

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

# GomokuChump

*Mirko kokot*

Zagreb, siječanj 2017.

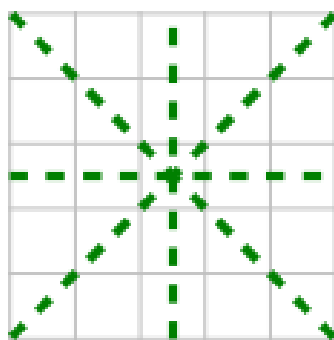
# SADRŽAJ

<b>1. Uvod</b>	<b>1</b>
<b>2. Baza znanja</b>	<b>2</b>
<b>3. Programsko rješenje</b>	<b>4</b>
3.1. Grafičko sučelje . . . . .	4
3.2. Implementacija rješenja . . . . .	5
3.3. Tijek igre . . . . .	6
<b>4. Zaključak</b>	<b>8</b>

# 1. Uvod

Cilj ovog projekta je implementacija ekspertnog sustava za igranje igre Gomoku.

Gomoku je apstraktna strateška igra za dva igrača na ploči. Cilj igre je napraviti niz od točno pet vlastitih figura (odakle i popularni naziv "Pet u nizu"). Dozvoljene orijentacije su vodoravno, okomito te dijagonale (slika 1.1). Matematički gledano, igra bez dodatnih ograničenja nije balansirana te je igrač koji je prvi na potezu u prednosti te postoji pobjednička strategija. Pravila implementirana u ovom radu su minimalna te igrač uvijek prvi započinje na praznoj ploči.



**Slika 1.1:** Dozvoljene orijentacije niza

Kako bi programsko rješenje pružilo izazov igraču, iskorišten je heuristički ekspertni sustav pomoću kojeg se odlučuje slijedeći potez računala.

## 2. Baza znanja

Program koristi bazu znanja u obliku YAML datoteke. Struktura datoteke je sljedeća:

*opis grupe 1 :*

points : *vrijednost koju pridonosi maska iz ovog skupa*

length : *veličina maske*

moves :

[ *maska poteza 1* ]

[ *maska poteza 2* ]

...

*opis grupe 2 :*

...

...

*opis grupe n :*

...

U bazi zadaju se grupe maski kombinacija niza te njihove vrijednosti kod odabira najboljeg poteza. Kod definiranja maske zadaju se ključevi za očekivanu vrijednost pozicije. Opisi ključeva su sljedeći:

– slobodna polja:

- -3 - polovica vrijednosti
- 0 - normalna vrijednost
- 3 - dvostruka vrijednost

– zauzetost polja:

- -1 - polje nesmije imati figuru
- 1 - polje mora imati figuru

Kada se razmatra pojedina maska, gleda se i njezina zrcalna slika. Pravila ključeva zauzetih polja odnose se samo na figure jednog igrača. Ukoliko je polje zauzeto suprotnom figurom (ili u slučaju *-1* istom), maska se automatski odbacuje.

Za kvalitetno rasuđivanje potrebno je postaviti relevantne i razmjerne vrijednosti pojedinih grupa maski.

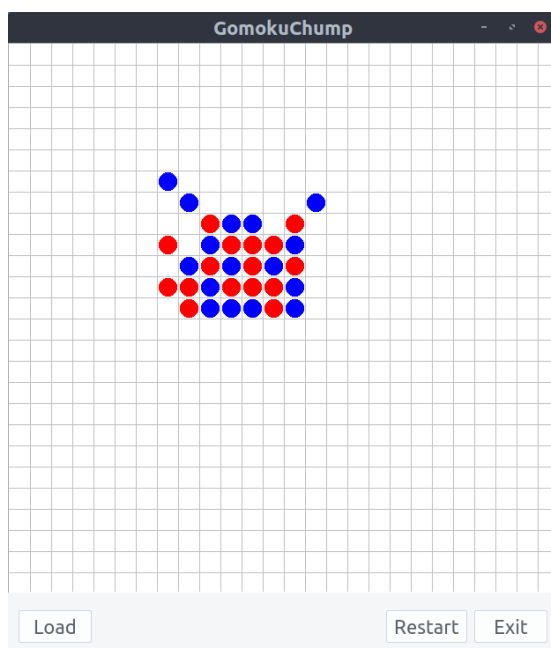
Na slici 2.1 prikazan je primjer dvije strukture gdje se može primjetiti različita veličina i broj maski. Također kod samih maski može se vidjeti kako nisu zadane zrcalne slike istih.

```
attack :
  points : 5
  lenght : 5
  moves :
    - [ 1, 1, 1, 0, 0 ]
    - [ 0, 1, 1, 1, 0 ]
    - [ 1, 1, 0, 1, 0 ]
    - [ 1, 1, 0, 0, 1 ]
    - [ 0, 1, 1, 0, 1 ]
    - [ 1, 0, 1, 0, 1 ]
ok :
  points : 3
  lenght : 6
  moves :
    - [ -3, 1, 1, 0, 0, -3 ]
    - [ -3, 1, 0, 1, 0, -3 ]
    - [ -3, 0, 1, 1, 0, -3 ]
```

Slika 2.1: Primjer strukture za dvije vrste poteza

## 3. Programsko rješenje

Programsko rješenje izrađeno je u programskom jeziku *C++* u *Qt framework* pomoću kojega je izrađeno i jednostavno grafičko sučelje (slika 3.1).



Slika 3.1: Grafičko sučelje

### 3.1. Grafičko sučelje

Kod pokretanja programa potrebno je učitati bazu znanja nakon čega automatski počinje igra. Tokom igre, ili po njezinom završetku, moguće je pokrenuti novu partiju pritiskom na tipku *Restart*, izaći iz programa sa tipkom *Exit* i zamijeniti učitane bazu znanja pritiskom tipke *Load* te odabirom datoteke baze prilikom čega se pokreće nova partija.

## 3.2. Implementacija rješenja

Unutar programa definirana je klasa *playingBoard* unutar koje se implementiranim metodama prati trenutno stanje na igračkoj ploči. Za lakšu implementaciju klase definirane su slijedeće strukture:

*boardState* - za praćenje stanja je li igra završena

- **bool** *win* - je li igra završena
- **pair<int, int>** *p1* - početna točka pobjedničkog niza
- **pair<int, int>** *p2* - završna točka pobjedničkog niza

*maskIndex* - za spremanje indeksa različitih ključeva unutar maske

- **vector<int>** *pos* - indeksi ključa 1
- **vector<int>** *neg* - indeksi ključa -1
- **vector<int>** *normal* - indeksi ključa 0
- **vector<int>** *extra* - indeksi ključa 3
- **vector<int>** *half* - indeksi ključa -3

*fieldWeight* - za spremanje učitanih maski

- **int** *size* - veličina maske
- **int** *points* - vrijednost maske
- **maskIndex** *index* - indeksi ključeva
- **vector<int>** *mask* - zapis maske

Implementirane metode klase *playingBoard*:

*private:*

- **bool** *checkForWin(int x, int y)* - provjerava okolinu polja (*x*, *y*) je li ostvaren uvjet pobjede. Vraća *true* ukoliko je pronađen niz od pet figura.
- **void** *recalculate(int x, int y, int m)* - računa novu težinsku vrijednost polja (*x*, *y*) u smjeru *m*

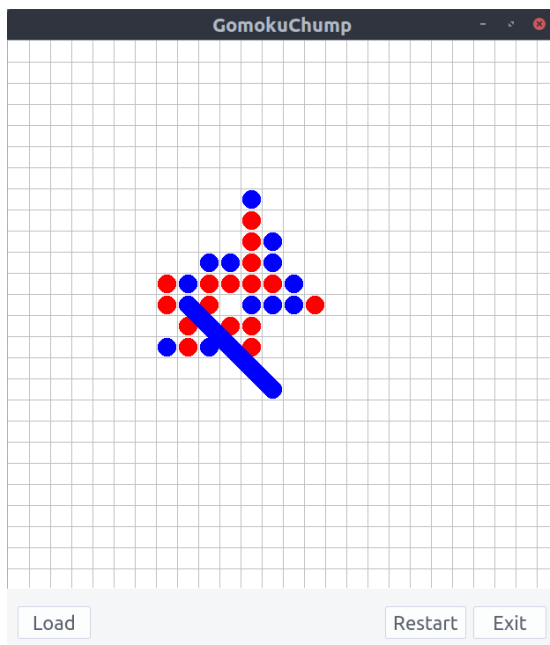
*public:*

- **bool** *set(int x, int y, int who)* - postavlja figuru igrača *who* na polje (*x*, *y*) te poziva metode *checkForWin* i *recalculate*. Vraća *true* ako je uspješno postavio figuru.
- **int** *get(int x, int y)* - vraća vrijednost trenutne figure na polju (*x*, *y*)
- **void** *doChumpsTurn()* - traži najpovoljniji potez te poziva metodu *set*

### 3.3. Tijek igre

U početnom stanju svako polje igrača ploče ima dodijeljene četiri težinske vrijednosti za svaki od smjerova te su oni inicijalizirani na vrijednost 0. Kao što je navedeno u uvodu, igrač je uvijek prvi na potezu te su njegove figure kružići plave boje, dok su figure s kojima igra računalno crvene boje. Klikom miša na kvadratić u grafičkom sučelju igrač odabire željeno polje.

Kod postavljanja figura jednog od igrača, prvo se provjerava postoji li niz od pet istih figura oko odigranog polja te u slučaju istog igra završava te se na sučelju prikazuje pobjednički niz (slika 3.2). Ukoliko potez nije rezultirao pobjedom, ažuriraju se težinske vrijednosti polja.



**Slika 3.2:** Primjer prikaza završetka igre

Radi optimizacije programa, ažuriraju se samo polja na koje je prethodni potez mogao utjecati. To su slobodna polja u smjerovima definiranim slikom 1.1 do najveće udaljenosti jednake maksimalnoj vrijednosti dužine maske definirane u bazi znanja. Za svako od odabranih polja, ažurira se težinska vrijednost poteza samo za smjer u kojem je prisutno polje koje je prethodno jedan od igrača odigrao.

Nova težinka vrijednost odabire se kao maksimalna vrijednost iz baze koja se može dodijeliti za neko od slobodnih polja iz baze znanja. Maksimalna vrijednost se određuje principom stabla odluke gdje se ispituje može li se primijeniti maska najveće vrijednosti, te u nepovoljnom slučaju ispituje se slijedeća maska. Za svaku od maski također vrijedi princip stabla odluke gdje se prvo ispituje može li se iskoristiti slobodno



polje maske sa ključem veće vrijednosti (opisi ključeva u poglavlju 2). Težinska vrijednost ne ovisi o vrsti figura u nizu jer je tokom igre jednako važno blokirati uspješne protivnikove napade kao i stvarati vlastite šanse za pobjedu.

Nakon igračevog poteza na redu je računalo. Za određivanje slijedećeg poteza računala traži se slobodno polje sa najvećim zbrojem sve četiri težinske vrijednosti, što rezultira najboljim trenutnim potezom za ometanje protivnika i stvaranje vlastite prilike.

Igrač i računalo se naizmjenice izmjenjuju na potezu sve dok jedan od poteza ne rezultira nizom od pet figura.

## 4. Zaključak

Programsko rješenje GomokuChump suprotstavlja se ljudskom igraču odabirom svojih poteza pomoću unaprijed poznate baze znanja. Kvaliteta i sposobnost ovisi o učitanoj bazi znanja, ali je već iz primjera priloženog uz program vidljivo kako je i relativno mala baza dovoljna da pruži kvalitetni izazov. Testirajući priloženu bazu sa gledanjem par poteza unaprijed, program postaje skoro nepobjediv (teoretski je moguća pobjeda, ali kod testiranja je računalo uvijek pobijedilo). Iz tog razloga kako bi igra zadržala svoju svrhu, zabaviti igrača, odabran je model koji gleda samo trenutno stanje čime ga je ponekad moguće nadmudriti.