Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matic Plut

Prepoznavanje pozicijskih motivov v šahovskih pozicijah

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matej Guid

Ljubljana, 2018

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Glavni cilj naloge je razviti in opisati postopek za samodejno prepoznavanje pozicijskih motivov v šahovskih pozicijah. Dodaten izziv pa je najdene šahovske motive uporabiti pri samodejnem iskanju podobnih šahovskih pozicij. Kandidat naj pri svojem delu uporablja programsko knjižnico *python-chess* in okolje *Jupyter Notebook*, za prepoznavanje podobnih pozicij pa odprtokodno knjižnico za iskanje po besedilih *Apache Lucene*. Povzete naj bodo glavne ideje postopka za samodejno prepoznavanje šahovskih motivov in opisane tehnike, ki se pri tem uporabljajo (npr. bitne šahovnice). Razložen naj bo tudi postopek iskanja podobnih šahovskih pozicij s pomočjo tehnik, ki se sicer uporabljajo za iskanje po besedilnih datotekah.

Zahvalil bi se mojemu mentorju doc. dr. Mateju Guidu za vso podporo pri izdelavi diplomske naloge. Posebna zahvala gre moji družini za spodbudo skozi vsa leta šolanja.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
QBE	query-by-example	Poizvedba s primerom
CQL	Chess Query Language	Šahovski poizvedovalni jezik
IR	Information Retrieval	Pridobivanje informacij
SAN	Standard Algebraic Notation	Algebraična notacija
FIDE	World Chess Federation	Svetovna šahovska federacija
FEN	Forsyth-Edwards Notation	Forsyth-Edwardsov zapis
PGN	Portable Game Notation	Prenosljivi zapis igre

Povzetek

Naslov: Prepoznavanje pozicijskih motivov v šahovskih pozicijah

Avtor: Matic Plut

V diplomski nalogi smo najprej pripravili program za samodejno prepoznavanje pozicijskih motivov. Usmerili smo se v prepoznavanje kmečkih struktur, kot so osamljeni kmetje, prosti kmetje, podvojeni kmetje, zaostali kmetje, kmečke verige in kmečki otoki. Program smo implementirali v programskem jeziku *Python* in s pomočjo knjižnice *python-chess*, ki omogoča enostavno uporabo bitnih šahovnic. Razviti program bi lahko uporabili denimo za samodejno komentiranje šahovskih partij.

V nadaljevanju smo naslovili problem iskanja vsebinsko podobnih šahovskih pozicij. Samodejno prepoznavanje kmečkih struktur smo kot komponento vključili v program, ki je sposoben poiskati vsebinsko podobne šahovske pozicije, pri tem pa upošteva tudi lastnosti kmečkih struktur na šahovnici. Uporabili smo tehniko iskanja po besedilnih datotekah, pri čemer nam je bila v pomoč odprtokodna knjižnica *Apache Lucene*. Razvoj je delno potekal v okolju *Jupyter Notebook*, ki med drugim omogoča tudi enostavno prikazovanje šahovskih pozicij.

Iskanje vsebinsko podobnih pozicij lahko uporabimo denimo pri samodejni pripravi šahovskih treningov, s čimer bi močno olajšali delo šahovskim trenerjem. Algoritem za iskanje podobnih pozicij bi lahko še izboljšali s pomočjo prepoznavanja dinamičnih, taktičnih motivov. Kompetentnost programa za ugotavljanje vsebinsko podobnih pozicij je bila potrjena s pomočjo ankete med šahisti.

Ključne besede: umetna inteligenca, šah, pozicijski motivi, kmečke strukture, bitne šahovnice, podobne šahovske pozicije, poizvedba z vzorcem.

Abstract

Title: Recognition of positional motifs in chess positions

Author: Matic Plut

In this diploma thesis, we first prepared a programme for automatic recognising of positional motifs. We focused on recognising of pawn structures, such as isolated pawns, passed pawns, doubled pawns, backward pawns, pawn chains, and pawn islands. The programme was implemented in the *Python* programming language with the help of the library *python-chess* that enables simple use of bitboards. The developed programme could be used for automatic annotation of chess games.

Then, we addressed the problem of searching for chess positions with similar content. The automatic recognising of pawn structures was integrated into the programme as a component, wherein the programme is able to find chess positions with similar content, taking into account the characteristics of pawn structures on a chessboard as well. We used techniques for information retrieval from text documents, using an open-source library *Apache Lucene*. Partly, the development took place in the environment *Jupyter Notebook*, which enables simple visualization of chess positions.

The search of positions with similar content can be used for automatic preparation of chess trainings, whereby the job of chess trainers can be made easier. We could improve the algorithm for searching similar positions by recognising dynamic, tactical motifs. The competence of the programme for discovering positions with similar content was confirmed by a survey carried out among chess players.

Keywords: artificial intelligence, chess, positional motifs, pawn structures, bitboards, similar chess positions, query by example.

Poglavje 1

Uvod

Dandanašnji programi za igranje šaha so tako rekoč nepremagljivi tudi za najboljše šahiste na svetu [1]. Njihova slabost pa je razlaganje in komentiranje šahovskih partij, zato so lahko zaporedja potez računalnika človeku nerazumljiva, kar daje šahistu ob partijah z računalnikom občutek nemoči.

Programi za igranje šaha temeljijo na preiskovalnih algoritmih s področja umetne inteligence. Računalnik zna zelo hitro in globoko preiskati kombinacije potez in iz njih izluščiti tisto najboljšo [15]. Človeški možgani tega niso sposobni, zato si pomagajo z učenjem pozicij oziroma pozicijskih motivov, ki nam služijo kot indikatorji dobrih ali slabih pozicij. Tu pridemo do največjega razkoraka med človekom in računalnikom, to je oceno trenutne šahovske pozicije.

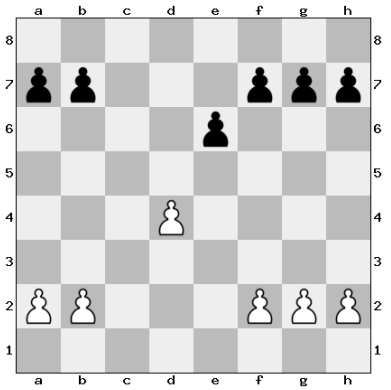
Računalnik za to oceno uporablja numerični sistem, šahist pa se skuša temu približati z upoštevanjem materiala na šahovnici in kar se da objektivno oceno postavitve figur. Pri objektivni oceni postavitve figur šahistu pomagajo tudi znani pozicijski motivi, ki so se skozi igranje šaha pokazali kot dobri ali slabi. Če bi računalniški programi znali prepoznavati te pozicijske motive in jih zapisati na šahistu prijazen način, bi lahko s tem pomagali pri boljšem razumevanju zapletenih sekvenc potez odigranih s strani šahovskih motorjev. Na ta način bi lahko računalnik olajšal delo šahovskih trenerjev, oziroma bi šahistom omogočil lažjo analizo partij.

1.1 Prepoznavanje šahovskih motivov

Obstaja zelo veliko šahovskih pozicijskih motivov. Nekateri so definirani bolj natančno, spet drugi pa imajo več različnih definicij oziroma navodil, kako jih prepoznati. Osredotočili se bomo na kmečke strukture (ang. *pawn structures*), saj kmetje, s svojo številčnostjo na šahovnici, predstavljajo "dušo" šahovske partije [20].

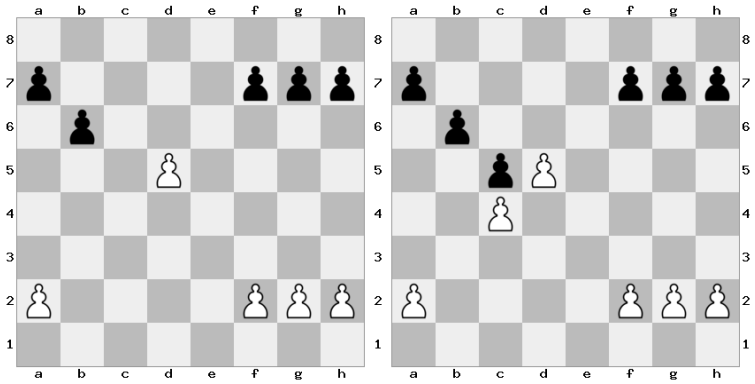
Navedimo nekatere najbolj pogoste kmečke strukture [18]:

- **osamljeni kmet** (ang. *isolated pawn*) – na njegovih sosednjih linijah ne najdemo niti enega kmeta iste barve (slika 1.1);



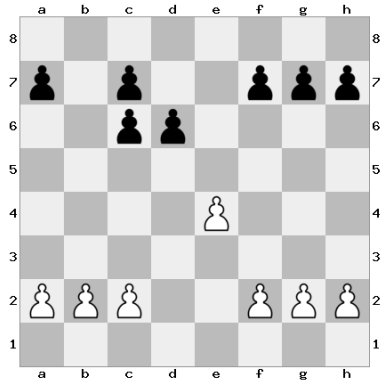
Slika 1.1: Primer osamljenega kmeta na polju *d4*.

- **prosti kmet** (ang. *passed pawn*) – pred njim, na njegovi ali sosednjih linijah ne najdemo nobenega kmeta nasprotne barve. Poznamo tudi branjenega prostega kmeta (ang. *connected passed pawn*). Njegova dodatna lastnost je, da je branjen z drugim kmetom (slika 1.2);



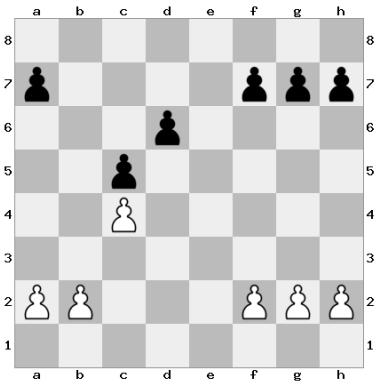
Slika 1.2: Levo, primer prostega kmeta na polju *d5*. Desno, primer branjenega prostega kmeta na polju *d5*.

- **podvojeni kmet** (ang. *doubled pawn*) – na njegovi liniji najdemo še vsaj enega kmeta iste barve (slika 1.3);



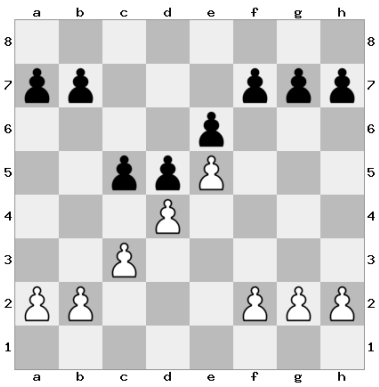
Slika 1.3: Primer podvojenih kmetov na poljih *c6* in *c7*.

- **zaostali kmet** (ang. *backward pawn*) – ni osamljen, ob njem se nahaja bolje napredovali kmet iste barve. Za seboj nima kmeta, ki bi ga lahko zaščitil. Po navadi se nahaja na drugi ali tretji vrsti oziroma na šesti ali sedmi, če je črne barve. Njegov premik naprej ni varen, onemogoča ga nasprotni kmet (slika 1.4);



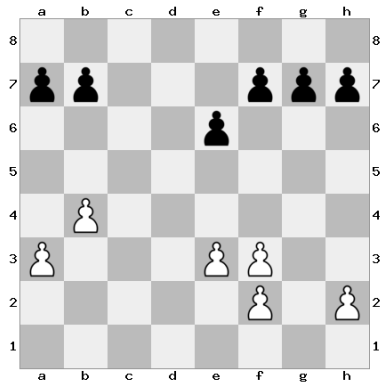
Slika 1.4: Primer zaostalega kmeta na polju *d6*.

- **kmečka veriga** (ang. *pawn chain*) – je serija treh ali več povezanih kmetov na diagonali. Kmetu, ki je v verigi pomaknjen najbolj nazaj, pravimo tudi koren verige (ang. *base*). Kmetu, ki je pomaknjen najbolj naprej, pa pravimo glava verige (ang. *wedge*) (slika 1.5);



Slika 1.5: Primer kmečkih verig: belo sestavljajo kmeti na poljih *b2* (koren), *c3*, *d4* in *e5* (glavo), črno pa kmeti na poljih *f7* (koren), *e6* in *d5* (glava).

- **kmečki otok** (ang. *pawn island*) – je skupina kmetov, ki je odrezana od ostalih kmetov (slika 1.6);



Slika 1.6: Primer, ko ima beli tri kmečke otoke, črni pa dva.

Poznamo še druge kmečke strukture, a bomo v nadaljevanju te diplomske naloge našo pozornost namenili zgoraj opisanim.

1.2 Komentiranje šahovskih partij

Močni šahovski programi so že zelo težki nasprotniki človeškim šahovskim velemojstrom. Kljub temu pa je njihova sposobnost *razlage*, zakaj so nekatere poteze dobre, nekatere pa slabe, precej omejena. Nekaj avtomatskih sistemov komentiranja šahovskih partij že poznamo. Napredek na tem področju še vedno močno zaostaja za močjo šahovskih programov. Tipični "komentarji" na nivoju predlaganja naslednje najboljše poteze in numerične ocene pozicije, so dokaj slaba pomoč šahistu, ki se želi naučiti pomembnih konceptov za posameznimi potezami [22].

Čeprav se bomo v diplomski nalogi ukvarjali z iskanjem podobnih pozicij, bomo predstavili nekatere koncepte, ki jih lahko uporabimo v namene izboljšanja komentiranja šahovskih partij. Z avtomatskim prepoznavanjem pozicijskih motivov lahko šahistu, na človeku bolj prijazen način razložimo posamezno pozicijo v partiji. S tem mu lahko omogočimo razumevanje numerične ocene pozicije. Prepoznavanje in zapis pozicijskih motivov pa je lahko dobra osnova za razširitev na prepoznavanje in zapisovanje dinamičnih

elementov šahovskih partij (npr. taktični motivi, ang. *tactical motifs*) [11].

1.3 Iskanje vsebinsko podobnih pozicij

Algoritmi za iskanje po obstoječih šahovskih bazah so v zadnjih letih izredno napredovali. Poznamo več iskalnih algoritmov in povpraševalnih jezikov (ang. *query language*) za namen iskanja podobnih položajev figur na šahovnici. Večina do sedaj uporabljenih pristopov temelji le na vizualnem prepoznavanju, posledično pa nastaja vse večja potreba po kontekstnem iskanju [25].

Obsotoječi sistemi za iskanje šahovskih pozicij, ki temeljijo na pristopu poizvedb s primerom (QBE, ang. *query-by-example*), so omejeni na iskanje enakih rezultatov [26]. To lahko pomeni, da nekatere poizvedbe ne prinesejo rezultatov (število možnih pozicij na šahovnici so ocenili na 10^{43} [14]). Taki sistemi pozicije po navadi uredijo v zgoščevalne tabele, največkrat se v ta namen uporablja *Zobrist Hashing* [27].

Problem omejitve iskanja enakih pozicij rešuje šahovski poizvedovalni jezik (CQL, ang. *Chess Query Language*). CQL omogoča iskanje približnih pozicij, s pomočjo wild-card poizvedb [7]. Ta pristop ima tri slabosti:

- Uporabnik mora podati pravilno zapisati poizvedbo s pomočjo kompleksnega poizvedovalnega jezika (poizvedba *"(position [RQ]b2 bg8)"* poišče pozicije z belo trdnjavo ali belo kraljico na polju *b2* in črnim tekačem na polju *g8*).
- Implementacija CQL porabi veliko časa za poizvedbe tudi pri srednje velikih bazah.
- Rezultate ovrednoti samo z *boolean* vrednostjo, torej pove samo, ali je pozicija podobna ali ne. Ne omogoča sistema razvrščanja pozicij po podobnosti.

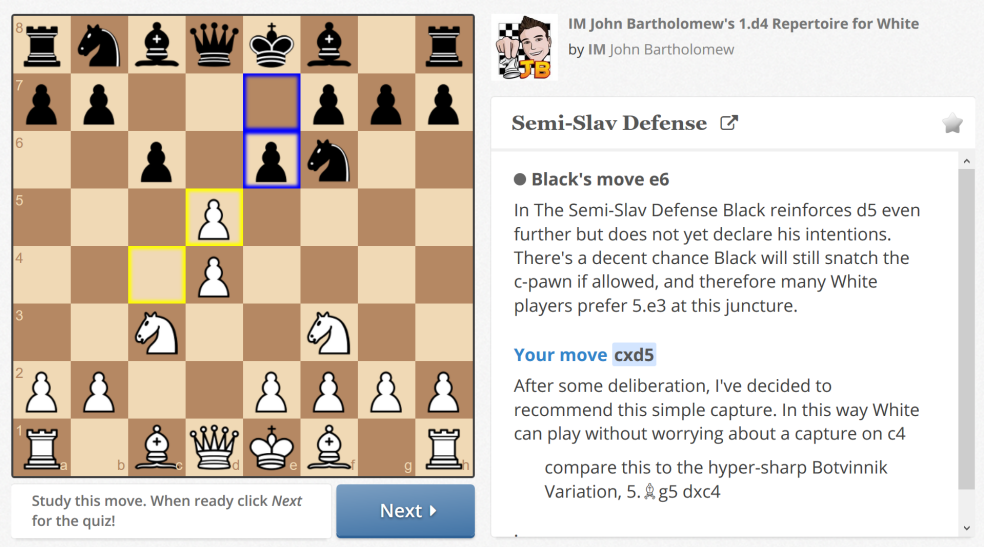
O tej tematiki razlaga članek *Retrieval of Similar Chess Positions* [10]. Govori o problemih prepoznavanja šahovskih pozicij, ki so podobne tistim iz

že znanih zbirk arhiviranih šahovskih partij, s pomočjo metod in algoritmov s področja iskanja in izbiranja informacij (IR, ang. *information retrieval*). Prednost IR je, da omogoča uporabo standardne invertirane organizacije shranjenih šahovskih pozicij, kar omogoča učinkovito iskanje informacij. Podobnost med dvema šahovnicama definiramo na način, da vsako stanje najprej predstavimo v tekstovni obliki. Ta tekstovna predstavitev je oblikovana tako, da predstavi pozicijo, dostopnost in povezljivost med šahovskimi figurami. Taka predstavitev pozicije nam omogoča, da zgradimo iskalni sistem, ki temelji na ključnih besedah z uporabo avtomatsko generiranih stavkov. V diplomski nalogi smo se odločili pristope iskanja, opisane v tem članku, nadgraditi z uporabo dodatnega domenskega znanja prepoznavanja kmečkih struktur.

1.4 Samodejno pripravljeni treningi

Danes najdemo veliko spletnih aplikacij (npr. *chess.com* [6], *chessable.com* [5], primer na sliki 1.7), ki ponujajo šahovske treninge (npr. taktične treninge, ang. *tactics training*). Večina teh je pripravljena s strani šahovskih trenerjev oziroma šahovskih ekspertov. S samodejno pripravo treningov bi razbremenili šahovske trenerje, hkrati pa bi lahko personalizacijo treningov neizmerno pohitrili in jo dali na razpolago širši množici šahistov.

Iskanje vsebinsko podobnih pozicij je dobra osnova za samodejno pripravo treningov. Pri tem uporabljeni koncepti so lahko uporabljeni pri razširitvi za iskanje podobnih taktičnih problemov v vsebinsko podobnih pozicijah. S tem bi bila omogočena tudi personalizacija treningov (npr. generiranje taktičnih treningov iz lastnih partij), vendar so pri tem pomembni še drugi vidiki, kot je naprimer ugotavljanje težavnosti šahovskih problemov [24].



Slika 1.7: Šahovski trening z aplikacijo *chessable.com*.

1.5 Pregled diplomske naloge

Glavni prispevki diplomske naloge so predstavljeni v poglavju 3.

- Poglavje 2 predstavlja metode in orodja, uporabljene v diplomski nalogi. Prvi del poglavja vsebuje predstavitev zapisa šahovskih partij. Drugi del opisuje način predstavitve in obdelave šahovskih pozicij v računalniških programih. Tretji del pa opisuje orodja, uporabljena pri implementaciji programa.
- Poglavje 3 opiše implementacijo in delovanje našega programa.
- Poglavje 4 predstavi rezultate in metode evalvacije našega programa.
- Poglavje 5 zaključi s pregledom opravljenega dela in končnimi ugotovitvami. Predlaga tudi možne izboljšave in ideje za prihodnje delo.

Poglavje 2

Metode

2.1 Zapisovanje šahovskih partij

2.1.1 Algebraična notacija

Algebraična notacija (SAN, ang. *Standard Algebraic Notation*)[23] je danes najpogosteje uporabljen zapis premikov šahovskih figur. Priznan je tudi s strani Svetovne šahovske federacije (FIDE, ang. *World Chess Federation*) [9]. Njegov glavni namen je predstavitev premikov figur na človeku prijazen način. Za zapis figur se uporabljajo črke, predstavljene v tabeli 2.1. Kmetov pri anotiranju potez na zapisujemo.

Ime figure	Angleška oznaka	Slovenska oznaka
kralj	K	K
dama	Q	D
trdnjava	R	T
lovec	B	L
skakač	N	S
kmet	P	P

Tabela 2.1: Oznake figur.

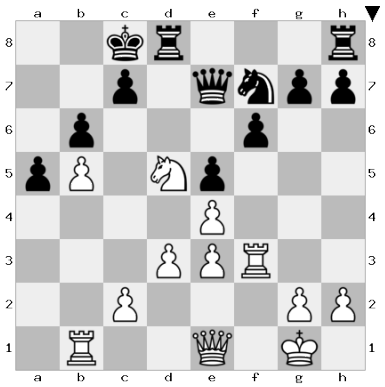
Za zapis polj na šahovnici uporabljamo par, črke za linije in številke za vrste.

Vsaka linija je definirana s črkami od *a* do *h*, vsaka vrsta pa s številkami od *1* do *8*. Pri zapisu potez se porablja kombinacija figure in ciljnega polja. Poleg klasičnih potez moramo znati zapisati tudi posebne poteze, kot so: jemanje, rokade in promocije. Pri jemanju se med figuro in ciljno polje vrine črka *x*. Malo rokado označimo z *0-0*, veliko pa z *0-0-0*. V primeru promocij moramo zapisati, v katero figuro promoviramo kmeta, kar naredimo z enačajem in oznako figure. V tabeli 2.2 je prikazanih nekaj primerov.

Algebraični zapis	Opis premika
Nf3	skakač na *f3*
e4	kmet na *e4*
Nxd5	skakač na *d5* z jemanjem
0-0	mala rokada
e8=Q	kmet na *e8* s promocijo v damo

Tabela 2.2: Primeri zapisa premikov.

2.1.2 Zapis pozicij



Slika 2.1: Primer šahovske pozicije.

Za zapisovanje pozicije se najpogosteje uporablja Forsyth-Edwardsov zapis (FEN, ang. *Forsyth-Edwards Notation*) [8]. Pozicijo s slike 2.1 zapišemo na sledeči način:

```
2kr3r/2p1qnpp/1p3p2/pP1Np3/4P3/3PPR2/2P3PP/1R2Q1K1 b - - 1 17
```

Zapis vsebuje 6 polj ločenih s presledki:

- Postavitev figur na šahovnici. Zapis se začne z osmo vrsto in skrajno levim poljem, oziroma poljem z oznako *a8*. Po vrsti so označene figure s črkami. V primeru, da je figura bele barve, se zapiše veliko tiskano črko, če pa je figura črne barve, jo zapišemo z malo tiskano črko. Kadar na polju ni figure, je zapisano število praznih polj do naslednje figure v vrsti. Vrste na šahovnici ločimo z znakom /.
- Barva figur igralca na potezi. Z w je označen beli, z b pa črni igralec.
- Možnost rokade. S črkami K,Q,k,q se opiše, katere rokade so igralcema še na voljo (npr. Q pomeni, da beli igralec še vedno izpolnjuje pogoje za veliko rokado). V primeru, da rokad ni več na voljo, se zapiše znak -.
- Premik "en passant". V primeru, da je premik v tej potezi mogoč, se zapiše polje z algebraičnim zapisom. Če pa poteza ni možna, se zapiše znak -.
- Število polpotez od zadnjega premika kmeta ali jemanja.
- Števec polnih potez. Začne se z ena in se poveča ob vsaki potezi črnega igralca.

2.1.3 Zapis partij

Za zapis celotne partije je najpogosteje uporabljen prenosljivi zapis igre (PGN, ang. *Portable Game Notation*) [19]. Njegova struktura je človeku prijazna. Poleg tega zapis PGN podpira veliko šahovskih programov.

PGN je razdeljen na opis in poteze. Opis je sestavljen z oznakami, ki se začnejo z oglatim oklepajem, sledi ime oznake, vrednost oznake v navednicah, konča pa se z oglatim zaklepajem. Standard zahteva, da je v opis vključenih

7 nujnih oznak. Sledijo zapisi polnih potez, ki se zapišejo s številko poteze in piko ter polpotezo belega in črnega igralca v algebraičnem zapisu. Primer:

```
[Event "ime turnirja"]
[Site "mesto, država (S tremi črkami, npr. ZDA)"]
[Date "datum v obliki LLLL.DD.MM"]
[Round "zaporedno kolo"]
[White "priimek, ime"]
[Black "priimek, ime"]
[Result "1-0 (rezultat partije)"]

1. e4 e5 2. Bc4 Nc6 3. Qh5 Nf6 4. Qxf7#
```

2.2 Bitne šahovnice

Za predstavitev šahovnice v programih se pogosto uporabljajo bitne šahovnice (ang. *bitboards*) [2]. Te so specializirana varianta podatkovne strukture bitno polje (ang. *bit array*). Uporaba bitnih šahovnic nam omogoča optimizacijo hitrosti in/ali porabe prostora pri velikih izračunih v algoritmih programa. Biti na bitni šahovnici so medsebojno povezani s pomočjo pravil igre in pogosto tvorijo informacije o poziciji na šahovnici. Lahko pa jih uporabimo tudi kot maske za transformacije ali poizvedovanje o poziciji [12].

Bitna šahovnica je torej podatkovna struktura, ki jo lahko implementiramo s 64-bitnim številom (64 bitov predstavlja 64 polj na šahovnici). V programskih jezikih so taka števila navadno označena s tipom *long*. Vsak bit takega števila si lahko predstavljamo kot binarno informacijo o nekem polju. Za primer si oglejmo, kako predstaviti vsa črna polja na šahovnici:

```
0101010110101010010101011010101001010101101010100101010110101010
```

Lažjo predstavitev nam omogoči tabela 2.3, kjer smo ničle zamenjali s pikami.

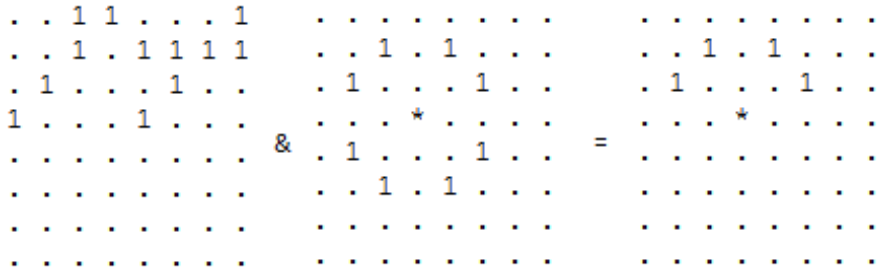
.	1	.	1	.	1	.	1
1	.	1	.	1	.	1	.
.	1	.	1	.	1	.	1
1	.	1	.	1	.	1	.
.	1	.	1	.	1	.	1
1	.	1	.	1	.	1	.
.	1	.	1	.	1	.	1
1	.	1	.	1	.	1	.

Tabela 2.3: Črna polja, predstavljena z bitno šahovnico.

Predstavitev bitnih šahovnic, kot niz logičnih vrednosti, pa nam omogoča izvajanje logičnih operacij, kot so IN, ALI, NEGACIJA itd. Logične operacije so zelo hitre, saj se v procesorju izvedejo nad vsemi istoležnimi elementi hkrati. Kot primer si oglejmo logični IN (v programskih jezikih po navadi označen z &):

$$010 \wedge 111 = 010 \tag{2.1}$$

Primer te operacije nad bitnimi šahovnicami pa lahko vidimo na sliki 2.2.



Slika 2.2: Polja s črnimi figurami & napadi skakača = napadena polja.

2.3 Orodja in programske knjižnice

2.3.1 Knjižnica python-chess

Pri prepoznavanju kmečkih struktur smo uporabili knjižnico *python-chess* [21]. Ta podpira obdelovanje vseh formatov omenjenih, v poglavju 2.1. Funkcije podprte v razredih knjižnice, so implementirane s pomočjo bitnih šahovnic. Zaradi tega je izvajanje programov, napisanih s pomočjo te knjižnice, hitro in prostorsko učinkovito. Ponujenih pa je tudi veliko funkcij, ki podpirajo pogosto uporabljene poizvedbe, oziroma transformacije, za delo s šahovskimi pozicijami in partijami. Za izračune najboljših potez se lahko knjižnica poveže tudi s šahovskimi motorji (npr. *Stockfish 8*). Podprto je tudi grafično prikazovanje šahovnic, oziroma posameznih pozicij.

Naštejmo nekaj nam pomembnih razredov:

- ***Piece*** – je razred, ki predstavlja šahovsko figuro. Opisana je z dvema spremenljivkama; tipom figure (*piece_type*) in barvo figure (*color*). Podprta je tudi funkcija *symbol()*, ki nam vrne črkovni zapis za figuro (angleška oznaka iz tabele 2.1).

- ***Board*** – je razred, ki predstavlja šahovnico s figurami. Za nas so zanimive predvsem naslednje funkcije:

  - *set_fen(fen)* – šahovnici priredi pozicijo na podlagi vnešenega zapisa FEN.

  - *pieces(piece_type, color)* – vrne objekt tipa *SquareSet*, ki predstavlja množico figur želenega tipa in barve.

  - *piece_at(square)* – je funkcija, ki sprejme zaporedno številko polja (v tej knjižnici so polja oštevilčena od *0* do *63*, *0* predstavlja polje *a1*, *63* pa polje *h8*). Rezultat funkcije je objekt tipa *Piece*, ki predstavlja figuro na želenem polju.

  - *attacks(square)* – vrne množico polj, ki jih figura na podanem polju (*square*) napada. Če na tem polju ni figure, vrne prazno

množico.

- ◇ *attackers(color, square)* – za podano polje (*square*) vrne napadalce (oz. množico polj, na katerih se nahajajo), katerih barva je podana s spremenljivko *color*.
- ◇ *set_piece_at(sqaure, piece)* – podano figuro (*piece*) postavi na polje, opisano s spremenljivko *square*. Če se na tem polju že nahaja figura, jo funkcija zamenja.

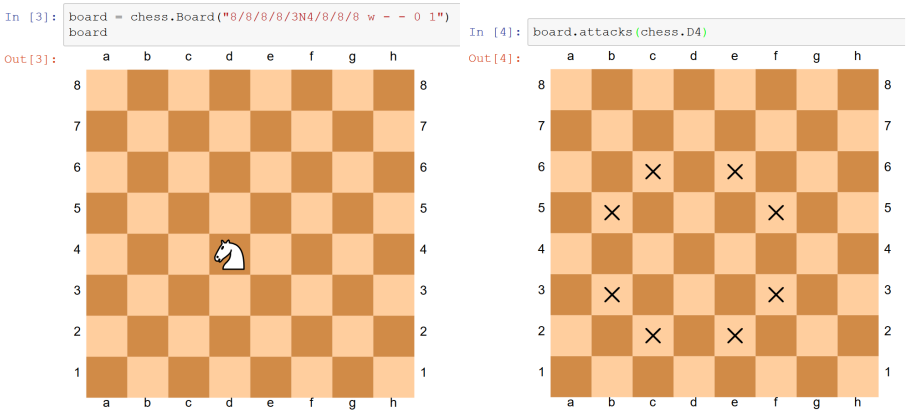
- ***SquareSet*** - je razred, ki predstavlja bitno šahovnico. Posebnost tega razreda je v tem, da nam omogoča predstavitev bitne šahovnice kot množico polj (oz. seznam polj). Tako lahko v bitno šahovnico dodajamo in odstranjujemo posamezna polja s funkcijama *add(sqaure)* in *remove(square)*. Nad objekti tipa *SquareSet* lahko izvajamo tudi bitne operacije. Te so podprte tudi s funkcijami na sliki 2.3.

```
def union(self, other):
    return self | other

def intersection(self, other):
    return self & other

def difference(self, other):
    return self & ~other

def symmetric_difference(self, other):
    return self ^ other

def update(self, other):
    self |= other

def intersection_update(self, other):
    self &= other

def difference_update(self, other):
    self &= ~other

def symmetric_difference_update(self, other):
    self ^= other
```

Slika 2.3: Funkcije bitnih operacij v razredu *SquareSet*.

2.3.2 Jupyter Notebook

Jupyter Notebook [13] je odprtokodna spletna aplikacija, ki omogoča ustvarjanje in deljenje dokumentov. Ti lahko vsebujejo tudi programsko kodo, ki jo lahko zaganjamo kar znotraj aplikacije. Podpira več kot 40 programskih jezikov. V sklopu diplomske naloge smo ga uporabili v povezavi s programskim jezikom *Python*. Aplikacija dobro sodeluje tudi s prej opisano knjižnico *python-chess*, saj omogoča zelo lahko in pregledno prikazovanje šahovskih pozicij in bitnih šahovnic. Primer grafičnega prikaza najdemo na sliki 2.4.



Slika 2.4: Grafični prikaz v *Jupyter Notebooku*. Na levi prikaz šahovske pozicije, na desni pa prikaz bitne šahovnice napadov skakača.

2.3.3 Knjižnica Apache Lucene

Za iskanje podobnih pozicij smo uporabili visokozmogljivo in odprtokodno knjižnico za iskanje po besedilu *Apache Lucene Core* [16]. Knjižnica je v celoti napisana v programskem jeziku *Java*, zato deluje na različnih platformah. Primerna je skoraj za vsako aplikacijo, ki potrebuje iskanje po tekstovnih datotekah.

Ponuja skalabilno in hitro indeksiranje podatkov pri zelo majhnih zahtevah po kapaciteti delovnega spomina (kopica velikosti 1 MB). Omogoča veliko različnih poizvedb, kot so: poizvedbe s frazami (ang. *phrase queries*),

poizvedbe po obsegu (ang. *range queries*) itd. Rezultate poizvedb je mogoče razvrščati z različnimi rangirnimi modeli.

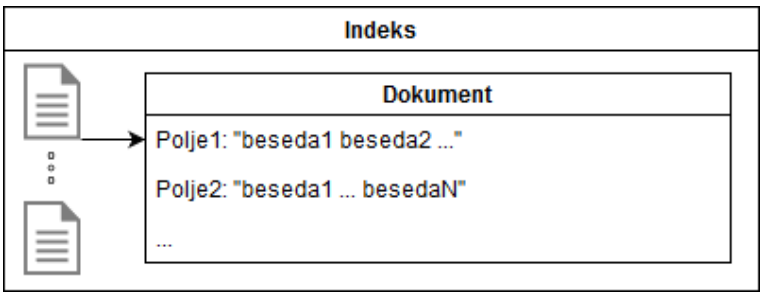
Lucene indeks

Na tem mestu si bomo ogledali osnovne koncepte knjižnice. Ti nam bodo kasneje služili pri lažjem razumevanju in predstavi sestave naše aplikacije.

Glavna enota knjižnice je indeks (ang. *index*), ki je sestavljen iz množice dokumentov (ang. *documents*).

- dokument je sestavljen iz zaporedja polj (ang. *fields*);
- polje je poimenovano zaporedje izrazov (oz. besed, ang. *terms*);
- izraz je zaporedje bajtov;

Enaki sekvenci bajtov v dveh različnih poljih sta klasificirani kot različna izraza. Torej je izraz enolično določen s parom; imenom polja in zaporedjem bajtov.



Slika 2.5: Diagram strukture *Lucene* indeksa.

Rangirna funkcija Okapi BM25

V diplomski nalogi smo uporabili rangirno funkcijo *Okapi BM25* [3]. To je funkcija, ki razvršča množico dokumentov (oz. besedil) na podlagi pojavitve poizvedbenih izrazov (oz. besed) v vsakem besedilu, ne glede na njihovo pozicijo v dokumentu.

Če imamo poizvedbo Q, ki vsebuje izraze $q_1, ..., q_n$, je BM25 ocena za dokument D, izražena z enačbo:

$$score(D, Q) = \sum_{1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \tag{2.2}$$

kjer je $f(q_i, D)$ frekvenca pojavitve izraza q_i v dokumentu D; $\mid D \mid$ je dolžina dokumenta D (število besed), *avgdl* pa je povprečna dolžina dokumenta iz celotne zbirke dokumentov (ang. *collection*); k_1 in b sta konstanti, kjer $k_1 = 1.2$ in $b = 0.75$; $IDF(q_i)$ pa izračunamo po enačbi:

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5)}{n(q_i) + 0.5} \tag{2.3}$$

kjer je N število dokumentov v zbirki, $n(q_i)$ pa je število dokumentov, ki vsebujejo izraz q_i.

Analizator besedil

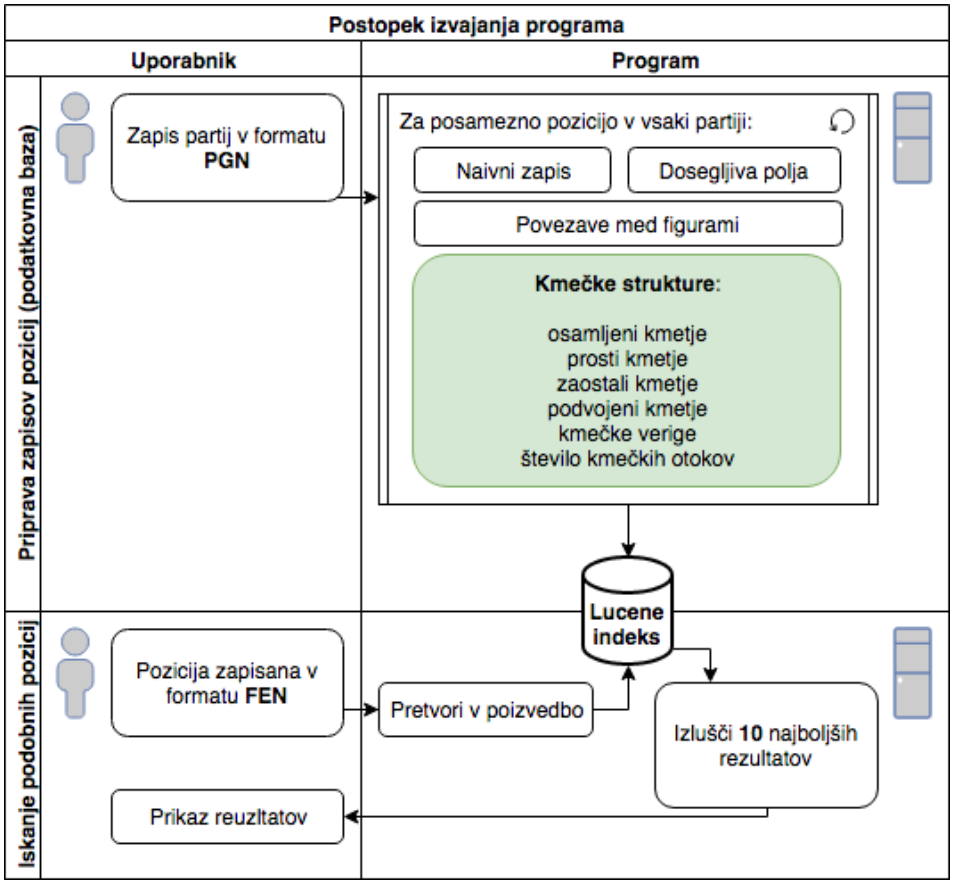
Za potrebe iskanja podobnih šahovskih pozicij smo morali prilagoditi standardni analizator besedila (ang. *Standard Analyzer*), ki ga ponuja knjižnica. Odvzeli smo mu komponento, ki vse velike tiskane črke spremeni v male tiskane črke. S tem smo dosegli, da analizator loči med velikimi in malimi tiskanimi črkami, kar nam omogoča, da ločimo med črnimi in belimi figurami v poziciji.

Poglavje 3

Opis programa

Program je napisan v programskem jeziku *Python* in programskem jeziku *Java*. Za kombinacijo dveh programskih jezikov smo se odločili zaradi knjižnic *python-chess* in *Apache Lucene Core*. Izvaja se ga lahko v Linux ali Windows okolju. Delovanje implementiranega programa lahko razdelimo na dve enoti (slika 3.1). Prva je priprava zapisov pozicij, druga pa iskanje podobnih pozicij po bazi podatkov. Poganjanje programa je omogočeno s pomočjo odprtokodne spletne aplikacije *Jupyter Notebook*, opisane v poglavju 2.3.2. Iskanje podobnih pozicij je implementirano s pomočjo knjižnice za iskanje po besedilih *Apache Lucene Core*, opisane v poglavju 2.3.3.

Za zgled smo uporabili pristope, omenjene v članku *Retrieval of Similar Chess Positions* [10]. Na diagramu (slika 3.1) je prikazan postopek izvajanja programa. Glavni prispevek naše diplomske naloge je komponenta za samodejno prepoznavanje kmečkih struktur (na sliki 3.1 označena z zeleno), katere cilj je izboljšati postopek iskanja podobnih šahovskih pozicij, ki je opisan v prej omenjenem članku [10].

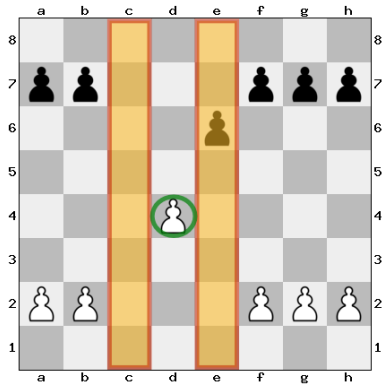


Slika 3.1: Diagram izvajanja programa.

3.1 Samodejno prepoznavanje kmečkih struktur

V tem poglavju si bomo ogledali dve od šestih implementacij prepoznavanja kmečkih struktur. Prva je prepoznavanje osamljenega kmeta, druga pa prepoznavanje kmečkih verig. Za ti dve smo se odločili zato, ker prva prikaže elegantno uporabo bitnih šahovnic, druga pa je primer prepoznavanja kmečke strukture z več kmeti, kjer smo uporabili rekurzijo.

3.1.1 Osamljeni kmet



Slika 3.2: Primer zapisa definicije osamljenega kmeta z bitno šahovnico.

Definicija osamljenega kmeta pravi, da na njegovih sosednjih linijah ne najdemo niti enega kmeta iste barve (podpoglavje 1.1). Na podlagi te definicije smo implementirali funkcijo za prepoznavanje osamljenega kmeta (algoritem 1).

Algoritem 1 Funkcija za prepoznavanje osamljenega kmeta

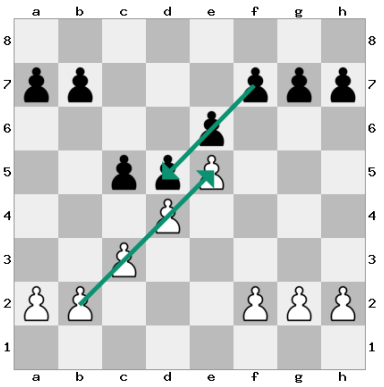
Require: kmet se nahaja na *polju* na *šahovnici*

```
1:  function OSAMLJENIKMET(šahovnica, polje)
2:      maskaOsamljenegaKmeta ← sosednjaStolpca(polje) ▷ glej sliko 3.2
3:      barvaKmeta ← šahovnica.figura(polje).barva
4:      kmeti ← šahovnica.kmeti(barvaKmeta)
5:      if maskaOsamljenegaKmeta & kmeti = 0 then
6:          return True
7:      else
8:          return False
9:      end if
10: end function
```

Funkcija kot vhodne parametre sprejme dve spremenljivki. Prva je objekt *šahovnica*, za implementacijo tega objekta poskrbi knjižnica *python-chess* (razred *Board*). Spremenljivka *polje* pa predstavlja polje, na katerem se nahaja kmet, za katerega želimo preveriti ali je osamljen ali ne. Spremenljivki *maskaOsamljenegaKmeta* in *kmeti* sta bitni šahovnici.

3.1.2 Kmečke verige

Kmečka veriga je serija treh ali več povezanih kmetov na diagonali (podpoglavje 1.1). Za prepoznavanje te kmečke strukture smo implementirali algoritem 2. Vhodna parametra funkcije *kmečkeVerige* sta objekt *šahovnica* in spremenljivka *barva*, ki pove, katero barvo kmečkih verig naj funkcija poišče. Za vsakega kmeta izbrane barve s pomočjo zanke poiščemo levo ali desno kmečko verigo. Pri tem si pomagamo z rekurzivnima funkcijama *najdiDesnoVerigo* (algoritem 3) in *najdiLevoVerigo*. Na koncu v rezultatu obdržimo samo najdaljše verige, za kar poskrbi funkcija *očistiVerige*.



Slika 3.3: Primer kmečkih verig.

Algoritem 2 Algoritem za prepoznavanje kmečkih verig

```
1: function kmečkeVerige(šahovnica,barva)
2:     kmeti ← šahovnica.kmeti(barva)
3:     verige[]
4:     for polje in kmeti do
5:         veriga ← praznaBitnaŠahovnica
6:         veriga ← najdiDesnoVerigo(šahovnica,polje,kmeti,veriga)
7:         if veriga.velikost > 2 then
8:             verige.dodaj(veriga)
9:         end if
10:        veriga ← praznaBitnaŠahovnica
11:        veriga ← najdiLevoVerigo(šahovnica,polje,kmeti,veriga)
12:        if veriga.velikost > 2 then
13:            verige.dodaj(veriga)
14:        end if
15:    end for
16:    return očistiVerige(verige)
17: end function
```

Algoritem 3 Funkcija *najdiDesnoVerigo*

```
1: function najdiDesnoVerigo(šahovnica,polje,kmeti,veriga)
2:     if polje not in kmeti then
3:         return veriga
4:     end if
5:     veriga.dodaj(polje)
6:     napadi ← šahovnica.napadi(polje)
7:     if polje in desniRob then
8:         return veriga
9:     end if
10:    return najdiDesnoVerigo(šahovnica,desniNapad,kmeti,veriga)
11: end function
```

3.2 Priprava indeksa za iskanje podobnih pozicij

Preden smo se lotili iskanja podobnih pozicij, smo si morali priskrbeti veliko bazo šahovskih partij. Izbrali smo partije šahovskega velemojstra Magnusa Carlsena. Zbirka vsebuje preko 2700 partij, zapisanih v PGN zapisu [17]. Uporabili smo pozicije od trinajste polne poteze naprej. Tako se izognemo delu, ko so si pozicije različnih partij zelo podobne. Temu segmentu igre pravimo otvoritev (ang. *opening*).

Vsako pozicijo smo zapisali z izrazi, po zgledu iz članka *Retrieval of Similar Chess Positions* [10]. Ta zapis lahko razdelimo na štiri dele:

- **Naivni zapis** – Opisuje točno pozicijo figur (npr. *Rb1*, kar opisuje belo trdnjavo na polju *b1*).

- **Dosegljiva polja** – Za vsako figuro zapišemo, katera polja so dosegljiva v eni potezi. Vsako dosegljivo polje obtežimo z verjetnostjo pojavitve figure na tem polju, oziroma oddaljenostjo od originalne pozicije figure (npr. *Ra1|0.89*, kjer je uporaba znaka | pogojena z uporabo knjižnice *Apache Lucene Core*). Za izračun uteži uporabimo formulo:

$$w(p, x, y, x', y') = 1 - \frac{7 \times d((x, y), (x', y'))}{64} \tag{3.1}$$

  kjer p predstavlja pozicijo na šahovnici. Originalno pozicijo figure predstavljata koordinati x in y. Koordinati dosegljive pozicije označujeta *x'* in *y'*. Označba *d((x,y)(x',y'))* pa predstavlja Chebyshevo razdaljo [4] (minimalno število potez kralja med dvema poljema) med originalno in dosegljivo pozicijo figure.

- **Povezave med figurami** – Zapisali smo napade in branjenja med figurami. Primer napada, kjer bela kraljica napada črnega kmeta na polju *a5*, zapišemo s *Q>pa5*. Primer branjenja, kjer bela kraljica brani belo trdnjavo na polju *b1*, pa zapišemo s *Q<Rb1*.

- **Kmečke strukture** – Z izrazi smo zapisali še kmečke strukture. Primere zapisov najdemo v tabeli 3.1, postopki prepoznavanja kmečkih struktur pa so opisani v poglavju 3.1.

Kmečka struktura	Opis
Ie3	beli **osamljeni kmet** na polju e3
fa5	črni **prosti kmet** na polju f5
'fa5	črni **branjeni prosti kmet** na polju f5
lc7	črni **zaostali kmet** na polju c7
S{e3-e4}	bela **podvojena kmeta** na poljih e3 in e4
W[c2/d3/e4]	bela **kmečka veriga** s korenom na polju c2 in glavo na polju e4
P(2) p(2)	beli in črni imata vsak po dva **kmečka otoka**

Tabela 3.1: Primeri zapisa kmečkih struktur.

Sledi skrajšan primer zapisa pozicije s slike 2.1.

```
Rb1 Qe1 Kg1 Pc2 Pg2 Ph2 Pd3 Pe3 Rf3 Pe4 pa5 ...
Ra1|0.89 Rc1|0.89 Rd1|0.78 Rb2|0.89 Rb3|0.78 ...
Q>pa5 r>Nd5 N>pb6 R>pf6 N>pf6 N>pc7 N>qe7
Q<Rb1 R<Qe1 Q<Kg1 K<Pg2 K<Ph2 P<Pd3 Q<Pe3 ...
lc7
'fa5
S{e3-e4}
W[c2/d3/e4] w[c7/b6/a5] w[g7/f6/e5]
P(2) p(2)
```

S takimi zapisi smo sestavili *Lucene* indeks šahovskih pozicij. V našem indeksu smo vsako pozicijo predstavili z dokumentom, sestavljenem s tremi polji:

- **ID** – polje predstavlja enolični identifikator vsake pozicije, ki je sestavljen iz zaporedne številke partije, naziva dogodka, imen obeh igralcev in zaporedne številke poteze. Primer:

```
831[6th EU-chT]Sutovsky, Emil-Carlsen, Magnus/13 W
```

- **FEN** – polje s predstavitvijo pozicije v formatu FEN. To polje nam služi za lažje prikazovanje podobnih pozicij ob poizvedbah.

- **vsebina** – je polje, ki vsebuje naš tekstovni zapis pozicije. To polje je za nas ključnega pomena, saj ga primerjamo s poizvedbo našega programa.

3.3 Iskanje podobnih šahovskih pozicij

Pri implementaciji iskanja podobnih šahovskih pozicij smo morali paziti predvsem na preprostost uporabe. Zato kot vhodni parameter program sprejme izbrano šahovsko pozicijo zapisano v formatu FEN. Ta zapis pozicije program nato pretvori v zapis pozicije opisan v podpoglavju 3.2, s to razliko, da izpusti del opisa dosegljivih polj. Zapis dosegljivih polj v poizvedbi izpustimo, ker dosegljiva polja predstavljajo alternativo zapisom točnih pozicij figur (dosegljiva polja pa smo že ovrednotili v indeksu, oz. v zapisu pozicij v indeks). Pozorni pa smo morali biti tudi pri upoštevanju rezerviranih znakov v poizvedovalnem jeziku knjižnice *Apache Lucene Core*. Ob pojavitvi oglatih ali zavitih oklepajev smo morali te znake označiti s predhodnim znakom \. Sledi skrajšan primer zapisa pozicije s slike 2.1.

```
Rb1 Qe1 Kg1 Pc2 Pg2 Ph2 Pd3 Pe3 Rf3 Pe4 pa5 ...
Q>pa5 r>Nd5 N>pb6 R>pf6 N>pf6 N>pc7 N>qe7
Q<Rb1 R<Qe1 Q<Kg1 K<Pg2 K<Ph2 P<Pd3 Q<Pe3 ...
lc7
'fa5
S\{e3-e4\}
W\[c2/d3/e4\] w\[c7/b6/a5\] w\[g7/f6/e5\]
P(2) p(2)
```

Kot izhod pa nam program ponudi deset najbolj podobnih pozicij iz desetih različnih partij. Za grafični prikaz pozicij smo poskrbeli z aplikacijo *Jupyter Notebook*.

Poglavje 4

Rezultati

Glavni rezultat diplomske naloge je program za iskanje vsebinsko podobnih šahovskih pozicij s pomočjo samodejnega prepoznavanja pozicijskih motivov. Evalvacijo delovanja programa lahko razdelimo na dva dela. Prvi je samodejno prepoznavanje kmečkih struktur, drugi pa iskanje podobnih šahovskih pozicij. V ta namen smo izvedli dva eksperimenta, opisana v naslednjih podpoglavjih. Pri obeh nam je pomagal domenski ekspert, FIDE šahovski mojster.

4.1 Prepoznavanje kmečkih struktur

Prepoznavanje kmečkih struktur smo testirali na 104 skrbno izbranih primerih iz knjige *Understanding Pawn Play in Chess* [18]. Pripravili smo tabelo z enoličnimi identifikatorji in FEN zapisi vsake pozicije. To smo kot vhodni parameter podali testni skripti, ki je tabelo dopolnila s stolpci rešitev funkcij za prepoznavanje kmečkih struktur (dodatek A). Tabelo smo dali v vpogled šahovskemu ekspertu, ki je ocenil rešitve našega programa (slika 4.1). Ugotovil je 100 odstotno klasifikacijsko točnost našega algoritma.

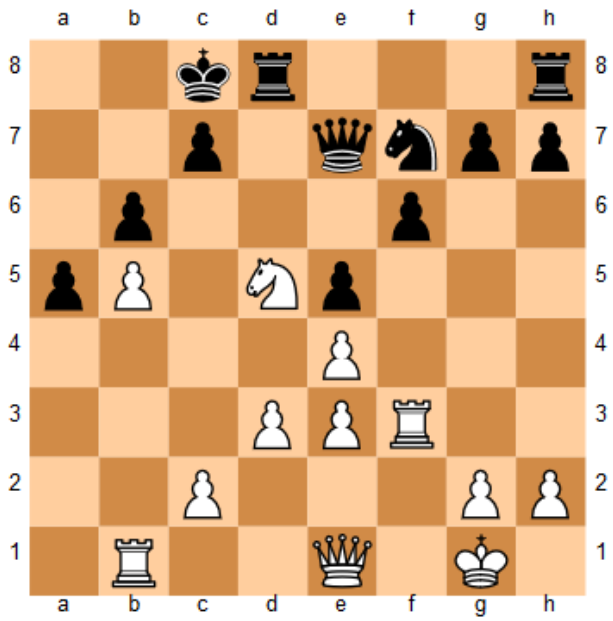
S tem eksperimentom smo ugotovili tudi nekaj o frekvenci pojavljanja posameznih kmečkih struktur. V izbranih 104 primerih smo prepoznali največ osamljenih kmetov, ostale kmečke strukture pa so si med seboj dokaj iz-

enačene (tabela 4.2 in graf na sliki 4.2). Prav tako so si pozicije izenačene tudi v povprečju števila belih in črnih kmečkih otokov (tabela 4.2).

```
#go to the next position
index += 1
if index >= max_index:
    index = 0

#show position
print("Position ID: " + ids.iloc[index])
print("Isolated: " + isolated.iloc[index])
print("Passed: " + passed.iloc[index])
print("Doubled: " + doubled.iloc[index])
print("backwards: " + backward.iloc[index])
print("Chains: " + chains.iloc[index])
print("Islands: " + islands.iloc[index])
board = chess.Board(fens.iloc[index])
board
```

```
Position ID: X_018
Isolated: -
Passed: 'fa5
Doubled: S{e3-e4}
backwards: lc7
Chains: W[c2/d3/e4],w[c7/b6/a5],w[g7/f6/e5]
Islands: P(2),p(2)
```

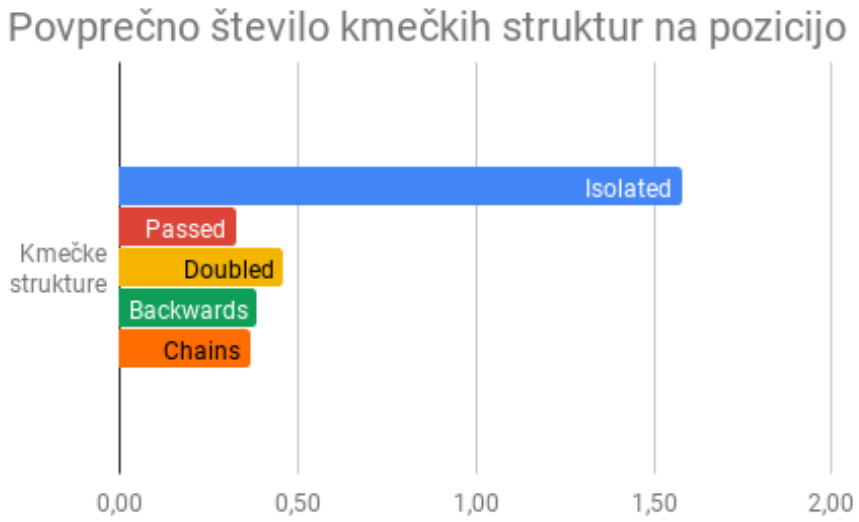
Out[21]:



Slika 4.1: Primer izpisa ene vrstice iz tabele pozicij (dodatek A) v programu, namenjenem pregledu pravilnosti prepoznavanja kmečkih struktur.

	Pozicije	*Isolated*	*Passed*	*Doubled*	*Backwards*	*Chains*
Skupaj	104	164	34	48	40	38
Pozicije s strukturo	104	75	27	39	37	26
Pozicije brez struktur	0	29	77	65	67	78
Povp. št. struktur na pozicjo	3,12	1,58	0,33	0,46	0,38	0,37
Povp. št. struktur na pozicjo s strukturo	3,12	2,19	1,26	1,23	1,08	1,46

Tabela 4.1: Statistična analiza prepoznanih kmečkih struktur v pozicijah iz dodatka A.



Slika 4.2: Graf povprečnega števila prepoznanih kmečkih struktur na pozicijo (dodatek A).

	White Islands	*Black Islands*
Skupaj	245	240
Pozicije s strukturo	104	104
Pozicije brez struktur	0	0
Povp. št. struktur na pozicjo	2,36	2,31

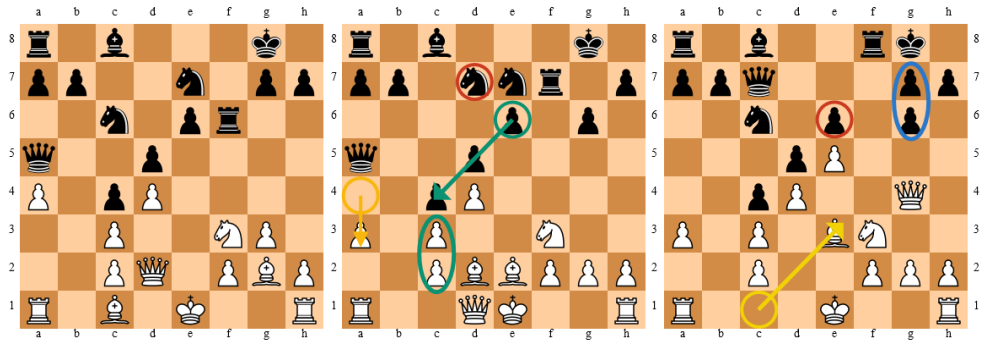
Tabela 4.2: Statistična analiza kmečkih otokov v pozicijah iz dodatka A.

4.2 Podobnost šahovskih pozicij

Izkazalo se je, da je ocenjevanje podobnosti šahovskih pozicij lahko zelo subjektivno. Zato smo tudi tu uporabili znanje šahovskega eksperta, ki je preizkusil naš program na nekaj primerih. Baza podobnih pozicij je bila sestavljena iz partij šahovskega velemojstra Magnusa Carlsena [17]. Delovanje našega programa smo ovrednotili s pomočjo ankete (dodatek B), ki so jo reševali šahisti. V naslednjem podpoglavju si bomo na primeru ogledali delovanje našega programa za iskanje podobnih šahovskih pozicij.

Analiza primera

Na sliki 4.3 najdemo primer vhodne in izhodnih pozicij (iz dveh različnih partij) programa za iskanje vsebinsko podobnih šahovskih pozicij. Na levi je vhodna (originalna) pozicija, na sredini njej najbolj podobna pozicija, na desni druga najbolj podobna pozicija.



Slika 4.3: Originalna pozicija (levo) in dve najbolj podobni poziciji iz partij svetovnega prvaka Magnusa Carlsena (desno).

V maslednji analizi primera se bomo osredotočili predvsem na kmečke strukture. Za lažje razumevanje bomo uporabili poenostavljen sistem točkovanja podobnosti, kot smo ga uporabili v programu (*BM25* opisan v poglavju 2.3.3). Opišimo in primerjajmo tri pozicije na sliki 4.3:

- **Originalna pozicija** – Na šahovnici najdemo lep nabor kmečkih struktur. Opazimo lahko: belega osamljenega kmeta na polju *a4*, podvojena bela kmeta na poljih *c2* in *c3*, črno kmečko verigo na poljih *e6* (koren), *d5* in *c4* (glava), katere del je tudi črni zaostali kmet na polju *e6*. Beli ima tri kmečke otoke, črni pa dva. Te kmečke lahko zapišemo na sledeč način:

```
Ia4 1e6 S{c2-c3} w[e6/d5/c4] P(3) p(2)
```

- **Najbolj podobna pozicija** – Na šahovnici lahko opazimo bela podvojena kmeta na identični lokaciji kot v originalni poziciji. Ta struktura nam k oceni podobnosti prinese tri točke, prvi dve prispevata točni lokaciji obeh kmetov na poljih *c2* in *c4*, tretjo pa prispeva opis strukture podvojenih kmetov. Prav tako najdemo identično črno kmečko verigo kot v originalni poziciji, ta prinese v seštevek kar pet točk (tri točke prispeva točna lokacija kmetov, vsak svojo točko pa prineseta opis verige in opis zaostalega kmeta na polju *e6*). Na šahovnici lahko najdemo tudi osamljenega kmeta na polju *a3*. Slednji k podobnosti prispeva le *0,89* točke, saj se v zapisu te pozicije nahaja beseda *Pa4|0.89*. Če bi se ta kmet nahajal na originalnem polju *a4*, bi v seštevek prispeval dve točki (ena točka za točno pozicijo in ena točka za opis kmečke strukture). Število kmečkih otokov na obeh straneh je enako številu otokov v originalni poziciji, kar doda dve točki k oceni podobnosti. Primer figure, ki nima prispevka k oceni podobnosti, je skakač na polju *d7* (se ne nahaja na originalnem polju, na dosegljivih poljih ne najdemo skakača v originalni poziciji, prav tako pa si ne deli obramb in napadov z nobenim od skakačev v originalni poziciji). Omenjene kmečke strukture zapišemo na sledeč način:

```
Pa4|0.89 Ia3 1e6 S{c2-c3} w[e6/d5/c4] P(3) p(2)
```

- **Druga najbolj podobna pozicija** – V tej poziciji črni kmet na polju *e6* ni več zaostal, to je zaradi belega kmeta na polju *e5*, ki tudi povezuje dva kmečka otoka iz originalne pozicije, s tem zmanjša število

belih kmečkih otokov na dva in se tako še dodatno razlikuje od originalne pozicije. Ta kmet tvori tudi novo nastalo balo kmečko verigo na poljih c3, d4 in e5. Opis te verige nima vpliva na oceno podobnosti. Prav tako vpliva na oceno nima opis podvojenih črnih kmetov na poljih *g6* in *g7*. K dodatnemu zmanjšanju ocene pa pripomore še večja oddaljenost belega lovca na črnih poljih od originalne pozicije. Sledi zapis omenjenih kmečkih struktur in dosegljivega polja belega tekača.

```
Bc1|0.78 Ia3 S{c2-c3} s{g6-g7} W[c3/d4/e5]
w[e6/d5/c4] P(2) p(2)
```

V tabeli 4.3 najdemo BM25 oceni podobnosti za poziciji na sliki 4.3. Ti dve vrednosti sta nadpovprečni za naš algoritem, v povprečju najboljše ocene dosegajo ocene med 135 in 145. Poudariti moramo, da BM25 ocene niso primerne za primerjavo ocen med različnimi poizvedbami [3].

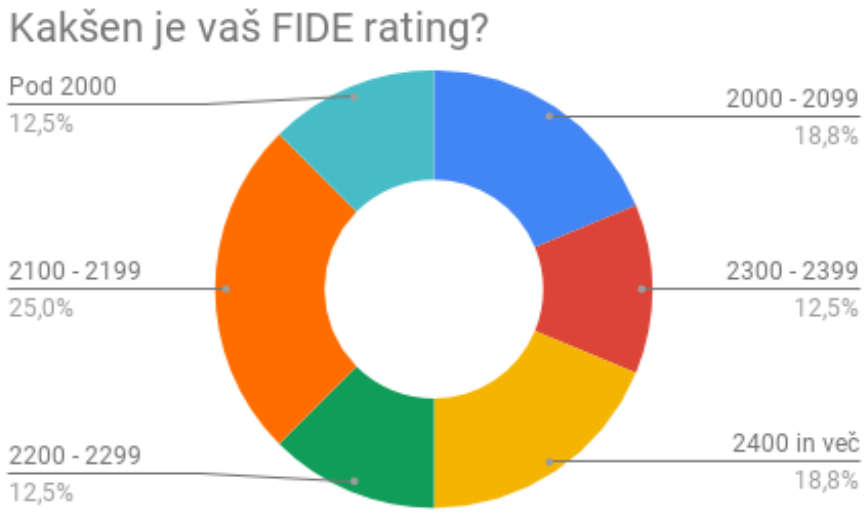
	BM25 ocena algoritma
Najbolj podobna pozicija	179.12532
Druga najbolj podobna pozicija	173.80762

Tabela 4.3: Ocene podobnosti našega algoritma za poziciji na sliki 4.3.

Analiza ankete

Preden se bralec loti branja analize ankete, se lahko sam preskusi v reševanju le-te v dodatku B (anketa je napisana v angleščini). Pripravili smo anketo, ki je obsegala 7 vprašanj. Anketo je rešilo 18 šahistov, dva od teh nista podala rešitev v pravilnem formatu (namesto zaporedja črk A, B, C in D sta podala samo eno črko), tako da smo ju iz analize izločili.

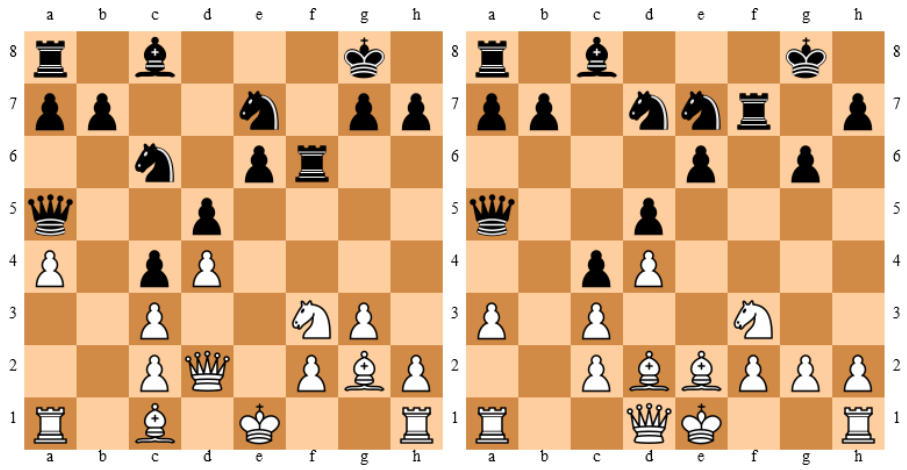
V prvem vprašanju smo anketiranca vprašali, kakšen je njegov FIDE rating (porazdelitev je prikazana z diagramom na sliki 4.4).



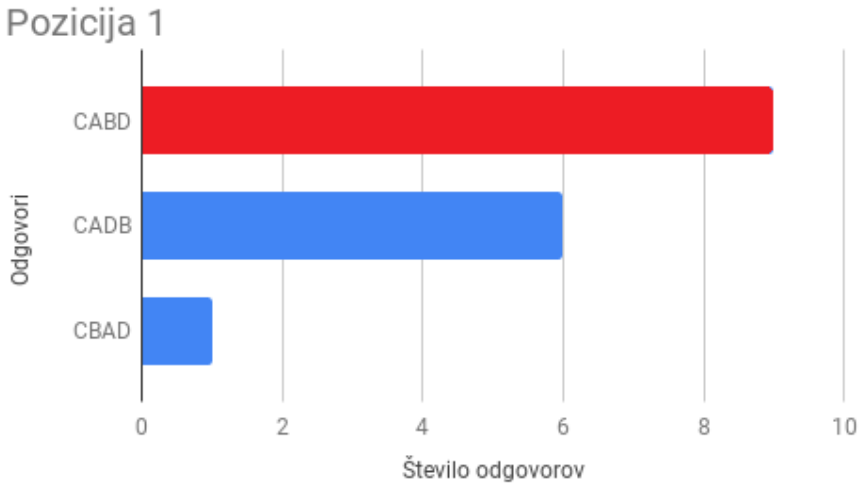
Slika 4.4: Tortni diagram porazdelitve FIDE ratinga anketirancev.

Naslednjih pet vprašanj se nanaša na oceno podobnosti pozicij. Vsako vprašanje je sestavljeno iz originalne (vhodne) pozicije in štirih podobnih pozicij (označenih s črkami A, B, C in D), ki nam jih je za vhodno pozicijo vrnil naš program (vse štiri pozicije so bile med najboljšimi desetimi v rezultatih programa). Za vhodne pozicije smo izbrali pozicije z velikim številom kmečkih struktur. Naloga anketiranca je bila da te pozicije razvrsti po podobnosti glede na originalno pozicijo (npr. $CBAD$). Sledijo si slike z originalno in najbolj podobno pozicijo glede na izhod našega programa, za

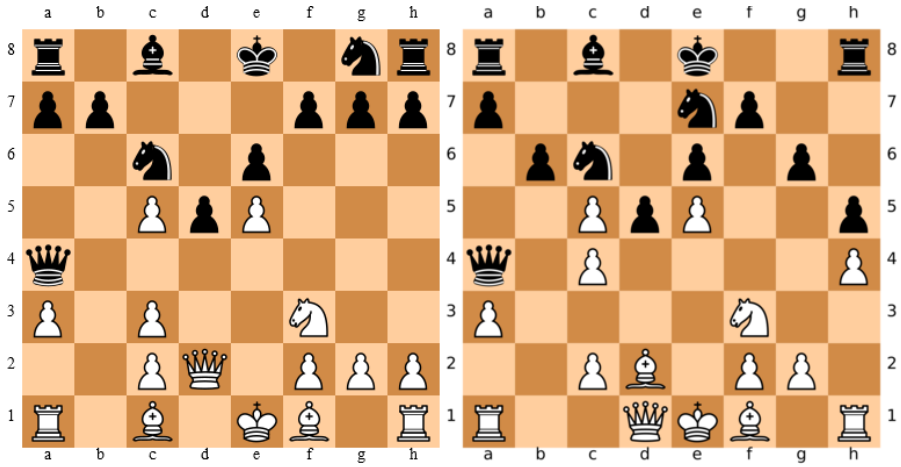
vsak primer pa še histogrami z odgovori šahistov.



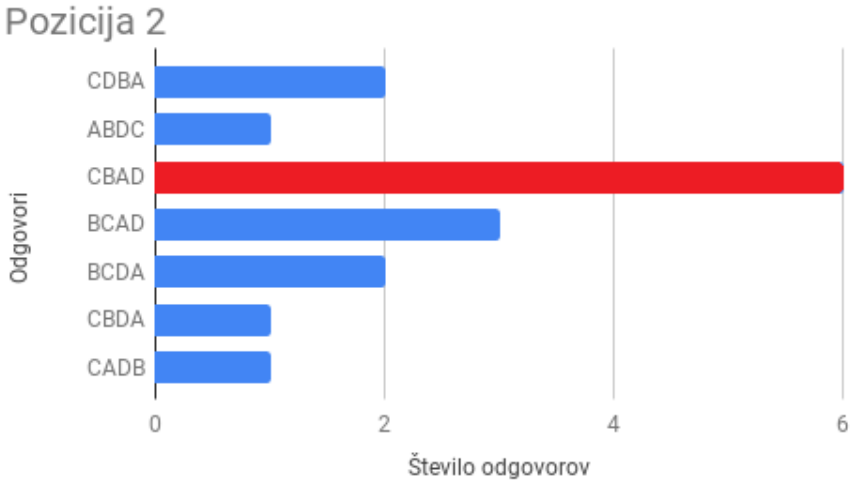
Slika 4.5: Prvi primer. Levo originalna pozicija, desno njej najbolj podobna.



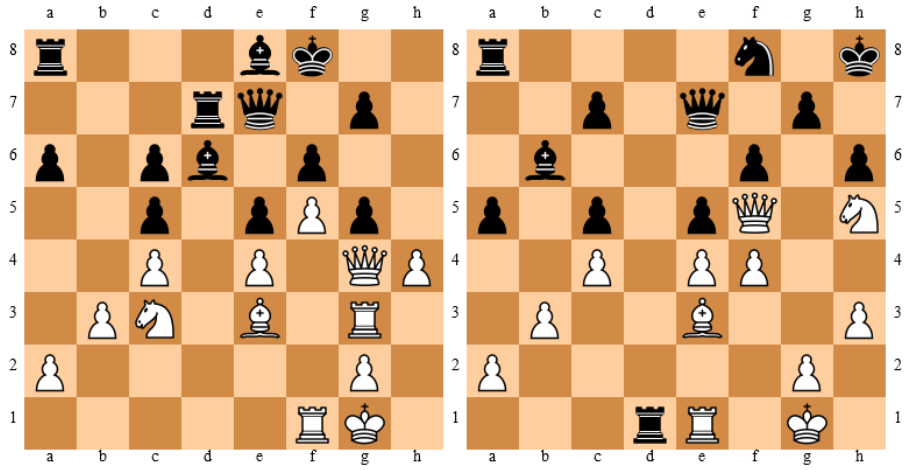
Slika 4.6: Porazdelitev odgovorov za prvi primer v anketi. Z rdečo barvo je označen vrstni red, ki ga predlaga naš algoritem.



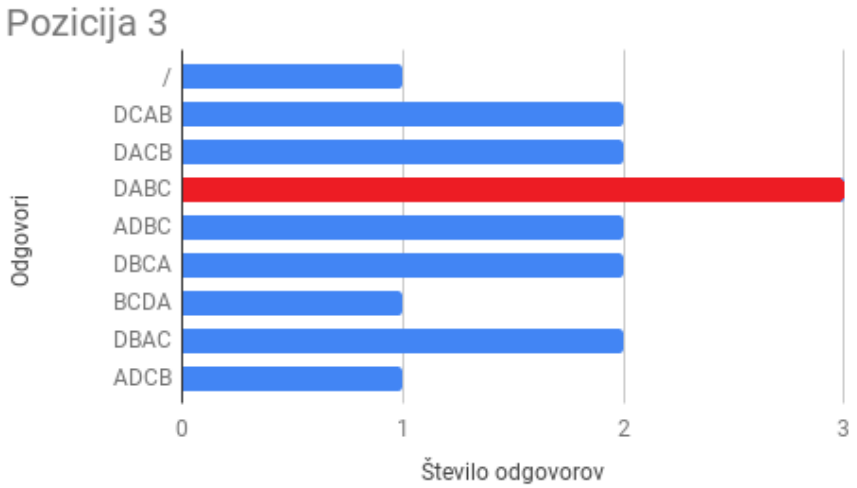
Slika 4.7: Drugi primer. Levo originalna pozicija, desno njej najbolj podobna.



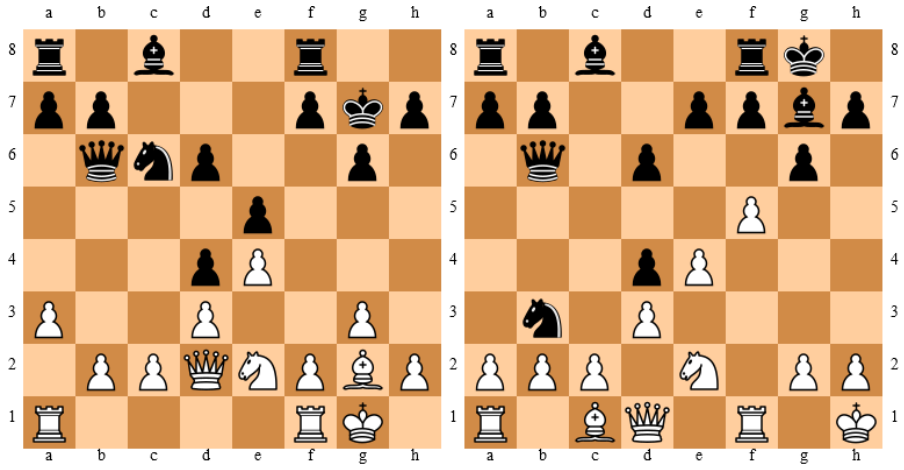
Slika 4.8: Porazdelitev odgovorov za drugi primer v anketi. Z rdečo barvo je označen vrstni red, ki ga predlaga naš algoritem.



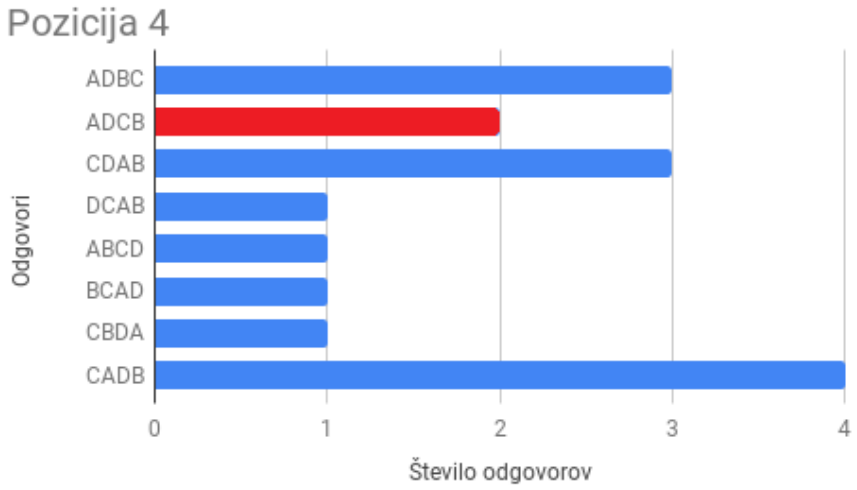
Slika 4.9: Tretji primer. Levo originalna pozicija, desno njej najbolj podobna.



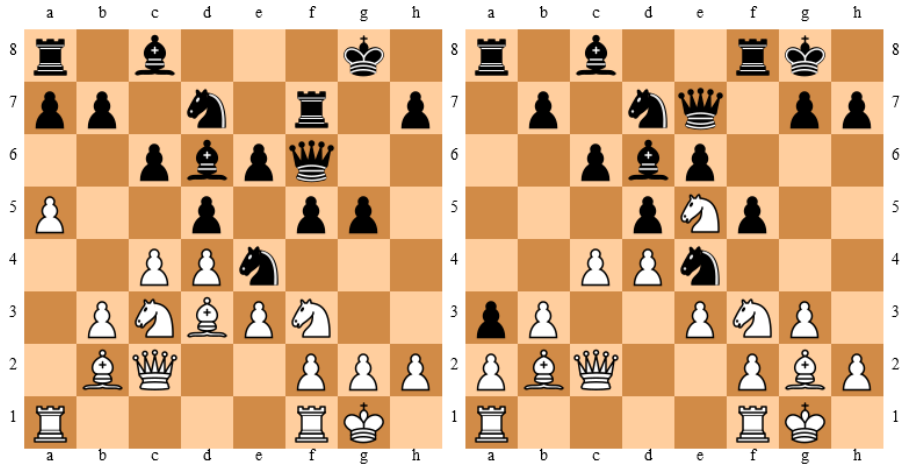
Slika 4.10: Porazdelitev odgovorov za tretji primer v anketi. Z rdečo barvo je označen vrstni red, ki ga predlaga naš algoritem.



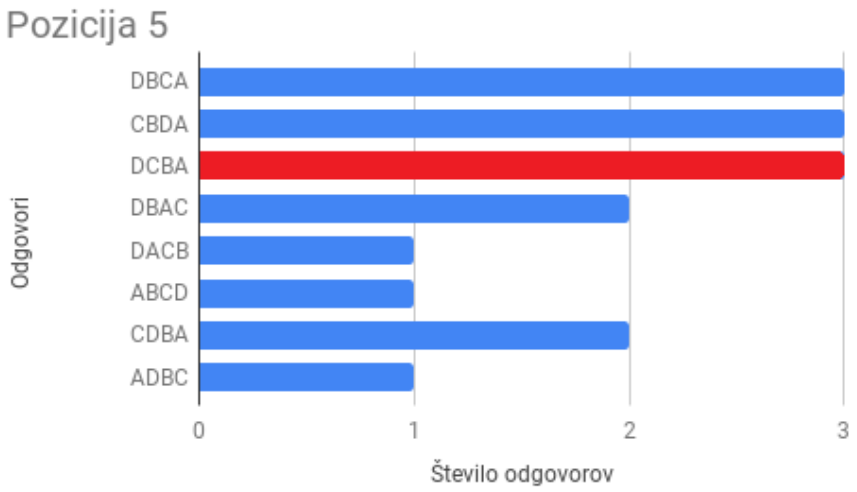
Slika 4.11: Četrti primer. Levo originalna pozicija, desno njej najbolj podobna.



Slika 4.12: Porazdelitev odgovorov za četrti primer v anketi. Z rdečo barvo je označen vrstni red, ki ga predlaga naš algoritem.

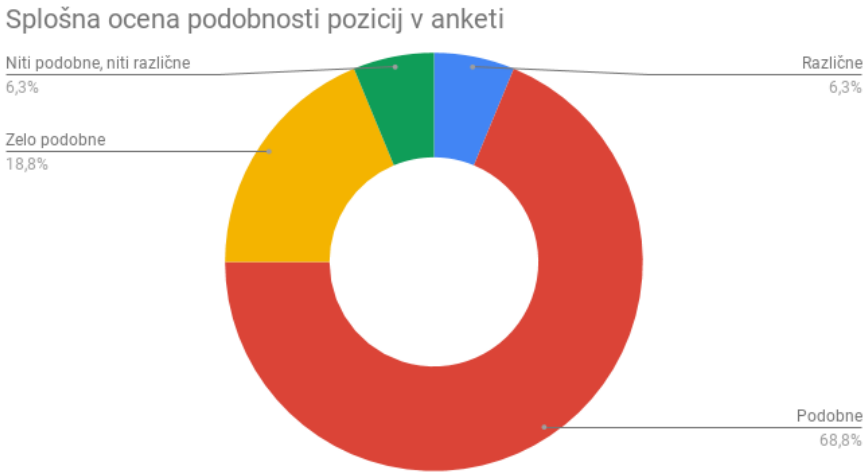


Slika 4.13: Peti primer. Levo originalna pozicija, desno njej najbolj podobna.



Slika 4.14: Porazdelitev odgovorov za peti primer v anketi. Z rdečo barvo je označen vrstni red, ki ga predlaga naš algoritem.

Pri zadnjem vprašanju smo anketiranca prosili, da opisno oceni (v povprečju) podobnost najbolj podobne pozicije z originalno. Šahisti so večinsko ocenili pozicije kot precej podobne, kar lahko vidimo tudi na tortnem diagramu na sliki 4.15. Noben od anketirancev ni ocenil pozicij kot zelo različnih.



Slika 4.15: Tortni diagram z mnenjem šahistov o podobnosti pozicij.

Glavni ugotovitvi na pogladi ankete:

- Med šahisti ni jasnega strinjanja o podobnosti pozicij. Podobnost je resnično subjektivna, kar lepo dokazujejo zgornji histogrami.
- Vsi šahisti so opazili podobnost med izhodiščno pozicijo in samodejno najbolj podobno pozicijo iz partij Magnusa Carlsena.

Poglavje 5

Zaključki

Cilj diplomske naloge je bil izdelati program za samodejno prepoznavanje pozicijskih motivov v šahovskih pozicijah. Izmed mnogih pozicijskih motivov smo se odločili za prepoznavanje kmečkih struktur. Implementirali smo samodejno prepoznavanje kmečkih struktur: osamljenega kmeta, prostega kmeta, podvojenega kmeta, zaostalega kmeta, kmečke verige in kmečkega otoka. Pri programiranju smo se osredotočili na hitrost algoritmov za prepoznavanje, zato smo skušali v veliki meri uporabiti tehniko programiranja z bitnimi šahovnicami. V veliko pomoč v tem delu naloge je bila programska knjižnica *python-chess*. Ob testiranju in analizi rezultatov našega programa smo ugotovili, da so nekatere kmečke strukture bolj pogoste v šahovskih pozicijah. Znotraj tega sklopa diplomske naloge smo pripravili tudi tekstovni zapis posameznih kmečkih struktur.

Kot dodatni izziv smo se lotili iskanja podobnih šahovskih pozicij. Za ta namen smo se odločili uporabiti tehniko iskanja po besedilnih datotekah. Za prepoznavanje podobnih šahovskih pozicij smo uporabili odprtokodno knjižnico *Apacahe Lucene*. Pripravili smo si zbirko šahovskih pozicij, kjer smo vsako pozicijo zapisali kot besedilo. Pozicije smo opisali po zgledu iz članka *Retrieval of Similar Chess Positions* [10]. Posnemali smo zapis točne lokacije figur, zapis dosegljivih polj vsake figure in zapis napadov in obramb med figurami. Tem zapisom pa smo dodali še svoje zapise kmečkih struktur,

s katerimi smo želeli doseči večjo vsebinsko podobnost izhodnih rezultatov programa za iskanje podobnih šahovskih pozicij.

Pri evalvaciji rešitev obeh programov nam je pomagal domenski ekspert, FIDE šahovski mojster, ki je ugotovil 100 odstotno klasifikacijsko točnost samodejnega prepoznavanja kmečkih struktur. Prepoznavanje kmečkih struktur smo testirali na 104 primerih šahovskih pozicij. Baza za prepoznavanje podobnih pozicij pa je obsegala preko 2700 partij, kar znese preko 170.000 šahovskih pozicij. Evalvacijo programa za iskanje podobnih pozicij smo opravili s pomočjo ankete, katero je rešilo 18 šahistov. Ugotovili smo, da je podobnost zelo subjektivna, šahisti pa so se strinjali o podobnosti samodejno najdenih pozicij.

Koncepte samodejnega prepoznavanja kmečkih struktur lahko uporabimo kot osnovo sistema za samodejno komentiranje šahovskih pozicij oziroma partij. V prihodnje lahko prepoznavanje kmečkih struktur razširimo še na druge pozicijske motive (npr. lovci na oporiščih) in celo na samodejno prepoznavanje dinamičnih šahovskih motivov (npr. taktični motivi). Obe izboljšavi lahko uporabimo za še boljše delovanje programa za prepoznavanje podobnih šahovskih pozicij, katerega bi lahko še dodatno izboljšali z dodatnimi eksperimenti na obliki poizvedb, ki jih omogoča knjižnica *Apache Lucene*. Ta program je lahko dobra osnova za samodejno pripravo šahovskih treningov, kar bi nam odprlo vrata k hitrejši pripravi personaliziranih šahovskih treningov in kasneje tudi k njihovi avtomatizaciji.

Literatura

[1] Jean-Marc Alliot. Who is the master? *ICGA Journal*, 39(1):3–43, 2017.

[2] Bitne šahovnice. Dosegljivo: `https://en.wikipedia.org/wiki/Bitboard`. [Dostopano: 25. 09. 2018].

[3] Okapi bm25. Dosegljivo: `https://en.wikipedia.org/wiki/Okapi_BM25`. [Dostopano: 25. 09. 2018].

[4] Chebysheva razdalja. Dosegljivo: `https://en.wikipedia.org/wiki/Chebyshev_distance`. [Dostopano: 25. 09. 2018].

[5] Spletna apliakcija za učenje šaha chessable.com. Dosegljivo: `https://www.chessable.com`. [Dostopano: 25. 09. 2018].

[6] Taktični treningi na chess.com. Dosegljivo: `https://www.chess.com/tactics`. [Dostopano: 25. 09. 2018].

[7] G Costeff. The chess query language: Cql. *ICGA Journal*, 27(4):217–225, 2004.

[8] Forsyth–edwards notation (fen). Dosegljivo: `https://en.wikipedia.org/wiki/Forsyth-Edwards_Notation`. [Dostopano: 25. 09. 2018].

[9] Svetovna šahovska organizacija. Dosegljivo: `https://www.fide.com/`. [Dostopano: 25. 09. 2018].

[10] Debasis Ganguly, Johannes Leveling, and Gareth J.F. Jones. Retrieval of similar chess positions. In *Proceedings of the 37th International*

ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR '14, pages 687–696, New York, NY, USA, 2014. ACM.

[11] Matej Guid, Martin Možina, Jana Krivec, Aleksander Sadikov, and Ivan Bratko. Learning positional features for annotating chess games: A case study. In *International Conference on Computers and Games*, pages 192–204. Springer, 2008.

[12] Ernst A Heinz. How darkthought plays chess. *ICGA Journal*, 20(3):166–176, 1997.

[13] Jupyter notebook. Dosegljivo: `http://jupyter.org/`. [Dostopano: 25. 09. 2018].

[14] David Levy. *Computer chess compendium. poglavje Programming a Computer for Playing Chess*. Springer Science & Business Media, 2013.

[15] David Levy and Monroe Newborn. How computers play chess. In *All About Chess and Computers*, pages 24–39. Springer, 1982.

[16] Knjižnjica apache lucene. Dosegljivo: `http://lucene.apache.org/core/`. [Dostopano: 25. 09. 2018].

[17] Zbirka partij magnusa carlsena. Dosegljivo: `https://chess-db.com/public/pinfo.jsp?id=1503014`. [Dostopano: 25. 09. 2018].

[18] D. Marovic. *Understanding Pawn Play in Chess*. Grandmaster Shows How to Make the Most of Your Pawns. Gambit, 2000.

[19] Portable game notation (pgn). Dosegljivo: `https://en.wikipedia.org/wiki/Portable_Game_Notation`. [Dostopano: 25. 09. 2018].

[20] F.D. Philidor. *Le jeu des échecs*. 1792.

[21] Knjižnjica python-chess. Dosegljivo: `http://python-chess.readthedocs.io/en/latest/`. [Dostopano: 25. 09. 2018].

[22] Aleksander Sadikov, Martin Možina, Matej Guid, Jana Krivec, and Ivan Bratko. Automated chess tutor. In *International Conference on Computers and Games*, pages 13–25. Springer, 2006.

[23] Standardni algebrski zapis. Dosegljivo: `https://en.wikipedia.org/wiki/Algebraic_notation_(chess)`. [Dostopano: 25. 09. 2018].

[24] Simon Stoiljkovikj, Ivan Bratko, Matej Guid, and FRI UNI. A computational model for estimating the difficulty of chess problems. In *Proceedings of the Third Annual Conference on Advances in Cognitive Systems ACS*, page 7, 2015.

[25] Atsushi Ushiku, Shinsuke Mori, Hirotaka Kameko, and Yoshimasa Tsuruoka. Game state retrieval with keyword queries. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 877–880. ACM, 2017.

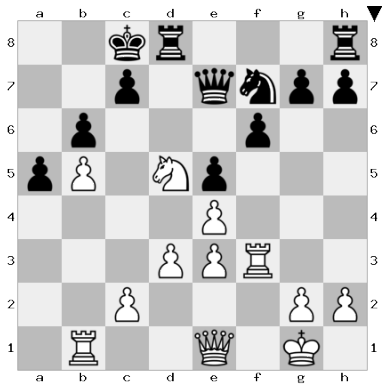
[26] Moshé M Zloof. Query-by-example: the invocation and definition of tables and forms. In *Proceedings of the 1st International Conference on Very Large Data Bases*, pages 1–24. ACM, 1975.

[27] Albert L Zobrist. A new hashing method with application for game playing. *ICCA journal*, 13(2):69–73, 1970.

Dodatek A

Rezultati prepoznavanja kmečkih struktur

Zaradi preglednosti je tabela A.1 prikazana na naslednji strani, izpustili pa smo tudi izhode funkcije za štetje kmečkih struktur.



Slika A.1: Primer šahovske pozicije iz spodnje tabele (*ID: X_018*).

Izhoda našega programa za primer na sliki A.1:

```
Isolated: /; Passed: 'fa5; Doubled: S{e3-e4}
Backwards: lc7;
Chains: W[c2/d3/e4] w[c7/b6/a5] w[g7/f6/e5];
Islands: P(2) p(2);
```

Tabela A.1: Tabela rezultatov prepoznavanja kmečkih struktur.

ID	*FEN*	*Isolated*	*Passed*	*Doubled*	*Backwards*	*Chains*
X_000	rnbq1rk1/pp3ppp/5n2/2bp4/1P6/P1N1P1N1/5PPP/R1BQKB1R w KQ - 0 1	id5	-	-	-	-
X_001	r2q1rk1/pp3pp1/1np4p/5bb1/3P4/1BN2N1P/PP3PP1/2RQR1K1 w - - 0 1	Id4	-	-	-	-
X_002	2rq1rk1/pb2bppp/1pn1pn2/8/3P4/P1NQ1N2/1PB2PPP/R1B1R1K1 w - - 0 1	Id4	-	-	-	-
X_003	2r1r1k1/1b3p1p/ppn1pbpB/7q/3P4/PB2QN2/1P3PPP/3RR1K1 w - - 0 1	Id4	-	-	-	-
X_004	r2qr1k1/pb2bpp1/1pn1pn1p/8/1P1P4/P1NQBN2/2B2PPP/R3R1K1 b - - 0 1	Id4	-	-	-	-
X_005	r1bq1rk1/4bppp/p3pn2/np1P4/8/P1N2N2/1PB2PPP/R1BQR1K1 w - - 0 1	Id5	-	-	-	-
X_006	r2q1r2/pp1b1pkp/6p1/3pN3/3P4/1Q6/PP3PPP/R3R1K1 w - - 0 1	Id4,id5	-	-	-	-
X_007	r3r1k1/1p2nppp/1qn5/p2p4/3Nb3/1NP3P1/PP3PP1/R2QRBK1 w - - 0 1	id5	-	S{g2-g3}	-	-
X_008	2r3k1/1R2Rpp1/1p1r2b1/pB1p4/P2P1PP1/8/1P6/6K1 w - - 0 1	Id4,id5	-	-	lb6	-
X_009	r1r5/pp1qnk1p/4Npp1/3p4/6Q1/8/PP3PPP/2R1R1K1 w - - 0 1	id5	fd5	-	-	-
X_010	r2qr1k1/1b2bppp/3n1n2/pP1p4/3N4/1PNQ1P2/2B3PP/R1B1R1K1 w - - 0 1	Ib3,ia5,Ib5,id5	Fb5,fd5	S{b3-b5}	-	-
X_011	1n2n1k1/2b2p1p/1BP3p1/p2p4/8/1PN2P2/4B1PP/6K1 w - - 0 1	ia5,id5	fd5,Fc6	-	Lb3	-
X_012	r4bk1/3q2pp/8/p2pBpQ1/3P4/1P3R1P/5PP1/6K1 w - - 0 1	Ib3,Id4,ia5,id5	-	-	-	-
X_013	3r2k1/r3bppp/pNQ1p3/q2P4/8/8/1P3PPP/2RR2K1 w - - 0 1	Ib2,Id5,ia6	-	-	-	-
X_014	2r2rk1/p3npp1/2p1p2p/q7/3P4/1QN5/PP3PPP/3RR1K1 w - - 0 1	Id4,ic6,ia7	-	-	-	-
X_015	6k1/5p2/2r1p1p1/p2p1n2/1q1P3p/1P5P/P3NPP1/1Q1R2K1 w - - 0 1	Id4,ia5	-	-	-	w[f7/e6/d5]
X_016	r5k1/1p1r1p2/2n2Qpp/3p1B2/p7/2P4P/Pq3PP1/3RR1K1 w - - 0 1	Ia2,Ic3,id5	-	-	-	-
X_017	r2qk1nr/1pp2ppp/p1p5/4p3/4P1b1/2NPPN2/PPP3PP/R2QK2R w KQkq - 0 1	-	-	S{e3-e4},s{c6-c7}	-	W[c2/d3/e4]
X_018	2kr3r/2p1qnpp/1p3p2/pP1Np3/4P3/3PPR2/2P3PP/1R2Q1K1 w - - 0 1	-	'fa5	S{e3-e4}	lc7	W[c2/d3/e4],w[c7/b6/a5],w[g7/f6/e5]
X_019	r2qk2r/p1pbbppp/2pp1n2/8/3NP3/2N5/PPP2PPP/R1BQ1RK1 w - - 0 1	ia7	-	s{c6-c7}	-	-
X_020	3r1bk1/p1p2ppp/8/PrpnPN2/3p4/1P6/2PB1PPP/1R2R1K1 w - - 0 1	ia7	-	s{c5-c7}	Lc2	-
X_021	r2qkb1Q/2p2p1p/p3pn2/1p3p2/3P4/3B1P2/PPP2P1P/R3K2R w - - 0 1	If2,Ih2,If3,ih7	-	S{f2-f3},s{f5-f7}	-	-
X_022	3rrb1k/1q4pp/pNp1bp2/P1pp4/4PB2/1P1Q4/2P2PPP/3RR1K1 w - - 0 1	ia6	-	s{c5-c6}	-	-
X_023	1k1n1r1r/p1p1qppp/1p2n3/2p1p1PQ/4P2P/P2PB1PB/1PP2R1K/5R2 w - - 0 1	-	-	S{g3-g5},s{c5-c7}	-	W[c2/d3/e4]
X_024	r3kb1r/1p1bpppp/1pn5/3p4/3P4/4PN2/PP1B1PPP/R3KB1R w - - 0 1	ib6,ib7	-	s{b6-b7}	-	W[f2/e3/d4]
X_025	2r1k2r/pp1bnppp/2n1p3/q1ppP3/P2P4/2P2N2/2PQBPPP/R1B1K2R w - - 0 1	Ia4	-	S{c2-c3}	-	W[c3/d4/e5],w[f7/e6/d5]
X_026	r2n1rk1/ppp1qppp/3p1nb1/1B1Pp1B1/4P1P1/2P2N1P/P1P2P2/R2QR1K1 w - - 0 1	Ia2	-	S{c2-c3}	-	w[c7/d6/e5]
X_027	3r1rk1/p3qpp1/4n1b1/2ppp1n1/4P1P1/2PB1P2/PNP3K1/R1BQR3 w - - 0 1	Ia2,Ic2,Ic3,ia7	-	S{c2-c3}	Lf3	-
X_028	4q1k1/p1p3pp/bp2p3/n4p2/2PP4/B1PB4/P4PPP/R4RK1 w - - 0 1	Ia2	-	S{c3-c4}	-	-
X_029	r3bk2/3rq1p1/p1pb1p2/2p1pPp1/2P1P1QP/1PN1B1R1/P5P1/5RK1 w - - 0 1	ic5,ia6,ic6	-	s{c5-c6},s{g5-g7}	-	w[g7/f6/e5]
X_030	6k1/p1nr2q1/bp2p2p/n1p1Ppp1/2P2P2/PBP1B3/4q1PP/2N2RK1 w - - 0 1	Ia3,Ic3,Ic4	-	S{c3-c4}	-	-
X_031	r1r3k1/5ppp/p7/3n4/8/5PBP/P4P2/R4RK1 w - - 0 1	Ia2,If2,If3,Ih3,ia6	-	S{f2-f3}	-	-
X_032	8/6p1/p4p1p/2R1nk2/3K4/r4PBP/P4P2/8 w - - 0 1	Ia2,If2,If3,Ih3,ia6	-	S{f2-f3}	-	-
X_033	1r3rk1/p1pn1ppp/4pq2/2p3Q1/3P3P/7R/PPP2PP1/2KR1B2 w - - 0 1	ic5,ia7,ic7	-	s{c5-c7}	-	-
X_034	7r/3nkppp/B1p1p3/3p4/3P4/bR2P3/P2BKPPP/8 w - - 0 1	Ia2	Fa2	-	-	W[f2/e3/d4],w[f7/e6/d5]
X_035	8/bB2k1pp/P2rp3/5p2/5P1P/2R1P3/4K1P1/8 w - - 0 1	Ia6	Fa6	-	le6	-
X_036	r5k1/2p2qpp/2p2p2/2Pb4/pB1P4/Q4P2/2P3PP/4R1K1 w - - 0 1	ia4,ic6,ic7	fa4	S{c2-c5},s{c6-c7}	-	-
X_037	2b5/2p5/8/2Pp1ppk/p2P3p/5PP1/5K1P/2B5 w - - 0 1	ia4	fa4	-	-	-
X_038	8/2Q2pk1/5qpp/r7/P7/7P/R4PP1/6K1 w - - 0 1	Ia4	Fa4	-	-	-

Tabela A.1: Tabela rezultatov prepoznavanja kmečkih struktur.

ID	*FEN*	*Isolated*	*Passed*	*Doubled*	*Backwards*	*Chains*
X_039	2r3k1/p4pp1/1p6/3P2p1/P7/1P5P/2N3P1/6K1 w - - 0 1	Id5	Fd5	s{g5-g7}	-	-
X_040	1n6/4k2p/p3ppp1/1pPp4/3P1PP1/3NP3/P3K2P/8 w - - 0 1	Ia2	'Fc5	-	Le3	W[e3/d4/c5]
X_041	r2r2k1/p4Rbp/Bp4p1/n4PN1/4p1b1/8/P2B2PP/4K2R w - - 0 1	Ia2,ie4	fe4	-	-	-
X_042	2k1r3/2pp2pp/p2B4/1p1b1p2/8/1BP2PQ1/PP1p2PP/4rRK1 w - - 0 1	-	fd2	s{d2-d7}	-	-
X_043	5r1k/pqnPb2p/1p1R2p1/2p5/2Q5/8/P4PPP/1R4K1 w - - 0 1	Ia2,Id7	'fc5,Fd7	-	-	-
X_044	2r2rk1/pb2q2p/1p3pp1/n2PN2Q/1b6/3B4/PB3PPP/3R1RK1 w - - 0 1	Ia2,Id5	Fd5	-	-	-
X_045	2r2bk1/1b5p/pn3rp1/1ppPpp2/P3P3/1PN1BP2/1K2B1PP/2R4R w - - 0 1	-	'Fd5	-	Lb3	W[g2/f3/e4/d5]
X_046	2rq1rk1/pb2npbp/2n1p1p1/2P5/8/2N2NP1/3BPPBP/1R1Q1RK1 w - - 0 1	Ic5,ia7	Fc5,fa7	-	-	-
X_047	3r2k1/p3qp1p/6p1/3np3/3Q4/5BP1/4PP1P/1R4K1 w - - 0 1	ia7	fa7	-	-	-
X_048	r1bb1rk1/1pp3pp/p1p5/2n1Pp2/5B2/2N2N2/PPP2PPP/R3R1K1 w - - 0 1	-	Fe5	s{c6-c7}	-	-
X_049	r2q1rk1/ppp1n1bp/4P1p1/2p1pp2/2P1P3/2N3P1/PP3PBP/R2Q1RK1 w - - 0 1	-	Fe6	S{e4-e6},s{c5-c7}	-	-
X_050	rn1qr1k1/pb1p1ppp/1p1Bpn2/8/2Pp4/P4NP1/1P2PPBP/R2Q1RK1 w - - 0 1	-	-	s{d4-d7}	-	-
X_051	3qn1k1/p1rprppp/Bp2p3/8/1PP5/P2Q2P1/4PP1P/2RR2K1 w - - 0 1	-	-	-	-	-
X_052	rnbq1rk1/2p1p1bp/p3pnp1/8/R2P4/1QN1BN2/1P3PPP/4KB1R w - - 0 1	Ib2,Id4,ia6,ie6,ic7,ie7	-	s{e6-e7}	-	-
X_053	5r1k/2p1p1b1/1nn3pp/8/3PN2P/4BN2/2r2PP1/R4RK1 w - - 0 1	Id4,ic7,ie7	-	-	-	-
X_054	rn1q1rk1/pbp1bppp/1p6/4N3/3Pp3/6P1/PPQ1PPBP/R1B2RK1 w - - 0 1	-	-	-	-	-
X_055	r1bq1rk1/pp2p2p/3p2p1/5n2/2P5/2N5/PP1QBPPP/R4RK1 w - - 0 1	-	-	-	-	-
X_056	r1bq1rk1/pp5p/3p2p1/4pn2/2P5/2N5/PP1QBPPP/R4RK1 w - - 0 1	-	-	-	ld6	-
X_057	r2q1rk1/pp1bp1bp/2np4/5p2/2PN1Pn1/2N1B3/PP1QB1PP/R4RK1 w - - 0 1	ih7	-	-	-	-
X_058	r6k/pp1bprqp/2np4/3N1P2/2P3p1/4B3/PP1Q2PP/R4RK1 w - - 0 1	-	-	-	-	-
X_059	2r3k1/p2pq1pp/1pn1p3/1N3r2/2P2P2/1P4PP/Pb2Q1K1/3R1R2 w - - 0 1	-	-	-	-	-
X_060	r2q1rk1/pb3ppp/3p4/2p5/8/4B3/PPP1QPPP/3RR1K1 w - - 0 1	ia7	-	-	-	-
X_061	3r2k1/p4ppp/2bpq2B/2p1r3/2P5/6QP/PP1R1PP1/3R2K1 w - - 0 1	ia7	-	-	-	-
X_062	r1b1k2r/pp4pp/1qnbpn2/3p4/3P4/3B1N2/PP2NPPP/R1BQ1RK1 w - - 0 1	Id4	-	-	-	-
X_063	r1b3k1/pp2n1pp/2n1pr2/q2p4/P1pP4/2P2NP1/2PQ1PBP/R1B1K2R w - - 0 1	Ia4	-	S{c2-c3}	Lc3,le6	w[e6/d5/c4]
X_064	2r1n3/1p2k2p/2bp2r1/4pp2/p1P5/P3BP2/1P1RBKPP/3R4 w - - 0 1	ih7	-	-	Lb2,ld6	-
X_065	r2r1nk1/pp3qpp/2p1pp2/8/3P1P1P/5QP1/PPP5/2KRRB2 w - - 0 1	-	-	-	le6	-
X_066	rn1qr1k1/pp3pbp/3pbnp1/2p5/2P1P3/2N2N2/PP2BPPP/R1BQ1RK1 w - - 0 1	-	-	-	ld6	-
X_067	r2qr1k1/pp3pbp/n2pbnp1/2p5/2P1P3/2N2N2/PP2BPPP/R1BQR1K1 w - - 0 1	-	-	-	ld6	-
X_068	r1bqr1k1/1p3pbp/2pp1np1/p1n5/2PNP3/2N3PP/PP3PB1/1RBQR1K1 w - - 0 1	-	-	-	ld6	-
X_069	r1bqkb1r/5p1p/p1np1p2/1p2p3/4P3/N1N5/PPP2PPP/R2QKB1R w - - 0 1	ih7	-	s{f6-f7}	ld6	-
X_070	1r1q1r1k/5p1p/3pb2b/1p1Bp3/8/2P1N1P1/1P3P1P/R2Q1RK1 w - - 0 1	ib5,ih7	-	-	-	-
X_071	r4rk1/1pq1bppp/p2pbn2/4n3/P3PR2/1NN1B3/1PP1B1PP/R2Q3K w - - 0 1	Ie4,id6	-	-	-	-
X_072	r1bq1rk1/pp2bppp/2np1n2/4p3/4PP2/1NN1B3/PPP1B1PP/R2Q1RK1 w - - 0 1	-	-	-	ld6	-
X_073	2rb1rk1/5ppp/p2p1n2/1p1BpP2/4P3/P3q1N1/1PP3PP/R2Q1R1K w - - 0 1	-	-	-	ld6	-
X_074	r1bq1rk1/pp2bppp/2np4/4pP2/4P1n1/2N1BN2/PPPQB1PP/R3K2R w - - 0 1	-	-	-	ld6	-
X_075	r1b1k1nr/pp3ppp/2n1p3/2PpP3/q7/P1P2N2/2PQ1PPP/R1B1KB1R w - - 0 1	Ic2,Ia3,Ic3,Ic5	-	S{c2-c3-c5}	-	w[f7/e6/d5]
X_076	r3k1r1/ppqbnp2/2n1p3/3pP3/5P2/P1pQ4/2P1N1PP/R1B1KB1R w - - 0 1	Ic2,Ia3	Fh2	-	lf7	w[f7/e6/d5]
X_077	2kr2r1/ppqb1p2/2n1p3/3pP3/3B1P2/P1pQ2P1/2P4P/R3KB1R w - - 0 1	Ic2,Ia3	Fh2	-	lf7	W[g3/f4/e5],w[f7/e6/d5]
X_078	r1b1kb1r/pp1n1ppp/2n1p3/q1ppP3/3P1P2/2P2N2/PP3KPP/R1BQ1BNR w - - 0 1	-	-	-	-	W[b2/c3/d4/e5],w[f7/e6/d5]

Tabela A.1: Tabela rezultatov prepoznavanja kmečkih struktur.

ID	*FEN*	*Isolated*	*Passed*	*Doubled*	*Backwards*	*Chains*
X_079	r1bq1rk1/1p1nbppp/p1p1pn2/2Pp2B1/3P4/2NBPN2/PP3PPP/2RQK2R w - - 0 1	-	-	-	-	W[f2/e3/d4/c5],w[b7/c6/d5],w[f7/e6/d5]
X_080	r2qnrk1/1p2bppp/p1p5/2npP3/5Bb1/2N1PN2/PP3PPP/1BRQK2R w - - 0 1	-	-	S{e3-e5}	-	w[b7/c6/d5]
X_081	r3k2r/pp1b1ppp/1qn1p3/3pPn2/1b1P4/1P3N2/PB2BPPP/RN1Q1K1R w - - 0 1	-	-	-	-	w[f7/e6/d5]
X_082	2r1k2r/1p1b1pp1/1qn1p2p/p2pP3/Pn1P2P1/1P3N2/1B3PKP/1BRQ3R w - - 0 1	-	-	-	Lb3	w[f7/e6/d5]
X_083	r3kb1r/p2n1ppp/b1n1p3/q1ppP3/1p1P1P2/2PB1N2/PP2NKPP/R1BQ3R w - - 0 1	-	-	-	-	W[b2/c3/d4/e5],w[f7/e6/d5]
X_084	r2nk2r/p2nb1pp/4p3/q2pP3/2pP1N1P/2P2N2/P1Q2KP1/R1B4R w - - 0 1	Ia2,ia7	-	-	le6	W[c3/d4/e5],w[e6/d5/c4]
X_085	r1bq1bkn/pp4r1/3p1n2/PN1Pp2p/1P2Ppp1/5P1P/3BBNP1/2RQ1RK1 w - - 0 1	-	-	-	Lg2	W[g2/f3/e4/d5],w[d6/e5/f4]
X_086	r2q2k1/ppN2rbp/3p1nn1/3Pp3/4Pp2/3N1P1b/PPQBB2P/2R2RK1 w - - 0 1	Ih2,ih7	-	-	-	W[f3/e4/d5],w[d6/e5/f4]
X_087	N4k2/pp3rbp/3p4/3Pp3/4Pp1n/3N3b/PPQBB1qP/2R1KR2 w - - 0 1	Ih2,ih7	'ff4	-	ld6	w[d6/e5/f4]
X_088	8/pp2kppp/4p3/8/1Pr5/P3PP2/3K1P1P/2R5 w - - 0 1	Ih2	-	S{f2-f3}	-	-
X_089	2r1k2r/p3bppp/1p2pn2/6B1/3P4/5N2/PPP2P1P/R3K2R w - - 0 1	If2,Ih2	-	-	-	-
X_090	2r4k/p1p2p1p/5p2/8/8/3r4/PP3PPP/R4RK1 w - - 0 1	if6,ia7,ic7,if7,ih7	-	s{f6-f7}	-	-
X_091	6k1/1R6/p4p2/7p/7P/5KP1/r4P2/8 w - - 0 1	ih5,ia6,if6	fa6	-	-	W[f2/g3/h4]
X_092	5k2/R7/p5K1/6P1/5r2/8/8/8 w - - 0 1	Ig5,ia6	Fg5,fa6	-	-	-
X_093	7r/p1pk2pp/2p2p2/2Pp4/3P4/8/PP1K1PPP/4R3 w - - 0 1	ia7	-	s{c6-c7}	lc6	-
X_094	2k5/p1p5/P1p4p/2Pp4/1R1P2pr/4KP2/1P4P1/8 w - - 0 1	ia7	-	s{c6-c7}	lc6	-
X_095	2r2rk1/p1pp1p1p/2p2p2/8/8/2PP1P2/PP3P1P/R3R1K1 w - - 0 1	If2,Ih2,If3,if6,ia7,if7,ih7	-	S{f2-f3},s{c6-c7},s{f6-f7}	-	-
X_096	r7/pkp4p/2p1p3/R1Pp1p2/3P1P2/8/PP1K1P1P/8 w - - 0 1	If2,Ih2,If4,ia7,ih7	-	S{f2-f4},s{c6-c7}	lc6,le6	-
X_097	3rr1k1/4b2p/p1q2pp1/1pp1p1P1/2n1P1Q1/2P5/PPB2PP1/R1B1R1K1 w - - 0 1	-	-	S{g2-g5}	lh7	-
X_098	1b1n4/p6p/1p2pkp1/2p5/2P2P2/1PN1B1P1/P5KP/8 w - - 0 1	ie6	-	-	-	-
X_099	8/p1bk4/1p2pBpn/2p5/2P2PP1/1P3NK1/P7/8 w - - 0 1	ie6,ig6	-	-	-	-
X_100	2b3k1/1p3ppp/p4n2/3p4/3N4/1P2PP2/P3B1PP/6K1 w - - 0 1	id5	-	-	Le3	-
X_101	2b5/5p1p/p1k3p1/P2p4/1K3P2/3BP1P1/7P/8 w - - 0 1	Ia5,id5,ia6	-	-	-	-
X_102	r4b1r/p2qkppp/4p3/1Q6/8/2P2P2/P4P2/1RB1K3 w - - 0 1	Ia2,If2,Ic3,If3,ia7	Fc3,fh7	S{f2-f3}	-	-
X_103	7r/B5bp/1R2p3/P2k1pp1/8/2P2P2/5P1P/3K4 w - - 0 1	If2,Ih2,Ic3,If3,Ia5	Fc3,Fa5	S{f2-f3}	-	-

Dodatek B

Anketa o podobnosti šahovskih pozicij

Chess position similarity assessment

One of my tasks for my engineering degree thesis was to develop an application for finding similar chess position. This survey will help with evaluation of my algorithm.

* Zahtevano

What's your FIDE ELO Rating? *

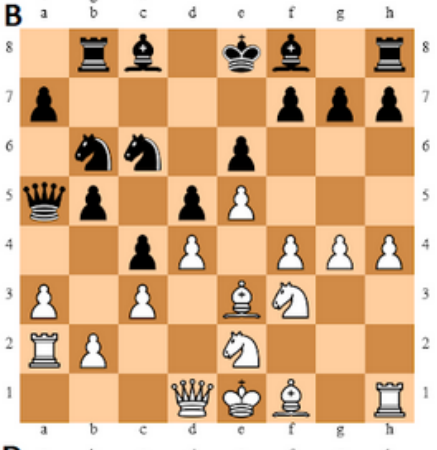
- ◯ Below 2000
- ◯ 2000 - 2099
- ◯ 2100 - 2199
- ◯ 2200 - 2299
- ◯ 2300 - 2399
- ◯ 2400 or more

Position 1 *

The first position is an original position. Sort positions A, B, C, and D, starting with the most similar positions to the original. (Answer example: "ABCD")

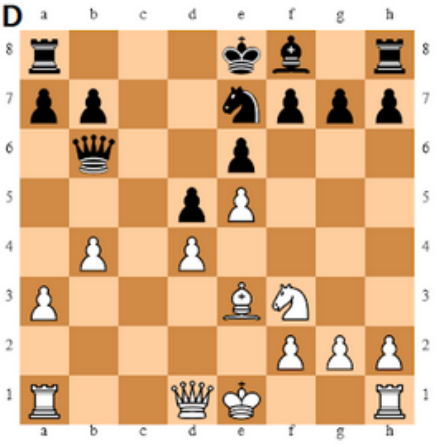
A

B



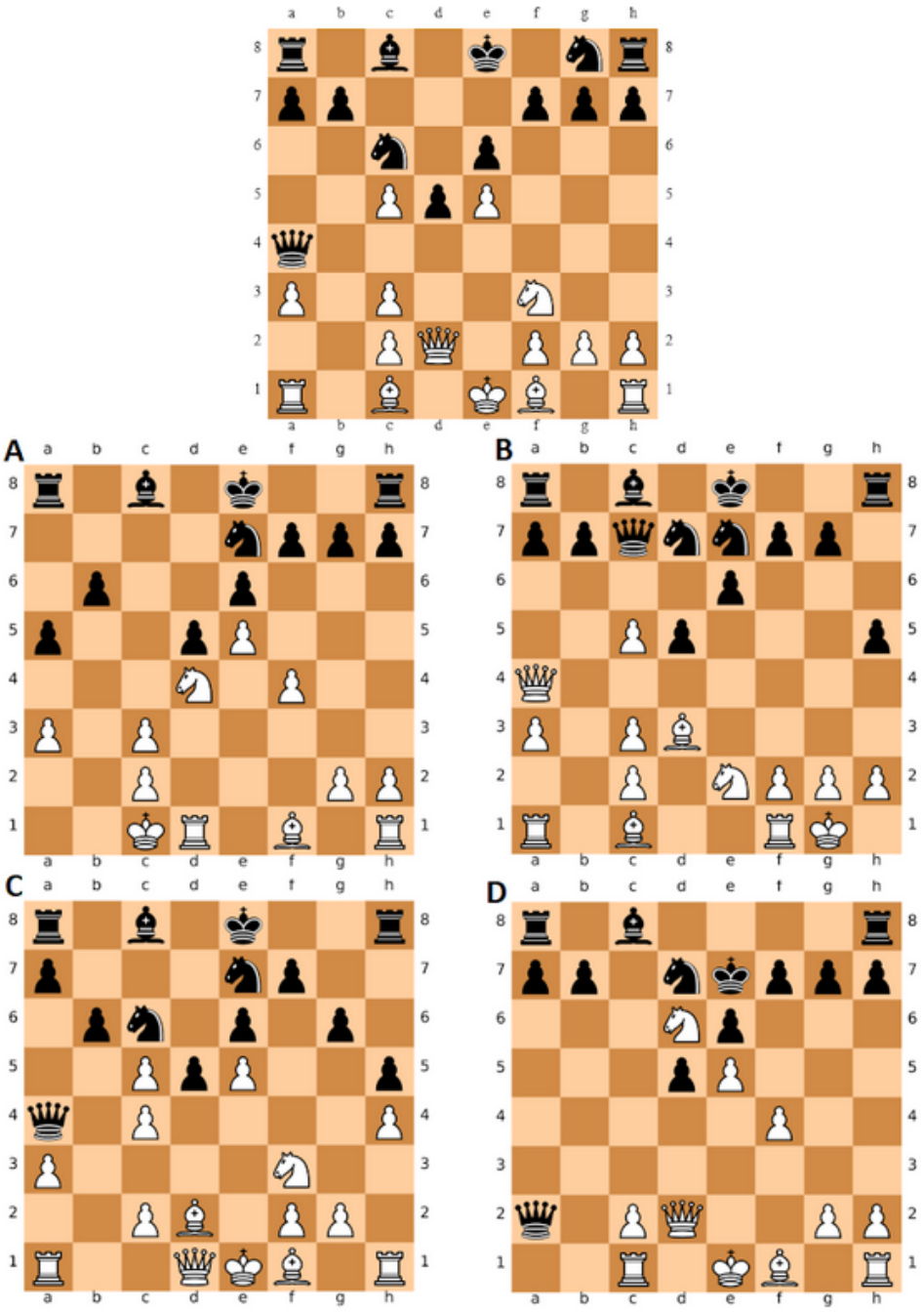
C

D



Position 2 *

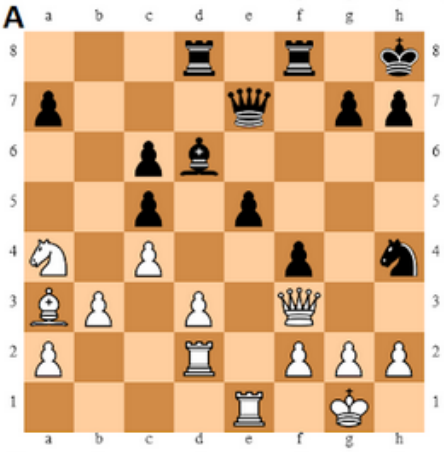
The first position is an original position. Sort positions A, B, C, and D, starting with the most similar positions to the original. (Answer example: "ABCD")



Position 3 *

The first position is an original position. Sort positions A, B, C, and D, starting with the most similar positions to the original. (Answer example: "ABCD")

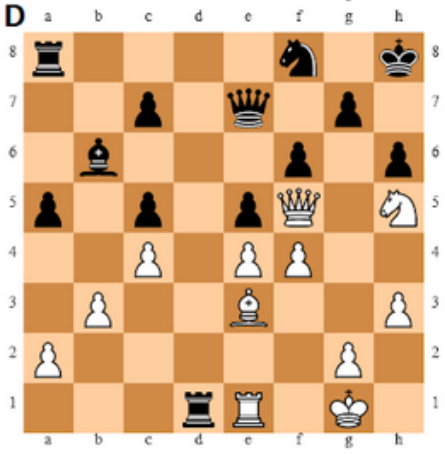
A



B

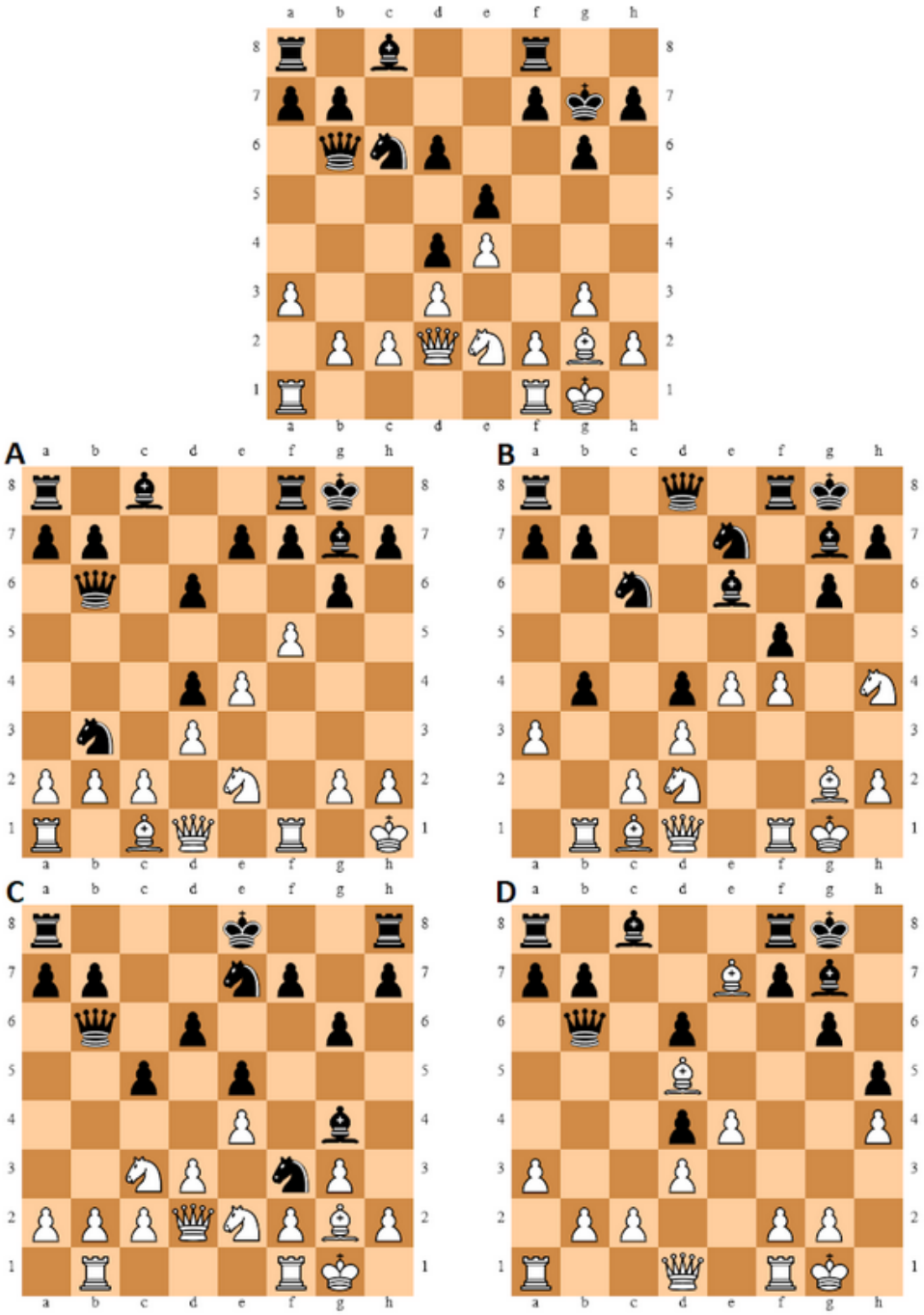
C

D



Position 4 *

The first position is an original position. Sort positions A, B, C, and D, starting with the most similar positions to the original. (Answer example: "ABCD")

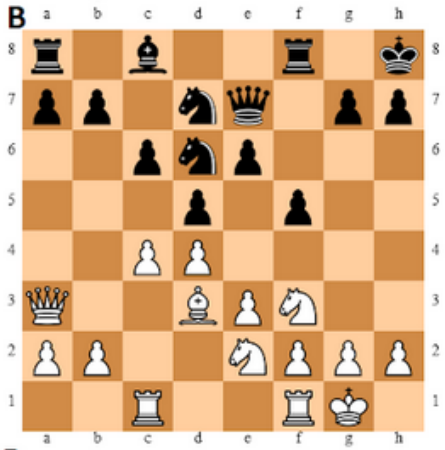


Position 5 *

The first position is an original position. Sort positions A, B, C, and D, starting with the most similar positions to the original. (Answer example: "ABCD")

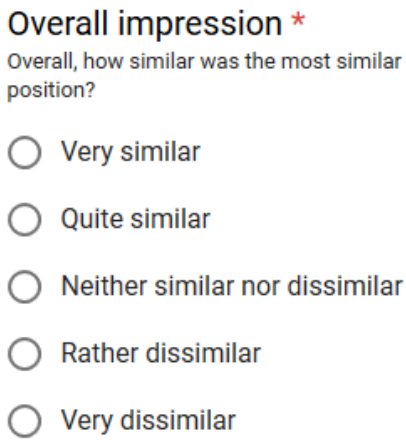
A

B



C

D


Overall impression *
Overall, how similar was the most similar position (on average) to the corresponding original position?
Very similar
Quite similar
Neither similar nor dissimilar
Rather dissimilar
Very dissimilar