Algoritmi za igranje šaha i tehnike analize pozicija

Duda, Daniel

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Daniel Duda

Algoritmi za igranje šaha i tehnike analize pozicija

Završni rad

Mentor: prof. dr. sc. Ana Meštrović

Rijeka, rujan 2024.

Sažetak

Ovaj završni rad bavi se razvojem Python desktop aplikacije za igranje i analizu šahovskih partija, s posebnim naglaskom na implementaciju algoritama za igranje šaha i analizu pozicija. Aplikacija je razvijena korištenjem Tkintera za grafičko sučelje te *python-chess* biblioteke za šahovsku logiku i integraciju PGN (eng. *Portable Game Notation*) formata. Cilj aplikacije je pružiti korisnicima intuitivno sučelje za igranje šaha protiv računalnog protivnika, koristeći *Chess Engine* s prilagođenih 10 razina težine, od najslabije prema najsnažnijoj. Aplikacija omogućuje pregled i pretraživanje baze podataka s više od 19.000 partija, pohranjenih u SQLite formatu.

Kao glavnu funkcionalnost nudi ploču za analizu (eng. *Analysis board*), koja korisnicima omogućuje povlačenje poteza, pohranjivanje tih poteza u PGN formatu, te pregled različitih varijanti za svaki potez. Osim toga, korisnik može pretraživati bazu podataka kako bi provjerio postoji li u bazi partija s identičnom pozicijom i, ako postoji, dobiti informacije o igračima i potezima koje su odigrali u toj situaciji. Nadalje, „Analysis board“ nudi pokretanje *chess enginea* koji na temelju trenutne pozicije vraća i prikazuje najjače moguće poteze.

Tijekom razvoja aplikacije naišao sam na izazove u optimizaciji pretraživanja velike baze partija, što je dovelo do određenih kašnjenja u obradi podataka. Iako aplikacija ne nudi značajan doprinos šahovskoj zajednici u usporedbi s naprednim alatima poput ChessBase 17, predstavlja koristan alat za početnike i napredne igrače u analizi partija i poboljšanju šahovskih vještina. U budućnosti planiram proširenje funkcionalnosti, uključujući poboljšanja korisničkog sučelja i dodavanje opcije igranja online partija.

Ključne riječi: Python; Tkinter; SQLite; desktop aplikacija; *python-chess*; PGN; *chess engine*; FEN; Analysis board; ChessBase

SADRŽAJ

1. Uvod

Tema ovog rada bavi se razvojem šahovskih algoritama i aplikacija za analizu partija, što je područje usko povezano s informatičkom strukom, posebno u domeni umjetne inteligencije. Povijesno gledano, računala su prolazila kroz faze u kojima su bila inferiorna u odnosu na ljudske šahovske velemajstore, no današnji napredni šahovski softveri poput Stockfisha ili AlphaZero omogućuju računalima da redovito nadmašuju čak i najbolje svjetske igrače. Područje je usko povezano s razvojem umjetne inteligencije iz razloga što šah pruža idealno okruženje za istraživanje i testiranje AI sustava, s obzirom na njegovu složenost i zahtjevnost u analizi poteza i strategija. Od sredine 1960-ih, istraživači u računalnim znanostima često su šah nazivali „drosophilom“ umjetne inteligencije, analogno korištenju vinske mušice u genetskim istraživanjima. Naime, šah je poput vinske mušise, dobro poznat i dovoljno jednostavan da služi kao eksperimentalna platforma, no unatoč svojoj jednostavnosti, može se koristiti za stvaranje važnog znanja o složenijim sustavima.

Značaj razvoja šahovskih aplikacija vidljiv je na mnogim razinama. Prvenstveno, u edukaciji šah postaje pristupačan širokom krugu korisnika zahvaljujući digitalnim alatima koji nude personalizirane treninge i interaktivne lekcije. Nadalje, napredni algoritmi za analizu partija omogućuju dubinske analize koje pomažu istraživačima u otkrivanju novih strategija i teorija. Takvi alati također služe u popularizaciji šaha kroz društvenu povezanost i masovne medije, dok se tehnike korištene u razvoju šahovskih algoritama primjenjuju i izvan šaha, primjerice u optimizaciji i strateškom razmišljanju u različitim industrijama.

Od svoje četvrte godine aktivno sam se bavio šahom, nastupao na državnim, europskim i svjetskim prvenstvima te moja motivacija za odabir ove teme proizlazi upravo iz dugogodišnjeg bavljenja šahom tijekom kojega sam se susretao sa raznim šahovskim aplikacijama, poput ChessBasea, koje su me inspirirale da razvijem vlastitu aplikaciju i dublje istražim kako šahovski algoritmi funkcioniraju. Iako nisam koristio opsežnu literaturu, ključna pomoć bila mi je dokumentacija *python-chess* biblioteke, koja je omogućila učinkovitu implementaciju osnovnih funkcionalnosti aplikacije.

2. Teorijska osnova

Kako bismo lakše razumijeli rad aplikacije i šahovskih algoritama u nastavku je objašnjena teorijska osnova algoritama i koncepti koji se pojavljuju u radu.

2.1. Algoritmi

U nastavku će biti predstavljena dva ključna algoritma koja se koriste u računalnoj analizi šahovskih pozicija, specifičnije Minimax algoritam i Alpha-Beta rezanje.

2.1.1. Algoritam Minimax

Minimax algoritam jedan je od važnijih algoritama u igrama s dva igrača, gdje se igrači izmjenjuju u povlačenju poteza, a „određuje optimalnu strategiju za igrača pretpostavljajući da protivnik također igra optimalno“ (Norvig, 2010). Temelji se na ideji da oba igrača igraju optimalno na način da igrač MAX nastoji maksimizirati svoj rezultat, dok igrač MIN pokušava minimizirati rezultat igrača MAX.

Algoritam modelira igru kao stablo, gdje su čvorovi različiti potezi, a listovi krajnje pozicije igre. Na svakom potezu jedan igrač (MAX) bira najbolju moguću opciju, dok protivnik (MIN) bira potez koji će smanjiti prednost igrača MAX. Svaka pozicija u stablu se evaluira pomoću funkcije procjene (heuristike), a algoritam vraća najbolji potez za MAX igrača nakon simulacije svih mogućih poteza do određenog nivoa dubine. U osnovi, minimax je rekurzivan algoritam koji procjenjuje sve moguće ishode igre do unaprijed zadane dubine i bira najbolji potez na temelju te procjene.

„Minimax algoritam koristi osnovnu funkciju evaluacije kako bi dodijelio vrijednost svakoj poziciji. Funkcija evaluacije uzima u obzir faktore kao što su materijalna prednost, kontrola središta ploče i druge strateške komponente igre“ (Norvig, 2010). Ti se rezultati zatim vraćaju prema vrhu stabla, gdje algoritam donosi konačnu odluku o najboljem potezu.

U svijetu šaha, ovaj algoritam predstavlja temelj *chess enginea* jer omogućuje računalu da analizira sve moguće poteze i protupoteze, kako bi ono donijelo najbolju odluku u bilo kojoj poziciji. Međutim, sam po sebi minimax nije dovoljno učinkovit za složene igre poput šaha, u kojima postoji ogroman broj mogućih poteza i grana stabla koje treba istražiti. Jedan način razmišljanja o složenosti šaha je razmatranje broja mogućih poteza i varijacija u igri. Prema procjenama matematičara Claudea Shannona, postoji oko 10^120 mogućih partija šaha, što nadmašuje broj atoma u poznatom svemiru, za koje se procjenjuje da ih ima oko 10^80. "Čak i ako bi svaka čestica u svemiru igrala partiju šaha svake sekunde od početka vremena, to ne bi bilo dovoljno da se iscrpi broj mogućih šahovskih partija" (Shaykh Academy, n.d.).

Zbog ove složenosti, minimax algoritam bez dodatnih optimizacija postaje prespor jer pokušava analizirati svaku moguću poziciju. Kako bi se optimiziralo pretraživanje i ubrzao proces donošenja odluka, razvijene su nadogradnje na minimax algoritam. Jedna od najvažnijih optimizacija je Alpha-Beta rezanje, tehnika koja značajno smanjuje broj čvorova koje treba analizirati bez kompromitiranja kvalitete odluka.

2.1.2. Alpha-Beta rezanje

Alpha-Beta rezanje je ključna optimizacija minimax algoritma koja omogućava učinkovito pretraživanje stabla igre smanjenjem broja čvorova koje treba pregledati. Osnovni cilj je eliminiranje grana stabla koje ne mogu utjecati na konačnu odluku minimax algortima. To se postiže korištenjem granica, alpha (najbolji potez igrača MAX) i beta (najbolji potez igrača MIN), koje "režu" nepotrebne grane. Primjerice, ako minimax procijeni da određena grana nudi lošiju vrijednost od prethodno pronađenih poteza, ta grana se može "prerezati" i dalje se ne razmatra. Na taj način, minimax može značajno ubrzati pretraživanje, dok još uvijek donosi iste odluke kao i bez rezanja.

S obzirom da se Alpha-Beta rezanje može primijeniti na stabla bilo koje dubine, a može prerezati ne samo listove stabla, već i čitava podstabla time se eksponencijalno smanjuje broj čvorova koje algoritam mora analizirati. U teoriji, Alpha-Beta rezanje može smanjiti složenost pretraživanja na pola, a učinkovitost pretraživanja ovisi o redoslijedu poteza. Shodno tome, jasno je da ova optimizacija jako poboljšava učinkovitost rada *chess enginea* čiji je zadatak upravo određivanje najboljeg mogućeg poteza u danoj poziciji. Naime, smanjenjem broja čvorova koje je potrebno analizirati, omogućuje im se da preskoče grane stabla koje ne utječu na konačni ishod, a time *engine* može brže donijeti optimalne odluke, čime poboljšava performanse. Kombinirano s povećanom računalnom snagom, ova tehnika omogućava dublje pretrage stabla, što izravno poboljšava sposobnost igranja. Ferreria je pokazao da „povećanje dubine pretrage rezultira ELO[1] razlikom od 86.5 bodova po nivou, omogućavajući modernim *engineima* da dosegnu razine od oko 3300 ELO bodova“ (Ferreira, 2013).

2.2. Teorijski koncepti

Za implementaciju analiza pozicija potrebno je pohraniti partije u standardnom formatu za pohranu šahovskih partija. U nastavku rada predstavljeni su teorijski koncepti vezani uz zapisivanje šahovske notacije i šahovskih pozicija.

2.2.1. PGN (eng. *Portable Game Notation*)

PGN (eng. *Portable Game Notation*) je standardizirani format za zapisivanje šahovskih partija u tekstualnom obliku, ali sa „pgn“ ekstenzijom. Razvijen je kako bi omogućio jednostavnu razmjenu, pohranu i analizu šahovskih partija između različitih šahovskih softvera, baza podataka i web platformi. Može se učitati i uređivati pomoću bilo kojeg jednostavnog uređivača teksta kao što je Notepad, a može sadržavati jednu ili više šahovskih partija. Standard je široko prihvaćen u šahovskoj zajednici upravo zbog svoje fleksibilnosti i jednostavnosti korištenja.

[1] ELO je igračev šahovski rating koji je izračunao FIDE (eng. *International Chess Federation*)

2.2.1.1. Struktura PGN datoteke

PGN struktura sastoji se od dva glavna dijela:

1. Metapodaci (eng. *headers*)
    - sadrže ključne informacije o partiji, poput imena igrača, datuma, turnira, lokacije, rezultata
    - svaki metapodatak je zapisan unutar uglatih zagrada i slijedi format ključ „vrijednost“

Primjer:

```
„ [Event "State Ch."]
[Site "New York, USA"]
[Date "1910.??.??"]
[Round "?"]
[White "Capablanca"]
[Black "Jaffe"]
[Result "1-0"]
[ECO "D46"]
[Opening "Queen's Gambit Dec."]
[Annotator "Reinfeld, Fred"]
[WhiteTitle "GM"]
[WhiteCountry "Cuba"]
[BlackCountry "United States"] “
```

2. Potezi (eng. *moves*)
    - koriste SAN (eng. *Standard Algebraic Notation*) koja je široko prihvaćena i razumljiva u šahovskoj zajednici

Primjer.

```
„ 1.d4 Nf6 2.c4 e6 3.Nc3 Bb4 4.Qc2 d5 5.cxd5 Qxd5 6.Nf3 Qf5 7.Qxf5 exf5 8.a3 Be7 9.Bg5
Be6 10.e3 c6 11.Bd3 Nbd7 12.O-O h6 13.Bh4 a5 14.Rac1 O-O 15.Ne2 g5 16.Bg3 Ne4 17.Nc3 Nxc3
18.Rxc3 Nf6 19.Rcc1 Rfd8 20.Rfd1 Rac8 1/2-1/2 „
```

2.2.1.2. Dodatne značajke PGN formata

1. Komentari
    - PGN omogućava dodavanje komentara unutar partije; komentari su zapisani unutar okruglih zagrada, npr. (Ovaj potez je dobar jer osvaja središte ploče)
2. Varijante
    - podržava varijante poteza, što omogućava analizu različitih mogućih scenarija unutar partije, često se zapisuju između okruglih zagrada, npr. 1.e4 e5 (1...c5) (1...e6) 2.Nf3 Nc6 3.Bb5

3. Vrijeme poteza
    - može pohraniti vremenske oznake za svaki potez, čime se prati koliko je vremena svaki igrač potrošio na određeni potez
4. Simboli za ocjenjivanje šaha
    - potezi mogu biti ukrašeni standardnim šahovskim simbolima procjene koji se u PGN tekstu definira znakom „$“ iza kojega slijedi numerički kôd tzv. NAG (Numerički anotacijski glif)
    - npr. e4 e5 2.Nf3 f6 $2 3.Nxe5 ! fxe5 4.Qh5+ Ke7 5.Qxe5+ Kf7 6.Bc4+ $18
    - kako bi se olakšala čitljivost uvedeni su simboli koji zamjenjuju NAG (npr. umjesto $1 piše se „!“), a detaljniji prikaz simbola prikazan je u Tablici 1.

Tablica 1. Simboli za ocjenjivanje šahovskih poteza, Izvor: (Chess assessment symbols, n.d.)

NAG	**Alias**	**Representation**	**Meaning**
$1	!	!	Good move
$2	?	?	Poor move or mistake
$3	!!	!!	Very good move
$4	??	??	Very poor move or blunder
$5	!?	!?	Interesting move
$6	?!	?!	Questionable move
$7 or $8		□	Only move
$9		⊠	Worst move
$10 or $11	=	=	Equal position
$13	~ or inf	∞	Unclear position
$14	+/= or +=	⩲	White has a slight advantage
$15	=/+ or =+	⩱	Black has a slight advantage
$16	+/-	±	White has a moderate advantage
$17	-/+	∓	Black has a moderate advantage
$18	+-	+−	White has a decisive advantage
$19	-+	−+	Black has a decisive advantage
$20		+−−	White has a crushing advantage

NAG	Alias	Representation	Meaning
$21		−−+	Black has a crushing advantage
$22		⊙	Zugzwang
$32		⟳	Development advantage
$36		↑	Initiative
$40		→	Attack
$44		□	Compensation
$132		⇆	Counterplay
$138		⊕	Zeitnot
$140		∆	With the idea…
$141		∇	Aimed against…
$142		⌓	Better is…
$143		≤	Worse is…
$145		RR	Editorial comment
$146		N	Novelty

2.2.1.3. Prednosti korištenja PGN formata

1. Standarizacija
    - PGN je prihvaćen kao standard u šahovskoj zajednici čime se osigurava interoperabilnost između različitih šahovskih softvera i platformi
2. Jednostavnost
    - lako je čitljiv i razumljiv ljudima i računalnim programima
3. Fleksibilnost
    - podržava dodatne informacije poput komentara, varijanti i vremenskih oznaka, što olakšava detaljnu analizu partija
4. Kompatibilnost
    - gotovo svi šahovski programi i baze podataka podržavaju PGN, što omogućava jednostavan uvoz i izvoz partija

Primjer jedne partije zapisane u PGN file-u koji se koristi u aplikaciji:

```
„ [Event "FIDE Candidates 2024"]
[Site "Toronto CAN"]
[Date "2024.04.09"]
[Round "5.1"]
[White "Praggnanandhaa,R"]
[Black "Nepomniachtchi,I"]
[Result "1/2-1/2"]
[WhiteTitle "GM"]
[BlackTitle "GM"]
[WhiteElo "2747"]
[BlackElo "2758"]
[ECO "C42"]
[Opening "Petrov"]
[Variation "classical attack, Marshall variation"]
[WhiteFideId "25059530"]
[BlackFideId "4168119"]
[EventDate "2024.04.04"]

1. e4 e5 2. Nf3 Nf6 3. Nxe5 d6 4. Nf3 Nxe4 5. d4 d5 6. Bd3 Bd6 7. O-O O-O 8. c4
c6 9. Nc3 Nxc3 10. bxc3 dxc4 11. Bxc4 Bf5 12. Bg5 Qa5 13. Nh4 Be6 14. Bxe6 Qxg5
15. Nf3 Qa5 16. Bh3 Qxc3 17. Rb1 b6 18. Rb3 Qa5 19. d5 cxd5 20. Ng5 h6 21. Nxf7
Kxf7 22. Rd3 Nd7 23. Rxd5 Nc5 24. Rxd6 Kg8 25. Qd5+ Kh8 26. Bf5 Nb7 27. Qxa5
Nxa5 28. g4 Nc4 29. Rd5 Rae8 30. h3 Ne5 31. Kg2 g6 32. Bc2 g5 33. Bf5 Re7 34.
Rd6 Kg7 35. Re1 Rf6 36. Rd5 Ng6 37. Rxe7+ Nxe7 38. Rd7 Kf8 39. Be4 a5 40. Kg3
Ng6 41. Bxg6 Rxg6 42. h4 Rc6 43. hxg5 hxg5 44. Rb7 a4 1/2-1/2 “
```

2.2.2. SAN (eng. *Standard Algebraic Notation*)

SAN (eng. *Standard Algebraic Notation*) je najrašireniji način označavanja šahovskih poteza. Funkcionira na način da svaki potez zapisuje kombinacijom slova i brojeva, koji označavaju figure i polja na ploči. Naime, šahovska ploča podijeljena je na osam redaka koji se označavaju brojevima od 1 do 8 i osam stupaca koji se označavaju slovima od „a“ do „h“, čime se stvara mreža od 64 polja. Vizualni prikaz šahovske mreže nalazi se na Slici 1. Figure su označene prvim slovom ili prvim slobodnim slovom iz engleskih naziva, dok se pješaci označavaju implicitno, bez slova. U Tablici 2. prikazane su oznake figura.

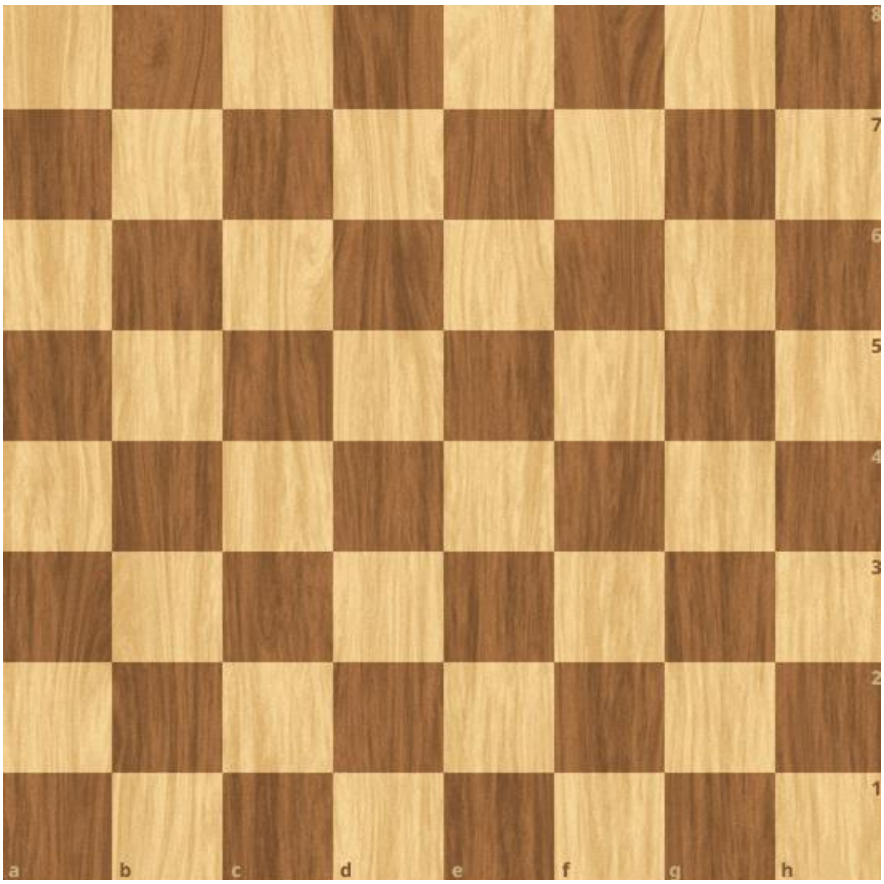
Primjer:

Potez pješakom na polje e4: e4

Potez skakačem na f3: Nf3

Mala rokada[2]: O-O

Velika rokada: O-O-O



Slika 1. Prikaz oznaka redova i stupaca na šahovskoj ploči

Tablica 2. Prikaz SAN oznaka šahovskih figura

Figura	Oznaka
Kralj (eng. *King*)	K
Kraljica (eng. *Queen*)	Q
Top (eng. *Rook*)	R
Lovac (eng. *Bishop*)	B
Skakač (eng. *Knight*)	N

[2] **Rokada** (rohada ili rošada) je specifičan **šahovski** potez. Svrha mu je bolja zaštita kralja. To je jedini potez prilikom kojega se pomiču dvije figure, a računa se kao jedan potez. Također je jedini potez u kome kralj može prijeći dva polja.

U slučajevima gdje više figura može ići na isto polje, dodaje se dodatna informacija, kao što je stupac ili redak odakle dolazi figura (npr. Nbd7 što znači da skakač sa „b“ stupca ide na polje d7).

U situacijama kada jedna figura „pojede“ (eng. *capture*) protivniku figuru (stane na isto polje i protivnikova figura se uklanja s tog polja i iz igre) u takav potez se umetne simbol „x“ (npr. exd5 predstavlja potez kada je pješak iz „e“ stupca „pojeo“ neku protivnikovu figuru na d5).

Dodatan slučaj je situacija promocije pješaka, tj. situacija kada pješak dolazi na prvi red protivnika (bijeli pješak na 8. red ili crni pješak na 1. red) te se tada taj pješak zamjenjuje figurom po želji igrača. Shodno njegovoj odluci potez se zapisuje npr. d8Q ukoliko je odlučio promovirati pješaka u kraljicu.

Dodatke kratice su:

1. + → dodavanje na kraj poteza što znači da je tim potezom dan šah, tj. napadnut protivnički kralj npr. Qd5+
2. ++ ili # → šah mat

2.2.3. FEN (eng. *Forsyth-Edwards Notation*)

FEN (eng. *Forsyth-Edwards Notation*) je šahovska notacija koja se koristi za zapisivanje trenutnog stanja šahovske ploče. Razvijen iz Forsythove notacije, pruža jednostavan i kompaktan način za opisivanje šahovskih pozicija.

2.2.3.1. Struktura FEN zapisa

Struktura FEN zapisa sastoji se od 6 dijelova koji opisuju:

1. Poziciju figura
    - pohranjuje svih 8 redaka zapisanih s vrha ploče (8. redak) prema dnu (1. redak), a brojevi predstavljaju prazna polja
    - velika slova u zapisu predstavljaju bijele figure, dok mala slova predstavljaju crne figure
    - primjer dijela vezanog uz poziciju figura FEN zapisa početne pozicije:

      „ rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR „
2. Igrača na potezu
    - sprema oznaku w/b ovisno o tome je li u toj poziciji na potezu bijeli ili crni
3. Prava na rokadu
    - ovdje bilježi mogu li bijeli ili crni napraviti rokadu pomoću oznaka „K“ i „Q“ (bijeli) odnosno „k“ i „q“ (crni)
    - slovo k predstavlja rokadu na kraljevu stranu (malu rokadu; O-O), a slovo q predstavlja daminu stranu (veliku rokadu; O-O-O)

4. En passant[3]
    - bilježi polje na kojem se može dogoditi en passant potez (npr. „e3“), a ukoliko taj potez nije moguć bilježi „-“
5. Broj polupoteza
    - pohranjuje broj polupoteza od zadnjeg poteza koji nije bio potez pješakom ili uzimanje figure (korisno zbog šahovskog pravila 50 poteza)
6. Ukupan broj poteza
    - zapisuje konačan broj poteza od početka partije do te pozicije (prvi potez bijeloga je potez 1)

Primjer cjelokupnog FEN zapisa početne pozicije:

„ rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1 „

FEN je izuzetno koristan za šahovske programe jer omogućava brzo postavljanje određene pozicije na ploči i jednostavno dijeljenje ili pohranjivanje pozicija. Svaka pozicija može biti točno opisana jednim retkom teksta, što omogućuje aplikacijama da brzo analiziraju trenutnu situaciju i generiraju moguće poteze. Jednostavan je za generiranje, parsiranje i obradu od strane šahovskih programa te olakšava dijeljenje specifičnih pozicija za analizu.

[3] **En passant** (fra. *u prolazu*) je potez u šahu kojim uzimamo („jedemo“) protivničkog pješaka koji je napredovao dva polja sa svog početnog mjesta i tako zauzeo mjesto odmah do našeg pješaka. Naš pješak ide dijagonalno iza protivničkog pješaka kojeg uzimamo.

3. Softversko rješenje

U ovom poglavlju cilj je obrazložiti odabrane tehnologije i predočiti opis cjelokupne strukture aplikacije, odnosno način na koji funkcioniraju grafičko sučelje (GUI), šahovska logika i baza podataka.

3.1. Arhitektura aplikacije

Aplikacija za šahovsku analizu razvija se koristeći Python, jedan od najpopularnijih programskih jezika danas. Python je odabran kao temeljni programski jezik zbog nekoliko ključnih razloga. Jedan od razloga svakako je jednostavnost i čitljivost, što značajno ubrzava razvoj aplikacija. Još važnije, Python nudi široku podršku u obliku opsežne dokumentacije i aktivne zajednice, čime omogućava brzo rješavanje problema i olakšava pronalazak resursa u usporedbi s drugim jezicima.

Uz Python kao temeljni programski jezik, u razvoju će se koristiti *Python-chess* biblioteka, zbog njegove fleksibilnosti i jednostavnosti u upravljanju šahovskom logikom. S obzirom da će biti potrebno pohraniti podatke o partijama u određenu bazu podataka, zbog svoje jednostavne implementacije i neovisnosti o vanjskim serverima koristit će se SQLite.

Za grafičko sučelje koristi se Tkinter, koji je integriran s Pythonom, nudi brzo i jednostavno kreiranje sučelja te je dovoljno moćan za aplikacije koje ne zahtijevaju složene vizualne elemente.

3.1.1. Python-chess

Python-chess (python-chess: a chess library for Python, 2024) je biblioteka koja omogućuje jednostavno upravljanje šahovskom logikom u Pythonu. Osim osnovnih operacija poput stvaranja ploče i poteza, biblioteka omogućava provjeru statusa igre, evaluaciju poteza i uključuje podršku za standardne formate šahovskih partija, kao što su PGN i FEN. Uz to, podržava interakciju sa *chess engine-ima*, poput Stockfisha, omogućujući generiranje poteza, analiziranje pozicija i pružanje povratnih informacija o najboljim potezima, a pruža i mogućnosti vizualizacije ploče putem SVG grafike.

3.1.2. SQLite

Odabir SQLite baze podataka temelji se na njezinoj jednostavnosti i činjenici da ne zahtijeva instalaciju dodatnog softvera, što je idealno za aplikacije koje trebaju jednostavno pohranjivanje podataka. SQLite uspješno će pohraniti više od 19 000 partija, uz dodatne partije koje korisnici generiraju tijekom igranja ili analiza. Iako SQLite zadovoljava potrebe ove aplikacije, u budućnosti bi mogla biti potrebna nadogradnja na baze poput MySQL ili MariaDB zbog povećanja broja partija koje u sličnim programima znaju dosezati i brojke od nekoliko desetaka milijuna.

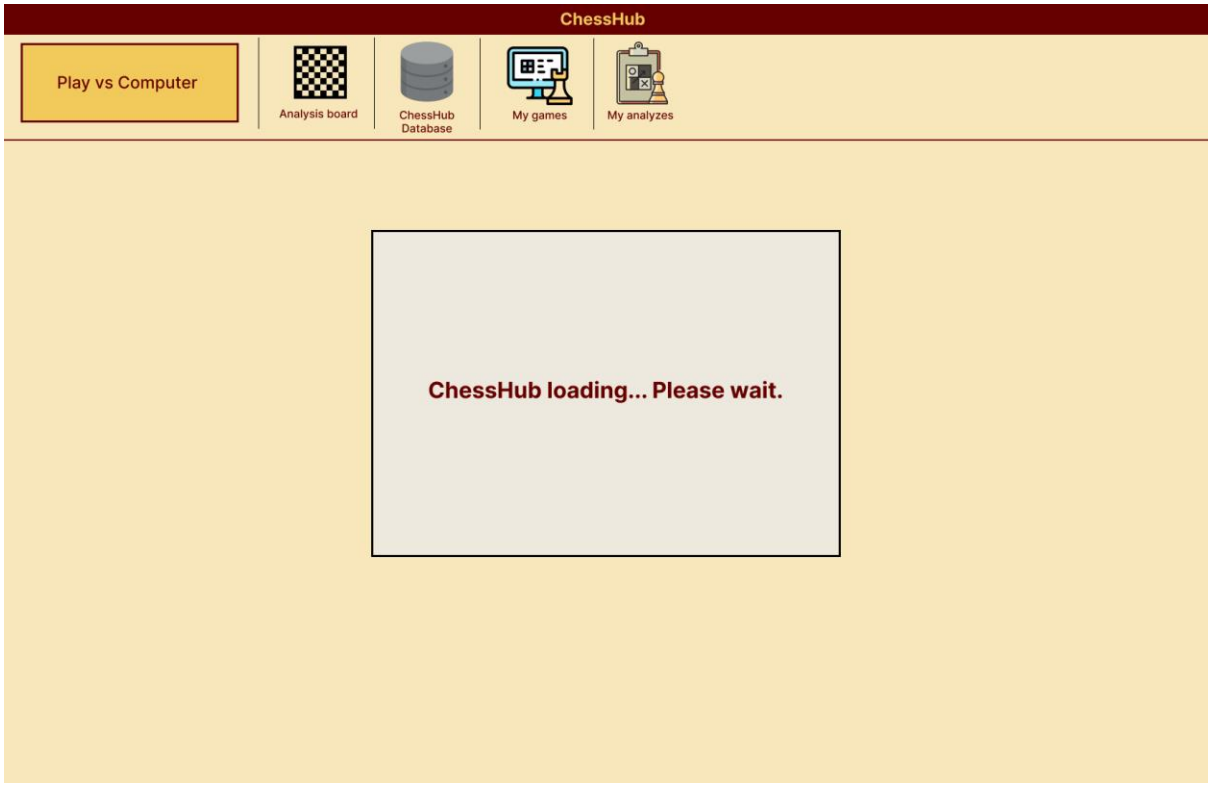
3.1.3. Tkinter

Za izradu GUI sučelja odabran je Tkinter zbog njegove integriranosti s Pythonom i jednostavnosti korištenja. Iako je jednostavan, Tkinter pruža dovoljno funkcionalnosti za izradu preglednog i intuitivnog sučelja, što je dovoljno s obzirom na to da je glavni fokus

projekta na šahovskim algoritmima i analizi pozicija. U budućnosti, za složenije funkcionalnosti ili poboljšanje korisničkog sučelja moglo bi se razmotriti korištenje naprednijih alata poput PyQt ili Kivy.

3.2. Funkcionalnosti aplikacije

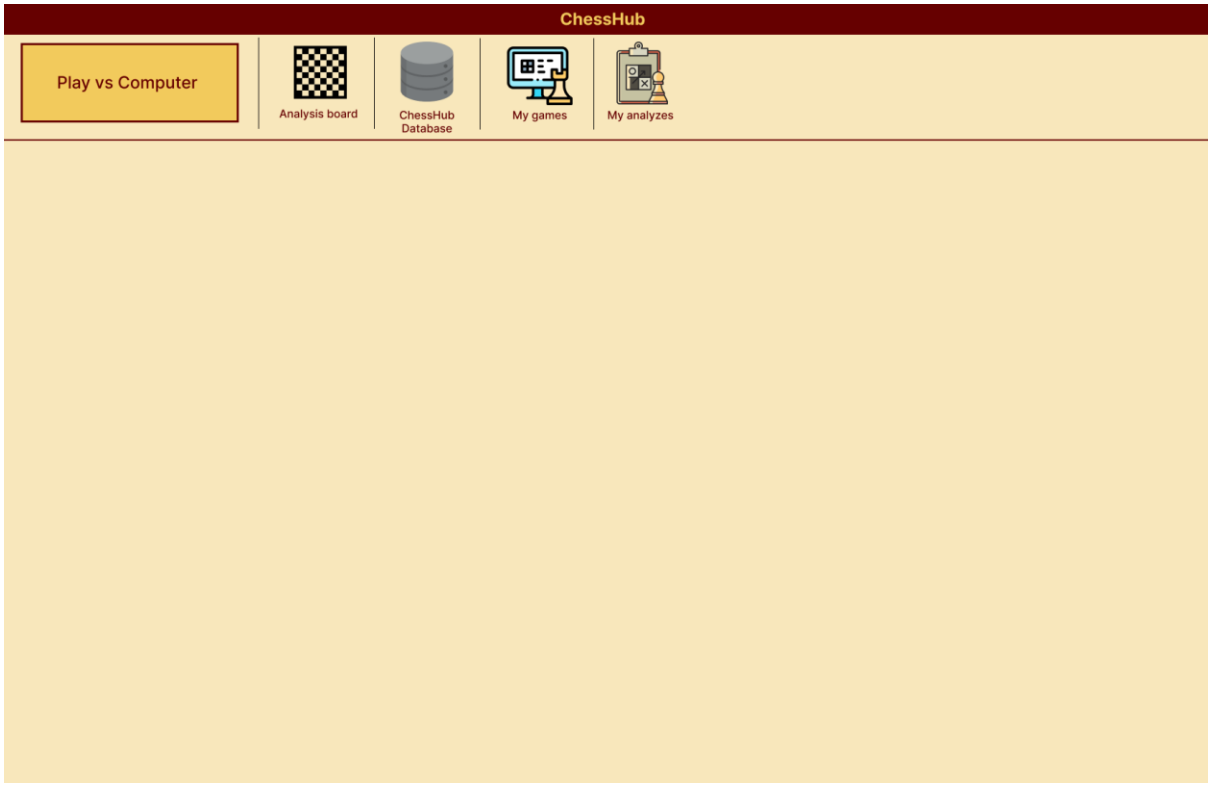
Prilikom pokretanja aplikacije, korisnicima se prikazuje početni zaslon (eng. *home screen*), ali i ekran za učitavanje (eng. *loading screen*) (Slika 2.) koji onemogućava korištenje aplikacije. Naime, ekran za učitavanje prikazuje se dok se ne završi logika aplikacije u pozadini koja obrađuje i pohranjuje šahovske partije iz PGN datoteke u unaprijed postavljenu SQLite bazu podataka. Pohranjene partije korisnicima će biti dostupne za pregled unutar "ChessHub Database" opcije.



Slika 2. Prikaz ekrana za učitavanje kao *mockup* u alatu Figma

Nakon učitavanja, korisnici se prebacuju na početni zaslon (Slika 3.), koji služi kao središte aplikacije. Na početnom zaslonu korisnicima se nudi nekoliko opcija za interakciju s aplikacijom:

1. Play vs Computer
2. Analysis board
3. ChessHub Database
4. My games
5. My analyzes



Slika 3. Prikaz početnog zaslona kao *mockup* u alatu Figma

3.2.1. Play vs Computer

Play vs Computer je funkcionalnost aplikacije koja omogućuje korisnicima da igraju šah protiv računala, koristeći ugrađeni *chess engine* (Stockfish). Ova opcija nudi razne postavke koje korisnici mogu prilagoditi prema vlastitim željama, uključujući odabir boje figura, razine snage računala i tempa igre. *Chess engine* je prilagođen na 10 različitih razina težine, od najlakše do najteže, kako bi se zadovoljile potrebe igrača različitih vještina.

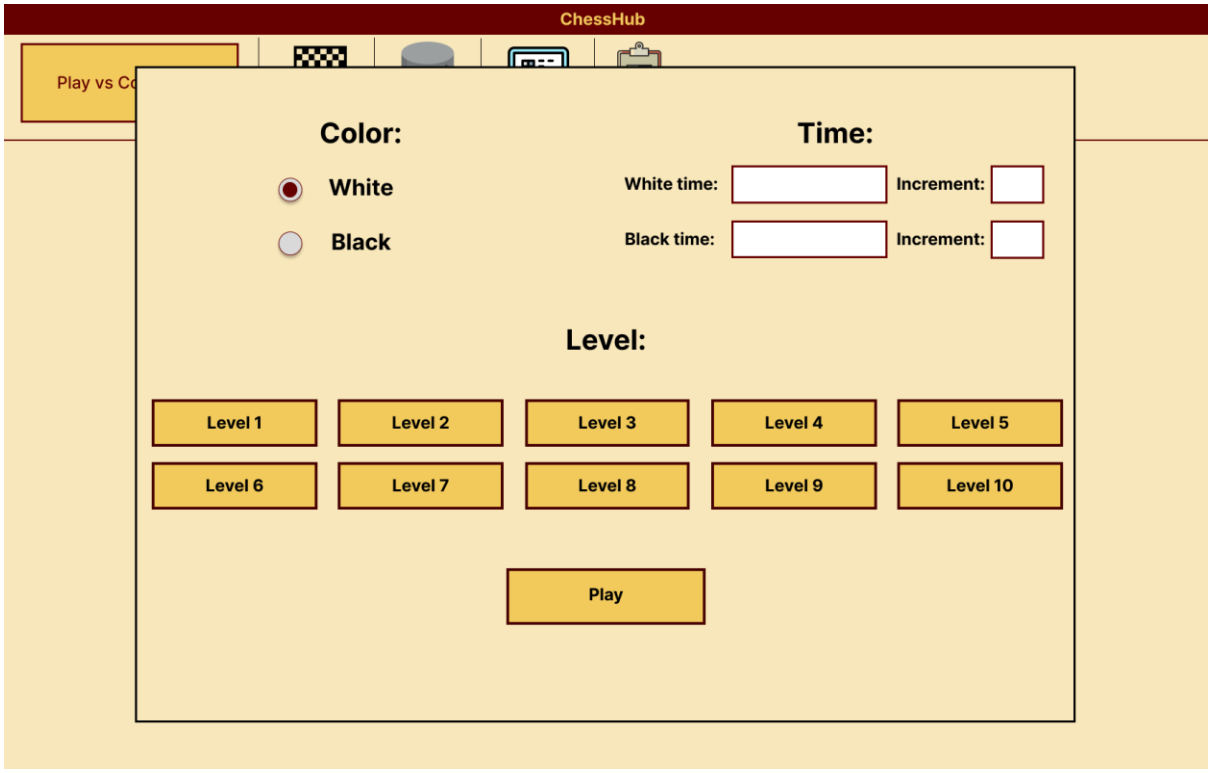
3.2.1.1. Ključne značajke Play vs Computer:

1. Interaktivna šahovska ploča
    - korisnici imaju na raspolaganju interaktivnu ploču na kojoj mogu povlačiti poteze koristeći miš
    - ploča se automatski prilagođava odabranoj boji figura
2. Ugrađeni *chess engine*
    - generiranje poteza računala na jačini ovisnoj o odabranoj razini snage
3. Praćenje vremena
    - korisnici definiraju vremenske postavke partije, uključujući ukupno vrijeme za igru i dodatak po potezu
    - šahovski sat prati vrijeme oba igrača
    - vrijeme se može prilagoditi ovisno o preferencijama igrača, npr. "5 minuta po igraču" ili "15+10" (15 minuta plus 10 sekundi dodatka po potezu)

4. Automatski završetak partije
    - kada igra dođe do kraja (npr. mat, pat, istek vremena ili remi zbog nedostatka materijala), partija se automatski završava, a korisniku se prikazuje rezultat igre
    - korisnik može tijekom igre odustati ili predati partiju, nakon čega će igra biti prekinuta i spremljena
5. Spremanje partija
    - na kraju svake partije korisnici mogu spremiti partiju u PGN
    - korisnik ima mogućnost pregledavanja odigranih i spremljenih partija unutar aplikacije

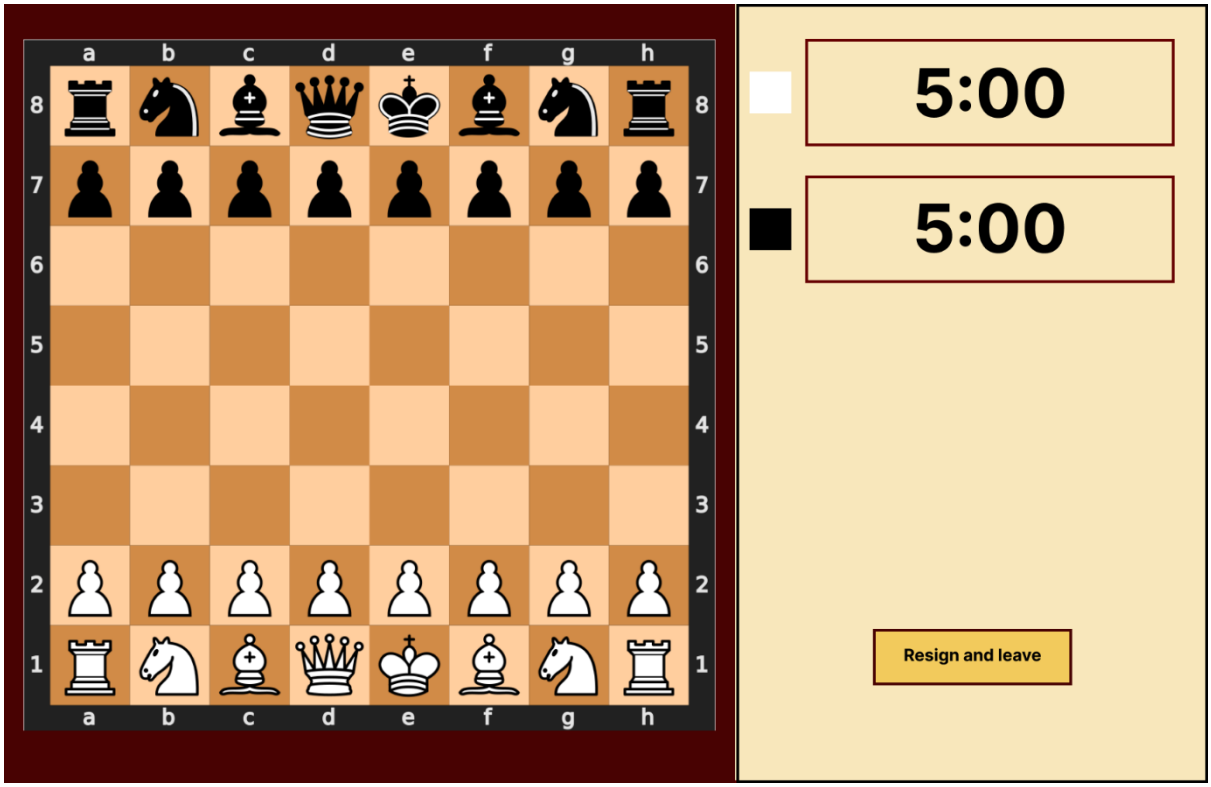
3.2.1.2. Interakcija korisnika u igri protiv računala

Ukoliko korisnik odluči igrati protiv računala, nakon klika na gumb „Play vs Computer“ otvorit će mu se izbornik u kojemu mora odabrati: boju figura kojom želi igrati, upisati tempo partije (s dodatkom po potezu ili bez), te odabrati razinu koja definira snagu računala (*chess enginea*). Prikaz ovih funkcionalnosti vidljiv je na Slici 4.



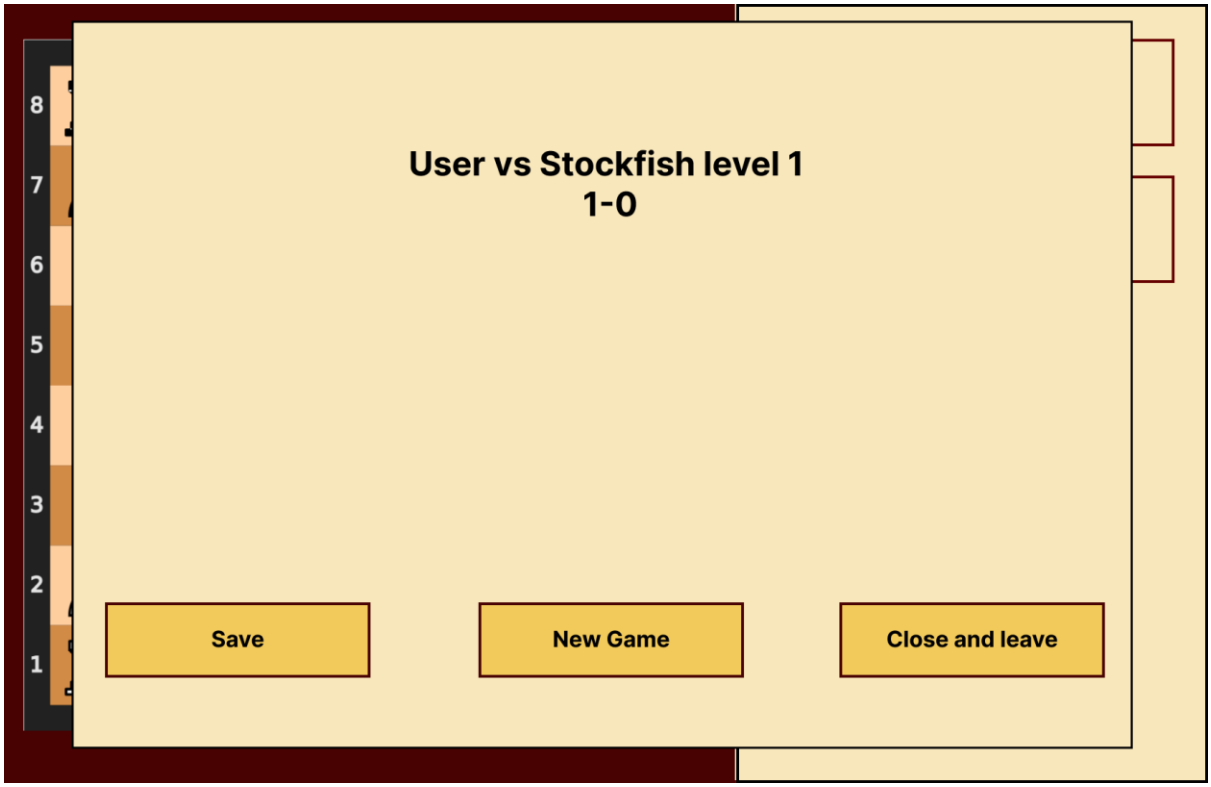
Slika 4. Prikaz prozora za odabir postavka igre protiv računala kao *mockup* u alatu Figma

Nakon odabira željene boje figura, razine računala i upisa tempa za svakog igrača korisnik klikom na gumb „Play“ prelazi u prozor namijenjen za igru koji je prikazan na Slici 5. U novi prozor proslijeđeni su podaci iz izbornika te je na temelju njih ploča okrenuta prema korisniku ovisno o tome je li odabrao bijele ili crne figure, a jačina računala prilagođena je odabranoj razini.



Slika 5. Prikaz ekrana za igranje protiv računala kao *mockup* u alatu Figma

Nakon što partija završi na neki od mogućih načina (šah mat, istek vremena jednog od igrača i sl.) prikazuje se novi prozor u kojemu je vidljiv rezultat partije te tri gumba sa opcijama „Save“, „New Game“ i „Close and leave“ (Slika 6.).



Slika 6. Prikaz rezultata igre protiv računala kao *mockup* u alatu Figma

Klikom na gumb „Save“ aplikacija sprema odigranu partiju u PGN formatu sa podacima o tome koju boju figura je imao korisnik, protiv koje razine računala je igrao, koliko vremena je ostalo na satu na kraju igre računalu, a koliko korisniku, rezultat partije i notaciju.

Gumb „New Game“ ponovno otvara prozor za odabir boje figura, razine računala i unos vremena prikazanog na slici 4, dok gumb „Close and leave“ zatvara otvorene prozore te ponovno otvara početni zaslon sa ažuriranom bazom podataka „my_games“ kako bi se mogla pregledati posljednje odigrana partija.

3.2.2. Ploča za analizu (eng. *Analysis board*)

Ploča za analizu predstavlja funkcionalnost unutar aplikacije koja korisnicima omogućuje detaljno analiziranje šahovskih partija. Na ovoj ploči korisnici mogu pregledavati, ponavljati i analizirati odigrane partije, testirati različite šahovske varijante i opcije bez utjecaja na rezultat igre.

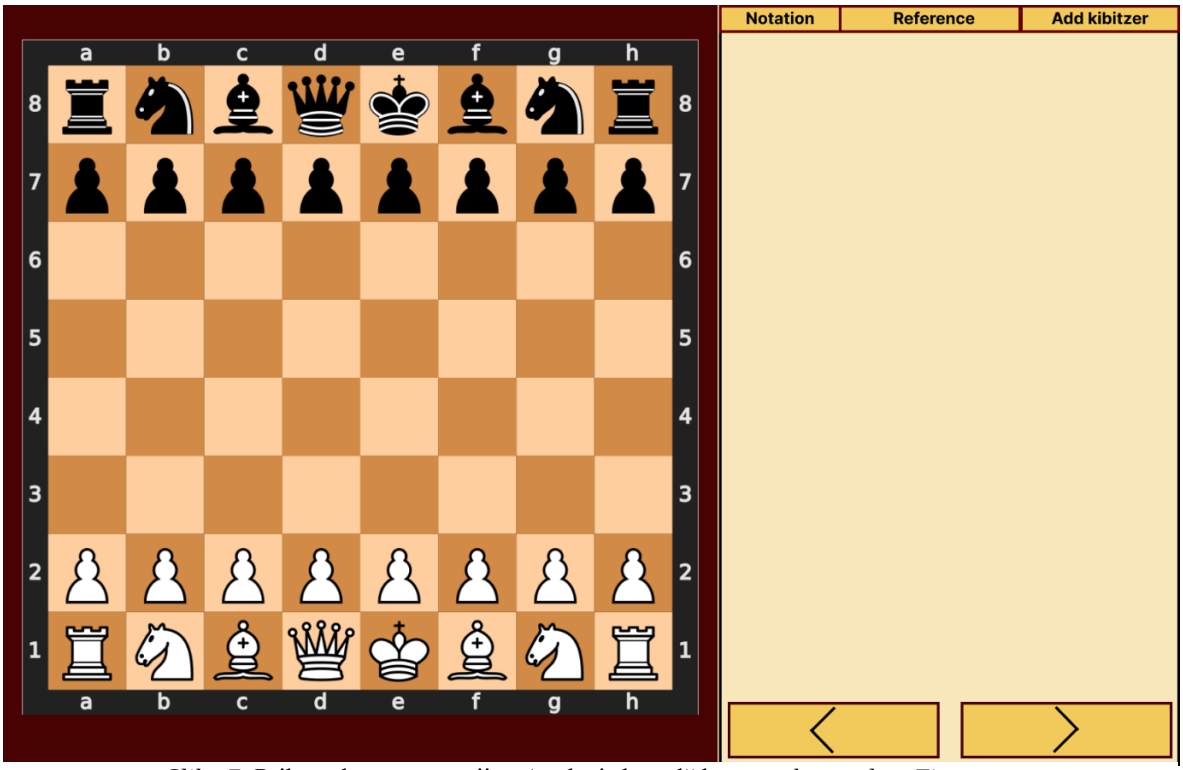
3.2.2.1. Ključne značajke ploče za analizu

1. Interaktivna šahovska ploča
    - korisniku je omogućeno slobodno povlačenje poteza za obje strane kako bi analizirao različite varijante
    - omogućene su opcije povratka na prethodni potez i ponovno odigravanje poteza unaprijed

2. Reference
    - ovisno o poziciji na šahovskoj ploči opcija Reference pretražuje bazu podataka te prikazuje partije u kojima se pojavila ista pozicija
    - za svaku partiju prikazuju se popratne informacije, uključujući imena igrača, njihov ELO rating, rezultat partije, godinu kada se partija odigrala, naziv turnira na kojemu se igrala i sl.
    - korisnicima se prikazuju svi potezi koji su igrani u partijama iz baze podataka te popis partija gdje se pozicija pojavila
3. Opcija *Add kibitzer*
    - ova opcija korisnicima pruža podršku *chess enginea* Stockfish[4] 17, koji u stvarnom vremenu prikazuje najbolje poteze za trenutnu poziciju
    - korisnici mogu vidjeti prvih nekoliko najboljih poteza, analizirati dubinu pozicije i procijeniti mogućnosti koje *engine* preporučuje

3.2.2.2. Interakcija korisnika s Analysis boardom

Nakon što korisnik u početnom zaslonu odabere opciju „Analysis board“, otvara mu se novi prozor koji prikazuje šahovsku ploču spremnu za analizu (Slika 7.). Novi prozor sastoji se šahovske ploče i bočnog panela s dodatnim opcijama „Notation“, „Reference“ i „Add kibitzer“.



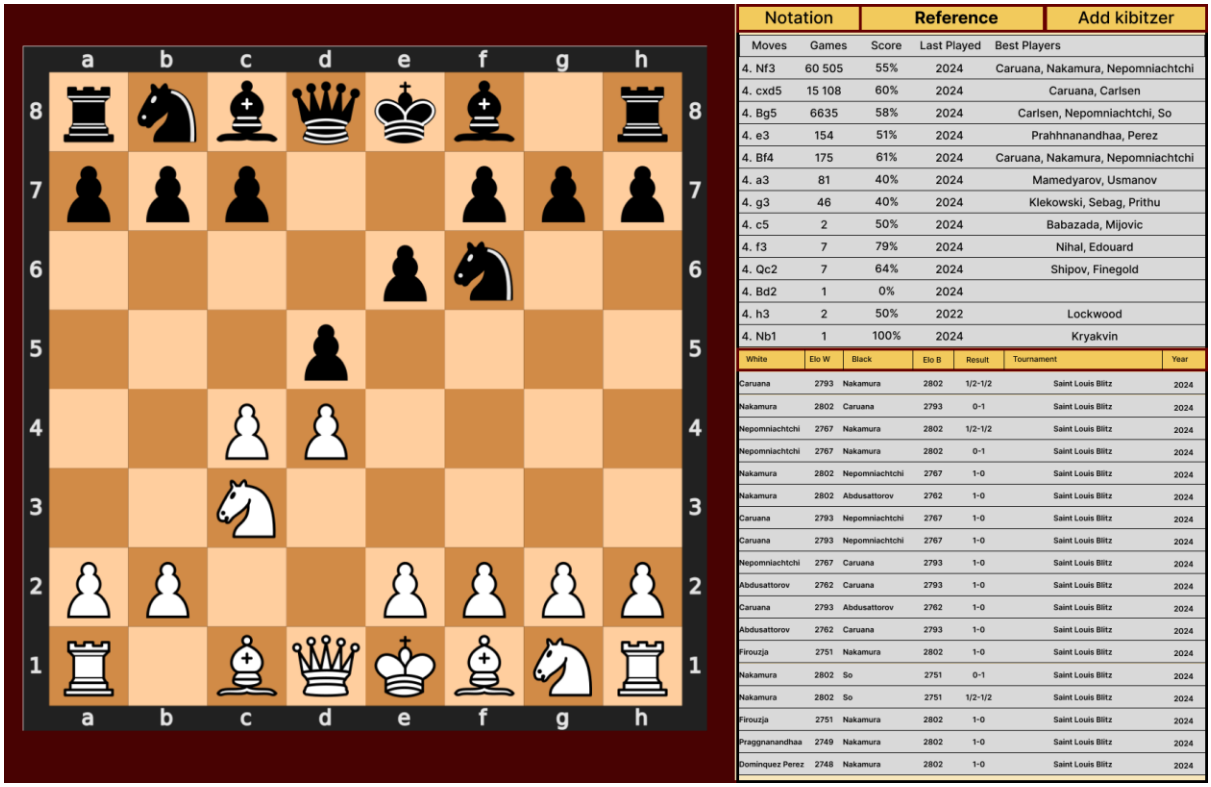
Slika 7. Prikaz ekrana za opciju „Analysis board“ kao *mockup* u alatu Figma

[4] Stockfish je besplatni *chess engine* otvorenog koda, dostupan za razne platforme

Nakon što korisnik odigra potez, u bočnom panelu bit će prikazana ažurirana verzija šahovske notacije sa istaknutim posljednje odigranim potezom, kao što se vidi na Slici 8.

Slika 8. Prikaz ekrana za opciju „Analysis board“ kao *mockup* u alatu Figma - Notation

Nadalje, korisnik može koristiti opciju Reference koja će u bočnom panelu ukloniti do sada prikazanu notaciju i prikazat će dohvaćene informacije iz baze podataka na temelju trenutnog stanja na ploči. Takva situacija vidljiva je na Slici 9.



Slika 9. Prikaz ekrana za opciju „Analysis board“ kao *mockup* u alatu Figma - Reference

Klikom na Add kibitzer opciju, uklonit će se podaci prikazani u bočnom panelu i prikazati oni vezani uz najbolje poteze generirane od strane *chess enginea* (Slika 10.).

Slika 10. Prikaz ekrana za opciju „Analysis board“ kao *mockup* u alatu Figma – Add kibitzer

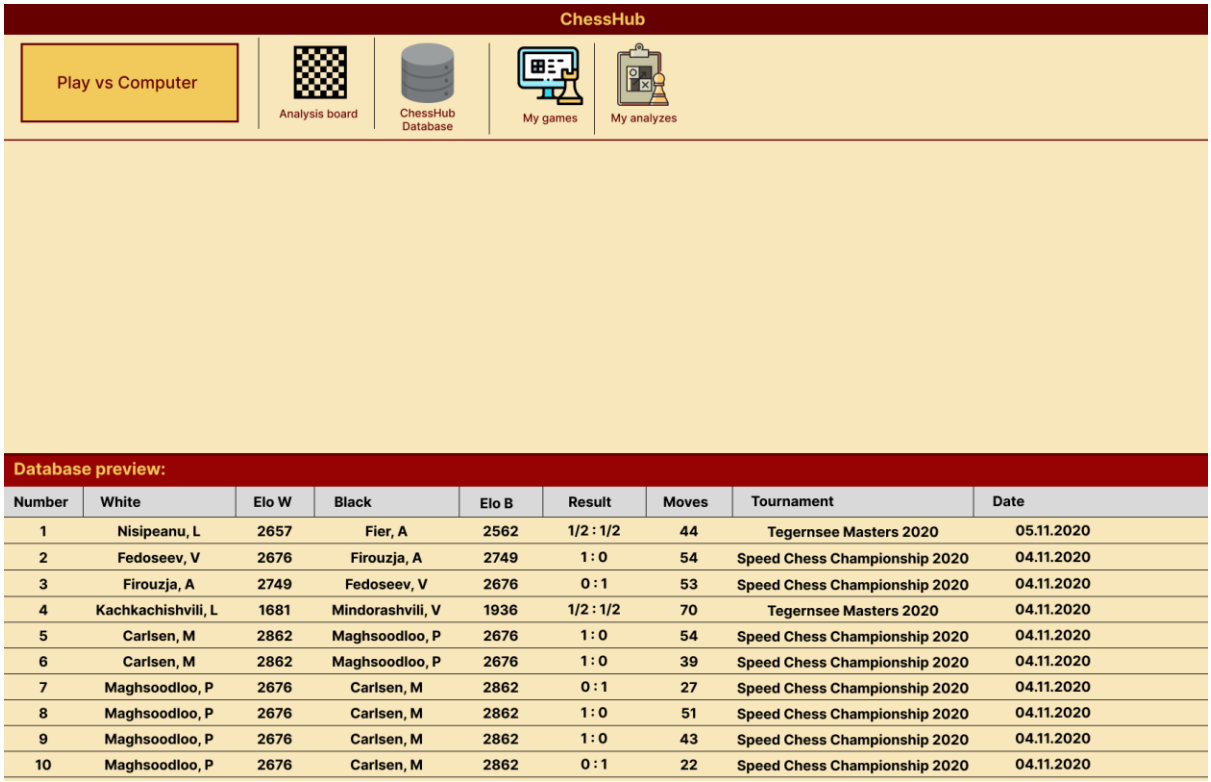
Kada je korisnik završio sa analizom i želi zatvoriti Analysis board prozor, pojavit će se prozor u kojemu može ispuniti podatke kako bi se analiza pohranila u bazi podataka i kako bi kasnije mogao nastaviti ili pregledati analizu. Prozor s neispunjenim podacima prikazan je na Slici 11.

Slika 11. Prikaz ekrana za spremanje analize kao *mockup* u alatu Figma

3.2.3. ChessHub Database

ChessHub Database je funkcionalnost aplikacije koja korisnicima omogućuje pristup i pregledavanje šahovskih partija pohranjenih u lokalnoj bazi podataka. Ova funkcionalnost povezana je sa pokretanjem aplikacije, kada se partije iz PGN datoteke automatski učitavaju i pohranjuju u bazu podataka radi lakšeg pretraživanja i analize.

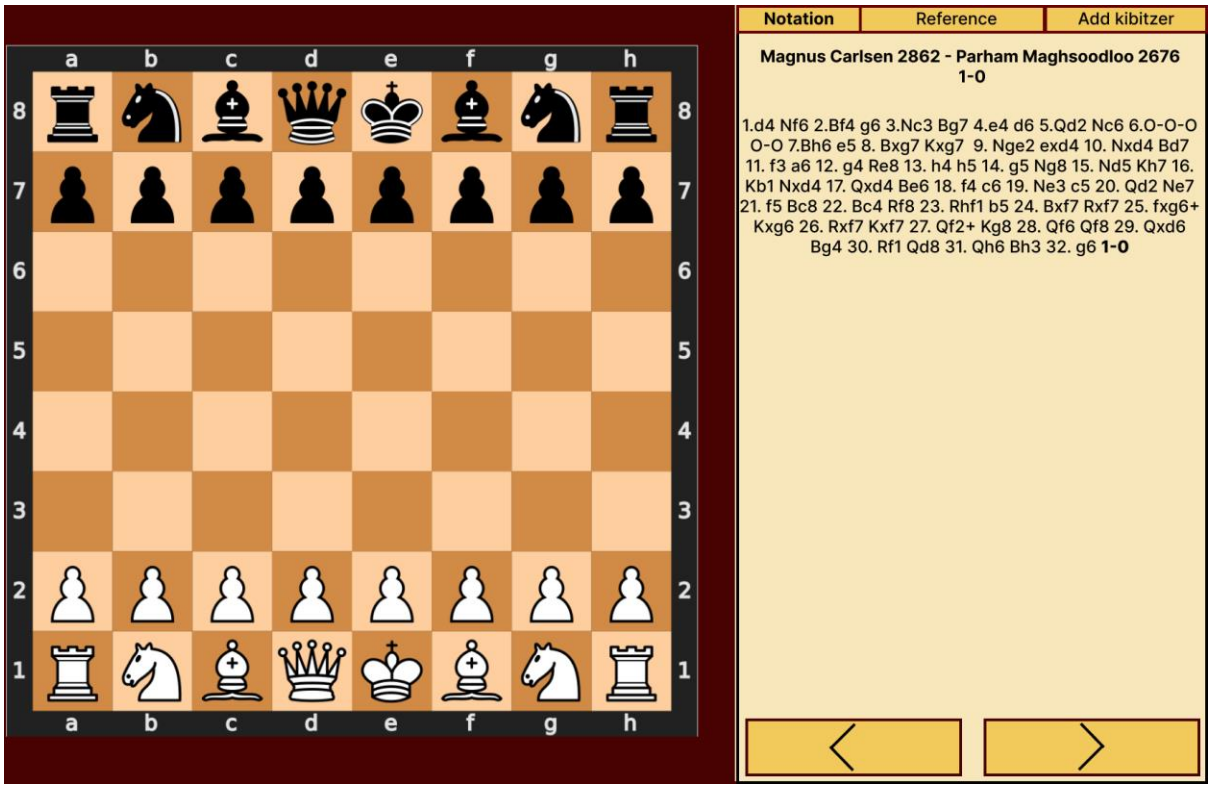
Naime, kada korisnik klikne na ChessHub Database u početnom zaslonu prikazuje mu se popis partija kroz koji može listati partije iz baze (Slika 12.).



Number	White	Elo W	Black	Elo B	Result	Moves	Tournament	Date
1	Nisipeanu, L	2657	Fier, A	2562	1/2 : 1/2	44	Tegernsee Masters 2020	05.11.2020
2	Fedoseev, V	2676	Firouzja, A	2749	1 : 0	54	Speed Chess Championship 2020	04.11.2020
3	Firouzja, A	2749	Fedoseev, V	2676	0 : 1	53	Speed Chess Championship 2020	04.11.2020
4	Kachkachishvili, L	1681	Mindorashvili, V	1936	1/2 : 1/2	70	Tegernsee Masters 2020	04.11.2020
5	Carlsen, M	2862	Maghsoodloo, P	2676	1 : 0	54	Speed Chess Championship 2020	04.11.2020
6	Carlsen, M	2862	Maghsoodloo, P	2676	1 : 0	39	Speed Chess Championship 2020	04.11.2020
7	Maghsoodloo, P	2676	Carlsen, M	2862	0 : 1	27	Speed Chess Championship 2020	04.11.2020
8	Maghsoodloo, P	2676	Carlsen, M	2862	1 : 0	51	Speed Chess Championship 2020	04.11.2020
9	Maghsoodloo, P	2676	Carlsen, M	2862	1 : 0	43	Speed Chess Championship 2020	04.11.2020
10	Maghsoodloo, P	2676	Carlsen, M	2862	0 : 1	22	Speed Chess Championship 2020	04.11.2020

Slika 12. Prikaz početnog zaslona s kliknutom opcijom „ChessHub Database“ kao *mockup* u alatu Figma

Nakon što korisnik pronađe partiju koja ga zanima, dvostrukim klikom u izborniku „Database preview“ može otvoriti partiju kako bi je pregledao. Takva situacija vidljiva je na Slici 13.

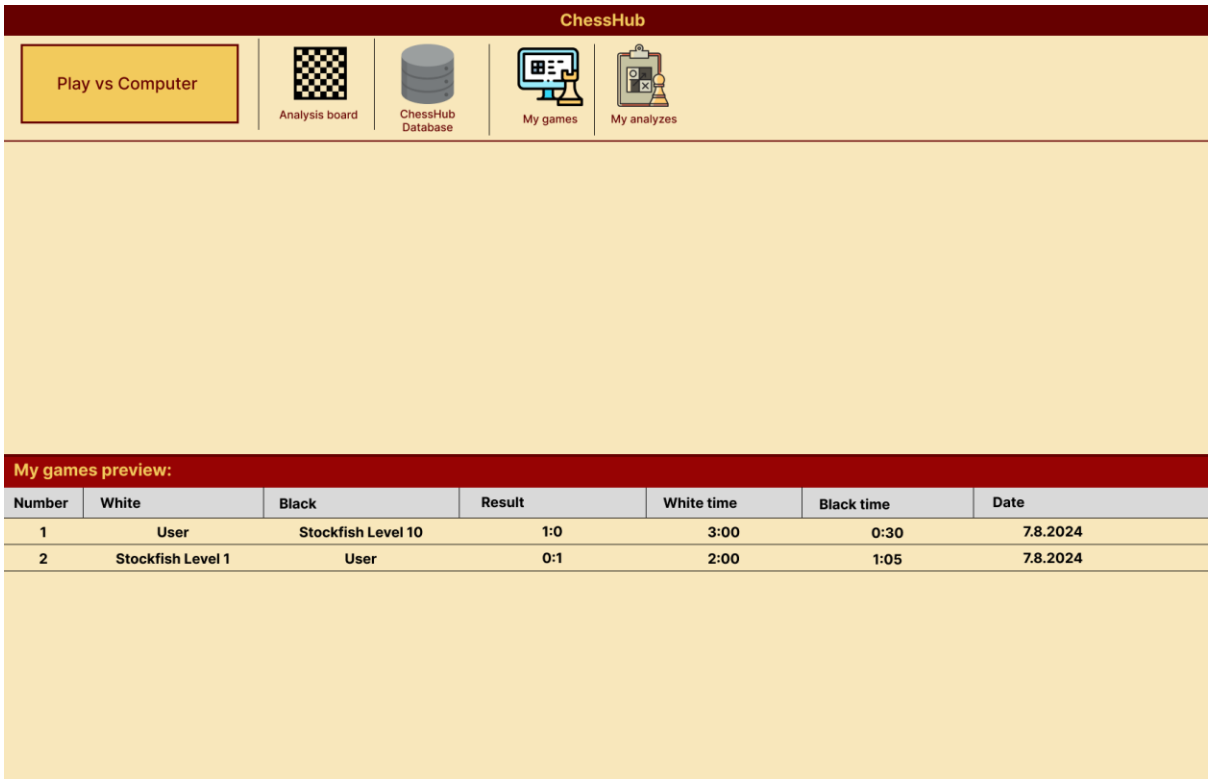


Slika 13. Prikaz „Analysis board“ ekrana za specifičnu partiju kao *mockup* u alatu Figma

3.2.4. Funkcionalnost *My games*

My games funkcionalnost vrlo je slična funkcionalnosti koja se naziva ChessHub Database, ali s jednom ključnom razlikom – ovdje se dohvaćaju i prikazuju isključivo partije koje je korisnik odigrao protiv računala unutar aplikacije. Te partije su pohranjene u posebnu tablicu baze podataka namijenjenu samo za internu pohranu partija odigranih u sklopu aplikacije.

Na Slici 14. vidljivo je da se nakon klika na gumb/opciju My games u početnom zaslonu prikazuje popis partija protiv računala te se nakon odabira željene partije otvara prozor za prikaz i analizu navedene partije (Slika 15.).



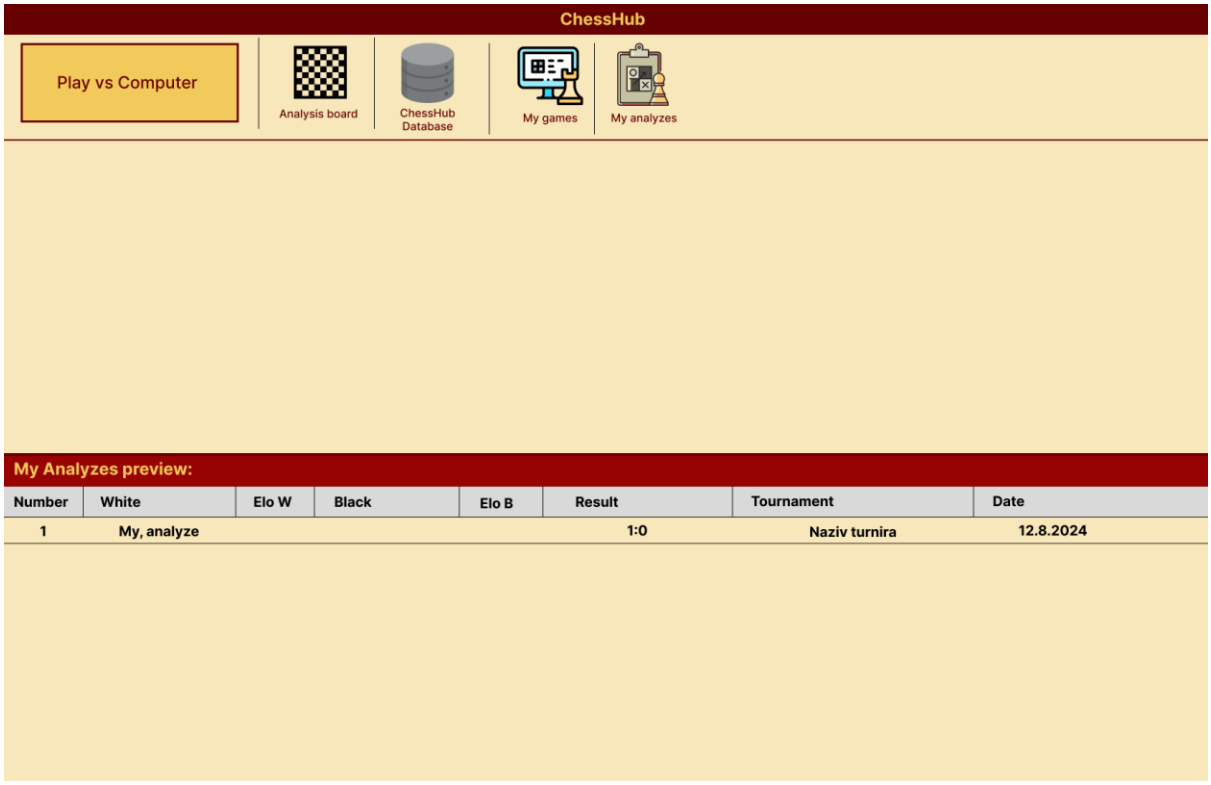
Slika 14. Prikaz početnog zaslona s kliknutom opcijom „My games“ kao *mockup* u alatu Figma

Slika 15. Prikaz „Analysis board“ ekrana za specifičnu partiju protiv računala kao *mockup* u alatu Figma

3.2.5. Funkcionalnost *My analyzes*

My analyzes funkcionalnost omogućava korisnicima pregledavanje, uređivanje i ponovno analiziranje njihovih vlastitih šahovskih analiza koje su prethodno spremili unutar aplikacije. Ove analize su pohranjene u posebnu tablicu baze podataka i dostupne za kasniji pregled i analizu. Funkcionalnost je vrlo slična My games, s time da je fokus na korisnikovim analizama šahovskih pozicija i partija, a ne odigranim partijama protiv računala.

Na Slici 16. prikazan je scenarij nakon klika na My analyzes u početnom zaslonu u kojemu se prikazuje popis spremljenih analiza te se nakon odabira željene analize otvara prozor u kojemu se ona prikazuje (Slika 17.).



Slika 16. Prikaz početnog zaslona s kliknutom opcijom „My analyzes“ kao *mockup* u alatu Figma

Slika 17. Prikaz „Analysis board“ ekrana za specifičnu analizu kao *mockup* u alatu Figma

4. Implementacija funkcionalnosti

U ovom poglavlju bit će predstavljen način implementacije sučelja i funkcionalnosti aplikacije ChessHub objašnjena u odlomku „3.2. Funkcionalnosti aplikacije“. Fokus će biti na detaljnom objašnjavanju tehničkih aspekata razvoja, uključujući različite *python-chess* biblioteke, Tkinter za izradu grafičkog sučelja, SQLite za pohranu i manipulaciju podacima u bazi podataka te *python-chess* za rad sa šahovskom logikom.

4.1. Prikaz ekrana za učitavanje i pohrana podataka u bazu

Kada se aplikacija pokrene, korisnicima se prikazuje ekran za učitavanje koji ih obavještava o tome da aplikacija učitava podatke u bazu. Ovaj privremeni prozor, prethodno prikazan na Slici 2., stvoren je kako bi korisnicima dao do znanja da se odvija proces obrade podataka iz PGN datoteke i njihovo spremanje u SQLite bazu podataka.

4.1.1. Kreiranje ekrana za učitavanje i pokretanje inicijalizacije u pozadini

Ekran za učitavanje implementiran je u funkciji *show_loading_screen(root)* unutar Tkintera tako što funkcija kreira zasebni prozor (*TopLevel*) sa tekstom *Loading ChessHub… please wait* i onemogućuje korisniku komunikaciju s glavnim prozorom dok se proces ne završi putem *grap_set()* funkcije (Prilog 1).

Nakon što se prikaže ekran za učitavanje, kreira se nova dretva (eng. *thread*) za paralelno pokretanje funkcije za inicijalizaciju podataka (*initialize_data)*, a cijeli kôd funkcije prikazan je u Prilogu 2. Dretva se kreira korištenjem klase *Thread* s ciljem da se učitavanje podataka iz PGN datoteke i njihovo pohranjivanje u bazu podataka izvrši u pozadini, dok glavni program i dalje prikazuje ekran za učitavanje.

4.1.2. Inicijalizacija baze podataka i provjera tablica

Funkcija *initialize_data()* najprije provjerava postoji li baza podataka sa šahovskim partijama i jesu li podaci o njima već pohranjeni. Prvi korak je definiranje putanje do SQLite baze podataka na kojem se baza podataka kreira i pohranjuje na lokalnom disku. Nakon definiranja putanje, unutarnjom funkcijom *table_exist_and_not_empty()* provjerava se postojanje tablice s danim imenom i sadrži li ona podatke. Cilj ove funkcije je optimizacija rada aplikacije iz razloga što kreiranje baze podataka parsiranjem PGN datoteke može potrajati nekoliko minuta te se ovako rješava problem na način da se baza podataka kreira prilikom prvog pokretanja aplikacije i funkcionira dok god postoji baza podataka na lokalnom disku.

U slučaju da baza podataka ne postoji, funkcija stvara *ChessDatabase* objekt iz *database_utils.py* datoteke koja definira rad baze podataka te se pozivaju metode *create_tables()* i *parse_pgn_and_store_in_db()* kako bi se kreirale potrebne tablice i pohranili podaci iz PGN datoteke u bazu podataka.

U Prilogu 3 prikazan je cijeli kôd metode *create_tables()* koja kreira sve potrebne tablice za pohranu podataka o šahovskim partijama. Najvažnija tablica povezana sa ekranom za učitavanje je tablica "*games*", koja sadrži informacije o svakoj šahovskoj partiji poput imena

igrača, rezultata, ELO ratinga, datuma natjecanja, stranice s koje su preuzete partije i notacije, a koristi stupac "*id*" kao primarni ključ koji se automatski povećava za svaku novu partiju.

Osim tablice "*games*", kreiraju se i dodatne tablice "*fens*", "*my_games*" i "*my_analyzes*". Tablica "*fens*" povezana je vanjskim ključem "*game_id*" s odgovarajućom partijom iz tablice "*games*" i pohranjuje FEN zapise svih pozicija unutar svake partije. Ova organizacija omogućuje brže pretraživanje baze podataka na temelju šahovskih pozicija. Tablice "*my_games*" i "*my_analyzes*" služe za pohranu partija odigranih protiv računala i analiza koje je korisnik spremio unutar aplikacije.

U Prilogu 4 prikazan je programski kôd metode *parse_pgn_and_store_in_db* koja pomoću *python-chess* biblioteke čita i obrađuje podatke iz PGN datoteke *twic1536.pgn* (Izvor: (Crowther, 2018)) te ih pohranjuje u bazu podataka. Metoda započinje otvaranjem PGN datoteke i uspostavljanjem veze s bazom podataka pomoću *connect_to_database()*. Nakon otvaranja datoteke, koristi se funkcija *chess.pgn.read_game* za čitanje jedne po jedne šahovske partije iz PGN datoteke. Unutar petlje, *game* objekt predstavlja jednu šahovsku partiju i za svaki takav objekt metodom *StringExporter* dobiva se notacija partije. U nastavku se kreira instanca šahovske ploče pomoću *chess.Board()*, što omogućuje praćenje šahovskih poteza kroz cijelu partiju. Generira se inicijalna FEN pozicija koja predstavlja početno stanje ploče koja se hashira pomoću *hash_fen*(*fen*), čime se osigurava efikasna pohrana i pretraga pozicija, nakon čega se inicijalizirana pozicija pohranjuje u listu *fens* zajedno s brojem poteza (*move_number*). Nakon pohrane inicijalne pozicije u listu, metodom *game.mainline_moves()* iterira se kroz sve poteze partije te se provjerava je li potez legalan u danoj poziciji (*board.is_legal(move)*). Ako nije, preskače se partiju i nastavlja s idućom, a ukoliko je legalan, svaki potez se primjenjuje na ploču te se nakon svakog poteza pohranjuje nova FEN pozicija i njena *hash* vrijednost. Kada se obrade svi potezi u partiji, podaci o partiji prikupljaju se u obliku rječnika *(game_data)* te pohranjuju u bazu metodama *save_to_gamesTable_database()* i *save_fens_to_database()*.

Obje metode koriste „Insert“ upit kojim definiraju koji će se podaci umetnuti u zadanu tablicu, a stvarne vrijednosti zamjenjuju se sa „?“. Metoda *save_to_gamesTable_database()* koristi *cursor.execute()* za pohranjivanje ključa rječnika u odgovarajući stupac u tablici *games* te vraća *cursor.lastrowid*, što je automatski generirani ID partije unutar tablice.

```
    def save_to_gamesTable_database(self, cursor, game_data):
        insert_query = '''
            INSERT INTO games (site, date, round, white, black, result, white_elo,
black_elo, eco, event_date, notation)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        '''
        cursor.execute(insert_query, (
            game_data['Site'], game_data['Date'], game_data['Round'],
            game_data['White'], game_data['Black'], game_data['Result'],
            game_data['WhiteElo'], game_data['BlackElo'],
            game_data['ECO'], game_data['EventDate'], game_data['Notation']
        ))

        return cursor.lastrowid
```

S druge strane, metoda *save_fens_to_database()* s obzirom da pohranjuje mnogo pozicija za istu partiju koristi *cursor.executemany()*, što omogućuje umetanje više redaka odjednom, gdje je svaki redak FEN pozicija.

```
def save_fens_to_database(self, cursor, game_id, fens):
    insert_query = '''
        INSERT INTO fens (game_id, move_number, fen, fen_hash)
        VALUES (?, ?, ?, ?)
    '''
    cursor.executemany(insert_query, [
        (game_id, fen['move_number'], fen['fen'], fen['fen_hash']) for fen in fens
    ])
```

Nakon što su svi podaci uspješno obrađeni i pohranjeni u bazu podataka, metoda *initialize_data* uklanja ekran za učitavanje pozivom *loading_screen.destroy()*. Korisnik se tada prebacuje na početni zaslon, koji služi kao glavno središte aplikacije i nudi razne opcije za interakciju.

4.2. Prikaz početnog zaslona aplikacije

Prilikom izrade početnog zaslona sučelja aplikacije, koristi se Tkinter biblioteka za Python koja omogućuje kreiranje grafičkog korisničkog sučelja (GUI). Funkcije definirane u ovom dijelu koda služe za izradu raznih komponenti korisničkog sučelja, uključujući izbornike, okvire, gumbe i ikone.

4.2.1. Kreiranje početnog zaslona i pozicioniranje elemenata

Osnovni prozor aplikacije kreira se pomoću *tk.Tk()* i pohranjuje u varijablu *root*. Ovaj prozor služi kao glavni okvir unutar kojeg se dodaju sve ostale komponente. Postavke poput naslova ("ChessHub"), dimenzija prozora i pozadinske boje podešavaju se odmah nakon inicijalizacije prozora. Veličina prozora automatski se prilagođava dimenzijama zaslona korisnika kako bi se aplikacija pravilno prikazala na različitim uređajima i rezolucijama. Pozadinska boja prozora definira se vrijednostima iz *config.py* datoteke.

4.2.2. Traka aplikacije

Na vrhu glavnog prozora nalazi se traka aplikacije (eng. *Appbar*), koja služi kao zaglavlje i prikazuje naziv aplikacije. Ova traka kreirana je pomoću *tk.Frame*, dok se tekst unutar nje prikazuje pomoću *Label* komponente. Traka je stilizirana koristeći vrijednosti za boju i font definirane u *config.py* datoteci. *Frame* komponenta se prostire vodoravno na vrhu prozora, dok se tekst unutar trake centrirano prikazuje kako bi istaknuo naziv aplikacije "ChessHub". Ovakva jednostavna traka aplikacije pomaže u identifikaciji aplikacije i pruža vizualnu konzistenciju.

4.2.3. Glavni sadržajni okvir i ukrasni elementi

Ispod trake aplikacije nalazi se glavni sadržajni okvir (eng. *Content frame*), kreiran pomoću *tk.Frame*. Ovaj okvir služi kao glavno područje gdje se smještaju svi interaktivni elementi aplikacije, poput gumba i ikona. Okvir se prostire vodoravno i pozicionira na vrh prozora s dodatnim razmakom (eng. *padding*) kako bi elementi unutar njega imali prostora.

Pozadina okvira također je definirana pomoću boje iz *config.py* kako bi se održala konzistentnost dizajna.

Kako bi se dodatno unaprijedilo korisničko sučelje i vizualno odvojile različite funkcionalne cjeline, unutar sadržajnog okvira dodaju se ukrasne horizontalne i vertikalne linije. Ove linije, izrađene pomoću *tk.Frame*, omogućuju odvajanje dijelova sučelja na jasan i pregledan način. Horizontalne linije dodaju se između grupa elemenata, dok se vertikalne koriste za vizualno odvajanje pojedinačnih elemenata unutar glavnog okvira.

4.2.4. Interaktivni elementi: gumbi i ikone

Osim „Play vs Computer“ gumba unutar glavnog sadržajnog okvira prikazuju se još i ikone s gumbom i opisom ostalih funkcionalnosti (*ChessHub database*, *My games*, *My analyzes*). Izgled ikona sa specifičnim opisom definiran je unutar funkcije *create_icon_with_button_and_label()*. Funkcija kreira ikonu s pripadajućim gumbima i oznakama te koristeći *load_and_resize_image()* iz *helper_methods.py* datoteke učitava i prilagođava veličinu ikone. Ikone i gumbi se smještaju unutar okvira koji se dodjeljuje u roditeljski okvir sa definiranim parametrom *command* ovisno o kojoj funkcionalnosti je riječ. Ispod ikone prikazan je dodatni *label* koji predstavlja naziv funkcionalnosti, a cijeli kôd *create_icon_with_button_and_label* funkcije vidljiv je u Prilogu 5.

```
def load_and_resize_image(path, size):
    image = Image.open(path)
    image = image.resize(size, Image.LANCZOS)
    return image
```

4.3. Implementacija funkcionalnosti igre protiv računala

Tijekom implementacije početnog zaslona izrađen je „Play vs Computer“ gumb koji predstavlja početak funkcionalnosti igre protiv računala.

4.3.1. Prozor za odabir postavki igre protiv računala

Nakon pritiska na gumb „Play vs Computer“ na glavnom zaslonu, funkcija *open_play_vs_computer_window()* otvara prozor za odabir postavki igre protiv računala. Korisniku su omogućene opcije za odabir boje figura, postavljanje vremenskih parametara te odabir razine snage računala.

Prozor je kreiran pomoću Tkinter funkcije *tk.TopLevel()* koja postavlja prozor iznad glavnog prozora aplikacije s naslovom „Play vs Computer“. Dimenzije prozora postavljene su na 1350x940 piksela, a unutar njega se stvara glavni okvir (*main_frame)* u koji će biti dodane sve komponente prozora (tekstualne, oznake, unosi za vrijeme, gumbi za odabir boje i razine). Oznake „Color“, „Time“ i „Level“ jasno upućuju korisnika na unos postavki, a pozicioniranje elemenata unutar okvira izvedeno je pomoću *grid* metode.

Za odabir boje figura koristi se okvir (*color_frame)* koji sadrži opcije „White“ i „Black“. Boja se pohranjuje u varijabli tipa *StringVar,* inicijalno postavljenoj na „White“. Za razliku od klasičnih radio gumba koje Tkinter nudi, ovdje su korištene prilagođene slike (*unchecked_image* i *checked_image*) za vizualno prikazivanje odabira boje. Slike se učitavaju

pomoću funkcije *load_and_resize_image_PhotoImage* iz datoteke *helper_methods.py*, koja ih prilagođava potrebnoj veličini i vraća je kao *PhotoImage* objekt, što je potrebno za prikazivanje slike unutar Tkinter sučelja. Nakon inicijalizacije okvira i varijable za boju, pomoću *tk.Canvas* kreiraju se dva radio gumba za boje („White“ i „Black“). Svaki gumb sadrži tekstualnu oznaku i odgovarajuću sliku koja vizualno označava trenutno odabranu boju. Klikom na bilo koji od gumba, poziva se funkcija *on_click*, koja mijenja vrijednost varijable *color_var* i poziva funkciju *update_buttons* radi ažuriranja prikaza odabranih slika.

Za unos podataka o vremenu igre koristi se okvir (*time_frame*), unutar kojeg se nalaze zasebni okviri za unos vremena i dodatka po potezu za bijelog i crnog igrača. Svaki od ovih okvira sadrži tekstualne oznake ("*White time*:", "*Increment*:", "*Black time*:") i polja za unos (*Entry*). Kako bi se osiguralo da su uneseni podaci ispravni, definirana je funkcija *validate_time_format()*, koja provjerava jesu li unosi u ispravnom brojčanom formatu. Ova funkcija registrira se i dodjeljuje svakom unosnom polju unutar vremenskog okvira kako bi se validacija obavljala u stvarnom vremenu prilikom unosa.

Za odabir razine snage računala kreiran je zaseban okvir (*level_frame*) u kojem se nalaze gumbi za svaku od mogućih razina snage, od 1 do 10. Varijabla *selected_level* tipa *StringVar* pohranjuje trenutno odabranu razinu. Gumbi za odabir razine organizirani su u dva reda unutar okvira (*row_1_frame* i *row_2_frame*), s po pet gumba u svakom redu. Klikom na gumb poziva se funkcija *select_level* (Prilog 5), koja postavlja vrijednost *selected_level* i vizualno ističe odabranu razinu, mijenjajući boju pozadine i teksta odabranog gumba.

Nakon što korisnik postavi sve željene opcije (boju figura, vrijeme igre za oba igrača i razinu računala), može započeti igru pritiskom na gumb „Play“. Ovaj gumb povezan je s funkcijom *on_play_button_click()* prikazanom u Prilogu 6, koja provodi provjeru unosa i upravlja prijelazom u prozor igre.

Funkcija *on_play_button_click()* prvo dohvaća vrijednost varijable *color_var* koja pohranjuje trenutni odabir boje figura („White“ ili „Black“), a u slučaju da nije odabrana niti jedna boja prikazuje se upozorenje s porukom: „Please select a color (White or Black).“ te se prekida daljnje izvođenje funkcije. U nastavku funkcija dohvaća unose za ukupno vrijeme i dodatak po potezu za bijelog i crnog igrača iz *Entry* polja. Ako korisnik nije unio vrijednost za dodatak po potezu, ona se postavlja na „0“. U slučaju da vrijeme za bilo kojeg igrača nije uneseno, prikazuje se upozorenje s porukom: „Please enter time for both White and Black players.“. Analogno nakon provjere vremena, provjerava se je li korisnik odabrao razinu snage računala pomoću varijable *selected_level* i u slučaju da nije odabrana niti jedna razina, prikazuje se upozorenje: „Please select a level.“ te se prekida daljnje izvođenje funkcije.

Ako su svi podaci uneseni ispravno, trenutni prozor se zatvara, a funkcija *start_game_screen()* iz *gameScreen.py* datoteke se poziva s argumentima koji definiraju postavke igre: boju figura, ukupno vrijeme i dodatak za oba igrača te odabranu razinu računala.

4.3.2. Prozor za igru protiv računala

Nakon što korisnik postavi sve željene postavke i pritisne gumb "Play", otvara se prozor za igru protiv računala. Ovaj prozor implementiran je pomoću funkcije *start_game_screen()* u *gameScreen.py* datoteci, koja postavlja elemente korisničkog sučelja potrebne za igru te omogućuje interakciju između igrača i šahovskog računala.

4.3.2.1. Inicijalizacija prozora i elemenata

Prozor za igru kreiran je pomoću *tk.Toplevel()* koji omogućuje otvaranje novog prozora s naslovom "Play vs Computer – {level}" u kojemu „*level"* predstavlja proslijeđeni *string* podatak o razini računala, npr. Level 1. Dimenzije prozora automatski se prilagođavaju veličini ekrana korisnika kako bi se osigurala preglednost svih elemenata. Unutar ovog prozora nalaze se dva glavna okvira:

1. Okvir za šahovsku ploču (*board_frame)*
    - **Vizualizacija ploče:** šahovska ploča prikazuje se pomoću *tk.Canvas* što omogućuje dinamičko crtanje i ažuriranje prikaza ploče nakon svakog poteza, a veličina ploče automatski se prilagođava dimenzijama prozora, osiguravajući optimalnu vidljivost i korištenje prostora na ekranu
    - **Prilagodba veličine:** ploča se postavlja tako da zauzima 70% širine prozora i 90% visine, što omogućuje igraču da jasno vidi šahovska polja i figure
    - **Interakcija s pločom:** *Canvas* je ključan za hvatanje korisničkih unosa poput klikova mišem, omogućujući igraču da povlači poteze klikom na figure na ploči
2. Informativni okvir (*info_frame)*
    - **Prikaz informacija o igri:** smješten je desno od ploče i služi kao središte informacija vezanih uz igru
        - **Tekstualni okvir za prikaz notacije poteza ():** implementiran pomoću *tk.Text,* a prikazuje sve dosadašnje poteze u standardnom šahovskom zapisu (PGN) s opcijom ažuriranja notacije nakon svakog poteza igrača ili računala
        - **Satovi za oba igrača:** prikazuje satove pomoću *tk.Label,* a vrijeme se ažurira u stvarnome vremenu tijekom igre; posjeduju dodatne vizualne elemente koji prikazuju boju igrača uz njihov odgovarajući sat
        - **Gumb „Resign and Leave":** smješten na dnu, a omogućuje korisniku da preda partiju i napusti igru s dodatnim opcijama za spremanje partije ili jednostavno napuštanje igre

4.3.2.2. Implementacija ChessGUI klase

Klasa *ChessGUI* predstavlja glavni element koji upravlja korisničkim sučeljem igre protiv računala. Unutar ove klase nalaze se ključne metode za interakciju sa šahovskom pločom, praćenje vremena, interakciju s računalnim protivnikom (Stockfish) i upravljanje prikazom poteza.

Prilikom kreiranja instance klase *ChessGUI* koriste se razni argumenti poput roditeljskog okvira (*board_frame*), glavnog prozora igre (*gameScreenWindow*), ploče (*board*), boje igrača *(player_color*), vremena za oba igrača (*white_time i black_time*), vremenskih dodataka *(white_increment* i *black_increment*) te razine snage računala (*level*). Na temelju korisnikovih unosa i ovih argumenata, kreira se objekt *chess_gui*, kao što je prikazano u nastavku:

```
chess_gui = ChessGUI(board_frame, gameScreenWindow, board, flipped=flipped,
player_color=chess.WHITE if color.lower() == "white" else chess.BLACK,
white_time=white_time, black_time=black_time, white_increment=white_increment,
black_increment=black_increment, level=level)
```

Unutar „*__init__*“ metode klase definiraju se i inicijaliziraju varijable i komponente korisničkog sučelja koje omogućuju cjelokupno upravljanje igrom. Prvo se postavlja *self.board,* koji prima *board* objekt iz biblioteke *python-chess.* Ovaj objekt koristi se za praćenje stanja igre uključujući položaj figura, red poteza i dostupne legalne poteze. Nadalje, vizualizacija šahovske ploče unutar korisničkog sučelja ostvaruje se pomoću *self.canvas,* koji se kreira kao *tk.Canvas* okvir. Unutar njega se crta šahovska ploča, a njezin prikaz se automatski prilagođava veličini ekrana, osiguravajući optimalno korištenje prostora pomoću metode *self.canvas.pack(expand=True, fill=tk.BOTH).* Zahvaljujući *self.canvas* elementu, implementirana je interakcija s pločom koja omogućuje igraču da klikne na šahovske figure i povlači poteze. Funkcija *draw_board* koristi *self.canvas* komponentu za prikaz trenutnog stanja ploče, dok se svaki potez na ploči pohranjuje unutar *board* objekta. Klikovi mišem na ploči hvataju se pomoću metode *on_click,* koja identificira odabrano polje na temelju koordinata klika. Ako je klik na ispravno polje i potez je legalan, figura se pomiče, a metoda *make_ai_move* se poziva kako bi Stockfish odigrao svoj potez.

Prije nego što se pozove metoda *make_ai_move* kako bi računalo odigralo potez, potrebno je uspostaviti vezu s računalnim *chess engineom* (Stockfish). Klasa ChessGUI vezu uspostavlja pomoću metode *init_engine*. Koristi se za inicijalizaciju *self.engine*, koji se pokreće pomoću *chess.engine.SimpleEngine.popen_uci(config.STOCKFISH_PATH)*. Ova metoda omogućuje aplikaciji komunikaciju s unaprijed instaliranim Stockfish *engineom*, koristeći univerzalno šahovsko sučelje (eng. *Universal Chess Interface*)[5] za razmjenu podataka. Stockfish je *chess engine* otvorenog koda, preuzet i instaliran sa službene stranice (Izvor: (Stockfish, 2024)). Putanja do instalacije Stockfish enginea definirana je u varijanti *STOCKFISH_PATH* unutar *config.py* datoteke:

```
STOCKFISH_PATH = "/home/daniel/Desktop/stockfish/stockfish-ubuntu-x86-64-avx2"
```

Nakon pokretanja *enginea*, koristi se metoda *init_engine* čiji je kôd prikazan u Prilogu 7. Ona konfigurira Stockfish na temelju razine snage protivnika (*level*) koju je korisnik odabrao u postavkama igre. Kroz ovu konfiguraciju, engine se prilagođava tako da postavlja parametre poput *skill_level*, *threads*, *hash* i *elo*, čime se dinamički određuje snaga igre računala te izazov s kojim će se korisnik suočiti. Navedeni kôd koristi unaprijed definiranu mapu *self.level_map*

[5] Universal Chess Interface (UCI) je protokol koji omogućuje komunikaciju *chess enginea* i korisničkog sučelja (Universal Chess Interface (UCI), 2024)

za dohvat vrijednosti konfiguracije koje odgovaraju razini protivnika. Razina određuje koliko će Stockfish biti težak protivnik. Na primjer, "Level 1" ima nižu vrijednost *skill_level* i koristi manje resursa (npr. samo jedan procesorski *thread*), dok "Level 10" koristi veći broj *threads* i veći *hash*, što omogućava engineu da radi s više podataka i dublje analizira šahovsku poziciju. Detaljan prikaz vrijednosti parametara za svaku razinu vidljiv je u Prilogu 8.

Ovi parametri omogućavaju prilagodbu snage Stockfish *enginea*. Parametar *skill_level* određuje opću razinu vještine *enginea*, pri čemu niže vrijednosti odgovaraju lakšem protivniku, dok više vrijednosti označavaju teže razine igre. *Threads* definira broj procesorskih niti koje Stockfish koristi za analizu, čime se povećava učinkovitost i brzina analize. *Hash* postavlja količinu memorije (MB) koju *engine* može koristiti za pohranu prethodno analiziranih pozicija, što izravno utječe na sposobnost *enginea* da "pamti" i analizira složenije pozicije. Parametar *elo* simulira stvarni ELO rating šahovskog igrača, omogućavajući korisniku da odabere razinu protivnika koja je u skladu s njegovim vlastitim vještinama. Dodatno, *move_overhead* i *nodestime* kontroliraju vrijeme koje *engine* troši na analizu svakog poteza, dok *limit_strength* omogućava ograničavanje snage *enginea* kako bi se ponašao kao slabiji protivnik na nižim razinama.

Nakon što je veza sa Stockfishom uspješno uspostavljena, omogućeno je igranje poteza računala pomoću metode *make_ai_move*. Ova metoda pokreće poseban *thread*, što osigurava da se računalo može savjetovati s *engineom* za odabir najboljeg poteza bez ometanja korisničkog sučelja. Metoda prvo provjerava je li igra završena (provjerom stanja *self.game_over* i *self.board.is_game_over()*). Ako igra nije gotova, započinje odbrojavanje vremena za računalo te se koristi *self.engine.play()* za analizu trenutne pozicije i pronalaženje najboljeg poteza. Nakon što se potez odabere i izvrši na ploči, metoda ažurira vizualni prikaz ploče i bilježi potez u notaciju.

Sastavni dio igre je i praćenje vremena za oba igrača. Unutar *__init__* metode definiraju se satovi za bijelog i crnog igrača koristeći *tk.Label*. Vrijeme za svakog igrača postavlja se na temelju korisničkih unosa (*white_time* i *black_time*) i pohranjuje u varijable *self.white_time* i *self.black_time*. Implementirane metode *start_white_timer* i *start_black_timer* pokreću odbrojavanje vremena za igrača na potezu. Sat se ažurira u stvarnom vremenu pomoću metoda *update_white_time* i *update_black_time,* koje smanjuju preostalo vrijeme za svakog igrača i osvježavaju prikaz na sučelju. Kada vrijeme za jednog od igrača istekne, igra se automatski završava pozivom metode *end_game*. Osim osnovnog odbrojavanja, dodane su i funkcionalnosti za pauziranje vremena (metode *pause_white_timer* i *pause_black_timer*), koje se aktiviraju svaki put kada igrač završi potez, uz dodavanje vremenskog bonusa definiranog varijablama *white_increment* i *black_increment*.

Posebna pažnja posvećena je i šahovskom pravilu promocije pješaka. Kada pješak dosegne posljednji red, metoda *promote_pawn* otvara novi prozor s opcijama za promociju pješaka (kraljica, top, lovac, konj). Metoda koristi *tk.Button* za prikaz slika figura i omogućuje igraču da odabere željenu figuru za promociju. Kada igrač odabere figuru, metoda generira odgovarajući potez promocije koristeći *chess.Move* i ažurira stanje ploče. Ova funkcionalnost osigurava poštivanje šahovskih pravila i ispravno bilježenje poteza tijekom igre.

Dodatna funkcionalnost koja se odvija nakon svakog odigranog poteza, a također je omogućena unutar klase *ChessGUI,* jest prikaz notacije poteza pomoću metode *update_notation().* Ova metoda koristi listu *self.notation_moves* koja pohranjuje sve odigrane poteze u igri. Na temelju tih podataka, metoda generira *string* u kojem su svi potezi organizirani po rednim brojevima. Na primjer, za prikaz poteza "1. e4 e5" koristi se *f-string* format kako bi se svaki par poteza (bijelog i crnog) prikazao uz odgovarajući redni broj. Unutar *self.notation_text,* koji je instanca *tk.Text*, notacija se ažurira na način da se prvo omogućava izmjena postavkom stanja na "*normal*", zatim briše trenutni sadržaj, dodaje novi *string* s notacijom, te na kraju ponovno onemogućava izmjene postavkom stanja na "*disabled*". Ova funkcionalnost osigurava da korisnik može pratiti tijek igre u standardnom šahovskom zapisu (PGN) te da taj zapis ostane nepromijenjen tijekom trajanja igre.

Korisnik također može prekinuti partiju u bilo kojem trenutku pritiskom na gumb "Resign and Leave". Ovaj gumb povezan je s funkcijom *on_resign_button_click,* koja otvara novi prozor za potvrdu predaje koristeći *tk.Toplevel*. Unutar ovog prozora, korisniku se prikazuje poruka s upitom: "Are you sure you want to resign and leave?" i dvije opcije: "Save and Leave" i "Leave". Gumb "Save and Leave" poziva unutarnju metodu *save_and_leave*. Ova metoda koristi *save_game_as_resignation* za spremanje trenutnog stanja igre s oznakom predaje, tj. generira PGN zapis partije, uključujući imena igrača, rezultat (npr., "0-1" ako je bijeli igrač predao) te vrijeme igre. Potom se podaci o partiji pohranjuju u bazu podataka pomoću ChessDatabase objekta koji sadrži funkciju *save_game_to_my_games* za pohranu podataka u „*my_games"* tablicu u bazi podataka. Nakon pohrane poziva se metoda *quit_game* koja zatvara sve prozore, prekida aktivne procese i vraća aplikaciju na početni zaslon (Prilog 9). Gumb "Leave" poziva unutarnju funkciju *just_leave,* koja jednostavno prekida igru bez spremanja podataka. Funkcija zatvara trenutni prozor za igru i resetira aplikaciju na početni zaslon.

U trenutku kada se završi partija, bilo pobjedom, remijem, istekom vremena ili predajom, poziva se metoda *end_game* (Prilog 10). Ova metoda prvo zaustavlja sve satove, zatvara *chess engine* i postavlja *self.game_over* na True, čime se označava završetak igre. Nakon toga metoda poziva *get_game_result* kako bi utvrdila rezultat igre (npr., "1-0" za pobjedu bijelog, "0-1" za pobjedu crnog, "1/2-1/2" za remi) te prosljeđuje te informacije metodi *show_result_menu*. Metoda *show_result_menu* otvara novi prozor (*tk.Toplevel*) koji korisniku prikazuje rezultat igre. U prozoru se nalaze tri gumba za interakciju sa krajem partije: "Save", "New Game" i "Close and Leave". Gumb "Save" poziva funkciju *save_game*, koja bilježi sve informacije o partiji, uključujući bijelog i crnog igrača, rezultat, vrijeme igre te sve odigrane poteze u PGN formatu. Ovi podaci se zatim pohranjuju u bazu podataka pomoću *save_game_to_my_games* metode. Nakon uspješne pohrane, zatvaraju se svi prozori te se aplikacija ponovno pokreće. Gumb "New Game" poziva metodu *new_game,* koja zatvara trenutni prozor i vraća korisnika na početni zaslon sa otvorenim prozorom za odabir postavki nove igre. Gumb "Close and Leave" poziva funkciju *close_and_leave,* koja zatvara sve prozore i ponovno pokreće aplikaciju, osiguravajući čisto stanje za novu igru.

4.4. Implementacija ploče za analizu

Ploča za analizu (eng. *Analysis Board)* implementirana je u sklopu aplikacije za pregled i analizu šahovskih partija, pružajući korisnicima alat za detaljno istraživanje šahovskih pozicija i varijacija. Prozor za analizu kreira se pomoću funkcije *open_analysis_board_window(),* koja se poziva klikom na „*Analysis board*“ ikonu na početnom zaslonu (Slika 3). Ona pokreće novo okruženje s ključnim komponentama korisničkog sučelja kao što su ploča za prikaz šahovskih pozicija, tekstualni okvir za bilježenje notacije poteza, te alati za interakciju i kontrolu analize. Unutar ovog prozora korisnik može pregledavati odigrane poteze, istraživati različite varijante igre, koristiti ugrađeni *chess engine (kibitzer)* za procjenu pozicija te manipulirati postavkama igre na jednostavan i intuitivan način.

4.4.1. Inicijalizacija prozora i elemenata

Prozor sa pločom za analizu *(analysisWindow)* otvara se pomoću *tk.Toplevel()* i automatski se prilagođava veličini ekrana korisnika. Otvoreni prozor sastoji se od dva glavna okvira:

1. Okvir za šahovsku ploču (*board_frame)*
    - **Vizualizacija ploče:** šahovska ploča prikazuje se pomoću *tk.Canvas* gdje se ploča crta pomoću metode *draw_board()* koja koristi biblioteke poput *cairosvg* i *PIL* za prikaz SVG grafike šahovskih figura i polja
    - **Prilagodba veličine:** ploča se postavlja tako da zauzima 70% širine prozora i 90% visine, što omogućuje igraču da jasno vidi šahovska polja i figure
    - **Interakcija s pločom:** omogućena je pomoću *on_click()* metode i *Canvasa* koji je ključan za hvatanje korisničkih unosa poput klikova mišem, omogućujući igraču da povlači poteze klikom na figure na ploči
2. Informativni okvir (*info_frame)*
    - **Prikaz informacija o igri:** smješten je desno od ploče i sadrži tri glavna okvira za prikaz notacije poteza, pregled referenci i dodavanje *chess enginea (kibitzer)*
    - U gornjem dijelu se nalaze gumbi za prebacivanje između notacije, referenci i kibitzera, a interakcija između tih prikaza ostvaruje se pomoću funkcije *select_button()* koja kontrolira vidljivost i sadržaj okvira
    - **Prikaz notacije (*notation_frame*)**: *notation_frame* sadrži *tk.Text* komponentu koja se koristi za prikaz i ažuriranje notacije poteza u standardnom šahovskom formatu (PGN). Metoda *update_notation()* ažurira sadržaj ovog okvira sa svakim novim potezom, generirajući PGN zapis igre i ističući aktivni potez žutom bojom pozadine i podebljanim tekstom. Navigacija kroz poteze omogućena je pomoću gumba za naprijed i nazad (*prev_button* i *next_button),* koji korisniku omogućuju pregled varijacija igre
    - **Prikaz referenca (reference_frame):** prikazuje informacije o najčešće odigranim potezima i partijama s istim pozicijama iz baze podataka. Metoda *reference()*

pretražuje i analizira bazu podataka šahovskih partija, generirajući tablicu s najpopularnijim potezima, brojem partija u kojima je igran taj potez, njihovom posljednjom odigranom godinom te najboljim igračima koji su igrali te poteze

- **Add kibitzer (kibitzer_frame):** analizira trenutnu poziciju na ploči pomoću Stockfish *chess enginea* i prikazuje najbolje poteze za određenu poziciju. Komunikacija sa Stockfishom ostvaruje se pomoću metode *analyze_position()* koja koristi *self.engine.analyse()* za dobivanje evaluacije pozicije. Unutar okvira prikazuje se najboljih pet varijanti za trenutnu poziciju pomoću *ttk.Treeview*

4.4.2. Implementacija klase *ChessGUI*

Klasa *ChessGUI* ključni je element koji upravlja funkcionalnostima ploče za analizu. Prilikom kreiranja instance ove klase, koriste se argumenti poput roditeljskog prozora (*root*), prozora za analizu (*analysisWindow*), ploče (*board*), tekstualnog okvira za notaciju (*notation_text*), navigacijskog okvira (*nav_frame*) i notacijskog *stringa* partije (*pgn_string*). Ovi parametri omogućuju stvaranje i upravljanje različitim komponentama prozora za analizu.

```
chess_gui = ChessGUI(board_frame, analysisWindow, board, notation_text, nav_frame, pgn_string=notation)
```

Unutar metode *__init__()*, definira se i inicijalizira šahovska ploča pomoću *self.canvas*, te se prikaz prilagođava veličini prozora pomoću metode *draw_board()*. Za svaku interakciju s pločom, poput odabira polja ili izvođenja poteza, koristi se metoda *on_click()*, koja bilježi korisnikove akcije i upravlja odabranim potezima na ploči.

Posebna pažnja posvećena je i pravilu promocije pješaka. Kada pješak dosegne posljednji red na ploči, metoda *promote_pawn()* otvara novi prozor s opcijama za promociju pješaka (kraljica, top, lovac, konj). Metoda koristi *tk.Button* za prikaz odgovarajućih figura, omogućujući korisniku da odabere željenu figuru za promociju. Nakon odabira, metoda ažurira stanje ploče, bilježi potez te osvježava vizualni prikaz ploče.

Metoda *update_notation()* ključna je za pravilno praćenje i vizualni prikaz tijeka igre unutar ploče za analizu te je prikazana u Prilogu 11. Koristi se za generiranje i ažuriranje šahovske notacije, koja se prikazuje unutar tekstualnog okvira. Prvi korak metode uključuje omogućavanje uređivanja okvira za notaciju postavljanjem atributa *state* na "normal", nakon čega se briše postojeći sadržaj. Zatim se koristi objekt *chess.pgn.StringExporter* za generiranje PGN zapisa trenutne igre, koji se potom unosi u tekstualni okvir pomoću *self.notation_text.insert()*. Na ovaj način, svi trenutni potezi i njihove varijacije postaju vidljivi korisniku u standardiziranom šahovskom zapisu. Nakon ažuriranja prikaza notacije, metoda inicijalizira prazan rječnik *self.fen_dict*, koji će služiti za praćenje svih pozicija na ploči pomoću FEN zapisa. Ovaj rječnik povezuje svaku poziciju s njenim odgovarajućim potezom i njihovim pozicijama unutar teksta. Metoda zatim prolazi kroz sve varijacije igre koristeći stog (eng. *stack*), koji sadrži čvor igre, kopiju trenutne ploče, indeks trenutne pozicije unutar notacije i indeks varijacije. Za svaki potez provjerava se njegova legalnost u trenutnoj poziciji, a ako je potez legalan, konvertira se u SAN i unosi na ploču. Zatim metoda traži poziciju tog poteza unutar tekstualnog okvira te dodaje podatke FEN zapisa prethodnog poteza u rječnik za buduće praćenje. U slučaju da je metodi proslijeđen argument *current_move*, ona dodatno traži

FEN zapis trenutne pozicije na ploči. Ako se pronađe odgovarajući zapis, metoda ističe taj potez unutar okvira koristeći s*elf.notation_text.tag_add(),* mijenjajući njegov stil tako da bude lako prepoznatljiv korisniku. Ovaj proces osigurava jasan i intuitivan prikaz igre, omogućavajući korisniku jednostavnu navigaciju kroz poteze i njihove varijacije. Na kraju, metoda ponovno onemogućuje uređivanje notacijskog okvira postavljanjem *state* na "*disabled*", čime osigurava integritet zapisa partije.

Metode za navigaciju kroz partiju, *prev_move()* i *next_move(),* te za manipulaciju varijantama partije, poput *promote_to_main_variation(), delete_variation()* i *demote_variation(),* čine osnovu za analizu partija unutar ploče za analizu, omogućujući korisnicima istraživanje različitih poteza i njihovih varijacija. Navigacija kroz partiju ostvarena je pomoću *prev_move()* i *next_move()* metoda, koje omogućuju kretanje kroz povijest partije, unatrag i unaprijed. Ove metode ne samo da ažuriraju trenutnu poziciju na ploči, već i automatski osvježavaju notaciju kako bi korisnik imao jasan pregled partije u bilo kojem trenutku. Kako bi se poboljšala interaktivnost, metode su povezane s tipkama na tipkovnici: pritiskom na tipku sa strelicom lijevo *(„<Left>“)* poziva se metoda *prev_move(),* dok pritiskom na tipku sa strelicom desno *(„<Right>“)* poziva se metoda *next_move().*

S druge strane, metode za manipulaciju varijantama partije: *promote_to_main_variation(), delete_variation()* i *demote_variation()* omogućavaju korisnicima analizu različitih poteza i prilagodbu tijeka partije, što je u šahovskoj praksi vrlo važno. Definirane funkcije integrirane su s modulom *chess.pgn* iz *python-chess* biblioteke, koji omogućuje rad sa šahovskim zapisima u PGN formatu. U aplikaciji im se pristupa putem odgovarajućih gumba ("Promote", "Delete", "Demote") koji se pojavljuju u sučelju u ovisnosti o trenutnom kontekstu igre. Na primjer, *promote_to_main_variation()* koristi *chess.pgn* strukturu za unapređenje trenutne varijante na glavnu liniju igre, čime omogućuje korisniku isticanje određenog poteza kao ključnog u analizi. Ipak, "Promote" gumb će se korisniku pojaviti jedino ako se trenutni potez nalazi u podvarijanti. Slično, metoda *delete_variation()* uklanja odabranu varijantu iz trenutnog čvora partije, ažurirajući interni model igre unutar *chess.pgn*. Konačno, *demote_variation()* omogućuje korisniku "spuštanje" varijante, odnosno promjenu statusa varijante iz glavne linije u alternativnu. Specifični uvjeti prikazivanja svakog od ova tri gumba definirani su u funkciji *update_variation_controls()* prikazanog u Prilogu 12.

Metoda *reference()* pruža korisnicima dodatne informacije i statističke podatke o trenutnoj šahovskoj poziciji na ploči. Implementacijom ove funkcije korisnicima se omogućuje pregled prošlih partija iz baze podataka (tablica „*games*“) u kojima se pojavila ista pozicija. Osim samih partija, metoda računa i prikazuje statističke podatke koji uključuju najbolje sljedeće poteze, njihove postotke uspjeha te informacije o igračima koji su te poteze koristili. Nakon svakog odigranog poteza ili ponovnog klika na gumb "Reference", svi prethodni elementi unutar okvira *reference_frame* se uklanjaju korištenjem metode *winfo_children(),* čime se osigurava ažurirani prikaz podataka o trenutnoj poziciji. Zatim se FEN zapis trenutne šahovske pozicije na ploči dohvaća pomoću *self.board.fen()* te se koristi za generiranje *hash* vrijednosti koja služi kao jedinstveni identifikator pozicije. Metodom *hash_fen,* definiranoj u *helper_methods.py* datoteci, dobiva se *hash* koji se koristi za dohvaćanje partija iz baze

podataka pozivom metode *chess_db.get_game_data_for_fen(fen_hash)* (Prilog 13), koja vraća listu partija koje sadrže navedenu poziciju.

Sljedeći dio metode odnosi se na stvaranje grafičkog sučelja unutar *reference_frame*. Prvo se kreira okvir *next_moves_info_frame,* unutar kojeg se postavlja tablica koja prikazuje najbolje sljedeće poteze za trenutnu poziciju. Stilovi tablice definirani su pomoću *ttk.Style()* kako bi se osigurao odgovarajući vizualni prikaz, uključujući prilagodbu fonta, boje pozadine, boje odabranog retka i obruba. Unutar tablice definiraju se stupci ("*Move*", "*Games*", "*Score*", "*Last Played*", "*Best Players*") i njihove širine, a potom se u tablicu unose podaci o potezima. Podaci potrebni za popunjavanje tablice, kao što su broj igara u kojima se potez pojavio, datum posljednjeg odigranog poteza, uspješnost poteza i najbolji igrači koji su ga koristili, dohvaćaju se pozivom izdvojene metode *get_next_moves().* Funkcija *get_next_moves()* odvojena je iz glavnog tijela *reference()* kako bi se pojednostavila logika i olakšalo održavanje koda. Budući da se analiza poteza može odvijati neovisno o stvaranju grafičkog sučelja, ova metoda izolira logiku obrade podataka te omogućuje njezino ponovno korištenje, čime se poboljšava modularnost i čitljivost koda. Osim toga, *get_next_moves()* prolazi kroz sve partije i identificira sljedeće moguće poteze iz danog FEN zapisa, računajući njihove atribute (broj pojavljivanja, uspješnost, datum posljednjeg igranja, najbolji igrači) i zatim vraća te podatke za daljnju obradu. Unutar *reference(),* pozivom metode *get_next_moves()* dobivaju se ti podaci, koji se potom koriste za popunjavanje tablice. Za prikaz svakog poteza i pripadajućih atributa unutar *ttk.Treeview* tablice koriste se i metode *calculate_score_for_move* i *calculate_best_players_for_move*, koje dodatno obogaćuju statističke informacije vezane uz prikazane poteze.

Nakon prikaza najboljih poteza stvara se još jedna tablica unutar *tree_frame*, koja prikazuje detalje partija koje uključuju trenutnu poziciju. Stupci ove tablice uključuju informacije poput broja partije, imena bijelog i crnog igrača, njihovih ELO ratinga, rezultata partije, naziva turnira i datuma. Svaki redak se popunjava podacima iz baze podataka za odgovarajuće partije (tablica „*games*“). Metoda također uključuje i mogućnost pregleda detalja svake partije dvostrukim klikom na redak u tablici. Naime, pomoću funkcije *on_item_double_click()*, dvostrukim klikom na određenu partiju dohvaćaju se svi relevantni podaci za tu igru, a ako su podaci uspješno dohvaćeni, pokreće se analiza odabrane partije pozivom metode *open_analysis_board_window*. Ova metoda otvara novi prozor s detaljima partije, uključujući prikaz notacije i poteza, čime korisniku omogućuje analizu odabrane pozicije.

Metoda *add_kibitzer()* omogućava korisnicima pregled trenutne šahovske pozicije pomoću Stockfish *enginea*, pružajući uvid u najbolje poteze i procjene pozicija. Prilikom poziva ove metode, najprije se provjerava je li "*kibitzer*" već aktivan pomoću varijable *self.is_kibitzer_active*. Ako nije, metoda se pokreće postavljanjem *self.is_kibitzer_active* na *True* i pozivanjem metode *run_kibitzer()*. U suprotnom, ako je već aktivan, metoda zaustavlja ovaj proces postavljanjem *self.is_kibitzer_active* na *False* te zatvara Stockfish *engine* pomoću *self.engine.quit().* Na taj način se osigurava da *engine* nije pokrenut više puta istovremeno, što bi moglo dovesti do preopterećenja resursa.

Metoda *run_kibitzer()* zadužena je za kontinuiranu analizu pozicije sve dok je *kibitzer* aktivan. Analiza pozicija pomoću *chess enginea* funkcionira tako da se pohrani trenutni FEN zapis pomoću *self.board.fen()* te se proslijedi metodi *analyze_position()* koja koristi Stockfish *engine* za evaluaciju pozicije. Pomoću *self.root.after(2000, self.run_kibitzer)* metoda osigurava da se analiza automatski ponavlja svake dvije sekunde, pružajući tako korisniku ažurirane informacije o trenutnoj situaciji na ploči. U situaciji kada korisnik zaustavi analizu pozicije od strane *chess enginea*, metoda prekida ovaj ciklus i analiza se zaustavlja.

Da bi se prikaz rezultata analize omogućio u korisničkom sučelju, koristi se metoda *add_kibitzer_frame()* koja provjerava postoji li okvir za kibiciranje (*self.kibitzer_frame*). Ako nije kreiran, metoda inicijalizira okvir i konfigurira njegov izgled. U gornji dio okvira dodaje se oznaka s tekstom "Powered by: Stockfish 17", čime se daje do znanja da je analiza podržana Stockfish *engineom*. Pomoću *ttk.Treeview* tablice unutar okvira konfigurira se prikaz informacija o potezima, gdje se definira nekoliko stupaca: "Moves" za prikaz sekvence poteza i "Eval" za prikaz evaluacije pozicije u brojčanom obliku, što korisniku omogućava pregled trenutne procjene pozicije i najboljih poteza koje preporučuje *engine*. Ako *self.kibitzer_frame* već postoji, metoda ga samo "podiže" na vrh korisničkog sučelja kako bi bio vidljiv korisniku.

Ključna komponenta analize je metoda *analyze_position(),* koja prima FEN zapis trenutne ploče, postavlja ploču na tu poziciju i koristi Stockfish *engine* za procjenu. Analiza se vrši s vremenskim ograničenjem od dvije sekunde, nakon čega se generira pet najboljih varijanti poteza. Metoda zatim briše sve postojeće rezultate iz tablice *self.kibitzer_tree* i ažurira ju s novim podacima. Prikazuju se potezi pomoću SAN i njihova evaluacija, kako bi korisnik mogao brzo uočiti najbolje mogućnosti i njihov utjecaj na poziciju. Dubina pretraživanja, koja označava koliko „detaljno“ Stockfish analizira poziciju, također se prikazuje pomoću *self.kibitzer_info_label.config().* Ovaj detalj je važan jer veća dubina obično znači precizniju procjenu pozicije. Za svaki potez provjeren u analizi, metoda ga dodaje na ploču, preračunava ocjenu i zatim ga uklanja, čime osigurava konzistentnost ploče za buduće analize.

4.4.3. Implementacija prozora za pohranu analize u bazu podataka

Kada korisnik odluči zatvoriti prozor ploče za analizu, metoda *on_closing()* generira PGN zapis trenutne igre i otvara prozor za pohranu analize. Unutar ove metode koristi se *chess.pgn.StringExporter* za kreiranje PGN zapisa bez zaglavlja, uključujući sve varijacije i komentare partije. Dobiveni PGN zapis prosljeđuje se metodi *open_save_analysis_window()* iz datoteke *save_analyze_screen.py* koja inicijalizira prozor za unos podataka o partiji i pohranu analize u bazu podataka prikazan na Slici 11. Kako bi se spriječilo slučajno zatvaranje prozora bez pohrane podataka, metoda *analysisWindow.protocol("WM_DELETE_WINDOW", on_closing)* definira ponašanje pri zatvaranju prozora tako da se uvijek poziva *on_closing().*

Metoda *open_save_analysis_window()* kreira novi prozor unutar kojega korisnik može unijeti dodatne informacije o partiji, poput imena igrača, njihovih ELO ratinga, turnira, rezultata, te datuma partije. Unutar prozora definira se skup unosa pomoću *tk.Entry* i *tk.Radiobutton* elemenata. Također, koriste se funkcije za validaciju unosa, poput *validate_numeric_input(), validate_month_input(),* i *validate_day_input(),* kako bi se

osiguralo da unos podataka bude ispravan (npr. da ELO rating sadrži samo brojeve u rasponu od 0 do 9999).

Na dnu prozora nalaze se tri gumba za interakciju: "Save", "Reset", i "Discard". Gumb "Save" poziva metodu *save_analysis(),* koja prikuplja unesene podatke, generira šahovsku partiju koristeći biblioteku *chess.pgn*, te pohranjuje podatke u PGN datoteku i bazu podataka (tablica „*my_games*“). Pohrana u bazu podataka funkcionira na način da se podaci spremaju u rječnik (*game_data*), koji uključuje informacije poput imena igrača, rezultata, ELO ratinga, datuma partije, imena turnira, runde i notacije te se potom poziva metoda *db.save_analysis_to_database(game_data)* iz *database_utils.py* datoteke kako bi pohranila te podatke u bazu (Prilog 14). Nakon uspješnog pohranjivanja, aplikacija se ponovno pokreće pomoću *os.execl()* kako bi se osvježio prikaz pohranjenih analiza. Gumb "Reset" poziva funkciju *reset_fields(),* koja briše sve unesene podatke iz okvira za unos, omogućujući korisniku ponovno unošenje podataka. Ako korisnik odluči napustiti unos bez spremanja, gumb "Discard" jednostavno zatvara prozor.

4.5. Implementacija pregleda šahovskih partija iz baze podataka

Opcije za pregled partija iz baze podataka, poput ChessHub Database, My Games, i My Analyzes funkcionalnosti, omogućavaju korisnicima pregled šahovskih partija pohranjenih u lokalnoj bazi podataka. Iako sve tri funkcionalnosti dijele sličan dizajn i princip rada, ključne razlike odnose se na izvore podataka koje prikazuju i uvjete pod kojima se podaci pohranjuju.

Sve tri funkcionalnosti koriste *create_icon_with_button_and_label()* metodu za kreiranje ikona na početnom zaslonu, kako je objašnjeno u poglavlju 4.2.4. Interaktivni elementi: gumbi i ikone. Klikom na odgovarajuću ikonu otvara se pregled partija prikazan pomoću *Treeview* tablice, koja se dinamički puni podacima iz odgovarajuće baze podataka. Implementirana interaktivnost osigurava da korisnici mogu dvostrukim klikom na odabranu partiju ili analizu otvoriti njezinu detaljnu analizu u prozoru ploče za analizu (*analysisWindow*).

ChessHub Database opcija prikazuje popis partija iz baze podataka (tablica *games*), koja se puni parsiranjem PGN datoteke prilikom pokretanja aplikacije (poglavlje 4.1.). Ovo korisnicima omogućuje pristup velikom broju partija, koje se mogu pretraživati i analizirati. Funkcija *display_data()* zadužena je za prikazivanje partija u tablici, dohvaćajući podatke iz *games* tablice. Ona kreira i stilizira *Treeview* tablicu u kojoj se prikazuju informacije poput broja partije, imena bijelog i crnog igrača, njihovih ratinga (ELO), rezultata, stranice s koje su preuzete (eng. *Site*) i datuma. Metoda također dodaje funkcionalnost dvostrukog klika na određenu partiju, čime korisnik otvara partiju u novom prozoru ploče za analizu.

My Games funkcionalnost, s druge strane, prikazuje samo partije koje je korisnik odigrao protiv računala i potom ih pohranio u bazu. Ove partije pohranjuju se u zasebnu *my_games* tablicu unutar baze podataka. Ranije objašnjena metoda *save_game_to_my_games()* pohranjuje podatke o odigranoj partiji, uključujući bijelog i crnog igrača, rezultat, vrijeme igre i notaciju poteza. Kada korisnik klikne na "My Games" ikonu, metoda *display_my_games()* dohvaća podatke iz *my_games* tablice i prikazuje ih u tablici na

ekranu. Struktura prikaza slična je onoj u "ChessHub Database", no prikazani podaci prilagođeni su korisničkim partijama. Konkretno, stupci u "My Games" tablici uključuju: broj partije, imena bijelog i crnog igrača, rezultat, vrijeme igre za bijelog i crnog igrača te datum partije.

My Analyzes funkcionalnost omogućuje korisnicima pregled njihovih vlastitih analiza partija koje su pohranjene pomoću prozora za analizu. Kao što je ranije objašnjeno, korisniku se nudi mogućnost pohrane analize u *my_analyzes* tablicu korištenjem metode *save_analysis_to_database().* Ako su analize pohranjene, ova funkcionalnost omogućuje pregledavanje tih analiza klikom na ikonu "My Analyzes." Metoda *display_my_analyzes()* dohvaća podatke iz *my_analyzes* tablice i prikazuje ih u tablici, slično kao u ostalim funkcionalnostima. Međutim, u ovom prikazu podaci su specifični za analizu, uključujući broj partije, imena bijelog i crnog igrača, njihov ELO rating, rezultat, naziv turnira, datum i notaciju partije. Kao i kod "ChessHub Database" i "My Games," korisnik može dvostrukim klikom na redak otvoriti prozor za detaljnu analizu.

5. Zaključak

U ovom završnom radu uspješno je razvijena Python desktop aplikacija za igranje i analizu šahovskih partija, koristeći Tkinter za grafičko sučelje i *python-chess* biblioteku za implementaciju šahovske logike i integraciju PGN formata. Primarni cilj aplikacije bio je stvoriti korisničko sučelje koje omogućuje igračima različitih vještina da igraju šah protiv računalnog protivnika, analiziraju svoje partije te pretražuju lokalnu bazu podataka koja sadrži više od 19 000 šahovskih partija.

Jedna od ključnih značajki aplikacije je ploča za analizu, koja korisnicima omogućuje povlačenje poteza, pohranjivanje tih poteza u PGN formatu i pregled različitih varijanti za svaki potez. Također, ploča za analizu sadrži funkcionalnost poput "Reference", koja pretražuje bazu podataka na temelju trenutne pozicije na ploči i pruža korisnicima uvid u partije u kojima se ista pozicija pojavila. Opcija "Add Kibitzer" koristi *chess engine* za evaluaciju pozicije i prikaz najboljih poteza. Dodatno, aplikacija uključuje funkcionalnosti kao što su pregled partija u bazi podataka (ChessHub Database), pregled vlastitih odigranih partija (My Games) i pohranjenih analiza (My Analyzes). Aplikacija integrira *chess engine* i nudi 10 razina snage računalnog protivnika, prilagođavajući se igračima različitih vještina.

Tijekom razvoja aplikacije, posebna pažnja posvećena je optimizaciji pretraživanja velike baze podataka i analizi pozicija. Iako su tijekom izrade naišli izazovi, kao što su kašnjenja u obradi podataka prilikom pretraživanja baze, implementirane metode su uspješno otklonile probleme. Aplikacija tako pruža brz i efikasan pristup analizi šahovskih partija, omogućujući korisnicima pregled i analizu njihovih igara te uvid u partije iz opsežne baze podataka.

Iako aplikacija ne donosi značajan napredak u odnosu na napredne šahovske alate poput ChessBasea, predstavlja vrijedno rješenje za šahiste na različitim razinama znanja, nudeći im intuitivne alate za analizu i poboljšanje svojih šahovskih vještina. Ovaj rad također služi kao odlična polazna točka za daljnja istraživanja i razvoj u području šahovskih algoritama i tehnika analize pozicija.

U budućem radu postoji nekoliko smjerova za unaprjeđenje aplikacije. Jedan od njih je nadogradnja korisničkog sučelja kako bi se omogućila još veća interaktivnost i preglednost. Nadalje, dodavanje opcije igranja online partija protiv drugih igrača pružilo bi novu dimenziju korištenja aplikacije. Također, proširenje funkcionalnosti baze podataka, uključujući integraciju dodatnih šahovskih varijacija i pozicija može značajno unaprijediti sposobnost aplikacije za analizu i edukaciju šahista. Na taj način, aplikacija bi mogla postati sveobuhvatni alat za šahovsku zajednicu, zadovoljavajući potrebe i početnika i naprednih igrača.

Literatura

Chess assessment symbols. (n.d.). Dohvaćeno iz RPB Chessboard: https://rpb-chessboard.yo35.org/documentation/chess-assessment-symbols/

Crowther, M. (24. April 2018). *The Week in Chess*. Dohvaćeno iz TWIC Archive: https://theweekinchess.com/twic

Ensmenger, N. (2021). *Is chess the drosophila of artificial intelligence?* Retrieved from Academia.edu: https://d1wqtxts1xzle7.cloudfront.net/71019047/3fc7ec81458057e6f96de1cba095e84a05c4-libre.pdf?1633205273=&response-content-disposition=inline%3B+filename%3DIs_chess_the_drosophila_of_artificial_in.pdf&Expires=1725734282&Signature=Du4zkz35h87pi8lcG3qVicDKk

Ferreira, D. R. (2013). THE IMPACT OF SEARCH DEPTH ON CHESS PLAYING STRENGTH. *ICGA Journal*, 72.

Norvig, S. J. (2010). *Artificial Intelligence: A Modern Approach (4th ed.).* Retrieved from Texas A&M University: https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf

python-chess: a chess library for Python. (2024). Dohvaćeno iz Python Chess Library: https://python-chess.readthedocs.io/en/latest/

Shaykh Academy. (n.d.). Dohvaćeno iz There are more possible iterations of a game of chess than there are atoms in the known universe.: https://theshaykhacademy.com/general-knowledge/did-you-know/there-are-more-possible-iterations-of-a-game-of-chess-than-there-are-atoms-in-the-known-universe/

Stockfish. (2024). Dohvaćeno iz Download Stockfish for Linux: https://stockfishchess.org/download/linux/

Universal Chess Interface (UCI). (2024). Dohvaćeno iz Shredder: https://www.shredderchess.com/chess-features/uci-universal-chess-interface.html

Popis tablica

Popis slika

Popis priloga

Prilog 1: Prikaz ekrana za učitavanje

```
def show_loading_screen(root):
    loading_screen = tk.Toplevel(root)
    loading_screen.geometry(f"{400}x{200}")
    loading_screen.title("Loading...")
    loading_screen.configure(bg="#F8E7BB")

    loading_screen.grab_set()
    loading_screen.transient(root)
    loading_screen.update()

    loading_label = tk.Label(loading_screen, text="Loading ChessHub... Please wait.",
font=("Helvetica", 16), fg="#660000", bg="#F8E7BB")
    loading_label.pack(expand=True)

    return loading_screen
```

Prilog 2: Prikaz funkcije za inicijalizaciju podataka

```
def initialize_data(root, loading_screen):
    db_path =
'/home/daniel/Desktop/3.godinapreddiplomskogstudija/6.semestar/Zavrsni_Rad/ChessHub_Databas
e/data/database/chess_db.sqlite'

    def table_exists_and_not_empty(conn, table_name):
        cursor = conn.cursor()
        cursor.execute(f"SELECT name FROM sqlite_master WHERE type='table' AND
name='{table_name}';")
        if cursor.fetchone() is None:
            return False

        cursor.execute(f"SELECT COUNT(*) FROM {table_name};")
        row_count = cursor.fetchone()[0]
        return row_count > 0

    if os.path.exists(db_path):
        conn = sqlite3.connect(db_path)
        if table_exists_and_not_empty(conn, 'games'):
            print("Baza podataka i tablica 'games' postoje i nisu prazne.")
        else:
            print("Tablica 'games' ne postoji ili je prazna. Kreiram tablicu i parsiram
PGN.")
            db_thread = ChessDatabase()
            db_thread.conn = conn
            db_thread.parse_pgn_and_store_in_db(config.pgn_file_path)
            db_thread.create_tables()
            db_thread.close_connection()
    else:
        print("Baza podataka ne postoji. Kreiram novu bazu i tablice.")
        db_thread = ChessDatabase()
        db_thread.create_tables()
        db_thread.parse_pgn_and_store_in_db(config.pgn_file_path)
        db_thread.close_connection()

    loading_screen.destroy()
```

Prilog 3: Prikaz funkcije *create_tables*

```
def create_tables(self):
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS games (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                site TEXT,
                date TEXT,
                round TEXT,
                white TEXT,
                black TEXT,
                result TEXT,
                white_elo INTEGER,
                black_elo INTEGER,
                eco TEXT,
                event_date TEXT,
                notation TEXT
            )
        ''')

        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS fens (
                game_id INTEGER,
                move_number INTEGER,
                fen TEXT,
                fen_hash TEXT,
                FOREIGN KEY(game_id) REFERENCES games(id)
            )
        ''')

        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS my_games (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                white TEXT,
                black TEXT,
                result TEXT,
                white_time TEXT,
                black_time TEXT,
                date TEXT,
                moves TEXT
            )
        ''')

        self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS my_analyzes (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            white TEXT,
            black TEXT,
```

```
            result TEXT,
            white_elo INTEGER,
            black_elo INTEGER,
            date TEXT,
            tournament TEXT,
            round TEXT,
            notation TEXT
        )
        ''')
        self.conn.commit()
```

Prilog 4: Prikaz funkcije *parse_pgn_and_store_in_db*

```
    def parse_pgn_and_store_in_db(self, pgn_file_path, batch_size=1000):
        conn = self.connect_to_database()
        cursor = conn.cursor()

        with open(pgn_file_path, "r", encoding="ISO-8859-1") as pgn_file:
            game = chess.pgn.read_game(pgn_file)
            while game:
                exporter = chess.pgn.StringExporter(headers=False, variations=False,
comments=False)
                notation = game.accept(exporter)

                board = chess.Board()
                fens = []

                fen = board.fen()
                fen_hash = hash_fen(fen)
                move_count = 0

                moves = list(game.mainline_moves())

                fens.append({
                    "move_number": move_count,
                    "fen": fen,
                    "fen_hash": fen_hash
                })

                move_count += 1

                for idx, move in enumerate(moves):
                    try:
                        if board.is_legal(move):
                            board.push(move)
                        else:
                            print(f"nelegalan potez {move}. Preskacem partiju")
                            break
                    except AssertionError as e:
                        print(f"Greška prilikom primjene poteza {move}. Trenutni FEN:
{board.fen()}. Greška {e}")
                        break

                    fen = board.fen()
                    fen_hash = hash_fen(fen)

                    fens.append({
                        "move_number": move_count,
                        "fen": fen,
```

```
                    "fen_hash": fen_hash
                })

                move_count += 1

            game_data = {
                "Site": game.headers.get("Site", ""),
                "Date": game.headers.get("Date", ""),
                "Round": game.headers.get("Round", ""),
                "White": game.headers.get("White", ""),
                "Black": game.headers.get("Black", ""),
                "Result": game.headers.get("Result", ""),
                "WhiteElo": game.headers.get("WhiteElo", ""),
                "BlackElo": game.headers.get("BlackElo", ""),
                "ECO": game.headers.get("ECO", ""),
                "EventDate": game.headers.get("EventDate", ""),
                "Notation": notation.strip()
            }

            game_id = self.save_to_gamesTable_database(cursor, game_data)
            self.save_fens_to_database(cursor, game_id, fens)
            game = chess.pgn.read_game(pgn_file)

    conn.commit()
    cursor.close()
    conn.close()
```

Prilog 5: Prikaz funkcije *create_icon_with_button_and_label*

```
def create_icon_with_button_and_label(parent_frame, table_frame, image_path, image_size,
text, command=None, display_table_command=None):
    icon_image = helper_methods.load_and_resize_image(image_path, image_size)
    icon_photo = ImageTk.PhotoImage(icon_image)
    frame = tk.Frame(parent_frame, bg=config.background_color)
    frame.pack(side="left", padx=30)

    if display_table_command:
        icon_button = tk.Button(frame, image=icon_photo, bg=config.background_color,
borderwidth=0, highlightthickness=0, activebackground=config.background_color,
command=lambda: display_table_command(table_frame))
    else:
        icon_button = tk.Button(frame, image=icon_photo, bg=config.background_color,
borderwidth=0, highlightthickness=0, activebackground=config.background_color,
command=command)

    icon_button.image = icon_photo
    icon_button.pack(side="top")
    label = tk.Label(frame, text=text, bg=config.background_color,
fg=config.button_text_color, font=config.label_font)
    label.pack(side="top")
    return icon_button
```

Prilog 6: Prikaz funkcije *select_level i update_level_buttons*

```
    def select_level(level):
        selected_level.set(level)
        update_level_buttons()

    def update_level_buttons():
        for button in level_buttons:
            if button.cget("text") == selected_level.get():
                button.config(bg="#660000", fg="#F2CA5C")
            else:
                button.config(bg="#F2CA5C", fg="#660000")
```

Prilog 7: Prikaz funkcije *init_engine*

```
    def init_engine(self):
        self.engine = chess.engine.SimpleEngine.popen_uci(config.STOCKFISH_PATH)

        config_options = self.level_map[self.level]
        stockfish_skill_level = config_options["skill_level"]

        self.engine.configure({
            "Skill Level": stockfish_skill_level,
            "Threads": config_options["threads"],
            "Hash": config_options["hash"],
            "UCI_LimitStrength": config_options["limit_strength"],
            "UCI_Elo": config_options["elo"],
            "Move Overhead": config_options["move_overhead"],
            "nodestime": config_options["nodestime"]
        })
```

Prilog 8: Prikaz dodijeljenih parametara za svaku razinu igranja protiv računala

```
        self.level_map = {
            "Level 1": {"skill_level": 0, "threads": 1, "hash": 8, "elo": 1320,
"move_overhead": 400, "nodestime": 10, "limit_strength": True},
            "Level 2": {"skill_level": 2, "threads": 1, "hash": 16, "elo": 1400,
"move_overhead": 300, "nodestime": 20, "limit_strength": True},
            "Level 3": {"skill_level": 4, "threads": 1, "hash": 64, "elo": 1600,
"move_overhead": 200, "nodestime": 40, "limit_strength": True},
            "Level 4": {"skill_level": 6, "threads": 1, "hash": 128, "elo": 1800,
"move_overhead": 150, "nodestime": 80, "limit_strength": True},
            "Level 5": {"skill_level": 9, "threads": 2, "hash": 256, "elo": 2000,
"move_overhead": 125, "nodestime": 160, "limit_strength": True},
            "Level 6": {"skill_level": 12, "threads": 2, "hash": 512, "elo": 2200,
"move_overhead": 100, "nodestime": 320, "limit_strength": False},
            "Level 7": {"skill_level": 14, "threads": 4, "hash": 512, "elo": 2400,
"move_overhead": 75, "nodestime": 400, "limit_strength": False},
            "Level 8": {"skill_level": 16, "threads": 4, "hash": 1024, "elo": 2600,
"move_overhead": 50, "nodestime": 500, "limit_strength": False},
            "Level 9": {"skill_level": 18, "threads": 6, "hash": 1024, "elo": 2800,
"move_overhead": 30, "nodestime": 600, "limit_strength": False},
            "Level 10": {"skill_level": 20, "threads": 8, "hash": 2048, "elo": 3000,
"move_overhead": 20, "nodestime": 800, "limit_strength": False}
        }
```

Prilog 9: Prikaz funkcije *quit_game*

```
    def quit_game(self):
        self.pause_timers()
        self.engine.quit()
        self.gameScreenWindow.quit()
        self.root.quit()
        self.root.destroy()

        python = sys.executable
        os.execl(python, python, *sys.argv)
```

Prilog 10: Prikaz funkcije *end_game* i *get_game_result*

```
    def end_game(self, message="Game over"):
        self.pause_timers()
        self.engine.quit()
        self.game_over = True
        result = self.get_game_result()
        self.show_result_menu(result, message)

    def get_game_result(self):
        if self.white_time <= 0:
            return "0-1"
        elif self.black_time <= 0:
            return "1-0"
        if self.board.is_checkmate():
            return "0-1" if self.board.turn == chess.WHITE else "1-0"
        elif self.board.is_stalemate():
            return "1/2-1/2"
        elif self.board.is_insufficient_material():
            return "1/2-1/2"
        elif self.board.is_seventyfive_moves():
            return "1/2-1/2"
        elif self.board.is_fivefold_repetition():
            return "1/2-1/2"
        return "*"
```

Prilog 11: Prikaz funkcije *update_notation*

```
    def update_notation(self, current_move=None):
        self.notation_text.config(state="normal")
        self.notation_text.delete(1.0, tk.END)

        exporter = chess.pgn.StringExporter(headers=False, variations=True, comments=True)
        pgn = self.game.accept(exporter)

        self.notation_text.insert(tk.END, pgn)

        self.fen_dict = {}

        stack = [(self.game, chess.Board(), "1.0", 0)]

        while stack:
            node, board, current_position, variation_index = stack.pop()

            for i, variation in enumerate(node.variations):
                move = variation.move

                pre_move_fen = board.fen()

                if move not in board.legal_moves:
                    print(f"Illegal move: {move} in {board.fen()}")
                    continue

                san_move = board.san(move)
                board.push(move)

                move_idx = self.notation_text.search(san_move, current_position, tk.END)

                if not move_idx:
                    print(f"Potez {san_move} nije pronađen u tekstu.")
                    continue

                if pre_move_fen not in self.fen_dict:
                    self.fen_dict[pre_move_fen] = [(san_move, move_idx, i)]
                else:
                    self.fen_dict[pre_move_fen].append((san_move, move_idx, i))

                current_position = self.notation_text.index(f"{move_idx} +
{len(san_move)}c")
                stack.append((variation, board.copy(), current_position, i))
                board.pop()

        if current_move:
            try:
```

```
                current_fen = self.board.fen()

                if current_fen in self.fen_dict:
                    san_list = self.fen_dict[current_fen]
                    if len(san_list) == 1:
                        san_move, move_idx, var_idx = san_list[0]
                        end_idx = f"{move_idx} + {len(san_move)} chars"
                        self.notation_text.tag_add("highlight", move_idx, end_idx)
                    else:
                        # Ako postoji više poteza s istim FEN-om, usporedimo SAN i
varijaciju
                        for san_move, move_idx, var_idx in san_list:
                            if san_move == self.board.san(current_move):
                                end_idx = f"{move_idx} + {len(san_move)} chars"
                                self.notation_text.tag_add("highlight", move_idx, end_idx)
                                break
                else:
                    print("FEN nije pronađen u rječniku!")
            except AssertionError:
                print(f"Potez {current_move} nije legalan u trenutnoj poziciji.")
                print(self.board)

        self.notation_text.tag_config("highlight", foreground="black", font=("Inter", 16,
"bold"))
        self.notation_text.config(state="disabled")
```

Prilog 12: Prikaz funkcije *update_variation_controls*

```
    def update_variation_controls(self):
        parent_node = self.current_node.parent
        if parent_node:
            current_fen = self.current_node.board().fen()
            current_half_moves = self.get_halfmove_count_from_fen(current_fen)

            filtered_variations = []
            for variation in parent_node.variations:
                temp_board = parent_node.board().copy()

                temp_board.push(variation.move)

                variation_fen = temp_board.fen()
                variation_half_moves = self.get_halfmove_count_from_fen(variation_fen)

                if variation_half_moves == current_half_moves:
                    filtered_variations.append(variation)

            if self.current_node.is_main_variation():
                if len(filtered_variations) > 1:
                    print("Glavna varijanta ima podvarijante.")
                    self.promote_button.pack_forget()
                    self.delete_button.pack_forget()
                    self.demote_button.pack(side="left", padx=5)
                else:
                    print("Glavna varijanta nema podvarijanti.")
                    self.promote_button.pack_forget()
                    self.delete_button.pack_forget()
                    self.demote_button.pack_forget()

            else:
                filtered_variations_second = []
                for variation in self.current_node.variations:
                    temp_board = self.current_node.board().copy()
                    temp_board.push(variation.move)
                    variation_fen = temp_board.fen()
                    variation_half_moves = self.get_halfmove_count_from_fen(variation_fen)
                    if variation_half_moves == current_half_moves:
                        filtered_variations_second.append(variation)

                if len(filtered_variations_second) > 1:
                    print("Podvarijanta s alternativama.")
                    self.promote_button.pack(side="left", padx=5)
                    self.demote_button.pack(side="left", padx=5)
                    self.delete_button.pack_forget()
                else:
```

```
                print("Podvarijanta bez alternativa.")
                print("Prikazujem promote i demote gumbe.")
                self.promote_button.pack(side="left", padx=5)
                self.demote_button.pack_forget()
                self.delete_button.pack(side="left", padx=5)
                self.analysisWindow.update()
        else:
            print("Nema nadređenog čvora, početak partije.")
            self.promote_button.pack_forget()
            self.delete_button.pack_forget()
            self.demote_button.pack_forget()
```

Prilog 13: Prikaz funkcije *get_game_data_for_fen*

```
    def get_game_data_for_fen(self, fen):
        conn =
sqlite3.connect('/home/daniel/Desktop/3.godinapreddiplomskogstudija/6.semestar/Zavrsni_Rad/
ChessHub_Database/data/database/chess_db.sqlite')
        cursor = conn.cursor()

        # Query that finds all games that match the FEN
        cursor.execute('''
            SELECT g.id, g.white, g.black, g.white_elo, g.black_elo, g.result,
g.event_date, g.site, g.date, g.notation
            FROM games AS g
            JOIN fens AS f ON g.id = f.game_id
            WHERE f.fen_hash = ?
        ''', (fen,))

        games = cursor.fetchall()

        cursor.close()
        conn.close()

        return games
```

Prilog 14: Prikaz funkcije *save_analysis_to_database*

```
    def save_analysis_to_database(self, game_data):
        insert_query = '''
            INSERT INTO my_analyzes (white, black, result, white_elo, black_elo, date,
tournament, round, notation)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
        '''
        self.cursor.execute(insert_query, (
            game_data['white'], game_data['black'], game_data['result'],
            game_data['white_elo'], game_data['black_elo'], game_data['date'],
            game_data['tournament'], game_data['round'], game_data['notation']
        ))
        self.conn.commit()
```