

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Projekt kolegija Ekspertni sustavi

Autori: Vedran Brkić i Željko Kraljević

Zagreb, siječanj 2014.

1. Cilj projekta

Kao zadatak ovog projekta odabrana je implementacija ekspertnog sustava za igranje šaha u programskom jeziku LISP, zasnovanog na heurističkom znanju šahovskih eksperata u kombinaciji sa pretraživanjem mogućih poteza i predikcijom budućih poteza.

2. Uvod

Kod pristupa razvijanju programa za igranje šaha, neophodno je razmotriti problem složenosti šahovske igre. U početku igre svaki igrač može napraviti jedan od 20 mogućih poteza. Nakon dva polupoteza broj mogućih pozicija je 400, nakon četiri polupoteza 197 281, a već nakon deset polupoteza oko 7^{13} . Ukupno je moguće odigrati oko 10^{120} različitih partija šaha. Broj mogućih poteza u tipičnoj poziciji je približno 40. Prema tome, broj mogućih pozicija raste eksponencijalno s bazom 40, pa se pretraživanje svih pozicija pokazuje kao prilično neefikasno rješenje.

Mnogo bolje rješenje je odabir heurističkog pristupa, odnosno definiranje skupa heuristika, zasnovanih na znanju i predstavljenih skupom pravila, koje se dalje koriste pri odabiru poteza.

Osnovna ideja ovog projekta je iskoristiti bazu znanja zasnovanu na heuristikama kao temelj ekspertnog sustava za igranje šaha, provesti pretraživanje određeni broj poteza unaprijed, te kombinacijom pretraživanja poteza i baze znanja napraviti odabir poteza. Pritom pretraživanje ima za cilj minimiziranje mogućih gubitaka u najgorem slučaju.

3. Struktura ekspertnog sustava

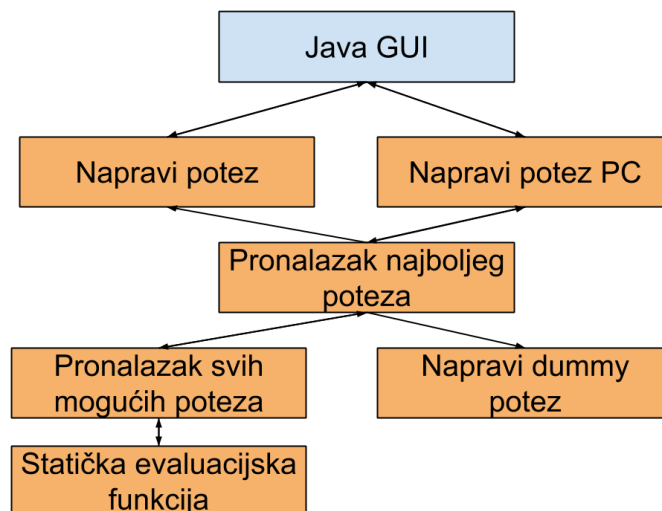
Sustav je u osnovi organiziran integracijom dvije podkomponente:

- Grafički korisnički interfejs u Javi
- Upravljački program u LISP-u

Java GUI služi samo za interakciju sustava s korisnikom i prikaz trenutnog razmještaja figura na ploči, dok su sve ostale funkcije sustava implementirane u programskom jeziku LISP.

Odigravanje poteza obavlja se u nekoliko koraka i ilustrirano je slikom 2. Nakon što čovjek napravi potez, PC pristupa pronalasku najboljeg poteza u nekoliko koraka. Najprije se pronađu svi mogući potezi iz trenutne pozicije. Potom se za svaki od tih poteza odredi

njegova povoljnost te se odabire i izvršava najbolji potez. Povoljnost poteza računa se unutar statičke evaluacijske funkcije.



Slika 2: Struktura ekspertnog sustava

Statička evaluacijska funkcija daje ocjenu trenutnog stanja na šahovskoj ploči i predstavlja osnovu ekspertnog sustava. Ocjena trenutnog stanja zasniva se isključivo na pravilima definiranim na osnovu heurističkog znanja. Drugim riječima, statička evaluacijska funkcija ne uzima u razmatranje moguće buduće poteze. Stoga igrač šaha koji za odabir poteza koristi samo statičku evaluacijsku funkciju može imati dobro pozicionirane figure, ali je istovremeno osjetljiv na upadanje u taktičke zamke (npr. napad na dvije figure veće vrijednosti figurom manje vrijednosti). Iz tog razloga provodi se pretraživanje poteza N koraka unaprijed s ciljem maksimiziranja statičke evaluacijske funkcije u N-tom koraku. Korištena procedura pronalaska najboljeg rješenja bit će detaljnije opisana u poglavlju 5.

4. Baza znanja sustava

Baza znanja sustava struktuirana je kao skup heuristika kojima se definira povoljnost razmještaja figura na šahovskoj ploči, odnosno stanja sustava. Ukupni skup heuristika može se raščlaniti prema značaju na:

- Primarne heuristike
- Sekundarne heuristike

Primarne heuristike važnije su od sekundarnih i imaju veći utjecaj na konačnu ocjenu stanja. U primarna karakteristike spadaju:

- Ukupna vrijednost preostalih figura

- Kontroliranost polja
- Zaštićenost i napadnutost figura
- Kontroliranost središnjih polja pijunima

Ukupna vrijednost preostalih figura je heuristika koja ima najveći udio u konačnoj ocjeni stanja sustava. Svakoj figuri pridružena je vrijednost proporcionalna njenom značaju prema tablici 1. Pritom su vrijednosti figura kojima igra ekspertni sustav uzete s pozitivnim predznakom, dok su vrijednosti figura kojima igra čovjek uzete s negativnim predznakom. Drugim riječima, od ukupne vrijednosti figura kojima raspolaže ekspertni sustav, oduzima se ukupna vrijednost figura kojima raspolaže čovjek.

Tablica 1: Vrijednosti pridružene figurama

Pijun	Konj	Lovac	Top	Kraljica	Kralj
1	3	3	5	9	25

Za svako polje na šahovskoj ploči određuje se koliko puta je polje napadnuto od strane figura oba igrača te se vrijednost proporcionalna razlici broja napada od figura sustava i broja napada od figura čovjeka dodaje ukupnoj ocjeni posmatranog stanja. Pretpostavimo da je polje F8 napadnuto je od strane bijelog igrača jednom figurom (kraljica) a od strane crnog s dvije figure (kraljica i top). Ako pretpostavimo da bijelim figurama igra ekspertni sustav, tada će primjena heuristike o kontroliranosti polja posmatrano polje rezultirati smanjenjem ukupne ocjene stanja za vrijednost $w*(-1)$, gdje je w težinski faktor heuristike.

Za svaku figuru igrača A na ploči, ukoliko je polje, prema prethodno objašnjenju heuristiki, kontrolirano figurama igrača B, i obrnuto, na ukupnu ocjenu stanja dodaje se vrijednost proporcionalna vrijednosti figure s nekim težinskim faktorom.

Korisnost pijuna koji se nalaze na nekom od središnjih polja veća je od njihove korisnosti na inicijalnim pozicijama. Iz tog razloga, ako se pijun nalazi na nekom od središnjih 16 polja, ukupnoj ocjeni dodaje se vrijednost proporcionalna vrijednosti pijuna. Pritom se još veća vrijednost dodaje kada je pijun pozicioniran na neko od četiri centralna polja.

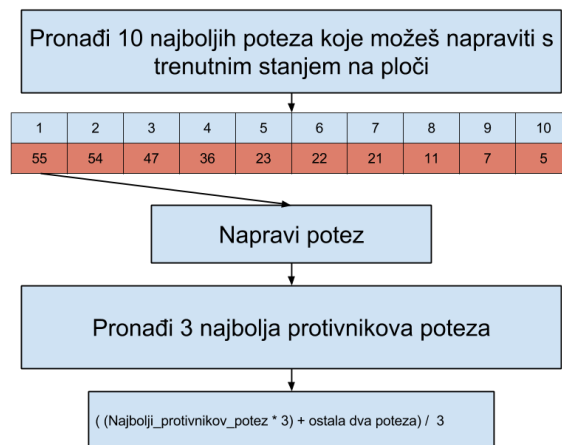
Sekundarne heuristike: za razliku od primarnih, utjecaj sekundarnih heuristika uglavnom je ograničen na početnu i finalnu fazu igre. Implementirane sekundarne heuristike uključuju:

- Poredak pijuna – izbjegavanje dupliranja
- Izbjegavanje pomicanja kralja i kraljice u početnoj fazi igre

- Napadnutost jačih figura slabijima
- Potenciranje razmjene figura istih vrijednosti u vodstvu
- ...

5. Pronalaženje najboljeg poteza

Algoritam pronalaženja najboljeg poteza varijacija je min-max algoritma odlučivanja. Min-max algoritam ima za cilj maksimiziranje dobiti uz minimalni gubitak. U kontekstu šahovske igre, algoritam gleda N poteza unaprijed, određuje stanje sustava korištenjem statičke evaluacijske funkcije u svakom koraku te izabire potez koji rezultira maksimalnom vrijednošću statičke evaluacijske funkcije na dubini N.



Slika 1: Blok dijagram pronalaženja najboljeg poteza

6. Programska implementacija

Kao što je već napomenuto, upravljački program ekspertnog sustava implementiran je u programskom jeziku LISP. Ovdje će biti navedene neke osnovne funkcije korištene pri implementaciji zajedno s objašnjenjem njihove uloge.

- *(posib)* – funkcija bez ulaznih parametara koja vraća sve moguće poteze iz trenutnog stanja na ploči
- *(value toSq)* – statička evaluacijska funkcija. Prima indeks polja na koje je došla figura u prethodnom koraku. Funkcija određuje svojevrsnu dobrotu trenutne stanja na šahovskoj ploči upotrebom niza heuristika. Vraća pozitivnu vrijednost ako je u prednosti ekspertni sustav, ili negativnu kada je u prednosti protivnički igrač.
- *(min-max depth firstLevel whoPlays)* – implementacija algoritma pronalazanja najboljeg poteza.
 - o *Depth* – broj poteza koje sustav gleda unaprijed.
 - o *FirstLevel* – parametar koji poprima vrijednost 1 ako se funkcija min-max poziva izravno.

- o *WhoPlays* – oznacava tko prvi igra. 1 – Čovjek, 0 – PC,
- *(mover)* – Funkcija koju koristi min-max funkcija prilikom pretraživanja. Omogućava izračunavanje vrijednosti statičke evaluacijske funkcije za moguće buduće pozicije bez utjecaja na stvarno stanje na šahovskoj ploči. Vraca listu najboljih poteza za trenutnog igrača.
- *(makeMove bestMove)* – Funkcija koju min-max koristi za stvarno odigravanje poteza. Kao argument prima najbolji potez određen min-max funkcijom kao listu s dva člana gdje je prvi početno a drugi određeno polje.
- *(makeMoveHuman move)* – Odigravanje poteza protivničkog igrača - čovjeka. Ulazni parametar identične je strukture kao kod *makeMove* funkcije.

7. Upute za pokretanje programa

Da bi mogli pokrenuti program potrebno je instalirati Allegro Common Lisp express verziju (možete je naći na <http://www.franz.com/downloads.lhtml>). U terminalu se pozicionirati u direktoriji `acl90express` te upisati:

```
./alisp -L <putanja do lispfunctions2.lisp>
```

Nakon toga otvara se ACL te se učitaje *lispfunctions* datoteka, za pokretanje igrača saha potrebno je upisati `(initChess)` nakon toga lisp čeka na konekciju od Java GUI-a.

Zadnji korak je pozicionirati se u direktoriji gdje se nalazi `chess.jar` datoteka te izvršiti naredbu:

```
java -jar chess.jar
```

GUI se nakon toga spaja na LISP te igra može početi.

8. Zaključak

Na kraju možemo reći da saha uopće nije lako implementirati ne samo u LISP-u već i u bilo kojem drugom programskom jeziku. Možda je bolje reći da je LISP u ovom primjeru bio mnogo zahvalniji od većine drugih programskih jezika.

Jedan od uzroka težine ovoga zadatka jest kompleksnost same igre. Izrazito je teško odrediti gdje se dogodila greška u programu ako je broj poteza koje je program napravio veći od 10^{11} . Također u većini slučajeva teško je reći je li dodana heuristika bila korisna ili ne, jer u

određenim situacijama izgleda da je vrlo dobra dok u drugim ne. Opcenito samo testiranje sustava predstavlja problem zbog vremena potrebnog da se utvrdi da li je dobiveni izlaz dobar ili posljedica slučajnosti.

Ali barem možemo reći da smo naučili neke osnove LISP-a i izgradnje ekspertnih sustava.