Sustavi otvorenog koda za igranje šaha

Kovač, Hrvoje

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Hrvoje Kovač

SUSTAVI OTVORENOG KODA ZA IGRANJE ŠAHA

Diplomski rad

Voditelj rada:
prof.dr.sc. Luka Grubišić

Zagreb, srpanj, 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Poštovani profesore Grubišić, puno Vam hvala na pomoći i pruženoj potpori pri izradi ovog rada.
Također, želio bih se zahvaliti svim prijateljima i kolegama, a posebno MNI-ovcima (uključujući i profesore), uz koje su dvije godine diplomskog studija proletjele.
Na kraju, najveće hvala mojoj djevojci, roditeljima, bratu i ostatku obitelji na pruženoj podršci i strpljenju.

Sadržaj

Uvod

Konstrukcija programa koji će igrati šah na racionalan način bila je izazov za pionire umjetne inteligencije pedesetih godina prošlog stoljeća. Napretkom tehnologije računalni programi za igranje šaha postajali su sve jači i krajem prošlog stoljeća šahovsko računalo Deep Blue pobijedilo je u meču svjetskog prvaka Garija Kasparova. Razvojem područja strojnog učenja, i šahovski su programi u sebi počeli sadržavati neuronske mreže, automatizaciju parametara i nove načine pretraživanja stabla. Pionir na ovom području je tvrtka DeepMind čiji je poznati program AlphaGo 2016. godine pobijedio svjetskog prvaka u igri Go, Leea Sedola. Manje je poznato da je nakon programa AlphaGo konstruiran program AlphaZero koji je krajem 2017. godine u meču pobijedio najjači šahovski program na svijetu, Stockfish, i tako najavio novo razdoblje u konstrukciji šahovskih programa.

Cilj ovog rada je istražiti način funkcioniranja programa za igranje šaha, konstrukcija programa za igranje šaha korištenjem poznatih algoritama iz područja umjetne inteligencije i pregled modernih trendova u svijetu računalnog šaha.
U prvom dijelu rada će biti opisana povijest računalnog šaha i njegova važnost za razvoj umjetne inteligencije, te razni problemi prisutni kod konstrukcije dobrog šahovskog programa. U drugom dijelu rada će biti pobliže opisani standardni dijelovi svakog šahovskog programa: reprezentacija ploče i poteza, evaluacija i pretraživanje. Na kraju tog dijela nalazi se opis vlastite implementacije navedenih dijelova programa.
U trećem dijelu rada opisani su načini automatizacije u modernim šahovskim sustavima s naglaskom na modernim tehnikama podržanog učenja. Opisane tehnike još uvijek se ne mogu koristiti na stolnim računalima, ali predstavljaju novi pristup računalnom šahu.

Poglavlje 1

Općenito o računalnom šahu

> Šah je igra za dva igrača na kvadratnoj ploči veličine 8x8, u kojoj svaki igrač ima 16 figura koje se mogu micati u skladu s dogovorenim pravilima. Cilj igre je matirati protivničkog igrača. [1] [1]

Osnovna pravila šaha nisu se mijenjala od 19. stoljeća, a uključuju pravila pomicanja figura i 3 posebna pravila: rohadu (zamjena mjesta kralja i topa), en passant (uzimanje pješaka neposredno pored vlastitog) i promociju pješaka (pretvorba u drugu figuru koja nije kralj). Igra je gotova kad je kralj matiran, tj. kad je kralj u šahu (napadnut) i nije moguće obraniti se od šaha niti pomaknuti ga. Kognitivna usmjerenost, jednostavna pravila i neovisnost o socio-ekonomskim faktorima, utjecali su na popularnost igre pa je šah danas najpopularnija strateška igra na ploči u svijetu s više od 600 milijuna redovnih igrača [3]. Inteligencija se tradicionalno definira kao sposobnost učenja i korištenja stečenog znanja i vještina, a vještina igranja šaha, prema nedavnim istraživanjima [2], pozitivno korelira s vizualno-prostornom, verbalnom i matematičko-logičkom inteligencijom. Rani istraživači u području umjetne inteligencije počeli su se intenzivno baviti konstrukcijom šahovskog programa, nazvavši šah "drosdophilom " (vinska mušica) umjetne inteligencije [19]. Uspoređujući računalni šah sa životinjom koja je bila temelj za razvitak moderne genetike želi se naglasiti da je problem konstrukcije programa koji igra šah, iako konceptualno jednostavan, bio osnova za proučavanje i razvoj kompleksnih grana umjetne inteligencije.

Razvojem teorije igara postalo je moguće formalizirati određene vrste igara i strategija koje igrači koriste. Ovaj pristup pokazao se izuzetno koristan i za računalni šah jer se na rezultatima poput poznatog Von Neumannovog minimaks teorema [28] bazira moderni računalni šah. U teoriji igara igrom nazivamo bilo kakvu konfliktnu situaciju među više suprotstavljenih strana (nazvanih igračima). Igre se obično klasificiraju s obzirom na neka njihova

[1] prijevod: H.K.

svojstva, a šah spada u kategoriju konačnih igara (konačno mnogo strategija) s potpunim informacijama i sumom nula (nazvanih još i matrične igre) za dva igrača. Strategijom zovemo mogućnosti za ponašanje igrača, tj. u slučaju šaha to su svi potezi na raspolaganju igraču. Igra s potpunim informacijama je ona u kojoj igrač uvijek ima dostupne sve informacije (nema skrivenih faktora), što je u šahu, očito, istina. Igra sa sumom nula znači da je gubitak prvog igrača jednak dobitku drugog, i obratno. Obično se rezultati odabira strategije zapisuju u matrici koja se zove matrica isplata. Matrica isplata za šahovsku partiju izgleda ovako:

Tablica 1.1: Matrica isplata za šah

		Crni		
		Pobjeda	Remi	Poraz
Bijeli	Pobjeda	x	x	1,-1
	Remi	x	0,0	x
	Poraz	-1,1	x	x

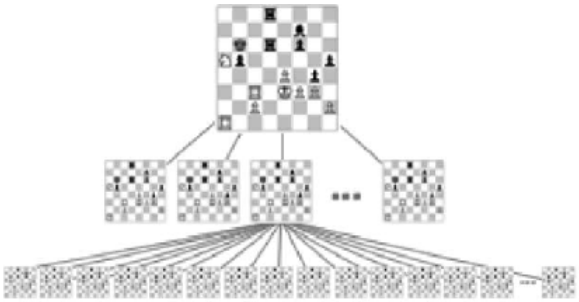
Temelj modernog računalnog šaha bazira se na sljedećem teoremu:

Teorem 1 (Minimax teorem)**.** *Svaka konačna, matrična igra za dva igrača ima optimalne miješane strategije. Neka su X i Y miješane strategije za igrače A i B, i neka je U matrica isplata. Tada je*

$$\max_X \min_Y X^T U Y = \min_Y \max_X X^T U Y = v,$$

gdje je v vrijednost igre, a X i Y rješenja igre. [28]

Interpretacija Von Neumannovog minimaks teorema u računalnom šahu glasi ovako: U svakoj poziciji pronađi najbolji potez prvog igrača (onaj koji maksimizira njegov dobitak), zatim pronađi najbolji potez protivnika (onaj koji minimizira njegov gubitak), zatim pronađi najbolji idući potez prvog igrača, itd. Konačna vrijednost igre je ona koja se postiže nakon što se pretraže svi mogući potezi i igra završi. Partiju šaha možemo prikazati stablom pretraživanja mogućih poteza, kojem je korijen trenutna pozicija, grane su svi mogući legalni potezi, a čvorovi su sve rezultirajuće pozicije.



Slika 1.1: Šah kao stablo [2]

Ako s b označimo faktor grananja tj. broj mogućih poteza [3] u poziciji, a sa d dubinu rješenja u stablu pretraživanja, tada se broj pretraženih čvorova do dubine d može izračunati po sljedećoj formuli:

$$N = b^d.$$

Empirijski je utvrđeno da je prosječan broj poteza nekog igrača u središnjici otprilike 35, a prosječna partija traje 40 poteza crnog i 40 poteza bijelog tj. 80 poteza pa uvrštavanjem dobivamo da je broj čvorova koje je potrebno pretražiti:

$$N \approx 35^{80} \approx 3.35 \cdot 10^{123}$$

Moderni paralelizirani šahovski programi mogu dostići brzinu od 70 milijuna pretraženih čvorova u sekundi [16], što znači da bi trenutno najbrži programi pretražili cijelo stablo u otprilike $1.518 \cdot 10^{108}$ godina.
Čak i primjenom superračunala, potpuno je jasno da je nemoguće pretražiti cijelo stablo i da će šah u doglednoj budućnosti ostati nerješiv primjenom grube sile. Drugi naivni pristup bio bi pokušati napraviti neku vrstu rječnika svih pozicija i za svaku poziciju predložiti najbolji potez (izračunat ili preporučen od strane velemajstora). Shannon prvi iznosi procjenu o broju mogućih različitih pozicija: $\frac{64!}{32!(8!)^2(2!)^6} \approx 10^{43}$ što uz ograničenje od 300 bajtova po poziciji čini pohranu takvog rječnika nemogućom (potrebno je otprilike $3 \cdot 10^{24}$ zettabytea prostora).

Pionir razvoja programa sposobnih za igranje šaha bio je Claude Shannon, koji je prvi opisao neke koncepte i probleme pri konstrukciji šahovskog programa. U najpoznatijem članku o računalnom šahu, Programiranje računala za igranje šaha [22], Shannon je

[2] http://bit.ly/2Dwfzv9 (18.1.2018.)

[3] U šahovskom žargonu potez podrazumijeva pomicanje figure bijelog i pomicanje figure crnog. Svjetska šahovska federacija FIDE definira potez kao pomicanje figure jednog igrača. U engleskoj literaturi razlikuje se termin move - potez bijelog i crnog i ply - potez nekog igrača

uvidio da bi konstrukcija programa zahtijevala nadilaženje raznih drugih problema slične prirode i većeg značaja, poput programa za logičku dedukciju, programa za preusmjeravanje telefonskih poziva, i sl . Shannon je u članku spomenuo i evaluacijsku funkciju koja bi za svaku poziciju vratila vrijednost +1, 0 ili -1 ovisno o tome tko pobjeđuje i najbolji put do kraja igre. Budući da je nemoguće pretražiti cijelo stablo, ideja prvih tvoraca šahovskih programa bila je pretražiti stablo do određene dubine i vratiti vrijednost evaluacijske funkcije koja bi približno ocijenila koliko je ta pozicija dobra za igrača. Vrijednost koju vraća evaluacijska funkcija nije nužno točna, budući da daje uvid samo u trenutno stanje ploče. Tipičan problem kod procjene pozicije je ako pozicija nije statična. Na primjer, ako je završni čvor pretrage pozicija u kojoj jedna strana ima veliku materijalnu prednost zato što je uzela protivnikovu damu svojom damom, evaluacijska funkcija tu će poziciju ocijeniti dobivenom, a možda će se u idućem potezu materijalno stanje opet izjednačiti. Ovaj problem zove se efekt horizonta i izuzetno je važan u računalnom šahu. Zbog efekta horizonta i sličnih problema, evaluacijska funkcija se mora nadopuniti heuristikama koje procjenjuju određene tipove pozicija. Shannon u radu navodi statičke parametre koje bi dobra evaluacijska funkcija trebala uzeti u obzir: materijalni odnos snaga, mobilnost pojedine strane, sigurnost kralja i pješačke slabosti. Ovi parametri najznačajniji su dijelovi svih današnjih evaluacijskih funkcija.

Od početaka programiranja računala za igranje šaha postojale su dvije ideje kako tražiti najbolje poteze:

1. Pretraživati sve moguće legalne poteze do određene dubine i pomoću evaluacijske funkcije ih rangirati

2. U danoj poziciji pokušati reducirati broj poteza na 3-4 najbolja, za njih dalje tražiti najbolje odgovore itd.

U početku su programeri, uključujući i svjetskog šahovskog prvaka Mihaila Botvinika, bili uvjereni u ispravnost drugog pristupa, ali pojavom $\alpha-\beta$ podrezivanja (objašnjenog kasnije), prvi se pristup pokazao boljim i gotovo svi današnji programi koriste podrezivanje unatrag.

Prilikom programiranja šahovskog programa obično se u računalo žele prenijeti još neka tipična ljudska znanja. Prije svega, u ta znanja spadaju neki poznati načini započinjanja partije koji se zovu otvaranja. Otvaranje je bilo koji niz poteza za koji se empirijski utvrdilo da je dovoljno dobar, to jest, da osoba koja igra otvaranje ne gubi u nekoliko poteza. O šahovskim otvaranjima napisano je daleko najviše šahovskih knjiga i ono je predmet intenzivnog proučavanja i danas. Zbog toga je u gotovo svim današnjim šahovskim programima implementirana neka vrsta knjige otvaranja koja sadrži niz varijanti.
Stvaranje dobre knjige otvaranja predstavlja interesantan izazov programerima jer:

- nisu sve varijante jednake duljine
- nisu sve varijante dobre tj., ako je igrač igrao tu varijantu i na kraju dobio partiju to ne znači da je varijanta dobra
- ljudska procjena neke varijante ne slaže se ponekad s računalnom procjenom. Dapače, u novije vrijeme, neke varijante koje su bile smatrane lošima vratile su se u praksu jer su računala pokazala moguća poboljšanja.

Još jedan segment igre koji se danas sve manje pretražuje su završnice, to jest pozicije na kraju partije s malo figura. Kod završnica je problem to što će evaluacijska funkcija često precijeniti šanse strane s višom vrijednosti materijala, dok su igračima poznati određeni tipovi teoretskih završnica za koje se pokazalo da su uz pravilnu igru remi. Od 1970-ih godina postoji ideja da se takve pozicije s manje figura na ploči spremaju u određenu vrstu baze podataka koja bi se služila retrogradnom analizom za izračun je li određena pozicija dobivena, remi ili izgubljena. Takve specijalizirane baze završnica, zvane tablebase, u sebi imaju spremljene sve moguće završetke partija za svaku moguću (legalnu) konstelaciju figura. Trenutno najveća kompletna baza završnica (tablebase) je Lomonosov baza završnica za 7 figura, veličine 140 TB, koja obuhvaća sve pozicije dva kralja i pet figura po izboru. I u ovom segmentu igre danas ljudi uče od računala budući da su neke pozicije, koje su prethodno smatrane remijima, zahvaljujući računalima, dokazano dobivene. Dapače, pojavom baza završnica pomaknule su se granice ljudskog shvaćanja:

> Praćenje poteza baze završnica je jezivo iskustvo. Velemajstor ih ne razumije bolje od nekoga tko je tek naučio igrati šah [4] [7]

Zahvaljujući Shannonovim idejama, računalni šah počeo se razvijati 50-tih godina prošlog stoljeća, a prvi zabilježeni program, koji je igrao na nekom turniru, bio je MacHack IV, koji je 1957. odigrao 5 partija protiv ljudskih igrača i postigao rezultat od 0.5/5 i performans rejting [5] od 1243. Deset godina kasnije, poboljšana je verzija istog programa odigrala 2/4 (2 dobivene i 2 izgubljene partije) uz performans rejting od 1640. Do kraja desetljeća računalni šah je postao toliko popularan da je 1970. održano prvo Sjevernoameričko računalno šahovsko prvenstvo. Nekoliko godina kasnije, zbog velikog se interesa, počelo održavati i Svjetsko računalno šahovsko prvenstvo (WCCC), koje se niz godina održavalo

[4] prijevod: H.K.

[5] Elo rejting sistem je način rangiranja u šahu pri čemu razlika u rejtingu služi kao prediktor rezultata. Za igrače jednakog rejtinga očekuje se da u meču ostvare jednak broj pobjeda, za rejting razliku od 100 bodova očekivani rezultat pobjede onog s većim rejtingom je 64%, za razliku od 200 bodova očekivano je 76 % itd. Rejting početnika iznosi oko 1000, majstora 2300, dok je za titulu velemajstora potrebno imati rejting viši od 2500. Više o rejtingu može se naći u članku Glickmana [12]

u Stockholmu. Do 1980. godine sve je više ljudi iz hobija pisalo programe koji su mogli igrati šah, ali su ti programi bili namijenjeni osobnim računalima i nisu se mogli mjeriti s programima osmišljenim za rad na najjačim akademskim računalima. Zbog toga se osim standardnog svjetskog prvenstva od 1980. počelo održavati i Svjetsko mikroračunalno šahovsko prvenstvo. Do kraja devedesetih, brzina i mogućnosti računala toliko su porasle da se većina mikroračunala mogla mjeriti s najjačim akademskim računalima. Zbog toga je 2001. godine ovo natjecanje ukinuto i od tada se za WCCC može koristiti bilo kakav hadrver. Najznačajnija godina za računalni šah bila je 1997., zbog povijesnog meča Kasparov protiv računala Deep Blue. Samo 8 godina ranije, Kasparov je pospdrno zamolio inženjere tadašnjeg IBM-ovog računalnog programa Deep Thought da ga "nauče da ranije predaje" , izjavivši pritom "Ne mogu zamisliti život sa saznanjem da je računalo jače od ljudskog mozga."[6] [23]

Rapidnim razvojem tehnologije, već za nekoliko godina, IBM je dizajnirao super-računalo Deep Blue, koje se sastojalo od 30 specijalno izrađenih procesora sposobnih pretraživati 200 000 000 čvorova u sekundi [13]. Deep Blue, u slavnom je meču pobijedilo Kasparova, tadašnjeg svjetskog prvaka i po mišljenju mnogih, najboljeg šahista svih vremena, rezultatom 3.5-2.5. Time je postalo prvo računalo koje je pobijedilo svjetskog prvaka u klasičnom šahu, nanijevši mu, usput, najteži poraz u karijeri (partija je trajala samo 19 poteza). Pobjeda računala nad svjetskim prvakom u kompleksnoj igri poput šaha imala je važno simboličko značenje za cijelo polje umjetne inteligencije, a označava i prekretnicu u modernoj šahovskoj povijesti.

Konačni dokaz dominacije šahovskih programa nad ljudima bili su mečevi Man vs. Machine, svjetskog timskog prvenstva, odigrani 2004. i 2005. godine. U oba meča, tada vodeći šahovksi programi Hydra, Fritz i Deep Junior, uvjerljivo su dobili neke od najjačih igrača svijeta. Godine 2014., održan je mini-meč od 4 partije između programa Stockfish (tada rejtinga većeg od 3200) i američkog igrača Hikarua Nakamure (tada 5. šahista u svijetu, rejtinga 2770), koji je u prve dvije partije mogao igrati uz pomoć programa Rybka (rejting 3100 bodova), a u druge dvije partije Stockfish je počinjao partiju sa 7 pješaka (jedan pješak nasumično je odabran i maknut). Meč je završio pobjedom Stockfisha 3-1 (1.5-0.5 u oba segmenta) i time potvrdio dominaciju računalnih programa nad ljudskim igračima.

Od 2001. godine započinje moderni oblik svjetskog računalnog šahovskog prvenstva koji dozvoljava natjecateljima potpunu slobodu što se tiče sklopovlja, a paralelno se održava i Svjetsko šahovsko softversko prvenstvo (WCSC) gdje se inzistira da svi programi imaju na raspolaganju isti hardver. Ipak, zbog izuzetno strogih kriterija za prisustvovanje na natjecanju, većina autora trenutno najjačih šahovskih sustava već niz godina bojkotira ovo službeno svjetsko prvenstvo, pa je kao mjera jačine programa uzeta rejting lista CCRL. Computer chess rating list rangira programe prema jačini Elo rejtinga koristeći Bayes

[6] prijevod H.K.

Elo paket i partije programa odigranih na jednakom hardveru. Slijedi rang lista najjačih šahovskih programa:

Tablica 1.2: Najboljih 5 šahovskih programa na svijetu

Program	Rejting
SugaR XPrO 1.2	3410
Komodo 11.2	3395
Houdini 6	3382
Deep Shredder 13	3291
Fire 5	3273

Za usporedbu, najveći Elo rejting svjetskog prvaka Magnusa Carlsena bio je 2882, što dovoljno govori o dominaciji šahovskih računala. U ovoj skupini, SugaR XPrO je sustav otvorenog koda i izvedenica najpoznatijeg i najjačeg sustava zvanog Stockfish. Komodo, Houdini i Deep Shredder su komercijalni softveri, a Fire 5 je besplatan. Komercijalni programi za igranje šaha u današnje su vrijeme rijetka pojava budući da postoje besplatne alternative otprilike jednake jačine. Iako su komponente i algoritmi na kojima se temelje šahovski programi poznati već pedesetak godina, svijet računalnog šaha još uvijek se razvija, a konstrukcija programa za igranje šaha još uvijek se smatra dobrim praktičnim zadatkom za vježbanje programiranja. Nemali je broj ljudi diplomirao i doktorirao isključivo na temama vezanim uz računalni šah budući da se u pozadini često nalaze interesantni matematički problemi poput De Bruijneovih nizova (kombinatorni problem korišten kod bitboard pristupa), optimizacijskih problema (poboljšavanje evaluacijske funkcije) i novih načina pretrage stabla (MTD(f), NegaScout).

Novije primjene umjetne inteligencije u šahovskim programima uključuju primjenu strojnog učenja kao alternativu pretraživanju i evaluacijskoj funkciji. U posljednjih dvadesetak godina razvijani su sustavi kod kojih se parametri evaluacijske funkcije određuju pomoću strojnog učenja, a 2015. godine napravljen je i prvi šahovski program koji je odbacio koncept klasičnog pretraživanja i, koristeći nenadzirano učenje, u 72 sata došao do Elo rejtinga od 2400 bodova, što je impresivan rezultat. U budućnosti je za očekivati da će ovaj novi pristup pomoću strojnog učenja dobiti na popularnosti.

Poglavlje 2

Osnovne komponente šahovskog računala

U ovom poglavlju bit će ukratko opisane osnovne komponente programa za igranje šaha, koncepti koji se nalaze iza određene implementacije i vlastita implementacija u programskom jeziku Python. U donju podjelu nije uključeno korisničko sučelje budući da šahovski program (engine) može funkcionirati i bez tog dijela, u sklopu nekog drugog programa (takav je npr. Stockfish). Svaki šahovski program sastoji se od tri osnovna dijela:

- reprezentacija ploče, figura, pravila i legalnih poteza
- pretraživanje u dubinu
- evaluacija pozicija u čvorovima gdje prestaje pretraživanje

Kod gotovo svih današnjih programa vidljiva je podjela na ove 3 komponente.

2.1 Reprezentacija ploče i figura

Reprezentacija ploče (polja i figura), predstavlja ”back-end” svakog šahovskog programa, budući da ispravna i brza reprezentacija omogućuje lakše generiranje poteza, vraćanje poteza, vodi računa o nekim specijalnim pravilima poput prava na rohadu ili en passant. Budući da je u vrijeme početaka rada na šahovskim programima memorija bila osnovni problem, osmišljeni su razni pametni načini implementacije ploče kao niza od 64 bajta pri čemu je svaki bajt nosio informaciju o tome koja se figura nalazi na tom polju. Ovaj način trošio je minimalnu količinu memorije, ali je prilikom generiranja poteza zahtijevao stalnu provjeru legalnosti poteza (nalazi li se figura još uvijek na ploči). Naknadno je ovo rješenje poboljšano dodavanjem dva reda polja označenih kao van ploče (sentinel value).

Na ovaj način provjera legalnosti poteza npr. skakača ili lovca, svodila se na jednu logičku operaciju. Postoje tri načina reprezentacije ploče:

- orijentiran na figure
- orijentiran na polja
- hibridni pristup

Kod pristupa orijentiranih na polja, ploča se reprezentira nizom bajtova gdje svaki bajt nosi informaciju o tome koja se figura nalazi na tom polju. Najpopularnija moderna implementacija je 0x88 koja koristi 2D niz duljine 16x8 (dvostruke duljine od veličine ploče). Struktura u kojoj se sprema je jednodimenzionalno polje s elementima zapisanim u heksadecimalnom zapisu što omogućava brzu provjeru je li destinacija određene figure legalno polje.

8	70	71	72	73	74	75	76	77	78	79	7a	7b	7c	7d	7e	7f
7	60	61	62	63	64	65	66	67	68	69	6a	6b	6c	6d	6e	6f
6	50	51	52	53	54	55	56	57	58	59	5a	5b	5c	5d	5e	5f
5	40	41	42	43	44	45	46	47	48	49	4a	4b	4c	4d	4e	4f
4	30	31	32	33	34	35	36	37	38	39	3a	3b	3c	3d	3e	3f
3	20	21	22	23	24	25	26	27	28	29	2a	2b	2c	2d	2e	2f
2	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
1	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	a	b	c	d	e	f	g	h								

Slika 2.1: 0x88 bitboard heksadecimalno

Za svako polje korišten je jedan četverobitni broj (nibble). Na primjer, 0x47 označava peti red i osmi stupac. Polja koja su legalna imaju gornja četiri bita jednaka nuli, dok polja koja se nalaze van ploče imaju barem jedan gornji bit jednak jedinici. Pri pomicanju lovca

s polja e1 [1] koje ima oznaku 0x04 do polja h4, koje ima oznaku 0x37, polja koja lovac napada uzduž te dijagonale mogu se provjeriti dodavanjem konstante 0x11 na prethodno polje. Pri pomicanju lovca dalje od polja h4 (0x37), dolazi se na polje 0x48 kod kojeg je u drugom broju najveći bit jednak jedan, što znači da se to polje nalazi izvan ploče. Provjera legalnosti poteza lovcem postiže se logičkom operacijom AND polja i broja 0x88. Ako je rezultat te operacije različit od 0 polje se nalazi izvan ploče. Ovakva provjera pomoću nekoliko logičkih operacija omogućila je brži pristup uz neznatno veću memoriju i danas se raširenо koristi.
Drugi pristup reprezentacije orijentiran je na figure. Za svaki tip figure pamti se njena lokacija na ploči što znači da u 64 bita možemo pamtiti lokaciju jedne od 12 vrsta figura - 6 za figure bijelog (kralja, pješake, skakače, lovce, topove, damu(e)), i 6 za figure crnog. Na ovaj način u svakom trenutku lako je dostupna informacija o tome gdje se koja figura nalazi, a stanje na ploči može se reprezentirati 64-bitnim binarnim brojem. Ideja ovakvog pristupa je korištenje jednostavnih binarnih operacija (bitwise shift, xor, maskiranje) kako bi se što brže generirala tablica polja koja npr. napada neka figura. Za figure poput kralja, skakača i pješaka posebno se pamte nizovi sa svim mogućim poljima na koja se mogu pomaknuti za sve moguće položaje. Za ostale figure koje "klize" po ploči nije praktično konstruirati takve nizove, već se ručno unose tablice smjera napada posebno za dijagonalne napade, posebno za horizontalni i vertikalni smjer. Moguće poteze određene figure koja se nalazi na ploči tada se dobiva pomoću nekoliko logičkih operacija.

Korištenjem ovih tablica, zvanih bitboard, moguće je iznimno brzo pronaći sva polja koja određena figura napada, smetaju li joj u nekom smjeru vlastite figure, šahira li određena figura kralja i sl. Gotovo svi bolji šahovski programi, uključujući Stockfish i Houdini, koriste opisane i znatno naprednije tehnike poput magic bitboarda kako bi što brže generirali poteze.

2.2 Generiranje poteza

Prema Larameeu [17] postoje tri glavna pristupa generiranju poteza:

- selektivni - analizom se određuje nekoliko potencijalno dobrih poteza, ostali se odbacuju
- inkrementalni- generira se nekoliko poteza istovremeno uz pretpostavku da će jedan od njih biti dovoljno dobar da neće biti potrebno generirati ostale

[1] U algebarskoj šahovskoj notaciji svako polje na šahovnici ima jedinstvenu oznaku. Stupci se označavaju slovima, a redovi brojevima pa su tako polja za bijele topove označena s a1 (donji lijevi kut) i h1 (donji desni kut),crni se lovci nalaze na poljima c8 i f8, itd. Ova notacija koristi se i za zapis partije.

- potpuni- odjednom se generiraju svi mogući legalni potezi za određenu poziciju

Selektivni pristup nije zaživio jer bi funkcija koja ispravno ocjenjuje je li neki potez dobar u određenoj poziciji bila iznimno zahtjevna za napisati i potrošila bi puno resursa za statičnu procjenu pozicije. U inkrementalnom i potpunom pristupu, fokus je na dobrom poretku čvorova koje želimo pretražiti, jer će bolji poredak omogućiti više rezova kod $\alpha-\beta$ podrezivanja. U većini današnjih šahovskih sustava obično se u svakoj poziciji generiraju svi mogući potezi, a za njihovo rangiranje koriste se pomoćne tablice i heuristike koje su opisane kasnije.

2.3 Evaluacijska funkcija

Heuristička evaluacijska funkcija (obično nazvana samo evaluacijska funkcija) "mozak" je šahovskog programa. Ona procjenjuje različite parametre poput materijalne situacije, mobilnosti, statičkih slabosti i vraća rezultat u obliku numeričke vrijednost iz nekog intervala. Pozitivne vrijednosti pokazuju da je bijeli u prednosti, a negativne da je u prednosti crni. Budući da šah nije rješiv u smislu pretraživanja cijelog stabla, ni evaluacijska funkcija nije garancija da je partija dobivena. Evaluacijska funkcija ključna je za odabir dobre strategije, što je iznimno teško realizirati u programu zato što se razlikuje od ljudskog načina procjene. Uzmimo na primjer središnjicu tipične šahovske partije. Dobar igrač će često imati ideju o tome kako ispravno igrati u određenom tipu pozicije, što je naučio iz vlastitog iskustva, ili iz prisjećanja načina igre jačih igrača u sličnim središnjicama. Što je bolji igrač, bit će veća baza zapamćenih pozicija i ključnih ideja. Problem kod pretvaranja ovog načina razmišljanja u kod je dvojake prirode. Prvo, bilo bi izuzetno teško pretočiti sve ideje jednog velemajstora u računalni kod zbog izuzetno velikog broja ideja, i drugo, ljudi često određene ideje primjenjuju i u "sličnim" pozicijama, a iznimno je teško odrediti što znači da su dvije pozicije slične. Zbog toga su evaluacijske funkcije kod računala fokusirane više na materijal (broj i jačina figura na ploči) i statičke pozicijske elemente, dok se "stvaranje ideja" realizira pretraživanjem.
Promotrimo redom što se sve uzima u obzir kod konstrukcije evaluacijske funkcije:

Materijalno stanje:

Materijalno stanje na ploči dominantan je faktor prilikom računanja vrijednosti evaluacijske funkcije budući da jaki igrači često mogu i jednog pješaka viška pretvoriti u pobjedu. Postoje opće prihvaćene vrijednosti figura u šahu koje se skaliraju pomoću pješaka: skakač ili lovac vrijede 3 pješaka, top - 5 pješaka, dama - 9 pješaka. Ove

vrijednosti modificirane su u šahovskim programima tako da su pomnožene sa 100 kako bi evaluacijska funkcija mogla vratiti vrijednost od npr. 0.5 ako je igrač bolji za "pola" pješaka [2] . Također, empirijski je utvrđeno i opće prihvaćeno da je lovac malo jači od skakača pa vrijednosti figura u računalnom šahu izgledaju ovako: pješak-100, skakač-300, lovac-325, top-500, dama-900, pri čemu 1 označava jednu stotninu pješaka (centipawn).
Materijalno stanje na ploči može se jednostavno izračunati pomoću sljedeće funkcije:

$$M = \sum_i (B_{wi} - B_{bi})V_i$$

gdje $i = 0$ znači pješak, $i = 1$ skakač itd. B_{wi} označava broj bijelih figura tipa i, a B_{bi} označava broj crnih figura tipa i, a V_i je vrijednost figure tipa i.
Ako je $M > 0$, tada je materijalna premoć na strani bijelog. Ako vrijedi $M < 0$ znači da crni ima više materijala, a $M = 0$ znači da je materijal jednak. Ovakvu statičnu evaluaciju uče početnici kako bi više cijenili figure više vrijednosti. U praksi takva jednostavna procjena materijala nije dovoljna jer se mora kombinirati s procjenom pozicijskih vrijednosti opisanih kasnije u ovom poglavlju. Tipičan primjer su gambiti, otvaranja u kojima jedna strana žrtvuje pješaka iz pozicijskih razloga. U ovom kontekstu potrebno je paziti i na knjigu otvaranja jer postoje otvaranja poput kraljevog gambita u kojima bijeli žrtvuje ponekad i dva pješaka kako bi razvio inicijativu i pokušao matirati crnog kralja. Ako računalo ima jednostavnu evaluacijsku funkciju i prerano napusti knjigu otvaranja, naći će se u lošoj poziciji i vrlo vjerojatno izgubiti.

Pozicijski faktori:

Postoji cijeli niz pozicijskih faktora koji su sastavni dio svake dobre evaluacijske funkcije. Neki od njih uključuju:

- razvoj - razvoj lakih figura (lovaca i skakača) je poželjan što prije u partiji - poželjno je što prije rohirati kralja
- prostor - poželjna je kontrola nad određenim dijelom ploče (npr centrom tj. poljima e4, d4, e5 i d5)
- mobilnost - penaliziraju se potezi koji dovode do pasivnih figura koje nemaju najširi mogući spektar kretanja ili se potiču potezi koji figure dovode na mjesta s kojih imaju više utjecaja. Tipični primjeri uključuju topove na otvorenim linijama, lovački par u "otvorenim" pozicijama, penaliziraju se lovci kojima pješaci brane kretanje, itd.

[2] Igrači interpretiraju vrijednosti evaluacijske funkcije otprilike ovako:$[-0.5, 0.5]$ - podjednako, $[-1.5, -0.5]$ i $[0.5, 1.5]$ - bolje za crnog/bijelog, $[-3, -1.5]$ i $[1.5, 3]$ - puno bolje za crnog/bijelog, i $[-1000, -3]$ i $[3, 1000]$ - lagani dobitak za crnog/bijelog

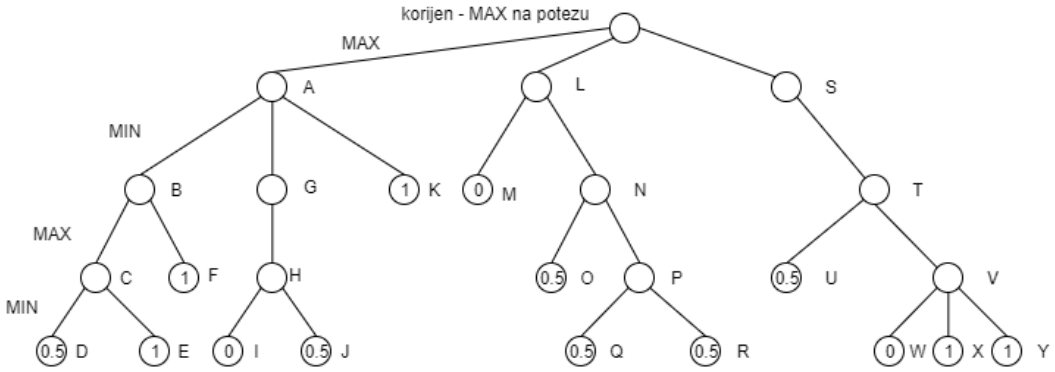
- sigurnost kralja - uključuje brojenje napadača (protivničkih figura u blizini kralja) i obrambenih snaga (vlastitih figura u blizini kralja)
- pješački lanci - penaliziraju se dupli pješaci (pješaci na istoj liniji) , izolirani pješaci (drugi pješak ih ne može braniti)

Gore opisani faktori su samo neki od parametara koji se nalaze u evaluacijskim funkcijama modernih šahovskih programa. Opisani faktori su statički i neovisni o poziciji pa njihovo ispravno namještanje zahtjeva izuzetnu pažnju. Program Stockfish u svojoj evaluacijskoj funkciji koristi stotinjak parametara među kojima su: dinamička vrijednost figura koja je različita u središnjici i završnici (vrijednost kralja u završnici raste), procjena sigurnosti kralja (uključuje brojanje napadača i obrambenih figura), procjena vrijednosti pješačkih lanaca, udvojenih pješaka i uznapredovalih pješaka itd. Procjena uključuje i razne apstraktne koncepte poput zarobljene figure, "visećih" pješaka, te razne kombinacije bonusa i penalizacija za svaku vrstu figura poput bonusa za topove na otvorenim linijama, bonus za skakačka "uporišta" i sl. Vrijednosti svih parametara ove izuzetno kompleksne evaluacijske funkcije "ručno" su namještene, ali je konačna evaluacija linearna kombinacija parametara.
U vlastitom programu evaluacija se radi pomoću tablica polja s podešenim vrijednostima za svaku figuru (piece-square table), procjene zaštite oko kralja, topova na otvorenim linijama i mobilnosti figura, a vrijednosti parametara podešene su ručno. Većina pozicijskih faktora izračunata je pomoću bitboard pristupa. Za svaki potez generiraju se polja koja određena vrsta figura drži pod kontrolom, te se prema tim poljima određuje mobilnost, zaštita oko kralja (pješačka zaštita, blizina drugih figura i sl.).

2.4 Pretraživanje

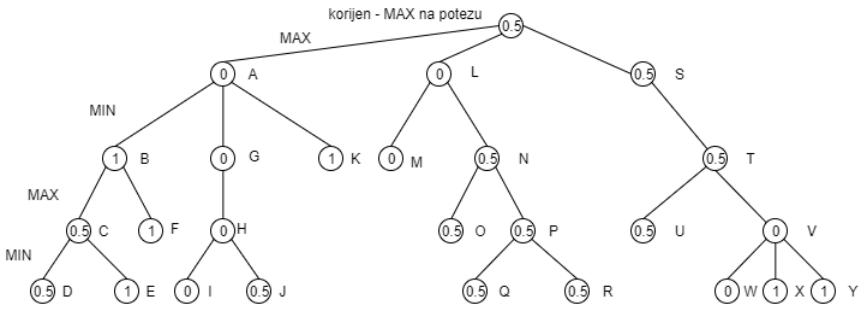
Pretraživanje je izuzetno bitan dio svakog šahovskog programa budući da računalo nadomješta ljudsku intuiciju pretraživanjem svih mogućih pozicija do zadovoljavajuće dubine. Pritom je također potrebno nekako rangirati poteze čemu služe razne heuristike koje ću kasnije opisati. Thompson [27] je utvrdio da šahovski program koji traži jedan potez dublje od protivničkog programa dobiva u otprilike 80% partija za male dubine pretraživanja (3 do 8), dok je za veće dubine zabilježen slabiji utjecaj na rejting. Pojava da se s dubinom pretraživanja smanjuje razlika u snazi programa (diminishing returns), predmet je istraživanja već 30 godina, a najnoviji rezultati uključuju članak Eda Shroedera [24] koji je utvrdio da je za prosječnu dubinu 11 i 12 razlika u snazi otprilike 130 Elo bodova.
Prilikom pretraživanja prirodno se pojavljuje struktura stabla u kojem je korijen trenutna pozicija na ploči, grane su legalni potezi, a čvorovi na dubini d su sve moguće pozicije nastale nakon d odigranih poteza. Budući da je šah simetrična igra, igrač koji je na potezu želi maksimizirati svoj dobitak, a njegov protivnik želi minimizirati dobitak prvog igrača što

rezultira načinom pretraživanja zvanim minimaks. Promotrimo na primjer sljedeće stablo:



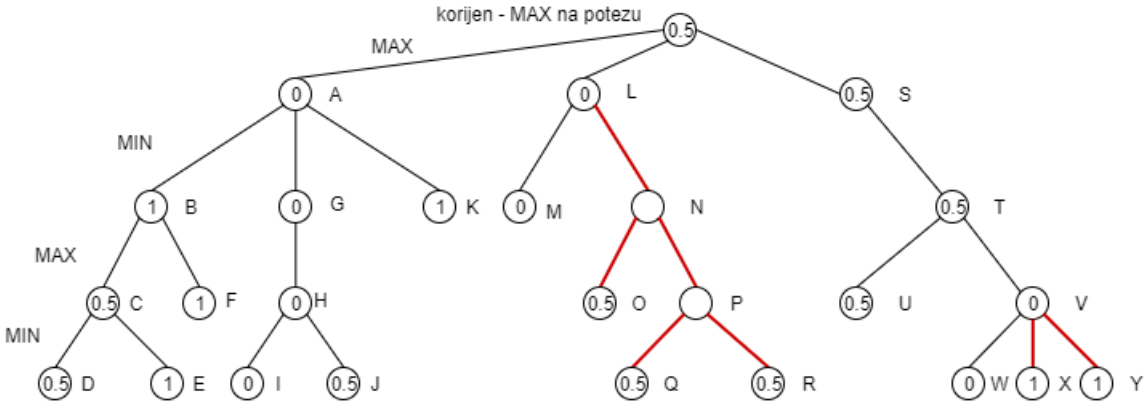
Slika 2.2: Minimaks pretraživanje

U gornjem stablu čvorovi se pretražuju prvo u dubinu slijeva na desno. Parne dubine su označene s MAX (dakle traži se maksimalni rezultat od navedenih) dok su neparne dubine označene s MIN (traži se najmanji rezultat jer bi to protivnik odigrao). Na početku se promatraju čvorovi D i E koji predstavljaju konačna stanja tj. završetak partije. Iz slike se vidi da čvor C mora poprimiti vrijednost čvora D ili E , a budući da se radi o MIN čvoru jasno je da bi protivnik sigurno birao manju vrijednost tj. 0.5. Analogno, prolaskom kroz ostale čvorove, svakom čvoru pridružujemo određenu vrijednost vraćajući se s veće dubine do korijena: Vrijednosti u čvorovima su tada redom:
B=MAX(C,F)=1, H=MIN(I,J)=0, G=MAX(H)=0, A=MIN(B,G,K)=0,
P=MIN(Q,R)=0.5, N=MAX(O,P)=0.5, L=MIN(M,N)=0, V=MIN(W,X,Y)=0,
T=MAX(V,U)=0.5, S=MIN(T)=0.5, korijen=MAX(A,L,S)=0.5
Vizualno to izgleda ovako:



Slika 2.3: Minimaks pretraživanje-rješenje

Već je prije spomenuto da je nemoguće na ovaj način pretraživati od prvog poteza zbog kompleksnosti šaha, pa je početno rješenje uključivalo pretraživanje do određene dubine nakon čega bi se primijenila neka heuristička evaluacijska funkcija. Sljedeće poboljšanje bilo je $\alpha - \beta$ podrezivanje koje funkcionira po sljedećem principu: podrezujemo kad god se za neki čvor ustanovi da potezi ni u kojem slučaju ne mogu biti povoljniji od nekog već istraženog poteza. [5] Ako su to potezi prvog igrača, onda je to α podrezivanje, ako su u pitanju potezi protivnika, onda je to β podrezivanje.
Jednom kad se odredi vrijednost čvora A, počinje pretraživanje podstabla čvora L. Nakon što je pronađena vrijednost čvora M, više nije potrebno pretraživati daljnje grane jer pretraživanjem podstabla iz čvora N neće se dobiti bolji rezultat za MIN.



Slika 2.4: $\alpha - \beta$ podrezivanje

U gore navedenim primjerima poslužili smo se konačnim rezultatima šahovske partije, ali gotovo uvijek su vrijednosti čvorova izračunate pomoću evaluacijske funkcije. U tipičnoj šahovskoj središnjici minimaks s $\alpha - \beta$ podrezivanjem pretražuje otprilike dva puta korijen broja čvorova standardnog minimaks algoritma. Zbog simetričnosti igre šaha (koliko je prvi igrač bolji, toliko je drugi igrač lošiji), u programu je implementirana drugačija vrsta minimaks pretraživanja, a to je negamaks. Prednost negamaksa je da se može implementirati rekturzivno unutar jedne funkcije koristeći činjenicu da je $max(a, b) = -min(-a, -b)$ za vrijednost poteza igrača.
Alfa-beta podrezivanje ovisi o redoslijedu poteza koji se pretražuju, budući da će se puno više grana podrezati ako su čvorovi sortirani. Knuth i Moore [10] pokazali su da je minimalni broj čvorova potrebnih da bi se potvrdila minimaks vrijednost stabla iznosi $w^{\lfloor \frac{d}{2} \rfloor} + w^{\lceil \frac{d}{2} \rceil} - 1$ gdje je w faktor grananja, a d dubina stabla. Dakle, dobro ugođen minimaks s $\alpha - \beta$ podrezivanjem u isto vrijeme može pretražiti dvostruko dublje od minimaks pretraživanja (složenost w^d) uz preduvjet da dobro sortira poteze.

Tablica 2.1: Usporedba broja pretraženih čvorova za različite inačice algoritma

Dubina	Negamaks	Negamaks s $\alpha - \beta$ podrezivanjem	Negamaks s $\alpha - \beta$ podrezivanjem i sortiranjem
2	421	197	60
3	9323	2500	595
4	178220	26789	2758
5	2735952	205215	26545
6	34245254	1937524	92121
7	Premalo memorije	8067433	762507

Slijedi opis nekih tehnika za ubrzavanje pretrage.

1. Transpozicijska tablica
   U tablici se pamte rezultati prošlih pretraživanja kako bi se uštedjelo vrijeme. Ako je neka varijanta već bila istražena, tada se umjesto ponovnog pretraživanja može upotrijebiti samo ocjena varijante i potez koji je zapisan u tablici. U nekim slučajevima moguće je da je ta vrijednost ujedno i najbolji potez što će rezultirati podrezivanjem svih ostalih grana prilikom pretraživanja. Svi rezultati istraživanja potvrđuju da je ova tehnika ključna za smanjivanje broja pretraženih čvorova. Lazar [18] je za 6 poteza duboko stablo ustvrdio da se korištenjem transpozicijske tablice pretražuje samo 5% čvorova koji bi se pretražili bez nje. Strejczek [26] je za razne varijacije dubine dobio da se korištenjem transpozicijske tablice za 6 do 9 poteza, ovisno o poziciji, broj pretraženih čvorova smanjuje za 52%-95%. Transpozicijska tablica u najjačim programima napravljena je kao konačna hash tablica u koju se vrijednosti unose prema određenom ključu koji se računa iz pozicije. U vlastitom programu transpozicijska tablica smanjila je broj čvorova za otprilike 30% što je ipak manje od ranije zabilježenih rezultata, no moguće je da je tome razlog sortiranje čvorova i maksimalno podrezivanje stabla pri čemu se gube čvorovi koji bi mogli biti spremljeni u tablici.

2. Iterativno produbljivanje
   Budući da pretraživanje do određene dubine D traje, ideja je da se pretražuje do manje dubine d i onda se ta pretraga produbi za jedan potez pa za dva i tako dalje dok se ne dođe do dubine D. Prednost ovog pristupa je u tome da ako pretraživanje nije gotovo, program ne vraća neki nasumični potez već najbolji koji je dobio u zadnjoj iteraciji. Također se polazi od pretpostavke da će u većini pozicija najbolji potez na dubini d ujedno biti i najbolji potez na dubini $d + 1$. Nedostatak koji se često ističe je da se ista pozicija pretražuje mnogo puta, no ono što oduzima najviše vremena u pretraživanju je upravo zadnja dubina do koje dolazi. Ako je $N_{d+1} = 35^{d+1}$

gdje je N broj pretraženih čvorova, 35 je broj legalnih poteza, a d je dubina stabla pretraživanja i

$$N_{1\ldots d} = \sum_{i=0}^{d} 35^i = \frac{35^{d+1}}{34}$$

za $d > 2$ tada je omjer

$$\frac{N_{1\ldots d}}{N_{d+1}} = \frac{1}{34} \approx 3\%$$

što znači da će sva pretraživanja do dubine d činiti samo 3 % pretraženih čvorova za dubinu $d + 1$. Iterativno produbljivanje često se koristi s kasnije opisanim MTD(f) pretraživanjem.

3. Ubojiti potez (killer move)
   Ponekad je neki potez toliko dobar da se zbog njega ne pretražuje većina ostalih čvorova. Ideja heuristike ubojitog poteza je da se taj potez pamti jer bi mogao biti dobar i u idućem pretraživanju. Prema Strejczekovim rezultatima, ubojiti potez smanjuje broj pretraženih čvorova od 16% do 85% ovisno o poziciji. U vlastitom programu nisu dobiveni toliko dobri rezultati budući da se sprema samo jedan potez (u jačim programima na svakoj dubini spremaju se tipično dva ubojita poteza zbog povećanja učinkovitosti).

4. Pretraživanje minimalnog okvira
   Ideja ove heuristike je smanjivanje okvira $[\alpha,\beta]$ do minimuma $\alpha = \beta - 1$. Ovaj pristup čini se besmislenim jer će sigurno vratiti neki od rubova, ali će se upravo rubna vrijednost koristiti za podrezivanje velikog broja čvorova. Podvrste ovog pristupa uključuju 2 popularna algoritma: Pretraživanje principijelne varijante (PVS) i MTD(f).
   Pretraživanje principijelne varijante koristi činjenicu da, ako imamo dobro početno rangiranje poteza i prvo smo pretražili najbolju varijantu, ostale varijante nije potrebno pretražiti u dubinu, već sa smanjenim okvirom. Nije potrebno znati njihovu pravu vrijednost, samo da neće biti bolji od prve pretražene varijante. Principijelna varijanta pretražuje se standardnim $\alpha - \beta$ algoritmom, a za ostale koristimo minimalni okvir. Ako je rangiranje poteza dobro, i pogođena je najbolja varijanta, ovaj će algoritam znatno brže vratiti vrijednost pozicije, a ako je početna varijanta bila suboptimalna, onda je vraćena beta vrijednost i cijelo stablo se pretražuje s proširenim okvirom. Ovaj algoritam često se izjednačava sa negamaks pandanom zvanim NegaScout.
   Algoritam MTD(f) [20] također koristi minimalni okvir kako bi našao pravu vrijednost. Na početku se, koristeći prošle iteracije, postavlja početna vrijednost i poziva $\alpha - \beta$ minimaks s minimalnim okvirom ($\alpha = \beta - 1$). Svakim pozivanjem MTD(f),

vraća se jedan od rubova koji se dalje koristi kao nova vrijednost za ponovno postavljanje okvira. MTD(f) se tipično koristi u kombinaciji sa spremanjem vrijednosti već pretraženih čvorova kako bi se izbjeglo višestruko pretraživanje istog dijela stabla. Uobičajeno je koristiti ovaj pristup u iterativnom produbljivanju koje garantira konvergenciju prema stvarnoj minimaks vrijednosti. Zbog izuzetne brzine MTD(f) je, uz Negascout, najrašireniji algoritam za pretraživanje u šahovskim programima.

5. Tiho pretraživanje (quiescence search)
   Evaluacijska funkcija u obzir uzima statičke elemente pozicije što zahtjeva statičnu poziciju za analizu. No što ako na kraju pretraživanja dolazi do pozicije u kojoj je crni žrtvovao topa, ali će u drugom potezu uzeti bijelu damu? Sama evaluacijska funkcija tu će poziciju ocijeniti kao bolju za bijelog (ima topa više) i dodijeliti joj najveću vrijednost jer ne "vidi" potez dalje. Zbog problema horizonta nije dovoljno samo pretraživati stablo do određene dubine, već je potrebno u završnim čvorovima razmotriti još koji potez dublje ako ima forsiranih poteza (šah, uzimanje figure i sl.). Ideja tihog pretraživanja je u svim čvorovima doći do pozicije u kojoj nema forsiranih poteza poput uzimanja figure i onda primijeniti evaluacijsku funkciju na tu "tihu" poziciju. Budući da se tiho pretraživanje poziva na listovima (leaf nodes), potrebno je nekako ograničiti eksponencijalni rast broja čvorova koje će ona pretražiti.

Osim navedenih poboljšanja, najbolji šahovski programi koriste još mnoge druge poput null move heuristike, redukcija i ekstenzija (dublje pretraživanje varijanti kad je npr. kralj u šahu), napadačkih i obrambenih obrazaca, slabih i jakih polja i sl. Svaka od opisanih tehnika pridodaje od 10 do 100 Elo bodova šahovskom sustavu, ali zahtjeva mnogo vremena za implementaciju.

2.5 Vlastiti program

U sklopu istraživanja načina rada modernih šahovskih sustava implementirane su neke od gore spomenutih tehnika i napravljena je vlastita inačica programa za igranje šaha. Fokus prilikom istraživanja i implementacije bio je na heuristikama koje poboljšavaju evaluaciju i na različitim metodama pretraživanja stabla.

Reprezentacija ploče, poteza i pravila

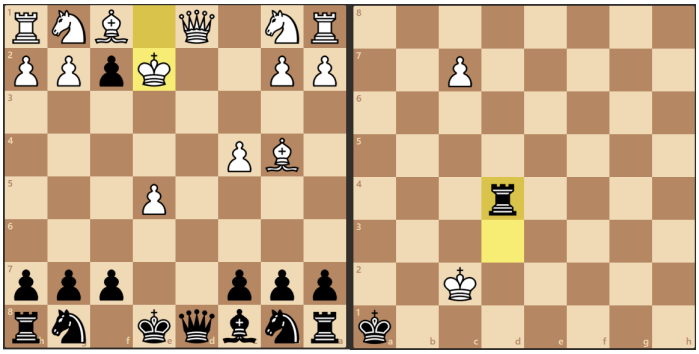
Program je napisan u programskom jeziku Python, a implementaciju ploče i poteza preuzeta je od programa Sunfish [4]. Šahovnica je implementirana kao polje od 120 memorijskih lokacija u obliku 12x10 ploče, kao na donjoj slici.

	0	1	2	3	4	5	6	7	8	9
	10	11	12	13	14	15	16	17	18	19
8	20	21	22	23	24	25	26	27	28	29
7	30	31	32	33	34	35	36	37	38	39
6	40	41	42	43	44	45	46	47	48	49
5	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69
3	70	71	72	73	74	75	76	77	78	79
2	80	81	82	83	84	85	86	87	88	89
1	90	91	92	93	94	95	96	97	98	99
	100	101	102	103	104	105	106	107	108	109
	110	111	112	113	114	115	116	117	118	119
		a	b	c	d	e	f	g	h	

Slika 2.5: Reprezentacija ploče u vlastitom programu

Ovakva implementacija omogućava jednostavnu provjeru legalnosti poteza (ako je zadnja znamenka jednaka 0 ili 9, ili je broj manji od 20, ili je broj veći od 99, polje nije na ploči). Zatim je definiran smjer kretanja na sljedeći način: kretanje u smjeru ”sjevera” (prema gore) jednako je promjeni polja za -10, analogno za jug +10, kretanje na istok je pomicanje za jedno polje desno pa se lokacija mijenja za 1, a lijevo za -1. Nakon toga definirani su svi mogući smjerovi kretanja svih figura, npr. za damu su smjerovi N,S,E,W,N+E,N+W,S+E,S+W. Za razlikovanje boje figura koriste se velika i mala slova pa su velikim slovima označene vlastite figure. Pravo na rohadu i en passant provjeravaju se prilikom generiranja poteza, isto kao i promocija pješaka. Implementiran je i bitboard pristup kao ”rječnik” vrsta figura gdje je za svaku figuru spremljena lista od 120 polja ploče pri čemu su s 1 označena polja koja ta figura brani/napada, a polja na kojima se nalaze figure označena su velikim tiskanim slovima. Za svaku poziciju bitboard se generira iznova jer je trenutno lakše ”pročitati” stanje na ploči nego ažurirati postojeći bitboard nakon odigranog poteza. Ovaj pristup može se optimizirati ako se implementira funkcija vraćanja poteza (takeback move).
Osim generiranja same ploče, potrebno je napraviti objekt koji će reprezentirati poziciju

sa svim njenim karakteristikama. U vlastitom programu pozicija se sastoji od sljedećih parametara: ploča, vrijednost (rezultat evaluacijske funkcije za tu poziciju, početno postavljen na 0), zastavice s pravima na rohadu za crnog i bijelog, zastavica za en passant te redni broj poteza (koristan kod određivanja strane koja je na potezu). Unutar te klase definirane su razne funkcije poput generatora poteza, rotiranja ploče, "praznog" poteza (null move) i odigravanja poteza. Generator poteza podrazumijeva kretanje niz svaki prethodno definirani smjer kretanja (N,W,E,S) za svaku figuru tako dugo dok je figura unutar granica ploče, ne preskače drugu figuru (osim skakača) i ne okupira njeno mjesto (ako nije u pitanju uzimanje). Uz to je još potrebno provjeravati pravo na rohadu, en passant pravilo i promociju. U programu Sunfish definirano je da se pješak na osmom redu automatski promovira u damu, što je popravljeno, budući da u nekim (izuzetno rijetkim) pozicijama promoviranje u damu nije najbolji potez.



Slika 2.6: Primjeri korisnosti promocije u figuru različitu od dame

Navedene pozicije su poznata zamka u otvaranju u kojoj je najbolji potez za crnoga uzimanje pješakom skakača i pretvaranje u skakača (fxg1=N+) i kraj poznate studije gdje bijeli dobiva potezom promocijom u topa (c8=R) jer promocija u damu samo remizira. Umjesto automatske pretvorbe, program pokreće 4 pretrage dubine 3 s različitim figurama u koje se pješak može promovirati (lovac, skakač, top i dama) i vraća figuru za koji je traženje dalo najbolji rezultat. Program je testiran na gore prikazanim pozicijama i u obje je pronašao najbolji potez. Ovakva implementacija ipak nije praktična jer će u velikoj većini slučajeva prilikom promocije dama predstavljati najbolje rješenje. Moguća izmjena je pretraživanje promocije u damu 3 poteza u dubinu i određivanje njene vrijednosti. Ako vrijednost tog poteza bude manja od neke fiksne vrijednosti (npr. 1000 - vrijednost jedne dame), tada se pretražuje stablo s promocijama u druge figure. Prema službenim pravilima FIDE [11] nakon trećeg pojavljivanja jednake pozicije na ploči partija se proglašava remijem, što je također implementirano u programu. Svaki odigrani potez sprema se u rječnik kao ključ, a broj ponavljanja kao vrijednost.

Evaluacijska funkcija

Evaluacijska funkcija u obzir uzima materijal prema sljedećoj skali: pješak - 100, skakač - 280, lovac - 320, top - 480, dama - 930, kralj - 60000. Za svaku vrstu figura definirane su vrijednosne tablice (piece-square table) s nagradom/penalizacijom za svako polje na ploči. Od ostalih faktora, implementirane su funkcije koje računaju zaštitu oko kralja, bonusi za topove na otvorenoj liniji i bonusi za mobilnost. Konačna evaluacijska funkcija je linearna kombinacija svih parametara sa težinskim faktorima određenim iz iskustva. Dobivena evaluacijska funkcija izgleda ovako:

```
def heur_eval(pos,movei):
    our_bitboard=init_bitboard
    opponent_bitboard=init_bitboard
    our_bitboard=update_bitboard(our_bitboard,pom_pos)
    rooks=rook_on_opfile(our_bitboard)
    opponent_bitboard=update_bitboard(opponent_bitboard,pos.rotate())
    defense_score=defense(our_bitboard,opponent_bitboard)
    mobil=mobility(pos,movei)
    return pos.score*0.9+defense_score*0.2+mobil+rooks
```

Slika 2.7: Evaluacijska funkcija

Sve navedene funkcije koriste bitboard pristup. Veći broj parametara evaluacije bio bi poželjan, ali bi usporio program. Interesantan način zaobilaska ovog problema je korištenje ”lijene” evaluacije (lazy evaluation) kod koje se vrijednost poteza do zadnje pretražene dubine računa pomoću jednostavne evaluacije (npr. samo materijal) dok se vrijednost listova računa pomoću proširene evaluacije.

Pretraživanje

Za pretraživanje se mogu koristiti dva pristupa: negamaks pretraživanje s $\alpha - \beta$ podrezivanjem ili MTD pretraživanje s inkrementalnim produbljivanjem koje unutar sebe poziva negamaks. Budući da MTD konvergira brže prema pravoj vrijednosti minimaksa ako je početna vrijednost bila blizu stvarnoj, konačni rezultati za zadnje dvije pretražene dubine se pamte u posebnoj varijabli. Potrebno je pamtiti dvije vrijednosti jer se za veće dubine vrijednost na neparnoj dubini, kad je zadnji potez igračev, i parnoj dubini, kad je zadnji potez protivnikov, znatno razlikuju (u većini slučajeva otprilike za vrijednost jednog pješaka). Vrijeme predviđeno za pretraživanje svakog poteza je 10 sekundi, a u tom vremenu program tipično pretraži do dubine 7. Unutar pretraživača potezi se rangiraju prema vrijednosti evaluacijske funkcije što osigurava optimalan redoslijed pretraživanja i najveći broj podrezivanja.

```
def negamax(pos,alpha,beta,depth,root):
    if depth==0:
        quiescence(pos,move)
    moves=sorted(pos.gen_moves(),key=pos.eval,reverse=True)
    best_value=-INFINITY
    for move in moves:
        value=-negamax(pos.move(move),-beta,-alpha,depth-1,False)
        best_value=max(value,best_value)
        alpha=max(alpha,value)
        if alpha>=beta:
            break
    return best_value
```

Slika 2.8: Negamaks algoritam s $\alpha - \beta$ podrezivanjem

```
def mtd(self, pos):
       for depth in range(1, 1000):
           lower, upper = -MATE_UPPER, MATE_UPPER
           while lower < upper:
               beta=max(g,lower+1)
               g = self.negamax_with_memory(pos, beta-1,beta, depth)
               if g<beta:
                   upper = g
               if g>=beta:
                   lower = g
           g = self.negamax_with_memory(pos, lower-1,lower,depth)
```

Slika 2.9: MTD algoritam s produbljivanjem

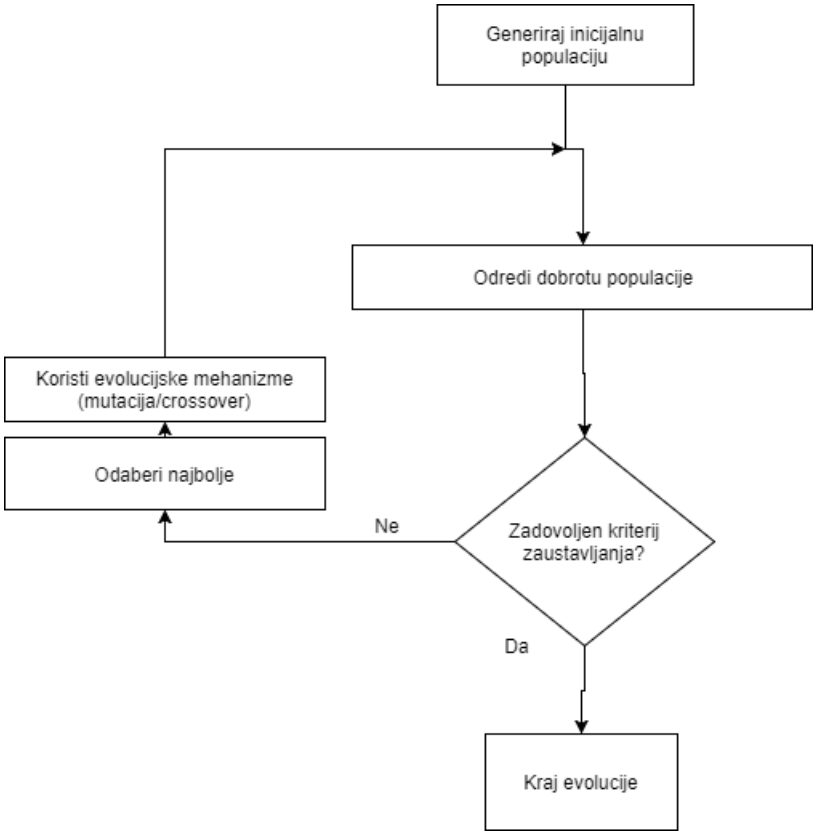
Najbolji potez čita se iz transpozicijske tablice na kraju pretrage. Implementirane su dvije transpozicijske tablice, jedna koja čuva potez, a druga koja čuva rezultat. Umjesto hash tablice korišten je poseban tip podataka sličan rječniku, pri čemu je ključ za spremanje u slučaju poteza samo pozicija (string koji reprezentira ploču), a u slučaju vrijednosti poteza ključ je uređena trojka (pozicija, dubina, korijen). Za pretragu listova koristi se tiho pretraživanje koje uzima u obzir sve moguće izmjene figura. Zbog eksponencijalnog rasta broja čvorova koje je potrebno provjeriti za tiho pretraživanje, ono se provodi samo na principijelnoj varijanti (prvoj koja se pretražuje). Osim ovih heuristika koristi se i ubojiti potez koji se također uzima iz transpozicijske tablice. Implementirana je i pretraga po vjerojatnosti opisana u idućem poglavlju.
Interakcija s korisnikom preuzeta je od programa Sunfish i djelomično modificirana za podržavanje dodatnih opcija poput spremanja bitboardova i provjeravanja remija. Dobiveni šahovski program još uvijek ne odgovara svim specifikacijama potrebnim za službena natjecanja i ostvarivanje Elo rejtinga pa još uvijek nema službeno određenu snagu, ali procjenjujem da je u rangu jačeg amaterskog igrača.

Poglavlje 3

Automatizacija u modernim sustavima za igranje šaha

3.1 Automatizirano podešavanje parametara evaluacijske funkcije

Većina današnjih šahovskih programa koristi se nekom vrstom minimaks pretraživanja s naglaskom na brzom podrezivanju čvorova i evaluacijskoj funkciji koja što bolje ocjenjuje stanje na ploči u završnim čvorovima. Budući da su najbolji algoritmi za pretraživanje poznati već dvadesetak godina, većina programera šahovskih programa fokusira se na parametre evaluacijske funkcije i podešavanje njihovih težina što je izuzetno zahtjevan posao. Prema Schaefferu [14], izbor komponenata i pripadnih težina u evaluacijskoj funkciji najteži je i vremenski najzahtjevniji posao prilikom konstrukcije dobrog šahovskog programa. Zbog toga su se neki programeri odlučili pokušati automatizirati proces traženja optimalnih parametara evaluacijske funkcije. Jedan od takvih pokušaja bilo je korištenje genetskih algoritama za računanje vrijednosti parametara. Genetski algoritmi nazvani su tako jer simuliraju evoluciju vrsta koristeći se evolucijskim pravilom o preživljavanju najjačih. Svi genetski algoritmi koriste se istom shemom:



Slika 3.1: Shema genetskog algoritma

Parametri evaluacijske funkcije mogu se reprezentirati jednim nizom bitova i na početku evolucije inicijalizira se mnogo takvih nizova s nasumičnim vrijednostima. Tada sve jedinke međusobno igraju partiju šaha i pobjednici se uzimaju u sljedeću generaciju (uz obavezne faktore poput mutacije i crossovera), i tada ponovno započinje proces traženja najboljeg pojedinca u generaciji. David, van den Herik i Koppel [8] dokazali su da je ugađanje parametara na ovaj način pojačalo snagu računalnog programa, a evaluacijske funkcije dobivene na ovaj način ponekad su odskakale od ”propisanih” vrijednosti (npr. pješak=100, skakač=520, lovac=570, top=824, dama=1710). Najbolja evaluacijska funkcija ovih autora vrednovala je damu kao 17 pješaka tj. umanjila je vrijednost pješaka na pola njegove ”standardne” vrijednosti, dok su relativni omjeri ostalih figura ostali približno jednaki standardnim. Unatoč interesantnim rezultatima, pokazalo se da taj pristup zahtjeva previše vremena za veći broj parametara, te da su vrijednosti parametara bile lošije od pažljivo pogođenih ručno unesenih vrijednosti.

Sljedeća metoda korekcije težine parametara koristi se u programu Stockfish i nazvana je Stockfish tuning method [15]. Implementacija ovog načina prilagođavanja parametara nije matematički korektna, ali digla je Elo rejting programa Stockfish za 50-ak bodova. Na početku procesa pretpostavi se da su vrijednosti varijabli koje se žele korigirati otprilike točne do na neki mali faktor δ (odabran na temelju iskustva). Različite inačice programa tada igraju partije, pri čemu se vrijednosti varijabli razlikuju za δ od "stare" vrijednosti (kod jedne verzije za $+\delta$, kod druge za $-\delta$). Nakon odigranog meča korigiramo vrijednost varijable ovisno o pobjedniku (i uz neki korekcijski faktor, kod Stockfisha to je 0.0002) i ponovimo proces. Tvorci Stockfisha koristili su varijabilan δ, istovremeno korigiranje više varijabli i $30000 - 100000$ superbrzih partija kako bi korigirali vrijednosti figura. Problem ovog pristupa je u tome što ne konvergira, već ga je potrebno zaustaviti u "pogodnom" momentu.

Treći pristup korigiranja težinskih parametara evaluacijske funkcije koristio se vrstom podržanog učenja, konkretnije TD (temporal difference) učenjem. Neka je S skup svih mogućih pozicija u igri šaha, i igru čini niz poteza u nekim trenutcima u vremenu $t = 1, 2, \ldots, n$. U trenu t igrač ima poziciju $x_t \in S$, i na raspolaganju poteze A_{x_t} (skup svih legalnih poteza u poziciji x_t). Odabravši potez $a \in A_{x_t}$ i odigravši ga, dovodi se u stanje x_{t+1} s vjerojatnosti $p(x_t, x_{t+1}, a)$. Na kraju partije igrač dobiva nagradu (1 za pobjedu, 0 za remi i -1 za poraz). Pretpostavimo da je partija trajala N poteza i neka $r(x_N)$ označava nagradu. Ako pretpostavimo da igrač koristi određenu strategiju opisanu funkcijom $a(x)$ za igru u trenutnom stanju x, tada je njegova očekivana nagrada iz svakog stanja $x \in S$ dana formulom:

$$J^*(x) := E_{x_N|x} r(x_N)$$

, pri čemu je očekivanje vezano uz vjerojatnosti $p(x_t, x_{t+1}, a(x_t))$. Za vrlo velike prostore S nije moguće izračunati $J^*(x)$ za svaki x, pa funkciju aproksimiramo koristeći parametarsku funkciju $\tilde{J} : S \times \mathbb{R}^k \to \mathbb{R}$, pri čemu pretpostavljamo da $\tilde{J}$ ima parcijalnu derivaciju po parametrima $w = (w_1, w_2, ..., w_n)$. Cilj je pronaći vektor w koji minimizira grešku između aproksimacije $\tilde{J}$ i J^*. Neka je $x_1, ..., x_N$ niz stanja u partiji. Za dani vektor w definiramo temporal difference

$$d_t := \tilde{J}(x_{t+1}, w) - \tilde{J}(x_t, w)$$

. Temporal difference d_t mjeri razliku između predviđenog rezultata funkcije J' u trenutcima $t + 1$ i t. Za J^* vrijedi sljedeće svojstvo:

$$E_{x_{t+1}|x_t}[J^*(x_{t+1}) - J^*(x_t)] = 0$$

, pa bi za dobru aproksimaciju $\tilde{J}$ trebalo vrijediti da je $E_{x_{t+1}|x_t} d_t$ blizu 0. Zbog lakše notacije pisat ćemo $J'(x_N, w) = r(x_N)$, pa je zadnja temporal difference jednaka $d_{N-1} = \tilde{J}(x_N, w) -$

$\tilde{J}(x_{N-1}, w) = r(x_N) - \tilde{J}(x_{N-1}, w)$. Evaluacija se radi na listovima stabla pretraživanja, i u skladu s tim program podešava vektor w prema formuli:

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) \Big[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \Big]$$

, pri čemu je $\nabla \tilde{J}(\cdot, w)$ vektor parcijalnih derivacija funkcije J', α je parametar koji kontrolira učenje, a $\lambda \in [0, 1]$ označava parametar koji određuje doseg gledanja unatrag. Za $\lambda = 0$, vektor parametara se prilagođava tako da funkcija $\tilde{J}(x_t, w)$ bude što bliže funkciji $\tilde{J}(x_{t+1}, w)$, dok za $\lambda = 1$, vektor parametara namješta se tako da funkcija $\tilde{J}(x_t, w)$ bude blizu konačnom stanju. Budući da gotovo svi šahovski programi koriste neku vrstu minimaks pretraživanja s heuristikama za ubrzavanje, a za procjenu heurističku evaluacijsku funkciju koja je većinom linearna (dakle pogodna za parcijalne derivacije), ideja ovog pristupa je primijeniti gornju formulu na šah koristeći sljedeće: $\tilde{J}_d(x, w)$ označava evaluacijsku funkciju za stanje x dobivenu primjenom $\tilde{J}(\cdot, w)$ na listove minimaks stabla na dubini d. Ovaj postupak nazvan je TDLeaf(λ) i preuzet je od Baxtera, Tridgella i Weavera [6] koji su, koristeći ovaj pristup s parametrima $\lambda = 0.7$ i $\alpha = 1$, uspješno pokazali primjenu TD(λ) algoritma u šahovskom sustavu KnightCap. Dakle, u konkretnoj partiji, u određenom stanju x minimaks pretraživanjem pretražilo bi se stablo do dubine d. Tada bi promatranjem promjene evaluacije kroz d poteza izračunali pogrešku za početnu poziciju koristeći se pritom različitim težinama (pogreška na dubini jedan značajnija je od pogreške na dubini 6). Pogreška na dubini d općenito se množi s težinskim faktorom λ^d. Težinski vektor programa KnightCap uključivao je 1468 komponenti od kojih su sve osim materijalne vrijednosti figura bile inicijalizirane na 0 i podešavane strojno. Ova implementacija pokazala se izuzetno uspješnom i KnightCap je nakon 1000 odigranih partija na internetskom serveru postigao jačinu internacionalnog majstora (rejting 2400). Jedini problem predstavljalo je mnoštvo parametara koji su usporavali evaluacijsku funkciju i time smanjivali dubinu do koje je KnightCap mogao pretraživati. Ova tehnika učenja pojačavanjem ostala je popularna i uspješno je primijenjena u šahovskim sustavima opisanim u idućem poglavlju.

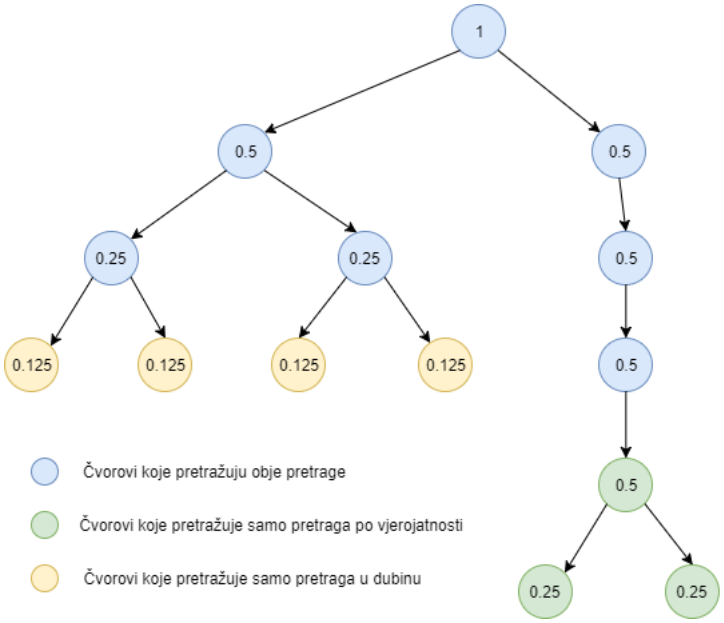
3.2 Strojno učenje i njegove primjene u računalnom šahu

U ovom poglavlju ukratko se opisuju najnoviji programi koji se, koristeći strojno učenje, polako približavaju vodećim šahovskim programima. Oba opisana programa rade najbolje koristeći više GPU-ova i više TPU-ova i paraleliziranim bibliotekama pa nije bilo moguće istražiti njihovo ponašanje. Također, trenutno još nisu poznati detalji programa Alphazero. Prvi opisani program je Giraffe, autora Matthewa Laia [16], koji se isključivo oslanja na TD učenje i igru protiv samog sebe. U zadnjem paragrafu ukratko je opisan početak novog

poglavlja u računalnom šahu koje je započelo u prosincu mečom AlphaZero protiv Stockfisha. U tom meču AlphaZero je uvjerljivo dobio najjači šahovski program na svijetu, Stockfish , nakon samo 4 sata učenja igranjem protiv samog sebe.

Šahovski program Giraffe

Prvi sustav otvorenog koda koji je u potpunosti odbacio klasičnu pretragu i fokusirao se isključivo na primjenu strojnog učenja za igranje šaha bio je Giraffe, konstruiran od strane Matthewa Laia 2015. godine. Prilikom konstrukcije evaluacijske funkcije koja će ispravno procijeniti vjerojatnost dobitka igrača, bilo je potrebno konstruirati dobru bazu značajki koja će predstavljati poziciju (mobilnost, napadački i obrambeni obrasci, materijal i sl.). Za identifikaciju tih značajki trenirane su neuronske mreže kako bi što bolje predvidjele Stockfishov sljedeći potez u 5 milijuna partija, pri čemu je veća preciznost ”nagrađivana”. Na kraju ovog procesa dobivene su 363 bitne značajke koje omogućavaju neuronskoj mreži da nauči koncepte poput centralizacije skakača i lovaca, okupiranje centra, jačine topa na sedmom redu i sl. Koristeći se gore opisanim TD postupkom i metodama za gradient descent, neuronska mreža korištena za evaluaciju u 72 sata treninga na 2x10 jezgrenom Intel Xeon procesoru, dostigla je razinu internacionalnog majstora i po pozicijskim karakteristikama dominira nad mnogo jačim šahovskim programima. Osim ovog novog pristupa evaluacijskoj funkciji, Giraffe koristi i drugačiju vrstu pretraživanja - probabilistička stabla. Umjesto standardnog pretraživanja do određene dubine, kao parametar za prekid pretraživanja koristi se vjerojatnost tog čvora da bude dio principijelne varijante. Prilikom pretrage potomaka određenog čvora mora vrijediti da je zbroj svih vjerojatnosti potomaka jednak vjerojatnosti roditelja, a na početku se ne preferira ni jedan čvor (tj. svi imaju jednaku vjerojatnost). U većini slučajeva pretraga po vjerojatnosti pretražit će skoro sve iste čvorove kao i standardni minimaks, ali u varijantama kod kojih je dio stabla reduciran pretraživanje po vjerojatnosti tražiti će dublje od standardnog minimaks pretraživanja.



Slika 3.2: Usporedba algoritama pretraživanja

Ovaj pristup mnogo je sličniji pretraživanju ljudskog igrača koji forsirane varijante uvijek gleda dublje (ako je moguće do trenutka kad više nema forsiranih poteza), dok se varijante s većim faktorom grananja razmatra do manje dubine. Rezultati Laia [16] pokazuju da je razlika vidljiva (26 ± 12 rejting bodova) i da probabilističko stablo predstavlja prirodniju alternativu pretraživanju u dubinu. Implementacija takvog stabla ne zahtjeva veliku modifikaciju postojećeg koda, dovoljno je postaviti konstantu za dubinu pretrage po vjerojatnosti i umjesto parametra dubine proslijediti vjerojatnost nekog čvora koja se određuje brojačem mogućih poteza u svakoj poziciji. Ovakvo pretraživanje izuzetno je korisno u završnici, kad je broj figura reduciran, a broj mogućih poteza smanjen, ili prilikom forsiranog matiranja (ili napada na kralja), kad većina poteza podrazumijeva šahiranje. Kombinacija "normalnog" pretraživanja i pretraživanja po vjerojatnosti također predstavlja interesantan pristup. Ako se u nekoj grani stabla pojavi "forsirana" varijanta (npr. do 5 mogućih poteza), smisleno je dalje prijeći na pretraživanje po vjerojatnosti budući da je vjerojatno u pitanju neka izmjena ili šahiranje kralja pa je smisleno pokušati pretražiti što dublje.
Za verziju vlastitog programa koja koristi pretraživanje po vjerojatnosti, početni čvor (korijen) imao je vjerojatnost 1, a na svakoj dubini vjerojatnost posjećivanja svakog čvora računala se kao omjer vjerojatnosti roditelja i broja čvorova na toj dubini. Kriterij za zaustavljanje bio je broj 10^{-8} koji je određen eksperimentalno (za veće veličine pretraživanje traje predugo, za manje nije dovoljno precizno). U sljedećoj tablici uspoređeno je vrijeme i brzina pretraživanja za sve tri inačice šahovskog programa.

Tablica 3.1: Usporedba nekih kriterija različitih verzija šahovskog programa

		Prosječna vrijednost
Pretraživanje po vjerojatnosti	Broj pretraženih čvorova (n)	1583180
	Vrijeme (s)	30,97
	Brzina (n/s)	49550,69
Negamaks pretraživanje dubina 6	Broj pretraženih čvorova (n)	743885
	Vrijeme (s)	13,47
	Brzina (n/s)	53160,3
Negamaks pretraživanje dubina 7	Broj pretraženih čvorova (n)	3198683
	Vrijeme (s)	56,86
	Brzina (n/s)	53581,13

Iz tablice se vidi da je brzina pretraživanja negamaks inačica programa ipak malo brža od pretraživanja po vjerojatnosti. Prosječni broj pretraženih čvorova najvažniji je parametar jer govori koliko je duboko program pretražio stablo. Prema tom kriteriju, negamaks dubine 7 trebao bi biti najjači program od ove tri verzije zato jer pretražuje puno više čvorova. Prilikom testiranja verzije s vjerojatnosnim stablom, odigrana su dva meča od 6 partija protiv obje negamaks verzije. Rezultati mečeva zapisani su u sljedećoj tablici:

Tablica 3.2: Rezultati mečeva različitih inačica programa

	Verzija pretraživanja po vjerojatnosti
Negamaks verzija, dubina=6	2-4
Negamaks verzija, dubina=7	5-1

Jasno je da je od navedene tri inačice negamaks pretraživanje do dubine 7 uvjerljivo najjači program, budući da pretražuje puno više čvorova, ali mu je zato potrebno i najviše vremena. Pretraživanje po vjerojatnosti čini se boljim načinom pretraživanja, no teško je ispravno podesiti prag tako da ono bude zaista usporedivo s negamaks pretraživanjem do određene dubine (u ovom slučaju verzija koja pretražuje po vjerojatnosti bi vjerojatno bila slična ”negamaksu 6.5”). U završnicama partija pretraživanje po vjerojatnosti uistinu daje brže rezultate, ali kod npr. samog matiranja kralja (kad je na raspolaganju samo 1 ili 2 poteza), oba su načina pretraživanja jednako dobra jer i negamaks pretraživanje brzo dolazi do velike dubine.

Šahovski program AlphaZero

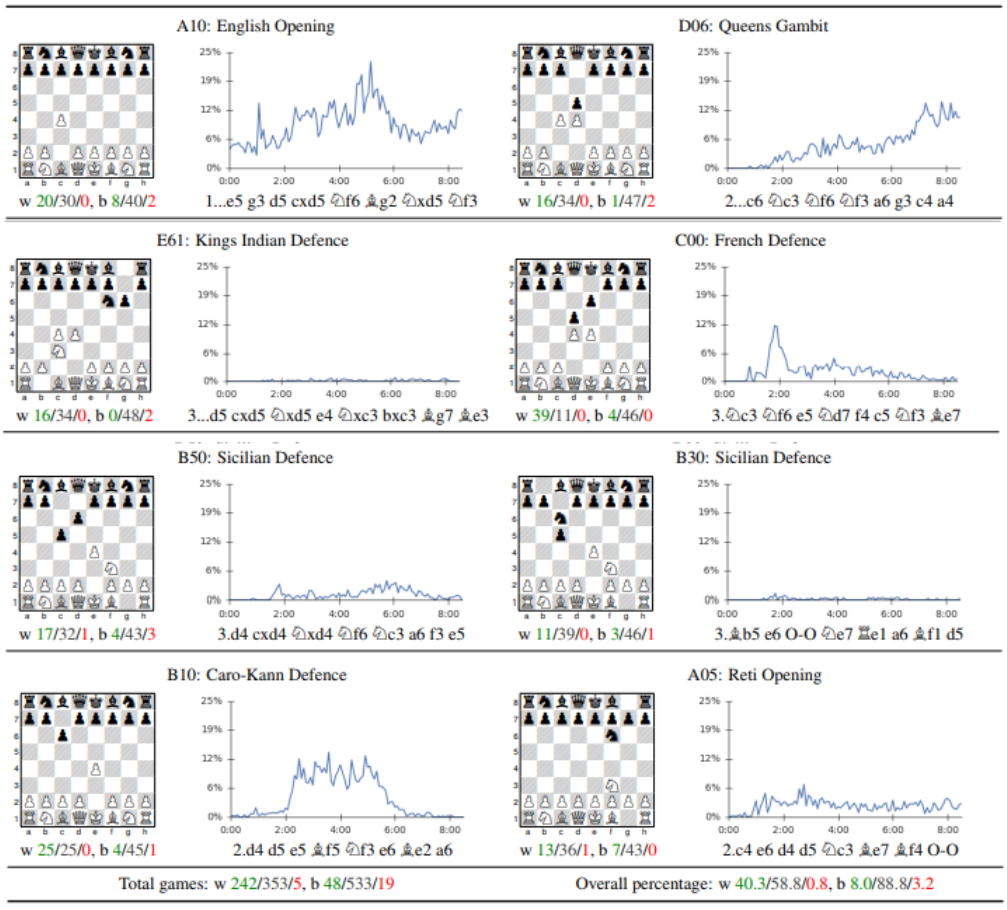
U šahovskom svijetu, 6. prosinca 2017., munjevitom se brzinom proširila vijest o novom najjačem šahovskom računalu, AlphaZero, koje je deklasiralo program Stockfish pobije-

divši ga rezultatom 64-36, bez poraza. Ovaj rezultat mnogi smatraju najvećim dosegom u šahovskoj povijesti još od meča Kasparov-Deep Blue, a uzevši u obzir da funkcionira na drugačiji način od gotovo svih šahovskih programa na svijetu, program AlphaZero odmah je pobudio veliki interes šahovske publike. Algoritam je produkt tvrtke DeepMind, otkupljene od strane Googlea 2014. godine, a njegov "predak" je sad već nadaleko poznat program AlphaGo, koji je u ožujku 2016. godine pobijedio svjetskog prvaka u Go-u Leea Sedola rezultatom 4-1. Program AlphaGo koristio se neuronskim mrežama koje su bile trenirane nadziranim učenjem, pomoću partija najjačih ljudskih igrača, i podržanim učenjem igrajući sam protiv sebe. Manje je poznato da je tijekom iste godine objavljen rad [9] u kojem se predstavlja AlphaGo Zero, novija verzija programa AlphaGo, koja koristi isključivo podržano učenje. AlphaGo Zero je u meču dobio AlphaGo 100-0 i pritom pokazao mnoge nove Go strategije kojih se nije dosjetio ni jedan ljudski igrač. Svi ovi rezultati izuzetno su značajni jer je DeepMind tim pokazao da je podržano učenje (reinforced learning) igrom protiv samog sebe , uz dovoljno vrijeme za treniranje, bolje od nadziranog učenja korištenjem partija ljudskih igrača. AlphaGo Zero koristio se sa samo 4 TPU-a (tensor processing unit), dok je originalni AlphaGo bio distribuiran na više mašina i koristio ukupno 48 TPU-a.

Nakon ovih, veoma značajnih rezultata, DeepMind tim je odlučio modificirati AlphaGo Zero program u jedinstveni algoritam AlphaZero koji, neovisno o vrsti igre (tako dugo dok je to igra za dva agenta sa savršenim informacijama), kreće od osnovnih pravila i kroz podržano učenje poboljšava svoju igru [25]. AlphaZero koristi duboku neuronsku mrežu (deep neural network) koja kao ulazni argument uzima poziciju s , a vraća vektor vjerojatnosti p s komponentama $p_a = P(a|s)$ za svaki potez a, i skalarnu vrijednost v koja predstavlja očekivani rezultat z iz pozicije s, $v \approx E[z|s]$. Umjesto minimaks pretraživanja, koristi se pretraživanje stabla koristeći Monte Carlo metode [21]. Za svaki potez stablo se pretražuje nasumičnim generiranjem niza mogućih scenarija i računanjem srednje vrijednosti tog poteza u svim scenarijima u kojima se pojavljuje. Pritom se preferiraju potezi koji još nisu isprobani, čija je vjerojatnost pojavljivanja veća, i potezi za koje heuristička evaluacijska funkcija daje bolji rezultat. Parametri neuronske mreže koja evaluira poziciju treniraju se na ovaj način i na kraju partije budu korigirani tako da se minimizira greška između očekivanog rezultata i ostvarenog rezultata. Za treniranje je korišteno 5000 TPU-ova za generiranje igara i 64 za treniranje mreža. Već u fazi učenja program je redovito igrao mečeve protiv Stockfisha, i nakon 4 sata učenja nadmašio je Stockfishovu razinu igre. Zatim je, nakon 9 sati treniranja i 44 milijuna odigranih trening partija, završna verzija programa AlphaZero odigrala meč protiv Stockfisha.

Uvjeti odigravanja meča bili su neobični: jedna minuta po potezu, Stockfish nije smio koristiti knjigu otvaranja ni tablebase, ali je na raspolaganju imao 64 CPU dretve i

hash tablicu od 1 GB. Osim samog rezultata, AlphaZero je kroz igru došao do zanimljivih otkrića za sve šahiste: program je iznova ”otkrio” neka od najčešćih otvaranja (ljudima su trebala stoljeća da ih usavrše) i tijekom treninga odbacio neka od njih. Ova saznanja o kvaliteti nekih otvaranja, iako preliminarna, zasigurno će imati dugoročne posljedice na šahovski svijet, pogotovo na svjetsku elitu. Prema programu AlphaZero, najbolji potez za bijelog je 1.c4 (Englesko otvaranje), koje bilježi rast popularnosti u zadnjih desetak godina u odnosu na 1.d4 (igra Daminim pješakom). Druga često igrana otvaranja poput Kraljeve indijske obrane, Francuske obrane i Caro-Kanna pokazala su se neučinkovitima protiv programa takve jačine.



Slika 3.3: Broj partija i rezultati programa Alphazero u raznim šahovskim otvaranjima

U članku je priloženo i 10 oglednih partija Stockfisha protiv AlphaZeroa, koje su također zanimljive. Moje je osobno mišljenje da AlphaZero igra šah na mnogo "ljudskiji" način, često žrtvujući materijal za dugotrajne pozicijske faktore. Ovaj aspekt posebno je zanimljiv jer je u većini šahovskih sustava materijal dominantna komponenta evaluacijske funkcije, dok se svi ostali parametri često podcjenjuju. Također, AlphaZero bolje koristi neke strateške motive koji su razumljivi ljudima, a teški za pretvoriti u programski kod. Svi ovi rezultati još uvijek su upitni, budući da je 100 partija premali uzorak da bi se moglo zaključiti nešto o stilu igre šahovskog programa. Interesantna je činjenica da AlphaZero, usprkos golemoj memoriji na raspolaganju, pretražuje "samo" 80 000 čvorova u sekundi u usporedbi sa Stockfishom koji pretražuje 70 milijuna čvorova u sekundi. Ovaj rezultat pokazuje da uz dobar odabir značajki program za igranje šaha može funkcionirati puno bolje i smanjiti potrebu za pretraživanjem stabla u dubinu, što je glavni fokus svih računalnih programa za igranje šaha.

Slika 3.4: Pozicijska žrtva lovca iz 2. objavljene partije meča

Na slici se nalazi pozicija prije 30. poteza izvrsne strateške partije programa AlphaZero, koji u gornjoj poziciji žrtvuje lovca za pješaka bazirajući žrtvu na pozicijskim karakteristikama (prostornoj prednosti, slabom kralju crnog, crnom lovcu koji je izvan igre). Partija je trajala još 40 poteza (20 bijelog i 20 crnog), a program Stockfish primijetio je da je izgubljen tek 20 poteza nakon žrtve.

Ubrzo nakon objave članka, pojavila su se i razna pitanja o korektnosti meča. Jedan od tvoraca Stockfisha, Tore Romstad, tako je na jednom forumu rekao da rezultati meča nisu bitni jer meč nije odigran po pravilima za računalni šah (slično pravilima za turnirski šah) pa neke od Stockfishovih specijalno dizajniranih vremenskih heuristika (prepoznavanje kritičnih trenutaka i trošenje više vremena u tim pozicijama) nisu mogle biti od koristi što je bitno oslabilo taj program. Također, usporedba je i promašena time što je Stockfish komercijalno dostupan program (dapače open source), dok je AlphaZero specijaliziran program koji radi na skupim računalnim komponentama, pa je samim time nedostupan prosječnom korisniku u dogledno vrijeme. Nadalje, Stockfish je na raspolaganju imao samo 1 GB memorije za transpozicijske tablice, što je premalo za spremanje svih rezultata pretrage za dubinu do koje Stockfish pretražuje (u središnjici više od 20 poteza). No s druge strane, potrebno je napomenuti da je AlphaZero algoritam koji funkcionira za više igara. Tako je uz minimalne modifikacije (promjena pravila igre i nekih parametara) nakon 2 sata treniranja igrao bolje od vodećeg programa za igru Shogi, a nakon 8 sati treniranja igrao je igru Go bolje od programa AlphaGo Zero. Usprkos (opravdanim) kritikama i usprkos tome što nije moguće upotrijebiti ovu novu tehnologiju, vjerujem da je DeepMind tim napravio idući korak u evoluciji računalnog šaha upotrijebivši pritom potpuno novi pristup. Iako komercijalno nedostupan i skup, AlphaZero je pomaknuo granice računalnog šaha i pokazao novi put razvoja. Problem konstrukcije pametnog programa za igranje šaha bio je jedan od centralnih izazova za rane istraživače na polju umjetne inteligencije i prigodno je da nakon 50 godina opet bude prezentiran kao rezultat jedne nove znanstvene discipline.

Bibliografija

[1] *Definicija šaha*, dec 2017, https://www.merriam-webster.com/dictionary/chess, pristupljeno: svibanj 2018.

[2] F. Gobet B. Macnamara D. Hambrick G. Campitelli A. Burgoyne, G. Sala, *The relationship between cognitive ability and chess skill: A comprehensive meta-analysis*, http://doi.org/10.1016/j.intell.2016.08.002, pristupljeno: svibanj 2018.

[3] AGON, *AGON releases new chess player statistics from YouGov*, aug 2012, https://www.fide.com/component/content/article/1-fide-news/6376-agon-releases-new-chess-player-statistics-from-yougov.html, pristupljeno: svibanj 2018.

[4] Thomas Dybdahl Ahle, *Sunfish*, 2014, https://github.com/thomasahle/sunfish, pristupljeno: svibanj 2018.

[5] J. Šnajder B. Dalbelo Bašić, *Pretraživanje prostora stanja*, 2013, http://degiorgi.math.hr/~singer/ui/ui_1415/UI-2-PretrazivanjeProstoraStanja.pdf, pristupljeno: svibanj 2018.

[6] Jonathan Baxter, Andrew Tridgell i Lex Weaver, *TDLeaf(lambda): Combining Temporal Difference Learning with Game-Tree Search*, http://arxiv.org/abs/cs.LG/9901001, pristupljeno: svibanj 2018.

[7] Russ Cox, *Play Chess with God*, 2008, https://research.swtch.com/chess, pristupljeno: svibanj 2018.

[8] O. E. David, H. J. van den Herik, M. Koppel i N. S. Netanyahu, *Genetic Algorithms for Evolving Computer Chess Programs*, IEEE Transactions on Evolutionary Computation, http://ieeexplore.ieee.org/document/6626616, pristupljeno: svibanj 2018.

[9] Karen Simonyan Ioannis Antonoglou Aja Huang Arthur Guez Thomas Hubert Lucas Baker Matthew Lai Adrian Bolton Yutian Chen Timothy Lillicrap Fan Hui Laurent Sifre George van den Driessche Thore Graepel & Demis Hassabis David Silver, Julian Schrittwieser, *Mastering the game of Go without human knowledge*, http://www.nature.com/articles/nature24270, pristupljeno: svibanj 2018.

[10] Donald E. Knuth i Ronald W. Moore, *An analysis of alpha-beta pruning*, (1975), 293–326.

[11] FIDE, *Pravila šaha FIDE*, jan 2018, http://bit.ly/2BxA5qe, pristupljeno: svibanj 2018.

[12] Mark E. Glickman, *A comprehensive guide to chess ratings*, http://www.glicko.net/research/acjpaper.pdf, pristupljeno: svibanj 2018.

[13] IBM, *Deep blue (chess computer)*, 2008, http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/, pristupljeno: svibanj 2018.

[14] M. Hlynka J. Schaeffer, M. Jussila, *Temporal Difference Learning Applied to a High-Performance Game-Playing Program.*, 2001, https://dl.acm.org/citation.cfm?id=1642090.1642163, pristupljeno: svibanj 2018.

[15] Joona Kiiski, *Stockfish's tuning method*, 2011, https://chessprogramming.wikispaces.com/Stockfish's+Tuning+Method, pristupljeno: svibanj 2018.

[16] Matthew Lai, *Giraffe: Using Deep Reinforcement Learning to Play Chess*, (2015), http://dblp.org/rec/bib/journals/corr/Lai15a, pristupljeno: svibanj 2018.

[17] François Dominic Laramée, *Chess Programming Part II: Data Structures*, 2000, https://www.gamedev.net/articles/programming/artificial-intelligence/chess-programming-part-ii-data-structures-r1046, pristupljeno: svibanj 2018.

[18] Sashi Lazar, *Analysis of Transposition Tables and Replacement Schemes*, (1996).

[19] J. McCarthy, *Chess as the Drosophila of AI*, (1990), http://link.springer.com/chapter/10.1007/978-1-4613-9080-0_14, pristupljeno: svibanj 2018.

[20] Aske Plaat, *MTD(f), a new minimax algorithm faster than NegaScout*, http://people.csail.mit.edu/plaat/mtdf.html, pristupljeno: svibanj 2018.

[21] Casella P. Robert, C.P., *Monte Carlo statistical methods*, (1998), http://bit.ly/2sSp3dt, pristupljeno: svibanj 2018.

[22] Claude E. Shannon, *Programming a computer for playing chess*, (1950), http://pdfs.semanticscholar.org/64a0/3e75e867b67de855d6d6c6013df9600d53ae.pdf, pristupljeno: svibanj 2018.

[23] Harold C. Shonberg, *Kasparov Beats Chess Computer (for Now)*, 1989, http://www.nytimes.com/1989/10/23/nyregion/kasparov-beats-chess-computer-for-now.html, pristupljeno: svibanj 2018.

[24] Ed Shroeder, *Experiments in computer chess: The value of depth and diminishing return effects*, 2012, http://www.top-5000.nl/ply.htm, pristupljeno: svibanj 2018.

[25] Thomas; Schrittwieser Julian; Antonoglou Ioannis; Lai Matthew; Guez Arthur; Lanctot Marc; Sifre Laurent; Kumaran Dharshan; Graepel Thore; Lillicrap Timothy; Simonyan Karen; Hassabis Demis Silver, David; Hubert, *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, http://arxiv.org/abs/1712.01815, pristupljeno: svibanj 2018.

[26] Marek Strejsczek, *Some aspects of chess programming*, Magistarska radnja, University of Lodz, 2004, http://www.top-5000.nl/ps/SomeAspectsOfChessProgramming.pdf, pristupljeno: svibanj 2018.

[27] Condon J.H. Thompson, K., *"Belle" - in Chess skill in Man and Machine*, (1977), http://www.springer.com/gp/book/9780387908151, pristupljeno: svibanj 2018.

[28] J. von Neumann, *Zur Theorie der Gessellschaftsspiele*, (1928), http://doi.org/10.1007/BF01448847.

Sažetak

Računalni šah je zbog svoje veze s inteligencijom 50-ih godina prošlog stoljeća postao glavnim predmetom istraživanja pionira na polju umjetne inteligencije. U ovom radu istražuje se način funkcioniranja programa za igranje šaha i detaljno se objašnjavaju elementi svakog modernog šahovskog programa. Također, obrazlaže se vlastita implementacija struktura i algoritama za konstrukciju šahovskog programa. U drugom dijelu rada pobrojani su novi trendovi u konstrukciji šahovskih programa s naglaskom na tehnikama strojnog učenja koje se koriste u zadnjih desetak godina.

Summary

Due to its correlation with intelligence, computer chess has become the main subject of pioneer research in the field of artificial intelligence in the 1950s. In this thesis, the functionality of a chess program is explored and the basic elements of every modern chess program are explained in detail. Also, an implementation of structures and algorithms for the construction of a chess program is given. The second part of the thesis lists new trends in the construction of chess programs with emphasis on machine learning techniques used in the last ten years.

Životopis

Rođen sam 15. lipnja 1992. godine u Čakovcu. Nakon završene osnovne škole, 2007. godine upisujem Prirodoslovno-matematičku gimnaziju u Čakovcu. Zanimanje za matematiku potaklo me da 2011. godine upišem preddiplomski sveučilišni studij Matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu. Godine 2015. završavam preddiplomski studij, dobivam titulu univ. bacc. math. i odlučujem nastaviti studiranje na diplomskom sveučiličnom studiju Matematika i informatika; smjer: nastavnički. Godine 2016. zapošljavam se u tvrtki Photomath gdje i danas radim kao kreator matematičkog sadržaja. Iduće godine kao dio grupe autora Kovač, Kovač, Krišto objavljujem znanstveni rad: Nastavni plan i program informatike i informiranost učenika gimnazije o sigurnosti i zaštiti zdravlja pri radu s računalom. Krajem 2017. godine počinjem volonterski raditi s nadarenim osnovnoškolcima u sklopu programa RADDAR (rad s nadarenima), a od ožujka 2018. godine zaposlen sam kao asistent na visokom učilištu Algebra. Od početka 2018. godine također sudjelujem kao autor u projektu pisanja udžbenika iz informatike za više razrede osnovne škole.