Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,

RAČUNALNIŠTVO IN INFORMATIKO

Martin Konečnik

GRAFIČNI VMESNIK ZA IGRANJE ŠAHA

Diplomsko delo

Maribor, avgust 2014

GRAFIČNI VMESNIK ZA IGRANJE ŠAHA

Diplomsko delo

Študent:	Martin Konečnik
Študijski program:	Univerzitetni študijski program Računalništvo in informacijske tehnologije
Mentor:	doc. dr. Borko Bošković
Somentor:	red. prof. dr. Janez Brest
Lektorica:	Boža Božena Lesjak, prof. RP, svetovalka

Grafični vmesnik za igranje šaha

Ključne besede: Grafični vmesnik, programska knjižnica Qt, protokol UCI

UDK: 004.5:794.1(043.2)

Povzetek

V okviru diplomske naloge smo izdelali grafični uporabniški vmesnik za igranje šaha v programskem okolju Qt. Preučili smo določena programska orodja in se soočili s problemom učinkovitega testiranja in zagotavljanja pravilnega delovanja vmesnika ob povezavi s šahovskim pogonom (umetna inteligenca).

Končni produkt je popolnoma delujoč vmesnik za igranje šaha proti drugemu igralcu ali pogonu. Poleg osnovnih funkcionalnosti igralcu pomaga z barvanjem dovoljenih polj in preprečuje premikanje figur na nedovoljena polja. Omogoča tudi nastavitve vmesnika, kot so sprememba barve polj na šahovnici, nalaganje slik figur iz datoteke ter spreminjanje velikosti šahovnice. Poleg privzetega jezika, angleščine, vsebuje še prevod v slovenščino. Jezik se samodejno izbere glede na jezik okolja, v katerem se je aplikacija zagnala.

Chess Graphical User Interface

Key words: graphical user interface, Qt library, UCI protocol

UDK: 004.5:794.1(043.2)

Abstract

This thesis is focused on the problem of creating a graphical user interface for chess in Qt development environment. It introduces the tools we used in the implementation of this application. Furthermore it focuses on the problem of efficient testing and ensuring the connection between the interface and chess engine (artificial intelligence) is working flawlessly.

The final product is a completely functional chess user interface which allows two player games and playing against an engine. Along with the basic functionality it also helps the player by coloring allowed moves and preventing unallowed moves. The product also allows cosmetic changes such as changing of square colors, loading custom pieces from file and chess board scaling. It not only includes the default language, English, but also a Slovene translation that is loaded automatically according to the environment language where the application was started.

Kazalo vsebine

Kazalo slik

Seznam uporabljenih kratic

- DLL – Dinamic-link Library
- GCC – GNU Compiler Collection
- GNU – GNU's Not Unix!
- GUI – Graphical User Interface
- IDE – Integrated Developer Environment
- MinGW - Minimalist GNU for Windows
- UCI – Universal Chess Interface

1 Uvod

Diplomsko delo obravnava izdelavo grafičnega uporabniškega vmesnika za igranje šaha (*angl. „chess GUI"*). Pokriva le izdelavo vmesnika brez pogona (*angl. „chess engine"*) oz. umetne inteligence (*angl. „artificial intelligence*"), ki je potrebna za igro proti računalniku. Vmesnik bo omogočal igro med dvema igralcema, za igro proti računalniku pa je v vmesnik potrebno vključiti še pogon.

Sicer obstaja dosti grafičnih vmesnikov za igranje šaha, vendar imajo vsi vmesniki določene pomanjkljivosti in slabosti, predvsem pa ne omogočajo vseh funkcionalnosti, ki jih od vmesnika želimo. Zato je cilj diplomskega dela implementirati lasten grafični uporabniški vmesnik za igranje šaha, ki bo po eni strani uporabljal karseda malo računalniških virov, po drugi strani pa bo uporabniku omogočal prijazen in funkcionalen vmesnik.

Diplomsko delo je predstavljeno v šestih poglavjih. V drugem je na kratko opisanih nekaj podobnih programov, po katerih smo se zgledovali. V naslednjem poglavju so opisana orodja uporabljena pri izdelavi vmesnika, naveden pa je tudi opis protokola UCI, s katerim smo delali. V četrtem poglavju smo opisali značilnosti in možnosti grafičnega uporabniškega vmesnika, kot je na primer spreminjanje izgleda figur ter polj na šahovnici. V predzadnjem poglavju podajamo način, kako smo preverili pravilnost delovanja vmesnika v igri med dvema igralcema ter proti različnim šahovskim pogonom, na koncu pa povzamemo opravljeno delo.

2 Sorodna dela

V tem poglavju bomo na kratko opisali dela, ki so sorodna našemu projektu. Iz teh del so bile črpane nekatere ideje, ki smo jih nato v celoti ali delno implementirali v diplomskem delu. Vsi ti projekti so v razvoju bistveno dalj časa, tako da nudijo naprednejše funkcije, kot naš grafični vmesnik.

2.1 ChessX

ChessX (Slika 2.1) je odprtokodna baza za šah. Z njim lahko shranjene igre organiziramo, analiziramo, spreminjamo. Je večplatformni in deluje na platformah Windows, Linux ter Mac OS X.

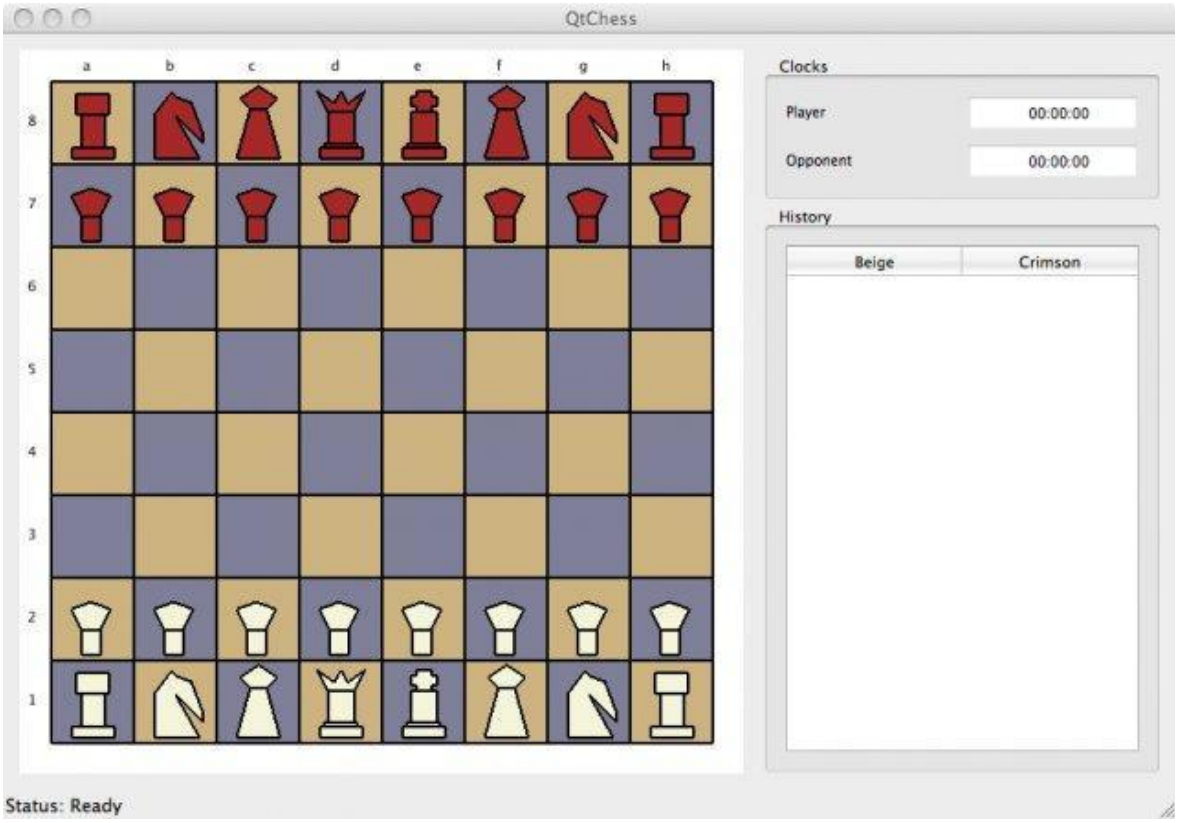
Slika 2.1: ChessX [24]

Podobno kot naša aplikacija tudi ChessX uporablja Qt 4.8.1, razen za Mac OS X, kjer je v uporabi različica Qt 5.3.1.

Kot odprtokodna aplikacija ima ChessX mnogo razvijalcev, od katerih se nekateri ne ukvarjajo več z razvijanjem kode. Med temi gre omeniti imena Marius Roets, Sean Estabrooks, Rico Zenklusen, William Hoggarth in Ejner Borgbjerg, ki so vsi napisali precejšnje dele kode, Heinz Hopfgartner pa je aplikacijo testiral za Mac OS X.

Aplikacija je na voljo v mnogo jezikih. Original je napisan v angleščini, podprti pa so tudi nemščina, francoščina, italijanščina, danščina ter drugi jeziki, ki pa niso vzdrževani [23].

2.2 Qt Chess (Cutechess)



Slika 2.2: Qt Chess [25]

Qt Chess (Slika 2.2) je zbirka orodij za delo s pogoni za šah. Vsebuje grafični vmesnik, konzolno aplikacijo za samodejno igranje med pogoni ter knjižnico za delo z različnimi pogoni za šah. Tako kot naš vmesnik je tudi ta razvit s knjižnico Qt v programskem jeziku C++.

Med naštetimi vmesniki je najpreprostejši in tako najbolj podoben našemu. Kljub temu pa velja omeniti, da ima za seboj daljše razvojno obdobje [6][19].

2.3 XBoard

Slika 2.3: Xboard med igro nestandardnega šaha [26]

Xboard (Slika 2.3) je najnaprednejši grafični vmesnik med naštetimi. Poleg običajnega šaha podpira tudi xiangqi (kitajski šah), shogi (japonski šah) ter mnoge druge variacije šaha in omogoča uporabo različnih pogonov. Omogoča tudi igranje šaha prek elektronske pošte, tako da razpozna poteze prejete po pošti ter pošilja poteze opravljene na naši strani

igralne plošče. Poleg tega omogoča tudi igranje prek spleta.

Poleg protokola UCI, ki ga uporabljamo, vmesnik omogoča igranje šaha tudi prek lastnega protokola Xboard, oz. WinBoard [29].

3 Opis uporabljenih tehnologij

3.1 Uporabljena orodja

Orodja uporabljena pri razvoju naše aplikacije:

- Qt Designer,
- Qt Creator (4.8.5),
- MinGW (4.4.0) in
- Bitbucket.

Uporabljene so bile 32-bitne verzije aplikacij.

3.1.1 Qt Designer

Qt Designer je večplatformni program za urejanje postavitve elementov na grafičnem uporabniškem vmesniku ter za upravljanje z okni. V orodju Microsoft Visual Studio ga lahko primerjamo z Microsoftovimi Okni (*angl. „Windows Forms"*) [3]. Qt Designer omogoča izjemno hitro izdelavo aplikacij s pomočjo različnih postavitev ter okvirjev, ki lahko vsebujejo različne elemente (*angl. „widgets"*) [20].

Za Qt smo se odločili iz treh razlogov. Prvi je hitrost aplikacij napisanih v tem okolju, proti aplikacijam napisanim na primer v okolju .NET z jezikom C#, v katerem smo do sedaj pisali večino aplikacij. Ta faktor je sicer zelo pomemben in bi lahko že sam po sebi botroval k izbiri razvijalnega okolja, vendar ga ne škodi podkrepiti. Naslednji razlog je možnost preprostejšega prenosa aplikacije med platformami. Načeloma bi moralo biti mogoče kodo napisano s Qt v jeziku C++ prevesti na različnih platformah s podobnimi rezultati. Včasih temu sicer ni tako in so zato potrebne spremembe, a bi načeloma morale biti preprostejše, kot razvoj celotnega programa v drugem jeziku [10]. Zadnji razlog sicer nima ničesar opraviti s kodiranjem in je bolj praktične narave. Qt je namreč odprtokoden in tako ni potrebno plačevati za razvoj aplikacij v le-tem.

3.1.2 Qt Creator

Qt Creator je večplatformno integrirano razvijalno okolje (*angl. „integrated development environment"* - *IDE*). Je orodje, ki omogoča razvijanje tako okenskih, kot konzolnih aplikacij ter aplikacij za druge naprave.

3.1.3 MinGW

MinGW je zbirka neplačljivih orodij, ki vključuje GCC [9] prevajalnik za platformo Windows. Primerna je za razvijanje aplikacij na tej platformi. Ni odvisna od dinamičnih povezovalnih knjižnic (v nadaljevanju DLL) tretjih oseb, odvisna pa je od nekaterih knjižnic DLL, ki so del operacijskega sistema Windows. Najpomembnejša je *„MSVCRT.DLL"*, ki je zaganjalna knjižnica (*angl. „runtime library"*) Microsofta [17].

3.1.4 Bitbucket

Za kontrolo različice programa ter možnost prevrtenja različice nazaj (*angl. „rollback"*), je bila uporabljena zastonjska različica spletnega orodja Bitbucket.

Ta različica omogoča ustvarjanje ene ali več map (*angl. „repository"*) ter uporabo orodja s strani največ petih uporabnikov. Orodje omogoča tudi izbiro, ali želimo, da so mape vidne le nam, ali več ljudem [2].

Omeniti velja še, da Bitbucket za kontrolo različic (*angl. „version control"*) uporablja Mercurial ali Git. Mi smo uporabili Git [28].

3.2 Knjižnica Qt

V projektu so bile uporabljene le knjižnice, ki so že vgrajene v ogrodje Qt. Za to smo se odločili iz dveh razlogov; posebne potrebe po knjižnicah ni bilo, poleg tega pa je eden izmed ciljev aplikacije, da je čim preprosteje prenosljiva.

3.3 Protokol umetne inteligence

Izdelan program omogoča kompatibilnost z vsemi pogoni, ki za komunikacijo uporabljajo protokol UCI (*Universal Chess Interface*) [11]. Za podporo tega protokola smo se odločili, ker je preprostejši za uporabo in je večinoma zamenjal starejši *Chess Engine*

Communication Protocol, ki ga uporablja Xboard.

Grafični uporabniški vmesnik s podporo UCI ne samo sprejema in pošilja podatke pogonu, ampak tudi vsebuje predstavitev pozicije in je tako zmožen odločati o koncu igre, kje je možna rokada, promocija ter poteza en passant.

Kljub temu, da je protokol UCI vse bolj razširjen, pa prejema tudi dosti kritik. Zelo dobro jih povzame Robert Hyatt v intervjuju s Frankom Quisinskyjem leta 2002 [13], ki pa je danes še vseeno relevanten. Prevedeno iz angleščine: „UCI mi preprosto ni všeč. Zaobide **vse** krmilne parametre. Pogonu pove, kdaj naj misli, išče, prekine iskanje, itd. To je v nasprotju z mojim načrtovanjem in ne nameravam spremeniti Crafty-ja, da bi podpiral nekaj, kar je tako različno od WinBoard/XBoard protokola, ki obstaja že **dolgo** časa in deluje **popolnoma** dobro. UCI odstrani veliko kritičnih odločitev, za katere je bolje, da jih opravi pogon in ne grafični vmesnik."

Na drugi strani so seveda podporniki tega protokola, kot na primer Fabien Letouzey [12], ki v intervjuju leta 2005 poudari preprostost implementacije: „Izbira protokola UCI je osnovana na principih načrtovanja programske opreme, ki jih ni preprosto razložiti. V bistvu je programerska stvar in ne pričakujem, da bodo uporabniki pogonov razumeli. Naj vam pa dam namig: pomislite na mlade pogone WinBoard, ki ste jih preizkusili; koliko jih je podpiralo predvidevanje (*angl. „pondering"*) … brez hroščev??? UCI je dober za programerje, ker vodi do manj hroščev v kodi."

Na podlagi teh intervjujev se jasno vidi, da imata oba vodilna protokola svoje prednosti, prav tako pa svoje slabosti. WinBoard je mogoče res boljši protokol; ko je implementiran brez večjih hroščev. Včasih pa je bolje morebitno kvaliteto zamenjati za zanesljivost. Če je že pogon poln hroščev, bodo ti seveda vplivali tudi na grafični vmesnik in povzročili precejšnjo zmedo.

Povedali smo, kaj protokol UCI je in kaj so njegove prednosti oz. slabosti, nismo pa še razložili, kako deluje. Komunikacija poteka s pomočjo vzhodno izhodnih tokov in pip med procesi. Ob zagonu pogon počaka na ukaz *„uci"*. Ta ukaz pogonu sporoči, da vmesnik želi komunicirati prek protokola UCI. Ob prejemu tega ukaza se pogon identificira in pošlje možnosti, ki jih je mogoče spremeniti. Nato pogon z vrstico *„uciok"* sporoči, da je poslal vse podatke in je pripravljen na nadaljevanje. Po prejemu tega ukaza lahko tako

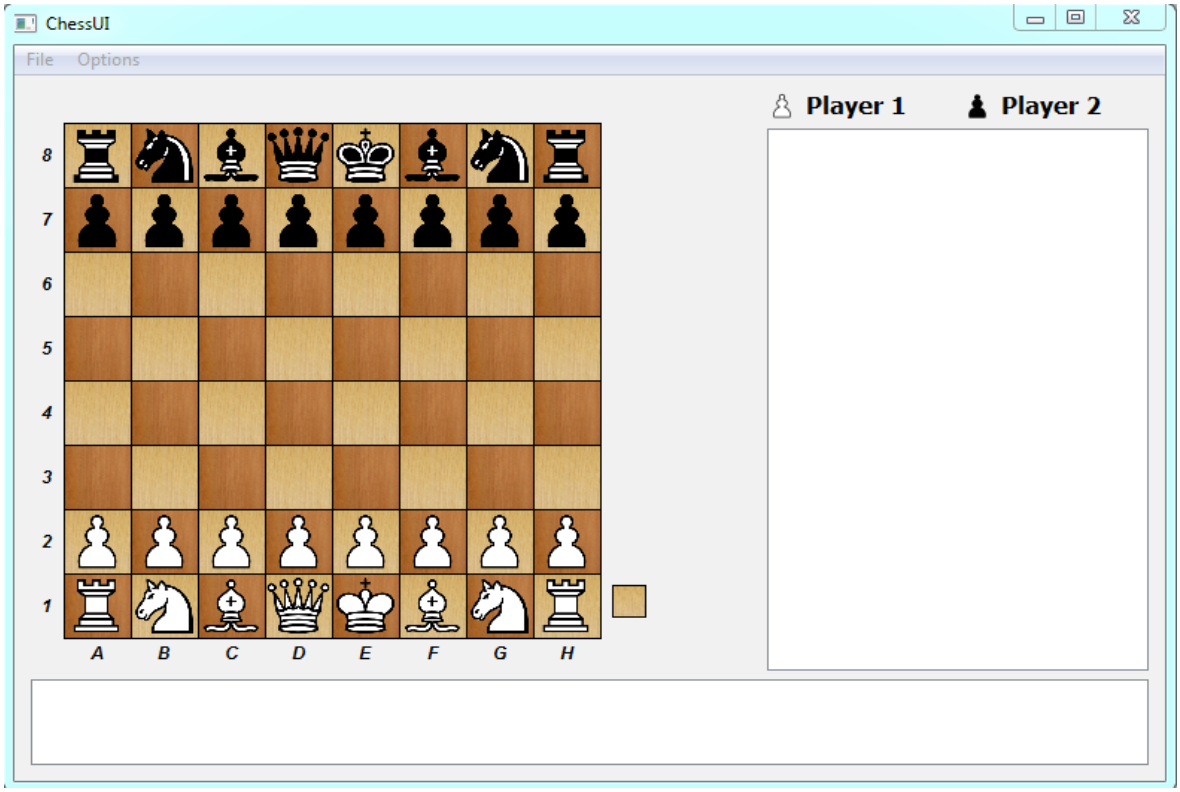
spremenimo možnosti pogona. Najpogosteje to storimo z ukazom tipa *„setoption name <ime> value <vrednost>”* Po tem, ko pošljemo vse nastavitve, moramo obvezno preveriti stanje pogona z ukazom *„isready”*. Ukaz vedno pošiljamo, dokler pogon ne odgovori z *„readyok”*. S tem je vse pripravljeno za začetek igre. Igro začnemo z ukazom *„ucinewgame”*. Temu ukazu mora obvezno slediti ukaz *„isready”*, saj lahko odziv na ta ukaz traja nekaj časa. Za določitev poteze moramo pogonu poslati še dva ukaza. Prvi je ukaz tipa *„position startpos moves <poteza1> <poteza2> ...”*, ki vsebuje poteze do sedaj opravljene na obeh straneh. V primeru, da se je igra šele začela, pošljemo samo *„position startpos”*.

Poteze so formata *„a1a2”*. Takšna implementacija zagotovi, da bo pogon vedno vedel, kakšen je položaj figur na šahovnici, tudi če sam ne shranjuje vrednosti. Nadaljujemo lahko z različnimi variacijami ukaza *„go ...”*. V programu smo uporabili predvsem ukaza *„go movetime x”* in *„go depth x”*, ki pogonu ukažeta iskanje do izteka določenega časa, oz. iskanje do določene globine. Ti vrednosti je mogoče nastaviti v nastavitvah vmesnika (več v 4.1). Do prekinitve iskanja pogon vrača vrstice s podatki, ki se začnejo z besedo *„info”*, ko se iskanje prekine, pa pošlje še vrstico *„bestmove <poteza1> ponder <poteza2>”*. Prvi podatek je izbrana najboljša poteza, drugi pa pričakovan odgovor nasprotnega igralca. V primeru da igralec opravi to potezo, lahko pogon v času nasprotnikove poteze začne iskati svojo. Po tem, ko pogon pošlje ta podatek, je pripravljen na iskanje naslednje poteze (Slika 3.1) [16].

```
uci
id name Strelka 5.5
id author Jury Osipov
option name Hash type spin default 128 min 4 max 2048
uciok
ucinewgame
isready
readyok
position startpos
go depth 3
info depth 2
info depth 2 score cp 6 time 1 nodes 61 nps 61000 pv e2e4 g8f6
info depth 3
info depth 3 score cp 45 time 3 nodes 468 nps 156000 pv e2e4 g8f6 b1c3 b8c6 g1f3
 e7e6 d2d4 b7b6
info depth 3 score cp 45 time 4 nodes 889 nps 222250 pv e2e4 g8f6 b1c3 b8c6 g1f3
 e7e6 d2d4 b7b6
bestmove e2e4 ponder g8f6
```

Slika 3.1: Demonstracija zagona pogona Strelka izven uporabniškega vmesnika

4 Implementiran grafični vmesnik



Slika 4.1: Grafični vmesnik ob zagonu

Grafični vmesnik (Slika 4.1) je napisan v angleščini zaradi možnosti širše uporabe, vključuje pa tudi prevod v slovenščino. Jezik je samodejno izbran glede na jezik okolja, v katerem se aplikacija zažene. Če prevod ne obstaja, se uporabi privzet jezik (angleščina). V osnovi je vmesnik razdeljen na 4 vidne dele. Zgoraj je menijska vrstica z možnostmi za spremembo izgleda vmesnika, ter izbiro načina igre (proti drugemu igralcu, oz. proti pogonu), levo šahovnica, na desni strani izpis igralcev ter okno s podatki o potezah, spodaj pa prikaz informacij, ki jih pogon posreduje našemu grafičnemu vmesniku.

Poleg vidnih delov, pa je pomembna funkcija vmesnika tudi implementirana povezava s pogoni, ki podpirajo protokol UCI. Več o tem v 4.5.

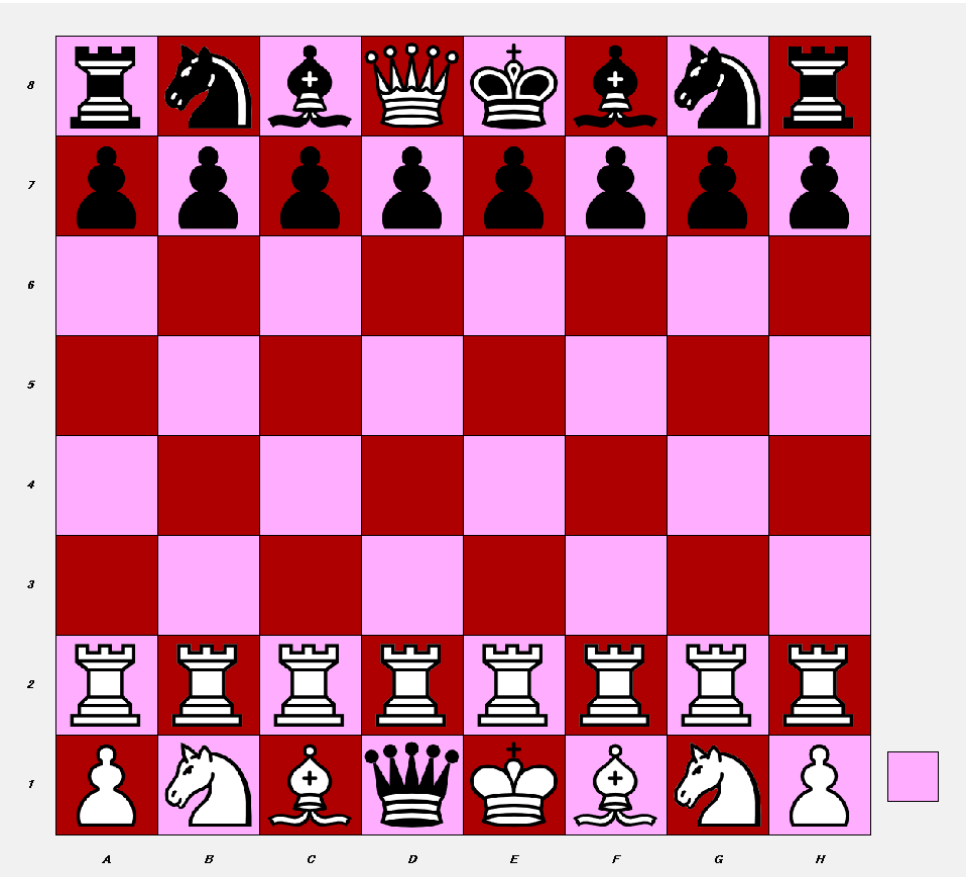
Osnovna koda za šahovnico in preverjanje veljavnosti potez je bila vzeta iz projekta Chess

Symphony [5] s soglasjem avtorja.

4.1 Vrstica z menijem

Vrstica z menijem je razdeljena na „*File*", „*Options*" ter *„About"*. Pod prvim lahko zaženemo igro proti drugemu igralcu ali računalniku, ali pa zapremo program. Druga izbira vsebuje nastavitve vmesnika ter mogoče spremembe izgleda:

- izbira barve igralca (beli ali črni),
- izbira poti do pogona (relevantno le, če igramo proti pogonu),
- določitev globine iskanja (relevantna le, če preišče vse veje do določene globine pred iztekom časa),
- določitev časa iskanja,
- sprememba barv polj (barva iz palete, lahko pa tudi slikovne datoteke) (Slika 4.2),



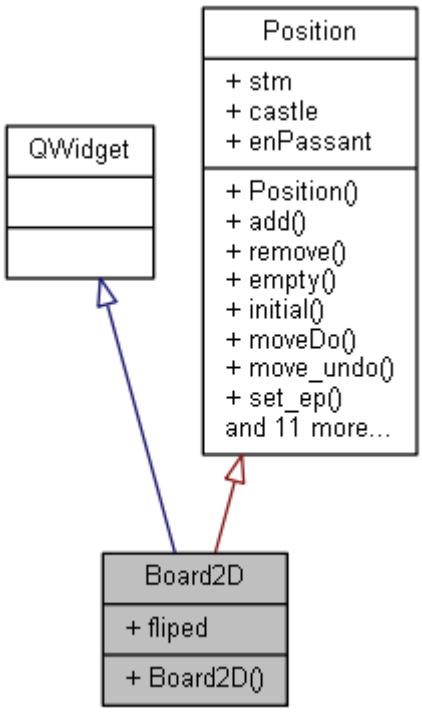
Slika 4.2: Demonstracija možnih sprememb izgleda šahovnice

- sprememba figur (izbira poti do figur; figure morajo biti poimenovane enako kot so v privzeti mapi (*white pawn* = wp, *black knight* = bn, itn.)) in
- sprememba fonta uporabljenega na šahovnici.

Nastavitve so shranjene v xml datoteki, ki je ob spremembi vedno posodobljena. Ohranitev nastavitev bi lahko dosegli tudi z uporabo razreda QSettings, ki je Qt razred, namenjen prav temu. Uporabljen ni bil iz preprostega razloga; implementacija shranjevanja v xml datoteko je bila implementirana, preden smo izvedeli za ta priročen razred. Ker ohranjevanje nastavitev deluje, smo se odločili, da ponovna implementacija za QSettings ni smiselna.

4.2 Šahovnica

Je najpomembnejši del vmesnika. Šahovnica je vizualna predstavitev premikov, ki jih opravi igralec/pogon. V kodi je predstavljena z razredom Board2D (Slika 4.3), ki je podedovan od razredov QWidget (osnovni razred za objekte v Qt) ter Position, ki skrbi za začetno pripravo šahovnice (*initial()*), dodajanje (*add(...)*) in odstranjevanje (*remove(...)*), preverjanje zasedenosti polj (*empty()*), določanje tipa in veljavnosti potez (*moveDo(...)*), en passant (*set_ep()*), zaznavanje šaha ipd. Poleg tega razred Position vsebuje tudi ogrodje za razveljavitev premikov (*move_undo(...)* ipd.), ki pa še ni bila implementirana [1].



Slika 4.3: Razredni diagram razreda Board2D z le javnimi podatki

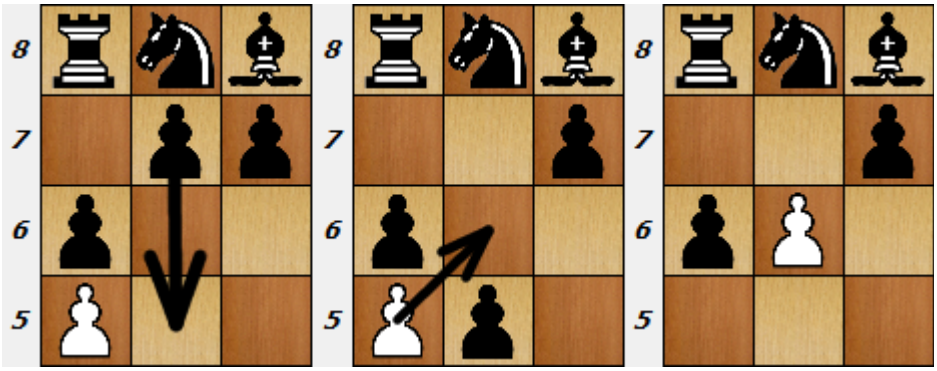
Posamezno polje je predstavljeno z numerično strukturo (*angl. „enumeration"*) Square. Polja so sproti preračunana, tako da spreminjanje velikosti okna ne bi smelo voditi do napak v premikanju figur.

Razlika med Board2D in QWidget je predvsem v tem, da Board2D poleg parametrov, ki jih

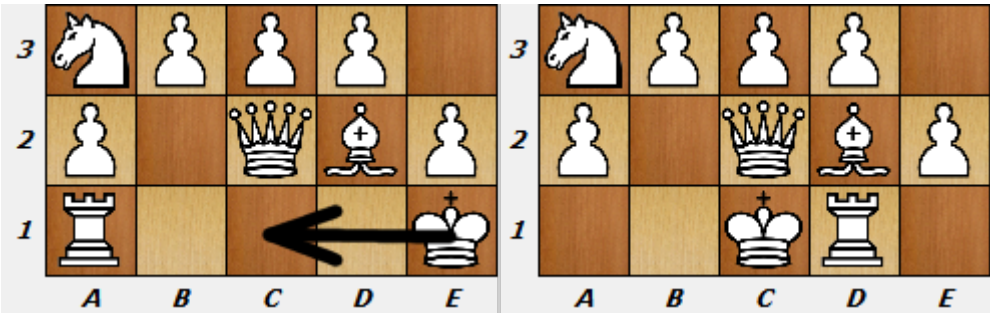
vsebuje QWidget v konstruktorju, sprejme še kazalec na razred Settings, v katerem so shranjene privzete nastavitve šahovnice.

Na šahovnici so vedno omogočene le dovoljene poteze, ki se za posamezno figuro ob kliku nanjo obarvajo modro. V primeru, da je kralj napaden, se njegovo polje obarva rdeče in s tem opozori igralca, da so mogoče le poteze, ki prekinejo šah. Če takšnih potez ni, program igralca obvesti o koncu igre. Obstajajo tudi tri posebne poteze, ki ne sledijo osnovnim pravilom premikanja figur. To so en passant (Slika 4.4), rokada (Slika 4.5) ter promocija kmeta v drugo figuro, ko doseže zadnje polje na šahovnici.

Okence desno od šahovnice sporoča, kateri igralec je na vrsti (barva belega polja pomeni, da je na vrsti igralec z belimi figurami in barva črnega polja, da je na vrsti igralec s črnimi figurami).



Slika 4.4: En passant

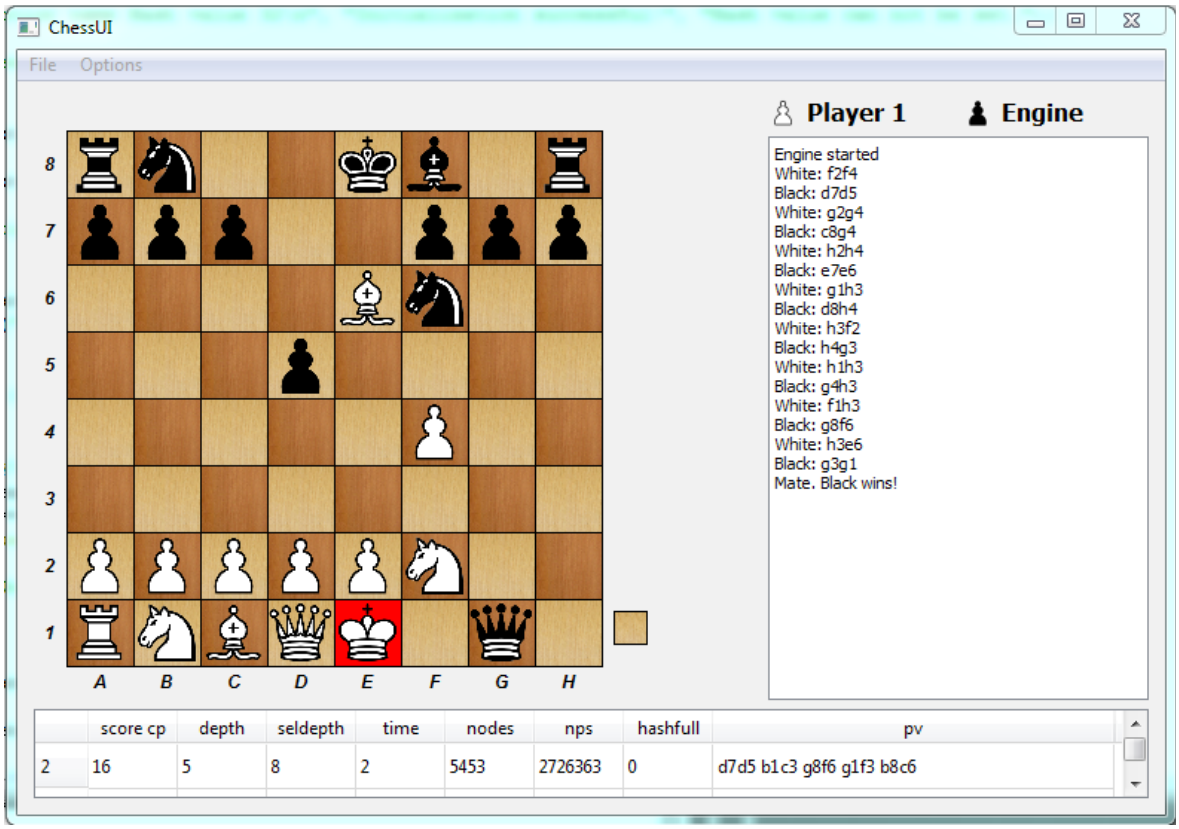


Slika 4.5: Rokada

4.3 Podatki o poteku igre

Okno na desni strani (Slika 4.7) vsebuje osnovne podatke o igri. Sporoči nam ime pogona (v primeru, da pogon ne pošlje svojega imena, je poimenovan „*Engine*“) ter izpisuje

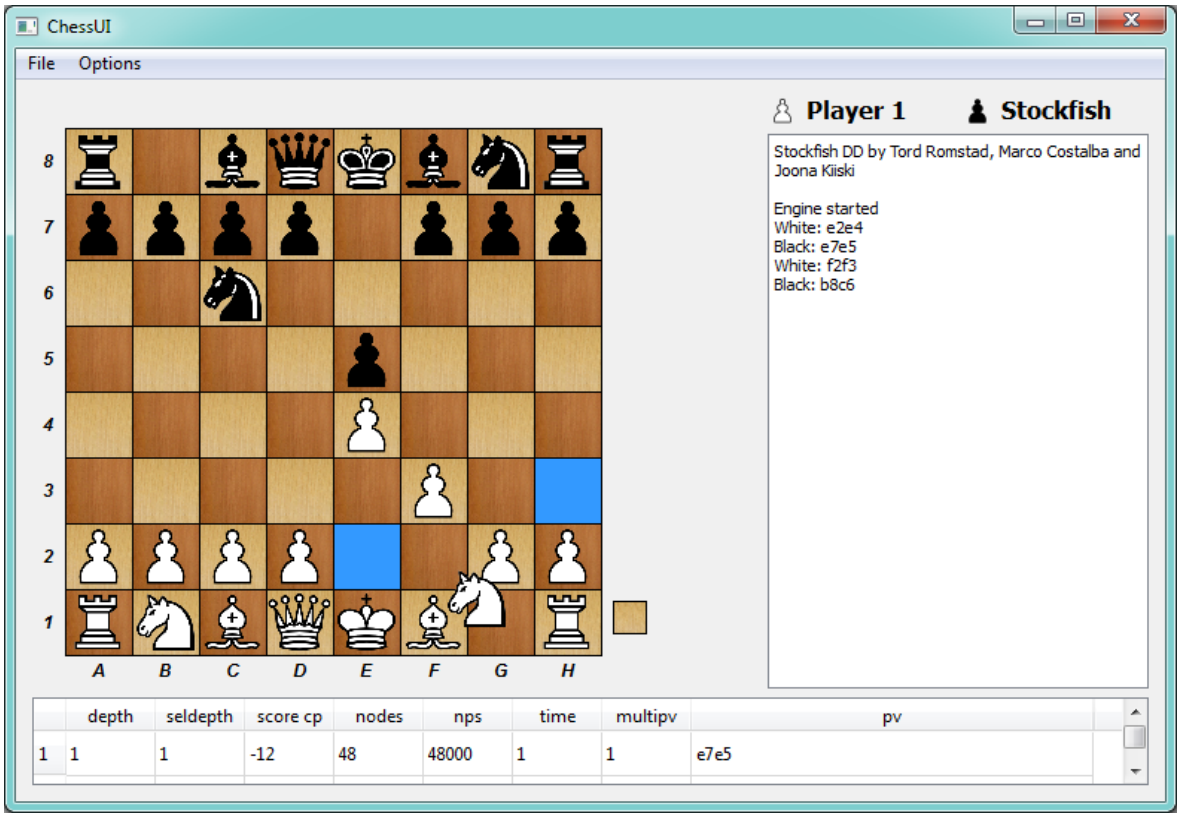
pomembne spremembe, kot je zgodovina odigranih potez ter konec igre.



Slika 4.6: Konec igre proti pogonu BBChess

4.4 Informacije pogona

V oknu na dnu (Slika 4.6) so podatki, ki jih pogon posreduje naši aplikaciji in se med posameznimi pogoni razlikujejo. Zaradi tega se oblika tabele določi ob zagonu pogona. To dosežemo tako, da preberemo vsako vrstico, ki ne nosi končne poteze (identificirana z „*bestmove*") ter s pomočjo regularnih izrazov ločimo imena, ki jih zapišemo v naslovno vrstico tabele ter podatke, ki jih vnesemo v tabelo. To pa v vseh primerih ne zadošča, saj nekateri pogoni ne vrnejo vedno vseh podatkov. Strelka na primer v prvi iteraciji na vsaki globini vrne le globino, za tem pa polno vrstico podatkov, medtem ko Stockfish, na katerem smo v osnovi gradili, v vsaki vrstici vrne vse podatke. Zaradi tega smo se odločili, da preverjamo, če je število prebranih elementov večje od števila stolpcev v tabeli. V primeru, da se to zgodi, v tabelo dodamo manjkajoče stolpce. Pri pogonu, ki je nekonsistenten pri pošiljanju podatkov, bi to seveda lahko vodilo do napak, vendar smo se odločili, da je to najboljša rešitev.

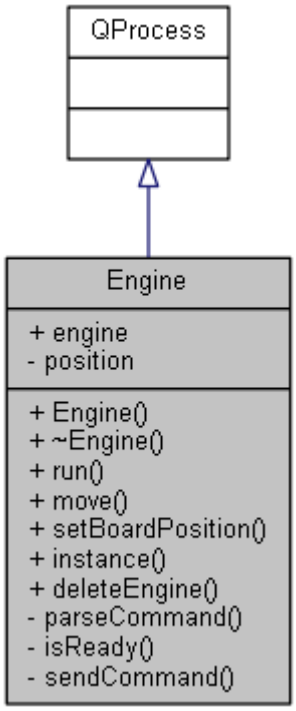


Slika 4.7: Grafični vmesnik med igro s pogonom Stockfish

Različni pogoni prikazujejo različne podatke. Glavna podatka, ki naj bi ju vrnili vsi pogoni, sta globina (*angl. „depth"*) ter niz potez glavne variante (*angl. „Principal Variation"*). Pogon Stockfish (Slika 4.7) vrne še podatke o maksimalni dinamični globini iskanja, ki je odvisna od pregledanih pozicij (*angl. „seldepth"*), oceno pozicije v desetinkah vrednosti kmeta (*angl. „score cp"*), število preiskanih pozicij (*angl. „nodes"*), število preiskanih pozicij na sekundo (*angl. „nodes per second*) ter število glavnih variant, ki jih preiskuje (stolpec *„multipv*"). Običajno preiskuje eno glavno varianto, v primeru analize določene pozicije, pa se lahko to število poveča. Tako lahko uporabnik vidi, kako dobra so različna nadaljevanja igre.

4.5 Povezava s pogonom

Drugi najpomembnejši del programa, takoj za vizualno predstavitvijo šahovnice, je povezava programa z zunanjim pogonom. To je v večjem delu opravljeno v razredu Engine (Slika 4.8), ki lastnosti podeduje od razreda QProcess. V tem razredu poteka vsa



Slika 4.8: Razredni diagram razreda Engine

komunikacija med vmesnikom ter naloženim pogonom. Zagon pogona se prične v metodi *„instance()”*, ki preveri, če pogon že obstaja. V primeru, da pogon že obstaja, vrne kazalec na instanco pogona, če ne, pa ustvari novega in vrne kazalec nanj. Po klicu te funkcije lahko varno kličemo metodo *„run()”*, ki poskusi zagnati pogon in vrne *„true”*, če je zagon uspešen, oz. *„false”* , če pogona ni mogoče zagnati. V tem primeru v vmesniku prikažemo razlog za neuspeh. Spremenljivka *„position”* tipa *„std::string”* vsebuje seznam že opravljenih potez, ki jih pogonu posreduje vmesnik prek metode *„setBoardPosition(...)”*. Za uporabo *„std::string”* in ne Qt-jevega QString smo se odločili, ker metode razreda QProcess pričakujejo parametre v formatu *„const char *”* in je pretvorba tako preprostejša. Tu so še metoda *„move(...)”*, ki skrbi za interpretacijo poteze ter njeno posredovanje vmesniku ter *„deleteEngine()”*, ki poskrbi, da se pogon vedno zapre. Ostanejo še zasebne metode *„parseCommand(...)”* za razpoznavo niza s potezo, *„sendCommand(...)”* za pošiljanje ukazov in preverjanje odziva nanje in *„isReady()”*, ki preveri, če je pogon pripravljen na naslednji ukaz [7].

Ostale metode, ki so navedene v nadaljevanju, so podedovane od razreda QProcess, ki je razred za komunikacijo z zunanjimi aplikacijami (v našem primeru pogonom) [8]. Omogoča zagon druge aplikacije znotraj našega programa ter izjemno preprosto komunikacijo. Za komunikacijo je na voljo več metod, v tem programu pa uporabljamo predvsem dve. To sta „*write(...)*” ter „*readLine()*”. Seveda je treba poskrbeti še za sinhronizacijo. Za to sta na voljo „*waitForBytesWritten()*” ter „*waitForReadyRead()*”. Privzeto sta nastavljeni, da počakata 30 sekund, vendar je to v našem primeru bistveno preveč. Na srečo je čas čakanja mogoče spremeniti preprosto s tem, da v metodi vpišemo želen največji čas čakanja v milisekundah. Da ne pride do napak, je pred branjem dobro klicati še metodo „*canReadLine()*”, ki vrne true, ko so podatki pripravljeni na branje ter false, ko podatkov ni [27].

Pogonu sporočimo pozicijo, za katero začne iskati potezo z ukazom „*position startpos*

moves ...", ki mu sledi seznam odigranih potez. Nato zaženemo iskanje z ukazom „*go depth x*" (Slika 4.9) za iskanje omejeno z globino, oz. „*go movetime x*" (Slika 4.10), za časovno omejeno iskanje. Časovno omejeno je navadno boljša izbira, saj imamo boljši nadzor nad trajanjem iskanja ne glede na zmogljivost računalnika, medtem ko se čas iskanja do iste globine lahko razlikuje zaradi različnih računalnikov in tudi pozicij. Seveda pa časovno omejeno iskanje na različnih računalnikih ne da vedno enakih rezultatov.

```
//depth search
command = std::string("go depth ") + QString::number(depth).toStdString() + "\n";
write(command.c_str());
waitForBytesWritten();
```

Slika 4.9: Izsek kode za iskanje v globino

```
//timed search
command = std::string("go movetime ") + QString::number(time).toStdString() + "\n";
write(command.c_str());
waitForBytesWritten();
```

Slika 4.10: Izsek kode za časovno omejeno iskanje

Preostanek dela opravi pogon. Razred Engine prejema podatke, in jih prek signalov (*angl. „signals"*) in rež (*angl. „slots"*) pošilja v glavni razred uporabniškega vmesnika, ki jih obdela in izpiše na ustreznih mestih. Signali in reže so izjemno uporabna funkcionalnost. V bistvu sta signal in reža metodi dveh funkcij. Signal je metoda v razredu, iz katerega podatke želimo poslati, reža pa metoda v razredu, ki podatke sprejme. Deluje pa preprosto tako, da ko je signal v prvem razredu klican, se v drugem razredu kliče reža, ki je vezana nanj. Prek signala tako njegove parametre pošljemo v drug razred brez potrebe po dodatnem povezovanju razredov. V primeru (Slika 4.11) iz razreda Engine v razred ChessUI prek signalov in rež pošljemo tekst: „Tekst za izpis".

Razred Engine bere in pošilja vrstice, dokler ne pride do vrstice, ki se začne z „*bestmove*" in vsebuje podatke o potezi, ki ga pogon želi opraviti. Ta podatek nato pošlje v razred Board2D, kjer se ta poteza tudi opravi ter grafičnemu vmesniku, ki poteze izpiše v oknu opisanem v poglavju 4.3.

```
//chessui.h
private slots:
    void showRightText(const QString &message);

//engine.h
signals:
    void updateRightText(const QString &tmp);

//chessui.cpp
connect(engine, SIGNAL(updateRightText(const QString&)), this, SLOT(showRightText(const QString&)));

//engine.cpp
updateRightText("Tekst za izpis.");
```

Slika 4.11: Demonstracija delovanja signalov in rež

Za konec ostane še zadnji, dostikrat spregledan vendar vseeno pomemben problem. Zagotoviti moramo, da se pogon ob zaustavitvi programa vedno izklopi. To dosežemo z ukazom *„quit”*. Priporočljivo je, da se prepričamo, da se je pogon res ustavil in zaprl, preden zaustavimo program. V nasprotnem primeru se lahko zgodi, da pogon še vedno teče v ozadju in porablja računalniške vire, čeprav ga ne potrebujemo več.

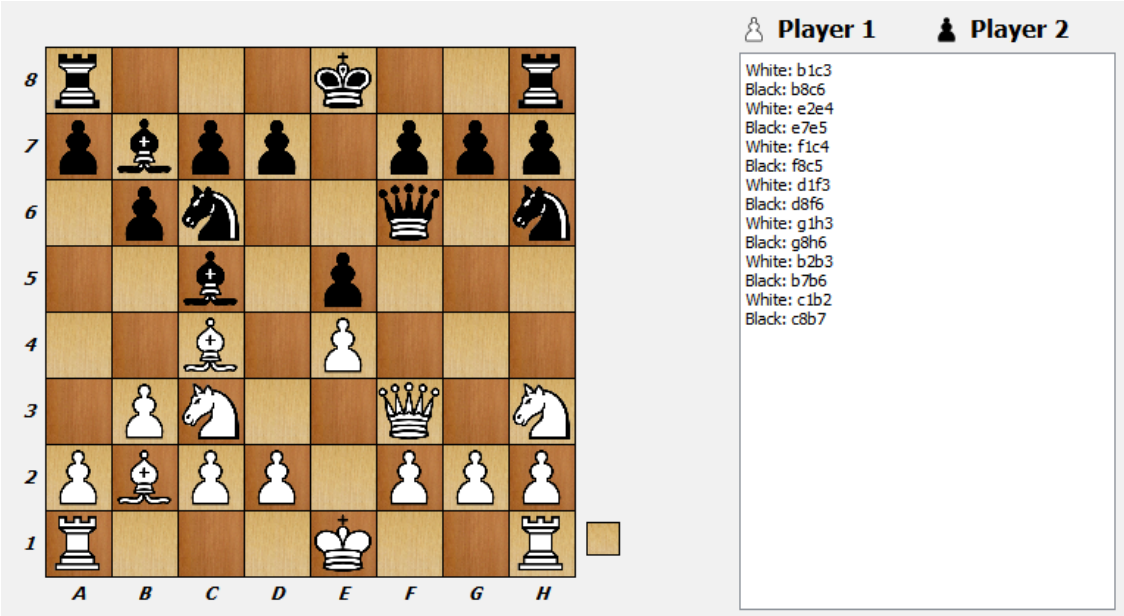
5 Testiranje

5.1 Načrt testiranja

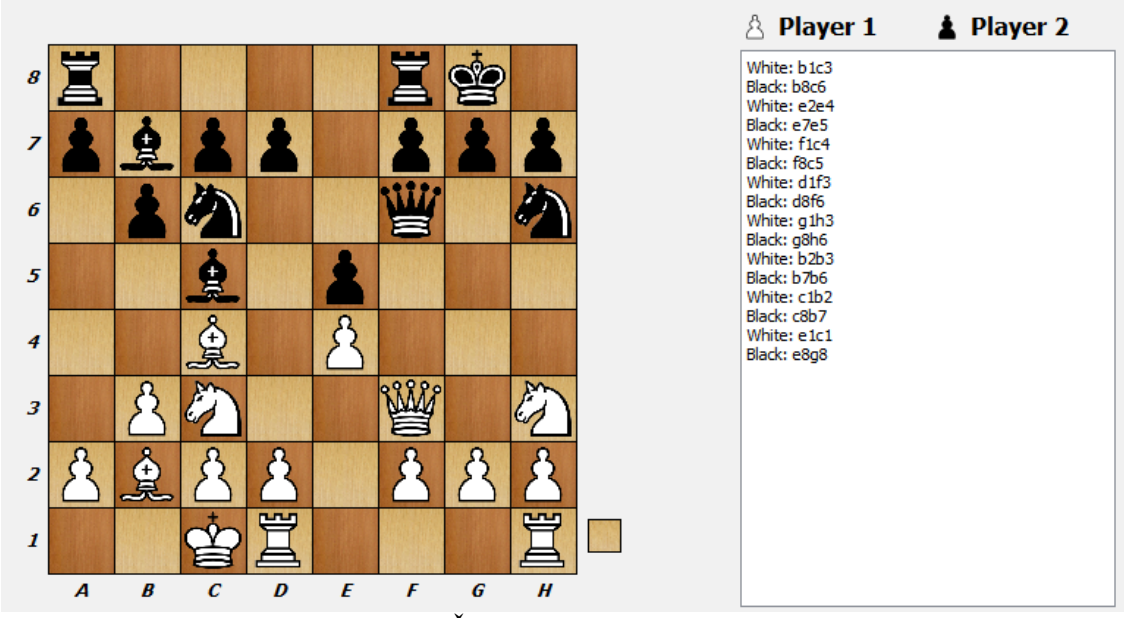
Testiranje bomo izvedli v treh fazah. Najprej bo funkcionalnost programa testirana brez pogona, da se preveri pravilno delovanje samega vmesnika. Zatem bo vmesnik popolnoma testiran s pomočjo pogona Stockfish. Ob zadovoljivih rezultatih bomo testiranje razširili še na nekaj drugih pogonov. V teh primerih bo testirana predvsem kompatibilnost pogonov z vmesnikom. Razlog za to odločitev je, da pogoni, čeprav uporabljajo enak protokol, vseeno ne pošiljajo ukazov v popolnoma enaki obliki in tako lahko pride do napak v interpretaciji odgovora pogona. Od končnega izdelka se pričakuje, da bo načeloma združljiv z vsemi pogoni, ki komunicirajo s pomočjo protokola UCI.

5.1.1 Vmesnik

1. Zaženemo vmesnik. Privzeta nastavitev igre je igralec proti igralcu. Navedene poteze so poteze belega igralca, črni pa, če ni navedeno drugače, na svoji strani opravlja identične poteze.
2. Pravilnost prepoznave neveljavnih potez [14][15]. A1 na A3, B1 na B3, C1 na C3, D1 na D3 ter E1 na E3. Figure se morajo vrniti na svoje mesto, polja za mogoče poteze se morajo pobarvati nazaj na prvotno barvo. V primeru, da figura nima prostih mest, na katera bi se lahko premaknila, je ne moremo pobrati.
3. Veljavnost potez. Konja z B1 premaknemo na C3. Kmeta z E2 premaknemo na E4. Tekača z F1 na C4. Kraljico z D1 na F3. Konja z G1 na H3. Kmeta z B2 na B3. Tekača s C1 na B2 (Slika 5.1). Za vse figure spremljamo, če so polja, ki se obarvajo kot polja potez, veljavna. Na beli strani poberemo kralja. Obarvati se morajo polja C1, D1, D2, F1, G1. Prestavimo ga na polje C1. Trdnjava se mora iz polja A1 prestaviti na D1. Na črni strani poberemo kralja. Obarvati se morajo zrcalna polja kot na beli. Kralja premaknemo na polje G8. Trdnjava na H8 se mora prestaviti na F8 (Slika 5.2).

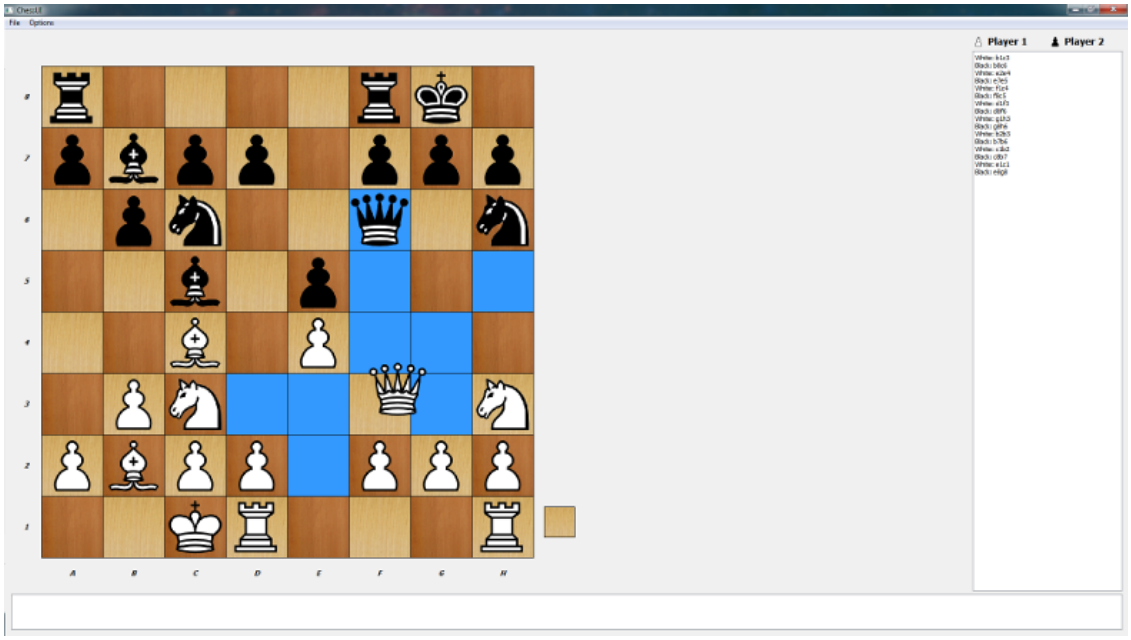


Slika 5.1: Stanje na šahovnici po izvedbi osnovnih potez



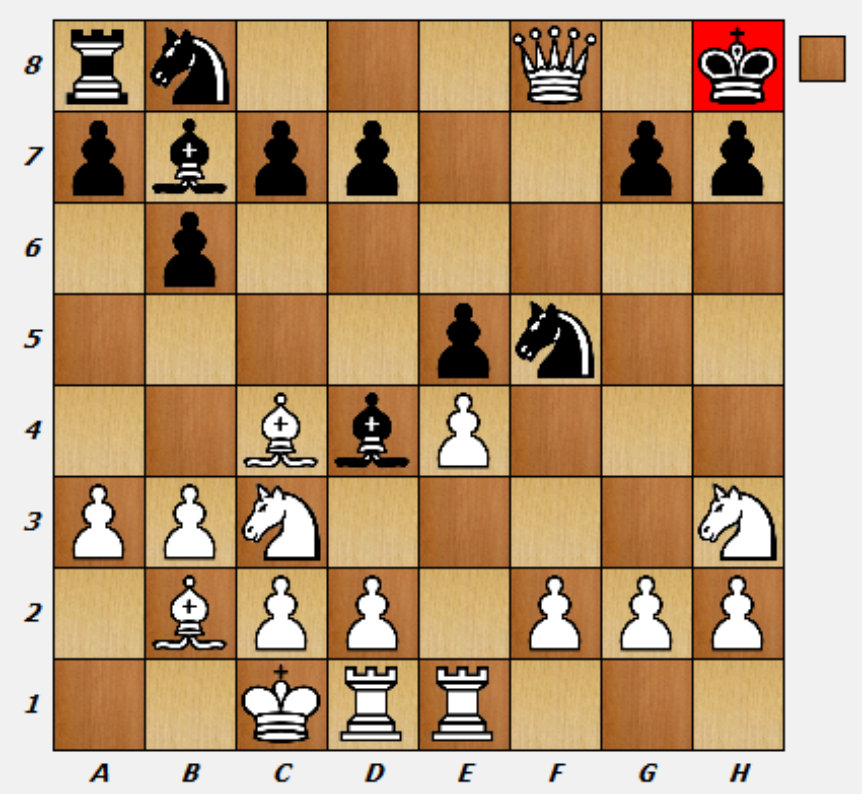
Slika 5.2: Šahovnica po izvedbi rokad

4. Spreminjanje velikosti. Okno povečamo na velikost celotnega zaslona. Poberemo kraljico. Obarvati se morajo polja E2, D3, E3, G3, F4, G4, F5, H5 in F6 (Slika 5.3). Prestavimo jo na polje F4. Velikost spremenimo nazaj na prvotno. Okno nato poljubno povečamo ali zmanjšamo.



Slika 5.3: Mogoči premiki bele kraljice v povečanem oknu

5. Napadanje. Z belo kraljico na F4 poberemo črno kraljico na F5. Črni prestavi konja s C6 na B8. Bela kraljica pobere kmeta na F7. Polje, na katerem je kralj, se mora obarvati rdeče, kar nakazuje, da je pod napadom. Črni poskusi pobrati kmeta na A7, vendar ga ne more. Opravi lahko le poteze, ki bi prekinile šah. Tako kralja prestavi z G8 na H8. Beli prestavi kmeta z A2 na A3. Črni prestavi tekača s C5 na D4. Beli prestavi trdnjavo s H1 na E1. Črni prestavi konja s H6 na F5. Beli prestavi kraljico s F7 na F8. Črni prestavi konja s H6 na G8, da prekine šah. Beli s kraljico na F8 pobere konja na G8. Na desni se izpiše, da je igra končana in da je beli zmagal. Premikanje figur ni več mogoče (Slika 5.4).
6. Nastavitve. V meniju kliknemo na „*Options->Settings*”. Spremenimo barvi šahovnice na poljubni barvi iz palete. S klikom na „*Ok*” se morajo polja šahovnice pobarvati z ustreznima barvama.
7. Ohranjanje nastavitev. Zapremo program in ga ponovno zaženemo. Vse spremembe (z izjemo velikosti okna) morajo še vedno biti vidne.



Slika 5.4: Končna pozicija šahovnice po končanem testiranju

5.1.2 Stockfish

Stockfish je trenutno najboljši pogon za šah. Poleg tega je vredno še omeniti, da je podprt za platforme Windows, Mac OS X, Anroid ter Linux in ga tako lahko uporabljamo na večini naprav [22].

1. Zopet odpremo nastavitve in izberemo „*Play against computer as Black*”. Preverimo, da pot v oknu „*Engine*” vodi do izvršljive datoteke pogona Stockfish (32 bit). Nato v menuju „*File*” izberemo „*New game against computer*”. Zagnati se mora nova igra, črni morajo biti na spodnji strani šahovnice. Na desni strani je izpisano ime pogona (če je na voljo), ime belega igralca pa se spremeni v ime pogona, oz. „*Engine*”, če ime pogona ni navedeno. Vrstica „Engine started” sporoči, da se je pogon pravilno zagnal in je pripravljen za komunikacijo. Računalnik opravi prvo potezo.
2. Za nadaljnje poteze ni nujno, da bodo vedno enake, saj lahko računalnik zaradi

razpoložljivih virov na dveh računalnikih igra drugače. Premaknemo kmeta s C2 na C4. Ko računalnik opravi svojo potezo, se polja, ki so bila z našimi potezami obarvana, pobarvajo nazaj na prvotno barvo. Prestavimo kmeta z B2 na B4. Nato prestavljamo kmete na desni strani po eno polje naprej, dokler nas računalnik ne matira. Če vse deluje pravilno, nas bo matiral v nekaj potezah.

3. Ob koncu igre preverimo še pravilnost izpisov potez na desni strani, ter pravilnost prepoznanega konca igre. Pod šahovnico je še tabela, ki opisuje proces mišljenja računalnika. V njej je naenkrat vidna ena vrstica, vendar se lahko na desni strani premikamo med vrsticami.
4. V nastavitvah spremenimo čas mišljenja na 5 sekund, zaženemo novo igro in testiramo, če to računalnik upošteva. V primeru, da konča prehitro, povečamo še globino iskanja.

5.1.3 Drugi pogoni

Ponovimo postopek kot pri prejšnji točki, s tem, da v nastavitvah izberemo pot do drugega pogona. Program bomo testirali še s pogoni BBChess (1.3b 64 bit) [4], Rybka (232a) [21] ter Strelka (5.5) [18].

5.2 Rezultati testiranja

Program se je v večini primerov odzival po pričakovanjih. Pri šahiranju včasih nastopi težava, da so mogoče neveljavne poteze, vendar le občasno. Na podlagi rezultatov lahko sklepamo, da vmesnik deluje za vsako solidno implementacijo protokola UCI.4

6 Zaključek

Izdelali smo popolnoma delujoč grafični vmesnik za igranje šaha v programskem okolju Qt in z uporabo protokola UCI. V povezavi med vmesnikom in protokolom ni bilo zaznanih napak. Prav tako tudi spreminjanje velikosti šahovnice ne povzroča težav, saj so ob vsaki spremembi lokacije in velikosti polj ponovno preračunane.

Testiranje samega igranja je prineslo zadovoljive rezultate. Nedovoljeni premiki niso mogoči, polja z dovoljenimi potezami se pravilno obarvajo, ko figuro poberemo, ko jo odložimo, se vrnejo na prvotno barvo. Enako velja za situacijo, ko je napaden kralj in se obarva njegovo polje.

Pri izdelavi programa smo se soočili z nekaj problemi, kot na primer edinstveni premiki konja, ki za razliko od ostalih figur ne uporablja polja vida (*angl. „line of sight"*), temveč lahko skoči na katerokoli polje, ki ustreza njegovemu vzorcu premikanja. Poseben primer je tudi navidez najpreprostejša figura, kmet. Kmet je namreč edinstven v tem, da ne napada v isti smeri kot se premika, poleg tega pa zanj obstajata še dve posebni potezi; en passant ter promocija, ki ju je seveda potrebno obravnavati posebej.

Poleg samega delovanja šahovskega vmesnika smo se soočili še s problemom povezovanja z različnimi pogoni, ki jim je skupno le, da komunicirajo s pomočjo protokola UCI. Zaradi tega je potrebno zagotoviti, da so ukazi vedno interpretirani pravilno, v primeru napačne interpretacije, pa je napako potrebno odkriti in odpraviti. Do hudih napak v delovanju lahko vodi tudi preprosta napaka, da ukazu pozabimo dodati znak za konec vrstice, saj ga brez tega pogon ne izvede.

6.1 Možnosti za nadaljnje delo

Kljub temu, da je program zaključena celota, ki omogoča igranje šaha proti različnim pogonom ter med igralci, še vedno obstajajo možnosti za izboljšave, kot na primer.

- igra pogona proti pogonu,
- možnost, da pogon igralcu predlaga najboljšo potezo v dani situaciji,

- možnost razveljavljanja potez (ogrodje je že pripravljeno),
- omejitev časa igranja,
- podpora lokalne igre prek dveh računalnikov, oz. prek spleta,
- podpora naprednejših ukazov, ki jih podpirajo nekateri pogoni,
- podpora protokola Xboard/WinBoard,
- prevod programa za platformo Linux in
- prevod v več jezikov.

Literatura

[1] Balagurusamy , E., Object Oriented Programming With C++, Tata McGraw-Hill, 2008

[2] Bitbucket. Dostopno na: https://bitbucket.org [30. 07. 2014]

[3] Blanchette, J., Summerfield, M., C++ GUI Programming with Qt 4 (2nd Edition), Prentice Hall, 2008

[4] Bošković, B., BBChess. Dostopno na: http://labraj.uni-mb.si/en/Chess_program_BBChess [22. 07. 2014]

[5] Bošković, B., Chess Symphony. Dostopno na: http://sourceforge.net/p/ferichess/code/HEAD/tree/ [10. 08. 2014]

[6] Cutechess. Dostopno na: http://cutechess.com/ [14. 07. 2014]

[7] Eckel, B., Thinking in C++ (2nd Edition) Volume 1, 2000

[8] Ezust , A., Ezust, P., Introduction to Design Patterns in C++ with Qt (2nd Edition), Prentice Hall, 2011

[9] GCC. Dostopno na: https://gcc.gnu.org/ [30. 07. 2014]

[10] Heni, M., Beckermann, A., Open Source Game Development: Qt Games for KDE, PDA's and Windows, Charles River Media, 2005

[11] Huber, R., Meyer-Kahlen, S., UCI. Dostopno na: http://chessprogramming.wikispaces.com/UCI [14. 07. 2014]

[12] Intervju s Fabienom Letouzeyjem, stran 7, 96, April 05, 2005. Dostopno na: http://www.playwitharena.com/?Newsticker:Archive_7, [21. 07. 2014]

[13] Intervju z Robertom Hyatom. Dostopno na: http://www.top-5000.nl/int/crafty.htm, [21. 07. 2014]

[14] King D., Šah od prvih potez do mata, Didakta, 2002

[15] Learn to Play Chess, Dostopno na: http://www.chess.com/learn-how-to-play-chess, [11. 08. 2014]

[16] Lovász, L., Iványi, A., Algorithms of Informatics Volume 1, 2007

[17] MinGW. Dostopno na: http://www.mingw.org/ [30. 07. 2014]

[18] Osipov, Y., Strelka. Dostopno na: http://www.chess.com/download/view/strelka-55 [22. 07. 2014]

[19] Qt Chess. Dostopno na: https://github.com/john812/Qt-Chess.git [11.03.2013]

[20] Qt. Dostopno na: http://qt-project.org/ [14. 07. 2014]

[21] Rajlich, V., Rybka. Dostopno na: http://www.rybkachess.com/ [22. 07. 2014]

[22] Romstad, T., Costalba, M., Stockfish. Dostopno na: http://stockfishchess.org/ [14. 07. 2014]

[23] Rudolf, M., ChessX. Dostopno na: http://chessx.sourceforge.net/ [14. 07. 2014]

[24] Slika ChessX. Dostopno na: http://chessx.sourceforge.net/images/chessx08-vista.png [08. 08. 2014]

[25] Slika Qt Chess. Dostopno na: http://a.fsdn.com/con/app/proj/qtchess/screenshots/321739.jpg [08. 08. 2014]

[26] Slika Xboard. Dostopno na: http://www.gnu.org/software/xboard/whats_new/4.5.0/Spartan.png [08. 08. 2014]

[27] Timothy, B., Introduction To Object-Oriented Programming, 3/E, Addison-Wesley, 2008

[28] Torvalds, L., Git. Dostopno na: http://git-scm.com/downloads [30. 07. 2014]

[29] Xboard. Dostopno na: http://www.gnu.org/software/xboard/ [22. 07. 2014]

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

I Z J A V A O A V T O R S T V U

Spodaj podpisani Martin Konečnik

z vpisno številko E1043251

sem avtor diplomskega dela z naslovom: Grafični vmesnik za igranje šaha

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr., Borka Boškovića

  in somentorstvom (naziv, ime in priimek)

  red. prof., Janeza Bresta

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne _____

Podpis avtorja:

_____

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

Podpisani mentor:

doc. dr. Borko Bošković

in somentor:

red. prof. Janez Brest

Izjavljava, da je študent

Ime in priimek: Martin Konečnik

Št. indeksa: E1043251

Na programu: Računalništvo in informacijske tehnologije (UN)

izdelal zaključno delo z naslovom:

Grafični vmesnik za igranje šaha

(naslov zaključnega dela v slovenskem in angleškem jeziku)

Chess Graphical User Interface

v skladu z odobreno temo zaključnega dela, Navodilih o pripravi zaključnih del in najinimi navodili.

Preverila in pregledala sva poročilo o plagiatorstvu.

Datum in kraj: Podpis mentorja:

Datum in kraj: Podpis somentorja: