Šah kao računalna igra

Rajić, Ivan

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Rajić

ŠAH KAO RAČUNALNA IGRA

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Rajić

Matični broj: 0016140085

Studij: Poslovni sustavi

ŠAH KAO RAČUNALNA IGRA

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Mario Konecki

Varaždin, travanj 2024.

Ivan Rajić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Tema ovog rada je izrada računalne igre šah. Igru sam odlučio izraditi u obliku aplikacije u Visual studiu pomoću razvojnog alata Flutter koristeći Dart kao programski jezik. Ova aplikacija se sastoji od 4 izbora: Igraj, Prvaci, Otvaranja i Zagonetke. Prilikom pritiska na „Igraj“ nudi se izbor igranja protiv računala ili protiv prijatelja. Računalo koristi alpha-beta pruning algoritam prilikom odabira poteza s time da prioritizira poteze koji uključuju uzimanje figura.

U izborniku „Prvaci“ nalazi se lista svih službenih svjetskih prvaka počevši od Wilhelma Steinitza (1886. g.) do sadašnjeg prvaka Ding Lirena. Svaka stranica uključuje sliku svjetskog prvaka i ispod slike tekst koji ukratko opisuje glavne značajke te osobe.

Izbor „Otvaranja“ prikazuje par popularnih otvaranja koji se koriste u igri. Svaka sekcija uključuje sliku otvaranja te ispod slike tekst koji opisuje to otvaranje.

„Zagonetke“ su šahovski problemi koje korisnik ima mogućnost rješavati unutar aplikacije. Izborom zagonetki pojavljuje se pozicija koja zahtjeva odabir pravog poteza. Prilikom pogrešno odigranog poteza pojavljuje se prozor koji obavještava korisnika o netočnom potezu te mu nudi mogućnost ponovnog pokušaja. Neke zagonetke zahtjevaju odabir više točnih poteza prilikom čega računalo automatski igra poteza za suprotnu stranu.

Ovaj rad opisuje razvoj računalne aplikacije šah. U radu su korištene biblioteke za prikaz šahovske table kao i za šahovsku logiku. Implementiran je alpha-beta pruning algoritam koji računalo koristi prilikom odabira najboljeg poteza pri čemu mu pomaže evaluacijska funkcija u kojoj se nalazi vrijednost svake figure. Na dubini 3 računalo je sposobno izračunati kratke kombinacije i odigrati dobar potez u većini pozicija.

Ključne riječi: računalna igra ; programiranje ; algoritam odabira poteza; proces izrade računalne igre

Sadržaj

1. Uvod

Tema ovog rada je izrada računalne igre šah. Prilikom izrade ovakve ili bilo koje druge igre, potrebno je vizualizirati kako bi željena igra trebala izgledati te tek onda odlučiti koji alati su najbolji za realizaciju zamišljenog cilja. Nakon proučavanja mogućnosti za izradu ove igre, na kraju sam odlučio da je razvojni alat Flutter i programiranje u jeziku Dart najbolje rješenje. Napravio sam igru u sklopu aplikacije „Chessmasters“ koja korisniku nudi igru protiv računala koju sam implementirao koristeći alpha-beta podrezivanje za odabir najboljeg poteza. Pored toga, aplikacija nudi mogućnost rješavanja šahovskih zagonetki kao način vježbanja igre te neke korisne informacije o svjetskim šahovskim prvacima kao i o popularnim šahovskim otvaranjima.

Budući da sam i sam ljubitelj ove drevne igre, zanimalo me je kako računala kostantno uspijevaju pobjeđivati najbolje šahiste i na koji način oni računaju u igri kompleksnoj kao što je šah. Iako ne znamo točno koliko je varijanti šahovskih partija moguće, po procjeni američkog matematičara Claude-a Shanonna ta brojka je toliko velika da prevazilazi broj atoma u vidljivom svemiru i iznosi 10^{120}. Ova brojka pokazuje kako je sa sadašnjom tehnologijom brute-force metoda rješavanja šaha nemoguća. Iako su računala sposobna računati mnogo brže i bolje od ljudi, to i dalje nije ni izbliza dovoljno za izračun svih mogućih varijanti što nas dovodi do problema programiranja jakog šahovskog programa.

Tražeći najefikasniji način za izračun poteza, naišao sam na minimax algoritam. Iako je minimax veoma dobar prilikom računanja najboljeg poteza, i dalje ostaje problem dubine i vremena koje je potrebno za cijeli proračun. Srećom, ovaj algoritam se može optimizirati alpha-beta podrezivanjem tako da se ne računaju sve varijante ukoliko je određeni potez definitivno bolji od drugog mogućeg, te se taj drugi potez kompletno odbacuje. Princip na kojem radi ovaj algoritam ću detaljnije objasniti u ovom radu.

Iako završena igra funkcionira na željeni način, uvijek ostaje prostor za napredak i poboljšanja. Moguće je testirati realnu snagu programa puštajući ga da igra protiv ljudi i vidjeti ključne, ponavljajuće greške te ih iskorijeniti. Ostala poboljšanja u igri se mogu napraviti nakon puštanja aplikacije i uvida u iskustva korisnika.

2. Planiranje razvoja računalne igre

Da bi se napravila računalna igra kao što je šah, potrebno je osnovno poznavanje šahovskih principa i strategija kao i znanje programiranja kako bi se željena igra implementirala. Osim toga, potrebno je proučiti tehnologije u kojima se može napraviti jedna ovakva igra te na kraju naučiti raditi u izabranoj tehnologiji.

Prilikom planiranje izrade računale igre, potrebno je osnovno poznavanje šahovske igre:

1. **Cilj**: Cilj igre je matiranje protivničkog kralja
2. **Figure i potezi**: potrebno je znati gdje se svaka figura nalazi na početku kao i kako se kreće
3. **Pravila**: Potrebno je znati sva pravila igre

Šah je vrsta računalne igre u kojoj postoji mogućnost igranja jedan na jedan odnosno igranja protiv prijatelja, igranje protiv računala, rješavanja zagonetki i igranje protiv pravih ljudi koristeći različit tempo igre kao što je blitz ili rapid. Većina dostupnih aplikacija nudi jednu ili nekoliko ovih mogućnosti. Chess.com i lichess.org su najpopularnije web stranice za online igranje šaha i nude sve navedene mogućnosti. Navedene stranice koriste Elo rejting sustav za izračun snage igrača ili računala. Elo snaga računala protiv kojih se može igrati varira od potpunog početnika pa do veoma jakog bota koji ne može pobijediti ni svjetski šahovski prvak. Moj cilj je bio napraviti bot koji se nalazi na sredini između ta dva spektra, što znači igrač prosječnog Elo rejtinga. Pored toga, cilj mi je dodati opciju igranja protiv prijatelja kao i rješavanja šahovskih zagonetki.

Nakon proučavanja tehnologija u kojima bih mogao ostvariti zamišljene ciljeve, odlučio sam se za razvojni alat Flutter i Dart kao programski jezik. Budući da je igranje šaha putem aplikacija puno jednostavnije u 2D obliku, a Flutter nudi upravo to kao i ostali niz mogućnosti koje su mi potrebne za realizaciju zamišljene igre, odluka je pala na njega. Iako je recimo chess.com optimiziran za desktop kao i za mobitel, ja sam odlučio napraviti mobilnu aplikaciju koristeći Android Studio mobilni emulator i Visual Studio Code kao uređivač izvornog koda u kojem sam napravio cijeli programski dio.

2.1. Šah kao igra

Šah je strateška igra na tabli sa 64 polja (8 x 8) koja spada pod kategoriju abstraktne strateške igre. To je igra za dvije osobe gdje jedan igrač ima bijele, a drugi crne figure. Svaki igrač počinje igru sa 16 figura,a to su: kralj, dama, dva topa, dva skakača, dva lovca i osam pješaka. Igra može završiti pobjedom jednog od igrača (šah-matom) ili remijem. Remi se može dogoditi u sljedećim slučajevima: nedovoljno materijala za šah-mat, tri puta ponovljena pozicija, patom (igrač ne može odigrati ni jedan legalan potez a kralj mu nije pod šahom) ili pravilom 50 poteza gdje ni jedan igrač nije pojeo ni jednu figuru niti pomjerio nijednog pješaka u rasponu od 50 poteza. Bijeli povlači prvi potez te iz tog razloga ima malu prednost. Kako šah nije riješena igra, ne možemo sa sigurnošću znati je li prednost bijelog zbog prvog poteza dovoljna za pobjedu ili crni može remizirati sa najboljom mogućom igrom obe strane. Generalno je prihvaćeno da crni ima dovoljno resursa da neutralizira početnu inicijativu bijelog i s vremenom izjednači poziciju. U šahu nema skrivenih informacija i oba igrača imaju cijeli pregled igre. Nema ni elementa sreće pa je tako igra kompletno bazirana na taktičkim i strateškim vještinama. Iako su pravila jednostavna, šah je poznat po svojoj ogromnoj kompleksnosti.

Također postoji mnogo varijanti šaha koje su bazirane na šahu. Te varijante mogu sadržavati druga pravila, različit početni položaj figura, nove vrste figura ili čak elemente sreće i skrivenih informacija. Jedna od najpopularnijih varijanti šaha je Fischer Random ili Chess960 koju je izmislio jedanaesti svjetski šahovski prvak Bobby Fischer. U varijanti Chess 960 šahovska tabla, figure i pravila su isti kao i u običnom šahu sa razlikom da su figure na zadnjem redu postavljene nasumično. Odatle i naziv Chess960 jer postoji točno 960 jedinstvenih mogućih početnih pozicija u ovoj varijanti. Fischer random je varijanta koju je predložio Bobby Fischer kako bi se smanjila teoretska zasićenost te povećala kreativnost.

Iako postoji još mnogo varijanti šaha, u ovom radu bavit ću se klasičnim šahom sa standardnim pravilima. U natjecateljskom šahu se koristi i šahovski sat kako bi se ograničilo vrijeme trajanja partije. U slučaju da jednom igraču istekne vrijeme, on automatski gubi partiju. Tri su glavne podjele vrste šaha s obzirom na tempo igre: standardni, ubrzani ili rapid i brzopotezni ili blitz šah. U standardnom tempu oba igrača imaju po 90 minuta, u rapidu od 10 do 60 minuta, a u blitzu ispod 10 minuta. Internet je brzopotezni šah doveo na drugi nivo te danas imamo bullet ali i hyperbullet gdje igrači imaju minutu ili samo 30 sekundi za cijelu partiju. Osim šahovskog znanja potrebna je i vještina preciznog korištenja miša.

2.1.1. Računalna igra šah kroz povijest

Turbochamp je prvi šahovski program kojeg je kreirao Alan Turing 1948. godine. Iako ga nijedno računalo tada nije moglo pokrenuti, Turing je ručno simulirao program i tako igrao poteze. Iako ne dobro, program je bio sposoban igrati šah i smatra se početkom primjene umjetne inteligencije u šahu.

Kako su računala postajala naprednija 1950-tih godina počinje utrka za kreiranje najboljeg šahovskog programa. U to vrijeme programi su bili znatno lošiji od profesionalnih šahista ali su postavili temelj za moderne programe.

1974. godine održano je prvo računalno šahovsko prvenstvo u Stockholmu na kojem je pobijedio računalni program po imenu Kaissa. Kaissa je program razvijen u Moskvi 1970. i na prvenstvu je pobijedio sve druge programe.

Deep Blue je šahovski program koji je napravljen 1996. godine. Za njega je napravljeno superračunalo na kojem je pokrenut u meču protiv tadašnjeg svjetskog prvaka Gari Kasparova. Iako je Kasparov pobijedio meč 4-2, sjedeće godine nadograđena verzija pobjeđuje Kasparova 3.5-2.5. Pobjeda umjetne inteligencije nad čovjekom 1997. godine postaje povijesno važan događaj i bitna prekretnica u samom razvoju umjetne inteligencije [6].

Slika 1: Gari Kasparov vs Deep Blue

(Izvor: https://www.kasparov.com/timeline-event/deep-blue/, 2024.)

2.2. Faze razvoja računalne igre

Razvoj ove računalne igre sam podijelio na 5 glavnih faza:

1. **Planiranje i dizajn**
2. **Programiranje aplikacije**
3. **Implementacija algoritma i evaluacijske funkcije**
4. **Optimizacija i ispravljanje grešaka**
5. **Testiranje**

U fazi planiranje i dizajn potrebno je provesti istraživanje tematike i donijeti bitne odluke glede izrade aplikacije kao što su alati i metode pomoću kojih ćemo izraditi igru te vizualizirati cjelokupni izgled igre. U ovoj fazi potrebno je proučiti slične igre koje su već kreirane te pogledati na koji način su razvijene kako bi se dobila ideja o tome kako napraviti vlastitu igru. Prilikom gledanja drugih igara iz kojih uzimamo bitne ideje, ključno je razumjeti pomoću kojih alata su izrađene i jesmo li u mogućnosti pristupiti tim alatima. Također, moramo utvrditi koji tip igre želimo napraviti, koja nam je ciljana grupa korisnika, za koju platformu želimo napraviti igru i ključna svojstva koja igra treba imati.

Programiranje aplikacije predstavlja pisanje koda koji igru čini funkcionalnom te također određuje i sam dizajn igre. Ova faza obuhvaća i učenje programskog jezika Dart kao glavnog jezika za Flutter u kojem se realizira igra. Službena dokumentacija je dobar početak gdje se možemo upoznati sa osnovama Fluttera. Osim toga, velika pomoć su online tečajevi i YouTube kanali kojim se može besplatno pristupiti. Zahvaljujući velikoj grupi korisnika, Stack Overflow je zajednica u kojoj se može pronaći odgovor na svaku nedoumicu. Ovo su neki od načina pomoću kojih se može naučiti raditi u Flutteru, a ima ih još. Ja smatram da je najbolji način za učenje izrada vlastitog projekta.

U trećoj fazi se programira algoritam koji računalo koristi prilikom računanja najboljeg poteza, a za to mu pomaže evaluacijska funkcija koja sadrži informacije o vrijednosti svake figure. Iako ova faza tehnički spada pod programiranje aplikacije, ovo je ipak programiranje računala koje samostalno igra poteze te je to jedno veliko svojstvo aplikacije. Kako je ovo bitna cjelina na kojoj se stavlja veliki fokus u ovom radu, odlučio sam je postaviti kao zasebnu fazu. Ova faza uključuje traženje optimalnog algoritma za računanje poteza. Prilikom traženja najboljeg algoritma, potrebno je obratiti pažnju na to koliko je algoritam težak odnosno koliko procerske moći i vremena zahtjeva za svoj izračun. To je bitno da se ne bi implementirao algoritam za koji računalo uzima previše vremena tijekom kojeg je korisnik primoran dugo čekati. Dobra evaluacijska funkcija može poslužiti na način da se optimizira funkcioniranje algoritma za izračun poteza.

Nakon završene aplikacije, u četvrtoj fazi, uočavaju se greške u dizajnu ili performansama te se radi na njihovom otklanjanju kao i potencijalnom optimiziranju aplikacije. Neke moguće greške prilikom izrade aplikacije su: loše korisničko iskustvo, fragmentacija, loša sigurnost i slabe performanse. Korisničko iskustvo se poboljšava tako što se kreira jednostavna i intuitivna igra prilagođena korisniku. Fragmentacija je problem gdje igra nije kompatibilna sa različitim uređajima, operacijskim sustavima ili veličinom ekrana. Ovaj problem se rješava na način da se osigura da je igra kompatibilna na različitim platformama i uređajima te je na njima treba testirati. Problem sa sigurnošću može biti slaba enkripcija ili loš način za spremanje podataka. Ovo se rješava pojačavanjem enkripcije i poboljšavanjem sigurnosti spremanja podataka. Slabe performanse uključuju korištenje previše resursa, brzo trošenje baterije ili sporo učitavanje. Potrebno je vidjeti gdje su najslabije performanse i optimizacijom koda riješiti problem.

Na samom kraju se treba izvršiti testiranje, a to uključuje službeno puštanje igre te prikupljanje što više korisničkog iskustva u pogledu grešaka ili bugova, nedostataka ili prijedloga za poboljšanja na kojime se onda ponovno radi i igra optimizira u svrhu poboljšavanja korisničkog iskustva. Postoje više vrsta testiranja, a neke od njih su: funkcionalno testiranje, testiranje performansi, sigurnosni testovi, testovi kompatibilnosti itd. Svi ovi testovi su bitni kako bi se osiguralo da igra funkcionira na željeni način sa dobrim performansama, da je dobro korisničko iskustvo i sigurnost kao i da je kompatibilna sa različitim platformama i uređajima na kojima se može igrati.

2.3. Vrsta igre i dizajn

Postoji više vrsta aplikacija kao što su edukativne, zabavne, igre, društvene, aplikacije za produktivnost itd. Moj cilj je bio napraviti aplikaciju koja će prije svega biti igra zabavnog karaktera ali će također uključivati i edukativne ili informativne elemente.

Da bih postigao da igra bude zabavna i edukativna ili informativna, osim opcije igranja protiv računala ili prijatelja, uključio sam opcije u kojima dajem zanimljive podatke o svjetskim prvacima kao i o šahovskim otvaranjima.

U sekciji o prvacima, prikazana je slika svakog službenog svjetskog prvaka, od koje do koje godine je bio prvak te tekst koji ukratko opisuje njegova glavna postignuća, stil igre ili neke poznate činjenice. Moguće je listati svjetske prvake prema sljedećem ili se vraćati na prethodnog svjetskog prvaka kao i izlazak iz ove sekcije navigacijom nazad koja se nalazi u gornjem lijevom kutu.

Sekcija Otvaranja funkcionira na sličan način kao i sekcija Prvaci, a korisnik može saznati bitne informacije o najpopularnijim šahovskim otvaranjima. Također je omogućena navigacija između otvaranja kao i vraćanje na glavnu stranicu.

Na samom kraju sam želio omogućiti korisniku da razvija taktičke vještine tako što će rješavati postavljene zagonetke i vježbati uočavanje poznatih obrazaca. Ovaj dio spada pod edukativne svrhe budući da korisnik ima priliku rješavati pozicije koje mogu doći u pravim partijama. To je moguće u sekciji Zagonetke u kojoj se pred korisnika postavlja pozicija i traži da pronađe najbolji mogući potez. Mogućnost pokušaja je neograničen, a većina zagonetki zahtijeva kratke kombinacije od 3 poteza. Prilikom zagonetki koje zahtjevaju više poteza, računalo je programirano da automaski odigra potez za protivnika.

Iako je na samom početku zamišljeni dizajn bio takav da uključuje 4 navedene sekcije na početnoj stranici, prilikom izrade igre javila se potreba za proširenjem dizajna u sekciji Igraj budući da korisnik može birati igru između prijatelja ili računala kao i boju figuru u slučaju igre protiv računala. Dodana su 2 izbornika za izbor protivnika i odabir figura koja se nalazi u sekciji Igraj.

2.3.1. Igra u 2D ili 3D obliku

Izbor između 2D ili 3D prikaza table i figura za igru uveliko ovisi o tome što tko preferira kao i o specifičnim svojstvima koje igrač želi imati u online igri. Razlike u 3D i 2D prikazu su sljedeće:

2D šah:

- **Jasnoća**: 2D prikaz preferira velika većina igrača iz razloga što on pruža čist i jednostavan prikaz šahovske table i figura. Ovo također omogućava lakšu vizualizaciju i računanje pri igri.
- **Brzina:** Moguće je brže igrati na 2D prikazu jer je puno lakše razlikovati figure i polja.
- **Jednostavnost:** Budući da je 2D prikaz jednostavniji i iziskuje manje resursa, mnogo je bolji za igrače koji koriste stariju tehnologiju ili imaju slabiju internetsku vezu.

3D šah:

- **Realizam:** 3D prikaz omogućuje realističnije prikaz table i figura tako da igrači imaju doživljaj kao da igraju na pravoj fizičkoj tabli s pravim figurama.
- **Estetika:** Neki igrači vole vizualni prikaz 3D figura sa mogućnostima rotacije i zumiranja.
- **Detalji:** Za neke igrače 3D prikaz je više vjerodostojan i zanimljiviji zbog različitih detalja koji se mogu dodati u ovom obliku u odnosu na 2D oblik.

Iako oba prikaza nude neke različite stvari u online igranju šahu, velika većina online stranica kao i aplikacija preferira 2D prikaz koji je popularan među velikom većinom igrača. Mislim da je glavni razlog za to što 2D prikaz nudi veoma jasan i čist pogled na tablu i figure te estetski izgled nije od velikog značaja u ovoj igri iako je moguće promijeniti određene stvari i u 2D prikazu (izgled figura, boju šahovske table…) Ja sam se odlučio za 2D prikaz jer nudi jasnoću i jednostavnost te je mnogo lakše igrati šah online u ovom obliku. Sljedeća slika prikazuje vizualnu razliku između ova dva navedena prikaza igre.

Slika 2: 2D vs 3D prikaz igre

(Izvor: https://steamcommunity.com/sharedfiles/filedetails/?id=3049863218, 2024.)

3. Alati potrebni za razvoj igre

Nakon što se odlučimo koju vrstu igre ćemo napraviti, potrebno je odlučiti se i za način na koji ćemo je napraviti odnosno koje alate ćemo koristiti pri njenoj izradi. Budući da sam se ja odlučio za Flutter kao najoptimalnije rješenje, ujedno sam odlučio koristiti i Android studio odnosno Android SDK (Software Development Kit) koji mi je potreban za pokretanje Android emulatora koji oponoša karakteristike klasičnog Android uređaja. Kako Flutter koristi Dart kao glavni programski jezik te je optimiziran za Visual Studio Code, imam glavne alate koje ću koristiti pri izradu svoje mobilne igre.

Dart je objektno orijentirani programski jezik koji je razvio Google. Njegova glavna uloga je razvoj velikih, kompleksnih aplikacija. Iako ima dosta jednostavnu sintaksu, sličan je programskim jezicima C#, Java i Javascript. Osim sintakse, prednosti ovog jezika za razvoj mobilnih aplikaciju su: brzo i efikasno izvođenje koda, razvoj aplikacija za iOS i Android sa istom bazom koda, dobra podrška za izradu widgeta što omogućava laku i jednostavnu integraciju sa Flutterom kao i podrška za objektno orijentirano programiranje.

Prednosti korištenja Visual Studio Code-a u odnosu na Visual Studio razvojno okruženje jest u tome što VS Code ima višeplatformsku kompatibilnost pa tako se može lako pokrenuti na Windowsu, Linuxu i macOS-u. Osim toga, VS Code je poznat po brzim performansama, pogotovo u usporedbi sa Visual Studio-m. Lako se pokreće i brzo izvršava naredbe. Također je dosta jednostavniji, a zahvaljujući velikom broju ekstenzija, po potrebi se mogu instalirati novi jezici, debugger-i i različiti alati za pomoć pri izradi aplikacije.

3.1. Flutter 3

Flutter je razvojni alat otvorenog koda kojeg je razvio Google za razvoj multiplatformskih aplikacija iz jedne baze koda. Flutter 3 podržava Android, iOS, Windows, Linux, macOS i web aplikacije. Osim navedenih, neke od glavnih značajki Fluttera su:

- **Programski jezik Dart** - Optimiziran za izradu korisničkih sučelja
- **Widgeti**- Flutter je jezik koji za sve koristi widgete koji su osnovne građevne jedinice korisničkog sučelja unutar Flutter aplikacija
- **Hot reload** – korisna funkcija koja omogućuje trenutno uočavanje promjena u kodu bez potrebe za ponovnim pokretanjem aplikacije
- **Performanse** – Flutter aplikacije se kompajliraju u izvorni ARM kod i za cilj imaju jedinstveno korisničko iskustva na Android kao i na iOS uređajima
- **Zajednica korisnika** – Flutter ima veliku zajednicu aktivnih korisnika što omogućava lakše učenje

Slika 3: Flutter razvojni alat

(Izvor: https://www.appify.digital/post/flutter-app-development, 2024.)

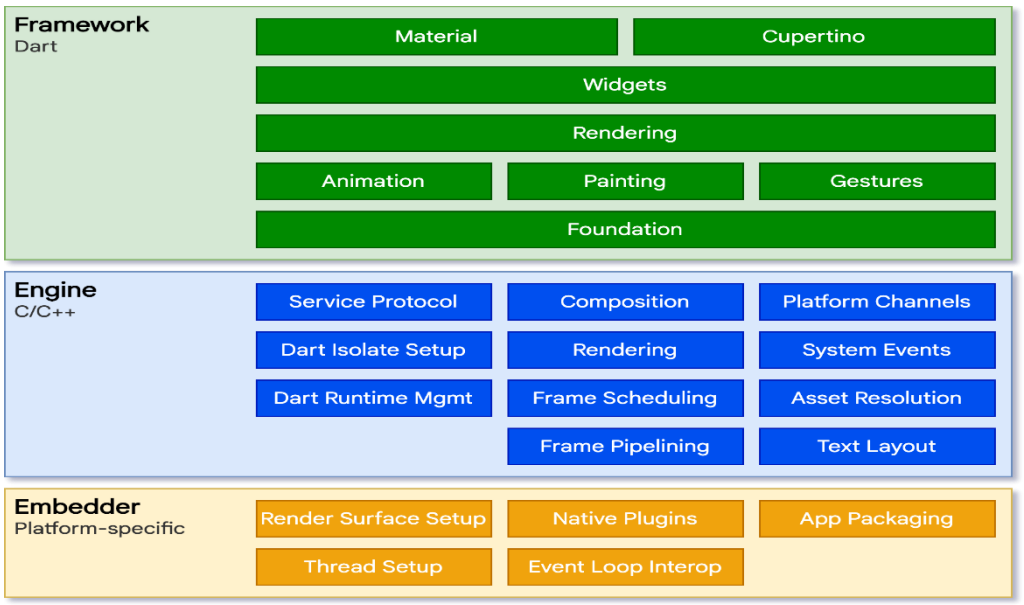
Widgeti su osnovni razvojni elementi korisničkog sučelja aplikacije. Oni formiraju hijerarhiju baziranu na kompoziciji gdje se svaki widget nalazi unutar svog roditeljskog widgeta i od njega uzima kontekst. Postoje dvije vrste widgeta, a to su Stateless i Stateful widgeti. Stateless widgeti ne sadrže promjenjivo stanje tj. ne mijenjaju se kroz vrijeme. Primjer ovakvih widgeta su: ikona, tekst i gumb. Stateful widgeti, s druge strane, sadrže stanje koje se može promijeniti tijekom životnog vijeka widgeta kao na primjer checkbox.

Flutter se sastoji od više arhitekturnih slojeva, a viši slojevi koriste funkcionalnosti nižih slojeva.

Framework (Dart) sloj je najviši sloj napisan u Dart-u i to je onaj sloj kojim se služe programeri koji rade u Flutteru. Ovaj sloj, koji koristi programski jezik Dart, je organiziran u više biblioteka koji pružaju funkcionalnosti kao što su animacije, renderiranje, izgled, izradu logike aplikacije [1].

Engine (C/C++) je sloj koji dolazi nakon Framework sloja. Ovaj sloj je primarno napisan u programskom jeziku C++ i pruža servise poput grafičkog renderiranja (koristeći Skia grafičku biblioteku), raspored teksta i ostali pribor za razvijanje softvera [1].

Embedder sloj je najniži sloj koji sadrži funkcionalnosti niske razine koje su specifične platformi, što znači da je kod pisan u različitim programskim jezicima za Android, iOS, Windows itd [1].



Slika 4: Flutter arhitekturni slojevi

(Izvor: https://docs.flutter.dev/resources/architectural-overview, 2024.

3.1.1. Programski jezik Dart

Dart je programski jezik koji je kreirao Google. Dizajniran je za široku skupinu aplikacija uključujući web i mobilne aplikacije. Neke osnovne stvari o ovome jeziku su:

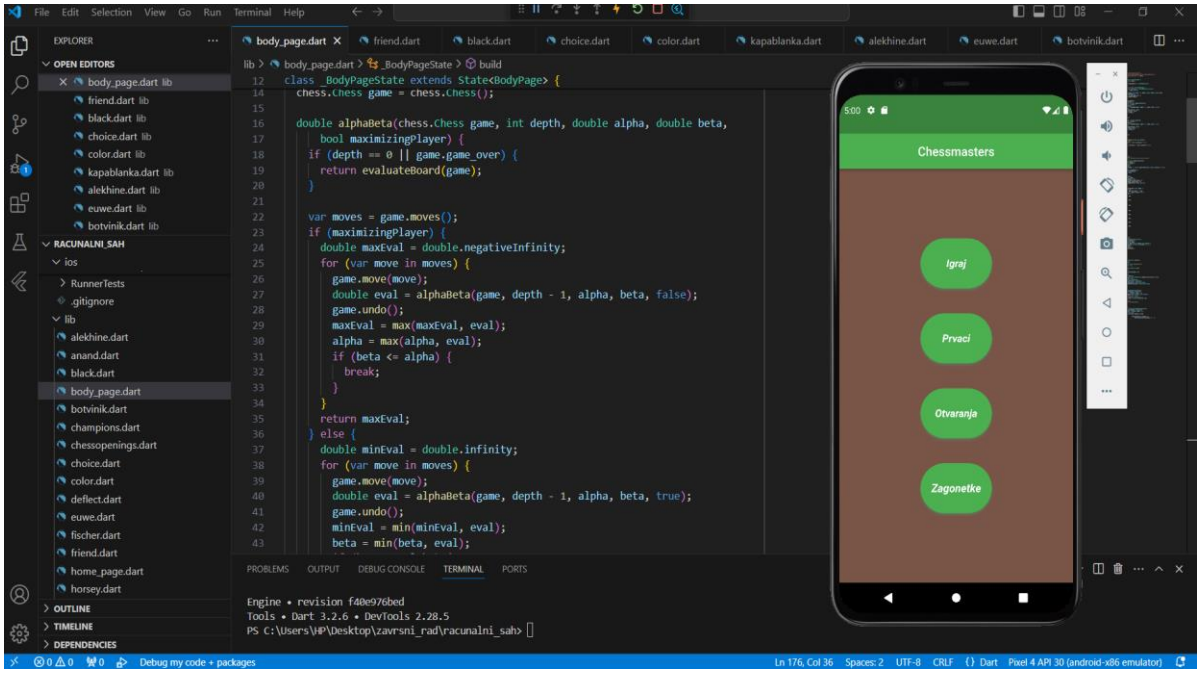
- **Objektno orijentiran:** Dart se bazira na klasama te je objetno orijentiran sa sintaksom koja sliči programskom jeziku C.
- **Sakupljanje smeća:** Dart ima automatsko upravljanje memorijom putem metode sakupljanja smeća (garbage collection).
- **Višeplatformski jezik:** Dart se može koristiti za razvoj web, mobilnih, serverskih i desktop aplikacija.
- **Asinkrono programiranje:** Dart podržava asinkrono programiranje sa svojstvima poput `async` i `await`.
- **Način kompajliranja:** Dart se prevodi u ARM, x64 i RISC – V mašinski kod za mobilne, desktop i backend aplikacije. Također se može kompajlirati u JavaScript i WebAssembly za web aplikacije.
- **Podržava više paradigmi:** Dart podržava funkcionalno, imperativno, objektno orijentirano programiranje.
- **Bogata biblioteka:** Dart dolazi sa veoma bogatom standardnom bibliotekom koja pruža širok raspon ugrađenih funkcija i paketa. Ovo omogućava lakše izvođenje zadataka bez oslanjanja na biblioteke iz drugog izvora.
- **Hot reload:** Slično Flutteru, Dart također podržava hot reload koji programeru omogućuje da automatski vidi promjene u kodu bez potrebe za ponovnim pokretanjem aplikacije.

Ovo su osnovna svojstva programskog jezika Dart koja pokazuju njegove prednosti. Pored navedenih, Dart ima jako dobre performanse te je veoma pouzdan što ga čini jednim od najboljih izbora za izradu različitih vrsta aplikacija [2].

3.2. Visual Studio Code

Visual Studio Code (VS Code) je besplatni uređivač izvornog koda kojeg je razvio Microsoft. Prilagođen je za Windows, Linux, i macOS [3]. Glavne karakteristike VS Code-a su:

- **Višejezična podrška:** VS Code podržava širok raspon programskih jezika kao što su Python, Java, JavaScript, C++, C# i mnoge druge.
- **Sučelje prilagođeno korisniku:** Intuitivno sučelje koje olakšava korisniku da piše, uređuje i razumije kod.
- **Ekstenzije:** Ima mnogo ekstenzija koje značajno pomažu korisniku
- **IntelliSense:** Značajna funkcija koja dovršava kod i pomaže korisniku
- **Debugger:** VS Code dolazi sa moćnim programom za pronalaženje pogrešaka



Slika 5: Korisničko sučelje „ Visual Studio Code“

(Izvor: autor, 2024.)

3.3. Android Studio

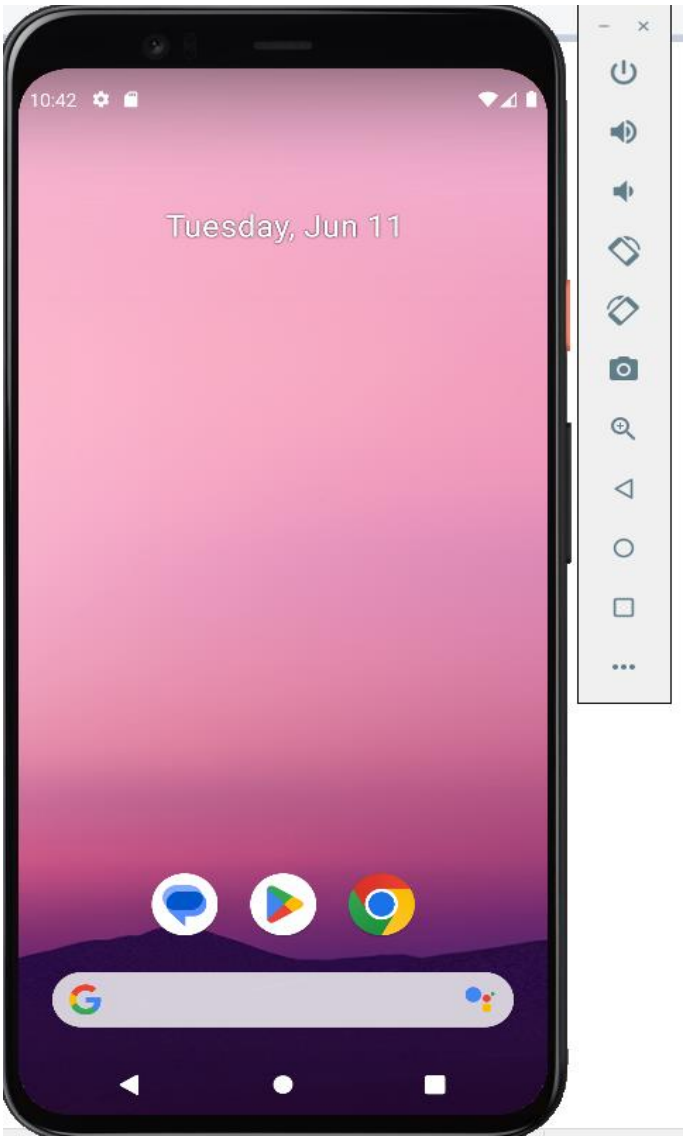
Android Studio je službeno integrirano razvojno okruženje (IDE) za razvoj Android aplikacija. Razvio ga je Google, a baziran je na IntelliJ IDEA razvojnom okruženju koje je veoma popularno za programske jezike poput Jave i Kotlina. Android Studio pruža mnogo alata koji služe za dizajn, razvoj, testiranje i objavljivanje Android aplikacija [4].

Android studio IDE ima dosta sličnosti sa Visual Studio Code uređivačem pa tako sadrži: inteligentni uređivač koda, fleksibilan razvojni sistem, istovremeno uređivanje, Android Emulator i drugo. Inteligentni uređivač koda koristi svojstva slična IntelliSense-u pa tako predlaže osnovni i pametni završetak koda ali koristi i automatski završetak koda koji je potpomognut umjetnom inteligencijom. Razvojni sistem koji se koristi je Gradle, a to je fleksibilni, razvojni sistem otvorenog koda. Istovremeno uređivanje je funkcija koja je veoma bitna te osigurava promjene na emulatoru i fizičkim uređajima u stvarnom vremenu. Android Emulator predstavlja digitalnu kopiju Android uređaja koja simulira svojstva pravog fizičkog uređaja te tako omogućuje programerima testiranje i debugiranje aplikacije bez potrebe za pravim fizičkim uređajem [4].

Slika 6: Android Studio logo

(Izvor: https://logowik.com/android-studio-2021-logo-vector-svg-png-free-download-27580.html, 2024.)

Android Studio SDK (Software Development Kit) je set alata i bibilioteka za razvoj Android aplikacija i to je ono što je meni bilo zaista potrebno u Android Studi-u. Jedan od najbitnijih alata koji mi je potreban za razvoj igre jest Android Emulator. Kao što sam već naveo, Emulator simulira funkcije i svojstva pravog uređaja te tako znatno olakšava posao izrade aplikacije [4].



Slika 7: Android Emulator

Izvor: autor

3.4. Pub.dev

Pub.dev je službeni repozitorij paketa za Dart i Flutter kojeg je razvio Google tim za Flutter te ga oni i održavaju. Služi kao glavna točka gdje programeri mogu otkrivati, dijeliti i koristiti pakete u svojim Dart i Flutter projektima. Pub.dev sadrži tisuće dostupnih paketa otvorenog koda te pruža uvide u kvalitetu paketa i njihovu popularnost. Ovaj repozitorij mi je od velikog značaja budući da sam iz njega preuzeo widget za prikaz šahovske table kao i biblioteku koja sadrži pravila, logiku poteza ali i omogućava konverziju u formate FEN i PGN.

Chess 0.7.0 i flutter_chess_board 1.0.1 su dva paketa koja sam preuzeo iz Pub.dev repozitorija.

```
import 'package:flutter/material.dart';
import 'package:flutter_chess_board/flutter_chess_board.dart';
import 'package:chess/chess.dart';
```

Na ovaj način uvozim potrebne pakete. Njihova instalacija se izvršava sljedećim naredbama:

```
$ flutter pub add flutter_chess_board
$ flutter pub add chess
```

Ove naredbe dodaju dvije linije u paketu pubspec.yaml unutar Fluttera:

```
dependencies:
   flutter_chess_board: ^1.0.1
    chess: ^0.7.0
```

S time smo uspješno preuzeli i instalirali navedene pakete unutar Flutter projekta. Kako bi prikazali šahovsku tablu, osim preuzimanja `flutter_chess_board: ^1.0.1` paketa, potrebno je napisati određeni kod.

```
import 'package:flutter/material.dart';
import 'package:flutter_chess_board/flutter_chess_board.dart';


class BodyPage extends StatefulWidget {
  const BodyPage({super.key});
  @override
  State<BodyPage> createState() => _BodyPageState();
}


class _BodyPageState extends State<BodyPage> {
  ChessBoardController controller = ChessBoardController();
  chess.Chess game = chess.Chess();


@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Chessmasters'),
        centerTitle: true,
      ),
      backgroundColor: Colors.brown,
      body: Column(
        children: [
          const SizedBox(height: 30),
          Expanded(
            flex: 4,
            child: Padding(
              padding: const EdgeInsets.symmetric(horizontal: 16.0),
              child: Center(
                child: ChessBoard(
                  controller: controller,
                  boardColor: BoardColor.brown,
                  boardOrientation: PlayerColor.white,
                ),
              ),
            ),
          ),
```

Ovim kodom prikazujemo šahovsku tablu koja je smeđe boje i okrenuta je prema igraču sa bijelim figurama. U kodu je dodan AppBar koji ispisuje tekst 'Chessmasters' odnosno ime igre.

`ChessBoardController` u ovome kodu igra važnu ulogu jer on drži cijelu igru i moguće je raditi različite stvari pomoću njega [5]:

- **Učitati novu partiju**

```
controller.loadPgn('demo PGN')

controller.loadFen('demo FEN')
```

- **Odigrati potez**

```
controller.makeMove(from: 'e2', to: 'e4')
```

- **Vratiti potez**

```
controller.undoMove()
```

- **Provjeriti stanje igre**

```
controller.isCheckMate()

controller.isDraw()

controller.isStaleMate()

controller.isThreefoldRepetition
```

- **Brojati poteze**

```
controller.getMoveCount()

controller.getHalfMoveCount()
```

4. Programiranje igre

U ovom dijelu se vrši programiranje igre u odabranom alatu. Za početak, cilj mi je napraviti početnu stranicu u kojoj se nalazi izbornik u kojem korisnik bira željenu opciju.

```
class HomePage extends StatelessWidget {
  const HomePage({super.key});

  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        width: double.infinity,
        height: double.infinity,
        color: Colors.brown,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              style: ElevatedButton.styleFrom(
                shadowColor: Colors.greenAccent,
                elevation: 3,
                shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(40.0)),
                minimumSize: const Size(120, 80),
              ),
              onPressed: () {
                Navigator.of(context).push(
                  MaterialPageRoute(
                    builder: (BuildContext context) {
                      return const Choice();
                    },
                  ),
                );
              },
              child: const Text(
                'Igraj',
                style: TextStyle(
```

```
            fontSize: 17,
            fontStyle: FontStyle.italic,
            fontWeight: FontWeight.bold),
      ),
    ),
```

U ovom dijelu koda kreiram gumb koji sadrži tekst „Igraj“ i koji vodi do drugog zaslona putem metode Navigator.of(context).push(route) koji je standardna metoda za prebacivanje na drugi zaslon u Flutteru.

Metoda build vraća widget koji definira izgled stranice. Center widget u sebi sadrži Container widget koji zauzima širinu i visinu ekrana te ima smeđu pozdadinu. Unutar fluttera, većina stvari se radi tako što se stavlja widget na widget i iz tog razloga oni imaju ogromnu ulogu. Container sadrži druge bitne widgete za pozicioniranje, boju i veličinu. Column widget je widget za pozicioniranje koji je centriran na glavnoj osi (MainAxisAlignment.center).

ElevatedButton je dizajn gumba koji sam izabrao za sve gumbove u ovoj aplikaciji. Dizajn odnosno Style definira stil gumba, uključujući boju, oblik i veličinu.

OnPressed je funkcija koja se poziva kada se gumb pritisne. U ovom slučaju, pritiskom na gumb „Igraj“ funkcija vraća stranicu „Choice“.

Ostale sekcije „Prvaci“, „Otvaranja“ i „Zagonetke“ su napravljene na isti način s tim da je izmijenjen tekst gumba kao i ime klase kojoj vodi Navigator.

Slika 8: Početni zaslon „Računalni šah“

(Izvor: autor, 2024.)

4.1. Minimax i Alpha-beta podrezivanje

Minimax algoritam je metoda odlučivanja u teoriji igara i umjetnoj inteligenciji koja se koristi za optimizaciju odluka u igrama s dva igrača, na primjer šah, dama, križić-kružić itd. Postoje dva igrača:

1. Maximizing Player – pokušava dobiti najveću moguću vrijednost

2. Minimazing Player – pokušava dobiti najmanju moguću vrijednost

Algoritam kreira stablo koje predstavlja sve moguće poteze do određene dubine (broj poteza unaprijed) gdje svaki čvor predstavlja trenutno stanje odnosno ocjenu pozicije.

Počevši od korijena stabla, algoritam prolazi kroz sve moguće poteze rekurzivno i kreira grane za svaki potencijalni događaj odnosno potez. Ishod svakog poteza se ocjenjuje korištenjem evaluacijske funkcije koju ću detaljnije opisati u nastavku.

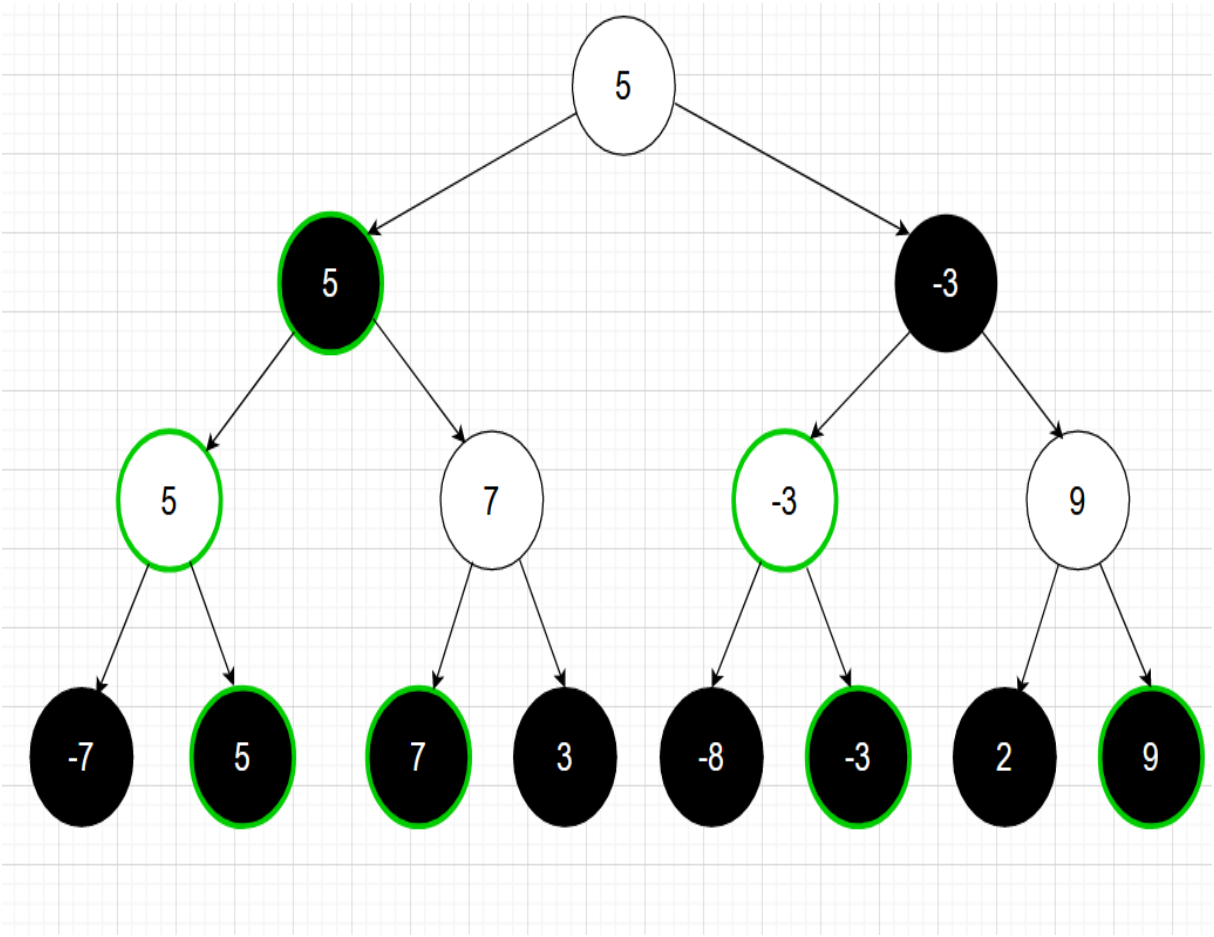
U ovisnosti o tome čiji je potez u određenoj poziciji, algoritam će izabrati minimalnu ili maksimalnu evaluaciju u određenoj situaciji te tako odabrati optimalan potez. U kodu to izgleda ovako:

```
double minimax(chess.Chess game, int depth,bool maximizingPlayer) {
    if (depth == 0 || game.game_over) {
      return evaluateBoard(game);
    }

    var moves = game.moves();

    if (maximizingPlayer) {
      double maxEval = double.negativeInfinity;
      for (var move in moves) {
        game.move(move);
        double eval = minimax(game, depth - 1, false);
        game.undo();
        maxEval = max(maxEval, eval);
      }
      return maxEval;
     else {
      double minEval = double.infinity;
      for (var move in moves) {
        game.move(move);
        double eval = minimax(game, depth - 1, true);
        game.undo();
        minEval = min(minEval, eval);
```

```
        }
        return minEval;
    }
}
```



Slika 9: Primjer stabla pretrage pomoću minimax algoritma

(Izvor: autor, alat: https://app.diagrams.net/, 2024.)

Alpha-beta podrezivanje je nadograđeni minimax algoritam i koristi se kako bi se optimizirala odnosno smanjila pretraga unutar stabla. Kako je šah veoma kompleksna igra i mogućnosti su velike, minimax troši velike resurse pri pretrazi svakog poteza budući da on pretražuje svaki mogući potez.

Algoritam ima dvije vrijednosti koje su zovu alpha i beta. Alpha predstavlja minimalnu ocjenu koju može postići maximizing player, a beta maksimalnu ocjenu koju može postići minimizing player. Na početku, alpha ima vrijednost minus beskonačno, a beta plus beskonačno što su najgori mogući scenariji za oba igrača.

Prilikom pretrage, algoritam odbacuje grane koje ne mogu utjecati na krajnji rezultat. Prilikom ocjene poteza, ako minimizing player's beta postane manja od maximizing player's alphe (beta<alpha), ostali potezi u tom čvoru se ne računaju.

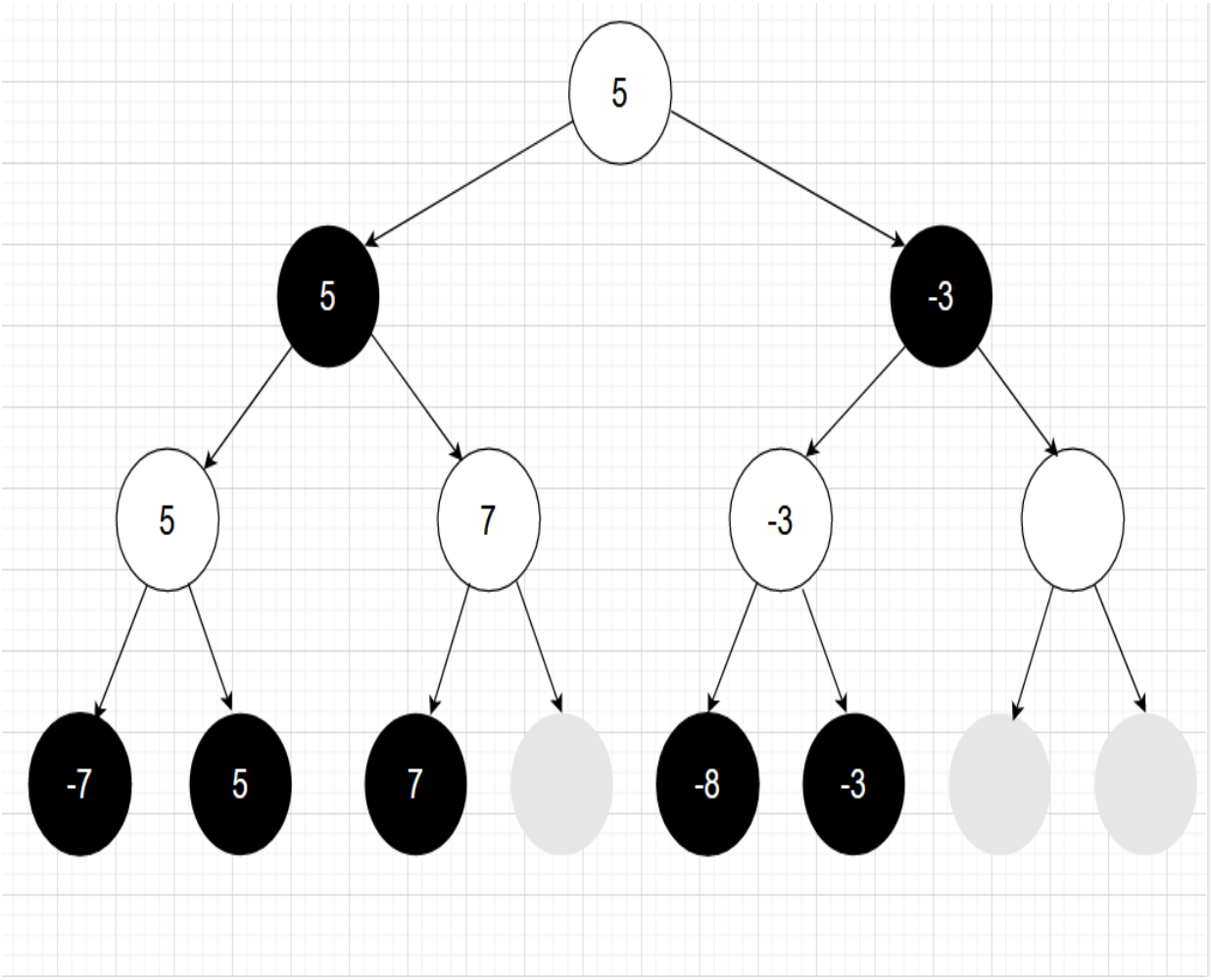
Kao rezultat pretrage, ovaj algoritam vraća isti potez kao standardni minimax algoritam s tim da pregledava manje čvorova i tako znatno poboljšava efikasnost pretrage. U kodu to izgleda ovako:

```
      double alphaBeta(chess.Chess game, int depth, double alpha, double
beta,
      bool maximizingPlayer) {
    if (depth == 0 || game.game_over) {
      return evaluateBoard(game);
    }

  var moves = game.moves();
    if (maximizingPlayer) {
      double maxEval = double.negativeInfinity;
      for (var move in moves) {
        game.move(move);
        double eval = alphaBeta(game, depth - 1, alpha, beta, false);
        game.undo();
        maxEval = max(maxEval, eval);
        alpha = max(alpha, eval);
        if (beta <= alpha) {
          break;
        }
      }
      return maxEval;
```

```
  } else {
    double minEval = double.infinity;
    for (var move in moves) {
      game.move(move);
      double eval = alphaBeta(game, depth - 1, alpha, beta, true);
      game.undo();
      minEval = min(minEval, eval);
      beta = min(beta, eval);
      if (beta <= alpha) {
        break;
      }
    }
    return minEval;
  }
}
```

Za razliku od minimax algoritma, ovaj kod sadrži dodatne parametre alpha i beta. Za maximizing player smo stavili da je alpha veća vrijednost između alphe i zadnje ocjene(`maxEval = max(maxEval, eval)`). Ako je beta manja ili jednaka vrijednosti alpha, pretraga se prekida. Slično se događa i sa minimizing player gdje stavimo da je beta jednaka najmanjoj vrijednosti između bete i zadnje ocjene(`beta = min(beta, eval`). Zatim, ako je beta manja ili jednaka vrijednosti alpha, pretraga se prekida.



Slika 10: Alpha-beta podrezivanje

(Izvor: autor, alat: https://app.diagrams.net/, 2024.)

4.1.1. Evaluacijska funkcija

Evaluacijska funkcija služi kao orijentir prilikom ocjene šahovske pozicije.

```
double evaluateBoard(chess.Chess game) {
    double score = 0.0;
    String board = game.fen.split(' ')[0];
    for (int i = 0; i < board.length; i++) {
      switch (board[i]) {
        case 'P':
          score -= 100;
          break;
        case 'p':
          score += 100;
          break;
        case 'N':
        case 'B':
          score -= 300;
          break;
        case 'n':
        case 'b':
          score += 300;
          break;
        case 'R':
          score -= 500;
          break;
        case 'r':
          score += 500;
          break;
        case 'Q':
          score -= 900;
          break;
        case 'q':
          score += 900;
          break;
    }
      }
   return score;
}
```

U ovom kodu možemo vidjeti da funkcija uzima chess.Chess objekt koji predstavlja trenutnu ocjenu pozicije kao argument. Funkcija izvlači FEN (Forsyth–Edwards Notation) string iz objekta chess.Chess koristeći game.fen.split(' ')[0]. FEN string predstavlja trenutnu poziciju odnosno položaj svake figure na tabli.

Naredba switch dodjeljuje brojčanu vrijednost svakoj figuri:

- **P** (crni pješak): -100
- **p** (bijeli pješak): 100
- **N** (crni skakač)
- **B** (crni lovac): -300
- **n** (bijeli skakač):
- **b** (bijeli lovac): 300
- **R** (crni top): -500
- **r** (bijeli top): 500
- **Q** (crna dama) :-900
- **q** (bijela dama): 900

Rezutat se zbraja i dodjeljuje varijabli score koja na kraju vraća ocjenu trenutne pozicije. Ova jednostavna funkcija ocjenjuje poziciju samo na osnovu broja i opće vrijednosti figura. To je veoma dobro polazište za evaluacijsku funkciju, a prilikom pravljenja kompleksnije evaluacijske funkcije, trebaju se uzeti u obzir stvari poput sigurnosti kralja, matnih prijetnji, prostorne prednosti određene strane, položaj i aktivnost svake od figura itd.

Mnogo je bitnih faktora koji se moraju razmotriti prilikom konačne ocjene pozicije i mnogo je načina za optimizaciju ove funkcije. Čak i tada, ona može dati pogrešnu ocjenu. Postoje primjeri određenih pozicija gdje jedna strana ima veliku materijalnu prednost što znatno utječe na evaluacijsku funkciju, ali slabija strana ima kreativne načine da spasi remi ili čak pobijedi. Takvi slučajevi su veoma rijetki i studiozni, ali bitno je uzeti u obzir da računalo može pogriješiti prilikom ocjene šahovske pozicije.

Slika 11: Igra protiv računala

(Izvor: autor, 2024.)

4.2. Prvaci i otvaranja

Ova sekcija u aplikaciji prikazuje sve službene svjetske šahovke prvake počevši od Wilhelma Steinitza (1886. g.) do trenutnog svjetskog prvaka Ding Lirena (2024. g.). Na svakoj stranici se nalazi slika svjetskog prvaka, a ispod slike tekst koji korisniku prikazuje najbitnije informacije o toj osobi, a to su npr. od kada do kada je bio svjetski prvak, koja su mu ostala velika postignuća, koji mu je bio stil igre, po čemu je bio poznat itd.

```
import 'package:flutter/material.dart';
import 'package:racunalni_sah/lasker.dart';

class WorldChampions extends StatefulWidget {
  const WorldChampions({super.key});

  @override
  State<WorldChampions> createState() => _WorldChampionsState();
}

class _WorldChampionsState extends State<WorldChampions> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Wilhelm Steinitz'),
        centerTitle: true,
      ),
      body: Container(
        color: Colors.brown,
        child: Column(
          children: [
            const SizedBox(height: 30),
            Image.asset('images/steinitz.jpg'),
            const SizedBox(
              height: 10,
            ),
            const Divider(),
            Container(
              padding: const EdgeInsets.all(10.0),
              width: double.infinity,
              color: Colors.brown,
              child: const Center(
                child: Text(
                  'Tekst o svjetskom prvaku. ',
                  textAlign: TextAlign.start,
                  style: TextStyle(
                      fontSize: 18,
                      fontStyle: FontStyle.italic,
                      fontWeight: FontWeight.bold),
                ),
              ),
            ),
            const SizedBox(height: 20),
          ],
```

```
            ),
          ),
      );
      }
      }
```

U ovom dijelu koda kreiram klasu WorldChampions. Na samom vrhu aplikacije se nalazi AppBar sa imenom svjetskog prvaka kojeg opisujem, u ovom slučaju 'Wilhelm Steinitz'. Ispod AppBara nalazi se Body koji je zapravo Container sa smeđom pozadinom. On sadrži Column widget u kojem je prikazana slika svjetskog prvaka. Slika je uvezena pomoću Image.assets putem images/steinitz.jpg'.

Nakon slike nalazi se tekst Container koji sadrži tekst widget. Između slike i teksta nalazi se Divider koji služi kako bi odvojio sliku od teksta. On stvara horizontalnu liniju koja se koristi kako bi se vizualno odvojio sadržaj unutar aplikacije i tako doprinosi estetskom izgledu. Tekst widget sadrži informacije o osobi na slici i podatke kao što su veličina i stil fonta.

```
bottomNavigationBar: BottomAppBar(
        color: Colors.brown,
        child: Row(
          mainAxisAlignment: MainAxisAlignment.end,
          children: <Widget>[
            SizedBox(
              width: 100,
              child: ElevatedButton(
                onPressed: () {
                  Navigator.of(context)
                    .push(MaterialPageRoute(builder:(BuildContext context) {
                    return const Lasker();
                  }));
                },
                child: const Icon(Icons.keyboard_double_arrow_right),
              ),
            ),
          ],
        ),
      ),
```

Na samom dnu stranice se nalazi BottomAppBar widget koji služi kako bi se omogućila navigacija između ostalih stranica. To sam implementirao pomoću gumba tipa ElevatedButton na kojem se nalazi ikona strelice koja označava moguće listanje. On se nalazi u donjem desnom kutu. Unutar gumba se nalazi funkcija (Navigator) kojim prelazimo na sljedećeg svjetskog prvaka, u ovom slučaju Laskera. Za ostale svjetske prvake sam implementirao i gumb u donjem lijevom kutu koji omogućava vraćanje na prethodnog svjetskog prvaka.

Slika 12: Prvaci – Vasilij Smislov

(Izvor: https://www.cooliranje.com/cool/Sah/Legende-saha/693095/Vasilij-Vasiljevic-Smislov--Vasilij-Smislov-, autor, 2024.)

Sekcija otvaranja je napravljenja na sličan način kao i prvaci. Appbar sadrži teskt koji označava ime otvaranja prikazanoj na slici ispod. BottomAppBar sadrži gumbe za navigaciju.

Slika 13: Otvaranja „Računalni šah“

(Izvor: autor, 2024.)

4.3. Zagonetke

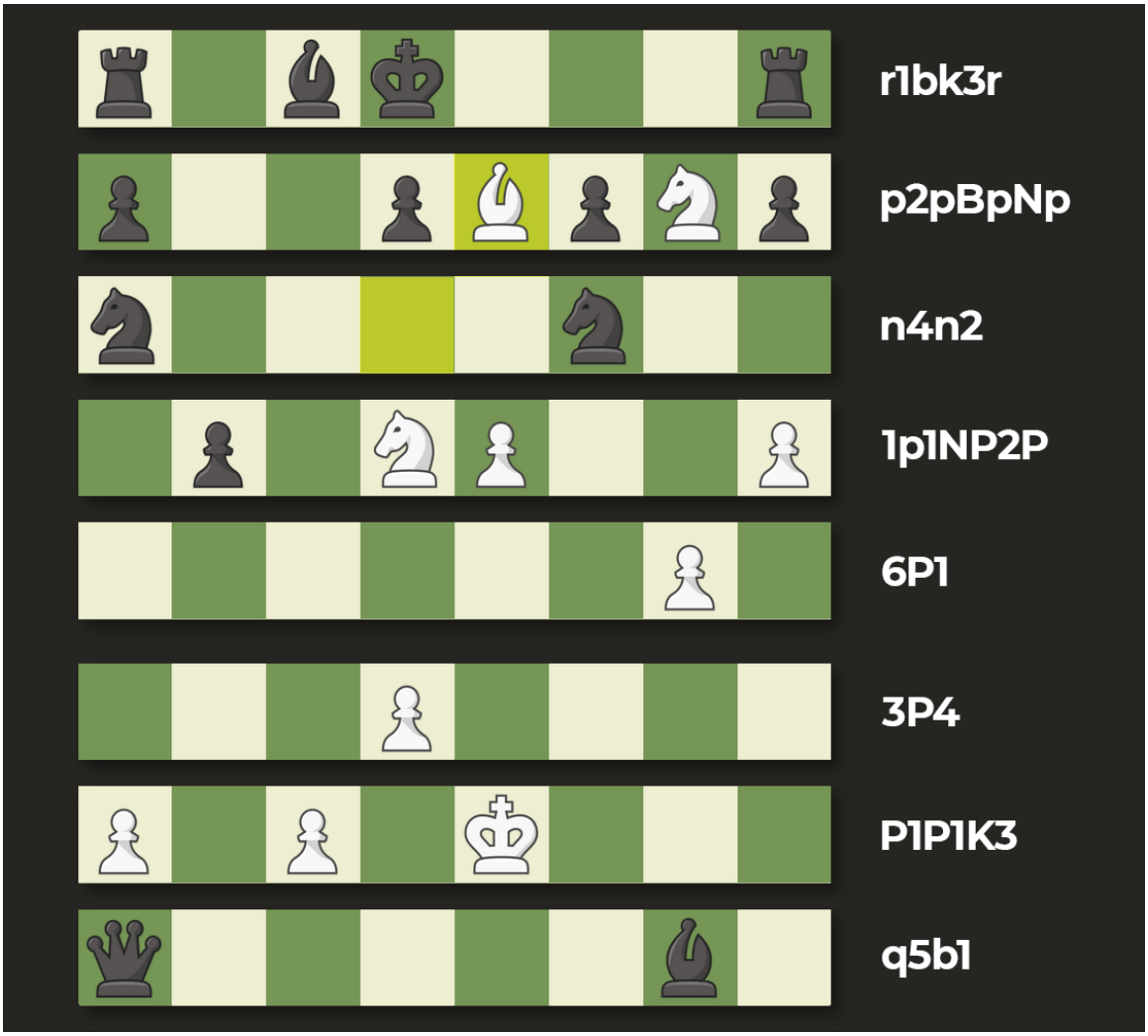
U ovoj sekciji korisnik ima priliku rješavati šahovske zagonetke i na taj način vježbati. Ulaskom na zagonetke pred korisnika se postavlja pozicija taktičkog karaktera u kojoj postoji potez ili redoslijed poteza koji forsirano dovode do pobjede. Traži se unos pravog poteza, a ukoliko korisnik unese pogrešan potez, pojavljuje se poruka o krivom potezu i nudi se mogućnost ponovnog pokušaja. Pokušaji su neograničeni, a kada se zagonetka uspješno rješi, na ekran iskače poruka čestitke. Moguće je ići na druge zagonetke ili se vraćati na stare.

Slika 14: Zagonetke „Računalni šah“

(Izvor: autor, 2024.)

4.3.1. Fen notacija

Forsyth-Edwards notacija (FEN) je standardna notacija za prikazivanje određene šahovske pozicije. Korištenjem posebnih znakova moguće je prikazati bilo koju poziciju na šahovskoj tabli. Sljedeća slika ilustruje na koji nafin FEN notaciju funkcionira.



Slika 15: FEN notacija „Računalni šah“

(Izvor: https://www.chess.com/terms/fen-chess, 2024.)

Slika prikazuje način na koji je moguće prikazati određenu šahovsku poziciju koristeći slova i brojeve. Pozicija na prethodnoj slici bi se znakovno na računalu prikazala ovako: 'r1bk3r/p2pBpNp/n4n2/1p1NP2P/6P1/3P4/P1P1K3/q5b1'.

Slovo k je oznaka za kralja, q za damu, b za lovca, n za skakača i p za pješaka. Označavanje počinje od zadnjeg reda sa desne strane crnom, odnosno od polja a8 i koristi se znak '/ ' kako bi se označio prelazak na novi red. Figure crnog se označavaju malim slovima, a figure bijelog velikim. Prazna polja između figura se označavaju brojevima, što znači da ako vidimo broj 3, to znači da se nalaze 3 prazna polja između figura.

U prethodno opisanom slučaju, vidimo da se crni top nalazi na polju a8, poslije njega vidimo broj 1 što predstavlja prazno polje, b znači da je na polju c8 crni lovac, na d8 je crni kralj, 3 predstavlja 3 prazna polja i na kraju r označava položaj crnog topa na h8. Sve ovo je bitno kako bi razumjeli na koji način je moguće postaviti željenu poziciju u računalo. Budući da sam sam postavljao zagonetke, FEN notacija mi je bila potrebna kako bih postavio svaku poziciju na tablu i tako omogućio njeno rješavanje. Sljedeći kod predstavlja poziciju prikazanu na slici 4 sa iznimkom da je bijeli lovac sa polja c4 pomjeren na polje d5 (odigran je pogrešan potez) :

```
class _Tactics extends State<Legal> {
  ChessBoardController controller = ChessBoardController();
  List<String> correctMoves = ['Bxf7+', 'Nd5#'];
  int moveIndex = 0;

  @override
  void initState() {
    super.initState();
    controller.game.load(
        'rnbqkbnr/1pp2ppp/p2p4/4N3/2B1P3/2N5/PPPP1PPP/R1BbK2R w
KQkq - 0 5');
  }
```

U ovom dijelu koda kreiram listu naziva correctMoves koja se sastoji od dva poteza, a to su Bxf7+, što označava da bijeli lovac uzima(x) na f7 sa šahom(+), a drugi potez Nd5# označava da bijeli skakač dolazi na polje d5 i daje šah-mat (#).

```
        controller.game.load('rnbqkbnr/1pp2ppp/p2p4/4N3/2B1P3/2N5/PPPP1
    PPP/R1BbK2R w KQkq - 0 5');
```

U ovom dijelu koda postavljam željenu poziciju za rješavanje u standarnoj FEN notaciji. Nakon oznake za zadnji red vidimo slovo 'w' što značava da je bijeli na potezu, 'KQkq' je oznaka da bijeli kao i crni ima mogućnost rokade na obe strane, znak ' – ' znači da nije moguće odigrati en passant potez, ' 0 ' je oznaka za „halfmove number“ što označava koliko poteza je prošlo od poteza pješakom ili uzimanja. Ova oznaka je korisna u slučaju pravila „50 poteza“ koje glasi da ako nema uzimanja figura niti pokretanja pješaka u rasponu od 50 poteza, partija se proglašava remijem. Zadnja oznaka ' 5 ' označava koliko je ukupno poteza odigrano do trenutne pozicije dobijene FEN notacijom.

```
    onMove: () {
                String pgn = controller.game.pgn();
                if (pgn.isNotEmpty) {
                  List<String> moves = pgn.split(' ');
                  String lastMove = moves.last;
                  if (lastMove != correctMoves[moveIndex]) {

ScaffoldMessenger.of(context).hideCurrentSnackBar();
                    ScaffoldMessenger.of(context).showSnackBar(
                      SnackBar(
                          duration: const Duration(days: 1),
                          content: Row(
                              mainAxisAlignment:
                                  MainAxisAlignment.spaceBetween,
                              children: <Widget>[
                                const Text('Pogrešno'),
                                SnackBarAction(
                                  label: 'Pokušajte ponovno',
                                  onPressed: () {
                                    setState(() {
                                      if (moveIndex == 0) {
                                        controller.game.load(

'rnbqkbnr/1pp2ppp/p2p4/4N3/2B1P3/2N5/PPPP1PPP/R1BbK2R w KQkq - 0 5');
                                      } else if (moveIndex == 1) {
                                        controller.game.load(
```

```
'rnbqkbnr/1pp2ppp/p2p4/4N3/2B1P3/2N5/PPPP1PPP/R1BbK2R w KQkq - 0 5');
                                            controller.game.move('Bxf7+');
                                            controller.game.move('Ke7');
                                          }
                                        });
                                      },
                                    ),
                                  ])),
                            );
                          } else {
                           String message = moveIndex == correctMoves.length -1
                                ? 'Bravo!'
                                : 'Točno....nastavite';

ScaffoldMessenger.of(context).hideCurrentSnackBar();
                           ScaffoldMessenger.of(context).showSnackBar(
                             SnackBar(
                               content: Text(message),
                               duration: const Duration(days: 1),
                             ),
                           );
                           moveIndex++;
                           if (moveIndex == 1) {
                             controller.game.move(('Ke7'));
                           }
                         }
                       }
                     }
```

OnMove funkcija je glavna funkcija koja služi za logiku nakon poteza na šahovskoj tabli. Portable game notation (pgn) je standardni format za pisanje šahovskih partija i na početku onMove funkcije koristimo `String pgn = controller.game.pgn();` kako bi dobili željenu poziciju. Nakon toga provjeravamo je li pgn prazan ili ne (`if (pgn.isNotEmpty)` što znači da je bar jedan potez odigran u slučaju da pgn nije prazan. Pgn string dijelimo funkcijom split kako bi mogli zapisati željeni potez te zadnji odigrani potez spremamo u varijablu lastMove.

U sljedećem koraku želimo osigurati da je odigran željeni potez. Ukoliko je odigran pogrešan potez (`correctMoves[moveIndex]`) događa se sljedeće:

1. Bilo koji trenutni SnackBar se sakriva korištenjem `ScaffoldMessenger.of(context).hideCurrentSnackBar()`.
2. Novi SnackBar se prikazuje sa tekstom „Pogrešno“ i SnackBarAction sa tekstom „Pokušajte ponovno“ koji, kada se pritisne, pokreće setState call.
3. Unutar setState, pozicija se resetira na originalnu što korisniku omogućava da ponovno pokuša pronaći ispravan potez.

Ukoliko je odigran točan potez, pojavljuje se SnackBar sa porukom „Točno….nastavite“ ukoliko je potrebno pronaći više točnih poteza ili „Bravo“ ako je zagonetka riješena do kraja. MoveIndex se povećava za 1 što znači da se potez 'Ke7' odigrava odmah kako bi korisnik automatski mogao tražiti sljedeći potez. Ovaj dio koda drži cijelu logiku vezanu za ispravno funkcioniranje zagonetki.

5. Zaključak

U ovom radu detaljno sam opisao elemente potrebne za ravoj računalne igre šah. Ti elementi obuhvaćaju planiranje i željeni dizajn, programiranje igre kao i implementaciju algoritma kojeg računalo koristi pri traženju najboljeg poteza u bilo kojoj poziciji. Evaluacijska funkcija služi kao orijentir pri ocjeni pozicije. Mogućnost realizacije svih ovih elemenata sam pronašao u razvojnom alatu Flutter koristeći Visual Studio Code za razvijanje potrebnog koda. Dart je glavni jezik u kojem je napravljena aplikacija „Chesmasters“.

Konačni rezultat rada jest jedna multifunkcionalna igra koja korisniku nudi zabavu ali i mogućnost učenja. Korisnik može testirati svoju šahovsku snagu pokušavajući pobijediti računalo, ali ima i priliku rješavati zagonetke i na taj način vježbati uočavanje obrazaca što je vjerojatno i najvažnija stvar za napredak u igri. Pored toga, može saznati korisne informacije o najpoznatijim šahovskim otvaranjima kao i svim svjetskim šahovskim prvacima.

Najzahtjevniji dio u ovoj igri bio je pronalazak i implementacija algoritma za pronalazak najboljeg poteza. To zapravo jest i sama suština ovog rada, implementacija AI programa koji je sposoban sam igrati poteze iza kojih postoji određena logika. Alpha-beta podrezivanje je moćan algoritam koji ima veliku prednost u odnosu na minimax u tome što odbacuje računanje nepotrebnih poteza i tako štedi vrijeme ali i računalne resurse. Implementirao sam ovaj algoritam koristeći osnovnu evaluacijsku funkciju da bi računalo moglo razumjeti odnose među figurama.

Iako program funkcionira na željeni način, mogućnosti za poboljšanja su beskonačne. Moguće je poboljšati evaluacijsku funkciju tako da program shvati i relativnu vrijednost figura u odnosu na poziciju. Moguće je poboljšati i algoritam na razne načine tako da bude brži i efikasniji pri računanju. Ne smijemo zanemariti i činjenicu da je na samom kraju algoritam ipak ovisan o procesorskoj snazi uređaja koji ga pokreće. Povećavanjem dubine računanja se značajno poboljšava performansa ali se i značajno gubi na brzini.

Osim ovoga, mogućnosti za poboljšanja se nalaze i u ostalim sekcijama kao i u dizajnu igre. U sekciji zagonetke moguće je dodati još zagonetki i posložiti ih po težini te dodati Elo rejting sustav za korisnika gdje bi korisnik rješavao zagonetke i dobijao rejting poene u odnosu na uspješnost rješavanja zagonetki.

Ovo su neki od primjera za poboljšanja, a vjerujem da ih ima još. Nakon objavljivanja igre na tržište, dobijaju se povratne informacije od korisnika i igra se ažurira, ispravljaju se greške ili nedostaci te se uvode nove funkcionalnosti koje dodatno poboljšavaju igru.

6. Literatura

[1] Flutter 3 2024., https://docs.flutter.dev/

[2] Dart 2024. , https://dart.dev/

[3] Visual Studio Code 2024., https://code.visualstudio.com/

[4] Android Studio 2024., https://developer.android.com/studio

[5] Pub.dev 2024., https://pub.dev/

[6] P. Gebhardt „The history of chess AI“ , https://blog.paessler.com/author/patrick-gebhardt, 2024.

7. Popis slika

7.1. Slike korištene unutar igre

Slika svjetskog prvaka: Wilhelm Steinitz (Izvor: https://bs.wikipedia.org/wiki/Wilhelm_Steinitz, 2024.)

Slika svjetskog prvaka: Emanuel Lasker (Izvor: https://www.faz.net/aktuell/feuilleton/buecher/rezensionen/sachbuch/eine-monumentale-emanuel-lasker-biographie-bei-exzelsior-18296130.html, 2024.)

Slika svjetskog prvaka: José Raúl Capablanca (Izvor: https://commons.wikimedia.org/wiki/File:Chess_Fever_Kapablanka.jpg, 2024.)

Slika svjetskog prvaka: Alexander Alekhine (Izvor: https://russkiymir.ru/en/publications/319438/, 2024.)

Slika svjetskog prvaka: Max Euwe (Izvor: https://www.fide.com/news/1090, 2024.)

Slika svjetskog prvaka: Mihail Botvinik (Izvor: https://www.championat.com/other/article-4852071-kak-iz-sbornoj-sssr-po-shahmatam-isklyuchili-chempiona-mira-pochemu-mihail-botvinnik-ne-sygral-na-olimpiade-1952.html, 2024.)

Slika svjetskog prvaka: Vasilij Smislov (Izvor: https://www.cooliranje.com/cool/Sah/Legende-saha/693095/Vasilij-Vasiljevic-Smislov--Vasilij-Smislov-, 2024.)

Slika svjetskog prvaka: Mikhail Tal (Izvor: https://en.chessbase.com/post/in-memory-of-mikhail-tal, 2024.)

Slika svjetskog prvaka: Tigran Petrosian (Izvor: https://www.britannica.com/biography/Tigran-Vartanovich-Petrosyan, 2024.)

Slika svjetskog prvaka: Boris Spaski (Izvor: https://newideal.aynrand.org/ayn-rands-open-letter-to-comrade-spassky-chessmaster/, 2024.)

Slika svjetskog prvaka: Bobby Fischer (Izvor: https://www.chess.com/bs/club/the-bobby-fischer-club-1, 2024.)

Slika svjetskog prvaka: Anatoly Karpov (Izvor: https://x.com/dgriffinchess/status/954325288292421632, 2024.)

Slika svjetskog prvaka: Gari Kasparov (Izvor: https://www.nytimes.com/2020/12/18/arts/music/garry-kasparov-classical-music.html, 2024.)

Slika svjetskog prvaka: Vladimir Kramnik (Izvor: https://en.chessbase.com/post/grandmaster-chef-vladimir-kramnik, 2024.)

Slika svjetskog prvaka: Viswanathan Anand: (Izvor: https://shreevedic.com/viswanathan-anand-astrology-numerology-analysis/, 2024.)

Slika svjetskog prvaka: Magnus Carlsen (Izvor: https://variety.com/2016/film/reviews/magnus-review-magnus-carlsen-1201919852/, 2024.)

Slika svjetskog prvaka: Ding Liren (Izvor: https://www.theguardian.com/sport/2024/jan/05/chess-world-champion-ding-liren-to-return-at-tata-steel-wijk-aan-zee, 2024.)