Beskonfliktni raspored figura na šahovskoj ploči u programskom jeziku C++.

Rajšić, Robin

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

Beskonfliktni raspored figura na šahovskoj ploči u programskom jeziku C++

Završni rad

Robin Rajšić

Osijek, 2021 godina.

SADRŽAJ

1. UVOD

Problemi beskonfliktnih prikaza figura na šahovskoj ploči su stari koliko i sama igra, ali ih je popularizirao sastavljač šahovskih problema Max Bezzel[1] kada je objavio zagonetku „osam dama“ 1848. god. Kasnije se ta zagonetka proširila na „*n* dama“ čije algoritme za rješavanje su objavljivali slavni matematičari poput Carla Gaussa[2] i Siegmunda Günthera[3]. 1972. god. Edsger Dijkstra je koristio upravo ovaj problem za ilustraciju strukturalnog programiranja. Problem i rješenje koje će biti opisani u ovom završnom radu su varijacija osnovnog problema sa *n* dama. Dodatno će korisnik moći unijeti *n* figura gdje je $n < 10$, a figure su kralj, dama, skakač, lovac i top. Za rješavanje problema korist će se algoritam „sirove snage“ (engl. „*brute force*“) koji će prolaziti redom od prvog do zadnjeg polja te pokušavati postaviti sve figure na sva polja. Za unos broja figura i veličinu šahovske ploče koristiti će se grafičko korisničko sučelje napisano u programskom jeziku *pythonu* te za sami prikaz mogućih rješenja koristiti će se grafičko sučelje napisano u programskom jeziku C++-u.

U trećem poglavlju dan je opis i definicija šaha te izgled šahovske ploče i šahovskih figura.

U četvrtom poglavlju opisano je razvojno okruženje pomoću kojeg je izrađen program.

U petom poglavlju detaljno je opisana implementacija programa te primjeri i analiza rješenja.

Commented [ČL1]: (engl. „*brute force*“)

Commented [ČL2]: Nepotrebno ovako odvojiti

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je pisanje programa koji korisniku nudi mogućnost unosa broja različitih šahovskih figura i veličinu šahovske ploče te na temelju unosa izračunati i prikazati jedan ili više beskonfliktnih rasporeda figura na šahovskoj ploči ovisno o gumbu koji je korisnik pritisnuo.

[1] Max Bezzel, 1824 – 1871, Njemački sastavljač šahovskih problema
[2] Carl Friedrich Gauss, 1777 – 1855, Njemački matematičar i fizičar
[3] Siegmund Günther, 1848 – 1923, Njemački geograf i matematičar

2. Šahovski problem „*n* dama“

Šahovski problem „*n* dama“ i danas je aktualan zbog svoje matematičke kompleksnosti. Kompleksnost je dijelom zbog ne simetrične strukture šahovske ploče. Ako stavite kraljicu oko centra šahovske ploče ona napada 27 polja, a ako ju stavite uz rub onda napada 21 polje. Matematička kompleksnost se smanjuje ako se koristi ploča u obliku torusa u kojoj dijagonale omotavaju ploču s lijeva prema desno te odozgora prema dolje. U članku „*A lower bound for the n-queens problem*“ autori članka Zur Luria i Michael Simkin koriste nasumični pohlepni algoritam (eng. „*random greedy algorithm*“) za postavljanje kraljica na toroidalnu ploču te brojanjem dostupnih izbora prilikom svakog koraka nasumičnog pohlepnog algoritma dobije se minimalni broj ukupnih rješenja [1]. U radu „*The n-queens problem*“ autori Candida Bowtell i Peter Keevash rješenje su opisali također sa nasumičnim pohlepnim algoritmom [2]. Za rješavanje problema ovog završnog rada koristiti će se algoritam vraćanja unatrag (engl. „*backtracking algorithm*“) u kojem će se postaviti prva figura u prvi lijevi stupac te druga figura u slijedeće slobodno mjesto s lijeva prema desno i tako dalje. U slučaju da do~~Ovaj rad je proširenje temeljnog šahovskog problema „*n* dama“. Problem „*n* dama“ opisan je u OEIS-u (*On-Line Encyclopedia of Integer Sequences*). OEIS je baza podataka na internetu u kojoj su spremljene sekvence cijelih brojeva. Napravio i održavao ju je Neil Sloane[4] dok je radio za AT&T[5]. Sloane je predsjednik OEIS zaklade [1]. OEIS danas sadrži 341,962 sekvenci. U OEIS bazi podataka opisana su dva rješenja za rješavanje problema „*n* dama“. Osnovna rješenja opisana su u sekvenci A002562 [2] opisuje rješenje problema u kojem se izbacuju duplicirana rješenja nakon što se izvrši refleksija i rotacije ploče. Sva rješenja opisana su u sekvenci A000170 [3]. Program ovog završnog rada nudit će mogućnost prikaza osnovnih i svih rješenja te će nuditi mogućnost odabira više šahovskih figura sa različitim veličinama šahovske ploče.~~

Commented [ČL3]:
Iz uputa:

Pregled područja teme (DRUGO POGLAVLJE)

Poglavlje može imati proizvoljan naziv (ali se mora uklapati u strukturu i nazivlje samog rada). U navedenom poglavlju student mora opisati aktualne znanstvene i/ili praktične dosege u području rada koji se obrađuje (engl. State of the Art) i potkrijepiti ga referencama (preporuča se 5 ili više). Primjerice, ako student koristi određenu metodu/metode, opisati što se inače koristi u svrhu(e) u koju se koristi i za studentov rad. Ako takve metode ne postoje, pregledom literature obrazložiti i potkrijepiti studentove tvrdnje i navesti i obrazložiti koje su najsličnije metode/postupci rješavanja problema koji se rješava u radu. U slučaju da se radi o, primjerice, računalnoj aplikaciji, tada usporediti s aplikacijama iste ili slične svrhe, a ne pisati o dosezima tehnologija za izradu te aplikacije.

Formatted: Font: Do not check spelling or grammar

Formatted: Quote Char

Formatted: Font: Do not check spelling or grammar

3. ŠAH

Šah je igra na ploči izvorno nastala u sjevernoj Indiji u 6. stoljeću, a potom se proširila u Perziju. Arapsko osvajanje Perzije rezultiralo je širenjem šaha u muslimanskim zemljama, te preko njih i po cijelom svijetu. Tijekom 15. stoljeća promijenili su se pokreti šahovskih figura u Europi, te je s time nastao šah kakvog danas znamo [1].

3.1. Definicija šaha

Šah je igra za dva igrača u kojoj uz početni postav figura i pravila kretanja figura, dva igrača naizmjence pomiču po jednu figuru, a pobjednik je igrač koji prvi postavi svoju figuru na polje na kojemu je protivnički kralj.

3.2. Šahovska ploča

Šahovska ploča za igru šah sastoji se od 32 svijetla i 32 tamna polja, odnosno osam redova i osam stupaca sa naizmjeničnim pojavljivanjem svijetlih i tamnih polja. Pojam veličine šahovske ploče odnositi će se na broj redova i stupaca, te bi za originalnu veličinu šahovske ploče veličina šahovske ploče bila 8×8. Za potrebe ovog rada korisnik programa će moći izabrati sljedeće veličine šahovske ploče: 3×3, 4×4, 5×5, 6×6, 7×7, 8×8. Za crtanje šahovskih ploča koristit ćemo Windows-ov program *Paint3D.* Boja tamnog polja je *RGB(212,140,68),* a njena vrijednost u heksadekadskom obliku je: *#D48C44*. Na slici 3.1. je prikazano tamno polje.

Sl. 3.1. Prikaz tamnog polja

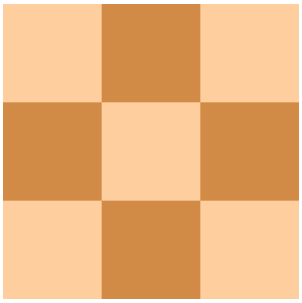
Boja svijetlog polja je *RGB(252,204,156)*, te njena vrijednost u heksadekadskom obliku je: *#FCCC9C*. Na slici 3.2. je prikazano svijetlo polje.

Sl. 3.2. Prikaz svijetlog polja

Boje su dobivene iz službenog predloška [7] ~~[2]~~. Slika 3.3. prikazuje primjer šahovske ploče veličine 3×3.

Field Code Changed

Formatted: Font: Do not check spelling or grammar



Sl. 3.3. Šahovska ploča veličine 3×3

3.3. Izgled figura

Za prikaz šahovskih figura koristit će se crne figure uređenog predloška besplatnog za korištenje [8] ~~[3]~~. Uređivanje službenog predloška se mora napraviti da bi se izbacila slika figure pješaka i na mjestu te slike se stavlja slika slova „*x*“ koja će označavati zauzeto polje. Zauzeto polje odnosi se na ono polje na koje je moguće pomaknuti šahovsku figuru. Slika 3.4. prikazuje po redu slike: zauzetog polja, kralja, kraljice, lovca, skakača i topa. Istim redoslijedom u kodu će se brojevima od 1 do 6 referirati na slike figura.

Field Code Changed

Formatted: Font: Do not check spelling or grammar

Sl. 3.4. Izgled šahovskih figura[6]

Commented [ČL4]: Ukoliko figurice nisu autorsko djelo, potrebno je navesti izvor.

4. RAZVOJNO OKRUŽENJE

Za programiranje glavnog algoritma za traženje mogućih prikaza beskonfliktnih prikaza šahovskih figura na ploči koristit će se programski jezik C++, te biblioteka SFML [9] ~~[4]~~ za grafički prikaz beskonfliktnih rasporeda. Za unos veličine šahovske ploče i broja figura koristit će se programski jezik *python* [10] ~~[5]~~ s programskim sučeljem *tkinter* [11] ~~[6]~~ za grafičko korisničko sučelje.

Field Code Changed
Formatted: Font: Do not check spelling or grammar
Field Code Changed
Formatted: Font: Do not check spelling or grammar
Field Code Changed
Formatted: Font: Do not check spelling or grammar

4.1. Programski jezik C++

Programski jezik C++ je objektno orijentirani programski jezik kojeg je razvio Bjasne Stroustrup 1985. god. te je ubrzo postao najkorišteniji programski jezik. C++ je prvobitno razvijen kao ekstenzija programskoga jezika C da bi se ispunili nedostaci koje C ima, poput objektno orijentiranog programiranja. Iako je prvi put objavljen prije gotovo 36 godina C++ se još uvijek razvija te se redovno izdaju nadogradnje. Zbog svoje mogućnosti rukovođenjem memorijom na niskoj razini programski jezik C++ se danas koristi u područjima u kojima je potrebna kontrola nad procesorom i ostalim sklopovljem, poput razvijanja raznoraznih softverskih alata, razvijanja video igara te razvijanje svih sustava u kojima je brzina izvođenja bitna [12] ~~[7]~~. Programski jezik C++ će se u ovom radu koristiti za računanje beskonfliktnih rasporeda figura, a za prikaz istih će se koristiti SFML programska biblioteka. SFML je biblioteka za C++ koja pruža jednostavno sučelje raznim komponentama osobnog računala, te služi za jednostavan razvoj igara i multimedijskih aplikacija. Sastavljena je od pet modula: *system, window, graphics, audio* i *network* od kojih će se za potrebe rada programa koristiti *window* i *graphics* moduli [9] ~~[4]~~.

Field Code Changed
Formatted: Font: Do not check spelling or grammar
Field Code Changed
Formatted: Font: Do not check spelling or grammar

[6] Izgled šahovskih figura je izmijenjena verzija slike sa wikipedije besplatne za korištenje [22]

Formatted: Font: Do not check spelling or grammar

4.2. Programski jezik python

Za razliku od programskog jezika C++ koji koristi kompajler (engl. *compiler*), program koji prevede cijeli programski kod odjednom i onda ga tek izvršava, programski jezik Python je jezik koji prilikom izvršenja izvodi *interpreter*, program koji prevodi i odmah izvršava liniju po liniju programa. Python je programski jezik visoke razine te služi za pisanje kompliciranih algoritama u kojima nije bitna brzina izvođenja kao ni rukovođenje memorijom [8]. U ovom radu za prikaz grafičkog korisničkog sučelja za unos broja figura i veličinu ploče koristit će se trenutna verzija python-a, a to je *3.8.6*. uz pomoć Tkinter programskog sučelja. Tkinter je standardno python-ovo sučelje za razvoj grafičkih aplikacija [11] ~~[6]~~. Elementi koji će se upotrebljavati za izradu grafičkog korisničkog sučelja iz tkinter-a su *Font, Label, PhotoImage, OptionMenu, Canvas, Frame* i *Button.*

Field Code Changed

Formatted: Font: Do not check spelling or grammar

5. IZRADA PROGRAMA

5.1. Izrada grafičkog sučelja za unos i ispis rezultata u python-u

Za slaganje elemenata po glavnom prozoru koristit će se *grid* geometrijski rukovoditelj elemenata programskog sučelja tkinter [9]. Sučelje će nuditi tri gumba, prvi za ispis svih mogućih prikaza beskonfliktnih rasporeda figura, drugi za ispis osnovnih prikaza beskonfliktnih rasporeda figura i treći će prikazati prvi mogući prikaz beskonfliktnog rasporeda figura.

5.1.1. Izrada glavnog prozora

Izrada grafičkog sučelja započinje se s izradom glavnog prozora s imenom „*Chessboard Menu*“. Glavni prozor se izrađuje sa definiranjem varijable koja postavlja s Tkinter-ovim konstruktorom. Nakon definicije varijable, postavlja se ime prozora i zabranjuje se mijenjati veličinu prozora. Izrada glavnog prozora je prikazana u kodu (Sl. 5.1.) ~~(Sl. 5.1.)~~ ~~(Kod 5.1.).~~

```
13   window = Tk()
14   window.title("Chessboard Layout Menu")
15   window.resizable(width = FALSE, height = FALSE)
```

(~~Kod~~ Sl. 5.1.) Isječak kôda izrade glavnog prozora

Commented [ČL5]: Ovo je i dalje slika pa treba biti:

Sl. 5.1. Isječak kôda izrade glavnog prozora

Za sve kodove (isječke koda) dalje u nastavku potrebno je ovo promijeniti.

5.1.2. Učitavanje slika

Slike figura su pojedinačno izrezane iz slike izgleda šahovskih figura prikazanih u slici 3.4. i spremljene. Slike pojedinačnih figura se učitavaju prema kodu (Sl. 5.2.)(~~Sl. 5.2.~~)(~~Kod 5.2.~~).

```
52  queen_img = ImageTk.PhotoImage(Image.open("images/pieces/queen.png"))
53  king_img = ImageTk.PhotoImage(Image.open("images/pieces/king.png"))
54  rook_img = ImageTk.PhotoImage(Image.open("images/pieces/rook.png"))
55  bishop_img = ImageTk.PhotoImage(Image.open("images/pieces/bishop.png"))
56  knight_img = ImageTk.PhotoImage(Image.open("images/pieces/knight.png"))
```

(~~Kod~~ Sl. 5.2.) Isječak kôda učitavanja slika

5.1.3. Definiranje fontova

~~Zatim se definiraju pisma koji će se koristiti u grafičkom sučelju.~~ Pismo koje će se koristiti je google-ovo pismo *Roboto* [15] ~~[9]~~ koje je javno dostupno i besplatno za korištenje. Pismo je potrebno učitati za svaku veličinu koja će se koristiti. Pismo se učitava prema kodu (Sl. 5.3.)(~~Sl. 5.3.~~)(~~Kod 5.3.~~).

Field Code Changed

Formatted: Font: Do not check spelling or grammar

```
35  roboto16=tkFont.Font(family="Roboto", size=16)
36  roboto12=tkFont.Font(family="Roboto",size=12)
```

(~~Kod~~ Sl. 5.3.) Isječak kôda učitavanja pisma

5.1.4. Unos veličine šahovske ploče

Prema kodu (Sl. 5.4.)(~~Sl. 5.4.~~)(~~Kod 5.4.~~) definira se niz koji sadrži veličine šahovskih ploča definiranih u poglavlju 3.2.

```
9   BOARD_SIZE_OPTIONS = ["3x3", "4x4", "5x5", "6x6", "7x7", "8x8"]
```

(~~Kod~~ Sl. 5.4.) Isječak kôda deklaracije veličina ploča

Etiketa za unos i padajući izbornik unosa veličina šahovskih ploča postavljaju se u glavni prozor prema kodu (Sl. 5.5.)(~~Sl. 5.5.~~)(~~Kod 5.5.~~).

```
21  board_size_label = Label(text="Select board size: ",font=roboto16).grid(row=0,column=0)
22  board_size_option = StringVar(window)
23  board_size_option.set(BOARD_SIZE_OPTIONS[0]) # default value
24  board_size_select= OptionMenu(window, board_size_option, *BOARD_SIZE_OPTIONS)
25  board_size_select.grid(row=0,column=1,pady=10)
```

(~~Kod~~ Sl. 5.5.) Isječak kôda izrade etikete za unos i izbornika

gdje je:

- *board_size_label* – etiketa,
- *board_size_option* – padajući izbornik,
- *board_size_select* – varijabla koja će sadržavati izabranu veličinu šahovske ploče

Zadana veličina board_size_select-a je 3~~3~~×3.

5.1.5. Unos broja figure

Prema kodu (Sl. 5.6.)~~(Sl. 5.6.)~~(~~Kod 5.6.~~) definira se niz koji sadrži moguće brojeve pojedinih šahovskih figura.

```
11   PIECE_OPTIONS = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
```

(~~Kod~~ Sl. 5.6.) Isječak kôda deklaracije polja mogućeg broja figura

Prikaz etikete, slike i padajućeg izbornika za figuru radi se kao što je prikazano u primjeru koda (Sl. 5.7.)~~(Sl. 5.7.)~~(~~Kod 5.7.~~). Primjer koda (Sl. 5.7.)~~(Sl. 5.7.)~~(~~Kod 5.7.~~) se odnosi na figuru dame, za ostale figure proces je identičan i ponavlja se za sve ostale figure.

```
41   queen_label=Label(text="Queen",font=roboto16).grid(row=2,column=0)
42   queen_canvas = Canvas(window, width = 106, height = 106)
43   queen_canvas.create_image(60, 50, image=queen_img)
44   queen_canvas.grid(row=4,column=0)
45   queen_count_option = StringVar(window)
46   queen_count_option.set(PIECE_OPTIONS[0]) #default value
47   queen_count_select= OptionMenu(window, queen_count_option, *PIECE_OPTIONS)
48   queen_count_select.grid(row=4,column=1)
49   queen_count_select.config(font=roboto16)
50   queen_count_menu = window.nametowidget(queen_count_select.menuname)
51   queen_count_menu.config(font=roboto12)  # Set the dropdown menu's font
```

(~~Kod~~ Sl. 5.7.) Isječak kôda prikaza etikete za unos i padajućeg izbornika

gdje je:

- *<piece>_label* – etiketa s nazivom figure,
- *<piece>_canvas* – mjesto na koje se učitava slika figure,
- *<piece>_count* – varijabla koja sadrži broj figura,
- *<piece>_count_select* – padajući izbornik za unos broja figura,

Zadana vrijednost varijable *<piece>_count* je nula.

5.1.6. Gumbi

Naziv, boja i veličina gumbova se izrađuje prema kodu (Sl. 5.8.)~~(Sl. 5.8.)~~(~~Kod 5.8.~~).

```
196  button_all_layouts = Button(window, text="View All",height=2,width=12,bg="#d5e4f7",
197  font=roboto16, command=startAllLayouts).grid(row=9,column=5,pady=10)
198  button_fundemental_layouts= Button(window,text="View Some",height=2,width=12,bg="#dcf7d5",
199  font=roboto16,command = startFundementalLayout).grid(row=10,column=5)
200  button_first_layout = Button(window, text="View One",height=2,width=12,bg="#ffcda1",
201  font=roboto16, command=startFirstPossibleLayout).grid(row=11,column=5,pady=10)
```

(~~Kod~~ Sl. 5.8.) Isječak kôda određivanja naziva, boje i veličine gumba

gdje je:

- *button_<command>* – gumb,
- *text* – naziv gumba,
- *command* – funkcija koju gumb poziva

5.1.7. Okvir za ispis

Okvir za ispis se radi prema kodu (Sl. 5.9.)~~(Sl. 5.9.)(Kod 5.9.)~~.

```
output_label = Label(window,text ="Output:",font=roboto16).grid(row=7,column=7)
output_frame = Frame(window,width = 400, height = 200, bg = "#e8e8e8")
output_frame.grid(row=9,column=6,columnspan = 4,rowspan=3,padx=20)
```

(~~Kod~~ Sl. 5.9.) Isječak kôda izrade okvira za ispis

Važno je napomenuti da ako se dogodi greška poput unosa nula figura ili unosa previše figura za šahovsku ploču okvir će poprimiti crvenu boju te će ispisati grešku koja se dogodila. Ako nema greške prilikom izlaska iz prikaza beskonfliktnih rasporeda pritiskom na tipku *enter* okvir će poprimiti zelenu boju te će ispisati ukupan broj prikazanih rasporeda.

5.1.8. Funkcija za pokretanje glavnog programa napisanog u C++-u

Postoje tri zasebne funkcije koju pokreću tri ponuđena gumba. Razlika u funkcijama je u broju algoritma kojeg šalju dok pokreću glavni program. Funkcija *startAllLayouts()* pokreće glavni program s algoritmom broj 1, funkcija *startFundementalLayouts()* pokreće glavni program s algoritmom broj 2 i funkcija *startFirstPossibleLayout()* pokreće glavni program s algoritmom broj 3 kao što je opisano u poglavlju 5.1. Prvo se dohvaćaju brojevi svih figura i veličina ploče prema (Sl. 5.10.)~~(Sl. 5.10.)(Kod 5.10.)~~.

```
    board_size=board_size_option.get()[:1]
    queen_count = queen_count_option.get()
    king_count = king_count_option.get()
    rook_count = rook_count_option.get()
    bishop_count = bishop_count_option.get()
    knight_count = knight_count_option.get()
```

(~~Kod~~ Sl. 5.10.) Isječak kôda dohvaćanja brojeva svih figura i veličina ploče

Zatim se pokreće glavni program napisan u programskom jeziku C++, te se sprema njegov izlaz u varijablu *output*. (Sl. 5.11.)~~(Sl. 5.11.)(Kod 5.11.)~~.

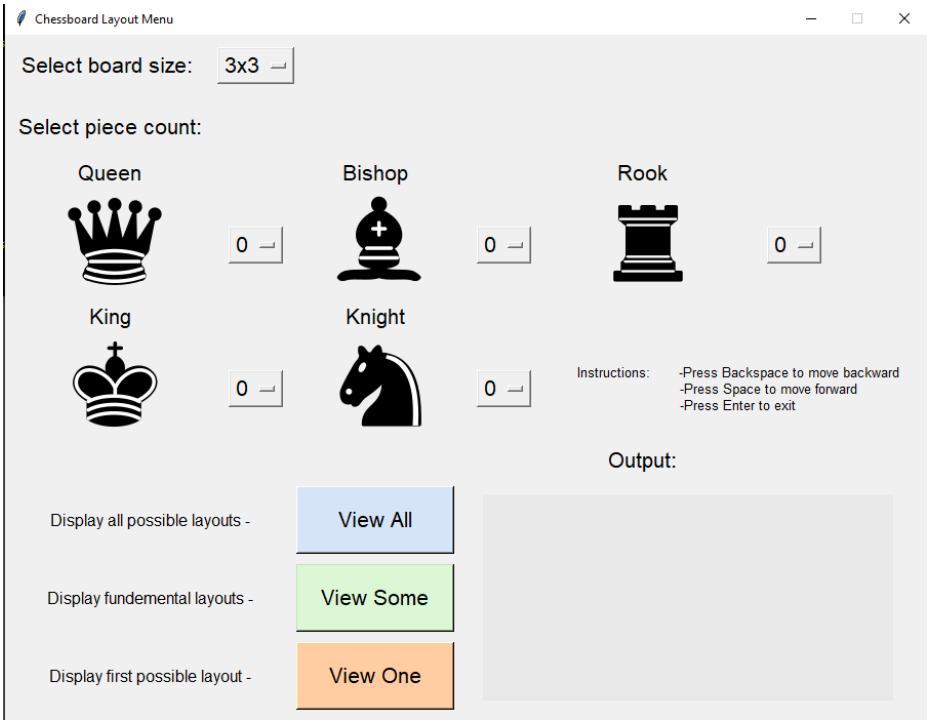
```
        output = os.system("ChessLayoutEngine.exe 3 "+str(board_size)+" "+str(queen_count)+
        " "+str(king_count)+" "+str(rook_count)+" "+str(bishop_count)+" "+str(knight_count))
```

(~~Kod~~ Sl. 5.11.) Isječak kôda pokretanja glavnog programa

Vidljivo je kako se program pokreće s varijablama po redoslijedu: Broj algoritma, veličina ploče, broj dama, broj kraljeva, broj topova, broj lovaca i broj skakača.

5.1.9. Prikaz grafičkog sučelja

Na slici 5.1 ~~je~~ prikazano je dobiveno grafičko korisničko sučelje.



Sl. 5.1. Prikaz grafičkog sučelja

5.2. Izrada grafičkog sučelja za prikaz beskonfliktnog rasporeda figura na šahovskoj ploči u C++-u sa SFML bibliotekom

5.2.1. Programiranje početnog sučelja i popunjavanje elemenata

Programiranje prozora u kojem će biti prikaz rasporeda figura na šahovskoj ploči radi se na prema kodu (Sl. 5.12.)~~(Sl. 5.12.)(Kod 5.12.)~~

```
68      RenderWindow window(VideoMode(g_board_size * size, g_board_size * size), "Chessboard");
```

(~~Kod~~ Sl. 5.12.) Isječak kôda izrade glavnog prozora

gdje je

- *RenderWindow* – klasa koja radi objekt prozor,
- *window* – prozor,
- *g_board_size* – globalna varijabla koja sadrži veličinu šahovske ploče,
- *size* – globalno definirana varijabla koja sadrži veličinu stranice jednog polja u pikselima te iznosi *106.*

-Vidljivo je kako su visina i širina nepromjenjive veličine ali ovise o veličinama šahovske ploče. Dalje se učitavaju slike pomoću klase *Texture* kao što je prikazano u kodu (Sl. 5.13.)(~~Sl. 5.13.~~)(~~Kod 5.13.~~)

```
70      Texture piecesTexture, boardTexture;
71
72      piecesTexture.loadFromFile("images/pieces/chess_pieces.png");
73      boardTexture.loadFromFile(BoardImageFactory::createBoard());
```

(~~Kod~~ Sl. 5.13.) Isječak kôda učitavanja slika

gdje je:

- *piecesTexture* – objekt koji sadrži sliku figura prikazanu na slici 3.4.,
- *boardTexture* – objekt koji sadrži sliku šahovske ploče,
- *BoardImageFactory* – klasa koja vraća sliku šahovske ploče ovisno o odabranoj veličini.

Prema kodu (Sl. 5.14.)(~~Sl. 5.14.~~)(~~Kod 5.14.~~) učitane slike se zatim pretvaraju u programske reprezentacije slika koje se mogu crtati *Sprite* klasom.

```
75      Sprite sAllPieces(piecesTexture);
76      Sprite sBoard(boardTexture);
```

(~~Kod~~ Sl. 5.14.) Isječak kôda pretvaranja slika u programske reprezentacije

Broj spremljenih ploča sprema se u varijablu *board_count* prema kodu (Sl. 5.15.)(~~Sl. 5.15.~~)(~~Kod 5.15.~~). Objekt bEngine i njegovo instanciranje opisani su u kodu (Sl. 5.29.)(~~Sl. 5.29.~~)(~~Kod 5.30.~~).

```
78      int board_count = bEngine.getBoards().size();
```

(~~Kod~~ Sl. 5.15.) Isječak kôda spremanja broja spremljenih ploča

gdje je:

- *bEngine* – objekt koji sadrži spremljene šahovske ploče koje su rješenje za odabrane parametre.

Za učitavanje slika figura i zauzetih polja definira se niz od onoliko elemenata koliko ima spremljenih ploča u kojem će svaka varijabla predstavljati broj figura i zauzetih polja po šahovskoj ploči, kao što je prikazano u kodu (Sl. 5.16.)~~(Sl. 5.16.)(Kod 5.16.)~~

```
84      int *pieces_count_per_board = new int[board_count];
85      for (int i = 0; i < board_count; i++)
86          pieces_count_per_board[i] = bEngine.getBoards()[i]->countOccupied();
```

(~~Kod~~ Sl. 5.16.) Isječak kôda spremanja broja zauzetih polja

Za brojanje figura i zauzetih polja se koristi *countOccupied()* funkcija koju smo definirali u dijelu koda (Sl. 5.17.)~~(Sl. 5.17.)(Kod 5.17.)~~

```
58  int Chessboard::countOccupied() {
59      int counter(0);
60      for (int i = 0; i < g_board_size; i++) {
61          for (int j = 0; j < g_board_size; j++) {
62              if (m_board[i][j] != 0)counter++;
63          }
64      }
65      return counter;
66  }
```

(~~Kod~~ Sl. 5.17.) Isječak funkcije *countOccupied()*

Dalje se alociraju i učitavaju sve *Sprite* figure i zauzeta polja prema kodu (Sl. 5.18.)~~(Sl. 5.18.)(Kod 5.18.)~~

```
89      Sprite** pieceSprite = new Sprite*[board_count];
90      for (int i = 0; i < board_count; i++)
91          pieceSprite[i] = new Sprite[pieces_count_per_board[i]];
92
93
94      for (int i = 0; i < board_count; i++)
95          for (int j = 0; j < pieces_count_per_board[i]; j++)
96              pieceSprite[i][j].setTexture(piecesTexture);
```

(~~Kod~~ Sl. 5.18.) Isječak kôda učitavanja *Sprite* figura

Učitane figure i zauzeta polja se zatim postavljaju na šahovske ploče prema kodu (Sl. 5.19.)~~(Sl. 5.19.)(Kod 5.19.)~~

```
100     for (int k = 0; k < board_count; k++)
101         loadPosition(bEngine.getBoards()[k]->getBoard(), pieceSprite, k);
```

(~~Kod~~ Sl. 5.19.) Isječak kôda učitavanja pozicija figura

Figure i zauzeta polja postavljaju se za svaku šahovsku ploču odvojeno u petlji s funkcijom *loadPosition()* po kodu (Sl. 5.20.)~~(Sl. 5.20.)(Kod 5.20.)~~

Commented [ČL6]: Izbjeći pisanje „dalje“, „zatim“, „potom“. Nije ovo hodogram.

Commented [ČL7]: Alociranje i učitavanje …

Commented [ČL8]: Promijeniti ovo u cijelom tekstu.

```
131  void loadPosition(int** board, Sprite** pieceSprite, int board_number) {
132      int k = 0;
133      for (int i = 0; i < g_board_size; i++)
134          for (int j = 0; j < g_board_size; j++) {
135              int n = board[i][j];
136              if (!n)
137                  continue;
138              int x = abs(n) - 1;
139              int y = n > 0 ? 1 : 0;
140              pieceSprite[board_number][k].setTextureRect(IntRect(size * x, size * y, size,
141              pieceSprite[board_number][k].setPosition(size * j, size * i);
142              k++;
143          }
144  }
```

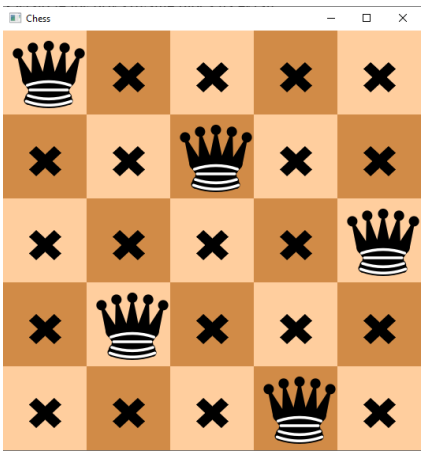
(~~Kod~~ Sl. 5.20.) Isječak funkcije *loadPosition*

SFML biblioteka radi na principu beskonačnih petlji i događajima koja spadaju u klasu *Event*. Prva beskonačna petlja se ponavlja dok se ne zatvori glavni prozor, a unutar petlje se definira događaj *e* i ugniježđena petlja koja čeka prvi događaj. Prvi događaj koji se definira je zatvaranje prozora. Zatim se definiraju naredbe unesene s pritiskom tipki na tipkovnici. Postoje 3 naredbe s kojima će korisnik raspolagati, odnosno 3 tipke na tipkovnici. Pritiskom na tipku razmak povećat će se indeks šahovske ploče, te samim time će se na ekranu prikazati sljedeća ploča. Pritiskom na tipku *backspace* događa se obrnuto, a pritiskom na tipku *enter* grafički prikaz se prekida te se broj prikazanih ploča vraća u početno grafičko korisničko sučelje opisano u poglavlju 5.1. Unutar početne beskonačne petlje se neprekidno crtaju i prikazuju šahovske ploče i šahovske figure. Potpuna implementacija koda prikazana je u kodu (Sl. 5.21.)~~(Sl. 5.21.)~~~~(Kod 5.21.)~~.

```
104     while (window.isOpen()) {
105         Event e;
106         while (window.pollEvent(e)) {
107             if (e.type == Event::Closed)
108                 window.close();
109 
110             if (e.type == Event::KeyPressed) {
111 
112                 if (e.key.code == Keyboard::Space)
113                     if(board_number < board_count-1)board_number++;
114 
115                 if (e.key.code == Keyboard::BackSpace)
116                     if(board_number>0)board_number--;
117 
118                 if (e.key.code == Keyboard::Enter)
119                     return board_count;
120             }
121         }
122         window.clear();
123         window.draw(sBoard);
124         for (int i = 0; i < pieces_count_per_board[board_number]; i++)
125             window.draw(pieceSprite[board_number][i]);
126 
127         window.display();
128     }
129 }
```

(~~Kod~~ Sl. 5.21.) Isječak kôda implementacije beskonačne petlje otvorenog prozora

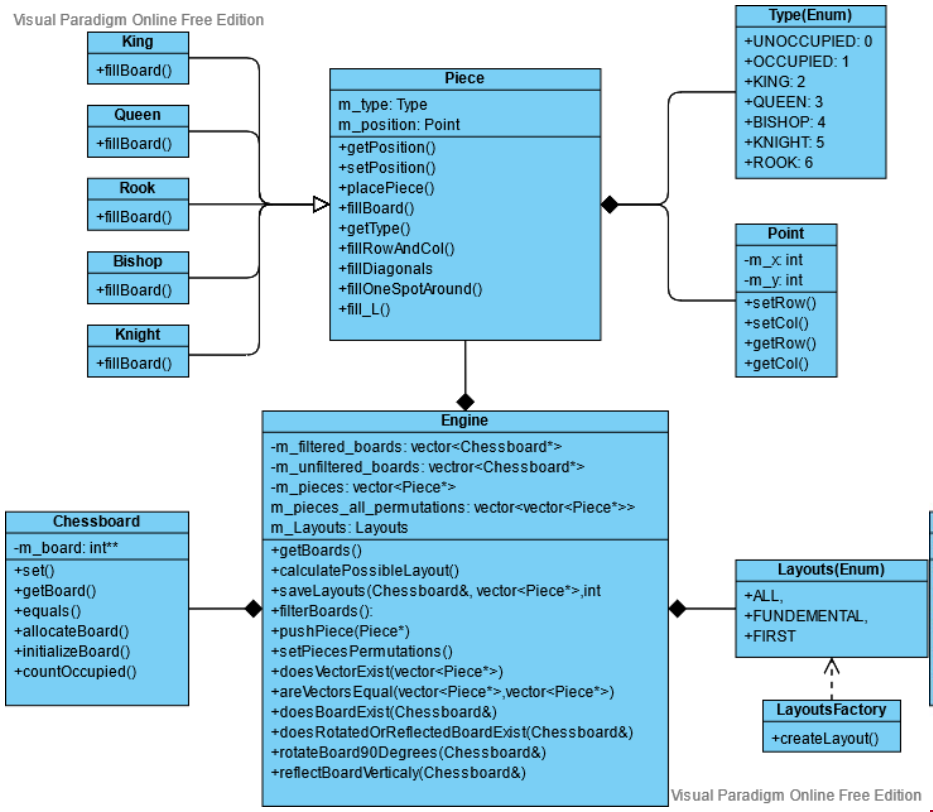
Na slici 5.2. je prikazan primjer beskonfliktnog rasporeda pet dama na šahovskoj ploči veličine 5×5.



Sl. 5.2. Prikaz jednog od ukupno dva različita beskonfliktna rasporeda pet dama na šahovskoj ploči veličine 5×5

5.3. UML dijagram klasa

UML (*Unified Model Language*) je jezik za modeliranje opće namjene. U programskom ~~inžinjerstvu~~inženjerstvu se koristi za vizualizaciju dizajna sustava [11]. Na slici 5.3. je prikazan UML dijagram klasa.



Sl. 5.3. UML dijagram klasa

Commented [ČL9]: Napisati izvor dijagrama

Commented [RR10R9]:

5.4. Prikaz mogućih poteza za svaku figuru

Pod prikazom mogućih poteza smatra se popunjavanje matrice šahovske ploče na način da se stavi broj 1 samo na ona polja na koja se figura može pomaknuti. Ukupno četiri funkcije će se koristiti za prikaz mogućih kretanja svih pet figura. Za označavanje redaka koristit će se slovo „*i*“, te slovo „*j*“ za stupce. Na sljedećim slikama unutar ovog poglavlja brojevi unutar uglatih zagrada će

prikazivati indeks retka i stupca [*i*][*j*], a brojevi unutar udaljenosti *d* biti će izračunati sljedećom izrazom

$$d(i-x, j-y) \qquad (5.224.)$$

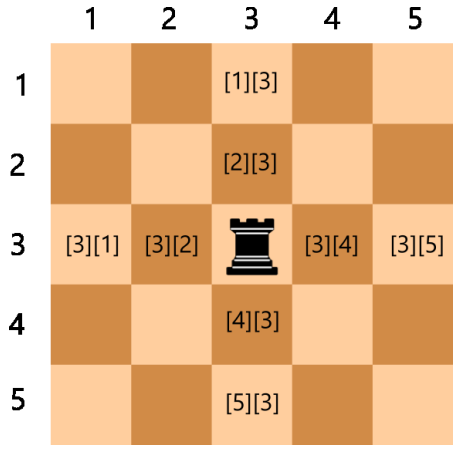
gdje je:

- *i* – indeks retka,
- *j* – indeks stupca,
- *x* – indeks retka figure,
- *y* – indeks stupca figure

Primjeri funkcija za popunjavanje šahovske ploče imati će dvije ugniježđene *for* petlje za prolazak kroz retke i stupce dvodimenzionalne matrice ploče, te će primati indeks retka figure, indeks stupca figure i veličinu šahovske ploče.

5.4.1. Prikaz mogućeg kretanja topa

Prikaz mogućeg kretanja topa je prikazan na slici 5.4. ~~5.4. 5.4.~~



Sl. 5.4. Prikaz mogućeg kretanja topa

Sukladno pravilima šaha top se može kretati samo u istom retku ili stupcu. Za funkciju za popunjavanje ploče vrijedi uvjet: „ako je indeks retka *i* jednak indeksu retka figure *x* ili ako je indeks stupca *j* jednak indexu stupca figure *y* onda stavi broj 1“. (Sl. 5.22.)~~(Sl. 5.22.)(Kod 5.23.)~~ prikazuje implementaciju cijele funkcije.

Commented [ČL11]: Što predstavlja ova oznaka?

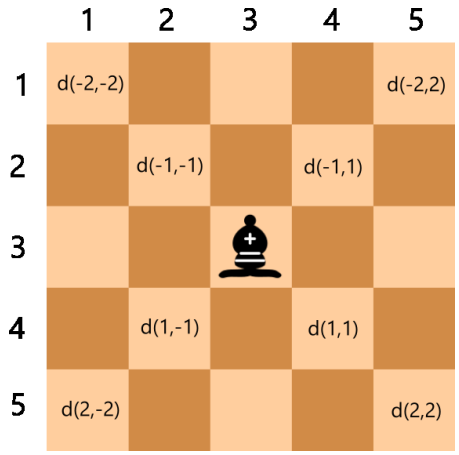
Formule se numeriraju također s dva broja, od kojih je prvi broj poglavlja, a drugi broj formule unutar poglavlja. Oznaka se stavlja u visini formule uz desni rub stranice, zatvorena je u okrugle zagrade, a brojevi su odvojeni crticom. Npr. prva formula u drugom poglavlju imala bi oznaku (2-1).

```
7   bool Piece::fillRowAndCol(Chessboard& board) {
8
9       for (int i = 0; i < g_board_size; i++)
10          for (int j = 0; j < g_board_size; j++){
11              if (i == m_position.getRow()){
12
13                  if (board(i,j) > Type::OCCUPIED)
14                      return FAILURE;
15
16                  board.set(i, j, Type::OCCUPIED);
17              }
18              if (j == m_position.getCol()) {
19
20                  if (board(i,j) > Type::OCCUPIED)
21                      return FAILURE;
22
23                  else board.set(i, j, Type::OCCUPIED);
24              }
25          }
26
27      return SUCCESS;
28  }
```

(~~Kod~~ Sl. 5.~~222223~~.) Isječak kôda kretanja topa

5.4.2. Prikaz mogućeg kretanja lovca

Na slici 5.~~5.5.5.5.~~ prikazano je moguće kretanje lovca s popunjenim poljima izračunatim pomoću izraza 5.~~5.~~5.~~22~~.



Sl. 5.5. Prikaz mogućeg kretanja lovca

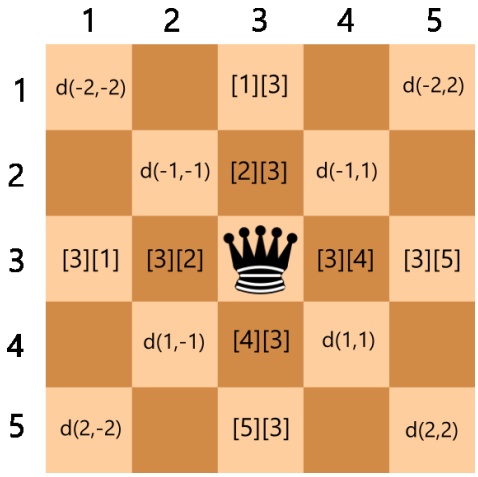
Sukladno pravilima šaha lovac se može kretati samo po dijagonalama. Na slici 5.65.65.6 može se uočiti sljedeći uvjet popunjavanja ploče: „ako je apsolutna vrijednost razlike indeksa retka *i* i indeksa retka figure *x* jednaka apsolutnoj vrijednosti razlike indeksa stupca *j* i indeksa stupca figure *y* onda u pripadno polje postavi broj 1“. (Sl. 5.23.)(Sl. 5.23.)(Kod 5.24.) prikazuje cijelu implementaciju funkcije.

```
30    bool Piece::fillDiagonals(Chessboard& board) {
31        for (int i = 0; i < g_board_size; i++)
32            for (int j = 0; j < g_board_size; j++)
33                if (abs(i - m_position.getRow())
34                    == abs(j - m_position.getCol())) {
35
36                    if (board(i,j) > Type::OCCUPIED)
37                        return FAILURE;
38                    else board.set(i, j, Type::OCCUPIED);
39                }
40
41        return SUCCESS;
42    }
```

(Kod Sl. 5.232324.) Isječak kôda kretanja lovca

5.4.3. Prikaz mogućeg kretanja dame

Prikaz mogućeg kretanja dame je prikazan na slici 5.65.65.6.

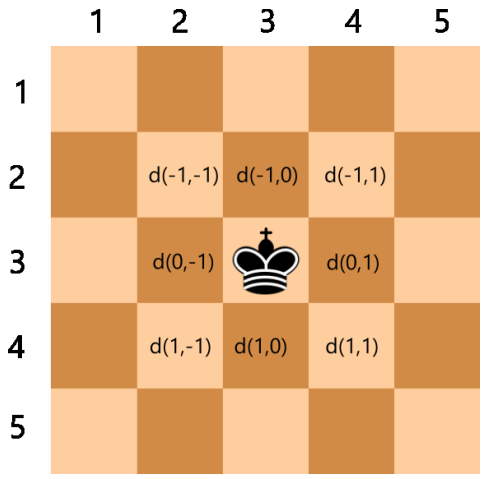


Sl. 5.6. Prikaz mogućeg kretanja dame

Sukladno pravilima šaha, dama se može kretati po istom retku, po istom stupcu i po dijagonalama, te bi za popunjavanje ploče pozvali dvije prethodno opisane funkcije popunjavanja istog retka i stupca i popunjavanja dijagonala.

5.4.4. Prikaz mogućeg kretanja kralja

Prikaz mogućeg kretanja kralja je prikazan na slici 5.7~~5.7~~5.7.



Sl. 5.7. Prikaz mogućeg kretanja kralja

Sukladno pravilima šaha kralj se može kretati po svakom polju oko svoje pozicije udaljeno za jedno polje. Na slici 5.7~~5.7~~5.7 može se uočiti sljedeći uvjet popunjavanja ploče: „ako je (apsolutna vrijednost razlike indeksa retka „i“ i indeksa retka figure „x“ manja ili jednaka 1) I (apsolutna vrijednost razlike indeksa stupca „j“ i indeksa stupca figure „y“ manja ili jednaka 1) onda u pripadno polje postavi broj 1“. (Sl. 5.24.)~~(Sl. 5.24.)~~~~(Kod 5.25.)~~ prikazuje implementaciju cijele funkcije.

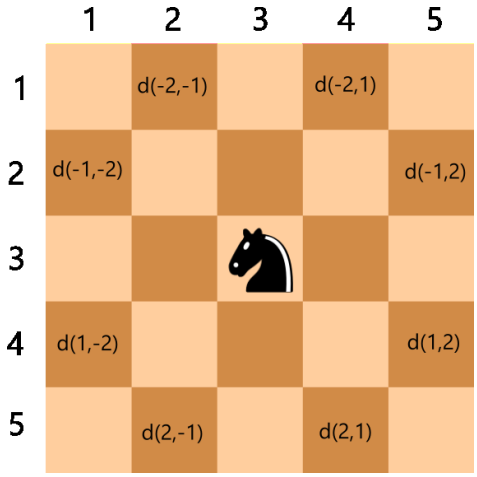
Commented [ČL12]: Poravnati s obje strane!

```
44  bool Piece::fillOneSpotAround(Chessboard& board) {
45      for (int i = 0; i < g_board_size; i++)
46          for (int j = 0; j < g_board_size; j++)
47              if ((abs(i - m_position.getRow())) <=
48                  1 && (abs(j - m_position.getCol())) <= 1) {
49
50                  if (board(i,j) > Type::OCCUPIED)
51                      return FAILURE;
52                  else
53                      board.set(i, j, Type::OCCUPIED);
54              }
55
56      return SUCCESS;
57  }
```

(~~Kod~~ Sl. 5.2424~~25~~.) Isječak kôda kretanja kralja

5.4.5. Prikaz mogućeg kretanja skakača

Prikaz mogućeg kretanja skakača je prikazan na slici 5.85.85.8



Sl. 5.8. Prikaz mogućeg kretanja skakača

Sukladno pravilima šaha skakač se može kretati po poljima do kojih može doći kretanjem u obliku slova „*L*“. Na slici 5.85.85.8 može se uočiti da su apsolutne vrijednosti razlike indeksa redaka „*i*“ i indeksa retka figure i apsolutne vrijednosti razlike indeksa stupaca „*j*“ i indeksa stupca figure uvijek 1 ili 2 i da nikad nisu jednake. Imajući to na umu uvjet za popunjavanje ploče glasi: „ako je [(apsolutna vrijednost razlike indeksa retka „*i*“ i indeksa retka figure „*x*“ jednaka broju 1) I (apsolutna vrijednost razlike indeksa stupca „*j*“ i indeksa stupca figure „*y*“ jednaka broju 2)] ILI [(apsolutna vrijednost razlike indeksa retka „*i*“ i indeksa retka figure „*x*“ jednaka broju 2) I (apsolutna vrijednost razlike indeksa stupca „*j*“ i indeksa stupca figure „*y*“ jednaka broju 1)] onda u pripadno polje postavi broj 1“. (Sl. 5.24.)(Sl. 5.24.)(Kod 5.25.) prikazuje implementaciju cijele funkcije.

```
59  bool Piece::fill_L(Chessboard& board) {
60      for (int i = 0; i < g_board_size; i++)
61          for (int j = 0; j < g_board_size; j++)
62              if ((abs(i - m_position.getRow()) ==
63                  2 && abs(j - m_position.getCol()) == 1)
64                  || (abs(i - m_position.getRow()) ==
65                      1 && abs(j - m_position.getCol()) == 2)) {
66
67                  if (board(i,j) > Type::OCCUPIED)
68                      return FAILURE;
69                  else board.set(i, j, Type::OCCUPIED);
70              }
71
72      return SUCCESS;
73  }
```

(~~Kod~~ Sl. 5.2~~52~~5~~26~~.) Isječak kôda kretanja skakača

5.5. Popunjavanje i permutacija niza figura

Glavni algoritam koji traži i sprema moguće beskonfliktne rasporede figura sekvencijalno prolazi po poljima šahovske ploče, te da bi se pronašle sve kombinacije prvo se moraju izračunati sve permutacije niza unesenih figura. Na slici 5.9.~~5.9.5.9.~~ je prikazan primjer dvaju različitih permutacija istog niza koji sadrži jednu damu, jednog lovca i jednog kralja.

Sl. 5.9. Primjer dvaju različitih permutacija istog niza

U izvornom kodu programa niz koji sadrži unesene figure zove se *m_pieces*, a tip niza je *vector* koji se nalazi unutar *std::vector* biblioteke. U kodu (Sl. 5.26.)~~(Sl. 5.26.)(Kod 5.27.)~~ je prikazana deklaracija niza *m_pieces*.

```
14      std::vector<Piece*>m_pieces;
```

(~~Kod~~ Sl. 5.2~~62~~6~~27~~.) Isječak kôda deklariranje vektora

Tip niza *vector* se koristi zbog fleksibilnog i jednostavnog rukovođenja dinamičkom memorijom. Program sam alocira i briše memoriju prilikom uništavanja niza. Početni niz se popunjava unutar

glavnog dijela programa odnosno *main* funkcije koja će primiti broj figura prilikom pokretanja programa. Program prima znakovne varijable opisane u poglavlju 992 koje se moraju pretvoriti u varijable cijelih brojeva kao što je prikazano u kodu (Sl. 5.27.)(Sl. 5.27.)(Kod 5.28.).

```
20      int* x = new int[argc];
21      for (int i = 1; i < argc; i++)
22          x[i] = atoi(argv[i]);
```

(Kod Sl. 5.272728.) Isječak kôda popunjavanja početnog niza

gdje je:

- x – niz koji sprema cjelobrojne vrijednosti,
- *atoi()* – funkcija koja pretvara znakovnu vrijednost u cjelobrojnu vrijednost

Potom se niz *x* sprema u zasebne varijable radi lakšeg rukovođenja. (Sl. 5.28.)(Sl. 5.28.)(Kod 5.29.).

```
24      const Layouts chosen_layouts = LayoutsFactory::createLayout(x[1]);
25      g_board_size = x[2];
26      const int queen_count = x[3];
27      const int king_count = x[4];
28      const int rook_count = x[5];
29      const int bishop_count = x[6];
30      const int knight_count = x[7];
```

(Kod Sl. 5.282829.) Isječak kôda spremanja niza *x* u zasebne varijable

Sljedeće se definira objekt klase *Engine* koji sadrži niz *m_pieces* i zatim se niz popunjava s funkcijom *pushPiece()* kao što je prikazano u kodu (Sl. 5.29.)(Sl. 5.29.)(Kod 5.30.).

```
36      Engine bEngine = Engine(chosen_layouts);
37
38      for (int i = 0; i < queen_count; i++)
39          bEngine.pushPiece(new Queen());
40
41      for (int i = 0; i < king_count; i++)
42          bEngine.pushPiece(new King());
43
44      for (int i = 0; i < rook_count; i++)
45          bEngine.pushPiece(new Rook());
46
47      for (int i = 0; i < bishop_count; i++)
48          bEngine.pushPiece(new Bishop());
49
50      for (int i = 0; i < knight_count; i++)
51          bEngine.pushPiece(new Knight());
```

(Kod Sl. 5.292930.) Isječak kôda popunjavanja nizova figura

Definicija funkcije *pushPiece()* je prikazana u kodu (Sl. 5.30.)(Sl. 5.30.)(Kod 5.31.).

```
89    void Engine::pushPiece(Piece* piece) {
90        m_pieces.push_back(piece);
91    }
```

(Kod Sl. 5.303031.) Isječak kôda popunjavanja niza

Nakon popunjavanja početnog niza *m_pieces* moraju se izvršiti sve permutacije s ponavljanjem niza *m_pieces* koje će se spremiti u niz koji sadrži nizove pod nazivom *m_all_pieces_permutations*. Deklaracija niza *m_pieces_all_permutations* prikazana je u kodu (Sl. 5.31.)(Sl. 5.31.)(Kod 5.32.).

```
15        std::vector<std::vector<Piece*>> m_pieces_all_permutations;
```

(Kod Sl. 5.313132.) Isječak kôda deklaracije niza koji sadrži niz

Prije popunjavanja svih permutacija niz *m_pieces* potrebno je sortirati. Koristit će se ugrađena funkcija *std::sort* koja prima početak niza, kraj niza i kriterij za sortiranje. Kako je klasa *Piece* vanjsko generirana klasa, funkcija *std::sort* ne zna po kojem kriteriju se mora sortirati. Klasa *Piece* u sebi sadrži *enumerator Type* koji je zapravo brojevna reprezentacija figura, te će se iskoristiti jednostavna *lambda* funkcija koja prima dva parametra tipa *Piece* i vraća usporedbu njihovih tipova. Ta usporedba će se koristiti za kriterij sortiranja. Implementacija *lambde* prikazana je u kodu (Sl. 5.32.)(Sl. 5.32.)(Kod 5.33.).

```
96        std::sort(m_pieces.begin(), m_pieces.end(),[](Piece*& p1, Piece*& p2) {
97            return p1->getType() < p2->getType();
98            });
```

(Kod Sl. 5.323233.) Isječak kôda sortiranja niza pomoću *lambda* funkcije

Popunjavanje niza nizova *m_pieces_all_permutations* izvršiti će se pomoću ugrađene funkcije *std::next_permutation* koja prima početak i kraj niza, te kriterij po kojem će se permutacije razlikovati, kao i kod funkcije *std::sort compiler* ne zna kako da ih razlikuje te se ponovno koristi ista *lambda* funkcija kao kriterij razlikovanja. Petlja je prikazana u kodu (Sl. 5.33.)(Sl. 5.33.)(Kod 5.34.).

```
100     do {
101         if (!doesVectorExist(m_pieces))
102             m_pieces_all_permutations.emplace_back(std::move(m_pieces));
103         counter++;
104     } while (std::next_permutation(begin(m_pieces), end(m_pieces), [](Piece
105         return p1->getType() < p2->getType();
106         }));
```

(~~Kod~~ Sl. 5.3333~~34~~.) Isječak kôda popunjavana niza koji sadrži nizove

gdje je:

- *doesVectorExist()* – funkcija koja provjerava da li niz već postoji u nizu nizova,
- *deepCopyVector()* – funkcija koja izvršava duboko kopiranje niza datog niza i vraća novostvoreni jednaki niz

Definicija funkcije *doesVectorExit()* je prikazana u kodu (Sl. 5.34.)~~(Sl. 5.34.)(Kod 5.35.)~~.

```
109 bool Engine::doesVectorExist(std::vector<Piece*> v) {
110     for (auto i : m_pieces_all_permutations)
111         if (areVectorsEqual(i, v))return true;
112
113     return false;
114 }
```

(~~Kod~~ Sl. 5.3434~~35~~.) Isječak funkcije *doesVectorExit*

gdje je:

- *areVectorsEqual()* – funkcija koja uspoređuje dva niza te vraća *true* ako su nizovi jednaki ili *false* ako nisu

5.6. Programiranje algoritma za dohvaćanje šahovskih ploča s beskonfliktnim rasporedom figura

Glavni algoritam za traženje beskonfliktnih rasporeda figura na šahovskoj ploči sastoji se od dvije funkcije. Obje funkcije vraćaju *boolean* vrijednost. Vrijednost *true* vratiti će se ako se uspio pronaći barem jedan beskonfliktni raspored danih figura, te vrijednost *false* ako nije moguće prikazati beskonfliktni raspored figura. (Sl. 5.35.)~~(Sl. 5.35.)(Kod 5.36.)~~ prikazuje deklaracije funkcija koje zajedno čine glavni algoritam programa.

```
25  bool calculatePossibleLayouts();
26  bool saveLayouts(Chessboard& board, std::vector<Piece*> pieces, int piece_index);
```

(~~Kod~~ Sl. 5.3535~~36~~.) Isječak kôda deklaracije glavnih funkcija

Prva funkcija *calculatePossibleLayouts()* se poziva iz početne funkcije *main* te ona poziva drugu funkciju *saveLayouts*() za svaku moguću permutaciju niza figura. (Sl. 5.36.)(Sl. 5.36.)(Kod 5.37.) prikazuje pozivanje *calculatePossibleLayouts* funkcije u *main* funkciji.

```
58      if (bEngine.calculatePossibleLayouts() == FAILURE)
59          return -1;
```

(Kod Sl. 5.363637.) Isječak kôda provjere uspješnosti algoritma

gdje je

- *function* – varijabla koja sadrži broj funkcije ovisno o pritisnutom gumbu, može biti od 1 do 3

5.6.1. Prva funkcija glavnog algoritma za spremanje beskonfliktnih rasporeda figura

Za početak mora se definirati jedna *boolean* varijabla koja će pratiti uspjeh pronalaska beskonfliktnog rasporeda. Zatim za svaku permutaciju definira se nova, prazna šahovska ploča i kopija niza šahovskih figura, te se poziva rekurzivna funkcija *saveLayouts* s prethodno definiranom praznom šahovskom pločom, kopijom niza šahovskih figura i brojem 0 koji označava početni indeks šahovske figure. Ako *saveLayouts* funkcija vrati *true*, to znači da je beskonfliktni raspored uspješno pronađen te se *boolean* varijabla postavi na *true* i izlazi se iz petlje. Cijela implementacija funkcije prikazana je u kodu (Sl. 5.37.)(Sl. 5.37.)(Kod 5.38.).

```
11  bool Engine::calculatePossibleLayouts() {
12      bool isSuccess = false;
13
14      while (!m_pieces_all_permutations.empty()) {
15          Chessboard* board = new Chessboard();
16          std::vector<Piece*> temp_pieces = std::move(m_pieces_all_permutations.back());
17
18          if (saveLayouts(*board, temp_pieces, 0))
19              isSuccess = true;
20          break;
21
22          m_pieces_all_permutations.pop_back();
23      }
24
25      return isSuccess;
26  }
```

(Kod Sl. 5.373738.) Isječak funkcije *calculatePossibleLayouts*

Nakon svih završenih iteracija vraća se rezultat uspjeha.

5.6.2. Druga funkcija za spremanje beskonfliktnih rasporeda figura

saveLayouts() je funkcija koja je zapravo temelj cijelog programa. To je rekurzivna funkcija koja prolazi kroz sva polja na šahovskoj ploči i pokušava postaviti figuru na ploču, ako ne uspije prelazi na sljedeće polje. Za početak definira se jedna varijabla tipa *boolean* s početnim stanjem *false* te će postati *true* tek kada se postave sve figure na ploču. Potom se definira jedna privremena šahovska ploča koja je kopija ploče poslana u funkciju koja će služiti za isprobavanje postavljanja figura. Sljedeće, prolazi se kroz svako polje na ploči i provjerava da li je polje dostupno (nije na putu kretanja drugih figura), te ako je mjesto slobodno, na to mjesto se pokušava postaviti figura s funkcijom *placePiece()* koja će pozvati jednu od funkcija kretanja figura opisanih u poglavlju 5.3, te će postaviti figuru na to polje. Ako je figura uspješno postavljena provjerava se da li je to posljednja figura u nizu, te u slučaju da je, odnosno da je postavljena posljednja figura u nizu onda se sprema ploča s rasporedom figura i ako je korisnik pritisnuo zadnji gumb (samo jedan raspored) rezultat se odmah vraća. A ako nije posljednja figura u nizu onda se rekurzivno poziva funkcija *saveLayouts*() s indeksom slijedeće figure, te ovisno o uspješnom rekurzivnom vraćanju funkcije postavlja se rezultanta varijabla *isSuccess* na *true,* te ako je korisnik odabrao prikaz prvog mogućeg rasporeda taj rezultat se odmah i vraća. Poslije svih obiđenih polja vraća se rezultat. Potpuna implementacija funkcije prikazana je u kodu (Sl. 5.38.)~~(Sl. 5.38.)~~~~(Kod 5.39.)~~.

Commented [ČL13]: Započeti rečenicu veznikom nije baš preporučljivo

```
32  bool Engine::saveLayouts(Chessboard& board, std::vector<Piece*> pieces, int piece_index) {
33      bool isSuccess = false;
34      Chessboard temp_board = std::move(board);
35
36      for (int i = 0; i < g_board_size; i++)
37          for (int j = 0; j < g_board_size; j++)
38              if (temp_board(i,j) == Type::UNOCCUPIED) {
39                  if (pieces[piece_index]->placePiece(i, j, temp_board) == SUCCESS) {
40                      if (piece_index == (pieces.size() - 1)) {
41
42                          m_unfiltered_boards.emplace_back(new Chessboard(std::move(temp_board)));
43                          isSuccess = true;
44                          if (m_layouts == Layouts::FIRST)
45                              return isSuccess;
46
47                      } else {
48                          if (saveLayouts(temp_board, pieces, piece_index + 1)) {
49
50                              isSuccess = true;
51                              if (m_layouts == Layouts::FIRST)
52                                  return isSuccess;
53                          }
54                      }
55                  }
56                  temp_board = std::move(board);
57              }
58
59      return isSuccess;
60  }
```

(~~Kod~~ Sl. 5.3~~8~~3~~9~~.) Isječak funkcije *saveLayouts*

gdje je:

- *pieces* – niz figura dobivene permutacije,
- *piece_index* – indeks figure u nizu
- *piece_count* – broj svih figura u nizu

5.7. Filtriranje ploča

Funkcija za filtriranje ploča *filterBoards()* poziva se iz početne funkcije *main* nakon funkcija za spremanje ploča beskonfliktnih rasporeda figura opisanih u poglavlju 5.6. Filtriranje ploča se mora izvršiti zato što funkcija *calculatePossibleLayouts()* sprema sve ploče svih permutacija koje sadrže i identične ploče. Ovisno o pritisnutom gumbu varijabla *m_layouts* može imati tri različite vrijednosti, te se s *switch* uvjetom provjerava o kojoj vrijednosti se radi. Ako je korisnik pritisnuo prvi gumb, odnosno želi vidjeti sve beskonfliktne rasporede figura na ploči onda se moraju izbaciti samo duplicirane ploče ili ako je korisnik pritisnuo drugi gumb, odnosno želi vidjeti samo osnovne rasporede figura na ploči onda se moraju izbaciti duplicirane ploče te ostale ploče koje kada se rotiraju i reflektiraju postanu duplicirane, te ako je korisnik pritisnuo treći gumb onda se sprema samo jedna spremljena ploča s rasporedom figura. Cijela implementacija funkcije prikazana je u kodu (Sl. 5.39.)~~(Sl. 5.39.)(Kod 5.40.)~~.

```
62  void Engine::filterBoards() {
63
64      switch (m_layouts) {
65      case Layouts::ALL:
66          for (auto board : m_unfiltered_boards)
67              if (!doesBoardExist(*board))
68                  m_filtered_boards.emplace_back(std::move(board));
69          break;
70
71      case Layouts::FUNDEMENTAL:
72          for (auto board : m_unfiltered_boards)
73              if (!doesRotatedOrReflectedBoardExist(*board))
74                  m_filtered_boards.emplace_back(std::move(board));
75          break;
76
77      case Layouts::FIRST:
78          m_filtered_boards.push_back(m_unfiltered_boards.back());
79          break;
80
81      default:
82          std::cout << "Error impossible function call\n";
83          break;
84      }
85  }
```

(~~Kod~~ Sl. 5.393940.) Isječak funkcije *filterBoards*

gdje je:

- *m_temp_boards* – niz koji sadrži sve spremljene ploče,
- *doesBoardExist()* – funkcija koja provjerava duplicirane ploče
- *doesRotatedOrReflectedBoardExist()* – funkcija koja provjerava duplicirane ploče, te duplicirane ploče nakon rotacije i refleksije

5.7.1. Provjera dupliciranih ploča

Funkcija *doesBoardExist()* provjerava postoje li duplicirane ploče, a definicija funkcije je prikazana u kodu (Sl. 5.40.)~~(Sl. 5.40.)(Kod 5.41.)~~.

```
125 bool Engine::doesBoardExist(Chessboard& board) {
126
127     for (auto existingBoard : m_filtered_boards) {
128         if (board.equals(*existingBoard))
129             return true;
130     }
131     return false;
132 }
```

(~~Kod~~ Sl. 5.404041.) Isječak funkcije *doesBoardExist*

gdje je:

- *board.equals*() – funkcija koja uspoređuje predanu ploču s trenutnom, te vraća *true* ako su ploče identične ili *false* ako su ploče različite

5.7.2. Provjera dupliciranih ploča nakon izvršavanja rotacije i refleksije

Funkcija *doesRotatedOrReflectedBoardExist()* provjerava postoje li duplicirane ploče, te ploče koje postanu duplicirane nakon rotacije i refleksije. Prvo se moraju provjeriti duplicirane ploče kao u funkciji *doesBoardExist(),* zatim se definiraju dvije privremene ploče kao kopije trenutne ploče koja se provjerava u nizu, te se privremene ploče četiri puta rotiraju za 90 stupnjeva i reflektiraju i se provjerava da li su postale duplicirane ploče. Potrebno je izvršiti samo vertikalnu refleksiju zbog svih rotacija ploče. Potpuna implementacija funkcije prikazana je u kodu (Sl. 5.41.)~~(Sl. 5.41.)~~~~(Kod 5.42.)~~.

```
134  bool Engine::doesRotatedOrReflectedBoardExist(Chessboard& const board) {
135
136      for (auto existingBoard : m_filtered_boards) {
137          if (board.equals(*existingBoard))
138              return true;
139
140          Chessboard rotated_tempBoard = *existingBoard;
141          Chessboard reflected_tempBoard = *existingBoard;
142
143          for (int i = 0; i < 4; i++) {
144              rotated_tempBoard=std::move(rotateBoard90Degrees(rotated_tempBoard));
145              reflected_tempBoard=std::move(reflectBoardVerticaly(rotated_tempBoard));
146
147              if (board.equals(std::move(rotated_tempBoard))
148                  || board.equals(std::move(reflected_tempBoard)))
149                  return true;
150          }
151      }
152
153      return false;
154  }
```

(~~Kod~~ Sl. 5.4141~~42~~.) Isječak funkcije *doesRotatedOrReflectedBoardExist*

gdje je:

- *rotateBoard90Degrees()* – funkcija koja simulira rotaciju ploče za 90 stupnjeva
- *reflectBoardVerticaly()* – funkcija koja simulira vertikalnu rotaciju

5.7.3. Simulacija rotacije ploče za 90 stupnjeva

Definicija funkcije *rotateBoard90Degrees()* je prikazana u kodu (Sl. 5.42.)~~(Sl. 5.42.)~~~~(Kod 5.43.)~~.

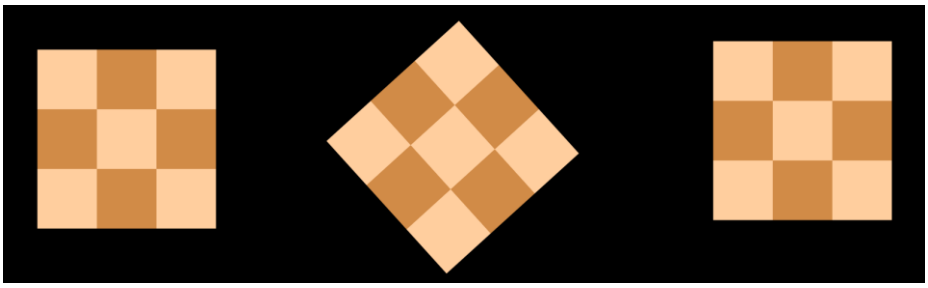
```
156  Chessboard Engine::rotateBoard90Degrees(Chessboard &board) {
157      Chessboard temp = std::move(board);
158      for (int i = 0; i < g_board_size; i++) {
159          for (int j = 0; j < g_board_size; j++) {
160              temp(i,j) = board(g_board_size - 1 - j, i);
161          }
162      }
163
164      return std::move(temp);
165  }
```

(~~Kod~~ Sl. 5.4~~24243~~.) Isječak funkcije *rotateBoard90Degrees*

gdje je:

- *board* – šahovska ploče za koju je potrebna simulacija rotacije za 90 stupnjeva

Na slici 5.10.~~5.10.5.10.~~ je grafički prikazana simulacija rotacije ploče za 90 stupnjeva.



Sl. 5.10. Grafički prikaz rotacije šahovske ploče za 90 stupnjeva

5.7.4. Simulacija vertikalne refleksije ploče

Definicija funkcije *reflectBoardVerticaly()* je prikazana u kodu (Sl. 5.43.)~~(Sl. 5.43.)(Kod 5.44.)~~.

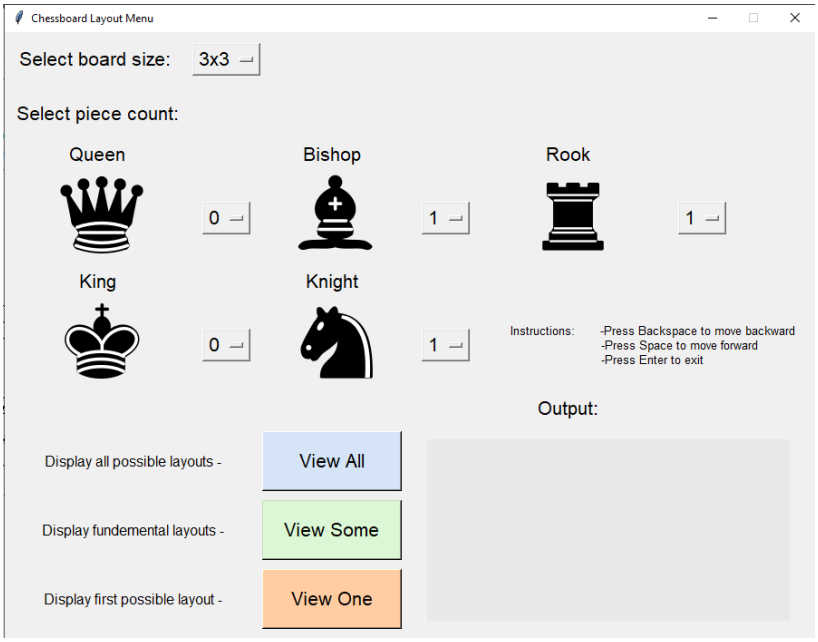
```
167  Chessboard Engine::reflectBoardVerticaly(Chessboard &board) {
168      Chessboard temp = std::move(board);
169      for (int i = 0; i < g_board_size; i++) {
170          for (int j = 0; j < g_board_size; j++) {
171              temp(i,j) = board(g_board_size - 1 - i, j);
172          }
173      }
174
175      return std::move(temp);
176  }
```

(~~Kod~~ Sl. 5.4~~34344~~.) Isječak funkcije *reflectBoardVerticaly*

5.8. Primjeri rješenja

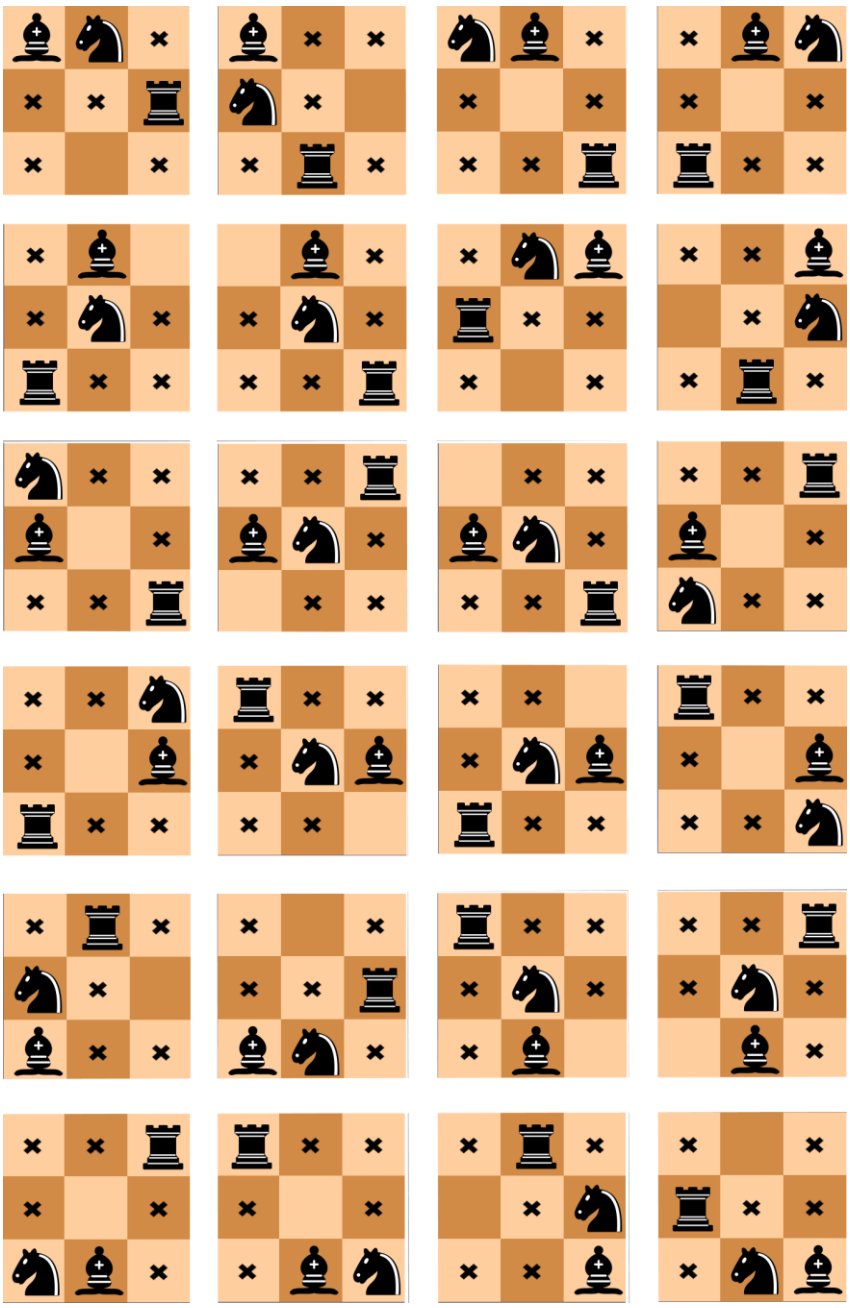
5.8.1. Prikaz rješenja za unos jednog topa, jednog skakača i jednog lovca na šahovskoj ploči 3×3

Na slici 5.445.445.45 prikazan je unos traženog broja figura i veličine šahovske ploče.



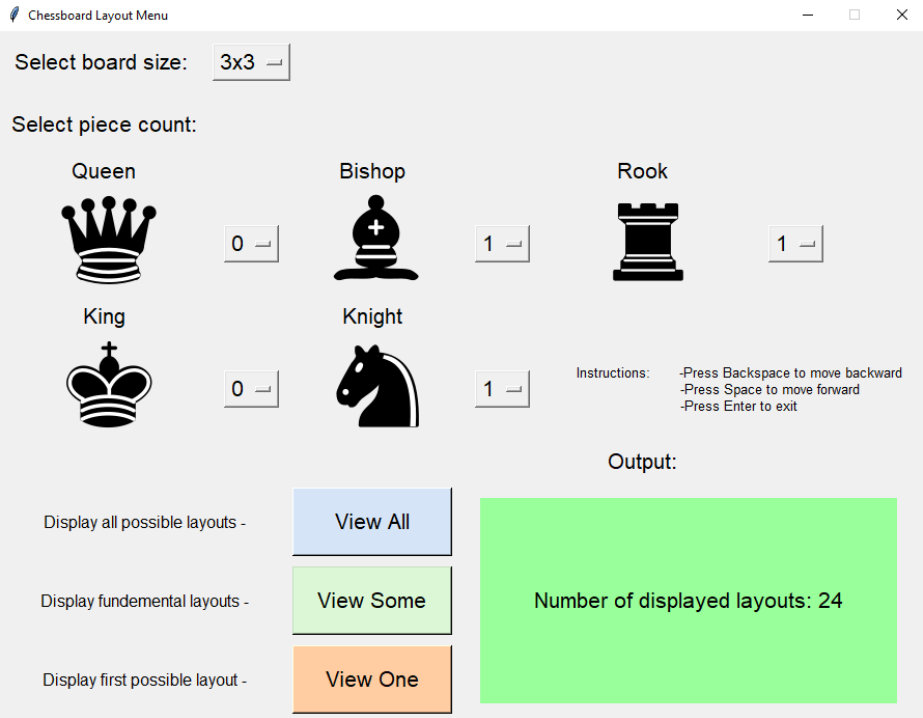
Sl. 5.444445.Početni unos figura i veličine šahovske ploče

Pritiskom na tipku „View All“ program izračuna te prikaže sve moguće rasporede unesenih figura. Slika 5.445.445.45 prikazuje sve moguće beskonfliktne rasporede za unesene figure.



Sl. 5.454546. Prikaz svih mogućih beskonfliktnih rasporeda za jednog topa, jednog lovca i jednog skakača na ploči veličine 3×3

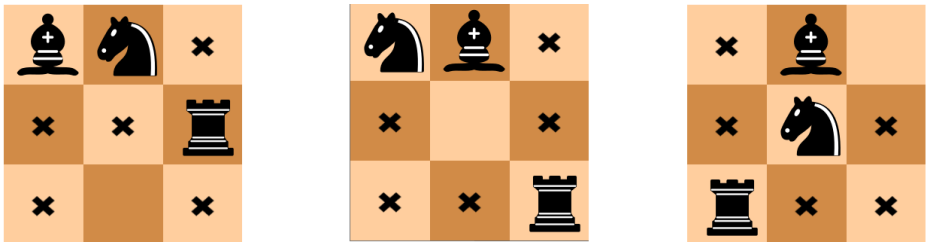
Na slici 5.455.455.46 je vidljivo kako algoritam postavljanja figura radi. Prvo pokušava postaviti lovca, zatim topa te na kraju skakača. Slika 5.465.465.47 prikazuje grafičko korisničko sučelje nakon završetka prikaza rasporeda figura.



Sl. 5.464647. Prikaz grafičkog korisničkog sučelja nakon izlaza iz prikaza rasporeda pritiskom na tipku *enter*

Pritiskom na tipku „View some“ program će izračunati te prikazati samo osnovne moguće rasporede unesenih figura, odnosno rasporede nakon što se uzme u obzir rotacija i refleksija šahovske ploče. Slika 5.475.475.48 prikazuje osnovne moguće beskonfliktne rasporede unesenih figura.

Field Code Changed

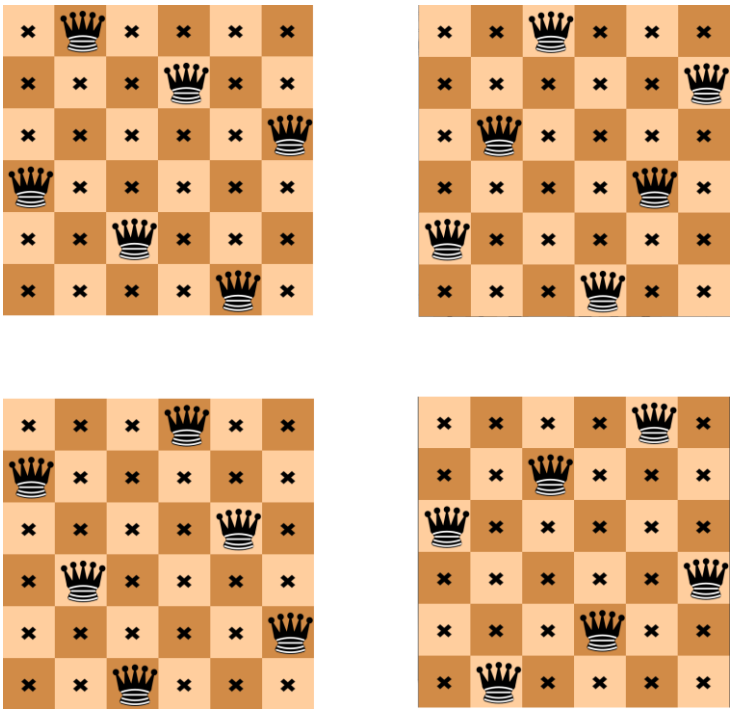


Sl. 5.474748. Prikaz osnovnih mogućih beskonfliktnih rasporeda za jednog topa, jednog lovca i jednog skakača na ploči veličine 3×3

Na kraju pritiskom na tipku „View One“ prikaže se samo prvi raspored.

5.8.2. Prikaz rješenja za unos šest dama na šahovskoj ploči 6×6

Slika 5.485.485.49 prikazuje sve moguće beskonfliktne rasporede za unos 6 dama na šahovskoj ploči 6×6

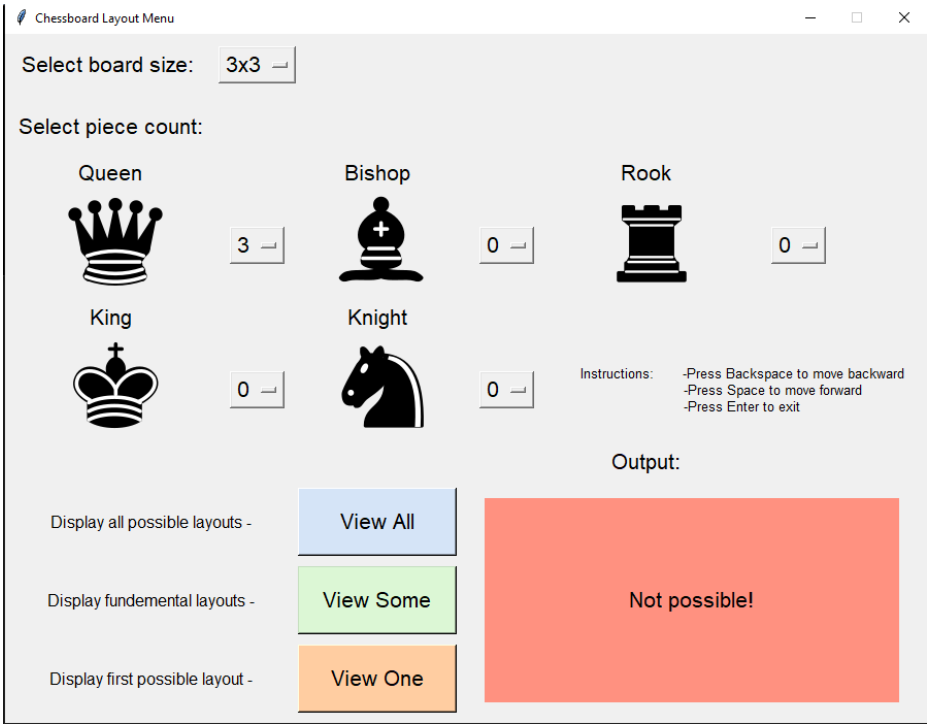


Sl. 5.484849.Prikaz svih mogućih beskonfliktnih rasporeda sa 6 dama na šahovskoj ploči 6×6

Na slici 5.48~~5.48~~~~5.49~~ je vidljivo kako uzimajući u obzir rotaciju i refleksiju te izbacujući duplikate ostaje samo jedan osnovni mogući raspored za šest dama na šahovskoj ploči veličine 6×6.

5.8.3. Primjer izlaza kod unosa za koje nije moguće prikazati beskonfliktne rasporede

Na slici 5.49~~5.49~~~~5.50~~ prikazano je grafičko korisničko sučelje za sve unose kod kojih je nemoguće izračunati beskonfliktne rasporede figura.



Sl. 5.4949~~50~~.Prikaz grafičkog korisničkog sučelja kod unosa 3 dame na ploču veličine 3×3

5.9. Analiza problema „n dama“

Koristeći ovaj problem moguće je prikazati sva rješenja za problem šahovski problem „*n* dama“ za slučaje da je *n* manji ili jednak broju 8. Tablica ~~1~~ prikazuje broj svih i osnovnih mogućih beskonfliktnih rasporeda u odnosu na broj dama za problem „*n* dama“.

Field Code Changed

Tablica 1 Rješenja za problem n dama

n	*Broj svih rasporeda*	*Broj osnovnih rasporeda*
3	0	0
4	2	1
5	10	2
6	4	1
7	40	6
8	92	12

~~Tablica 1. Rješenja za problem *n* dama~~

Dalje ćemo razmatrati vrijeme izvođenja algoritma. Tablica ~~2~~ prikazuje vrijeme izvođenja za sve prikaze rasporeda u odnosu na broj dama.

Field Code Changed

Tablica 2 Vrijeme izvođenja u odnosu na broj dama

n	*Vrijeme (m~~/~~s) za sve rasporede*	*Vrijeme (~~mm/~~s) za osnovne rasporede*	*Vrijeme (m~~/~~s) za prvi raspored*
4	1	1	0
5	11	29	1
6	149	145	5
7	4489	4558	1
8	15297	155021	516

~~Tablica 2. Vrijeme izvođenja u odnosu na broj dama~~

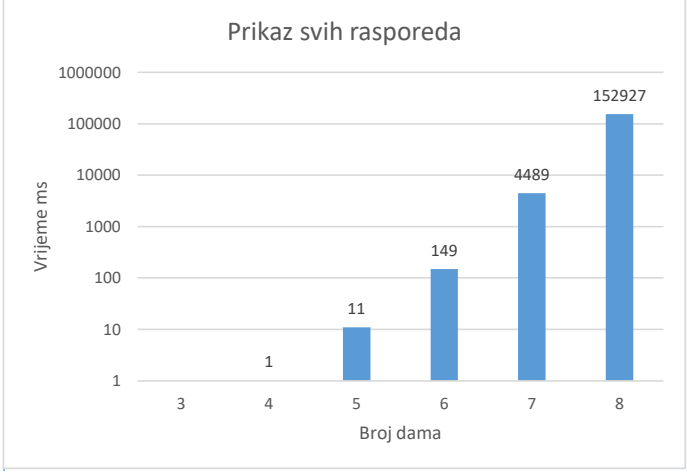
Commented [ČL14]: m/s?!!? to je jedinica za brzinu, a vjerujem da te mislili na ms – milisekunde

Promijeniti u cijelom dokumentu.

U tablici ~~2~~ zanimljivo je to što je za prikaz jednog rasporeda sa šest dama potrebno više vremena nego za prikaz sa sedam dama, to je zato što kod prikaza rasporeda sa šest dama nije moguće postaviti damu na prvo polje na ploči. Kod prikaza svih i osnovnih rasporeda vidljiv je eksponencijalni rast vremena izvođenja. ~~U grafikonu~~ Na slici Sl. 5.50~~Sl. 5.50~~Sl. 5.50~~Sl. 5.50~~Sl. 5.50~~Sl. 5.50~~Sl. 5.50~~Sl. 5.50~~1~~ grafički je prikazan eksponencijalni rast vremena izvođenja sa porastom broja dama.

Field Code Changed

Field Code Changed



Sl. 5.50.Prikaz grafičkog korisničkog sučelja kod unosa 3 dame na ploču veličine 3×3
~~Grafikon 1 Grafički prikaz eksponencijalnog rasta vremena izvođenja u odnosu na broj dama~~

U ovom završnom radu opisana je izrada programa koji omogućuje grafičko korisničko sučelje za unos broja šahovskih figura koje mogu biti: dama, kralj, top, lovac i skakač te unos veličine šahovske ploče. Nakon proizvoljnog unosa korisniku su ponuđena tri gumba, prvi gumb prikazuje sve moguće beskonfliktne rasporede, drugi gumb prikazuje samo osnovne beskonfliktne rasporede (rasporede koji izbacuju duplicirane ploče uzimajući u obzir rotaciju i refleksiju ploče) i treći gumb prikazuje prvi mogući beskonfliktni raspored figura. Veći broj figura zajedno sa većom šahovskom pločom podižu vrijeme izvođenja koje može biti i više minuta te treći gumb, odnosno, ispisivanje samo prvog prikaza omogućuje korisniku da na brzi način sazna postoji li uopće mogući beskonfliktni raspored figura na šahovskoj ploči. Cilj dizajna algoritma za pronalazak beskonfliktnih rasporeda figura je jednostavnost, skalabilnost i šira primjena. Algoritam je loše optimiziran i to se može primijetiti sa dužinom izvođenja kod većih brojeva kombinacija. Primjer optimizacije koja se može implementirati bila bi kod problema „*n* dama“, a to je preskakanje onih redaka i stupaca u kojima se već nalazi dama. Međutim to bi smanjilo jednostavnost i čitljivost koda te se nije implementirano. Program nudi jednostavno grafičko sučelje za prikaz beskonfliktnih rasporeda te jednostavne komande za pregledavanje beskonfliktnih rasporeda koje se unese pomoću tipkovnice. Prilikom izlaska iz prikaza rasporeda u početnom korisničkom sučelju ispisuje se ukupni broj prikazanih rasporeda ili poruka neuspješnosti ako nije moguće ispisati raspored za unesene figure. Osim raznih optimizacija algoritma moglo bi se i dizajnirati bolje i ljepše grafičko korisničko sučelje koje daje mogućnost mijenjanja veličine prozora

Commented [ČL15]: m/s -> ms

Commented [ČL16]: Nema grafikona… to je Slika #.#. Grafički prikaz…

aplikacije, ispis vremena izvođenja programa (ispis vremena trenutno je samo u konzoli), veći broj figura i veće moguće veličine šahovskih ploča.

LITERATURA

7. Bibliography

[1] M. S. Zur Luria, »A LOWER BOUND FOR THE n-QUEENS PROBLEM,« 9 7 2021. [Mrežno]. Available: https://arxiv.org/pdf/2105.11431.pdf. [Pokušaj pristupa 20 7 2021].

[2] T. n.-q. problem, 16 9 2021. [Mrežno]. Available: https://arxiv.org/pdf/2109.08083.pdf. [Pokušaj pristupa 20 7 2021].

[3] »Transfer of IP in OEIS to The OEIS Foundation Inc.,« 26 October 2009. [Mrežno]. Available: https://web.archive.org/web/20131206172532/http://oeisf.org/index.html#IPXFER.

[4] OEIS, »The On-Line Encyclopedia Of Integer Sequences,« [Mrežno]. Available: https://oeis.org/A002562.

[5] T. O.-L. E. O. I. Sequences. [Mrežno]. Available: https://oeis.org/A000170. [Pokušaj pristupa 20 7 2021].

[6] M. H.J.R., Povijest šaha, Oxford, 1913.

[7] Cburnett, »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Template:Chess_diagram. [Pokušaj pristupa 20 7 2021].

[8] »Wikipedia,« [Mrežno]. Available: https://commons.wikimedia.org/wiki/Template:SVG_chess_pieces. [Pokušaj pristupa 20 7 2021].

[9] »Simple and Fast Multimedia Library,« [Mrežno]. Available: https://www.sfml-dev.org/. [Pokušaj pristupa 20 7 2021].

[10 Python, »Python,« [Mrežno]. Available: https://www.python.org/. [Pokušaj pristupa 20 7 2021].

[11 »Tkinter,« [Mrežno]. Available: https://docs.python.org/3/library/tkinter.html. [Pokušaj pristupa 20 7 2021].

[12 B. Stroustrup, The C++ Programming Language, Bjarne Stroustrup, 1997.

[13 G. V. Rossum, The History of Python: A Brief Timeline of Python, 2009.

[14 »RIP Tutorial,« [Mrežno]. Available: https://riptutorial.com/tkinter/example/29713/grid--.

[15 »Fonts Google,« [Mrežno]. Available: https://fonts.google.com/specimen/Roboto. [Pokušaj pristupa 20 7 2021].

[16 A. -. Weskey, »Unified Modeling Language User Guide 2. edition,« 2005, p. 496.

[17 Xilinx, »Embedded System Tools Reference Manual - Embedded Development Kit,« Xilinx, 2008.

[18 Digilent, »Nexys3 Board Reference Manual,« Digilent, Pullman, WA, 2013.

[19 Xilinx, »MicroBlaze Processor Reference Guide - Embedded Development Kit EDK 10.1i,« Xilinx, 2008.

[20 P. Marwedel, Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems, Springer Netherlands, 2011.

[21 J. O. Hamblen, T. S. Hall i M. D. Furman, Rapid Prototyping of Digital Systems - SOPC Edition, Springer US, 2008.

[22 Cburnett, »Wikipedia,« [Mrežno]. Available: https://commons.wikimedia.org/wiki/Category:SVG_chess_pieces#/media/File:Chess_Pieces_Sprite.svg.

Commented [ČL17]: 2.9. Literaturu treba svrstati redom kojim se pojavljuje u radu i napisati na sljedeći način:
- *Primjer za članak iz časopisa:* 4

[1] Inicijali imena, prezime autora, naslov rada, naziv časopisa, broj časopisa (br./No.), broj sveska (sv./Vol.), str. (pp.) od – do, mjesec i godina izdanja.4
- *Primjer za referat objavljen u zborniku konferencije:*
[2] Inicijali imena, prezime autora, naslov referata, naziv konferencije, sv. (broj sveska), str. (stranice) od – do, mjesto, godina izdanja.
- Primjer za knjigu:
[3] Inicijali imena, prezime autora, naslov knjige, izdavač, mjesto, godina izdanja.
- *Primjer za web-stranicu:*
[4] Inicijali imena, prezime autora, naslov: podnaslov [online], nakladnik (nakladnik u tradicionalnom smislu ili organizacija odgovorna za održavanje stranice na internetu), mjesto izdavanja, godina izdavanja, dostupno na: URL [**datum zadnje posjete stranici**]

SAŽETAK

Tema ovog završnog rada je: „Beskonfliktni raspored figura na šahovskoj ploči u programskom jeziku C++“. Za izradu ovog rada bilo je potrebno poznavati programski jezik C++ koji se koristio za izradu algoritma koji računa i prikazuje beskonfliktne rasporede te programski jezik python koji se koristio za izradu grafičkog korisničkog sučelja. U grafičkom korisničkom sučelju je ponuđen unos broja šahovskih figura te unos veličine šahovske ploče. Ponuđena su i tri gumba, prvi prikazuje sve rasporede, drugi prikazuje osnovne rasporede i treći prikazuje jedan raspored. Program nudi grafičko sučelje za prikaz rasporeda, te ispis ukupnog broja prikazanih rasporeda u grafičkom korisničkom sučelju.

Ključne riječi: beskonfliktni raspored, osnovni rasporedi, svi rasporedi~~, osnovni rasporedi~~

Commented [ČL18]: **Sažetak** s ključnim riječima na hrvatskom jeziku (do 5, poredanih abecedno). Potrebno je opisati glavni problem, naznačiti smjernice kako je rješavan te naznačiti postignute rezultate završnog rada

Commented [RR19R18]:

Commented [ČL20]: **Sažetak** s ključnim riječima na hrvatskom jeziku (do 5, poredanih abecedno). Potrebno je opisati glavni problem, naznačiti smjernice kako je rješavan te naznačiti postignute rezultate završnog rada

ABSTRACT

Subject of this thesis is: „Non-attacking chessboard layout in C++ programming language“. For the making of this thesis it was necessary to undestand programming language C++ which was used to create the algorithm which computes and displays non-attacking chessboard layouts, also it was necessary to understand programming languge python which was used to create graphical user interface. In graphical user interface user is able to input number of chess pieces and size of chessboard. There are three buttons available, first button displays all layouts, second button displays fundemental layouts and third button displays one layout. Program displays graphical representation of chessboard layouts and the output which is number of displayed layouts shown in graphical user interface.

Keywords: all layouts, fundemental layouts, non-attacking layouts~~, all layouts, fundemental layouts~~

Commented [ČL21]: **Sažetak** s ključnim riječima na hrvatskom jeziku (do 5, poredanih abecedno). Potrebno je opisati glavni problem, naznačiti smjernice kako je rješavan te naznačiti postignute rezultate završnog rada

Commented [ČL22]: **Sažetak** s ključnim riječima na hrvatskom jeziku (do 5, poredanih abecedno). Potrebno je opisati glavni problem, naznačiti smjernice kako je rješavan te naznačiti postignute rezultate završnog rada

ŽIVOTOPIS

Autor ovog završnog rada, Robin Rajšić je student stručnog preddiplomskog studija Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek te sada radi kao programski inženjer u kompaniji Ericsson Nikola Tesla d.d.. Robin Rajšić pohađao je srednju školu Tehnička škola Kutina smjer Računarstvo. Nakon srednje škole napravio je pauzu od studiranja u kojoj se bavio konobarstvom. Tijekom studiranja Robin Rajšić je upoznat sa raznim programskim jezicima a to su: C++, C#, python, Java, HTML, SQL..itd. te je stekao znanje o algoritmima i strukturama podataka. Dana 29.09.2020. god. Robinu Rajšiću dodijeljeno je priznanje za postignut uspjeh u studiranju.