potezo - posledično tako matiramo. Primer tega je viden na sliki 4.4. Na sliki opazimo, da je črni kralj vezan na svojega lovca in beli to izkoristi tako, da svojega kralja premakne na ustrezno diagonalo. Program prepozna polja, kjer se da tako vezavo izkoristiti, in nagradi kralja, ko ta stoji na teh poljih.

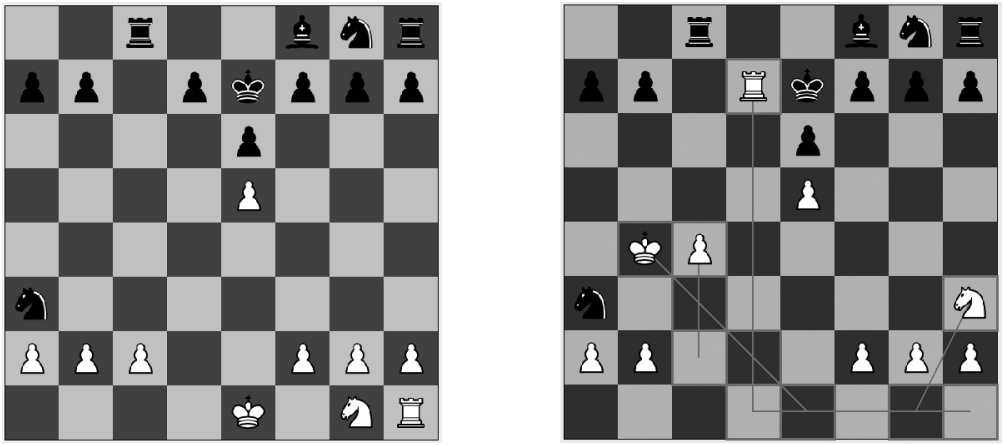
4.2 Iskalni algoritem

A* je algoritem za iskanje najkrajše poti od izhodišča do ciljnega vozlišča (v našem primeru matne pozicije) in je privzeta rešitev pri reševanju tovrstnih problemov. Algoritem v vsakem koraku razvije vozlišče n z najmanjšo oceno $f(n)$.

$$f(n) = g(n) + h(n)$$

$h(n)$ hevristična ocena vozlišča, $g(n)$ pa cena poti do vozlišča.

Domena iskanja matov ima posebno lastnost, da vse rešitve ležijo na isti globini (šah - in posledično tudi mat - se lahko zgodi le v zadnji potezi sekvence). To spoznanje je nekoliko v nasprotju z idejo algoritma A*, ki



Slika 4.4: Na sliki je prikazano, kako lahko beli matira črnega s pomočjo tako imenovanega italijanskega mata. Tako je poimenovan, ker ni mogoč v škotski verziji. Bela trdnjava napada črnega kralja, vendar če bi jo kralj vzel, bi s tem njegov lovec že v prvi potezi napadel belega kralja, kar ni dovoljeno.

poskuša najti najkrajšo rešitev in zato porabi veliko časa pri vozliščih blizu korena, kjub temu da tam rešitev v našem primeru gotovo ni.

Prvi pristop pri reševanju tega problema je bil ignorirati ceno poti $g(n)$. To sprevrže algoritem A* v algoritem, ki v vsakem koraku razvije vozlišče z najboljšo hevristično vrednostjo (ang. *Best-first search*). Ta algoritem navadno prioritizira vozlišča, ki so globoko v preiskovalnem drevesu, in se kljub navidezni naivnosti dobro obnese. Še posebej dobro deluje, kadar je na voljo veliko potez in mati niso "tesni"; pri tem mislimo na mate, ki se jih da doseči v recimo 8 potezah, kjub temu, da jih imamo na voljo 10.

Drugi pristop je bil ceno poti $g(n)$ določiti glede na to, koliko potez imamo na voljo. Denimo, da neka hevristika v povprečju izboljša svojo vrednost od začetne do končne pozicije za x, ter da imamo pri tem matnem problemu na voljo y potez. Potem lahko pričakujemo, da bomo v vsaki svoji potezi hevristično evalvacijo pozicije izboljšali za približno x/y. Primer: če moramo pokriti 9 polj v 9-ih potezah, si lahko privoščimo pokrivanje enega na potezo, če pa imamo 3 poteze, moramo v povprečju pokriti 3 polja na potezo. Iz tega

opažanja lahko definiramo ceno poti kot

$$g(n) = \text{Število porabljenih potez} * \frac{x}{y}$$

Vrednost x lahko pridobimo empirično za vsako hevristiko posebej. Ta pristop se je v splošnem boljše obnesel od zgornjega, čeprav je z njim veliko hevristik imelo težave pri matih, ki so imeli na voljo veliko potez.

Pri iskanju večkrat zaidemo v pozicijo, v kateri smo že bili, le da smo do nje prišli po drugi poti. Ker so poti do mata dostikrat (ampak ne vedno) izmenljive, je takih transpozicij ogromno. V ta namen smo si shranili vrednost vsake razvite pozicije v ustrezno strukturo (ang. *transposition tables*) in nato ignorirali vse pozicije do katerih smo prišli, vendar njihovo vrednost že imamo. To je pohitrilo iskanje za vsaj en red velikosti. O uporabljeni zgoščevalni funkciji bomo govorili v dodatku B.

Shema algoritma je podana v pseudokodi 1.

Algorithm 1 A* (začetnoVozlišče, čas)

```
1: preiskana ← ∅ //Množica že preiskanih vozlišč
2: Izračunaj f oceno za začetno vozlišče
3: prioritetnaVrsta ← vrsta z začetnim vozliščem
4: while prioritietnaVrsta ni prazna in čas ni potekel do
5:     vozlišče ← prvo vozlišče iz prioritetne vrste
6:     premakni vozlišče iz prioritne vrste v preiskana
7:     if vozliče je ciljno then
8:         return pot do vozlišča
9:     else
10:        for Za vse otroke vozlišča do
11:            if otrok ni v preiskana then
12:                Generiraj f oceno otroka ter ga dodaj v prioritetno vrsto
13:            end if
14:        end for
15:    end if
16: end while
17: return NULL
```

4.3 Ocena pristopov

V tem podpoglavju bomo predstavili rezultate testiranj naštetih hevristik in algoritmov. Na testni množici pozicij, ki vsebujejo mate, bomo te mate poskušali poiskati. Zanimalo nas bo, kako hitro bodo to nalogo opravile različne kombinacije hevristik in algoritmov. Merili smo tako povprečni čas kot število primerov, rešenih v manj kot n sekundah (za različne vrednosti n). V drugi fazi eksperimentov nas je zanimalo, kako sposobnost iskanja matov vpliva na sposobnost igranja celotne igre.

4.3.1 Testna množica

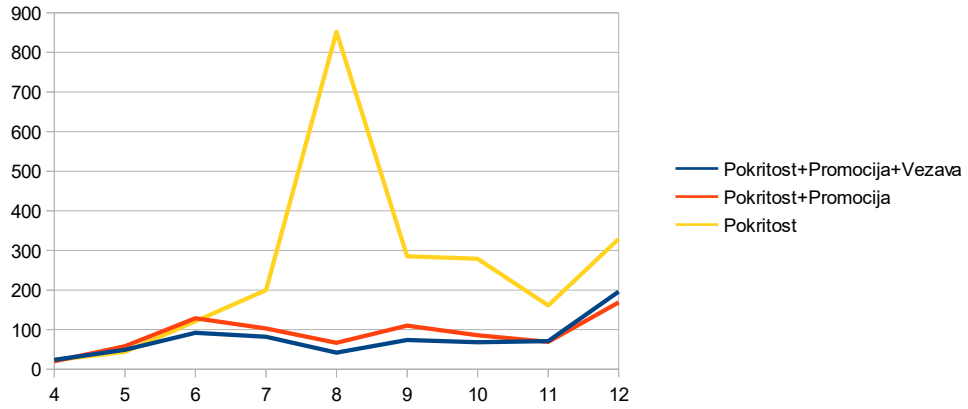
Matne pozicije smo vzeli iz pravih iger, odigranih med igranjem našega programa proti samemu sebi. Pri izvajanju teh eksperimentov je iskalnik matov imel na voljo ogromno časa (nekaj minut), tako da je obstoječ mat zagotovo našel. Za vsak krog dolžine od 4 do 12 potez smo naključno izbrali 100 pozicij, v katerih je prisoten mat. Na tak način smo naredili testno množico, v kateri je bilo prisotnih 900 matov. Mati v tretjem krogu so zelo redki in tudi zelo enostavni, zato so bili izpuščeni. Po 12. krogu je glede na material na šahovnici mat navadno trivialen ali pa nemogoč. Velikokrat je v tej fazi igre figur tako malo, da so potrebni posebni vzorci matiranja tekom več krogov (kakor pri običajnem šahu); te si bomo ločeno ogledali v poglavju 6.2. Distribucija matov v različnih krogih nam omogoča vpogled v to, kako dolžina rešitve vpliva na uspešnost algoritma.

4.3.2 Povprečni časi

Testirali smo povprečni čas, v katerem program najde mate v posameznem krogu. Časovna omejitev za iskanje posameznega mata je bila 60 sekund. Če program mata ni našel, se mu je za ta primer štel maksimalni čas.

Najprej podajmo opažanje, da izboljšavi *Promocija* in *Vezava* vedno koristno vplivata na katerokoli od uporabljenih hevristik. Obe sta enostavni za izračun in vodita iskanje do posebnih vrst matov, ki jih osnovne hevristike

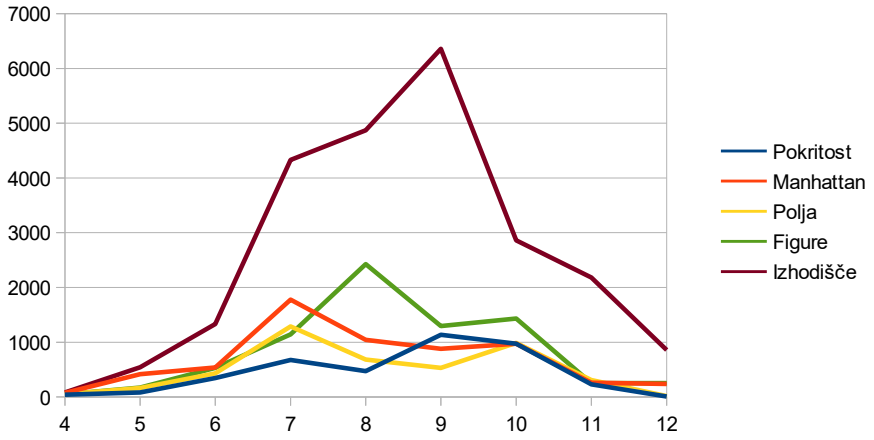
pogosto ne najdejo same. Primer takšnega napredka je viden na sliki 4.5. Meritev je bila narejena s hevristiko *Pokritost* ter A* algoritmom. Vidimo, da dodatek *Promocije* vidno izboljša delovanje, še posebej v pozicijah s 7 ali več potezami na voljo (tu so seveda promocije možne in smiselne). Največ takih matov se glede na to meritev zgodi okoli osmega kroga, kar ustreza našim izkušnjam z igro. Izboljšanje z dodatkom *Vezave* ni tako izrazito, saj tovrstni mati niso pogosti, vendar še vedno obstaja. Isti vzorec lahko opazimo tudi pri drugih kombinacijah hevristik in algoritmov. V nadaljevanju zato ti dve hevristiki avtomatično vključimo v vse preostale meritve, brez eksplicitnega poudarka o tem.



Slika 4.5: Graf prikazuje uspešnost hevristike *Pokritost* s potencialnim dodatkom *Promocije* in *Vezave*. X os prikazuje krog, v katerem se je mat zgodil, Y os pa povprečen čas (v milisekundah) za najden mat v tem krogu.

Najprej si poglejmo rezultate algoritma *Best-first*, z vsemi petimi hevristikami (slika 4.6). Vse hevristike delujejo bolje od *Izhodišča*, kar priča o njihovi koristnosti. *Pokritost* je najboljša izmed njih. Opazimo nekoliko nenavadno obliko domene; s povečevanjem kroga se težavnost problema najprej pričakovano poveča nato pa spet pade. Padec se zgodi zaradi manjše količine materiala na šahovnici, kar tudi pomeni, da imajo preostale figure več pro-

stora za gibanje in posledično lažje manevrirajo h kralju. Navadno so poteze kasneje v krogih tudi bolj med sabo izmenljive in obstaja več poti do mata.

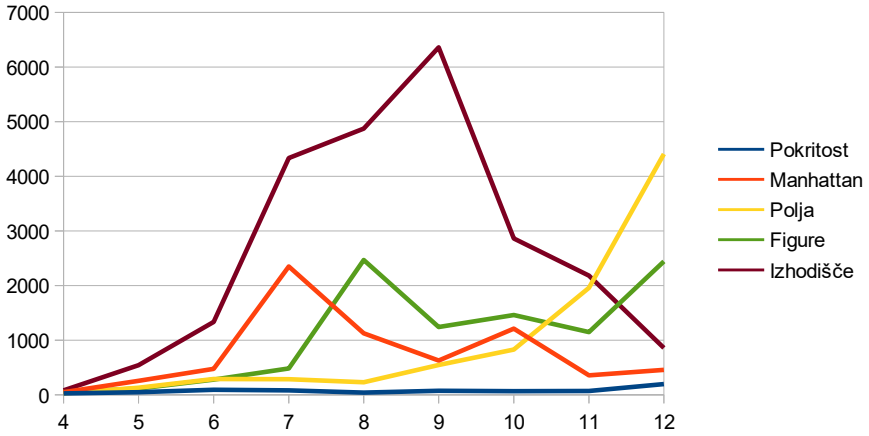


Slika 4.6: Graf prikazuje uspešnost petih hevristik z algoritmom Best-First search. X os prikazuje krog, v katerem se je mat zgodil, Y os pa povprečen čas (v milisekundah) za najden mat v tem krogu.

Poglejmo si še rezultate A* algoritma, z vsemi petimi hevristikami (slika 4.7). Hevristika *Pokritost* s tem algoritmom deluje izjemno dobro in ta kombinacija je bila na koncu tudi izbrana za naš program. Ostale hevristike delujejo boljše do vključno osmega kroga, nato njihova učinkovitost začne padati; v nekaterih primerih je celo slabša od *Izhodišča*. Glavni razlog za to je preiskovanje plitvih vozlišč namesto vozlišč z globino rešitve. Ceno poti pri teh hevristikah bi lahko ustrezno spremenili tako, da bi bila še manjša pri višjih krogih, vendar tudi v tem primeru te hevristike ne bi presegle uspeha *Pokritosti*.

4.3.3 Število matov v časovni omejitvi

Poleg povprečnih časov smo preverili tudi dejanske čase. Isto povprečje je mogoče doseči s konstantnim časom ali pa z izmenjevanjem zelo visokih in



Slika 4.7: Graf prikazuje uspešnost petih hevristik z algoritmom A*. X os prikazuje krog, v katerem se je mat zgodil, Y os pa povprečen čas (v milisekundah) za najden mat v tem krogu.

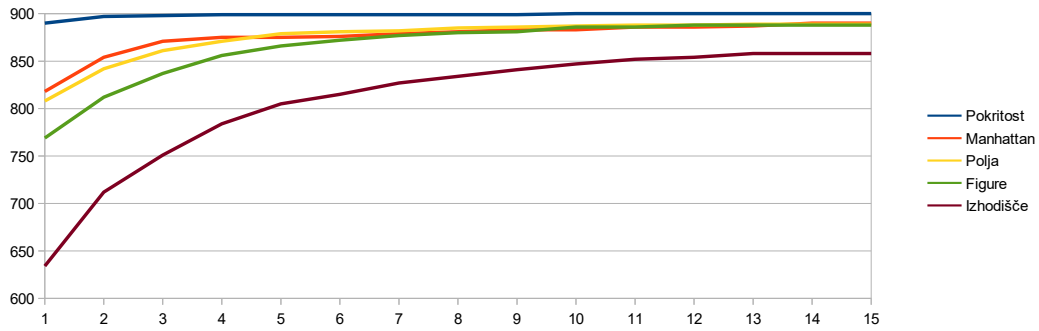
zelo nizkih - prva možnost je bolj zaželena. Rezultat je viden na sliki 4.8. Hevristika *Pokritost* se odlično obnese, v prvi sekundi najde več kot 98% matov, v desetih najde vse, ter je boljša od alternativ v vsaki točki tega grafa.

4.4 Vpliv na igro

V tem poglavju smo testirali, kako kvaliteta iskalnika matov vpliva na kvaliteto igre. V ta namen smo naredili 5 različic programa, vsaka izmed njih je uporabljala eno izmed petih matnih hevristik z algoritmom A*. Ostali deli programa so pri vseh različicah delovali enako.

Te različice programov so igrale vrsto iger druga proti drugi pri različnih časovnih omejitvah na krog; število iger v vsaki omejitvi je podano v tabeli 4.1.

Za preprečitev ponavljanja istega zaporedja potez v vsaki igri, smo v igranje druge faze (glej poglavje 5) dodali manjši naključni faktor. V tej fazi



Slika 4.8: Graf prikazuje uspešnost petih hevristik z algoritmom A*. Y os prikazuje število matov najdenih v manj kot X sekundah, kjer je X vrednost na X osi.

Tabela 4.1: Število iger za vsako časovno omejitev

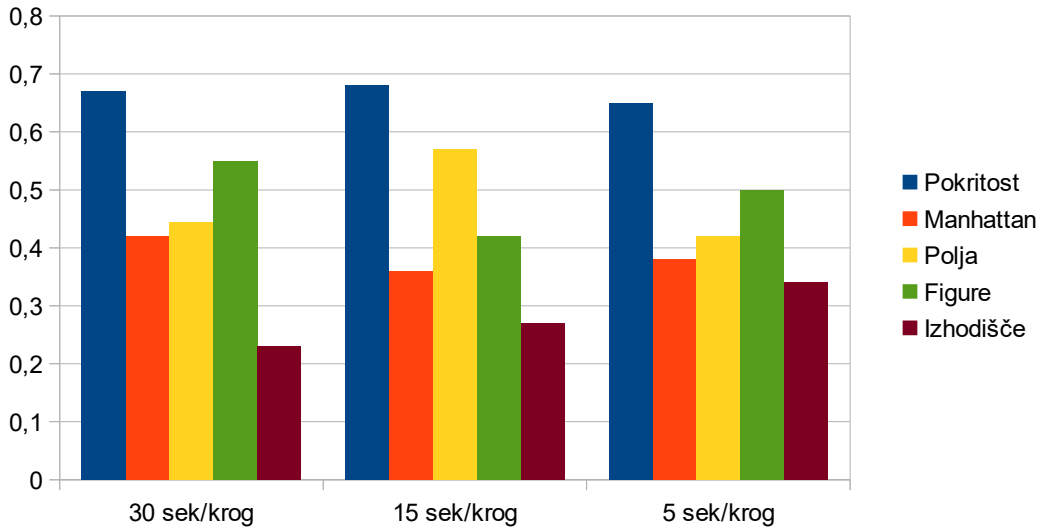
5 sek	15 sek	30 sek
600	400	200

preiskovanja ena izmed hevristik beleži že odigrane pozicije in jih evalvira glede na to, ali so vodile v zmago ali poraz; to usmerja igro k dodatni raznolikosti. Prav tako je bila izbira otvoritve naključna (glej poglavje 6.1). Enak pristop smo uporabili tudi pri kasnejših tovrstnih eksperimentih.

Rezultati so vidni na sliki 4.9. Ne glede na časovno omejitev na krog je relativen vrstni red matnih hevristik ostal enak. Hevristika *Pokritost* se je nepresenetljivo najboljše uvrstila, hevristika *Izhodišče* pa najslabše. Pri časovni omejitvi 30 sekund je *Pokritost* zmagala skoraj 3x več iger od *Izhodišča*. Ostale hevrstike so se uvrstile nekje med tema dvema; nekoliko presenetljiva je bila le hevristika *Polja*, za katero smo glede na rezultate v prejšnjem poglavju pričakovali boljši izid v nekaterih časovnih omejitvah.

Razlike so bile največje pri višjih časovnih omejitvah, kar je morda nekoliko neintuitivno (razlika v hitrosti bi se morala bolj poznati, ko je časa malo). Razlog je najverjetneje v tretji fazi preiskovanja (preprečevanja matov), kjer

je treba včasih ovreči veliko matov, preden najdemo varno potezo. Daljše preiskovanje omogoča dobrim hevristikam uspešno opravljanje te naloge, pri kratkih omejitvah pa v težkih pozicijah vse spodletijo.



Slika 4.9: Graf prikazuje uspešnost petih hevristik z algoritmom A*. Za vsako hevristiko, pri različnih časovnih omejitvah, je zabeležen odstotek dobljenih iger.

Poglavje 5

Iskanje dobrih sekvenc potez

V poglavju 3 smo delovanje programa razdelili v tri faze: iskanje mata, iskanje dobrih potez in preprečevanja mata. V tem poglavju se bomo posvetili drugi fazi tega postopka. Cilj bo najti nekaj dobrih zaporedij potez, od katerih upamo, da je vsaj ena odporna na nasprotnikov mat. Najprej si bomo pogledali hevristike, ki bodo vodile algoritem, nato pa še različne iskalne algoritme. V poglavju 5.3 bomo te algoritme primerjali med seboj, nato bomo v poglavju 5.4 testirali vpliv posamezne hevristike.

5.1 Hevristike

V nadaljevanju bomo našteli hevristike, ki smo jih spoznali za koristne pri igranju te igre. Za razliko od matnih hevristik, tukaj ne bomo izbrali ene hevristike, ampak bomo končno hevristiko definirali kot linearno kombinacijo vseh naštetih.

$$h(x) = \sum h_i * w_i$$

Uteži w_i smo nastavili ročno in jih navedli v tabeli 5.1. Katere hevristike in katere uteži so optimalne, je še relativno neraziskan problem. Veliko lastnosti pozicije, ki bi bile zelo koristne pri klasičnem šahu, tukaj nimajo nobene vrednosti, ter obratno. Primer tega je branjenje figur; ključnega pomena pri klasičnem šahu in precej nekoristno pri 1-2-3 šahu, saj lahko v tej različici

nasprotnikovo figuro vzamemo in nato napadalca umaknemo. Avtomatično učenje uteži na hevristikah bomo prepustili nadaljnjemu delu.

Material

Po matiranju je material najpomembnejši vidik igre. Prvemu igralcu, ki mu zmanjka koristnega materiala (blokirani kmetje so primer nekoristnega materiala), navadno preti poraz. Vrednosti figur za to igro še niso znane, vendar je njihov vrstni red podoben kakor pri klasičnem šahu. Kraljica je seveda največ vredna in njena prisotnost predstavlja veliko grožnjo za nasprotnika, zato se večina iger začne z eliminacijo obeh kraljic. Sledi top in nato še obe lahki figuri. Lovec je zaradi svoje mobilnosti boljši v začetni fazi igre, vendar mu vrednost proti končnici močno pade, saj ne more doseči polovice polj. Za skakača velja ravno obratno, dlje kot traja igra, večjo prednost ima nad lovcem. Vrednost kmeta je povsem odvisna od njegove možnosti promoviranja. Kmet blizu promocije je vreden skoraj toliko kakor kraljica, blokiran kmet pa nima skoraj nobene vrednosti.

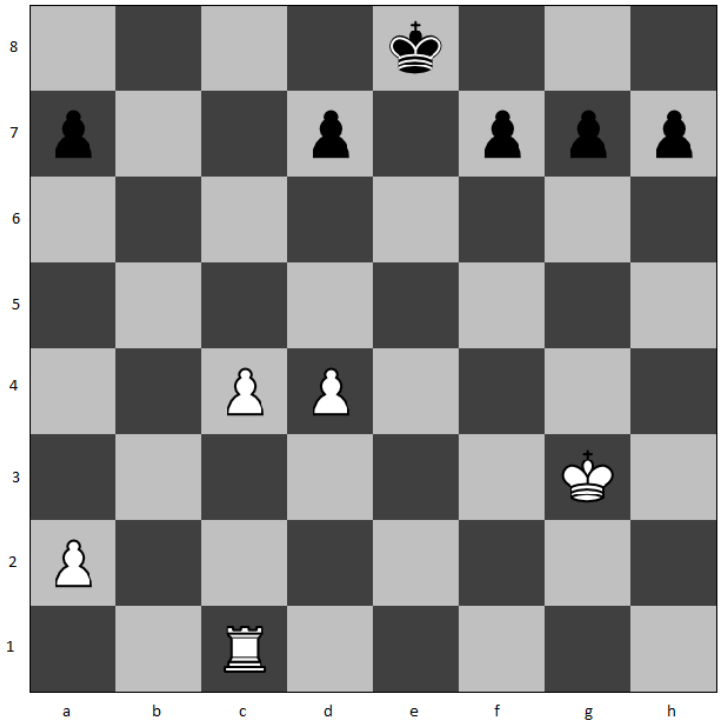
Za hevristiko smo uporabili klasične vrednosti figur (D=9, T=5, L=3....) ter dodatno vrednost kmeta upoštevali v naslednji hevristiki.

Promocije

Promocije so v tej igri zelo pogoste, saj lahko v petih potezah pripeljemo kmeta iz začetne pozicije na polje promocije. Vsak promovirani kmet lahko prinese veliko vrednosti svojemu igralcu, zato je pomembno, da zna program te promocije ustaviti. Tukaj pride do izraza koncept blokiranja kmetov; na pot proti promociji postavimo figuro. Alternativno lahko postavimo kralja tako, da bi na poti kmet dal šah in tako zaključil svoj krog. Primer blokiranih kmetov je prikazan na sliki 5.1. Navadno se vseh promocij okoli sedmega in osmega kroga ne da ustaviti, moramo pa poskrbeti, da je kasnejših čim manj.

Za ugotavljanje možnosti svojh promocij in preprečevanja nasprotnikovih smo naredili dve metriki. Prva za obe strani ugotavlja, koliko kmetov lahko promovira, druga za obe strani ugotavlja za koliko potez je od promo-

cije oddaljen najbližji kmet. Obe vrednosti uteženo prištejemo k evalvaciji pozicije. V primeru, da nasprotnik nima nobenega kmeta s sposobnostjo promocije, evalvaciji prištejemo še večjo nagrado. Prva metrika spodbuja blokiranje/eliminiranje kmetov, druga pa napredovanje svojih kmetov oziroma onemogočanje nasprotnikovih naprednih kmetov.



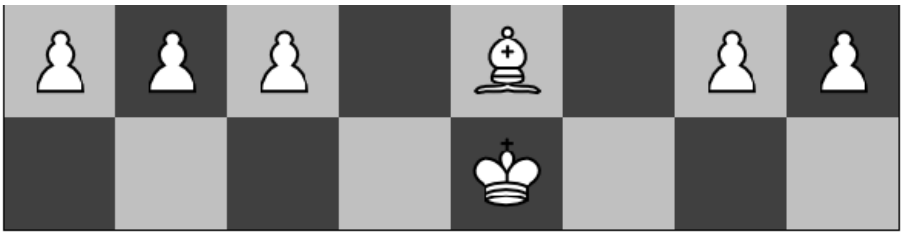
Slika 5.1: Noben izmed črnih kmetov ne more promovirati. Črnega kmeta v liniji a, blokira beli kmet v isti liniji. Kmeta v liniji d blokirata najprej kmet, nato še trdnjava. Kralj lahko blokira vse tri kmete na linijah f, g in h, saj ne morejo legalno priti mimo njega. Beli kmet na polju c4 lahko promovira.

Kraljeva varnost

V nasprotju s klasičnim šahom je kralj varnejši na odprtem, kakor stisnjen med svoje figure in obdan s kmeti. Kmetje navadno ne predstavljajo nobene

zaščite pred kombiniranim napadom (glej sliko 2.1) in samo omejujejo kraljev prostor umika. Izkaže se, da je najvarnejše biti na drugi vrsti (sedmi za črnega), saj tam rob šahovnice ne omejuje kralja. Hevristika nagradi ta položaj kralja, poleg tega nagrado poveča za vsako prazno polje okoli njega.

V nekaterih pozicijah je kralj koristen na prvi (zadnji za črnega) vrsti, saj na tak način lahko prisili, da nasprotnikova promocija povzroči šah, in se mora zgoditi na koncu kroga. To prepreči promovirani figuri, da bi vzela veliko naših figur, oziroma se postavila v boljšo pozicijo za dajanje mata. Pozicije, v katerih je to koristno, je težko prepoznati, zato kralja v zadnji vrsti nagradimo samo, če ima pred seboj lovca (glej sliko 5.2). Ta kombinacija kralja in lovca se izkaže za dobro obrambno idejo.



Slika 5.2: Črni lahko promovira samo s šahom (rezen, če se odloči za podpromocijo). Tak šah je enostavno blokirati z lovcem. Sledi, da kralj ne more biti matiran s strani promovirane figure, kar nasprotniku lahko izniči veliko matnih idej.

Šah

Kadar končamo krog s šahom, mora nasprotnik prvo potezo porabiti za umik iz njega. S tem mu lahko preprečimo njegov načrt za ta krog, če je za izvršitev tega načrta potreboval vse poteze kroga. Taki šahi so zato koristni in so nagrajeni v vrednosti enega kmeta.

Če krog končamo z matom, dobi seveda pozicija največjo vrednost. To se redko zgodi, saj mate navadno najde že iskalnik matov.

Poteze, ki ne povzročijo mata in ne vzamejo nasprotnikovega materiala, ampak samo približajo figure nasprotnikovemu kralju, se navadno ne splačajo. S tem početjem se samo nepotrebno izpostavijo in so navadno vzete še preden lahko v svojem naslednjem krogu uresničijo svojo matno grožnjo. Hevristika izključno napadalnih potez (z izjemo zgoraj omenjenega šaha) ne promovira.

Razvoj figur

Vrednost razvoja figur je v tej različici šaha vprašljiva. Figure na začetni poziciji so sicer nemobilne, vendar tudi težko dosegljive in imajo zato večjo možnost preživetja. Razvoja/mobilnosti figur zato eksplicitno ne nagrajujemo. Dodamo jim samo nekoliko večjo vrednost, če se nahajajo na zadnjih dveh vrstah šahovnice. Tam so odlično postavljene za matiranje ali pobiranje figur.

Shranjene pozicije

Za vsako doseženo pozicijo v nekem sklopu iger si program zabeleži, koliko krat je prišel do nje, in koliko zmag je dosegel iz nje z vsako barvo. To znanje lahko uporabi pri evalvaciji te pozicije kot dodaten člen v hevristiki. Utež tega člena je odvisna od števila iger, v katerih je do te poziciji prišel (večkrat ko jo je dosegel, bolj zanesljiva je njena ocena), ter naključnega člena. Naključni člen promovira k raziskovanju in prepreči, da bi vse igre programa s samim sabo tekle po istih potezah.

Izogibanje lovcu

V primeru, da ima nasprotnik samo enega lovca, hevristika nagradi figure, ki stojijo na poljih nasprotne barve. Teh figur lovec ne more doseči, njegova vrednost se tako zmanjša.

Tabela uteži

Zaradi celovitosti na tem mestu pokažemo tabelo z izračunom vrednosti za vsako našteto hevristiko. Uteževanje hevristik je bilo opravljeno ročno in narejeno glede na naše razumevanje te igre. Vrednosti hevristike so bile normalizirane glede na vrednost kmeta; ta ima vrednost 1. Vse hevristike so bile izračunane simetrično za oba igralca. Končna ocena pozicije je hevristična vrednost za aktivnega igralca, kateri odštejemo hevristično vrednost za nasprotnika.

Tabela 5.1: Tabela hevristik in njihovih uteži

Hevristika	Vrednost / Izračun
Kraljica	9
Trdnjava	5
Skakač	3
Lovec	3
Kmet	1
n kmetov z možnostjo promocije	$n * 0.4$
n kmetov z možnostjo promocije	$if(n == 0) - 6$
n polj do najbližje promocije	$-n * 0.4$
Kralj stoji na drugi vrsti	0.8
n praznih polj okoli kralja	$n * 0.25$
Obrambna formacija kralja in lovca	1.5
Obstoj figur na zadnjih dveh vrstah	2
n figur, dosegljivih nasprotnikovemu lovcu	$n * 0.2$
Šah	1
Mat	∞

5.2 Iskalni algoritem

Pri iskanju matov se je iskanje zaključilo na koncu kroga; nasprotnikovi odgovori nas niso zanimali. Za igranje celotne igre je seveda smiselno in najverjetneje potrebno bolj poglobljeno iskanje. Treba je na primer pričakovati

nasprotnikov odgovor in minimizirati njegovo moč. Klasični iskalni algoritem za to nalogo se imenuje Min-Max, vendar ga na naši domeni zaradi njene prevelike razvejanosti ne moremo enostavno uporabiti. Opis algoritma in njegove težave bomo predstavili v poglavju 5.2.1, nato bomo v poglavjih 5.2.2 in 5.2.3 predstavili predlagane rešitve. V poglavju 5.3 bomo različne tako dobljene algoritme primerjali med seboj.

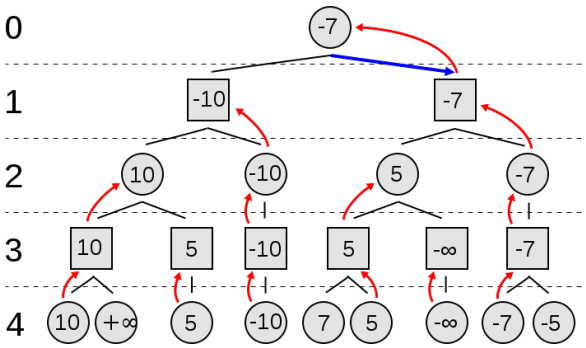
5.2.1 Min-Max

Min-Max je algoritem za igranje iger z ničelnim izidom, za dva igralca. Razvije vse poteze aktivnega igralca, za vsako potezo razvije vse nasprotnikove poteze, za vsako njegovo vse poteze aktivnega igralca in tako naprej do neke globine. Hevristično oceni liste, nato te ocene propagira navzgor. Pri propagiranju na lihih globinah vzame maksimum izmed vseh otrok vozlišča, pri sodih pa minimum. To odseva, da vsaka stran izbira zase najboljšo potezo na vsaki globini, kjer je na vrsti. Shema propagiranja na sliki 5.3.

Opomba: V prejšnjem odstavku smo besedo *poteza* uporabili kot: ena možnost za igranje svojega kroga. V primeru 1-2-3 šaha, kjer naredimo sekvenco premikov, lahko za namene tega algoritma celotno sekvenco premikov obravnavamo kot eno potezo.

Struktura algoritma je podana v pseudokodi 2. Ta specifična implementacija uporablja dejstvo, da je $max(a, b) = -min(-a, -b)$, kar poenoti propagiranje v sodih in lihih globinah. Algoritem s to manjšo modifikacijo imenujemo *Negamax*.

Težava aplikacije tega algoritma na našo domeno je v tem, da v kasnejših krogih ne moremo generirati niti vseh svojih sekvenc potez v doglednem času. Generacija vseh možnih odgovorov na vsako našo sekvenco potez še hitreje postane nemogoča. Za primer vzemimo sedmi krog, kjer imamo na voljo približno 20^7 možnih sekvenc potez (v dani poziciji lahko igramo približno 20 različnih potez, igramo jih sedemkrat zapored). Odgovorov na vsako sekvenco potez je 20^8, skupaj bi morali generirati 20^{15} sekvenc potez že pri globini preiskovanja 1. Tudi z upoštevanjem mnogih transpozicij je faktor



Slika 5.3: Slika prikazuje vračanje ocen listov navzgor po drevesu. Igralec na potezi bo izbral desno potezo, saj v najslabšem primeru (nasprotnik vedno optimalno odigra) vodi do boljšega lista kakor leva. Slika je vzeta iz spleta[13].

Algorithm 2 Negamax (vozlišče, globina, barva)

```
1: if globina= 0 then
2:     return barva∗hevristična ocena vozlišča
3: end if
4: najboljsaVrednost ← −∞
5: for otroci vozlišča do
6:     vrednost ← -negamax(otrok,globina-1,-barva)
7:     najboljsaVrednost ← max(najboljsaVrednost, vrednost)
8: end for
9: return najboljsaVrednost
```

razvejanosti prevelik za uporabo Min-Max algoritma v kakšerni koli realni časovni omejitvi.

5.2.2 Iskanje z algoritmom A*

Prvi predlagani algoritem deluje na isti način kakor iskalnik matov, z uporabo algoritma A*, z enako definiranim členom $g(x)$. Namesto matnih hevristik uporabimo hevristike definirane v poglavju 5.1. Preiskujemo torej samo svoj krog, brez upoštevanja nasprotnika. Enako kot pri iskanju matov, je že preiskovanje samo svojega kroga lahko velik izziv, in tudi, če ves čas posvetimo

samo tej nalogi, algoritem najverjetneje ne bo uspel razviti vseh potez, ki jih imamo na voljo.

Nasprotnik je upoštevan implicitno, saj kar nekaj hevristik upošteva nasprotnikove zmožnosti. Za primer vzemimo pozicijo, kjer lahko nasprotnik promovira: označimo jo lahko za slabo brez dejanskega izračuna njegovih potez. Preprečevalnik mata v naslednji fazi iskanja dodatno upošteva nasprotnikove zmožnosti dajanja mata.

A* moramo nekoliko prilagoditi spremenjenim potrebam. Namesto, da iskanje ustavimo pri razvoju ciljnega vozlišča, preiskujemo do poteka časa in si shranimo vsa končna vozlišča (pozicije na koncu kroga), razvrščena glede na oceno. Seznam tako pridobljenih vozlišč (in sekvenc potez, ki vodijo do njih) posredujemo preprečevalniku mata.

Opomba: V primeru, da nameravamo igro igrati na daljših časovnih omejitvah (nekaj minut na krog ali več), je A* smiselno nadomestiti s kakšnim podobnim algoritmom, recimo IDA*. Ta deluje na podoben način kakor A*, vendar nima težav s pomanjkanjem pomnilnika pri daljšem preiskovanju.

5.2.3 Kombiniranje algoritmov A* in Min-Max

Predvidevamo seveda, da v prejšnjem poglavju opisano kratkogledno iskanje ni dovolj za uspešno igranje te igre. Iskanje smo poskušali razširiti s kombiniranjem algoritma A* in algoritma Min-Max. Najprej si oglejmo naslednjo shemo:

1. Najdemo neko vnaprej določeno število najboljših zaporedij potez v svojem krogu. Pri tem preiskovanju uporabimo A* na način opisan v poglavju 5.2.2.

2. Za vsako izmed teh zaporedij poskušamo najti nasprotnikov najboljši odgovor. Znotraj nasprotnikovega kroga ponovno preiskujemo z A*.

3. Najboljše zaporedje potez je tisto, pri katerem je nasprotnikov najboljši odgovor kar se da slab (glede na oceno hevristike).

Ilustracija je na sliki 5.4.

Opazimo lahko podobnost med tem postopkom in Min-Max algoritmom, ki razvija do globine 1. Razlika je v tem, da ne razvijemo vseh svojih možnih zaporedij potez, ampak le tista, do katerih prispemo z A* algoritmom. Izmed razvitih zaporedij samo za najboljše računamo nasprotnikove odgovore (ponovno ne vseh).

Enako idejo lahko uporabimo za preiskovanje v poljubno globino. Končni algoritem dobimo, če shemo Min-Maxa spremenimo tako, da sledečo vrstico:

```
5: for otroci vozlišča do
6:     ...
8: end for
```

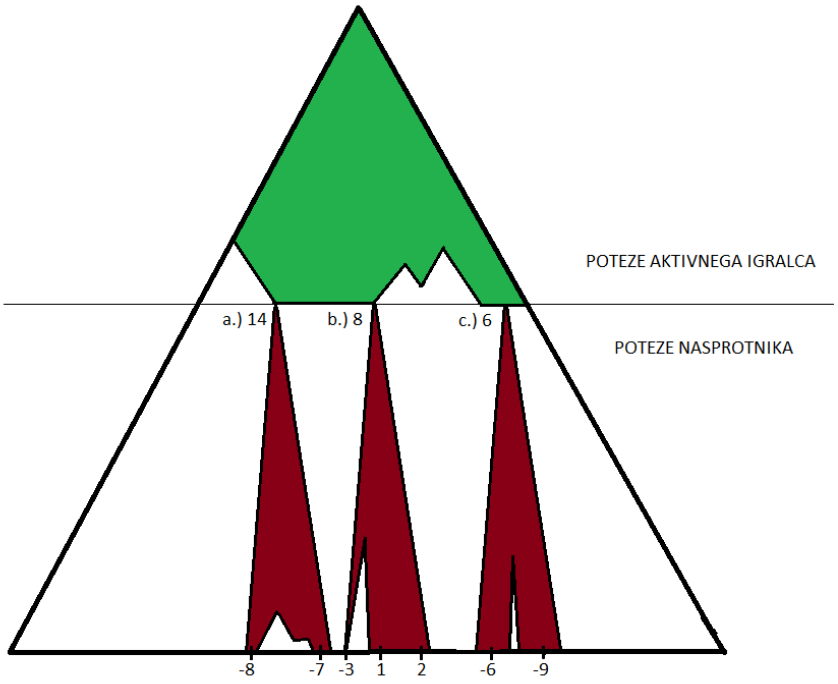
Zamenjamo s to:

```
5: for samo nekateri najboljši otroci vozlišča najdeni z A* do
6:     ...
8: end for
```

Razvejanost smo umetno omejili tako, da požrešno preiskujemo samo najboljše kandidate na vsaki globini. V testiranju v poglavju 5.3 smo ta algoritem uporabili na globini 0,1 in 2. (Globina 0 je enaka preiskovanju samo svojega kroga, opisanega v poglavju 5.2.2).

Z razvijanjem le nekaterih vozlišč izgbubimo nekatere dobre lastnosti Min-Max algoritma, ki zagotavljajo da, nasprotnikov odgovor ne more biti boljši od napovedanega. Z vsako dodatno globino zgubljamo dodatne linije igre in s tem potencialno delamo vedno večjo napako. Prav tako se z večanjem globine zvišuje število različnih pozicij, ki jih mora A* preiskati, in ima zato za vsakega manj časa in posledično daje manj zanesljive ocene. Na podlagi tega sklepamo, da se bo algoritem najboljše obnesel pri globini 1.

Odločiti se je potrebno tudi koliko izmed svojih najboljših zaporedij potez bomo razvijali naprej, ter kolko časa bomo za to porabili. Smiselno je več časa porabiti za analizo krogov bližje korenu, saj na tej analizi sloni vse preiskovanje njihovih otrok. Pri globini 1 smo se odločili naprej razviti 50



Slika 5.4: Zunanji trikotnik predstavlja iskalno drevo dveh krogov. Zeleno pobarvan del drevesa je preiskan del kroga igralca na potezi; rdeče so poteze v nasprotnikovem krogu. Prazen del trikotnika so poteze, ki niso bile razvite. Razmerje med razvitimi in nerazvitimi potezami je samo skicirano in odvisno od časa na razpolago. Nerazvitih je lahko več od razvitih za nekaj redov velikosti. V krogu aktivnega igralca so imele tri končne pozicije, a, b in c, največjo hevristično vrednost in so bile zato nadaljnje razvite. Najboljši nasprotnikov odgovor na pozicijo a ima vrednost -8. Najboljši odgovor na pozicijo b je -3 in na pozicijo c je -9. Pozicija b je torej najbolj obetavna in aktivni igralec bo za svoj krog izbral poteze, ki vodijo do nje.

zaporedij potez in na globni 0 porabiti enako časa kakor za celotno globino 1. Za izbiro najboljših potez za vozlišče na globini 1 imamo tako 50-krat manj časa, kakor za korensko vozlišče.

5.3 Testiranje algoritmov

V tem poglavju testiramo različice algoritma predstavljenega v poglavju 5.2.3. Različice so se razlikovale v globini preiskovanja. Testirali smo globine 0,1 in 2. Preiskovanje z globino 0 je enako preiskovanju opisanemu v poglavju 5.2.2. Testiranje smo izvedli na enak način kot v poglavju 4.4 - z medsebojnim igranjem različic.

Test smo ponovili s tremi različnimi časovnimi omejitvami in pri tem preiskusili, ali se globlje iskanje bolje obnese pri daljših časovnih omejitvah. Število iger v vsakem preiskusu je podano v tabeli 5.2.

Tabela 5.2: Število iger za vsako časovno omejitev

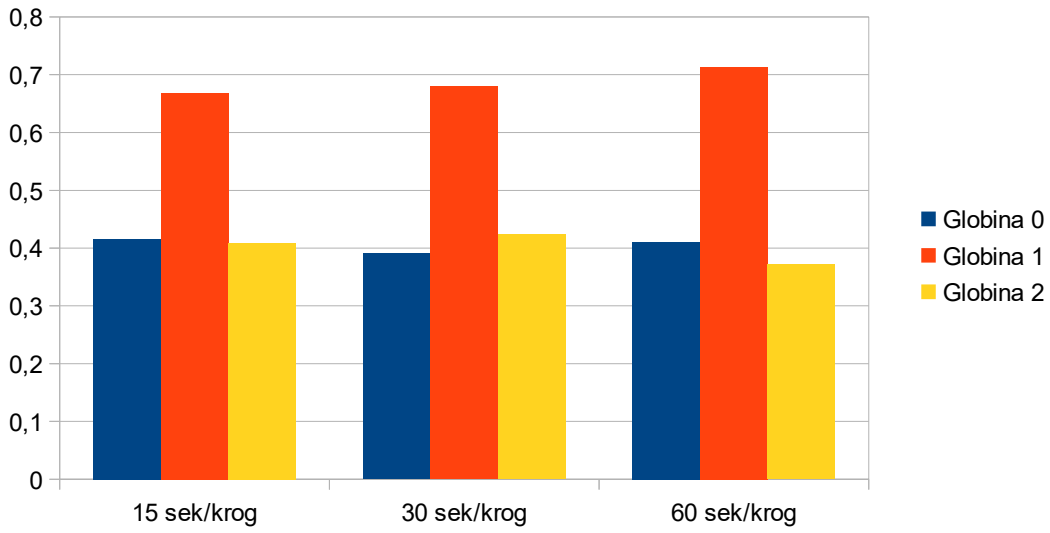
15 sek	30 sek	60 sek
300	250	200

Rezultati so podani na sliki 5.5. Vidimo, da se je preiskovanje v globino 1 (torej preiskovanje svojega kroga in nasprotnikovih možnih odgovorov) najbolje obneslo. Razmerje zmag je bilo presenetljivo enako v vseh časovnih omejitvah. Iskanje v globino 2 je imelo primerljive rezultate iskanju v globino 0. Napaka, ki jo algoritem naredi zaradi svoje požrešnosti, je na vsaki globini večja in najverjeteje je na globini 2 že prevelika in poslabša kakovost igre.

Algoritem s preiskovanjem v globino 1 je bil uporabljen v končni različici našega programa in v ostalih testiranjih.

5.4 Testiranje hevristik

V tem podpoglavju navajamo rezultate testiranja vpliva dveh hevristik, *Promocije* in *Varnost kralja*, na kakovost igre. Koristnost hevristike *Material* je samoumevna (čeprav specifična vrednost figur še ni znana, in je prepuščena nadaljnjemu delu). Naredili smo 4 različice programa, kjer prva uporablja vse v prejšnjem poglavju naštete hevristike, druga nima *Promocije*, tretja



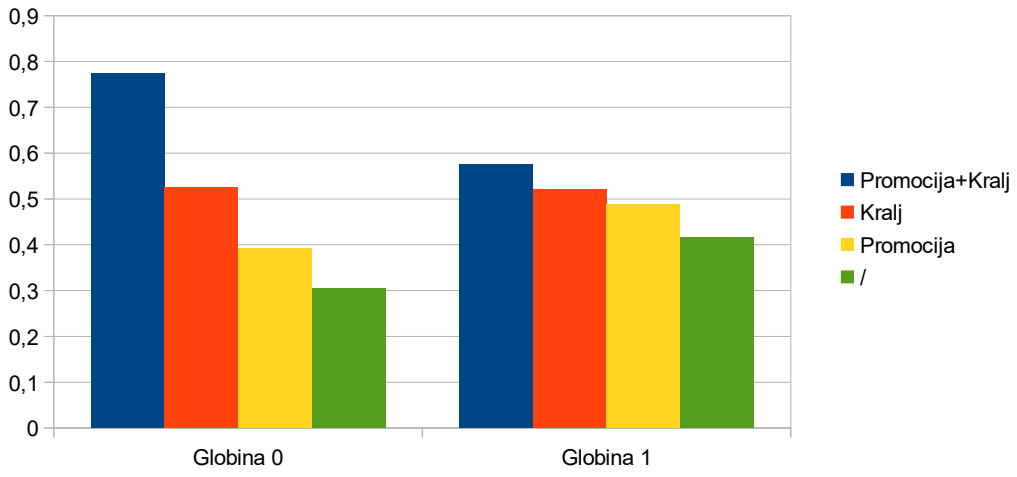
Slika 5.5: Graf prikazuje uspešnost različnih različic algoritma iz poglavja 5.2.3. Algoritmi se razlikujejo glede na globino preiskovanja. Za vsako različico algoritma, pri različnih časovnih omejitvah, je zabeležen odstotek zmag. Upoštevane so samo igre z odločenim izidom (zmaga ali poraz).

nima *Varnosti kralja* in četrta nobene izmed njih. Preiskus je bil ponovljen z algoritmom preiskovanja v globino 0 ter algoritmom z globino 1. Število iger v vsakem preiskusu je vidno v tabeli 5.3. Rezultati so prikazani na sliki 5.6.

Rezultati kažejo na to, da sta obe preiskušeni hevristiki koristni, saj se z izločitvijo poljubne izmed njiju odstotek zmag zmanjša. Varnost kralja se je izkazala za nekoliko pomembnejšo od preprečevanja promocij. Opazimo, da je pri preiskovanju v globino 0 razlika bolj očitna, kar je bilo tudi pričakovano (tako preiskovanje se močneje zanaša na dobro hevristično oceno pozicije).

Tabela 5.3: Število iger za vsako časovno omejitev

Globina 0	Globina 1
300	300



Slika 5.6: Graf prikazuje uspešnost različic programa z različnimi uporabljenimi hevristikami. Za vsako različico programa, pri različnih algoritmih, je zabeležen odstotek zmag. Upoštevane so samo igre z odločenim izidom (zmaga ali poraz).

Poglavje 6

Igranje otvoritev in končnic

Pri igranju otvoritev in končnic lahko programu pomagamo s specializiranim znanjem, specifičnim za ta del igre. Spremembe osnovnega delovanja računalniškega igralca in razlogi zanje so navedeni v naslednjih dveh podpoglavjih.

6.1 Otvoritve

Šahovski programi pri igranju klasičnega šaha uporabljajo otvoritveno knjižnico. To je drevo začetnih potez, za katere se je skozi izkušnje mnogih iger izkazalo, da so dobre. To privarčuje računanje teh potez v vsaki igri posebej ter na splošno izboljša kvaliteto igre. V našem programu smo uporabili isti pristop: iz tečaja na spletu[14] smo privzeli dobre začetne poteze za belega in črnega v prvih dveh krogih. Drevo bi lahko na isti način razvejali globlje do globine tri oziroma v nekaterih variacijah še globlje, vendar smo želeli več samostojne igre.

V prvih dveh krogih program naključno izbere izmed danih otvoritvenih potez na voljo. Z vključno tretjim krogom se igra nadaljuje po smernicah, opisanih v prejšnjih poglavjih. Alternativa naključnosti bi bila izbira otvoritvene poteze z najboljšim odstotkom zmag, vendar bi to zmanjšalo raznolikost iger.

Predlagane otvoritve smo preiskusili z igranjem najboljše verzije pro-

grama proti samemu sebi. Odigranih je bilo 300 iger s časovno omejitvijo 30 sekund na krog. Igre so bile razvrščene glede na to, s katero otvoritvijo so bile začete. Za vsako otvoritev smo izračunali odstotek dobljenih iger za belega. Pri tem smo šteli samo igre z odločenim izidom (remijev je bilo 8%). Poleg priporočenih otvoritev smo pri tem preiskusu v otvoritveno knjižnico dodali še nekaj drugih začetnih potez belega igralca, za katere vir [14] pravi, da so vprašljive. Priporočene otvoritve se začnejo s potezo 1.e2-e4 ali 1.d2-d4, alternativne pa z 1.d2-d3, 1.e2-e3, 1.f2-f3 ali 1.h2-h4. Rezultati so v tabeli 6.1.

Tabela 6.1: Delež zmag belega pri vsaki otvoritvi

Prvi krog	Drugi krog	Delež zmag belega igralca
1.e2-e4	2.d7-d5 f7-f6	50%
1.e2-e4	2.d7-d5 Sb8-c6	43%
1.e2-e4	2.d7-d5 d5xe4	50%
1.e2-e4	2.d7-d5 Sg8-h6	83%
1.d2-d4	2.d7-d5 c7-c6	50%
1.d2-d4	2.d7-d5 h7-h5	44%
1.d2-d4	2.d7-d5 Sb8-c6	39%
1.d2-d4	2.c7-c5 c5xd4	64%
1.d2-d3	2.d7-d5 Sg8-f6	27%
1.e2-e3	2.e7-e5 Sg8-h6	20%
1.f2-f3	2.d7-d5 c7-c5	25%
1.h2-h4	2.e7-e5 e5-e4	29%

Iz rezultatov najprej opazimo, da imajo vse alternativne začetne poteze belega dober odgovor črnega, ki vodi do slabih rezultatov za belega igralca. Vse kaže na to, da sta 1.e2-e4 in 1.d2-d4 edini pravi možnosti za prvo potezo belega. V primeru, da beli igralec začne eno izmed teh dveh potez, ima glede na to testiranje 55% verjetnost zmage. Odgovor črnega 2.d7-d5 Sg8-h6 je bil sicer priporočen, vendar nam je v praksi dal slabe rezultate (beli navadno nadaljuje z 3.Dd1-g4 Dg4-c8 Dc8-d8+). Če črni tega odgovora nikoli ne odigra, ima beli 49% verjetnost zmage. Ta rezultat priča o tem, da je igra s pravilno izbiro otvoritve uravnotežena za oba igralca.

6.2 Končnice

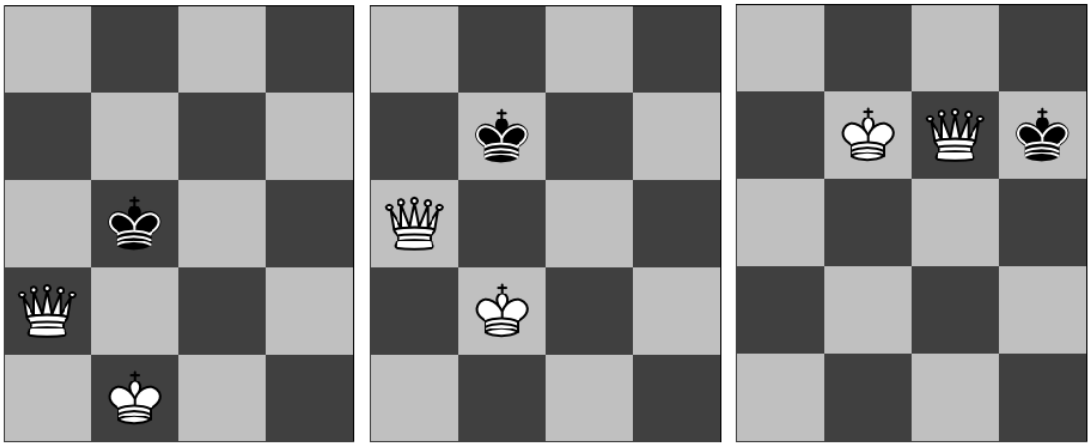
V tem poglavju bomo predstavili nekatere osnovne končnice. Za zmago v nekaterih izmed teh pozicij je potrebno znanje posebnih vzorcev matiranja. Poznavanje tovrstnih vzorcev je pomembno za razvoj posebnih hevristik, ki bodo znale voditi algoritem k zmagi. Pri vseh opisanih pozicijah ima nasprotnik samo kralja in morda nekaj kmetov; v primeru da bi imel zraven še kaj drugega materiala, je tega navadno enostavno vzeti, in nato nadaljevati z enostavnejšo končnico.

Zanimiva lastnost te šahovske različice je, da lahko v nekaterih pozicijah matira samo črni, beli pa se mora v zrcalni poziciji zadovoljiti z remijem. Razlog za ta pojav je v tem, da ima beli vedno liho število potez in zato v pozicijah, kjer ima kralj le dve možni polji, ne more končati kroga na začetnem polju. Enako pa po drugi strani ne velja za črnega, saj ima ta vedno sodo število potez. Črni lahko torej v nekaterih končnicah prisili belega kralja k premiku na drugo polje, medtem ko beli v enaki oz. zrcalni poziciji nima te možnosti. V nadaljevanju bomo pokazali tudi nekaj takih vzorcev.

6.2.1 Osnovne končnice

Mat s kraljico

Ta mat je najenostavnejši in ga lahko dosežeta oba igralca. Ima enostaven vzorec matiranja: vsakič se s svojo kraljico postavimo diagonalno ob nasprotnikovega kralja (glej sliko 6.1) in kraljico zaščitimo z našim kraljem. Na tak način kralja v vsakem koraku spravimo nekoliko bolj proti robu. Ko je enkrat na robu, matiranje postane trivialno. Pripomnimo, da to ni edini vzorec; alternativno lahko uporabimo enak vzorec kakor s trdnjavo, opisan v nadaljevanju.



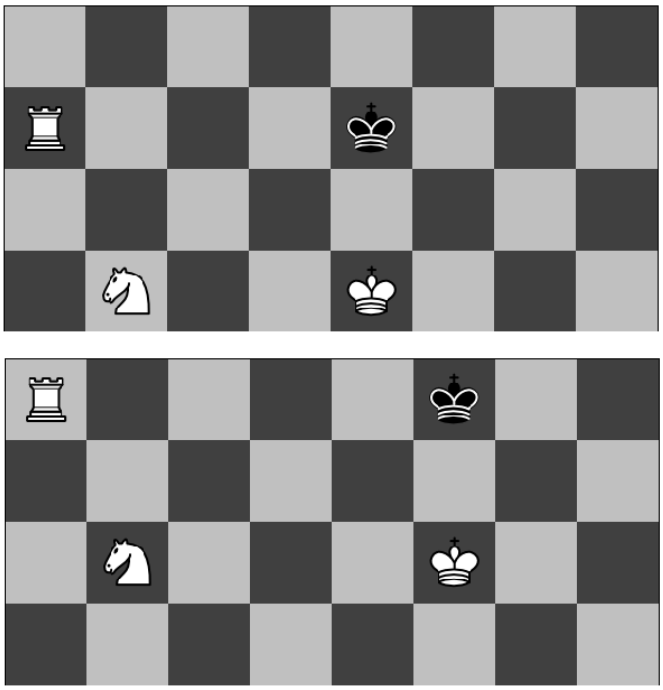
Slika 6.1: Kraljica se postavi tako, da sili kralja proti trdnjavi. Na drugi sliki se je kralj umaknil napadu, vendar ga v naslednjem krogu kraljica ponovno na isti način napade. S tem postopkom lahko kralja spravimo na rob in ga nato matiramo, kot lahko vidimo na tretji sliki.

Mat s trdnjavo in figuro

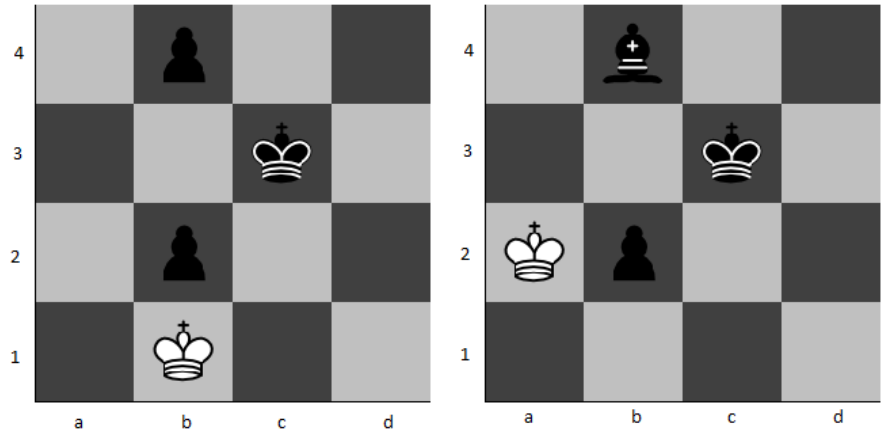
V nasprotju s klasičnim šahom, mat samo s trdnjavo ni možen, saj ni mogoče hkrati braniti trdnjave in prisiliti nasprotnega kralja k robu. Lahko pa to naredimo, če imamo še eno dodatno poljubno figuro. Vzorec matiranja je enak kakor pri klasičnem šahu, s to razliko, da z dodatno figuro branimo trdnjavo. Svojega kralja postavimo nasprotno njegovemu, dve vrstici stran, svojo trdnjavo pa na isto vrstico s kraljem (glej sliko 6.2).

Končnica s kmetom

Končnice z enim kmetom se navadno končajo z remijem, saj lahko nasprotnik blokira kmeta in nato vsak svoj krog ponovno konča na istem polju. Izpostavimo samo eno izjemo, ki ilustrira zakaj, nekatere končnice lahko zmaga samo črni. Na sliki 6.3 ima beli kralj na voljo le polja b1 in a2. Ne glede na to koliko potez ima beli, bo moral končati svoj krog na polju a2. Črni lahko nato promovira in zmaga. Zrcalna pozicija bi se končala z remijem.



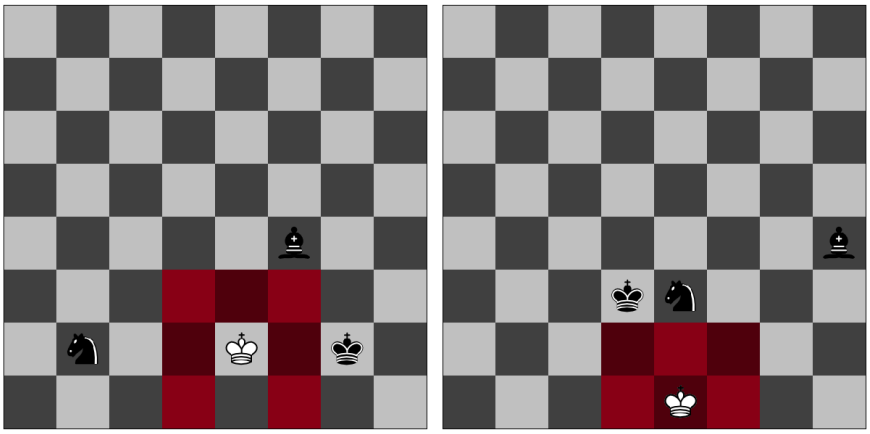
Slika 6.2: Kombinacija kralja in trdnjave prisili kralja na zgornji sliki bližje robu. Postopek ponovimo, dokler ne pridemo do mata na spodnji sliki.



Slika 6.3: Pozicija na levi je dobljena za črnega, ne glede na to kdo je trenutno na potezi. Beli bo moral svoj krog končati na polju a2 (na desni) in tako črnemu prepustiti promocijo.

Mat z dvema lahkima figurama

V nasprotju s klasičnim šahom je v 1-2-3 šahu mogoče matirati z dvema lahkima figurama (dva skakača, dva lovca, skakač in lovec), vendar lahko le matiranje s kombinacijo dveh lovcev uspešno izvedeta oba igralca. Ostala dva vzorca matiranja lahko izvede le črni. Vsi trije vzorci so relativno zapleteni in so bolj obširno opisani na spletu[15]. Za primer podamo kombinacijo na sliki 6.4. Samostojno najti te vzorce matiranja je zanimiv problem za hevristiko v končnici.

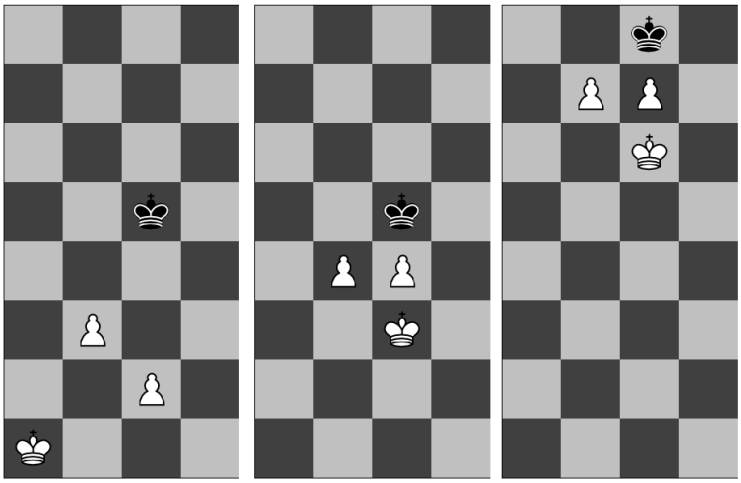


Slika 6.4: Rdeče pobarvana polja so pokrita s strani črnih figur. V poziciji na levi beli kralj lahko izbira le med dvema poljema in zaradi lihega števila potez konča na robu, kjer je nato matiran (na desni). Ti dve poziciji sta le del matnega vzorca, ki pripelje kralja iz sredine do roba.

Končnica z dvema povezanima kmetoma

Končnice z nepovezanimi kmeti se navadno končajo z remijem. Nasprotnikov kralj ima navadno dovolj potez, da vzame vse nebranjene kmete, ter nato blokira branjenega kmeta. Dva povezana kmeta sta lahko branjena hkrati, ter lahko v kombinaciji s kraljem matirata nasprotnika. Primer je ponazorjen na sliki 6.5. Črni kralj ima na izbiro biti matiran s strani kmetov ali pa jih

spustiti do promocije in nato izgubiti proti kraljici. Enaka izjema sta dva kmetaeno polje narazen, ki prav tako (če sta branjena in če nasprotnikov kralj ni v patu) vodita do zmage.



Slika 6.5: Kralj lahko prostovoljno vkoraka v mat, ali pusti promocijo.

6.2.2 Hevristike za končnice

V tem podpoglavju bomo predstavili hevristike za igranje končnic. Za končnico bomo vzeli vsako pozicijo, kjer ima vsaj eden izmed igralcev eno ali manj figur (brez štetja kmetov). Ta odločitev je nekoliko arbitrarna, vendar poenoti vse v prejšnji sekciji naštete pozicije. V teh pozicijah algoritem uporabi tukaj definirane hevristike, namesto klasičnih iz poglavja 5.1. Njegova igra v treh fazah sicer ostane enaka. Izkaže se, da s pomočjo teh namenskih hevristik lahko zmaga vse naštete osnovne končnice, brez dodatnega znanja o vzorcih matiranja.

Navadne hevristike

Nekatere hevristike iz poglavja 5.1 so uporabne tudi v končnici in se v isti obliki uporabijo tudi tukaj. To so *Material*, *Promocija* in *Izogibanje lovcu*.

Obramba

Svoje figure je navadno načeloma nesmiselno braniti pred nasprotnikovimi figurami, lahko jih pa branimo pred nasprotnikovim kraljem. Kakor pri običajnem šahu, kralj v končnici postane pomembna figura, zato se pojavi pomembnost zaščite lastnih figur pred njim. Hevristika najde vsa polja, ki jih nasprotnikov kralj lahko legalno doseže tekom naslednje poteze. Najde tudi vsa polja, ki so nezaščitena, nato kaznuje evalvacijo pozicije za vsako nezaščiteno dosegljivo polje. Dosegljivost igra pomembno vlogo, saj lahko večkrat implicitno zaščitimo figure tako, da jih naredimo nedosegljive. Primer je na sliki 6.4, kjer sta skakač in lovec nezaščitena, a nedosegljiva.

Pokritost

Pokritost je ena izmed matnih hevristik, ki pomaga tudi najti vzorce matiranja. Poleg tega promovira napad na kralja, kar je koristno tudi, ko nasprotnikov kralj poskuša blokirati kmeta.

Bližina roba

Hevristika najde vsa polja, na katerih lahko nasprotnikov kralj legalno konča krog. Opozorimo na distinkcijo s polji iz *Obrambe*: tukaj štejemo samo končna polja in zato lihost ali sodost števila potez pride do izraza. Nato izmed vseh polj najde tisto, ki je najbližje središču šahovnice. Oceno pozicije kaznuje glede na oddaljenost tega polja do roba. Hevristika torej ugotavlja, koliko blizu centra bo v najslabšem primeru kralj v naslednjem krogu. Pri večini matnih vzorcev se razdalja do roba zmanjša za ena vsak krog. Izkaže se, da je ta hevristika dovolj, da algoritem najde vse osnovne matne vzorce. Ponovno vzemimo primer na sliki 6.4. Kralj ima samo dve možni polji, samo eno izmed njiju je lahko končno. Tudi v najslabšem primeru bo imel razdaljo do roba 0.

Bližina središču

Kralj je nagrajen, čim bližje središču se nahaja. Nagrada je majhna, ampak v primeru, da kralj nima druge dolžnosti, denimo preprečevanja promocij, je položaj stran od roba tipično bolj oddaljen od mata.

Pat

Hevristika preveri, ali ima nasprotnik kakšno legalno potezo, in če je nima, ustrezno oceni pozicijo.

Poglavje 7

Zaključek in nadaljnje delo

Cilj magistrske naloge je bil izdelati dober program za igranje 1-2-3 šaha. Ta ima, zaradi kombinatorične eksplozije možnosti, ki jo povzroči vrsta zaporednih potez, ekstremno velik faktor razvejanosti. To naredi igro unikatno in zanimivo ter ustvari zelo zahtevno domeno, v kateri lahko raziskovalci testirajo nove algoritme in ideje.

Naš program sledi splošno priporočeni stategiji za to igro: (1) poišče mat, če je le ta izvedljiiv v danem številu potez, (2) v primeru, da mata ni, generira potencialno dobre sekvence potez, (3) odigra najboljšo izmed omenjenih sekvenc, ki hkrati preprečuje nasprotnikov mat. Prvo in drugo točko smo obravnavali kot dva ločena podproblema. Tretja je po vsebini sorodna prvi in uporablja isti algoritem za iskanje matov.

Pri iskanju matov smo osnovali in testirali več hevristik in algoritmov. Najboljša hevristika *Pokritost* je štela kolikokrat so napadena (in tako pokrita) polja okoli kralja. Z uporabo *Pokritosti* in algoritma A* je bil program sposoben zelo učinkovito najti mate v naši testni množici velikosti 900 matnih pozicij. Več kot 98% izmed testiranih matov je bilo najdeno v manj kot sekundi, vsi pa so bili najdeni v manj kot desetih. Pokazali smo tudi, da s povečevanjem učinkovitosti iskalca matov močno naraste delež zmag programa.

Za igranje druge faze smo razvili nekaj raznovrstnih hevristik; ocenjevale

so količino materiala v poziciji, varnost kralja, možnost promocij kmetov, razvoj figur in izogibanje figur pred nasprotnikovim lovcem. Uporabili smo uteženo vsoto teh hevristik in za dve izmed njih (*Promocija* in *Varnost kralja*) tudi testirali njihov vpliv na kvaliteto igre. Razvili in preiskusili smo še nekaj algoritmov, primernih za iskanje v prostoru z ogromno razvejanostjo. Za najboljšo se je izkazala kombinacija algoritma A* in algoritma Min-Max, ki preiskuje svoje poteze in še potencialne odgovore nasprotnika, nato pa preiskovanje zaključi.

Razvili smo tudi primeren nabor hevristik, specializiranih za igranje končnic, ki jih v igranju druge faze v končnicah uporabimo namesto zgoraj naštetih. Te hevristike so sposobne zmagati vse v tem delu obravnavane osnovne končnice in se dobro obnesejo tudi v bolj zapletenih končnicah.

Pogledali smo si v literaturi predlagane otvoritve, jih upoštevali pri osnovanju otvoritvene knjižice in s pomočjo iger programa samega proti sebi pokazali njihovo učinkovitost. Skoraj vse predlagane otvoritve so se empirično izkazale nad alternativami. S temi ekperimenti smo ugotovili tudi, da je igra precej verjetno uravnovešena za oba igralca - v eksperimentu (odigrano je bilo 300 partij) je s pravo izbiro otvoritve beli zmagal približno polovico (49%) vseh odločenih iger.

Implementirali smo program z grafičnim vmesnikom, ki igro z uporabo zgoraj naštetih metod uspešno igra. Z njim lahko človek igra proti drugemu človeku ali proti računalniškemu nasprotniku. Omogoča vnos poljubne pozicije na šahovnico in nato nadaljevanje igre iz nje, oziroma iskanje mata iz dane šahovske pozicije v podanem številu potez. Omogoča tudi shranjevanje igre in nalaganje shranjenih iger. Program je prosto dostopen na spletu[7].

V prihodnosti bi si želeli najti način, kako ta program vključiti v prave turnirje, kjer bi se lahko pomeril z najboljšimi igralci v 1-2-3 šahu.

V nadaljnem delu bomo poskušali avtomatsko ugotoviti dejanske vrednosti figur (v tem delu smo uporabili vrednosti iz klasičnega šaha). Zanimiva rešitev za ta izziv je opisana v delu[16], kjer se vrednosti v variacijah šaha učimo s spodbujevalnim učenjem. Dodatno imamo namen preiskusiti Monte-

Carlo metode preiskovanja, ki se navadno dobro izkažejo v igrah z visoko razvejanostjo. Le-te smo že poskusili implementirati, zaenkrat sicer z neobetajočimi rezultati, vendar je potrebno dodatno delo za potrditev ali ovržbo njihove učinkovitosti. Bolj konkretno, za iskanje matov bi lahko poskusili uporabiti *Nested Monte-Carlo Search* [17] in za iskanje potencialno dobrih sekvenc potez, kjer mata ni, *Monte-Carlo Tree Search* [18].

Literatura

[1] D. Pritchard, Popular chess variants, BT Batsford Limited, 2000.

[2] D. B. Pritchard, J. D. Beasley, The classified encyclopedia of chess variants, J. Beasley, 2007.

[3] A. Castelli, Scacchi Progressivi. Matti Eccellenti (Progressive Chess. Excellent Checkmates), Macerata-Italy, 1996.

[4] A. Castelli, Scacchi progressivi. Finali di partita (Progressive Chess. Endgames), Macerata-Italy, 1997.

[5] M. Leoncini, R. Magari, Manuale di Scacchi Eterodossi, Siena-Italy, 1980.

[6] G. Dipilato, M. Leoncini, Fondamenti di Scacchi Progressivi, Macerata-Italy, 1987.

[7] V. Janko, M. Guid, `http://www.ailab.si/progressive-chess/`.

[8] J. Beasley, Progressive chess: How often does the "italian rule" make a difference?, `http://www.jsbeasley.co.uk/vchess/italian_rule.pdf`, [obiskano 06-Marec-2015] (2011).

[9] S. Chinchalkar, An upper bound for the number of reachable positions, ICCA Journal 19 (3) (1996) 181–183.

[10] D. J. Wu, Move ranking and evaluation in the game of arimaa, Ph.D. thesis, Harvard University, Cambridge, USA (2011).

[11] T. Kozelek, Methods of MCTS and the game arimaa, Master's thesis, Charles University, Prague, Czech Republic (2010).

[12] Forsyth–edwards notation, https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation, [obiskano 21-Avgust-2015].

[13] Minimax, https://en.wikipedia.org/wiki/Minimax, [obiskano 28-Avgust-2015].

[14] Cazzeo, Progressive chess 7: Theory, https://www.youtube.com/watch?v=CLeQs7ZkSQU, [obiskano 21-Avgust-2015].

[15] Cazzeo, Progressive chess 6: Endgame theory, https://www.youtube.com/watch?v=aJ1Iej4b9Tk, [obiskano 28-Avgust-2015].

[16] S. Droste, J. Fürnkranz, Learning of piece values for chess variants, Tech. rep., Tech. Rep. TUD–KE–2008-07, Knowledge Engineering Group, TU Darmstadt (2008).

[17] T. Cazenave, Nested monte-carlo search., in: IJCAI, Vol. 9, 2009, pp. 456–461.

[18] G. Chaslot, Monte-carlo tree search, Maastricht: Universiteit Maastricht.

[19] Bitboards, https://chessprogramming.wikispaces.com/Bitboards, [obiskano 21-Avgust-2015].

[20] General setwise operations, https://chessprogramming.wikispaces.com/General+Setwise+Operations, [obiskano 28-Avgust-2015].

Dodatek A

Implementacija bitnih šahovnic

V tem poglavju se bomo ukvarjali s predstavitvijo šahovnice v programu. Ta predstavitev vpliva na časovno učinkovitost skoraj vseh ostalih delov programa, zato je njena izbira ključnega pomena. Najbolj intuitivna možnost je šahovnico narediti kot seznam dolžine 64; njegovi elementi utrezajo ustrezni figuri na šahovnici. Primer začetne pozicije s tako predstavitvijo lahko vidite v tabeli A.1. Slabost tega pristopa je v tem, da je potrebno za vsako analizo šahovnice (generacija potez, preverjanje matov, izračunavanje različnih hevristik) iterirati skozi vsa polja, včasih celo večkrat. Alternativa, ki se je pojavila zelo hitro v svetu šaha in jo sedaj uporabljajo vsi vodilni šahovski programi, je bitna šahovnica (Bitboard). V primeru našega programa je tranzicija na implementacijo z bitnimi šahovnicami program pospešila za približno faktor 5.

A.1 Definicija

Vzemimo neko 64-bitno število; taka števila so v programskih jezikih običajna (navadno označena *long*), saj se lepo skladajo s 64- (ali 32-) bitno procesorsko arhitekturo. Po drugi strani so za šah priročna, saj imajo enako bitov, kakor šahovnica polj. Vsak bit takega števila si lahko predstavljamo kot binarno informacijo o nekem polju (recimo prisotnost ali odsotnost figure). Tako

-4	-2	-3	-5	-6	-3	-2	-4
-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
4	2	3	5	6	3	2	4

Tabela A.1: Šahovnica je predstavljena kot eno- ali dvo dimenzionalni seznam dolžine 64. Prikazana je začetna pozicija. Vrednost 1 tukaj predstavlja kmeta, 2 skakača, 3 lovca itd... Negativne vrednosti so nasprotnikove figure.

interpretirano 64-bitno število imenujemo bitna šahovnica.

Za primer si poglejmo število 129, ki ga v binarnem lahko zapišemo kot:

$$129 = 0000000000000000000000000000000000000000000000000000000010000001$$

Če to število ustrezno prelomimo, ter za lažjo vizualizacijo ničle nadomestimo s piko, (tabela A.2) vidimo, da enice ustrezajo pozicijam trdnjave belega igralca v začetni poziciji.

V enem samem številu smo shranili vso informacijo o tem, kje stojijo

.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
1	.	.	.	.	.	.	1

Tabela A.2: Vizualizacija števila 129 v binarnem zapisu.

(in sorodno, kje ne stojijo) vse bele trdnjave. Trenutno pozicijo v igri lahko tako opišemo z dvanajstimi števili; za vsako kombinacijo tipa in barve figure enega. Navadno si shranimo tudi števila, ki predstavljajo vsa prazna polja, vsa polna polja, vsa polja z belimi figurami in vsa polja s črnimi figurami. Pri izračunavi funkcij o dani poziciji si lahko izračunamo še raznovrstne druge bitne šahovnice, recimo vsa napadena polja, vsa branjena polja, vsa polja okoli kralja itd...

Na videz s tem pristopom nismo nič pridobili, razen nekoliko prostorskega prihranka, vendar bomo videli, da se operacije na teh številih izvajajo na vseh bitih hkrati, kar pomeni, da z eno samo procesorsko operacijo obdelamo celotno šahovnico.

A.2 Operacije

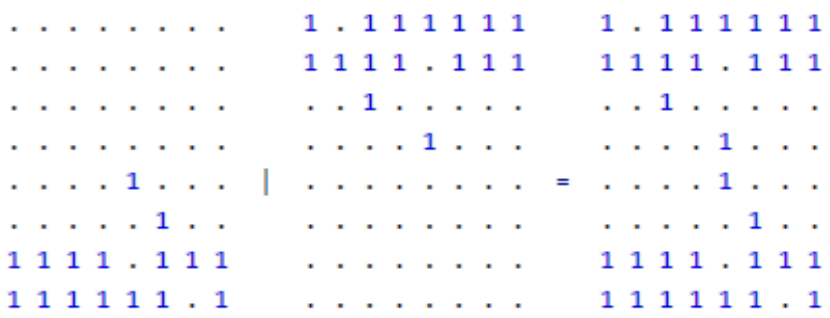
Tukaj predstavimo nekatere osnovne operacije nad bitnimi šahovnicami; za celoten pregled ostalih operacij in njihove uporabe lahko bralec pogleda na [19].

Kadar na število gledamo kot na niz logičnih vrednosti (izraženih kot 0 ali 1), lahko na njem izvajamo logične operacije, kot so IN, ALI, NEGACIJA itd... Logične operacije se v procesorju izvedejo nad vsemi istoležnimi elementi hkrati. Za primer si poglejmo logični ALI (v programskih jezikih navadno označen z |).

$$1\ 0\ 0 \mid 0\ 1\ 0 \ = 1\ 1\ 0$$

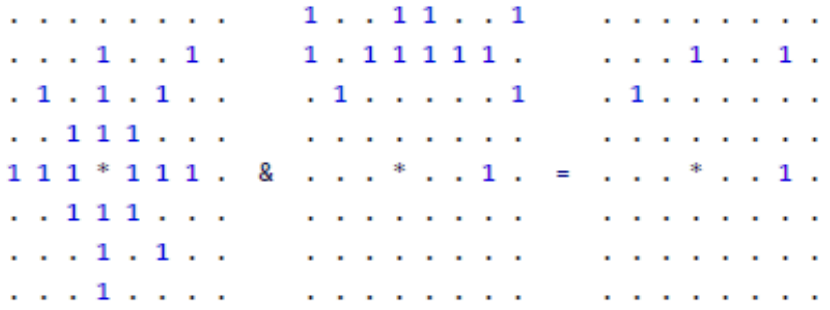
Vzemimo še večji primer, kjer operacijo ALI izvedemo nad dvemi bitnimi šahovnicami - belimi figurami in črnimi figurami. Rezultat te operacije (lahko jo obravnavamo kot unijo) je bitna šahovnica, ki predstavlja vsa zasedena polja (slika A.1).

Naslednji primer (slika A.2) pokaže operacijo IN (v programskih jezikih navadno označeno z &) med dvemi bitnimi šahovnicami - napadi bele kraljice in črnimi figurami. Razultat te operacije (lahko jo obravnavamo kot presek) je bitna šahovnica z označenimi vsemi napadenimi črnimi figurami. Vidimo,



Slika A.1: Polja z belimi figurami ALI polja s črnimi figurami = zasedena polja. Slika vzeta iz speta[20].

da lahko množico tako napadenih figur hitro in enostavno najdemo.



Slika A.2: Napadi kraljice IN polja s črnimi figurami = napadena polja. Slika vzeta iz spleta[20].

Operacija negiranja zamenja vrednosti logičnih spremenljivk.

$$\neg 010 = 101$$

.

Nazadnje si poglejmo še logični premik (v programskih jezikih navadno označen z << ali >>). Ta operacija premakne vse bite v levo ali desno za določeno število mest. Primer:

$$00000110 << 4 = 01100000$$

Specifično premik za 8 v eno ali drugo smer pomeni prestavitev vseh bitov na šahovnici eno vrstico višje ali nižje. To operacijo bomo uporabili v naslednjem sklopu, tako da definirajmo funkcijo:

$$sever(x) = x << 8$$

Sorodno lahko definiramo funkcije severoVzhod, severoZahod in ostale smeri neba.

A.3 Primer promocije

Iz prej definiranih elementarnih operacij bomo sedaj sestavili funkcijo za ugotavljanje, ali beli kmeti lahko promovirajo v točno n potezah. Z 'navadno' šahovnico moramo preveriti vse možne poti, za vsakega kmeta posebej. Kmet ima lahko več različnih poti do cilja, saj lahko na poti jemlje figure na levi ali na desni strani. Upoštevati moramo tudi to, da je dajanje šaha po poti nelegalno in take poti zavreči. Izkaže se, da se z bitnimi šahovnicami lahko izognemo kombinatorični preiskavi in problem rešimo v nekaj enostavnih korakih. Algoritem je podan v psevdokodi 4. Algoritem je enostavno spremeniti tako, da gleda promocije kmetov v n ali manj potezah, oziroma da vrne najmanjši n, v katerem lahko pride do promocije.

Algorithm 3 Algoritem za prepoznavanje možnih promocij v n potezah

```
1: kmetje ← beliKmetje
2: napadNaKralja ← jugoVzhod(črniKralj) | jugoZahod(črniKralj)
3: for n korakov do
4:     Premiki ← sever(kmetje) & praznaPolja
5:     Napadi ← (severoVzhod(kmetje) | severoZahod(kmetje)) & črneFigure
6:     kmetje ← (premiki | napadi) &(¬napadNaKralja)
7: end for
8: return (kmetje & vrsta8)! = 0
```

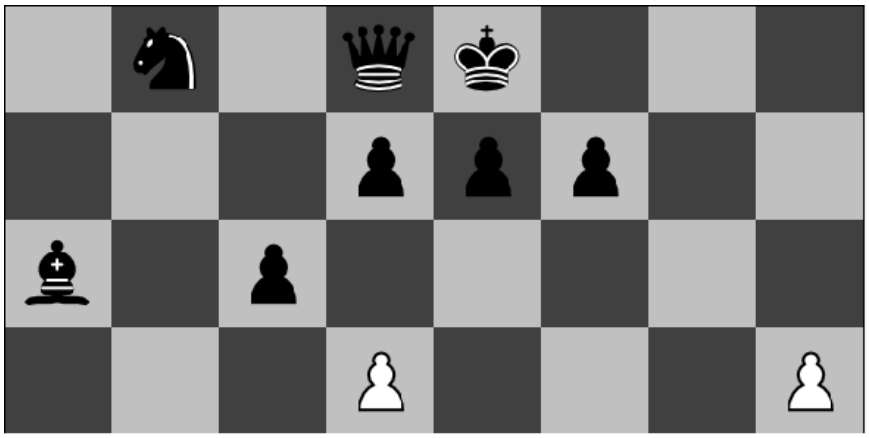
Vse spremenljivke, razen števca n, so bitne šahovnice. Spremenljivka *kmetje* predstavlja tista polja, ki jih kmetje lahko dosežejo po i-iteracijah

zanke (kjer je i med 0 in n). Pred prvo iteracijo ($i = 0$) je spremenljivka postavljena na trenutno pozicijo belih kmetov. Spremenljivka *napadNaKralja* predstavlja prepovedana polja, na katera kmetje ne smejo iti, sicer bi dali šah.

V vsaki iteraciji se izračunajo premiki kmetov naprej; to so prazna polja, ki so hkrati za polje bolj severno od trenutnih kmetov. Izračunajo se napadi kmetov; to so polja po diagonali naprej levo ali desno, ki so hkrati zasedena s črnimi figurami. Na koncu se izračuna pozicija kmetov v novi iteraciji, kot unija premikov in napadov, ki hkrati niso del spremenljivke *napadiNaKralja*.

Spremenljivka *vrsta8* je vnaprej izračunana vrednost, ki je pozitivna samo na poljih osme vrste. Presek te spremenljivke s pozicijo kmetov v n-ti iteraciji nam da pozicije kmetov, ki v n-ti iteraciji promovirajo.

Za primer si vzeimo pozicijo na sliki A.3. Tabele A.3, A.4, A.5, A.6 predstavljajo vrednost spremenljivke *kmetje*.



Slika A.3: Pozicija na kateri bomo simulirali algoritem. Ali lahko beli promovira v treh potezah? Na katerih poljih?

.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	1	.	.	.	1	
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Tabela A.3: Originalne pozicije kmetov.

.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	1	1	.	.	.	1	
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Tabela A.4: Možne pozicije kmetov po enem premiku. Desni kmet neovirano nadaljuje proti promociji, levi kmet ima izbiro dveh različnih polj; navaden premik in rušenje figure.

.	.	.	.	.	.	.	.
.	.	1	.	1	.	.	1
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Tabela A.5: Možne pozicije kmetov po dveh premikih. Levi kmet ima ponovno dve možnosti, vsaka izmed prejšnjih dveh je porodila eno novo legalno možnost.

.	1	1	1	.	.	.	1
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Tabela A.6: Polja na katerih lahko beli promovira. Hkrati so to možne pozicije kmetov po treh premikih.

Dodatek B

Shranjevanje transpozicij

Pri vseh uporabljenih preiskovalnih algoritmih se poskušamo izogniti transpozicijam; t.j. pozicijam, ki smo jih dosegli po drugi poti, in že preiskali. V ta namen želimo najti način, kako shraniti vse preiskane pozicije. Intuitivna možnost bi bila shraniti FEN notacijo vsake preiskane pozicije, vendar se izkaže, da je račun le-teh prepočasen in velikost tega niza prevelika.

Naš program smo pospešili z uporabo druge, že obstoječe zgoščevalne funkcije: zgoščevanjem Zobrist (Zobrist hashing). Ta funkcija vsaki poziciji predpiše 64-bitno število na naslednji način:

Pri inicializaciji generira naključno 64-bitno vrednost za vsako kombinacijo polja in figure (torej 64*12 vrednosti).

Algorithm 4 inicializiraj_zobrist

```
1: tabela ← 2-D tabela velikosti 64x12
2: for i od 1 do 64 do
3:     for j od 1 do 12 do
4:         tabela[i][j] ← naključna 64-bitna vrednost
5:     end for
6: end for
```

Vrednost pozicije je definirana z algoritmom 5. Vrednost je torej XOR med vsemi med inicializacijo shranjenimi števili, ki se pojavijo v dejanski poziciji.

Algorithm 5 vrednost_zobrist

```
1: h ← 0
2: for i od 1 do 64 do
3:     if polje i ni prazno then
4:         j ← index figure na polju i
5:         h ← h XOR tabela[i][j]
6:     end if
7: end for
8: return h
```

Računanje te funkcije lahko pospešimo tako, da si vrednost shranimo, nato pa jo posodobimo vsakič, ko se pozicija spremeni (primer algoritma pri premiku figure na prazno polje 6). Pri tem izkoristimo dejstvo, da je x XOR y XOR $y = x$ za vsak x in y.

Algorithm 6 posodobitev_zobrist(vrednost h, premik iz polja i na polje j

```
1: z ← index figure na polju i
2: h ← XOR tabela[i][z]
3: h ← XOR tabela[j][z]
4: return h
```

Na isti način lahko v vrednost funkcije dodamo druge informacije o poziciji, npr... koliko potez še imamo, tako da za vsako možno stanje te informacije generiramo naključno število, ter nato vrednosti dodamo ustrezen XOR člen.