SVEUČILIŠTE U SPLITU  
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

PRIMJENA UMJETNE INTELIGENCIJE U ŠAHU

Vjeran Cvitanić

Split, srpanj 2023.

|  |  |  |
| --- | --- | --- |
|  | SVEUČILIŠTE U SPLITU  FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE |  |

Sveučilišni preddiplomski studij: **Računarstvo**

Oznaka programa: 120

Akademska godina: 2022./2023.

Ime i prezime: **VJERAN CVITANIĆ**

Broj indeksa: 133-2020

**ZADATAK ZAVRŠNOG RADA**

Naslov: **PRIMJENA UMJETNE INTELIGENCIJE U ŠAHU**

Zadatak: U okviru ovog završnog rada potrebno je istražiti mogućnost primjene umjetne inteligencije za igranje šaha. Odabrati i detaljno proučiti *minmax* i *alpha-beta pruning* algoritme. Nakon toga potrebno je izraditi desktop aplikaciju koja će omogućiti korisniku igranje protiv druge osobe, protiv jednog od implementiranih algoritama umjetne inteligencije ili će moći odabrati da algoritmi igraju jedan protiv drugoga. Za kraj je potrebno usporediti performanse implementiranih algoritama.

Datum obrane: Datum

|  |  |  |
| --- | --- | --- |
|  |  | Mentor: |
|  |  |  |
|  |  | prof. dr. sc. Linda Vicković |

**IZJAVA**

Ovom izjavom potvrđujem da sam završni rad s naslovom „**Primjena umjetne inteligencije u šahu**“ pod mentorstvom prof. dr. sc. Linde Vicković pisao samostalno, primijenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući naveo u završnom radu citirao sam i povezao s korištenim bibliografskim jedinicama.

Student

Slika na kojoj se prikazuje rukopis, Font, kaligrafija, tipografija

Opis je automatski generiran

Vjeran Cvitanić

**SADRŽAJ**

[1 UVOD 1](#_Toc140133537)

[2 UMJETNA INTELIGENCIJA 2](#_Toc140133538)

[2.1 Povijesni razvoj 2](#_Toc140133539)

[2.1.1 Inteligentni strojevi 3](#_Toc140133540)

[2.1.2 Nastanak sustava umjetne inteligencije 3](#_Toc140133541)

[2.2 Trenutno stanje razvoja 5](#_Toc140133542)

[2.2.1 Svakodnevna primjena 6](#_Toc140133543)

[2.3 Budućnost 7](#_Toc140133544)

[3 UMJETNA INTELIGENCIJA U ŠAHU 8](#_Toc140133545)

[3.1 Šah kao razvojno okruženje umjetne inteligencije 8](#_Toc140133546)

[3.2 Povijest *chess enginea* 8](#_Toc140133547)

[3.2.1 AphaZero 8](#_Toc140133548)

[3.2.2 Stockfish 9](#_Toc140133549)

[3.3 Dosadašnji rezultati 9](#_Toc140133550)

[4 RAZVOJ APLIKACIJE ZA IGRU ŠAHA 10](#_Toc140133551)

[4.1 Klase 10](#_Toc140133552)

[4.2 Primijenjeni AI algoritmi 15](#_Toc140133553)

[4.2.1 *Minmax* Algoritam 15](#_Toc140133554)

[4.2.2 Alpha-beta pruning 22](#_Toc140133555)

[5 USPOREDBA RAZLIČITIH ALGORITAMA 27](#_Toc140133556)

[5.1 Usporedba broja pretraženih pozicija *minmax* i *alpha-beta* algoritama 27](#_Toc140133557)

[5.2 Usporedba različitih metoda za izračun vrijednosti pozicije 27](#_Toc140133558)

[5.2.1 Parametri za izračun vrijednosti pozicije 27](#_Toc140133559)

[5.2.2 Opis usporedbe 31](#_Toc140133560)

[5.2.3 Rezultati usporedbe 31](#_Toc140133561)

[6 ZAKLJUČAK 33](#_Toc140133562)

[LITERATURA 34](#_Toc140133563)

[PRILOZI 36](#_Toc140133564)

[Kazalo slika i tablica 36](#_Toc140133565)

[Kazalo slika 36](#_Toc140133566)

[Kazalo tablica 37](#_Toc140133567)

[Popis oznaka i kratica 37](#_Toc140133568)

[SAŽETAK I KLJUČNE RIJEČI 38](#_Toc140133569)

[SUMMARY AND KEYWORDS 39](#_Toc140133570)

# UVOD

Šah je strateška igra, nastala na području Indije u sedmom stoljeću. Igra se na ploči s osam redaka i osam stupaca, ukupno 64 polja, od kojih je pola bijelo, pola crno, a igraju dva igrača. Svaki igrač ima 16 figura (osam pješaka, po dva lovca, konja i kule, jednu damu i kralja), kretanje svake figure ograničeno je pravilima, a moguće je „pojesti“ protivničku figuru. Igrači se izmjenjuju nakon svakog poteza , a cilj igre je napasti protivničkog kralja nekom od figura na način da se protivnik ne može obraniti od napada.

Umjetna inteligencija (engl. *Artificial Intelligence*, AI) označava računalni sustav koji posjeduje sposobnost obavljanja zadataka čije izvršavanje zahtijeva korištenje određenih intelektualnih procesa. Ova tehnologija jedna je od najzanimljivijih iz područja računarstva zadnjih godina, a u pozornost šire javnosti došla je povodom objave ChatGPT-a, moćnog *chatbot* alata. Sustavi umjetne inteligencije primjenjuju se u medicini, industrijskim procesima, znanosti itd., a s obzirom na ubrzan razvoj ovih tehnologija, očekuje se sve veći utjecaj na svakodnevni život. Među istraženijim domenama primjene AI-a spada šah, zbog kompleksnosti i determinističke prirode igre.

Kreiranje umjetne inteligencije za igranje šaha doprinijelo je ponajviše razvoju *minmax* i njemu srodnih algoritama, koji grafom predstavljaju stanje igre i temelje se na pretraživanju čvorova grafa. *Minmax* je algoritam pretraživanja po dubini (engl. *depth first search* *algorithm)* te koristi *brute force* pristup u pretraživanju čvorova. Temelji se na pretpostavci da oba igrača igraju optimalne poteze. Najjednostavnija varijacija *minmax* algoritma, koja optimizira pretragu, naziva se *alpha-beta pruning*. Novost u svijetu šahovskog AI-a posljednjih godina je korištenje strojnog učenja (engl. *machine learning)*, koja odbacuje *brute force* pristup te podrazumijeva postupno učenje umjetne inteligencije.

U ovom radu obrađena je tema primjene umjetne inteligencije u igranju šaha. U drugom poglavlju dan je kratki uvod u temu općenitog razvoja umjetne inteligencije. Zatim je u trećem poglavlju predstavljena primjena umjetne inteligencije u igranju šaha. U četvrtom poglavlju kratko je opisano programsko rješenje igre šaha te je dano objašnjenje primijenjenih algoritama. Usporedba različitih algoritama dana je u petom poglavlju. Kratki osvrt dan je u zaključnom poglavlju.

# UMJETNA INTELIGENCIJA

Umjetna inteligencija definira se kao sposobnost digitalnog računala ili računalno upravljanog robota da vrši zadatke inače vezane uz inteligentna bića [1]. Podrazumijeva sustav koji posjeduje intelektualne procese, karakteristične ljudima, kao što je sposobnost zaključivanja, učenja i korištenja znanja i vještina u različitim domenama.

## Povijesni razvoj

Početak razvoja sustava umjetne inteligencije započeo je krajem prve polovice 20. stoljeća, no teorijska podloga nastala je mnogo prije. Filozofi i matematičari kroz povijest temeljili su vjeru u moguće postojanje umjetne inteligencije na pretpostavci da se proces ljudskog razmišljanja može simulirati mehaničkim procesima. Mnogi proslavljeni intelektualci poput Rene Descartesa i Gottfrieda Leibniza smatrali su da se svo racionalno razmišljanje može svesti na logički sustav, a poznati britanski filozof Thomas Hobbes je napisao: „razum je ništa drugo nego računanje“ [2]. U 20. stoljeću dogodio se bitan napredak u matematičkoj logici. Osnovu je postavio George Boole 1854. napisavši djelo „The Laws of thought“, da bi zatim Bertrand Russell i Alfred Whitehead 1913. objavili „Principia Mathematica“, jedno od povijesno najznačajnijih djela s područja matematike. Potaknut tim djelom, njemački matematičar David Hilbert izazvao je matematičare tog doba da daju odgovor na pitanje može li se svo matematičko zaključivanje formalizirati. Rezultat tog izazova bili su „Teorem nepotpunosti“ Kurta Godela te Turingov stroj britanskog matematičara Alana Turinga. Godel je dokazao da postoje granice matematičke logike, odnosno da postoje iskazi koje matematika ne može dokazati ni opovrgnuti. No, radovi ovog doba pokazali su da se svako matematičko zaključivanje, unutar tih granica, može mehanizirati, što je najjasnije iskazano Turingovim strojem, matematičkim modelom računanja kojim se dokazuje mogućnost postojanja stroja koji posjeduje sposobnost rješavanja bilo kojeg algoritma.

Sredinom 20. stoljeća neurologija je došla do saznanja da je ljudski mozak pojednostavljeno mreža ogromnog broja neurona koji mogu biti u dva stanja, ugašenom ili upaljenom. Claude Shannon, američki znanstvenik, proslavio se opisavši digitalne signale te prepoznavši da je moguće od elektroničkih sklopova predstaviti bilo koju logičku operaciju. Istovremeno, Turing je svoj doprinos dao i kroz djelo „Theory of computation“, gdje je pokazao da se svaki oblik računanja može digitalizirati. Sva ova saznanja dovela su do ideje o izradi elektroničkog mozga, inteligencije pogonjene električnim signalima.

### Inteligentni strojevi

Alan Turing je 1950. još jednom dao svoj obol razvoju umjetne inteligencije, kad je izmislio metodu danas poznatu kao „Turingov test“. Turing je shvatio da je dosta nezgodno definirati inteligenciju i tko (što) zaslužuje taj atribut te je stoga osmislio test: ako stroj može vršiti komunikacijom s čovjekom tako da sugovornik ne shvati razliku u odnosu na komunikaciju s drugom osobom, stroj se može smatrati inteligentnim [3]. Primijetio je da ljudi za rješavanje problema koriste potrebne informacije i logiku te je smatrao da strojevi mogu isto [4].

Najpoznatiji argument protiv Turingova testa je „The Chinese room argument“, američkog filozofa Johna Searla [5]. On nudi misaoni eksperiment u kojem kineski govornik komunicira s računalom koje se nalazi skriveno u sobi. Kineski govornik ne zna da razgovara sa strojem te kroz prozorčić ubacuje svoja pitanja i odgovore napisane na papiru. Po „Turingovom testu“, kad bi Kinez nastavio razgovarati sa strojem, ne shvaćajući da sugovornik nije osoba, dani stroj bi se proglasio inteligentnim. Searle pobija zaključak tako što predlaže sljedeću situaciju: sustav koji ne poznaje kineski, npr. osoba negovornik kineskog jezika, mogla bi uz pomoć kineskog rječnika održavati istu komunikaciju s kineskim govornikom, bez da zapravo zna kineski. Autor zaključuje da stroj na isti način komunicira, bez pravog razumijevanja kineskog pa nije ni moguće zaključiti išta o inteligenciji stroja. Nadalje, Searle navodi da računalni program može samo izvršavati naredbe, bez razumijevanja što radi te zaključuje da se nijedan računalni program ne može smatrati inteligentnim i da su za stvaranje inteligencije potrebni biološki, a ne mehanički procesi.

### Nastanak sustava umjetne inteligencije

Prvi model umjetne inteligencije, model umjetnih neurona, predložili su Warren McCulloch i Walter Pits 1943., a šest godina kasnije Donald Hebb je predložio promjenu algoritma za modifikaciju snage veze između neurona, pravilom koje danas zovemo Hebbovo učenje [6]. Prvi AI implementiran 1955. zvao se „Logic Theorist“ te je imao sposobnost dokazivanja i iskazivanja teorema. Jedanaest godina poslije Joseph Weizenbaum stvorio je prvi *chatbot,* nazvan „ELIZA“, sa ciljem da proučava komunikaciju između čovjeka i stroja. „ELIZA“ je bio jako jednostavan *chatbot* koji se koristio prepoznavanjem uzoraka i supstitucijom znakova.

Prvi široko praktični sustav umjetne inteligencije nastao je 1980. te se zvao „Expert System“. Funkcionirao je na način da je kroz razgovor sa stručnjacima u raznim područjima učio pa kasnije skupljeno znanje nudio klijentima kao uslugu.

Sedamnaest godina kasnije održala se šahovska partija između tadašnjeg svjetskog prvaka Gary Kasparova i računala *Deep Blue,* razvijenog u američkoj kompaniji *International Business Machines Corporation* (IBM). Događaj, prikazan na sliciSlika 1, dočekan je s velikim interesom javnosti te je nakratko postavio umjetnu inteligenciju u centar svjetske pozornosti, posebno zbog ishoda partije u kojoj je *Deep Blue* postalo prvo računalo koje je uspjelo pobijediti svjetskog prvaka.

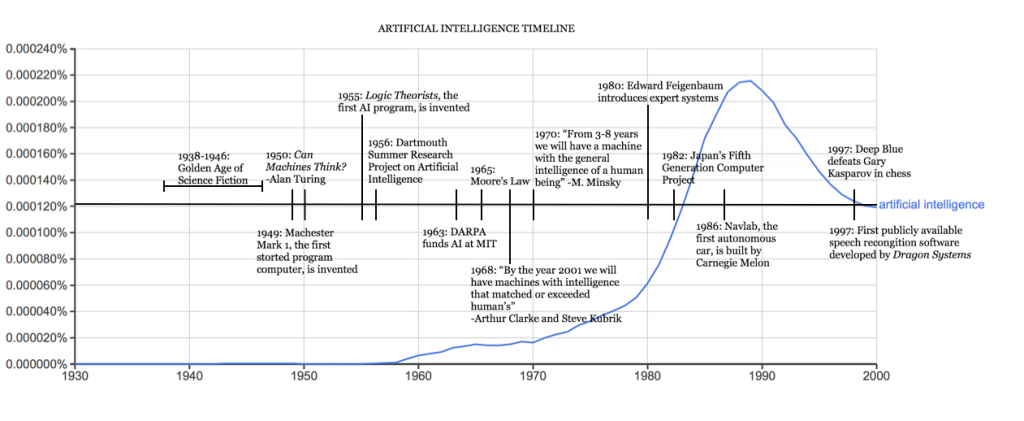
Slika na kojoj se prikazuje Ljudsko lice, osoba, odijevanje, čovjek

Opis je automatski generiran

Slika 1 Kasparov vs Deep Blue [7]

U narednim godinama, Kasparov je podržavao razvoj algoritama za šah, a 1998. pomogao je stvoriti varijantu nazvanu „Advanced chess“ [7]. U ovoj varijanti šaha čovjek i računalo igraju zajedno, kombinirajući brzinu stroja s čovjekovom intuicijom i sposobnosti prepoznavanja uzoraka. „Advanced chess“ je dobar primjer kako povezati čovjeka i stroj, te omogućiti čovjeku da se bavi kreativnijim i zabavnijim poslovima.

Na sliciSlika 2 dan je kronološki prikaz važnijih iskoraka u razvoju AI-a u dvadesetom stoljeću.



Slika 2 Razvoj AI-a [4]

## Trenutno stanje razvoja

Algoritam je konačni niz precizno definiranih naredbi za izvršavanje određenog problema [8]. Pod pojmom AI algoritam smatra se skup naredbi koji omogućava uređajima da uče, analiziraju podatke i donose optimalne odluke na temelju tih podataka. Sposobni su prepoznavati uzorke, razumjeti čovjekov jezik te rješavati razne probleme [9].

Dijele se u tri skupine: *supervised learning algorithms, unsupervised learning algorithms* i *reinforcement learning.* Kod *supervised learning* algoritama, podaci koji su dani sustavu za učenje su labelirani, odnosno skup ulaznih podataka dan je s odgovarajućim izlazom, za razliku od *unsupervised learning* algoritama. Prvi se koriste u prepoznavanju slika, zvuka i slično, dok se drugi koriste za grupiranje podataka. Kod *reinforcement learning* algoritama, sustav uči na osnovu povratnih informacija iz okoline, u obliku nagrada i kazni.

Najpopularniji pristup razvoja AI sustava je strojno učenje (engl. *machine learning*), kojim se omogućava uređajima da uče odrađivanje nekih zadataka bez da su eksplicitno programirani za te zadatke. Podvrsta strojnog učenja je duboko učenje (engl. *deep learning*), koje koristi neuralne mreže u učenju AI-a.

### Svakodnevna primjena

Dvije vrste sustava umjetne inteligencije su softverski i ugradbeni (engl. *embedded*) sustavi. Pod softverske sustave spadaju aplikacije za analizu slika, pretraživači, softver za prepoznavanje lica i glasa, a najrašireniji ugradbeni sustavi su roboti, autonomna vozila te Internet stvari (engl. *Internet of Things*, IoT) [10].

Među korisnim primjenama umjetne inteligencije, danas se ističu prepoznavanje slika te predviđanje stanja. Prepoznavanje slika (engl. *image recognition, classification*) odnosi se na svrstavanje danih podataka u klase, tj. grupe sličnih skupova podataka. Primjene prepoznavanja slika mogu se pronaći u dijagnostici, gdje liječnicima pomaže u prepoznavanju stanja pacijenata, primjerice rezultata magnetske rezonance ili rendgenske snimke (slikaSlika 3). Pokazalo se da sustavi umjetne inteligencije daju mnogo bolje rezultate od ljudi prilikom ranog prepoznavanja bolesti [11].

Slika na kojoj se prikazuje film za bilježenje x-zraka, Medicinsko snimanje, radiologija, medicinski

Opis je automatski generiran

Slika 3 Primjena AI u dijagnostici [12]

Prednosti AI sustava u predviđanju budućih stanja primjenjuju se u trgovanju dionicama. Radi na način da analizira ogromne količine podataka iz prijašnjih stanja pa na osnovu njih pokušava predvidjeti ponašanje burze. Sličan princip koristi se i u modernim vremenskim prognozama.

Umjetna inteligencija koristi se i u pretraživačima kako bi povećala kvalitetu pretraživanja *web*‑a te personalizirala pretrage korisniku. Također, razni algoritmi koriste se u softverima za odabir reklama te u algoritmima društvenih mreža kako bi se informacije prilagodile korisniku. Uskoro se očekuje sve veća primjena u proizvodnji hrane, gdje se želi smanjiti štetni utjecaj na atmosferu.

Danas najpopularniji AI proizvod je ChatGPT, jezični model, koji iznimno dobro oponaša čovjeka u komunikaciji. Zainteresiranost koju je izazvao ChatGPT rezultirala je brojnim pitanjima filozofa i znanstvenika zabrinutih zbog mogućih negativnih posljedica, poput širenja dezinformacija ili zamjene radnika umjetnom inteligencijom.

## Budućnost razvoja

Sustavi umjetne inteligencije trenutno su najprivlačnije područje računalne znanosti. Napredak na znanstvenom području zadnjih godina, osim što je probudio interes za daljnjim razvojem, omogućio je i razvoj sustava sa inovativnim i korisnim primjenama. S obzirom na ulaganja velikih *tech* kompanija te brz razvoj posljednjih godina, očekuje se da zanimanje za AI sustavima neće stagnirati.

Razvoj samovozećih automobila područje je koje se nalazi u centru pozornosti mnogih velikih kompanija, a u samom srcu sustava je AI. Samovozeći automobili te sustav samoupravljajućeg prometa (slika Slika 4), imali bi jako pozitivan utjecaj na život čovjeka.

Slika na kojoj se prikazuje cesta, put, scena, Kopneno vozilo

Opis je automatski generiran

Slika 4 Samoupravljajući promet [13]

# UMJETNA INTELIGENCIJA U ŠAHU

Igre u kojima se najčešće koristi umjetna inteligencija su šah, *Go*, sudoku i slične. Glavni motivi za korištenje umjetne inteligencije u igrama su želja za učenjem od AI-a te potreba za suradnjom čovjeka i AI sustava.

## Šah kao razvojno okruženje umjetne inteligencije

Šah je idealan kandidat za proučavanje razvoja umjetne inteligencije. Pravila igre su lišena faktora sreće. Šah se igra stoljećima, pa postoje ogromne baze odigranih igara, što služi kao *data set* za učenje AI-a. Iako ima konačan broj figura i polja, igra šaha ima praktički beskonačno mnogo mogućih varijacija, što ostavlja prostora za treniranje. Također, postojeća standardizirana metoda ocjenjivanja izvedbe i snage igrača (Elo bodovni sustav) omogućava dobru bazu za ocjenjivanje snage algoritama [14].

## Povijest *chess enginea*

Prvi šahovski program implementiran na elektroničkom računalu nastao je 1951. i mogao je pronaći najbolji potez samo u pozicijama u kojima je bio udaljen od šah mata dva poteza. Program je nastao na Mark 1 računalu, prvom potpuno elektroničkom računalu, koje nije bilo dovoljno jako za računati više od dva poteza u dubinu. Prvi potpuni računalni program za šah nastao je šest godina kasnije i mogao je odigrati cijelu partiju šaha, no za svaki potez mu je trebalo oko 8 minuta. U sedamdesetima su razvijeni prvi program za osobna računala te kasnije i prvo računalo specifično izrađeno za šah. U narednim desetljećima razvijali su se sve bolji programi i računala za šah, a početkom 21. stoljeća započelo je zlatno doba razvoja šahovskih programa, od Rybke i Komoda, do Stockfisha i AlphaZeroa.

### AphaZero

Google-ova AI kompanija Deep Mind 2017. godine razvila je novi *chess engine*, nazvan AphaZero, koji se razlikovao od dotada razvijenih algoritama. Umjesto *alpha-beta* algoritma (koji je optimiziranija varijanta *minmax* algoritma) i aproksimacije linearnom funkcijom, koristili su nelinearnu funkciju, neuralne mreže i Monte Carlo pretragu stabla (engl. Monte Carlo tree search) [15]. Također, algoritam ne posjeduje nikakvo znanje o šahu osim njegovih pravila. Posljedica ovog pristupa bila je mogućnost ponovnog učenja algoritma, za razliku od prijašnjih algoritama koji su se koristili *brute force* pristupom.

### Stockfish

Stockfish je najpoznatiji *chess engine*, nastao 2008. godine. Napisan je u c++ programskom jeziku i temelji se na *minmax* algoritmu s *alpha-beta pruningom* i funkciji za procjenu pozicije. Dizajniran je na način da prvo pretražuje agresivnije poteze, poteze koji tjeraju protivnika na reakciju, kao što su „jedenje“ figura i prijetnja šahom. Ovim pristupom može se, ispravnom implementacijom *alpha-beta* *pruninga*, smanjiti broj poteza koji se ocjenjuju te se posljedično smanjuje mogućnost previda dobre varijacije. Također, Stockfish koristi *null move* strategiju, gdje nekoliko poteza uzastopno igra poteze samo jednog igrača kako bi doznao koje se taktičke ideje mogu realizirati u danoj poziciji [16].

## Dosadašnji rezultati

Nažalost, mnogi znanstvenici ističu kako je razvoj AI sustava za šah postao sam sebi svrha, tj. trenutno se više koristi u svrhu učenja igre šaha, a manje u svrhu razvijanja novih i boljih metoda za razvoj AI-a.

Usprkos tome, šah se potvrdio kao iznimno plodonosno područje za razvoj umjetne inteligencije. Razvijanje AI sustava za šah pridonijelo je prepoznavanju korisnosti naizgled neefikasnog *brute force* pretraživanja, a također je pokazalo da su neke moderne metode AI sustava previše spore za upotrebu u stvarnom vremenu. Šah je imao veliku ulogu u razvoju metoda iterativnog pretraživanja. Iterativnim pretraživanjem se prvo računa stanje igre na dubini jedan potez, zatim na dubini dva poteza, itd. Naizgled, dojam je da se ovim pristupom samo povećava broj poteza koje treba izračunati, jer se prilikom izračuna na dubini N moraju ispočetka računati rezultati dubine N – 1. No, pokazalo se da ispravnom implementacijom, rezultati s manjih dubina mogu iskoristiti u svrhu prepoznavanja koje poteze treba prve izračunati na većim dubinama, što uz pomoć *alpha-beta pruninga,* metode opisane kasnije u radu, rezultira smanjenim brojem pozicija koje se uzimaju u izračun [14].

# RAZVOJ APLIKACIJE ZA IGRU ŠAHA

U svrhu izrade algoritama za šah, implementirana je desktop aplikacija za igranje šaha napisana u programskom jeziku Python. Aplikacija pruža mogućnost igranja protiv realiziranih algoritama te protiv druge osobe, s ili bez vremenske kontrole. Grafičko sučelje (engl. *Graphical user interface*, GUI), implementirano je pomoću *pygame* biblioteke, a još su korištene *time* i *sys* biblioteke.

## Klase

Aplikacija koristi osnovne principe objektno orijentiranog programiranja (engl. *object oriented programming*, OOP), poput klasa, nasljeđivanja i *method* *overridinga*. Na slici Slika 5 sustav je prikazan u obliku dijagrama klasa.

Slika na kojoj se prikazuje tekst, dijagram, Font, paralelno

Opis je automatski generiran

Slika 5 Dijagram klasa

Klasa *Board* središnja je klasa sustava. Sadrži podatke o pozicijama figura na ploči, što sprema u dvodimenzionalnu matricu dimenzija osam redaka i osam stupaca. Također, sadrži podatak koji je igrač na potezu (*white\_to\_move*) te kojom figurom želi igrati (*piece\_to\_move*). Stanja vezana za oba igrača kao što su pozicija kralja, prava na kratku i dugu rokadu (engl. *castling move*) i liste svih figura sprema u dvije instance klase *Player\_state*. U listu *stack\_of\_played\_moves*, implementiranu u obliku stoga, spremaju se svi odigrani potezi u obliku instanci klase *Move*. Klasa *Move* sadrži podatke o figuri koja se pomiče, njenoj početnoj i krajnjoj poziciji, „pojedenoj“ figuri te u slučaju promoviranja pijuna, podatke o promoviranoj figuri.

Metode klase *Board* koriste u odigravanju poteza, provjere stanja igre te procjene vrijednosti pozicije. Na slici Slika 5 su prikazane samo najvažnije metode, koje koriste funkcionalnosti ostalih implementiranih metoda klase. Implementacija metode *move*, koja na osnovu atributa objekta klase *Move* prepoznaje vrstu poteza pa zatim i zove metodu za odigravanje tog poteza, prikazana je na slici Slika 6.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font

Opis je automatski generiran

Slika 6 move metoda

Na slici Slika 7 prikazana je metoda *is\_game\_over* koja provjerava stanje igre, odnosno ima li igrač mogućih poteza te, ako nema, je li kralj u šahu, što bi značilo šah mat i kraj igre. Ako igrač nema poteza te mu kralj nije u šahu, tada igra završava neriješenim ishodom, odnosno pat pozicijom (engl. *stalemate*). Ako niti jedan igrač nema dovoljno figura za šah matirati protivnika, što se detektira metodom *is\_draw\_by\_insufficient\_material*, igra također završava neriješeno.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font

Opis je automatski generiran

Slika 7 is\_game\_over metoda

Klasa *Piece* predstavlja figure, a sadrži podatke o poziciji figure na ploči, tipu i boji figure. Glavna metoda ove klase je *calculate\_legal\_moves* metoda (slika Slika 8), koja računa moguće poteze figure.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font

Opis je automatski generiran

Slika 8 calculate\_legal\_moves metoda

Metoda radi na principu da, pomoću metode *calculate\_pseudo\_legal\_moves* prvo računa moguće poteze figure, bez provjere stavlja li taj potez kralja u šah, što bi potez učinilo nelegalnim (engl. *pseudo legal move*). Svaki potez se provjerava pomoću *is\_move\_valid* metode klase *Board*. Na slici Slika 8 je prikazan slučaj kad figura čiji se potezi računaju nije kralj. U slučaju da se pronalaze potezi za kralja, provjerava se želi li se odigrati *castling* potez jer je tada potrebno provjeriti i je li kralj u šahu na pozicijama koje mora prijeći da bi došao do odredišne pozicije.

Klasa *Bot* predstavlja *bota* koji se koristi jednim od implementiranih algoritama za igru. Sadrži podatke o boji figura *bota*, dubini pretraživanja te podatak je li izašao izvan sebi poznatih početnihpoteza. Metodom *is\_search\_over* provjerava se je li došlo do kraja igre ili dubine pretraživanja, a metodom *Algorithm* predstavljen je algoritam kojim *bot* igra šah.

Prikaz pozicije iz jedne odigrane igre dan je na slici Slika 9.

Slika na kojoj se prikazuje dvoranske igre i sportovi, Igre, igra na ploči, stolna igra

Opis je automatski generiran

Slika 9 - Prikaz pozicije iz jedne odigrane partije

Prilikom klika na figuru kojom se želi odigrati potez, iscrtaju se svi mogući potezi te figure u danoj poziciji (slika Slika 10). Na vrhu slike Slika 10 vidi se i prikaz zbroja vrijednosti figura oba igrača (na sredini) te preostalo vrijeme u sekundama (na stranama).

Slika na kojoj se prikazuje dvoranske igre i sportovi, Igre, igra na ploči, stolna igra

Opis je automatski generiran

Slika 10 Iscrtavanje mogućih poteza figure

U slučaju šaha, polje na kojem se nalazi kralj poprimi crvenu boju, kao na slici Slika 11.

Slika na kojoj se prikazuje dvoranske igre i sportovi, Igre, igra na ploči, stolna igra

Opis je automatski generiran

Slika 11 Kralj u šahu

## Primijenjeni AI algoritmi

U praktičnom dijelu rada dizajnirana su i implementirana dva algoritma za igranje šaha, temeljena na *minmax* te *alpha-beta pruning* algoritmima pretrage. Ovo poglavlje sadrži opis oba algoritma te njihovu implementaciju u programskom kodu.

### *Minmax* Algoritam

*Minmax* algoritam koristi se u *zero-sum, perfect information* igrama. *Zero-sum* igre su one u kojima je dobitak jednog igrača ujedno i gubitak suparnika pa je ukupna promjena u dobitcima uvijek jednaka nuli. Šah je jedna od *zero-sum* igara, jer igrači igraju na jednoj ploči gdje svaki potez donosi promjene obojici igrača i to podjednako, dovodeći kvalitete pozicija u obrnuto proporcionalan odnos, smanjujući kvalitetu pozicije jednog igrača točno onoliko koliko poboljšava poziciju suparnika. Također, šah je *perfect information* igra, tj. igra u kojoj faktor sreće ne utječe na ishod igre. Karakteristika *zero-sum, perfect information* igara je da imaju univerzalno rješenje, što znači da je moguće, uz dovoljno računanja, pronaći optimalnu strategiju, što je cilj *minmax* algoritma.

*Minmax* spada u algoritme pretrage s prvenstvom dubine (engl. *depth first search algorithm*). Koristi se prikazom mogućih stanja pomoću grafa, tj. čvorova čije su međusobne relacije prikazane vezama. Radi na principu da simulira cijelo tzv. stablo igre (engl. *game tree*), odnosno sve potencijalno odigrane poteze do kraja igre, tj. do dna stabla. Svaka istražena pozicija prikazana je čvorom u grafu. Svaka pozicija koja je moguć nastavak prijašnje pozicije (čvor roditelj) povezana je s roditeljskim čvorom (odnos roditelj-dijete), a nalazi se na dubini za jedan većoj od dubine čvora roditelja. Na slicislici Slika 12 prikazan je jednostavan graf.

Slika na kojoj se prikazuje tekst, snimka zaslona, krug, dijagram

Opis je automatski generiran

Slika 12 Prikaz stabla igre u obliku grafa

Nakon što algoritam dođe do dna stabla, čvor na posljednjoj poziciji predstavi vrijednošću, npr. pozitivna vrijednost za pobjedu, nula za remi i negativna vrijednost za poraz. Isto napravi za sve čvorove na posljednjoj dubini stabla koji imaju zajedničkog roditelja te usporedi vrijednosti čvorova. U slučaju da je na potezu igrač koji analizira poziciju (igrač koji želi maksimizirati vrijednost pozicije), algoritam će roditeljskom čvoru tim čvorovima na posljednjoj razini, pridružiti najveću vrijednost čvorova potomaka. Odnosno, među svim mogućim potezima u danoj poziciji, vratit će vrijednost najpovoljnijeg poteza, na taj način računajući vrijednost pozicije roditeljskog čvora. Slično, ako je na potezu suparnik, čvor roditelj će dobiti najmanju vrijednost potomaka, jer će algoritam pretpostaviti da će protivnik odigrati optimalan potez za sebe. Na slikamaSlika 13 i Slika 14 prikazan je proces odabira optimalnog poteza minimizirajućeg igrača.

Slika na kojoj se prikazuje tekst, krug, snimka zaslona, dijagram

Opis je automatski generiran

Slika 13 Minmax graf 1

Slika na kojoj se prikazuje tekst, krug, snimka zaslona, dijagram

Opis je automatski generiran

Slika 14 Minmax graf 2

Nakon što se izračunaju i vrijednosti za potomke drugog čvora na dubini N + 1, dobit će se stanje kao na sliciSlika 15, a na kraju će stablo izgledati kao na sliciSlika 16.

Slika na kojoj se prikazuje tekst, snimka zaslona, krug, dijagram

Opis je automatski generiran

Slika 15 Minmax graf 3

Slika na kojoj se prikazuje tekst, krug, snimka zaslona, dijagram

Opis je automatski generiran

Slika 16 Minmax graf 4

Na danom primjeru da se pročitati bit algoritma. Na dubini N + 1 igrač *minimizer*, među četiri moguća poteza iz pozicije odabire onaj koji mu daje najbolji ishod, onaj koji minimizira vrijednost, kod lijevog čvora to je pozicija s vrijednosti -4, dok je kod desnog to čvor s vrijednosti 1. Na dubini N *maximizer* ima potez te bira potez koji minimizira maksimalnu vrijednost protivnika (iz perspektive *minimizer-*a čvorovi na dubini N + 1 zapravo imaju vrijednosti 4 i -1), u ovom slučaju potez koji ga vodi do pozicije desnog čvora.

Algoritam ovako funkcionira na cijelom stablu igre, a na sliciSlika 17 prikazan je proces izvođenja *minmax* algoritma na nešto većem stablu.



Slika 17 Prikaz rada minmax algoritma [17]

#### Programsko rješenje

Zbog prevelikog broja mogućih jedinstvenih igara šaha, računala nisu sposobna simulirati cijelu igru šaha pa je potrebno prilagoditi *minmax* algoritam. Prvi korak je postaviti dubinu simuliranja, odnosno odrediti koliko poteza će se istražiti. No, ako se neće simulirati igra do samog kraja *game tree-a*, neće se doznati ishod nekog čvora pa je potrebno pronaći način za statičku procjenu vrijednosti pozicije. Statička procjena pozicije temelji se na kvaliteti pozicije, a određuje se heurističkim algoritmom za ocjenjivanje pozicije. Zatim se ta vrijednost rekurzivno vraća i uspoređuje s ostalim vrijednostima potomaka čvora. Heuristički algoritam je onaj koji neće dati savršeno ispravnu procjenu vrijednosti pozicije, već će na osnovu danih parametara i njihovih težinskih faktora dati približnu vrijednost.

U implementiranoj verziji algoritma, algoritam za ocjenjivanje pozicije služi se vrijednošću figura oba igrača, ali i dodatnim karakteristikama pozicije. Na primjer, dodatno se vrednuje sigurnost kralja, povezani i branjeni pijuni, aktivne figure te pijuni na šestom i sedmom redu. Negativnoj ocjeni pridonosili su „poduplani“ i izolirani pijuni, nezaštićeni kralj i slično.

Funkcija kojom je implementiran minmax algoritam sastoji se od tri dijela: detekcije je li čvor koji se analizira list (engl. *leaf node*), tj. završava li pretraga njime, dijela u kojem se odigravaju svi legalni potezi te rekurzivno poziva algoritam te *if-statementa ­*koji odlučuje koje se vrijednosti vraćaju.

Čvor je list ako se nalazi na dubini koja je zadana kao krajnja ili ako je pozicija koju čvor predstavlja rezultirala pobjedom jednog igrača ili neriješenim ishodom. Ako je čvor na zadanoj dubini pretraživanja, ne ulazi se u novi poziv *minmax*, već se vraća statička procjena vrijednosti pozicije čvora. Prikaz programskog rješenja dan je na slici Slika 18.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font, softver

Opis je automatski generiran

Slika 18 Funkcija za detekciju čvora lista i vraćanje odgovarajuće vrijednosti čvora

Procjena vrijednosti pozicije radi se iz perspektive oba igrača, odvojeno računajući vrijednost pozicije bijelog pa crnog. Metoda *static\_eval* (pozvana na slici Slika 18) vraća razliku tih vrijednosti. Vrijednost pozicije pojedinog igrača računa se na osnovu raznih parametara, poput zbroja vrijednosti figura, kvalitete pješaka, aktivnosti figura i sigurnosti kralja kao na slici Slika 19. Kvaliteta pijuna uzima u obzir je li pijun zaštićen drugim pijunima, je li izoliran, postoji li pijun iste boje koji mu je na putu (odnosno jesu li pijuni *doubled*) te ima li pijun slobodan put prema kraju ploče i promociji u jaču figuru.

Procjena vrijednosti šahovske pozicije pruža algoritmu logiku igre i strateško razmišljanje te se smatra ključnim faktorom u snazi algoritma. Kako bi se testiralo to stajalište, u petom poglavlju (5) rada dana je usporedba izvedbi algoritama temeljenih na različitim funkcijama za procjenu pozicija.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font

Opis je automatski generiran

Slika 19 Primjer metode za izračun vrijednosti pozicije

Logika algoritma implementirana je kako je prikazano na slici Slika 20, za *maximizer* igrača. Postavi se početna vrijednost varijabli koja prati vrijednost najboljeg poteza (u primjer *max\_val*) na jako malu vrijednost, za koju vrijedi da će barem jedan potez imati veću vrijednost. Zatim se u petlji odigra svaki potez svake figure *maximizer* igrača te se za svaki potez ulazi u novi poziv minmax funkcije, sa smanjenom vrijednosti varijable *depth* (hrv. dubina) i *False* vrijednosti *is\_max\_playing* varijable, što signalizira da je *minimizer* na potezu. Vrijednost koju vrati poziv minmax, *move\_value*, vrijednost je pozicije do koje se dođe odigravanjem optimalnih poteza oba igrača. Prije no što se poništi posljednje odigrani potez, ažurira se vrijednost *max\_val* varijable, ako je *move\_value* veći od *max\_val.* Ako se trenutni čvor nalazi na početnoj poziciji, što se provjerava *if-statementom* ako je trenutna dubina jednaka početnoj, u zasebne varijable pamte se vrijednosti najboljeg poteza i figure kojom se postiže najbolji potez. Nakon završetka petlje, vraća se vrijednost *max\_val,* odnosno najbolji potez i figura, ako je prvi čvor.

Slika na kojoj se prikazuje tekst, snimka zaslona, softver, operacijski sustav

Opis je automatski generiran

Slika 20 Programsko rješenje minmax algoritma

U slučaju da je potez promocija pijuna poziva se *minmax* za svaku figuru u koju se pijun može promovirati te je daljnji postupak vraćanja vrijednosti isti. Prikaz tog dijela koda izostavljen je sa slike Slika 20 zbog veličine slike pa je prikazan na slici Slika 21.

Slika na kojoj se prikazuje tekst, snimka zaslona, softver, operacijski sustav

Opis je automatski generiran

Slika 21 Promocija pijuna u minmax metodi

Implementacija algoritma kad igra *minimizer* je slična, s promjenama kod praćenja najboljeg poteza, gdje se traži najmanja vrijednost te kod rekurzivnih poziva funkcije, gdje se postavlja *is\_max\_playing*  u *True* vrijednost*.*

Kako bi se izbjeglo bespotrebno računanje kod prvih nekoliko poteza, implementiran je mehanizam prepoznavanja otvaranja i odigravanja slijedećeg poteza (poziv funkcionalnosti vidljiv je na sliciSlika 20). Ovaj pristup je moguć jer u šahu postoji skup otvarajućih poteza (engl. *opening moves, book moves*) koji se smatraju kvalitetnima.

### Alpha-beta pruning

*Alpha-beta pruning* algoritam najpoznatija je varijanta *minmax* algoritma. Karakteristika ovog algoritma je što ima sposobnost prepoznavanja čvorova koji nikako ne mogu doprinijeti u odabiru optimalnog poteza [18].

Na slici Slika 22 prikazan je graf, sličan grafu na čijem je primjeru objašnjen princip rada *minmax* algoritma, a u nastavku je korišten za objašnjenje prednosti *alpha-beta pruning* algoritma.

Slika na kojoj se prikazuje tekst, krug, snimka zaslona, dijagram

Opis je automatski generiran

Slika 22 Alpha-beta graf 1

Kako bi se pokazao način rada *alpha-beta* algoritma, postavljena je vrijednost lijevog djeteta desnog podgrafa u vrijednost -7.

Ono što se da zaključiti iz grafa poviše je da će *minimizer* igrač, s ciljem da odabere poziciju s minimalnom vrijednosti, izabrati opciju s vrijednosti -7 ako zaključi da je ona s najmanjom vrijednosti. Odnosno, vrijednost njegova čvora bit će sigurno manja ili jednaka -7 (slikaSlika 23).

Slika na kojoj se prikazuje tekst, krug, snimka zaslona, dijagram

Opis je automatski generiran

Slika 23 Alpha-beta graf 2

Sad je jasno da će igrač *maximizer*, želeći izabrati potez koji mu donosi najveću vrijednost, zasigurno izabrati potez koji mu donosi vrijednost -4. Do tog zaključka došao je bez saznanja o vrijednosti tri pozicije djeteta desnog čvora na dubini N+1 (slikaSlika 24), jer zna da do njih neće doći, ako igrač *minimizer* igra optimalno.

Slika na kojoj se prikazuje tekst, krug, snimka zaslona, dijagram

Opis je automatski generiran

Slika 24 Alpha-beta graf 3

Ovom metodom izbjegnuto je računanje procjena tri od osam završnih pozicija, a još efikasnija primjena algoritma je ranije u stablu, kao na sliciSlika 25, gdje je izbjegnuto računanje cijelih podstabala.

Slika na kojoj se prikazuje snimka zaslona, crta, dijagram, krug

Opis je automatski generiran

Slika 25 Alpha-beta algoritam [19]

#### Programsko rješenje

Implementacija *alpha-beta* algoritma vrlo je slična implementaciji *minmax* algoritma (slikaSlika 20). Promjena je što se vodi zapis o vrijednosti dvije dodatne varijable, *alpha* i *beta*, koje su dovoljne za pružanje logike samog algoritma. Usporedba njihovih vrijednosti i naredbe akcija koje slijede prikazane su pri dnu slikeSlika 26. Na slici Slika 26 izostavljen je dio koda koji je identičan u implementaciji *minmax* algoritma (slika Slika 20).

Slika na kojoj se prikazuje tekst, snimka zaslona, softver

Opis je automatski generiran

Slika 26 Alpha-beta algoritam implementacija

Vrijednosti *alpha* i *beta* varijabli se prosljeđuju u rekurzivne pozive, a u inicijalnom pozivu njihove vrijednosti su -*INF* (odnosno jako mala vrijednost, u kodu je vrijednost INF varijable postavljena na 10000) za *alpha* te +*INF* za *beta*.

Još jedna promjena je što kod *alpha-beta* algoritma postoji naredba za sortiranje poteza svake figure. Potezi se sortiraju na način da se na prvo mjesto stave *capture* potezi, zatim potezi koji zadaju šah i na kraju ostali (slikaSlika 27). Na ovaj se način bolje koristi algoritam, jer se povećava vjerojatnost da program prvo pronađe bolje poteze, a zatim postupkom obrezivanja zanemari veći broj neoptimalnih poteza.

Slika na kojoj se prikazuje tekst, snimka zaslona, softver

Opis je automatski generiran

Slika 27 Metoda za sortiranje poteza

Opisana implementacija algoritma je deterministička, odnosno za iste ulazne podatke dat će uvijek isti izlaz. Kako *bot* ne biuvijek u istoj poziciji odigravao isti potez, ako postoje podjednako dobri potezi, implementiran je mehanizam odabira nasumičnog poteza iz liste najboljih poteza.

# USPOREDBA RAZLIČITIH ALGORITAMA

*Minmax* i *alpha-beta* *pruning* algoritmi za istu dubinu pretraživanja i isti algoritam procjene vrijednosti pozicije rezultirat će istim najboljim potezom. Jedina razlika dva algoritma je što *alpha-beta* algoritam ima logiku prepoznavanja grana u stablu koje sigurno neće rezultirati potezom boljim od u tom trenutku pronađenog najboljeg poteza, što mu omogućava da smanji broj pretraženih čvorova i vrijeme uloženo u računanje, kao što je prikazano u prvoj usporedbi. S obzirom da povećanje dubine pretraživanja zahtijeva jako veliko povećanje vremena potrebnog za izračun (povećanje za jednu razinu zahtijeva onoliko vremena koliko je potrebno za prijašnju razinu puta broj poteza u poziciji, što je u prosječnoj poziciji oko 30), usporedba će se temeljiti na razlikama u metodama za procjenu vrijednosti pozicije.

## Usporedba broja pretraženih pozicija *minmax* i *alpha-beta* algoritama

Kako bi se pokazala prednost *alpha-beta* *pruning* algoritma nad *minmax* algoritmom, izvršena je usporedba broja pretraživanja koja rade dva algoritma u deset nasumično odabranih pozicija. Prvo su usporedbe izvršene pri dubini pretraživanja od dva poteza, a zatim pri dubini od tri poteza. Pri dubini od dva poteza, *minmax* algoritam je prosječno pretraživao 1136 pozicija, dok je *alpha-beta* algoritam prosječno pretraživao samo 578 pozicija, što rezultira omjerom 1.97. Na dubini pretraživanja od tri poteza, razlika je još veća te je krajnji omjer bio čak 2.17. *Minmax* algoritam je prosječno pretraživao 42894, a *alpha-beta* algoritam samo 19789 pozicija. Za očekivati je da će daljnim povećanjem dubine pretraživanja razlika biti još veća, zbog sposobnosti *alpha-beta pruning* algoritma da rano prepoznaje nepovoljne pozicije. Sitna promjena u algoritmu (pamćenje vrijednosti *alpha* i *beta* varijabli te sortiranje poteza po prioritetu) znatno je doprinijela logici algoritma te pozitivno utjecala na njegove performanse. Kvalitetnijim algoritmom za sortiranje poteza ranije bi se prepoznali najbolji potezi pa bi se dodatno smanjio broj pretraženih pozicija kod *alpha-beta pruning* algoritma.

## Usporedba različitih metoda za izračun vrijednosti pozicije

### Parametri za izračun vrijednosti pozicije

Metoda koja računa vrijednost šahovske pozicije algoritmu je jedini izvor informacija o stanju na šahovskoj ploči, a kao što je prethodno opisano u radu, temelji se na parametrima poput vrijednosti figura na ploči, njihovoj aktivnosti, odnosno opasnosti za protivničkog kralja, sigurnosti kralja i slično.

#### Vrijednost figura

Najjednostavniji pristup procjeni vrijednosti pozicije je pomoću sume vrijednosti figura. Nedostatak ovog pristupa je što se oslanja na sirovi podatak, a zanemaruje bitne strateške parametre.

U programskom rješenju pridijeljene su vrijednosti figurama koje se vide na slici Slika 28. Kralju nije pridijeljena vrijednost jer oba igrača u svakom trenutku imaju točno jednog kralja, pa je razlika vrijednosti uvijek jednaka nuli.

Slika na kojoj se prikazuje tekst, Font, snimka zaslona, broj

Opis je automatski generiran

Slika 28 Vrijednosti figura

Na slici Slika 29 dan je prikaz implementacije funkcije za izračun vrijednosti figura oba igrača. Izračunate vrijednosti se spremaju u atribute instance klase *board* te se pozivom druge metode računa razlika vrijednosti bijelih i crnih figura, što u ovom slučaju predstavlja procjenu vrijednosti stanja na ploči.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font

Opis je automatski generiran

Slika 29 Funkcija za izračun sume vrijednosti figura

#### Struktura pijuna

Struktura pijuna smatra se jednim od najboljih pokazatelja snage šahovske pozicije. Poželjna svojstva pijuna su da su povezani, odnosno da nema izoliranih pijuna, da si međusobno ne sprečavaju put i da se brane. Također, jako su vrijedni pijuni na predzadnjem redu ploče, jer u sljedećem potezu imaju pravo promoviranja u jaču figuru. Izračun vrijednosti različitih struktura pijuna te pojedinih pijuna implementiran je kroz četiri metode, kao što je prikazano na slici Slika 30.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font, broj

Opis je automatski generiran

Slika 30 Metoda za izračun vrijednosti pijuna

#### Sigurnost kralja

Metoda kojom se računa sigurnost kralja uzima u obzir poziciju kralja i zaštićenost pijunima oko njega. Siguran kralj pruža protivniku manje ideja za napad i samim time njegove poteze čini manje opasnima.

#### Pozicije figura

Pozicioniranje figura na ploči bitna je strateška karakteristika pozicije. Vrijednosti pozicioniranja ovisi o tipu i boji figure, a na slici Slika 31 prikazana je raspodjela vrijednosti pozicija za konja crne boje. Vrijednosti za bijelog konja su u obrnutom redoslijedu, a tako i za ostale figure. Na primjeru se vidi da se za konja najviše vrednuju centralne pozicije, i to na petom, šestom i sedmom redu, na kojima konj kontrolira velik dio ploče te je blizu protivničkom kralju.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font, broj

Opis je automatski generiran

Slika 31 Prikaz vrijednosti pozicija za crnog konja

Kralj je iznimka, jer su za njega definirane dvije matrice vrijednosti, ovisno o fazi igre. Tako će kralj u početku igre biti nagrađen za skrivanje u svojoj polovici ploče, dok će pri kraju igre, kad je manje opasnih figura na ploči, biti potaknut sudjelovati u napadu (slika Slika 32).

Slika na kojoj se prikazuje tekst, snimka zaslona, Font, broj

Opis je automatski generiran

Slika 32 Prikaz vrijednosti pozicija za crnog kralja

Ukupna vrijednost pozicioniranja figura dobije se sumom vrijednosti pozicije svake pojedine figure.

### Opis usporedbe

Opće je prihvaćeno, kao što je već navedeno u radu, da je kvalitetna implementacija algoritma za procjenu vrijednosti pozicije ključan faktor u snazi samog *minmax* (ili sličnog) algoritma. Kako bi se provjerila ispravnost te tvrdnje, suprotstavljene su tri implementirane metode za izračun vrijednosti pozicije. Sva tri algoritma za pretragu koriste *alpha-beta* algoritam, na dubini pretraživanja od tri poteza, tako da im je jedina razlika u metodi za izračun vrijednosti pozicije. Metode dijele zajedničke značajke, na način da se funkcionalnosti jednostavnije metode koriste kao dio složenije, kako bi se doznalo koliki utjecaj ima poboljšanje metode na performanse algoritma. Prva metoda je najjednostavnija od svih, a to je metoda temeljena na sumi vrijednosti figura. Druga metoda unaprijeđena je verzija prve, osim sume vrijednosti figura koristi se i logikom za prepoznavanje snage pojedinog pijuna te kraljeve sigurnosti. Posljednjoj, trećoj metodi, uz spomenute značajke prve dvije metode, implementirana je sposobnost pridjeljivanja pozicijske vrijednosti svakoj pojedinoj figuri. Test je izvršen na način da algoritmi odigraju 100 partija protiv drugog algoritma, svaki po 50 s bijelim figurama, sveukupno 300 partija. Očekivana je prevlast u broju pobjeda trećeg algoritma procjene pozicije, a zatim drugog, što bi bilo u skladu s početnom tvrdnjom.

### Rezultati usporedbe

Rezultati testa su potvrdili početnu pretpostavku. Kao što se očekivalo, treći algoritam, koji koristi najkompleksniju metodu izračuna vrijednosti pozicije, odnio je najviše pobjeda, te se pokazalo kako je mnogo kvalitetniji od protivnika, s rezultatima prikazanima u tablicama Tablica 1 i Tablica 2.

Tablica 1 Rezultati algoritma s metodom 1 protiv algoritma s metodom 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Algoritam 1 s bijelim figurama | Algoritam 1 s crnim figurama | Ukupno | Postotak |
| Algoritam 1 | 8 | 6 | 14 | 14% |
| Neriješeno | 21 | 20 | 41 | 41% |
| Algoritam 3 | 21 | 24 | 45 | 45% |

Tablica 2 Rezultati algoritma s metodom 2 protiv algoritma s metodom 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Algoritam 2 s bijelim figurama | Algoritam 2 s crnim figurama | Ukupno | Postotak |
| Algoritam 2 | 13 | 9 | 22 | 22% |
| Neriješeno | 22 | 19 | 41 | 41% |
| Algoritam 3 | 15 | 22 | 37 | 37% |

Protiv najprimitivnijeg, prvog algoritma, pobijedio je u 45 partija, izgubivši samo 14. Algoritam koji se koristio drugom metodom, srednjom po snazi, pokazao se kao dostojan protivnik, no algoritam s metodom 3 pokazao je svoju dominaciju s bijelim figurama. Rezultati iz tablice Tablica 2 ukazuju na kvalitetnu implementaciju metode za izračun pozicijske vrijednosti figura te njen značajan utjecaj na performanse algoritma.

Algoritam s metodom 2 pobijedio je protiv algoritma s metodom 1 u samo 34 partija, ali je izgubio samo 15 (tablica Tablica 3) i tako pokazao da mala promjena u procjeni pozicije, poput prepoznavanja strukture pijuna i sigurnosti kralja, može nedvojbeno pozitivno utjecati na performanse algoritma.

Tablica 3 Rezultati algoritma s metodom 1 protiv algoritma s metodom 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Algoritam 1 s bijelim figurama | Algoritam 1 s crnim figurama | Ukupno | Postotak |
| Algoritam 1 | 8 | 7 | 15 | 15% |
| Neriješeno | 30 | 21 | 51 | 51% |
| Algoritam 2 | 12 | 22 | 34 | 34% |

Iz danih rezultata jasno je vidljiv značajan utjecaj značajki metode procjene pozicije na performanse algoritma, iako su sva tri algoritma koristili istu dubinu pretraživanja.

# ZAKLJUČAK

Filozofi i znanstvenici stoljećima su raspravljali o mogućnosti izrade umjetnog sustava koji će posjedovati karakteristike ljudske inteligencije. Razvojem računarske znanosti u 20. i 21. stoljeću postignuti su temelji za implementaciju različitih sustava umjetne inteligencije, čije se usluge koriste u raznim domenama znanosti, inženjerstva te svakodnevnice. Šah se, kao deterministička i kompleksna igra, rano nametnuo kao povoljna domena istraživanja mogućnosti umjetne inteligencije. Ogromne baze podataka o odigranim partijama i otvaranjima te entuzijastična zajednica šahista pridonijeli su ubrzanom razvoju šahovskih sustava pogonjenih umjetnom inteligencijom, a pobjedama nad najboljim svjetskim šahistima, umjetna inteligencija je dobila na popularnosti. Algoritmi umjetne inteligencije za igranje šaha temelje se na algoritmima pretrage te algoritmima procjene vrijednosti pozicije. Najpoznatiji algoritam pretrage, nazvan *minmax* algoritam, koristi se dubinskim pretraživanjem čvorova stabla igre te pretpostavkom da oba igrača igraju optimalne poteze. Cilj rada bio je realizirati *minmax* algoritam te njegovu optimiziraniju verziju nazvanu *alpha-beta pruning* algoritam, prikazati razliku u performansama između ta dva algoritma te usporedbom performansi algoritama s različitim metodama izračuna vrijednosti pozicije pokazati utjecaj kvalitete metode na algoritam. Program je realiziran u *Python* programskom jeziku, a nudi mogućnosti igranja protiv druge osobe, jednog od implementiranih algoritama ili pokretanja igre između dva algoritma. Za potrebe usporedbe metoda za izračun vrijednosti pozicije, tri implementirane metode, primitivna temeljena na sumi vrijednosti figura, te druge dvije sa skaliranim sposobnostima strateškog zaključivanja, suprotstavljene su međusobno 300 puta, svaki par po 100 puta. Rezultati su bez dvojbe potvrdili pozitivan utjecaj korištenja kvalitetnijih metode za izračun vrijednosti pozicije na performanse algoritma. Također, bilo bi zanimljivo istražiti koliki je utjecaj dubine pretraživanja na performanse i kako se veća dubina uspoređuje s boljom metodom procjene vrijednosti pozicije u odražavanju na performanse algoritma. No, eksponencijalno povećanje broja mogućih pozicija prilikom povećanja dubine, pokazalo se kao ograničavajući faktor. Optimiziranija implementacija same logike šaha, kao i programsko rješenje u bržem programskom jeziku poput jezika c i c++ potencijalno bi dopustili izvršavanje pretrage na većim dubinama te posljedično stvorili više prostora za testiranje performansi različitih algoritama.

LITERATURA

1. Artificial intelligence, B.J.Copeland (19.03.2023.) s Interneta, <https://www.britannica.com/technology/artificial-intelligence/Reasoning>, zadnji pristup: 21.05.2023.
2. S Interneta: <https://en.wikipedia.org/wiki/History_of_artificial_intelligence>, zadnji pristup: 22.05.2023.
3. S Interneta, Benjamin St. George: <https://www.techtarget.com/searchenterpriseai/definition/Turing-test>, zadnji pristup: 23.05.2023.
4. S Interneta, Rockwell Anyoha: <https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>, zadnji pristup: 23.05.2023.
5. S Interneta, John McCarthy: <https://www-formal.stanford.edu/jmc/chinese.html>, zadnji pristup: 23.05.2023.
6. S Interneta: <https://www.javatpoint.com/history-of-artificial-intelligence>, zadnji pristup: 23.05.2023.
7. S Interneta, Demis Hassabis: <https://www.nature.com/articles/544413a>, zadnji pristup: 23.05.2023.
8. S Interneta, https://hr.wikipedia.org/wiki/Algoritam, zadnji pristup: 02.07.2023.
9. S Interneta, <https://www.techtarget.com/searchenterpriseai/tip/Types-of-AI-algorithms-and-how-they-work>, zadnji pristup: 22.06.2023.
10. S Interneta: <https://www.europarl.europa.eu/news/en/headlines/society/20200827STO85804/what-is-artificial-intelligence-and-how-is-it-used>, zadnji pristup: 23.05.2023.
11. S Interneta, <https://www.sentisight.ai/the-use-of-ai-image-recognition-in-medicine/>, zadnji pristup: 22.06.2023.
12. S Interneta, <https://theappsolutions.com/blog/machine-learning/computer-vision-for-healthcare/>, zadnji pristup: 22.06.2023.
13. S Interneta, <https://trafficinfratech.com/planning-improving-travel-safety-with-ai-powered-traffic-management/>, zadnji pristup: 22.06.2023.
14. S Interneta, T.A. Marsland, R.Levinson: <https://www.researchgate.net/publication/220814778_The_Role_of_Chess_in_Artificial_Intelligence_Research>
15. S Interneta, <https://chessentials.com/history-of-chess-computer-engines/>, zadnji pristup: 25.05.2023.
16. S Interneta, <https://blogs.cornell.edu/info2040/2022/09/30/game-theory-how-stockfish-mastered-chess/>, zadnji pristup: 25.05.2023.
17. S Interneta: https://en.wikipedia.org/wiki/Minimax , zadnji pristup: 21.05.2023.
18. S Interneta, chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://umm-csci.github.io/senior-seminar/seminars/spring2017/marckel.pdf, zadnji pristup: 13.06.2023.
19. S Interneta, <https://www.youtube.com/watch?v=l-hh51ncgDI>, zadnji pristup: 25.06.2023.

PRILOZI

Kazalo slika i tablica

Kazalo slika

[Slika 1 Kasparov vs Deep Blue [7] 4](#_Toc140133571)

[Slika 2 Razvoj AI-a [4] 5](file:///C:\Users\vjera\OneDrive\Radna%20površina\Zavrsni_doc\Završni\Završni_final.docx#_Toc140133572)

[Slika 3 Primjena AI u dijagnostici [12] 6](#_Toc140133573)

[Slika 4 Samoupravljajući promet [13] 7](#_Toc140133574)

[Slika 5 Dijagram klasa 10](#_Toc140133575)

[Slika 6 move metoda 11](#_Toc140133576)

[Slika 7 is\_game\_over metoda 12](#_Toc140133577)

[Slika 8 calculate\_legal\_moves metoda 12](#_Toc140133578)

[Slika 9 - Prikaz pozicije iz jedne odigrane partije 13](#_Toc140133579)

[Slika 10 Iscrtavanje mogućih poteza figure 14](#_Toc140133580)

[Slika 11 Kralj u šahu 15](#_Toc140133581)

[Slika 12 Prikaz stabla igre u obliku grafa 16](#_Toc140133582)

[Slika 13 Minmax graf 1 17](#_Toc140133583)

[Slika 14 Minmax graf 2 17](#_Toc140133584)

[Slika 15 Minmax graf 3 17](#_Toc140133585)

[Slika 16 Minmax graf 4 18](#_Toc140133586)

[Slika 17 Prikaz rada minmax algoritma [17] 18](file:///C:\Users\vjera\OneDrive\Radna%20površina\Zavrsni_doc\Završni\Završni_final.docx#_Toc140133587)

[Slika 18 Funkcija za detekciju čvora lista i vraćanje odgovarajuće vrijednosti čvora 19](#_Toc140133588)

[Slika 19 Primjer metode za izračun vrijednosti pozicije 20](#_Toc140133589)

[Slika 20 Programsko rješenje minmax algoritma 21](#_Toc140133590)

[Slika 21 Promocija pijuna u minmax metodi 22](#_Toc140133591)

[Slika 22 Alpha-beta graf 1 23](#_Toc140133592)

[Slika 23 Alpha-beta graf 2 23](#_Toc140133593)

[Slika 24 Alpha-beta graf 3 24](#_Toc140133594)

[Slika 25 Alpha-beta algoritam [19] 24](#_Toc140133595)

[Slika 26 Alpha-beta algoritam implementacija 25](#_Toc140133596)

[Slika 27 Metoda za sortiranje poteza 26](#_Toc140133597)

[Slika 28 Vrijednosti figura 28](#_Toc140133598)

[Slika 29 Funkcija za izračun sume vrijednosti figura 28](#_Toc140133599)

[Slika 30 Metoda za izračun vrijednosti pijuna 29](#_Toc140133600)

[Slika 31 Prikaz vrijednosti pozicija za crnog konja 30](#_Toc140133601)

[Slika 32 Prikaz vrijednosti pozicija za crnog kralja 30](#_Toc140133602)

Kazalo tablica

[Tablica 1 Rezultati algoritma s metodom 1 protiv algoritma s metodom 3 31](#_Toc140133603)

[Tablica 2 Rezultati algoritma s metodom 2 protiv algoritma s metodom 3 32](#_Toc140133604)

[Tablica 3 Rezultati algoritma s metodom 1 protiv algoritma s metodom 2 32](#_Toc140133605)

Popis oznaka i kratica

AI Artificial Intelligence

IoT Internet of Things

GUI Graphical user interface

OOP Object oriented programming

IBM International Business Machines Corporation

SAŽETAK I KLJUČNE RIJEČI

**Sažetak**

*U ovom radu objašnjena je primjena algoritama umjetne inteligencije za igranje šaha. Ukratko je predstavljen povijesni razvoj sustava umjetne inteligencije te njegove današnje primjene i smjer razvoja. Predstavljeni su razni algoritmi umjetne inteligencije s naglaskom na algoritme koji svoju primjenu nalaze u šahu. Ukratko su predstavljeni najkvalitetniji algoritmi za igranje šaha te su detaljno istraženi i objašnjeni minmax i alpha-beta pruning algoritmi, koji su temeljeni na prikazu mogućih stanja šahovske ploče kroz stablo igre s čvorovima. Implementacija dvaju navedenih algoritama, kao i potpunog programa za igru šaha, dana je u programskom jeziku Python. Dva parametra o kojima ovise performanse algoritama su dubina pretraživanja te metoda za izračun vrijednosti pozicije. Koristeći različite metode za izračun vrijednosti pozicije pri jednakim dubinama pretraživanja, pokazan je utjecaj kvalitete metode na performanse algoritma.*

**Ključne riječi**

*Umjetna inteligencija, minmax, alpha-beta obrezivanje, algoritam, šah*

SUMMARY AND KEYWORDS

**Title**

APPLICATION OF ARTIFICIAL INTELLIGENCE IN CHESS

**Summary**

*In this bachelor thesis, the application of artificial intelligence algorithms for playing chess is explained. The historical development of the artificial intelligence system and its current applications and direction of development are briefly presented. Various artificial intelligence algorithms are presented with an emphasis on algorithms that find their application in chess. The strongest algorithms for playing chess are briefly presented, while the minmax and alpha-beta pruning algorithms, which are based on the representation of the possible states of the chessboard through a game tree with nodes, are explored and explained in detail. The implementation of the two mentioned algorithms, as well as the complete program for the game of chess, is given in the programming language Python. The two parameters on which the performance of the algorithms depends are the search depth and the method for calculating the position value. The influence of the method for calculating the position value on the algorithm performance was demonstrated by comparing algorithms with different methods at equal search depths.*

**Keywords**

*Artificial intelligence, minmax, alpha-beta pruning, algorithm, chess*