Elektrotehniški vestnik 73(1): 31–37, 2006
Electrotechnical Review, Ljubljana, Slovenija

Objektno orientirano načrtovanje in implementacija računalniškega šaha[1]

Borko Bošković, Janez Brest, Viljem Žumer

Univerza v Mariboru, Fakulteta za elektrotehniko računalništvo in informatiko, Smetanova ulica 17, 2000 Maribor, Slovenija
E-pošta: borko.boskovic@uni-mb.si

Povzetek. V članku predstavljamo sestavne dele računalniškega šaha in njegovo objektno orientirano načrtovanje ter implementacijo. Načrtovani program smo implementirali v programskem jeziku java. Tako smo dobili programsko okolje, ki omogoča hitro spreminjanje programa in njegovo preizkušanje. Implementiran program, ki ga lahko zaganjamo na vseh platformah java, smo preizkusili in ugotovili, da je relativno zmogljiv. Program je pri testiranju rešil veliko testnih pozicij in dosegel pozitivni rezultat v igri z amaterskimi igralci šaha.

Ključne besede: šahovski program, objektno orientirano načrtovanje in implementacija, predstavitev šahovske igre, iskalni algoritmi, testiranje šahovskih programov

Object-oriented design and implementation of computer chess

Extended abstract. *The paper presents an object-oriented design and implementation of a chess program written in Java. It enables adding and testing additional representations of the game and search algorithms. This includes representation of chess game in the computer and search algorithms used in the developed chess program. The program which was tested with test positions and human chess players, solves many test positions and achieves positive scores with amateur chess players.*

Key words: chess program, object-oriented design end implementation, representation of the chess game, search algorithm, testing chess program

[1]Članek je bil predstavljen na konferenci ERK v sekciji študentskih člankov.

1. Uvod

Z naraščanjem zmogljivosti računalnikov in z razvojem iskalnih algoritmov in komponent računalniškega šaha se je povečevala tudi zmogljivost sistemov za igranje šaha. Le-ti so postali enakovredni človeku, celo svetovni šahovski prvak jih s težavo premaguje. Kako računalniku uspeva tako brilijantno igranje šaha? Da bi odgovorili na zastavljeno vprašanje, smo implementirali šahovski program, ki nam je omogočil preizkušanje iskalnih algoritmov in mehanizmov, ki se uporabljajo v računalniškem šahu. Za lažje preizkušanje različnih sestavnih delov šahovskih programov smo se odločili za objektno orientirano načrtovanje in implementacijo ter programski jezik java. Cena te odločitve je nekoliko manj zmogljiv šahovski program v primerjavi s programi, ki so napisani v jeziku C ali C++. Testiranje je pokazalo, da program kljub tej odločitvi zna rešiti dokaj zahtevne pozicije in zna povzročiti obilo preglavic amaterskim šahovskim programom in igralcem.

Jedro problema pri izdelavi šahovskih programov je zelo veliko število različnih kombinacij oz. pozicij, ki jih vsebuje šahovska igra. Šahovsko igro lahko predstavimo z drevesom igre. To je drevo, ki v korenu vsebuje začetno pozicijo, v listih drevesa pa končne pozicije igre. Velikost tako zgrajenega drevesa je približno 10^{123} vozlišč, ki pa ga danes ni mogoče preiskati z nobenim računalnikom ali strojem [10].

Tudi človek ni idealen igralec in postavlja se naslednje vprašanje. Koliko potez vnaprej oz. do katere globine je treba preiskovati iskalno drevo, da bomo premagali človeško strategijo? Ken Thompson je v 80. letih izvedel zanimiv poskus o korelaciji med globino iskanja in močjo igranja. Thompson je pustil igrati računalnik samega proti sebi z naraščanjem globine iskanja. Z analizo pridobljenih rezultatov je prišel do sklepa, da bi računalnik s hitrostjo preiskovanja bilijon pozicij na sekundo (globino 14) imel moč svetovnega šahovskega prvaka [10].

Sodobni šahovski programi za doseganje končnega

Prejet 24. december, 2004
Odobren 19. oktober, 2005

cilja (zmaga) vsebujejo več komponent. Te komponente oz. arhitekturo šahovskih programov predstavljamo v drugem poglavju. V tretjem poglavju predstavljamo načrtovanje in implementacijo našega programa. To poglavje se sestoji iz petih podpoglavij. V prvem podpoglavju je opisana predstavitev igre, drugo podpoglavje opisuje iskalne algoritme, tretje je namenjeno grafičnim uporabniškim vmesnikom, četrto predstavlja arhitekturo programa, peto pa podaja prednosti in slabosti našega programa. V četrtem poglavju predstavimo testiranje našega programa. Predstavljen je način testiranja šahovskih programov ter rezultati določenih iskalnih algoritmov našega programa. V petem poglavju je podan še kratek pregled opravljenega dela in zaključek.

2. Arhitektura šahovskih programov

Način programiranja in uporabljene ideje so zelo pomembni pri razvoju šahovskih programov. Danes najboljši programi, kot sta npr. Fritz in Junior, zmorejo preiskati več kot milijon pozicij na sekundo. Njihova realna moč je približno 2700 točk ELO. Z njimi se lahko enakopravno pomeri le nekaj 100 najboljših igralcev na svetu. V hitropoteznem šahu pa se z njimi lahko meri le peščica igralcev [10].

Za doseganje vrhunskih rezultatov šahovski programi vsebujejo naslednje sestavne dele: predstavitev igre, generator potez, iskalne algoritme, ocenitveno funkcijo, otvoritvene knjižnice in podatkovno bazo končnic [1, 2, 3, 4]. Iskalni algoritmi na podlagi ocenitvene funkcije (podaja statično oceno pozicije), predstavitve igre (predstavi igro v računalniku) in generatorja potez (generira vse mogoče poteze za določeno pozicijo) preiskujejo iskalno drevo in na koncu podajo izbrano potezo za nadaljevanje igre [2, 5, 6, 7, 8]. Otvoritvene knjižnice vsebujejo zbrana dolgoletna znanja šahovskih mojstrov in omogočajo šahovskim programom igranje nekaj začetnih potez brez uporabe iskalnih algoritmov. Podobno podatkovna baza končnic vsebuje podatke o šahovskih končnicah in omogoča šahovskim programom brezhibno igranje končnic. Z uporabo otvoritvenih knjižnic, iskalnih algoritmov in podatkovne baze končnic tako šahovski programi dosegajo vrhunske rezultate v vseh fazah igre.

3. Načrtovanje in implementacija

Da bi preizkusili omenjene mehanizme in algoritme računalniškega šaha, smo objektno načrtovali in implementirali šahovski program ter ga na koncu tudi preizkusili. Program smo implementirali v programskem jeziku java in tako omogočili njegovo izvajanje na različnih platformah. Načrtovali in implementirali smo bitno predstavitev igre [2] (temelji na bitnih operacijah), vmesnik UCI (*Universal Chess Interface*) za povezovanje šahovskih programov in grafičnih uporabniških vmesnikov in iskalne algoritme.

Program smo načrtovali tako, da so posamezne komponente računalniškega šaha med seboj neodvisne. Na primer, če želimo preizkusiti nov iskalni algoritem, je treba s pomočjo dedovanja implementirati abstraktne metode določenega abstraktnega razreda. Pri implementaciji teh metod ni potrebno skrbeti o predstavitvi igre, vmesniku UCI, merjenju časa itd. Zaradi hitrosti programa in vmesnikov vseh komponent ni bilo mogoče tako načrtovati. Primera takih komponent sta predstavitvi potez in figur. Predstavitev potez mora biti kompaktna in jedrnata. Z njimi se izvajajo operacije nad pozicijami, z njimi se manipulira in shranjujejo se za poznejšo uporabo [2, 8]. Poteze so prav tako vmesnik med iskalnim algoritmom in predstavitvijo igre. Tako vidimo, da predstavitev potez ni mogoče poljubno spreminjati. Zato smo se odločili, da takim komponentam določimo fiksno obliko.

3.1. Predstavitev igre

Predstavitev šaha mora omogočati shranjevanje in manipulacijo s šahovskimi pozicijami in potezami. Pozicija igre je trenutno stanje v igri, poteza igralca pa premik ene njegovih figur v trenutni poziciji. Predstavitev igre je treba zasnovati tako, da omogoča izvajanje naslednjih operacij:

- izvajanje določene poteze (v primeru zahtev uporabnika in kot del iskalnega algoritma),
- vračanje poteze (v primeru zahtev uporabnika in kot del iskalnega algoritma),
- predstavitev igre uporabniku,
- generiranje seznama vseh mogočih potez,
- shranjevanje potez in pozicij,
- primerjanje dveh pozicij in
- ocenjevanje pozicij.

Vse naštete operacije, razen predstavitve igre, morajo biti hitre, saj se uporabljajo v iskalnih algoritmih. V ta namen je treba izbrati določeno predstavitev potez in pozicij.

3.2. Iskalni algoritmi

Iskalni algoritem je osrednji del šahovskih programov. Le-ta na določen način preiskuje iskalno drevo s pomočjo ocenitvene funkcije, na koncu pa poda potezo

za nadaljevanje igre. Z matematičnega stališča lahko igro šah opišemo kot igro med dvema nasprotnikoma, s popolno informacijo in ničelno vsoto [2]. Kot vemo, je šah igra med dvema igralcema, in sicer med belim in črnim. Oba igralca imata popoln pregled nad celotno igro (pozicijo in potezami), zato takim igram pravimo tudi igre s popolno informacijo. Dodatna lastnost šahovske igre je "ničelna vsota". Določa jo strukturno ravnotežje v tekmovalni naravi igre. Igra poteka z izmenjavo potez med igralcema. Vsaka od legalnih potez prinaša določeno prednost oz. slabost za igralca na potezi oz. njegovega nasprotnika. Tako lahko npr. potezo p ovrednotimo za oba igralca (belega - $eval_w(p)$, črnega - $eval_b(p)$) in dobimo naslednji izraz oz. "ničelno vsoto":

$$eval_w(p) + eval_b(p) = 0$$

Na podlagi tega matematičnega opisa lahko zgradimo drevo igre. Drevo igre vsebuje približno w^d vozlišč, kjer w pomeni povprečno število potez na eno pozicijo, d pa povprečno število potez na partijo. Takega iskalnega drevesa praktično ni mogoče preiskati, zato šahovski programi uporabljajo algoritme, ki preiskujejo samo del drevesa igre. Ta del drevesa imenujemo tudi iskalno drevo. Iskalno drevo v korenu vsebuje trenutno pozicijo igre, v listih pa vozlišča, ki zadoščajo določenemu pogoju. Ta pogoj je lahko npr. določen z globino iskanja.

V okviru našega programa smo načrtovali in implementirali naslednje algoritme:

- *alfa-beta,*

  Osnovni algoritem šahovskih programov. Temelji na preiskovanju iskalnega drevesa do določene globine [9, 11, 12].

- *iskanje mirovanja,*

  Je namenjen ocenjevanju dinamičnih pozicij, ki jih ocenitvena funkcija ne more oceniti. Algoritem nadaljuje preiskovanje do statičnih pozicij, ki jih nato oceni ocenitvena funkcija [11, 12].

- *iterativno poglabljanje,*

  Omogoča časovno omejeno iskanje s pomočjo iterativnega poglabljanja oz. povečevanja globine iskanja v algoritmu alfa-beta [11, 12].

- *aspiracijsko iskanje,*

  Glede na rezultat iskanja iz prejšnje iteracije ugiba rezultat iteracije, ki jo bo pravkar preiskoval [11, 12].

- *iskanje na osnovi glavne variante,*

  Najprej poišče določeno glavno varianto (seznam izbranih potez), nato pa jo z uporabo iskanja z minimalnim oknom (alfa-beta algoritem, ki ima velikost okna 0) poskuša izboljšati [11, 12].

- *MTD(f)* in

  S pomočjo iterativnega poglabljanja, z iskanjem z ničelnim oknom in s transpozicijsko tabelo (tabela, ki vsebuje preiskane pozicije, na podlagi katerih iskalni algoritem dodatno klesti iskalno drevo) na določen način preiskuje iskalno drevo [5].

- *klestenje z ničelno potezo.*

  Je selektivni algoritem, ki za doseganje večje iskalne globine dodatno klesti iskalno drevo. S pomočjo ničelne poteze (zamenjava igralca na potezi) se drevo dodatno klesti z možnostjo, da določena pomembna vozlišča spregledamo [11, 12].

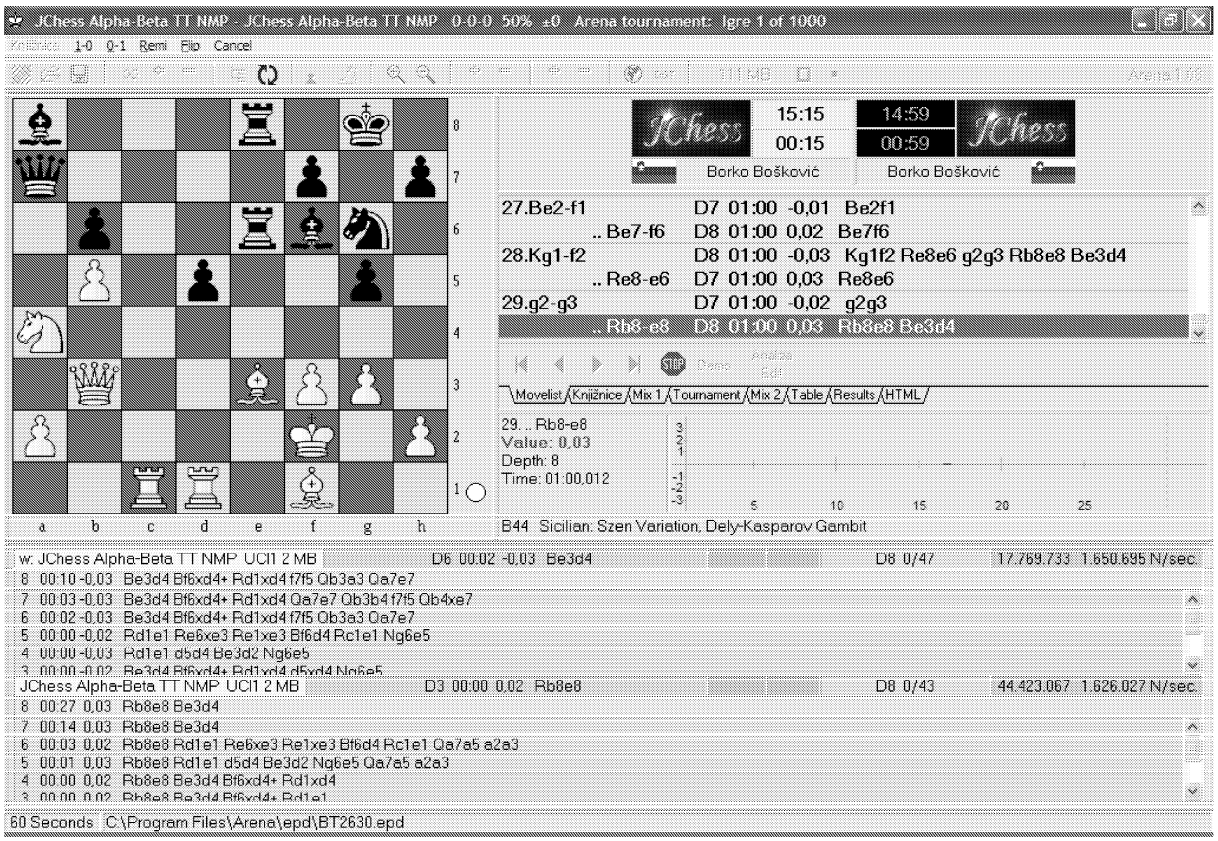
Poleg opisanih algoritmov program vsebuje še otvoritveno knjižnico. Otvoritveno knjižnico iskalni algoritem uporablja v začetni fazi igre. Preden se sproži iskalni algoritem, najprej pogledamo v otvoritveno knjižnico. V primeru, ko otvoritvena knjižnica vsebuje nadaljevanja igre, naključno izberemo eno ponujenih potez in končamo preiskovanje. V nasprotnem primeru preiskovanje nadaljujemo z iskalnim algoritmom. Podatkovne baze končnic naš program v tem trenutnku še ne vsebuje. V nadaljnjem razvoju programa jo bomo vključili, tako da jo bo iskalni algoritem lahko uporabil v končnih fazah igre.

3.3. Grafični uporabniški vmesnik

Pri implementaciji šahovskega programa je zelo pomemben tudi grafični uporabniški vmesnik. Njegova naloga je, da uporabniku omogoči uporabniško prijazno komuniciranje s šahovskim programom. Ker pa se v ozadju izvaja šahovski program, se od grafičnega vmesnika dodatno zahteva čim manjše obremenjevanje računalniških virov. Grafični vmesnik naj bi omogočal tudi primerjavo različnih šahovskih programov. Tako so razvijalci šahovskih programov začeli grafični vmesnik razvijati ločeno od šahovskega programa. V ta namen je standardiziran vmesnik UCI (*Universal Chess Interface*), s pomočjo katerega komunicirata šahovski program in grafični vmesnik. Tudi naš program ne vsebuje grafičnega uporabniškega vmesnika, ima pa implementiran vmesnik UCI, ki omogoča povezovanje našega programa z grafičnimi vmesniki, kot so npr. Arena, Jose, Fritz itd. Grafični vmesnik, ki smo ga mi uporabljali pri implementaciji in testiranju, se imenuje Arena [13] in prikazuje ga slika 1.

3.4. Arhitektura programa

Arhitekturo našega programa med izvajanjem sestavljajo tri niti. Prva nit je vmesnik UCI, ki komu-



Slika 1. Grafični uporabniški vmesnik Arena

Figure 1. Arena Chess Graphical User Interface.

nicira z grafičnim uporabniškim vmesnikom in koordinira preostali dve niti. Druga nit s pomočjo iskalnih algoritmov preiskuje drevo igre. Tretja nit pa skrbi za časovno omejena iskanja. Tedaj, ko mine čas, predviden za določeno pozicijo, tretja nit zaustavi iskalni algoritem in poda izbrano potezo oz. glavno varianto (seznam izbranih potez).

Čas, predviden za reševanje ene pozicije, določa uporabnik znotraj grafičnega uporabniškega vmesnika, kadar se odloča za tip igre oz. analize pozicije. Pri analizi, čas za preiskovanje določajo parametri protokola UCI. Pri igre čas za reševanje ene pozicije program določi po naslednji enačbi:

$$t_{poz} = t_d + \frac{t_{pr}}{poteze} \cdot 3,$$

kjer t_{poz} pomeni čas za reševanje pozicije, t_d dodatek časa na potezo, *poteze* število potez, ki jih mora program odigrati v časovni omejitvi t_{pr}. Ko je igra le časovno omejena, program nima informacije o številu preostalih potez, ki jih mora odigrati v določenem času. V tem primeru program predvideva, da se bo igra končala v 40 potezah.

Po poteku časa, ki je predviden za neko pozicijo, tretja nit sproži programsko prekinitev v drugi niti (iskalni algoritem). Kadar druga nit dobi programsko prekinitev, takoj poda trenutno najboljšo potezo in konča preiskovanje. Izbrana poteza se nato s pomočjo protokola UCI in niti UCI posreduje grafičnemu vmesniku, ki jo nato prikaže uporabniku.

Zaustavitveni kriterij pa ni nujno samo čas. Iskanje se lahko zaustavi glede na število preiskanih vozlišč, doseženo globino preiskovanja ali pa na zahtevo uporabnika oz. uporabniškega vmesnika.

3.5. Prednosti in slabosti

Prednosti, ki smo jih pridobili z razvojem našega programa, so:

- *program lahko zaganjamo na vseh platformah java,*

  Program je napisan v programskem jeziku java in omogoča njegovo izvajanje na vseh platformah java.

- *preprosto vzdrževanje in prilagajanje programa,*

  Program smo načrtovali tako, da omogoča preprosto dodajanje in testiranje novih predstavitev igre in iskalnih algoritmov. Te komponente so med seboj neodvisne in jih tako tudi lahko implementiramo.

- *testiranje posameznih komponent in*

Tabela 1: Uspešnost iskalnih algoritmov

Table 1. Efficiency of search algorithms

sek./poz.	WAC			ECM			WCSAC		
Algoritem	Rezultat	Uspešnost	Čas [s]	Rezultat	Uspešnost	Čas [s]	Rezultat	Uspešnost	Čas [s]
AB	164	54.7%	763	156	17.8%	3734	524	52.4%	2598
AB TT	213	71.0%	515	250	28.4%	3306	653	65.2%	1961
AB TT KNP	229	76.3%	446	293	33.3%	3129	688	68.7%	1805
GV	195	65.0%	625	200	22.8%	3518	595	59.4%	2246
GV TT	213	71.0%	520	249	28.3%	3304	653	65.2%	1959
GV TT KNP	226	75.3%	452	290	33.0%	3156	684	68.3%	1813
MTD(*f*)	193	64.3%	654	216	24.6%	3463	595	59.4%	2331

Legenda: AB – alfa-beta; AB TT – alfa-beta s transpozicijsko tabelo; AB TT KNP – alfa-beta s transpozicijsko tabelo in klestenjem z ničelno potezo; GV – iskanje na podlagi glavne variante; GV TT – iskanje na podlagi glavne variante s transpozicijsko tabelo; GV TT KNP – iskanje na podlagi glavne variante s transpozicijsko tabelo in klestenjem z ničelno potezo; MTD(*f*) – algoritem MTD(*f*).

Zaradi objektnega pristopa lahko obnašanje določenih komponent in njihovih nastavitev testiramo v različnih okoljih (nastavitvah programa).

- *testiranje različnih nastavitev programa.*

  Program lahko testiramo z različnimi nastavitvami ter tako povečamo zmogljivost programa.

Edina slabost naše implementacije je nekoliko počasnejši program zaradi programskega jezika java in tudi zaradi objektno orientiranega načrtovanja in implementacije.

4. Testiranje

Implementirane algoritme smo tudi testirali. Pri testiranju smo uporabili računalnik s procesorjem Pentium 4 2.8 GHz in 512 MB pomnilnika. Testiranje smo izvajali s pomočjo naslednjih testnih knjižnic: Encyclopedia of Chess Middlegames (ECM), Win at Chess (WAC), 1001 Wining Chess Sacrifices (WCS), BT2630 in LCT II, ki so javno dostopne na svetovnem spletu. Pri testiranju je program preiskoval od 1 200 000 do 2 000 000 pozicij na sekundo.

Tabela 1 prikazuje rezultate testiranja programa glede na različne iskalne algoritme. Pri tem testiranju smo uporabili testne knjižnice WAC, ECM in WCSAC, program pa smo časovno omejili na 5 sekund za reševanje ene pozicije. Iz rezultatov vidimo, da so algoritmi pri pozicijah, ki vodijo k zmagi (WAC in WCSAC), dosegeli od 52,1 do 70,5 odstotno uspešnost. Nekoliko slabše rezultate so dosegli pri pozicijah sredine igre (ECM). Tukaj so dosegli od 17.8 do 33.3 odstotno uspešnost. Razlog je po našem mnenju preprosta ocenitvena funkcija, ki vsebuje premalo šahovskega znanja. V tabeli 2 prikazujemo še skupne rezultate tega testiranja. Iz rezultatov vidimo, da se je kot najboljši algoritem izkazal alfa-beta s transpozicijsko tabelo in klestenjem z ničelno potezo.

Tabela 2: Skupna uspešnost

Table 2. Total efficiency

5 sek./poz.	WAC + ECM + WCSAC		
Algoritem	Rezultat	Uspešnost	Čas [s]
AB	844	38.7%	7095
GV	990	45.4%	6389
MTD(*f*)	1004	46.1%	6448
GV TT	1115	51.2%	5782
AB TT	1116	51.2%	5783
GV TT KNP	1200	55.0%	5421
AB TT KNP	1210	55.5%	5380

Tabela 3 prikazuje uspešnost našega programa glede na čas, ki ga je dobil za reševanje pozicij. Čim več časa smo dodelili programu, tem večja je bila njegova uspešnost. Tako je npr. uspešnost v primerjavi med 1 sekundo in 10 sekundami narasla za 16 odstotkov.

Tabela 3: Časovna uspešnost

Table 3. Time efficiency

AB TT KNP	WAC	ECM	WCSAC	Σ
1 sek./poz.	65.0%	23.8%	60.6%	43.8%
2 sek./poz.	71.3%	28.0%	64.7%	50.8%
5 sek./poz.	76.3%	33.3%	68.7%	55.5%
10 sek./poz.	80.7%	37.4%	73.1%	59.8%

Na podlagi testnih knjižnic BT2630 in LCT II smo ugotovili tudi relativno moč programa (rating). Rezultate prikazuje tabela 4. Iz rezultatov vidimo, da je moč programa približno 2000 točk ELO.

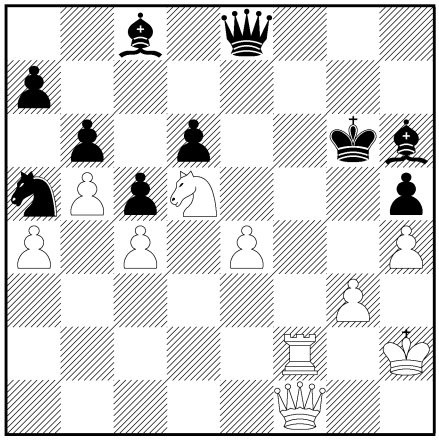
Tabela 4: Relativna moč programa

Table 4. Relative efficiency of the program

Knjižnica	Rezultat	ELO	Čas/poz.	Uspešnost
BT2630	12/30	2006	15 min.	40%
LCT II	7/ 35	2020	10 min.	20%

Podrobneje si poglejmo še analizo dveh rešenih

pozicij iz knjižnice BT2630. Prva pozicija je iz partije Kasparovo - Karpov, ki se je odigrala 1978 leta (pozicija 1). Najboljša poteza je 1. e4-e5, čeprav je kmet na e5 nezaščiten. Na potezo 1. .. De8xe5 sledi 2. Tf2e2. Zaradi šaha s Sd5-e7+ in jemanja lovca na c8, sledi 2. .. De5-h8 3. Df1-b1 Lc8-f5 4. Te2-e6 Kg6-g7 5. Db1xf5 in beli zmaga. Druga možnost je 1. .. d6xe5 na kar sledi 2. Tf2-f6+ Kg6-g7 3. Tf6-d6 De8-f7 4. Df1xf7 Kg7xf7 5. Td6xh6 in beli zmaga. Tretja možnost je 1. .. Lh6-g7 na kar sledi 2. e5xd6 Sa5-b7 3. Rf2-e2 De8-d7 4. Te5-e6 Dd7xe6 5. Sd5-d4 Kg6-h7 6. Sf4-e6 in zopet beli zmaga. Opisano pozicijo je naš program rešil, ko je dosegel globino iskanja 11 (864 sek.).



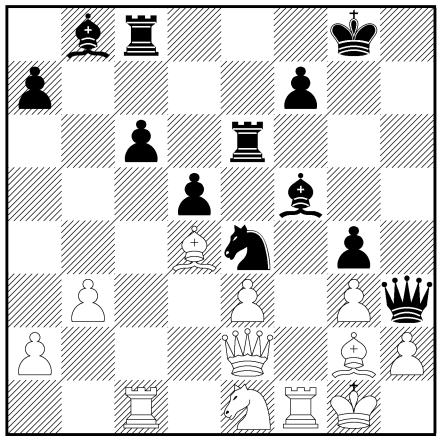
Pozicija 1: BT 15 (Kasparov - Karpov, 1978)

Druga pozicija je iz partije Karpov - Chandler, ki se je odigrala 1983. leta (pozicija 2). Rešitev pozicije je 1. .. Dh3h2. Sledi 2. Kg1h2 Se4xg3 3. De2-b5 Sg3-e2+ 4. Db5xb8 Te6-h6 5. Lg2-h3 Tc8xb8 6. Tf1xf5 Th6xh3+ 7. Kh2-g2 Se2xc1 in črni ima znatno boljšo pozicijo. Opisano pozicijo je naš program rešil, ko je dosegel globino iskanja 11 (545 sek.).

Program smo še dodatno testirali z drugimi šahovskimi programi in amaterskimi igralci šaha. Proti amaterskim igralcem je naš program dosegel približno 80 odstotno uspešnost. V primerjavi s trenutno najboljšimi šahovskimi programi, kot so Fritz, Junior, GNUChess itd., pa nam je z našim programom uspelo doseči remi.

5. Sklep

V prispevku smo predstavili objektno orientirano načrtovanje in implementacijo šahovskega programa. Na podlagi predstavitve igre program uporablja različne iskalne algoritme ter s pomočjo vmesnika UCI komunicira z grafičnim uporabniškim vmesnikom. Predstavitev igre in iskalne algoritme smo načrto-



Pozicija 2: BT 11 (Karpov - Chandler, 1983)

vali in implementirali tako, da omogočajo dodajanje in odstranjevanje določenih sestavnih delov računalniškega šaha in njihovo testiranje. Osnovni prispevki, ki smo jih tako proučili, so:

- objektno orientirano načrtovanje in implementacija šahovskega programa,
- sodobni algoritmi in koncepti računalniškega šaha in
- testiranje šahovskih programov.

Implementiran program smo tudi preizkusili. Ugotovili smo, da je program relativno zmogljiv. Naš program se ne more meriti z velemojstri šahovske igre in profesionalnimi šahovskimi programi, amaterskim šahovskim programom in amaterskim igralcem šaha pa zna povzročiti obilo preglavic.

V nadaljnjem delu bomo našemu programu dodali še podatkovno bazo končnic in algoritme za strojno učenje. S pomočjo teh algoritmov bomo poskušali predstavljen šahovski program učiti na lastnih izkušnjah in ga tako še izboljšati.

6. Literatura

[1] Borko Bošković: Implementacija računalniškega šaha. Diplomsko delo univerzitetnega študija, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, april 2004.

[2] Ernst A. Heinz: Scalable Search in Computer Chess (Algorithmic Enhancements and Experiments at High Search Depths). Friedr. Viewg & Sohn Verlagsgesellschaft mbH, december 1999.

[3] Erest A. Heinz: How DarkThought Plays Chess, Institute for Program Structure and Data Organization (IPD), ICCA Jurnal 20(3), 166-176, September 1997.

[4] Borko Bošković, Janez Brest, Objektno orientirano načrtovanje in implementacija računalniškega šaha Trinajsta mednarodna Elektrotehniška in računalniška konferenca ERK 2004, 27. - 29. september 2004, Portorož, Slovenija.

[5] Aske Plaat: RESEARCH RE:SEARCH & RE–SEARCH, PhD thesis, Erasmus University Rotterdam, 1996.
[6] Yngvi Biörnsson: Selective Depth–First Game Tree –Search, University of Alberta, PhD thesis, Edmonton, Spring 2002.
[7] Omid David Tabibi, Natan S. Netanyahu: Verified Null-Move Pruning, ICGA Journal, Vol. 25, No. 3, pp. 153-161, October 2002.
[8] Breuker D.M., Uiterwijk J.W.H.M. and Herik H.J. van den, Replacement Schemes for Transposition Tables, ICCA Journal, Vol. 17, No.4, pp. 183-193, December 1994.
[9] Ivan Bratko: Prolog in umetna inteligenca, Društvo matematikov, fizikov in astronomov SR Slovenije, Zveza organizacij za tehnično kulturo Slovenije, Ljubljana, 1989.
[10] A short history of computer chess, Frederic Friedel, dostopno na naslovu: http://www.chessbase.com/columns/column.asp?pid=102
[11] Strategy and board game programming, dostopno na naslovu http://www.ics.uci.edu/~eppstein/180a/w99.html
[12] Chess program Gerbil, dostopno na naslovu http://www.seanet.com/~brucemo/gerbil/gerbil.htm
[13] Grafični uporabniški vmesnik Arena, dostopno na naslovu http://www.playwitharena.com/

Borko Bošković je diplomiral leta 2003 na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Od leta 2000 je zaposlen v Laboratoriju za računalniške arhitekture in jezike, kjer se ukvarja s spletnim programiranjem, programskimi jeziki, genetskimi algoritmi in iskalnimi algoritmi s poudarkom na algoritmih za igranje iger med dvema igralcema, popolno informacijo in ničelno vsoto.

Janez Brest je diplomiral leta 1995, magistriral leta 1998 in doktoriral leta 2001 na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Od leta 1994 je zaposlen v Laboratoriju za računalniške arhitekture in jezike, kjer se ukvarja s spletnim programiranjem, s paralelnim in porazdeljenim procesiranjem s poudarkom na razvrščanju opravil. Njegovo področje dela so tudi programski jeziki, ukvarja pa se tudi z optimizacijskimi raziskavami.

Viljem Žumer je redni profesor na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Vodi Inštitut za računalništvo in Laboratorij za računalniške arhitekture in jezike. Področja, s katerimi se ukvarja, so programski jeziki, programiranje ter računalniške arhitekture.