LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

Šaha taktikas treniņprogramma

KVALIFIKĀCIJAS DARBS

Autors: Krišjānis Kallings
Stud. apl. nr.: kk15060
Darba vadītājs: Dr. dat. Jānis Zuters

RĪGA 2017

Anotācija

Kvalifikācijas darba mērķis ir izstrādāt programmatūru, kas palīdzēs šahistiem uzlabot savas šaha taktiskās domāšanas spējas interaktīvi risinot šaha uzdevumus. Projekta ietvaros tika izveidota programmatūras prasību specifikācija un programmatūras projektējuma apraksts, tika veikta testēšana un tā tika aprakstīta dokumentācijā.

Programma tiek veidota lai šahisti varētu trenēt sava taktiskās domāšanas spējas ar datora palīdzību, tādejādi papildinot savu trenēšanās iespēju klāstu un paātrinot treniņu procesu, jo atšķirībā no ierastās šahistu trenēšanās metodes, kur šahist izpēta jau izspēlētas šaha spēles reāla galdiņa, šaha situācijas tiek veidotas grafiskajā vidē, aiztaupot laiku, kas tiek patērēts galdiņa uzstādīšanai, kā arī novēršot cilvēka kļūdas risku nepareizi salikt galdiņu.

Atslēgasvārdi: Šahs, Šaha taktika, Treniņu programmatūra, Taktiskā domāšana

Abstract

The aim of this qualification project is to develop a software that will help chess players improve their tactical thinking skills by solving interactive chess related problems. Within the framework of this project software requirements specification and design documents were created, unit testing was done for the source code and this testing was then documented.

This software is designed for chess players, so that they can train their tactical thinking skills with the help of computers, thereby diversifying their range of training methods and speeding up the training process. Unlike traditional learning methods where chess players explore already played chess games on real chess board, these situations are displayed in a graphical environment, sparing the time that is necessary for setting up the chess board, as well as eliminating the risk of human error by wrongly setting up the chess board.

Keywords: Chess, Chess tactics, Training software, Tactical thinking

Chess tactics training software

Saturs

1. Ievads

Šahs ir divu spēlētāju stratēģijas galda spēle, kuras aizsākumi meklējami 1500 gadu senā pagātnē. Laikiem ejot tā ir mainījusies un tikusi uzlabota, bet tās popularitāte ir palikusi nemainīga. Šo galda spēli ir iecienījuši un spēlējusi daudzi populāri cilvēki, piemēram, aktieris Čārlijs Čaplins, Kubas Republika prezidents Fidels Kastro, filmu režisors Stenlijs Kubriks un daudzi citi. Ilgus gadus šīs spēles apgūšanā un prasmju uzlabošanā tika izmantotas vienas un tās pašas metodes – lasīt mācību grāmatas un uz šaha galdiņa izspēlēt dažādas šaha lielmeistaru spēles. Līdz ar tehnoloģisko uzplaukumu arī šaha jomā ir parādījušās jaunas iespējas. Jau ilgus gadus dators ir pietiekami jaudīgs, lai spētu uz priekšu analizēt vairāk šaha gājienus, nekā cilvēks. Jau 1997. gadā dators "Deep Blue" spēja pieveikt pasaules šaha čempionu Gariju Kasparovu. Var droši teikt, ka tehnoloģijas var un ir papildinājušas iespēju analizēt šaha izspēles un trenēt šaha spēles prasmes.

Šī kvalifikācijas darba mērķis ir izstrādāt programmatūru, kas palīdzēs šaha spēlētājiem trenēt taktisko domāšanu interaktīvi risinot šaha uzdevumus. Izveidotā programmatūra būs lielisks palīgs šaha spēlētājiem un šaha skolotājiem, kas vēlas dažādot šaha apguves metodes. Izstrādātā programmatūra šahistam ļauj ātri un ērti trenēt šaha taktisko domāšanu, izmantojot šīs programmatūras intuitīvo grafisko lietotāja saskani, interaktīvi risinot šīs programmatūras piedāvātos līmeņus, kā arī ar iespēju pašam pievienot jaunus līmeņus.

2. Programatūras prasību specifikācija

2.1. Ievads

2.1.1. Nolūks

Programmatūras prasību specifikācijas nolūks ir specificēt šaha taktikas treniņu programmas prasības. Atsaucoties uz šajā dokumentā sniegto informāciju, tiks izstrādāta programmatūra. Šī programmatūras prasību specifikācija ir paredzēta programmatūras izstrādātājam, lai varētu viennozīmīgi un nepārprotami realizēt programatūras prasības, izstrādājot šo programmatūru.

2.1.2. Darbības sfēra

Izstrādātā programmatūra ir paredzēta šahistiem, kas vēlas uzlabot savas šaha prasmes, interaktīvi risinot šaha uzdevumus. Programmatūra piedāvā vairākus līmeņus ar dažādām šaha situācijām, kurās ir jāatrod pareizākais risinājums - gājiens vai gājieni, kas veicina spēlētāja materiālo pārsvaru vai uzvaru. Lietotājam ir iespēja pievienot arī pašam savus, jaunus līmeņus.

2.1.3. Definīcijas

Forsaita-Edvarda Notācija (FEN) – standarta notācija, kas tiek izmantota lai aprakstītu konkrētu figūru izvietojumu uz šaha laukuma kādā konkrētā laika brīdī. (Skatīt 2. pielikumu)

FEN tipa fails - datne ar paplašinājumu ".fen"

Šahs - galda spēle un sporta veids, ko spēlē divi cilvēki.

Rinda - ir šaha lauciņu kopa, kas ir izvietota horizontāli uz šaha laukuma (Skatīt 1. pielikumu).

Kolonna - ir šaha lauciņu kopa, kas ir izvietota vertikāli uz šaha laukuma (Skatīt 1. pielikumu).

Standarta šaha spēles sākuma pozīcija – šaha figūru izvietojums uz laukuma, pirms ir izdarīts pirmais gājiens.

Šaha laukums – īpašs rūtots kvadrātveida laukums, kas sastāv no 64 kvadrātveida lauciņiem (astoņās rindās un astoņas kolonnas). Lauciņi izkārtoti divās krāsās, gaišā un tumšā krāsā (Skatīt 1. pielikumu)

Algebriskā notācija - nosacītu apzīmējumu sistēma atsevišķu šaha gājienu, visas partijas vai konkrētas pozīcijas pierakstīšanai. Katrai figūrai ir savs apzīmējums. Dažādās valodās figūru apzīmējumi atšķiras, jo parasti apzīmējumam tiek izvēlēts kāds no figūras nosaukuma pirmajiem burtiem.

Baltais spēlētājs - spēlētājs, kurš veic gājienus ar baltajām figūrām

Melnais spēlētājs - spēlētājs, kurš veic gājienus ar melnajām figūrām

***en passant* –** specifisks gājiens ar bandinieku, kas var notikt tikai uzreiz pēc pretinieka izdarīta gājiena ar bandinieku, kas no savas sākuma pozīcijas pārvietojas par diviem lauciņiem uz priekšu, nonākot blakus lauciņā uz vienas rindas ar spēlētāja bandinieku. Spēlētāja bandinieks var nosists pretinieka bandinieku ejot pa diagonāli uz priekšu un pretinieka bandinieka virzienā.

Piecdesmit gājienu noteikums - noteikums, kas paredz, ka spēlētājs var paziņot neizšķirtu, ja ir pēdējie 50 gājienos nav veikts nevies gājies ar bandinieku vai nosista kāda figūra.

Pus-gājienu pulkstenis – šaha programmēšanā izmantots objekts, kas skaita līdzi gājieniem, kas atbilst 50 gājienu likumam, nodrošinot, ka 50 gājienu likums tiek ievērots.

Pilns gājiens - viens pilns gājiens tiek skaitīts tad, kad savu gājienu ir veicis gan baltais, gan melnais spēlētājs.

Spēles līmenis - šīs programmatūras ietvaros tas ir uzdevums, par kuru informācija tiek glabāts FEN tipa failā un tas aptver šaha galdiņa sākuma pozīciju, līmeņa pareizo gājienu sarakstu un mājienu. Spēles līmenis tiek uzskatīts par pabeigtu, kad ir veikti visi gājienus, kas ir aprakstīti līmeņa pareizo gājienu sarakstā.

Mājiens – papildus informācija, kas saistīta ar līmeņa pabeigšanu, kas norāda uz kādu situācijas aspektu vai figūru ar kuru jāveic gājiens, bet nedod precīzu norādījum gājiena veikšanai.

Loģiskais līmenis – spēles reprezentācija un loģiskā apstrāde, bez grafiskas vides.

Grafiskās saskarnes līmenis – spēles vizuālā reprezentācija bez spēles loģika apstrādes.

Līmeņa pareizo gājienu saraksts – saraksts ar gājieniem, kas ir jāveic lai izietu atbilstošo spēles līmeni. Gājieni, kas neatrodas šajā sarakstā tiek uzskatīti par nepareiziem, pat ja tie pēc šaha teorijas būtu pareizāki. Šis saraksts ir daļa no FEN tipa faila.

2.1.4. Saistība ar citiem dokumentiem

Dokumenta sastādīšanā ievērotas standartu LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis", LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai", LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai" prasības atbilstoši šī dokumenta tvērumam.

2.1.5. Pārskats

Dokuments sastāv no 5 daļām:

Pirmajā daļa ir informācija par dokumenta nolūku, darbības sfēru, definīciju skaidrojumu, kā arī šī dokumenta saistību ar citiem dokumentiem.

Otrajā daļā ir vispārējs apraksts par produkta perspektīvām, struktūru funkcijām, lietotāja raksturiezīmēm, kā arī ierobežojumiem.

Trešajā daļā norādītas funkcionālās prasības par izstrādājamo programmatūru.

Ceturtajā daļā aprakstītas programmatūras saskarne prasības.

Piektajā daļā aprakstītas nefunkcionālās prasības par izstrādājamo programmatūru.

2.2. Vispārējs apraksts

2.2.1. Produkta perspektīva

Šis produkts ir pašpietiekams un nav lielākas sistēmas daļa, kā arī nav atkarīga no citām sistēmām. Izstrādājamā programma ir šaha treniņprogramma šahistu taktiskās domāšanas uzlabošanai.

2.2.2. Produkta funkcijas

- Spēles līmeņa ielādes funkcija - pirms katra līmeņa ielādē visu nepieciešamo informāciju par attiecīgo spēles līmeni.
- Šaha laukuma uzzīmēšanas funkcija - uzzīmē šaha laukumu tādu, kāds tas ir konkrētajā stāvoklī.
- Gājiena veikšanas funkcija - veic šaha figūru pārvietošanu uz šaha laukuma.
- FEN līmeņa fail ielasīšanas funkcija - ielasa FEN tipa fails, kas glabā informāciju par spēles līmeni.
- Mājiena uzrādīšanas funkcija - sniedz norādes par līmeņa izpildi uznirstošajā logā.
- Informācijas par spēli iegūšanas funkcija - sniedz informāciju par spēli un tās izstrādātāju uznirstošajā logā.
- Validācijas funkcija - pārbauda vai konkrētais gājiens ir tāds pats, kāds tas ir paredzēts līmeņa ietvaros.

2.2.3. Lietotāju raksturiezīmes

Lietotājam ir jāprot spēlēt galda spēli šahs, jāpārzina katras figūras gājienu specifika. Lietotājam nav jābūt programmēšanas vai dziļām datora pārvaldības zināšanām, lai varētu darboties ar programmatūru.

2.2.4. Vispārējie ierobežojumi

Programmatūras darbināšanai nepieciešams dators, kuram ir uzinstalēta Java virtuālā mašīna.

2.3. Funkcionālās prasības

2.3.1. Spēles sākuma izsaukšanas funkcija

Identifikators:

AA

Mērķis:

Funkcijas mērķis ir uzsākt spēli sākot no 1. līmeņa.

Ievaddati:

Zem nolaižamās izvēlnes “File” tiek nodrošināta poga “Start Game”

Apstrāde:

Funkcija izsauc funkciju BA padodot tai datu tipa String vērtību “level1”. Tālāk izsaucot funkciju CB un padara pogu “Hint”, kas atrodas zem nolaižamās izvēlnēs “File” neaktīvu.

Izvaddati:

Šī funkcija tiešā veidā datus neizvada, tos izvada šīs funkcijas izsauktās funkcijas.

Kļūdu apstrāde:

Kļūdas gadījuma uznirstošajā logā tiek izvadīts paziņojums ‘Could not load level "level1"’

2.3.2. Mājiena uzrādīšanas funkcija

Identifikators:

AB

Mērķis:

Funkcijas mērķis ir dot papildus informāciju spēlētājam, ar norāžu, jeb “mājiena” palīdzību, nesniedzot konkrētu informāciju par veicamajiem gājieniem.

Ievaddati:

Zem nolaižamās izvēlnēs “File” tiek nodrošināta poga “Show Hint”

Apstrāde:

Funkcija iegūst no spēles līmeņa iepriekš nolasīto mājiena tekstu.

Izvaddati:

Uznirstošajā logā tiek izvadīts konkrētā līmeņa mājiena teksts.

Kļūdu apstrāde:

Ja nav izdevies iegūt mājienu, tad uznirstošajā logā tiek izvadīts paziņojums “"Sorry, no hint was specified for this level"

2.3.3. Informācijas par spēli iegūšanas funkcija

Identifikators:

AC

Mērķis:

Šīs funkcijas mērķis ir sniegt informāciju par spēli, kādēļ tā tika radīta, par tās autoru un autora kontaktinformāciju.

Ievaddati:

Zem nolaižamās izvēlnēs “File” tiek nodrošināta poga “Show Hint”

Apstrāde:

Funkcija ielasa tekstu no faila, kurā ir sagatavots teksts, par šo spēli un tās autoru.

Izvaddati:

Uznirstošajā logā tiek izvadīts konkrētā līmeņa mājiena teksts.

Kļūdu apstrāde:

Nav.

2.3.4. Iziešana no programmas

Identifikators:

AD

Mērķis:

Beigt spēli un aizvērt programmu

Ievaddati:

Zem nolaižamās izvēlnēs “File” tiek nodrošināta poga “Exit”

Apstrāde:

Tiek atbrīvots grafiskās lietotāja saskarnes objekts.

Izvaddati:

Nav.

Kļūdu apstrāde:

Nav.

2.3.5. Līmeņa ielādes funkcija

Identifikators:

BA

Mērķis:

Šīs funkcijas mērķis ir no BB funkcijā ielasītajiem datiem, loģiskas līmenī sagatavot spēles laukumu ar tam piederošajām figūrām tā, lai tas būtu gatavs grafiskai uzzīmēšanai. Šo funkciju lietotājs nevar izsaukt tieši.

Ievaddati:

Šī funkcija saņem datu tipa String ievadi, kas apzīmē vēlamā līmeņa nosaukumu bez datnes paplašinājuma.

Apstrāde:

Šī funkcija izsauc funkciju BB, padodot tai datu tipa String ievaddatus, kas sastāv no ceļa uz mapi, kur atrodas līmeņu faili, šīs funkcijas ievaddatiem un datnes paplašinājuma. No iegūtajiem datiem tiek izveidots datu struktūra vārdnīca, informāciju par katru laukuma lauciņu un uz to novietotajām figūrām. Tiek saglabāti konkrētā līmeņa pareizie gājieni, kā arī mājiens.

Izvaddati:

Veiksmīgas apstrādes gadījuma tiek atgriezts datu tipa boolena mainīgais ar patiesu vērtību. Kļūdas gadījuma tiek atgriezts boolena mainīgais ar nepatiesu vērtību

Kļūdu apstrāde:

Ja apstrādes procesā ir radusies kāda kļūda, tiek izsaukta funkcija BC, tiek izdzēsts dati par mājienu.

2.3.6. FEN līmeņa fail ielasīšanas funkcija

Identifikators:

BB

Mērķis:

Ielasīt FEN tipa failu, kas glabā informāciju par konkrētā līmeņa laukuma izvietojumu, līmeņa pareizajiem gājieniem, un mājienu. Sagatavot šo informāciju tālākai apstrādei.

Ievaddati:

Tiek padots datu tipa String mainīgais, kas satur ceļu uz konkrētu nolasāmo failu.

Apstrāde:

Tiek ielasīts padotais fails, tiek izveidots masīvs, kas saglabā:

- figūru novietojumu uz laukuma, spēlētāju, kuram ir jāveic gājiens, rokāžu iespējas un *en passant* kauliņu;
- līmeņa pareizo gājienu sarakstu
- konkrētā līmeņa mājienu.

Izvaddati:

Tiek izvadīti datu tipa String masīvs.

Kļūdu apstrāde:

Kļūdas gadījuma tiek atgriezts tukšs datu tipa String masīvs.

2.3.7. Sākuma laukuma izveidošanas funkcija

Identifikators:

BC

Mērķis:

Šīs funkcijas mērķis ir loģiskas līmenī sagatavot spēles laukumu, ar figūrām pozīcijā, kas apzīmē standarta šaha spēles sākuma pozīcijas tā, lai tas būtu gatavs grafiskai uzzīmēšanai. Šo funkciju lietotājs nevar izsaukt tieši.

Ievaddati:

Nav.

Apstrāde:

Tiek izveidots datu struktūra vārdnīca, informāciju par katru laukuma lauciņu un uz to novietotajām figūrām, kas atbilst standarta šaha spēles sākuma pozīcijai.

Izvaddati:

Nav.

Kļūdu apstrāde:

Nav.

2.3.8. Figūras iespējamo gājienu funkcija

Identifikators:

BD

Mērķis:

Funkcijas mērķis ir iegūt konkrētās figūras visus iespējamos gājienus, kas ir iespējami konkrētajā situācijā, ņemot vērā šīs un citu figūru atrašanās vietu uz laukuma.

Ievaddati:

Nav.

Apstrāde:

Attiecīgi pēc katras figūras gājienu veikšanas specifikas, tiek atrasti lauciņi uz kuriem šai figūrai ir iespējams doties viena gājiena ietvaros, tai skaitā arī tādi lauciņi, uz kuriem atrodas pretinieka figūra, kuru iespējams nosist.

Izvaddati:

Tiek atgriezta datu kopa ar lauciņiem uz kuriem konkrētā figūra var veikt gājienu.

Kļūdu apstrāde:

Nav.

2.3.9. Lauciņa uzzīmēšanas funkcija

Identifikators:

CA

Mērķis:

Šīs funkcijas mērķis ir radīt vizuālu šaha laukuma lauciņa reprezentāciju attiecīgi dotajiem parametriem.

Ievaddati:

Laukums, kuram pieder šis lauciņš

Loģiskie dati par lauciņu:

- lauciņa kolonna
- lauciņa rinda
- figūra, kas atrodas uz lauciņa (ja uz lauciņa kāda atrodas)

Apstrāde:

Pēc lauciņa koordinātēm (rindas un kolonnas) tam tiek piemērota lauciņa krāsa - gaišs vai tumšs. Tiek izveidots kvadrāts ar tam piemēroto krāsu. Ja uz lauciņa ir novietota figūra, tad tiek ielasīts šīs figūras attēls un tāds tas attēlots virs lauciņa. Lauciņš tiek uzzīmēts.

Izvaddati:

Ekrānā redzams kvadrāts ar tam attiecīgu krāsu, uz tā ir attēlota figūra, ja uz lauciņa tāda atrodas.

Kļūdu apstrāde:

Nav.

2.3.10. Spēles laukuma uzzīmēšanas funkcija

Identifikators:

CB

Mērķis:

Funkcijas mērķis ir radīt vizuālu spēles laukuma reprezentāciju ar tam piederošajiem lauciņiem.

Ievaddati:

Šī funkcija izmanto laukuma programmas loģiskajā līmenī sagatavotus datus, kas satur informāciju par katru spēles lauciņu.

Apstrāde:

Katram loģiskajā līmenī izveidotam lauciņam tiek izsaukta funkcija CA, un katrs tās atgrieztais lauciņa objekts tiek pievienots laukumam un izkārtots pa laukumu 8 x 8 lielā režģī, katru lauciņu novietojot tam attiecīgajā vietā.

Izvaddati:

Ekrānā redzams kvadrāts, kas sastāv no 64 lauciņiem, uz kuriem ir attēlota figūra, ja uz lauciņa tāda atrodas.

Kļūdu apstrāde:

Nav.

2.3.11. Spēlētāja krāsas informatīvā funkcija

Identifikators:

CC

Mērķis:

Parādīt spēlētāja figūru krāsu

Ievaddati:

Ievaddati tiek iegūti no funkcijas BB saglabātiem datiem par lietotāja figūru krāsu.

Apstrāde:

Tiek nolasīta lietotāja figūras krāsa konkrētajā līmenī. Blakus spēles laukuma grafiskajai reprezentācijai tiek izveidots panelis.

Izvaddati:

Uz izveidotā paneļa tiek parādīts uzraksts "You are playing as WHITE" gadījumā, ja spēlētāja figūru krāsa ir balta, ja spēlētāja figūru krāsa ir melna, tiek parādīts uzraksts "You are playing as BLACK"

Kļūdu apstrāde:

Nav.

2.3.12. Spēlētāja gājiena veikšanas funkcija

Identifikators:

DA

Mērķis:

Šīs funkcijas mērķis ir veikt gājienu ar spēlētāja norādīto figūru uz citu spēlētāja norādīto lauciņu.

Ievaddati:

Pirmais klikšķis uz kāda lauciņa apzīmē lauciņu no kura tiks veikts gājiens (ja uz tā atrodas figūra).

Otrais klikšķis uz kāda no lauciņiem apzīmē lauciņu uz kuru tiks veikts gājiens.

Apstrāde:

Pēc pirmā klikšķa, lauciņš uz kuru tika izvēlēts, tiek iekrāsots citā krāsā, kā pārējie lauciņi. Pēc otrā klikšķa, ja uz pirmā lauciņa atradās kāda no spēlētāja figūrām, tiek izsaukta funkcija BD. Ja lauciņš uz kuru tiek veikts gājiens ir BD izvaddatu kopā un šis gājiens ir arī kā nākamais gājiens līmeņa pareizo gājienu sarakst, tad figūra tiek noņemta no pirmā lauciņa un piedēvēta otrajam lauciņam. Tālāk tiek izsaukta CB funkcija. Pēc gājiena lauciņi no pārējiem lauciņiem un uz tiek iekrāsoti atšķirīgi, lai varētu pamanīt no un uz kurieni ir veikts gājiens.

Izvaddati:

Ja otrais lauciņš ir figūras, kuru spēlētājs vēlas pārvietot, iespējamo gājienu kopā, bet nav kā nākamais gājiens līmeņa pareizo gājien sarakstā, tad uznirstošajā logā tiek izvadīts paziņojums "Wrong move, try again!"

Kļūdu apstrāde:

Nav.

2.3.13. Pretinieka gājiena veikšanas funkcija

Identifikators:

DB

Mērķis:

Šīs funkcijas mērķis ir veikt gājienus spēlētāja pretinieka vietā, attiecīgi pēc spēles līmeņa pareizo gājienu saraksta , tādejādi simulējot īstas spēles situācijas.

Ievaddati:

Lauciņš uz kura stāv figūra, kuru nepieciešams pārvietot.

Lauciņš uz kuru nepieciešams pārvietot figūru.

Apstrāde:

No funkcijas BB iegūtā spēles līmeņa pareizo gājienu saraksta tiek veikts pārvietota figūra no pašreizējā uz nākamo lauciņu, kas ir spēles līmeņa pareizo gājienu sarakstā. Tālāk tiek izsaukta CB funkcija. Pēc gājiena lauciņi no pārējiem lauciņiem un uz tiek iekrāsoti atšķirīgi, lai varētu pamanīt no un uz kurieni ir veikts gājiens.

Izvaddati:

Nav.

Kļūdu apstrāde:

Nav.

2.4. Ārējā saskarne

2.4.1. Lietotāja saskarne

Nepieciešama grafiskās saskarne, kurā jābūt redzamam šaha laukumam, izvēlnes rīkjoslai galvenē un panelim labajā sānā, papildu informācija, kuru nepieciešams izvietot grafiski.

Šaha laukumam jāsastāv no 64 atsevišķiem kvadrātiem, kur uz katra noklikšķinot, tas iezīmējas citā krāsā nekā pārējie laukumi. Spēles gājienus varēs veikt uzklikšķinot uz figūras, tādejādi to izvēloties, un tad uzklikšķinot uz lauciņa uz kuru šai izvēlētajai figūrai ir jāpārvietojas.

izvēlnes rīkjoslā jābūt nolaižamajai izvēlnei ar nosaukumu “File”. Noklikšķinot uz tās, jāatveras izvēlnei. Vienkārša ziņa ir jāattēlo kā uznirstošs logs, kas parāda ziņas tekstu un tai ir viena poga “Ok”. Ziņa par nākamo līmeni ir jāattēlo kā uznirstošs logs, kas parāda ziņas tekstu un tai ir divas pogas, “Next Level” un “Close” (Skatīt 4. pielikumu).

2.4.2. Aparatūras saskarnes

Programmatūrai jāspēj darboties datorā ar šādu minimālo konfigurāciju:

• Procesors: 1000 MHz Intel Celeron (vai labāks);

• RAM: 1GB (vai vairāk);

• Vismaz 1 GB brīvas atmiņas uz cietā diska;

2.4.3. Programmatūras saskarnes

Uz datora uzinstalēts Mac OS X 10.9 +, Windows XP vai Ubuntu 12.04 LTS, uz kuras ir uzinstalēta Java 7.0 versija.

2.5. Nefunkcionālās prasības

2.5.1. Veiktspējas prasības

Programmatūrai jāspēj ielādēt un izveidot līmeņa grafisko reprezentāciju 0.5 sekunžu laikā. Programmatūrai jāspēj veikt lietotāja gājiens 1 sekundes laikā.

2.5.2. Šaha laukuma vizuālā reprezentācija

Programmatūrai jāspēj ģenerēt šaha laukuma vizuālā reprezentācija ar tam piederošajām figūrām gan no baltā spēlētāja perspektīvas, gan melnā spēlētāja perspektīvas. Programmas logam jābūt ar iespēju mainīt tā izmēru.

2.5.3. Fiksēts ievades datu formāts

Programmatūrai jāspēj atpazīt speciāli strukturētus ieejas datus, kas ir saglabāti FEN tipa failos ” un jāspēj tos apstrādāt. Sistēmai jāspēj atpazīt jaunizveidotus FEN tipa failus un jāspēj tās apstrādāt. (Skatīt 2. Pielikumā)

3. Programatūras projektējuma apraksts

3.1. Ievads

3.1.1. Dokumenta nolūks

Programmatūras projektējuma apraksts (PPA) paredzēts izstrādājamajai šaha taktikas treniņu programmatūrai. Tas ir radīts, lai atvieglotu programmatūras analīzi, plānojumu, implementēšanu un lēmumu pieņemšanu tās izstrādes laikā.

3.1.2. Dokumenta nolūks

Programmatūra paredzēta šahistiem, kas vēlas uzlabot sava šaha taktiskās domāšanas spējas. Šī programmatūra piedāvā uz līmeņiem bāzētu interaktīvu dažādu šaha situāciju izspēli.

3.1.3. Saistītie dokumenti

Programmatūras prasību specifikācija.

Dokumenta sastādīšanā ir izmantots standarts LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai".

3.1.4. Izstrādes vide

Programmatūra tiek izstrādāta, izmantojot programmēšanas valodu Java, iebūvētās Java bibliotēkas datu apstrādei, kā arī izstrādes vidi IntelliJ Idea.

3.2. Klašu projektējums

3.2.1. Klašu dekompozīcija

3.2.1.1. Klase Play

Nolūks	Šī klase satur izsaukumu uz klases Table metodi, kas ģenerē grafisko logu
Deklarācijas fails	Play.Java
Implementācijas fails	Play.Java

3.2.1.2. Klase FENParser

Nolūks	Ielasīt FEN tipa failu
Deklarācijas fails	FENParser.Java
Implementācijas fails	FENParser.Java

3.2.1.3. Klase Game

Nolūks	Spēles gājienu un darbību veikšanas loģiskā apstrāde
Deklarācijas fails	Game.Java
Implementācijas fails	Game.Java

3.2.1.4. Klase Board

Nolūks	Izveidot loģisku šaha laukuma reprezentāciju
Deklarācijas fails	Game.Java
Implementācijas fails	Game.Java

3.2.1.5. Klase NextMove

Nolūks	Saglabāt informāciju par spēles līmeņa pareizajiem gājieniem
Deklarācijas fails	Game.Java
Implementācijas fails	Game.Java

3.2.1.6. Klase FENException

Nolūks	Radīt pielāgotu izņēmuma gadījuma paziņojumu, ja datne ar paplašinājumu ".fen" nav ielasāma
Deklarācijas fails	Game.Java
Implementācijas fails	Game.Java

3.2.1.7. Klase Player

Nolūks	Reprezentēt šaha spēlētāju loģiskā līmenī
Deklarācijas fails	Player.Java
Implementācijas fails	Player.Java

3.2.1.8. Klase Square

Nolūks	Glabāt un reprezentēt lauciņu loģiskā līmenī. Piešķirt lauciņam visas tam atbilstošās īpašības.
Deklarācijas fails	Square.Java
Implementācijas fails	Square.Java

3.2.1.9. Klase Piece

Nolūks	Veidot abstraktu figūras reprezentāciju
Deklarācijas fails	Piece.Java
Implementācijas fails	Piece.Java

3.2.1.10. Klase OccupiedSquareException

Nolūks	Radīt pielāgotu izņēmuma gadījuma paziņojumu, ja tiek mēģināts radīt jaunu figūru uz lauciņa, uz kura jau atrodas kāda figūra
Deklarācijas fails	Piece.Java
Implementācijas fails	Piece.Java

3.2.1.11. Klase Bishop

Nolūks	Veidot laidņa loģisko reprezentāciju
Deklarācijas fails	Bishop.Java
Implementācijas fails	Bishop.Java

3.2.1.12. Klase King

Nolūks	Veidot karaļa loģisko reprezentāciju
Deklarācijas fails	King.Java
Implementācijas fails	King.Java

3.2.1.13. Klase Knight

Nolūks	Veidot zirga loģisko reprezentāciju
Deklarācijas fails	Knight.Java
Implementācijas fails	Knight.Java

3.2.1.14. Klase Pawn

Nolūks	Veidot bandinieka loģisko reprezentāciju
Deklarācijas fails	Pawn.Java
Implementācijas fails	Pawn.Java

3.2.1.15. Klase Queen

Nolūks	Veidot dāmas loģisko reprezentāciju
Deklarācijas fails	Queen.Java
Implementācijas fails	Queen.Java

3.2.1.16. Klase Rook

Nolūks	Veidot torņa loģisko reprezentāciju
Deklarācijas fails	Rook.Java
Implementācijas fails	Rook.Java

3.2.1.17. Klase MoveUtil

Nolūks	Utilīt klase, kas palīdz izveidot sarakstu ar dāmas, laidņa vai torņa iespējamajiem gājieniem
Deklarācijas fails	MoveUtil.Java
Implementācijas fails	MoveUtil.Java

3.2.1.18. Klase Table

Nolūks	Grafiskās lietotāja saskarnes izveide un pārvaldība.
Deklarācijas fails	Table.Java
Implementācijas fails	Table.Java

3.2.1.19. Klase BoardPanel

Nolūks	Vizuāli reprezentēt šaha spēles laukumu ar tam piederošajiem lauciņiem
Deklarācijas fails	Table.Java
Implementācijas fails	Table.Java

3.2.1.20. Klase SquarePanel

Nolūks	Vizuāli reprezentēt lauciņu un piešķirt tam figūras, kas uz tā atrodas attēlu
Deklarācijas fails	Table.Java
Implementācijas fails	Table.Java

3.3. Klašu detalizēts projektējums

3.3.1. Klase Play

Klasē Play tiek realizēta *main* funkcija, kura izveido klases Table objektu.

3.3.2. Klase FENParser

Metodes:

parseFenFile

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	String path	String[]	Ielasa FEN tipa failu

Apstrāde:

Metodei tiek padots ceļš uz FEN tipa failu, tā ielasa šo failu, iegūst informāciju, kuru ievieto String tipa masīvā lielumā 3.

- 0. pozīcijā tiek saglabāta pirmā faila līnija, kas satur informāciju par šaha laukuma izkārtojumu, spēlētāju, kuram ir jāveic nākamais gājiens, iespējamajām rokādēm, un *en passant* bandinieku.
- 1. pozīcijā tiek saglabāta otrā faila līnija, kas satur spēles līmeņa pareizo gājienu sarakstu.
- 2. pozīcijā tiek saglabātas visu pārējo līniju konkatenācija, kas satur informāciju par mājienu.

3.3.3. Klase Game

Metodes:

getLevelCount

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	int	Iegūt kopējo līmeņu skaitu

Apstrāde:

Metodei atgriež privātu mainīgo LEVEL_COUNT

Game

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	Game	Klases Game konstruktors

Apstrāde:

Izsauc klases Board metodi *emptyBoard* un metodi *countLevels*

getGame

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	Game	Iegūt klases Game instanci

Apstrāde:

Metodei atgriež privātu mainīgo game

getHint

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	String	Iegūt mājienu

Apstrāde:

Metodei atgriež privātu mainīgo hint

countLevels

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	void	Saskaitīt visus FEN tipa failus

Apstrāde:

Metode saskaita visus failus ar datnes paplašinājumu ".fen", kas atrodas direktorijā, no kuras tiek ielasīti FEN faili un piedēvē šo vērtību mainīgajam LEVEL_COUNT

newLevel

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	String level	booean	Apstrādāt padoto parametru, izveidojot attiecīgo metodi, kas izveidots attiecīgo laukumu

Apstrāde:

Ja metodei padotais parametrs ir tukšs String tipa mainīgais, tad tiek izsaukta Board klases metode *defaultBoard*. un tiek atgriezts *true*. Ja tiek padots eksistējoša līmeņa nosaukums, tad tiek izsaukta klases Board metode *buildBoardFromFen* un tiek atgriezts *true*. Citā gadījumā tiek atgriezts *false*.

newGame

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	void	Izveido jaunu kases Game instanci

Apstrāde:

Metode izsauc klases *Game* konstruktoru un šo jauno objektu piedēvē mainīgajam game.

setLevelMoveList

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	List<NextMove> nextMoveList	void	Izveido jaunu kases Game instanci

Apstrāde:

Metode mainīgajam *levelMoveList* piedēvē padoto līmeņa pareizo gājienu saraksta vērtību.

setLastSquares

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	Square fromSquare Square toSquare	void	Tiek saglabāti pēdējie lauciņi no kura tika veikts un uz kuru tika veikts gājiens

Apstrāde:

Metode saglabā pēdējos lauciņus no kura tika veikts un uz kuru tika veikts gājiens.

getHeroe

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	Player	Atgriež spēlētāja reprezentējošu Player objektu

Apstrāde:

Metode atgriež spēlētāju reprezentējošu *Player* objektu *heroe*

getComputer

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	Player	Atgriež datoru reprezentējošu Player objektu

Apstrāde:

Metode atgriež datoru reprezentējošu *Player* objektu *heroe*

togglePlayerToMove

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	void	Pārslēdz nākamo spēlētāju starp balto un melno spēlētāju

Apstrāde:

Ja mainīgā *playerToMove Player* klases objekta lauka *color* vērtība ir *WHITE*, tad *playerToMove* tiek piedēvēta lauka *blackPlayer* vērtība, ja nē, tad *playerToMove* tiek piedēvēta lauka *whitePlayer* vērtība.

move

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	Player player Square fromSquare Square toSquare int iter	boolean	Veikt figūru izkārtojuma maiņu uz laukuma loģiskajā līmenī, tā lai tas atspoguļotu gājiena veikšanu.

Apstrāde:

Metode sākumā pārbauda vai spēlētājs, kurš to izsaucis ir spēlētājs, kuram ir jāizdara gājiens un vai uz lauciņa no kura tiks veikts gājiens atrodas figūra. Tālāk tiek pārbaudīts vai figūra, kas atrodas uz lauciņa ir tādā pašā krāsā ar spēlētāju, un vai lauciņš uz kuru tiks veikts gājiens ir starp lauciņiem uz kuriem šai figūrai ir iespējams doties. Pēc tam tiek pārbaudīts vai šī s gājiens atbilst līmeņa pareizo gājienu saraksta nākamajam gājienam.

Ja viss atbilst augstākminētajiem nosacījumiem, tad no lauciņa, no kur tiks veikts gājiens tiek noņemta figūra, šī figūra tiek piedēvēta lauciņam, uz kuru tiks veikts vēlamais gājiens. Tiek mainīts spēlētājs, kuram nākamajam ir jāveic gājiens. Tiek nomainīta krāsa abiem padotajiem lauciņiem, norādot, ka gājiens veikts no šiem lauciņiem. Lauciņiem, no kuriem tika veikts iepriekšējais gājiens, ja tādi pastāv, tiek atgriezta to lauciņu parastā krāsa.

Ja kāds pārbaudes solis netika pilnībā iziets, tad tiek izsaukta klases Table metode *showMessage*, ar ziņu "Wrong move, try again!"

3.3.4. Klase Board

getBoardMap

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	Map<String, Square>	Atgriezt vārdnīcu, kas satur visus klases *Square* objektus

Apstrāde:

Metode atgriež vārdnīcu, kas satur visus klases *Square* objektus

setPlayerToMove

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	Player playerToMove	void	Piešķir nākamā spēlētāja vērtību

Apstrāde:

Metode piešķir nākamā spēlētāja vērtību klases *Game* laukam *playerToMove*

setHeroe

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	Player heroe	void	Piešķir spēlētāja reprezentējošu *Player* objektu

Apstrāde:

Metode piešķir spēlētāja reprezentējošu *Player* objektu klases *Game* laukam *heroe*

setComputer

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	Player computer	void	Piešķir dator reprezentējošu *Player* objektu

Apstrāde:

Metode piešķir datora reprezentējošu *Player* objektu klases *Game* laukam *computer*

emptyBoard

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	void	Izveido šaha laukumu loģiskajā līmenī bez figūrām

Apstrāde:

Metode izveido 64 laukumiņus un pievieno tos vārdnīcai

buildBoardFromFen

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	String path	boolean	Izveido šaha laukumu loģiskajā līmenī pēc ielasītā FEN faila

Apstrāde:

Metode izsauc klases FENParser metodi *parseFenFile*, no atgrieztā rezultāta tiek izveidots attiecīgs laukums ar figūrām tajās pozīcijās, kurās tās ir apzīmētas FEN failā. Tiek izsauktas metodes *SetPlayerToMove* , *setHero* un *setComputer*. Tālāk tiek izsaukta metode *setLevelMoveList*, kā parametru padod rezultātu no metodes *parseMoveList* izsaukuma. Visbeidzot, metode piešķir laukam *hint* no FEN faila iegūto informāciju par mājienu.

parseMoveList

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	String moveString	List<NextMove>	Parsē no FEN faila nolasīto informāciju par līmeņa pareizo gājienu sarakstu

Apstrāde:

Metode parsē no FEN faila nolasīto informāciju par līmeņa pareizo gājienu sarakstu un atgriež to kā sarakstu, kas sastāv no klases *NextMove* objektiem.

defaultBoard

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	void	Izveido šaha laukumu loģiskajā līmenī standarta šaha spēles sākuma pozīcijā

Apstrāde:

Metode izsauc metodi *emptyBoard*, piešķir laukuma lauciņiem figūras attiecīgi standarta šaha spēles sākuma pozīcijai.

defaultBoard

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	void	Izveido šaha laukumu loģiskajā līmenī standarta šaha spēles sākuma pozīcijā

Apstrāde:

Metode izsauc metodi *emptyBoard*, piešķir laukuma lauciņiem figūras attiecīgi standarta šaha spēles sākuma pozīcijai.

3.3.5. Klase NextMove

NextMove

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	Square fromSquare Square toSquare	NextMove	Klases konstruktors

Apstrāde:

Metode piešķir objekta laukam *fromSquare* padoto vērtību *fromSquare* un objekta laukam *toSquare* padoto vērtību *toSquare*

getFromSquare

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	Square	Klases Atgriezt lauku *fromSquare*

Apstrāde:

Metode atgriež objekta lauku *fromSquare*

getTtoSquare

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	Square	Klases Atgriezt lauku *toSquare*

Apstrāde:

Metode atgriež objekta lauku *toSquare*

3.3.6. Klase FENException

FENException

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	FENException	Klases konstruktors

Apstrāde:

Metode ir *FENException* objekta konstruktors. Tā izsauc vecāka konstruktoru.

3.3.7. Klase Player

Player

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	Color color	Player	Klases konstruktors

Apstrāde:

Metode piešķir objekta laukam *color* padoto vērtību *color*

3.3.8. Klase Square

Square

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	char file int rank	Square	Klases konstruktors

Apstrāde:

Metode piešķir objekta laukiem attiecīgos padotos laukus, piešķir attiecīgo laukumiņa krāsu atkarībā no tā koordinātēm (kolonnas un rindas) un ievieto šo objektu vārdnīcā, kas satur visus laukumiņus

setToNormalColor

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	void	Piešķir lauciņam tā parasto krāsu

Apstrāde:

Metode piešķir objekta *Square* laukam tā parasto krāsu atkarībā no tā rindas un kolonnas

setToHighlight

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	void	Piešķir lauciņam krāsu, kas to izceļ uz citu fona

Apstrāde:

Metode piešķir objekta *Square* laukam *color* mainīgā *HIGHLIGHT* krāsu

setToMovedColor

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	void	Piešķir lauciņam krāsu, kas apzīmē, no vai uz šo lauciņu tika izdarīts iepriekšējais gājiens

Apstrāde:

Metode piešķir objekta *Square* laukam *color* mainīgā *MOVED_LIGHT* vai *MOVED_DARK* krāsu atkarībā no tā rindas un kolonnas

getRank

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	int	Iegūt objekta lauciņa rindu

Apstrāde:

Metodei atgriež privātu mainīgo *rank*

getFile

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	char	Iegūt objekta lauciņa kolonnu

Apstrāde:

Metodei atgriež privātu mainīgo *file*

toggleOccupied

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	void	Pārslēgt objekta lauka *occupied* vērtību

Apstrāde:

Metodei maina boolena tipa mainīgā *occupied* vērtību uz pretējo

isOccupied

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	boolena	Iegūt informāciju vai uz lauciņa atrodas figūra

Apstrāde:

Metodei atgriež privātu mainīgo *occupied*

setPieceOnSquare

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	Piece pieceOnSquare	void	Uzstāda figūru uz lauciņa

Apstrāde:

Metodei piedēvē objekta laukam *pieceOnSquare* padoto vērtību *pieceOnSquare*. Pārbauda vai uz lauciņa jau atradās kāda figūra. Ja nē, tad izsauc metodi *toggelOccupied*.

removePiece

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	void	Noņem figūru no lauciņa

Apstrāde:

Metodei piešķir laukam *pieceOnTile* vērtību *null*

getPiece

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	Piece	Iegūst figūru, kas atrodas uz lauciņa

Apstrāde:

Metodei atgriež privātu mainīgo *pieceOnTile*

getSquare

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	char file int rank	Square	Iegūst lauciņu

Apstrāde:

Metodei atgriež *Square* objektu no klases *Board* vārdnīcas *BOARD_MAP*

3.3.9. Klase Piece

Piece

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	char file int rank Color color	Piece	Klases konstruktors

Apstrāde:

Metode piešķir objektam lauka *square* vērtību, kuru iegūst izsaucot *Square* klases metodi *getSquare* un tai padodot šīs metodes padotos parametrus *file* un *char*. Objektam piešķir krāsu un lauka *hasMoved* vērtību *false*

setHasMoved

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	void	Apzīmē, ka figūra ir izkustējusies

Apstrāde:

Metode piešķir objekta lauka *hasMoved* vērtību *true*

getHasMoved

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	boolena	Iegūst vērtību, kas apzīmē vai ar figūru ir veikts gājiens

Apstrāde:

Metodei atgriež privātu mainīgo *hasMoved*

getSquare

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	Square	Iegūst lauciņu uz kuras atrodas figūra

Apstrāde:

Metodei atgriež privātu mainīgo *square*

setSquare

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	Square square	void	Piešķirt figūrai lauciņu uz kura tā atrodas

Apstrāde:

Metode piešķir objekta lauka *square* vērtību padoto vērtību *square*

3.3.10. Klase OccupiedSquareException

OccupiedSquareException

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	String message	OccupiedSquareException	Klases konstruktors

Apstrāde:

Metode ir *OccupiedSquareException* objekta konstruktors. Tā izsauc vecāka konstruktoru.

3.3.11. Klase Bishop

Bishop

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	char file int rank Color color	Bishop	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru

getMoves

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	Set<Square>	Iegūt visus iespējamos laidņa gājienus

Apstrāde:

Metode iterē cauri visiem iespējamajiem laidņa virzieniem un katrā iterācijā izsauc klase *MoveUtil* metodi *getPossibleMoves*.

getPieceName

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	String	Iegūt figūras nosaukumu

Apstrāde:

Metode atgriež datu tipa String vērtību “Bishop”

3.3.12. Klase King

King

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	char file int rank Color color	King	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru

King

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	char file int rank Color color boolean hasMoved	King	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru un, ja padotais parametrs *hasMoved* ir *true*, tad izsauc metodi *setHasMoved*.

getMoves

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	Set<Square>	Iegūt visus iespējamos karaļa gājienus

Apstrāde:

Metode iterē cauri visiem iespējamajiem karaļa gājiena virzieniem un pārbauda katru potenciālo lauciņu, vai uz to iespējams veikt gājienu.

getPieceName

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	String	Iegūt figūras nosaukumu

Apstrāde:

Metode atgriež datu tipa String vērtību “King”

3.3.13. Klase Knight

Knight

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	char file int rank Color color	Knight	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru

getMoves

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	Set<Square>	Iegūt visus iespējamos zirga gājienus

Apstrāde:

Metode iterē cauri visiem iespējamajiem zirga gājiena virzieniem un pārbauda katru potenciālo lauciņu, vai uz to iespējams veikt gājienu.

getPieceName

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	String	Iegūt figūras nosaukumu

Apstrāde:

Metode atgriež datu tipa String vērtību “Knight”

3.3.14. Klase Pawn

Pawn

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	char file int rank Color color	Pawn	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru

Pawn

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	char file int rank Color color boolean hasMoved	Pawn	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru un, ja padotais parametrs *hasMoved* ir *true*, tad izsauc metodi *setHasMoved*.

getMoves

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	Set<Square>	Iegūt visus iespējamos bandinieka gājienus

Apstrāde:

Metode iterē cauri visiem iespējamajiem bandinieka gājiena virzieniem un pārbauda katru potenciālo lauciņu, vai uz to iespējams veikt gājienu.

getPieceName

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	String	Iegūt figūras nosaukumu

Apstrāde:

Metode atgriež datu tipa String vērtību “Pawn”

3.3.15. Klase Queen

Queen

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	char file int rank Color color	Queen	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru

getMoves

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	Set<Square>	Iegūt visus iespējamos dāmas gājienus

Apstrāde:

Metode iterē cauri visiem iespējamajiem dāmas virzieniem un katrā iterācijā izsauc klase *MoveUtil* metodi *getPossibleMoves*.

getPieceName

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	-	String	Iegūt figūras nosaukumu

Apstrāde:

Metode atgriež datu tipa String vērtību “Queen”

3.3.16. Klase Rook

Rook

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
public	char file int rank Color color	***Rook***	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru

Rook

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	char file int rank Color color boolean hasMoved	Rook	Klases konstruktors

Apstrāde:

Metode izsauc vecāka konstruktoru un, ja padotais parametrs *hasMoved* ir *true*, tad izsauc metodi *setHasMoved*.

getMoves

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	Set<Square>	Iegūt visus iespējamos torņa gājienus

Apstrāde:

Metode iterē cauri visiem iespējamajiem torņa virzieniem un katrā iterācijā izsauc klase *MoveUtil* metodi *getPossibleMoves*.

getPieceName

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	String	Iegūt figūras nosaukumu

Apstrāde:

Metode atgriež datu tipa String vērtību “Rook”

3.3.17. Klase MoveUtil

getPieceName

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	Set<Square> moves Piece piece int fileDirection int rankDirection	Set<Square>	Palīdzēt iegūt konkrētās figūras visus iespējamos gājienus.

Apstrāde:

Metode iterē cauri tai padotajā virzienā pa lauciņiem, sākot no tā, uz kuras atrodas metodei padotā figūra un, ja ar šo figūru, pēc šaha noteikumiem, iespējams veikt gājienu uz šo lauciņu, tad tas tiek ievietots kopā, kas vēlāk tiek atriezta.

3.3.18. Klase Table

Table

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	Table	Klases konstruktors

Apstrāde:

Metode izveido grafiskās lietotāja saskarnes logu ar tajā novietotu šaha galdiņa reprezentāciju, uz kura neatrodas neviena figūra, labajā pusē no galdiņa atrodas panelis, bet pašā loga aušā ir izvēlnes rīkjosla.

newTable

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	void	Izveido jaunu klases Table instanci

Apstrāde:

Metode izsauc klases *Game* metodi *newGame* un tad izsauc klases Table konstruktoru, piešķirt tā vērtību laukam *table*.

render

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	void	Izveido grafiskās saskarnes interfeisu

Apstrāde:

Metode no jauna izveido visas grafiskās saskarnes loga komponentes, veic metožu izsaukumus lai tās tiktu no jauna uzzīmētas, kā arī veic metodes *makeComputerMove* izsaukumu.

getTable

Pieejamība	Parametri	Atgriežamais tips	Nolūks
public	-	Table	Atgriezt lauka *table* instanci

Apstrāde:

Metode atgriež lauka *table* instanci

makeComputerMove

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	void	Veikt nākamo gājienu no līmeņa pareizo gājienu saraksta, bez lietotāja metodes *move* izsaukuma.

Apstrāde:

Metode nogaida 1 sekundi, ielasa nākamo gājienu no līmeņa pareizo gājienu saraksta, izsauc klases *Game* metodi *move*, kas veic datora spēlētāja gājienu, tiek iekrāsoti lauciņi no kura tika veikts gājiens, uz kuru tika veikts gājiens, un tiek pārzīmēta šaha galdiņa reprezentācijas grafiskā saskarne.

createTableMenuBar

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	JMenuBar	Izveidot izvēlnes rīkjoslu

Apstrāde:

Metode izveido izvēlnes rīkjoslu, kuru aizpilda ar metodes *createFileMenu* palīdzību.

createFileMenu

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	JMenu	Izveidot rīkjoslas nolaižamo izvēlni

Apstrāde:

Metode izveido izvelni “File” ar 4 priekšmetiem :

- “Start Game” ar darbības klausītāju, kurš izveido 1. spēles līmeni, un izsauc metodi render.
- “Show Hint” ar darbības klausītāju, kurš izsauc metodi *showHint*.
- “About” ar darbības klausītāju, kurš parāda uznirstošajā logā failā sagatavotu ziņu par šo programmatūru.
- “Exit”, kas aizver programmatūru.

getAbout

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	String	Ielasa failu par ar informāciju par programmatūru

Apstrāde:

Metode pa rindai ielasa failā sagatavotu ziņu par šo programmatūru

showHint

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	String message	String	Iznirstošajā logā izvada mājiena ziņu

Apstrāde:

Metode pārbauda vai padotais parametrs *message* nav tukšs, ja nē, tad izsauc metodi *showMessage*

showMessage

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	String message String title	String	Iznirstošajā logā izvada ziņu

Apstrāde:

Metode izveido jaunu uznirstošo ziņu, kas tekstā satur padoto mainīgo *message* un kā virsrakstu satur padoto mainīgo *title*.

showMessage

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	String message String title	String	Iznirstošajā logā izvada ziņu

Apstrāde:

Metode izveido jaunu uznirstošo ziņu, kas tekstā satur padoto mainīgo *message* un kā virsrakstu satur padoto mainīgo *title*.

3.3.19. Klase BoardPanel

BoardPanel

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	BoardPanel	Klases konstruktors

Apstrāde:

Metode izveido laukuma grafiskās reprezentācijas objektu, kuram pievieno 64 jaunus klases *SquarePanel* objektus, kas reprezentē laukumiņus. Kā arī pievieno komponentes klausītāju, kas izsauc metodi *drawBoard,* ikreiz, kad laukumam tiek mainīti izmēri.

drawBoard

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	void	Uzzīmēt laukuma grafisko reprezentāciju

Apstrāde:

Metode sākumā noņem visus komponentes no konteinera, pievieno jaunus lauciņus laukumam, atkarībā no spēlētāja krāsas. Ja lietotājs ir baltais spēlētājs, tad a1 lauciņš atradīsies apakšējajā kreisajā stūrī, bet ja lietotājs ir melnais spēlētājs, tad lauciņš a1 atradīsies augšējā labajā stūrī. Tālāk validē laukumu un visbeidzot to uzzīmē tā grafisko reprezentāciju.

3.3.20. Klase SquarePanel

SquarePanel

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	SquarePanel	Klases konstruktors

Apstrāde:

Metode izveido laukumiņa grafiskās reprezentācijas objektu, ja uz tā atrodas figūra, virsū laukumiņam uzliek šīs figūras attēlu, izvedot peles klikšķa klausītāju ,kurš atkarība no iepriekš veiktajiem klikšķiem vai nu iezīmē izvēlēto laukumiņu izceļošā krāsa, vai nu noņem iezīmējumu iepriekš iezīmētam lauciņam, vai arī veic gājiena mēģinājumu.

Ja gājiens ir bijis veiksmīgs, tas bija pēdējais gājiens spēles līmeņa pareizo gājienu sarakstā, tad, ja vairāk nav līmeņi, tiek izvadīts uznirstošs paziņojums "You complected all levels!" un uzzīmēts tukšs laukums. Ja vēl ir palikuši līmeņi, tad tiek izvadīts uznirstošais lodziņš ar apsveikumu un iespēju izvēlēties vai pāriet uz nākamo līmeni, vai nē. Ja tiek izvēlēts pāriet uz nākamo līmeni, tad tas tiek ielādēts

assignPiece

Pieejamība	Parametri	Atgriežamais tips	Nolūks
private	-	void	Pievienot lauciņa grafiskajai reprezentācijai atbilstošo figūru

Apstrāde:

Metode pārbauda vai uz lauciņa atrodas šaha figūra. Ja uz tā atrodas šaha figūra, tad tiek ielādēta attiecīgas šaha figūras attēls un tas tiek no vietots pa virsu lauciņa grafiskajai reprezentācijai.

showNextMessage

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	int	Izvada uznirstošo logu ar apsveikumu par līmeņa pabeigšanu

Apstrāde:

Metode izvada uznirstošo logu ar apsveikuma ziņu "You complected this level!" un piedāvā iespēju vai nu turpināt ar nākamo līmeni, vai aizvērt logu.

resize

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	BufferedImage image int width int height	BufferedImage	Maina figūras attēla izmērus

Apstrāde:

Metode maina figūras attēla izmērus pēc tiem, kas tiek padoti un atgriež attēlu ar izmainītiem izmēriem.

drawSquare

Pieejamība	**Parametri**	**Atgriežamais tips**	**Nolūks**
private	-	void	Uzzīmēt lauciņa grafisko reprezentāciju

Apstrāde:

Metode sākumā noņem visus komponentes no konteinera, uzstāda lauciņa krāsu, pievieno tam figūras attēlu, validē un uzzīmē jaunu lauciņu.

4. Programatūras testēšanas dokumentācija

4.1. Testēšanas plāns

Tika veiktas sekojošas darbības:

- Testu noteikšana
- Provizorisko testu rezultātu noteikšana
- Testēšana
- Rezultātu dokumentācija

4.2. Testēšanas apraksts

Testēšana tika veikta ar vienībtestu palīdzību. Testēšanas laikā tiks testēta katra netriviāla metode.

4.3. Testēšanas žurnāls

4.3.1. Klase FENParser

Nr.	**Testa apraksts**	**Sagaidāmais rezultāts**	**Testēšanas rezultāts**
1	Tiek padots pareizi izveidots FEN fails	Tiek atgriezts String tipa masīvs lielumā 3 ar fen failā ietverto informāciju	+
2	Tiek padots nepareizi izveidots FEN fails	Atgriež *null*	+

4.3.2. Klase Game

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Direktorijā ir 10 faili, visi ar datnes paplašinājumu ".fen". Tiek izsaukta metode *countLevel*	Tiek atgriezts *int* 10	+
2	Direktorijā ir 11 faili, 10 no tiem ar datnes paplašinājumu ".fen", 1 ar datnes paplašinājumu ".txt". Tiek izsaukta metode *countLevel*	Tiek atgriezts *int* 10	+
3	Tiek izsaukta metode *newLevel* ar parametru "level2"	Tiek atgriezts *boolean true*	+
4	Tiek uzstādīts pamata spēles sākuma laukums. Metodei *move* tiek padoti mainīgie *Player* objekts ar krāsu *WHITE*, *Square* a2, *Square* a4. *int* 0	Tiek atgriezts *boolean true*, uz lauciņa a4 atrodas baltais bandinieks, uz lauciņa a2 figūras neatrodas.	+

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
5	Tiek uzstādīts pamata spēles sākuma laukums. Metodei move tiek padoti mainīgie Player objekts ar krāsu WHITE, Square a3, Square a4, int 0	Tiek atgriezts boolean false uz lauciņiem a3 un a4 figūras neatrodas.	+

4.3.3. Klase Board

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izsaukta metode *emptyBoard*	Izveidots laukums bez nevienas figūras	+
2	Tiek izsaukta metode *buildBoardFromFen*, tai tiek padots ceļš uz pareizu FEN failu	Tiek izveidots laukums pēc fen failā specificētās informācijas	+
3	Tiek izsaukta metode *buildBoardFromFen*, tai tiek padots ceļš uz nepareizi izveidotu FEN failu	Tiek izmests *FENException* izņēmums	+

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
4	Tiek izsaukta metode *buildBoardFromFen*, tai tiek padots nepareizs ceļš	Metode atgriež *false*	+
5	*Tiek izsaukta metode parseMoveList ar pareizu ievadi*	Tiek atgriezts saraksts ar vēlamajiem lauciņiem	+
6	Tiek izsaukta metode *defaultBoard*	Tiek izveidots laukums kur figūras ir novietotas standarta šaha spēles sākuma pozīcijā	+

4.3.4. Klase Square

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izveidots jauns *Square* objekts.	Jaunizveidotais *Square* objekts ir pieejams no klases *Board* vārdnīcā *BOARD_MAP*	+
2	Tiek izveidots jauns *Square* objekts, uzlikta uz tā figūra.	Jaunizveidotajam *Square* objektam ir redzama figūra, kas uz tā tika uzlikta.	+
3	Tiek izveidoti 64 lauciņi atbilstoši šaha laukumam.	Visiem lauciņiem ir piešķirta pareizā krāsa pēc šaha laukuma specifikas	+

4.3.5. Klase Bishop

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izveidots jauns *Bishop* objekts.	Jaunizveidotais Bishop objekts atrodas uz pareizā lauciņa	+
2	*Bishop* objekts tiek novietots uz tukša laukuma lauciņā e5	Metode *getMoves* visi lauciņi uz kuriem var doties laidnis.	+
3	*Bishop* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu lauciņā e5	Metode *getMoves* atgriež visus lauciņus uz kuriem laidnis var doties, tai skaitā pretinieka figūru sišanu.	+
4	*Bishop* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu laukuma malā	Metode *getMoves* visi lauciņi uz kuriem var doties laidnis.	+

4.3.6. Klase King

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izveidots jauns *King* objekts.	Jaunizveidotais King objekts atrodas uz pareizā lauciņa	+
2	Tiek izveidots jauns *King* objekts padodot parametru, *hasMoved true*.	Jaunizveidotais King objekts atrodas uz pareizā lauciņa, parametrs *hasMoved* ir *true*	+

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
3	*King* objekts tiek novietots uz tukša laukuma lauciņā e5	Metode *getMoves* visi lauciņi uz kuriem var doties karalis.	+
4	*King* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu lauciņā e5	Metode *getMoves* atgriež visus lauciņus uz kuriem karalis var doties, tai skaitā pretinieka fugūru sišanu.	+
5	*King* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu laukuma malā	Metode *getMoves* visi lauciņi uz kuriem var doties karalis.	+

4.3.7. Klase Knight

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izveidots jauns *Knight* objekts.	Jaunizveidotais *Knight* objekts atrodas uz pareizā lauciņa	+
2	*Knight* objekts tiek novietots uz tukša laukuma lauciņā e5	Metode *getMoves* visi lauciņi uz kuriem var doties zirgs.	+
3	*Knight* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu lauciņā e5	Metode *getMoves* atgriež visus lauciņus uz kuriem zirgs var doties, tai skaitā pretinieka figūru sišanu.	+

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
4	*Knight* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu laukuma malā	Metode *getMoves* visi lauciņi uz kuriem var doties zirgs.	+

4.3.8. Klase Pawn

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izveidots jauns *Pawn* objekts.	Jaunizveidotais *Pawn* objekts atrodas uz pareizā lauciņa	+
2	Tiek izveidots jauns Pawn objekts padodot parametru, *hasMoved true*.	Jaunizveidotais *Pawn* objekts atrodas uz pareizā lauciņa, parametrs *hasMoved* ir *true*	+
3	*Pawn* objekts tiek novietots uz tukša laukuma lauciņā e5	Metode *getMoves* visi lauciņi uz kuriem var doties bandinieks.	+
4	*Pawn* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu lauciņā e5	Metode *getMoves* atgriež visus lauciņus uz kuriem bandinieks var doties, tai skaitā pretinieka fugūru sišanu.	+
5	*Pawn* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu laukuma malā	Metode *getMoves* visi lauciņi uz kuriem var doties bandinieks.	+

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
6.	Tiek izveidos *Pawn* objekts uz f3 lauciņa. Tad tiek izveidots vēl viens *Pawn* objekts uz f3 lauciņa	Tiek izmests izņēmums *OccupiedSquareException* ar ziņu "This Square is occupied"	+

4.3.9. Klase Queen

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izveidots jauns *Knight* objekts.	Jaunizveidotais *Knight* objekts atrodas uz pareizā lauciņa	+
2	*Knight* objekts tiek novietots uz tukša laukuma lauciņā e5	Metode *getMoves* visi lauciņi uz kuriem var doties zirgs.	+
3	*Knight* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu lauciņā e5	Metode *getMoves* atgriež visus lauciņus uz kuriem zirgs var doties, tai skaitā pretinieka figūru sišanu.	+
4	*Knight* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu laukuma malā	Metode *getMoves* visi lauciņi uz kuriem var doties zirgs.	+

4.3.10. Klase Rook

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izveidots jauns *Rook* objekts.	Jaunizveidotais *Rook* objekts atrodas uz pareizā lauciņa	+
2	Tiek izveidots jauns *Rook* objekts padodot parametru, *hasMoved true*.	Jaunizveidotais Rook objekts atrodas uz pareizā lauciņa, parametrs *hasMoved* ir *true*	+
3	*Rook* objekts tiek novietots uz tukša laukuma lauciņā e5	Metode *getMoves* visi lauciņi uz kuriem var doties tornis.	+
4	*Rook* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu lauciņā e5	Metode *getMoves* atgriež visus lauciņus uz kuriem tornis var doties, tai skaitā pretinieka fugūru sišanu.	+
5	*Rook* objekts tiek novietots uz laukuma ar citām figūrām, kas bloķē daļu gājienu laukuma malā	Metode *getMoves* visi lauciņi uz kuriem var doties tornis.	+

4.3.11. Klase Table

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izsaukts klases *Table* konstruktors	Izveidots objekts ar atbilstošajiem elementiem	+
2	Loģiskajā līmenī tiek ielādēts jauns līmenis un izsaukta metode *render*	Tiek izveidoti visi attiecīgie objekti grafiskajā līmenī atbilstoši loģiskajam līmenim	+
3	Loģiskajā līmenī tiek veikts gājiens ar klases *Game* metodes *move* izsaukumu un izsaukta metode *render*	Gājiens atspoguļojas arī grafiskajā līmenī	+
4	Tiek veikts metodes *showMessage* izsaukums	Tiek izveidots jauns *JOptionPane* objekts ar atbilstošajiem parametriem	+

4.3.12. Klase BoardPanel

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izsaukts klases *BoardPanel* konstruktors	Ir izveidots grafisks objekts *BoarPanel* ar 64 bērniem *SquarePanel* objektiem	+

4.3.13. Klase SquarePanel

Nr.	Testa apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
1	Tiek izsaukts klases *SquarePanel* konstruktors	Ir izveidots grafisks objekts *SquarePanel*	+
2	Tiek izsaukts klases *SquarePanel* konstruktors un loģiskajā vidē attiecīgajam lauciņam tiek pievienota figūra	Attiecīgās figūras attēls tiek pievienots *SquarePanel* objektam	+

5. Projekta organizācija

Programmatūras izstrādē tika izmantots “Ūdenskrituma modelis”.

Uzsakot sistēmas izstrādi tika apzinātas visas prasības, kas tika apkopotas programmatūras prasību specifikācijā. Tika apzinātas vairākas iespējas kā saglabāt visu līmenim nepieciešamo informāciju un tika nonākts pie izvēles par labu FEN notācijai, to modificējot atbilstošajai projekta specifikai.

Tālāk tika izstrādāts programmatūras projektējuma apraksts.

Kad tas tika pabeigts, tika uzsākta programmatūras programmēšana. Izstrādes laikā programmas prasību apraksts nedaudz mainījās, tika veiktas nelielas korekcijas, lai uzlabotu sistēmas darbību.

Kad programmēšana tika pabeigta, tika veikta vienībtestēšana katrai projektējuma specifikācijās aprakstītajai funkcijai. Kur bija nepieciešams, tika pielabots programmatūras kods, lai testi izpildītos veiksmīgi.

6. Kvalitātes nodrošināšana

Kvalitāte tika nodrošināta, pirms programmatūras izstrādes izveidojot programmatūras prasību specifikāciju un programmatūras projektējuma aprakstu, kas tika izstrādāti saskaņā ar programmatūras prasību specifikācijas ceļvedi LVS 68:1996 un ieteicamo praksi programmatūras projektējuma aprakstīšanai LVS 72:1996.

Katrai programmas funkcijai tika veikta vienībtestēšana, pārliecinoties par tās pareizu darbību.

7. Konfigurāciju pārvaldība

Lai veiktu kvalitatīvu konfigurāciju pārvaldību, tika izmantots versiju kontroles rīks "GitHub", kas nodrošina iespēju efektīvi sekot līdzi veiktajām izmaiņām, kā arī veikt programmatūras izstrādi uz vairākām darbstacijām.

8. Darbietilpības novērtēšana

Darbietilpības novērtēšanai tika izvēlēts Basic COCOMO modelis. Tam nepieciešams aptuvens programmatūras koda rindiņu skaits, kas tika parēķināts ar funkcijpunktu palīdzību.

	Sarežģīti	**Vidēji sarežģīti**	**Vienkāršs**	**Funkcijpunktu skaits**
Ievadi	6x0	4x0	3x0	0
Izvadi	7x1	5x0	4x2	15
Iekšējie datu faili	15x0	10x0	7x1	7
Ārējie interfeisa faili	10x0	7x0	5x0	0
Vaicājumi	6x0	4x1	3x5	19
Nekoriģēto funkcijpunktu skaits				**41**

Iegūtie nekoriģētie funkcijpunkti tika pareizināti ar vērtības pielāgošanas koeficientu (value adjustment factor (VAF)), kas šim projektam ir **0,68**. Iegūto koriģēto funkcijpunktu skaits ir **27,9** funkcijpunkti. Iegūtos koriģētos funkcijpunktus sareizina ar programmēšanas valodas Java konstanti, kas vidēji ir **53**. Iegūts tiek rezultāts ir ~ **1479** pirmkoda koda rindiņas.

Pielietojot COCOMO modeli $\boldsymbol{E = 2,4 * (KLOC)1,05}$, tiek iegūts aptuvenais darbietilpības novērtējums **3,6** mēneši. Reālajā dzīvē programmētājam šīs programmatūras izstrāde aizņēma aptuveni **3** mēnešus.

9. Secinājumi

Darbā izvirzītais mērķis ir sasniegts, ir izveidota interaktīva šaha taktikas treniņu programmatūra ar grafisko lietotāja saskarni, kas piedāvā interaktīvi risināt šaha situācijas vārākos līmeņos, tādejādi uzlabojot lietotāja šaha taktisko domāšanu. Visas programatūrai paredzētās iespējas ir izstrādātas pilnā apmērā.

Programmatūrai tika uzrakstīta prasību specifikācija un projektējuma apraksts. Tika veikta un dokumentēta vienībtestēšana.

Tika papildinātas zināšanas objektorientētajā programmēšanas paradigmā un programmēšanas valodā Java.

Tika apgūta darbietilpības novērtēšanas metode COCOMO, ar kuras palīdzību tika izrēķināts, ka šīs programmatūras izstrādes darbietilpība atbilst 3,6 personmēnešiem.

Izstrādes laikā tika iegūtas padziļināti apgūtas prasmes ar Java lietojumprogrammu saderības rīkkopu Swing, kas paredzēta grafiskās lietotāja saskarnes programmatūru veidošanai.

Izmantotā literatūra

1. A.Huny, D.Thomas , The Pragmatic Programmer: From Journeyman to Master 1999, Addison-Wesley, 2005

2. Latvijas Valsts Standarts LVS 68:1996 – "Informācijas tehnoloģija. Programminženierija. Programmatūras prasību specifikācijas ceļvedis." SIA "Latvijas Valsts Standarts", 1996.

3. Latvijas Valsts Standarts LVS 72:1996 – "Informācijas tehnoloģija. Programminženierija. Ieteicama prakse programmatūras projektējuma aprakstīšanai." SIA "Latvijas Valsts Standarts", 1996.

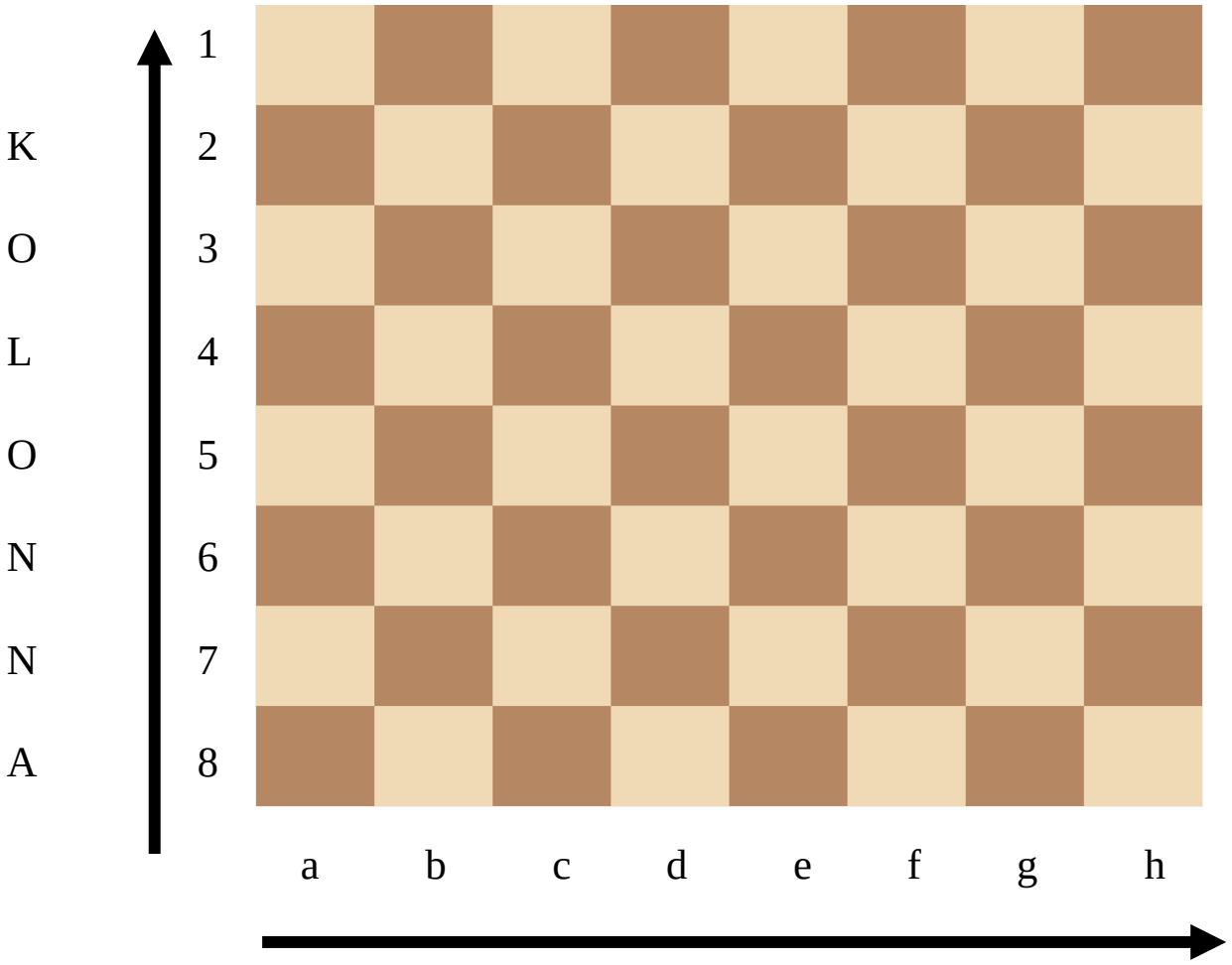
4. Creating a GUI With JFC/Swing [Tiešsaite] (29.05.2017) Pieejams: http://docs.oracle.com/javase/tutorial/uiswing/

5. Akadēmiskā terminu vārdnīca [tiešsaiste] (29.05.2017.). Pieejams: http://termini.lza.lv

6. Tezaurs.lv Skaidrojošā vārdnīca [tiešsaiste] (29.05.2017.). Pieejams: http://tezaurs.lv

Pielikumi

1. Pielikums Šaha spēles laukums



2. Pielikums Forsaita-Edvarda Notācija (FEN)

Forsaita-Edvarda Notācija (FEN) ir standarta notācija, kas apraksta visu figūru atrašanos konkrēta laika mirklī. FEN dod visu nepieciešamo informāciju, lai varētu atsākt šaha spēli no konkrētas vietas spēlē.

Standarta variantā FEN tiek izmantos sekojoši:

FEN saglabā noteiktu momentu šaha spēlē, visu vienā teksta līnijā izmantojot ASCII rakstzīmju kopā. Teksta failam tikai ar FEN datiem vajadzētu tikt saglabātam ar datnes paplašinājumu ".fen".

FEN rakstzīmju virkne sastāv no sešām daļām:

Figūru novietojums (no baltā spēlētāja skatu punkta). Tiek aprakstīta katra rinda, sākot ar 8. Katras rindas šahā figūru novietojums un tukšie laukumi tiek aprakstīti sākot no kolonnas "a" līdz kolonnai "h". Sekojot Algebriskajai Notācijai (AN), katrai figūrai ir savs unikāls burts, kas ir atvasināts no angļu valodas (bandinieks = "P" (pawn), zirgs = "N"(knight), laidnis "B"(bishop),dāma = "Q" (queen), karalis ="K" (king)). Baltās figūras ir piedēvēti lielie burti ("PNBRQK") , savukārt melnajām figūrām ir piedēvēti mazie burti ("pnbrqk"). Tukšie laukumi tiek apzīmēti ar ciparu no 1 līdz 8, kas attiecīgi apzīmē cik daudz pēc kārtas ir tukšo laukumu. Katra laukuma rinda tiek atdalīts ar uz priekšu vērstu slīpsvītru "/".

Aktīvā krāsa. Burts "w" nozīmē, ka nākamā gājiena tiesības pieder spēlētājam ar baltajām figūrām, savukārt burts "b" nozīmē, ka gājiena tiesības pieder spēlētājam ar melnajām figūrām.

Iespējamās rokādes. Ja abiem spēlētājiem nav iespējamas rokāde, tad tas tiek apzīmēts ar domuzīmi "-". Pretējā gadījuma šajā daļā parādās viens vai vairāki sekojošie burti: "K" (Baltais spēlētājs var veikt rokādi karaļa pusē), "Q" (Baltais spēlētājs var veikt rokādi dāmas pusē), "k" (Melnais spēlētājs var veikt rokādi karaļa pusē), "q" (Melnais spēlētājs var veikt rokādi dāmas pusē);

Iespējamā "en passant" gājiena laukums Algebriskajā Notācijā. Ja nav neviena "en passant" gājiena, tad šī daļa ir apzīmēta ar domuzīmi "-".

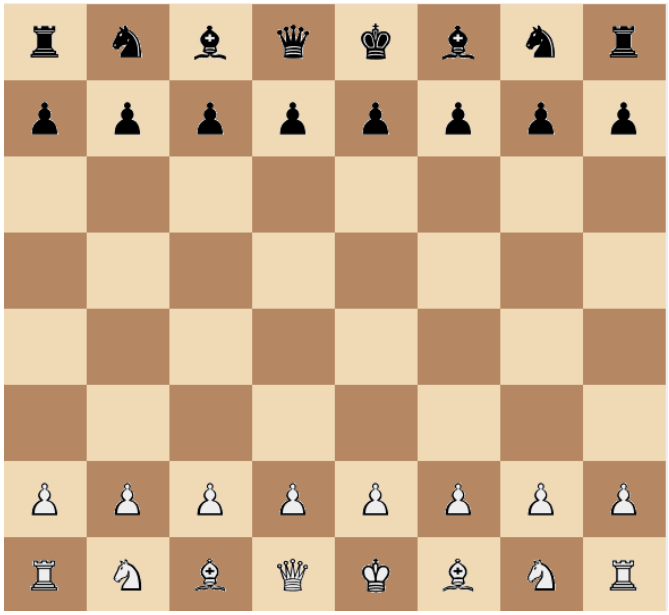
Pus-gājienu pulkstenis. Šī daļa tiek apzīmēta ar skaitli, cik gājienus ir veikti no pēdējās figūras nosišanas vai pēdējā bandinieka gājiena. Tas tiek izmantots, lai sekotu līdz piecdesmit gājienu noteikumam.

Pilno veikto gājienu skaits. Skaitlis, kurš nosaka cik pilnu gājienu ir veikti. Tas sākas no 1 un palielinās par 1 pēc katra melnā spēlētāja gājiena.

Standarta FEN notācijas piemērs:

Šaha spēles sākums, kad neviens vēl nav izdarījis gājienu (Skatīt attēlā 1):

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1



Attēls 1 Standarta šaha spēles sākuma pozīcija

3. Pielikums Pielāgotā Forsaita-Edvarda Notācija

Šīs programmatūras nolūkiem FEN notācija tika pielāgota programmatūras vajadzībām.

Pirmās četras FEN daļas palika nemainītas, savukārt pus-gājienu skaits un pilno veikto gājienu skaits šīs programmatūras ietvaros nav nepieciešamas, tāpēc tās netika iekļautas.

Papildus tika pievienotas vēl 2 daļa lai pielāgotais FEN sevī ietvertu informāciju par katra spēles līmeņa pareizajiem gājieniem, kā arī mājiena informāciju:

Spēles līmenim paredzētie gājieni tiek apzīmēti pielāgotā Algebriskajā Notācijā apzīmējot tikai figūras ar kuru tiek veikts gājiens laukumu un uz kuru laukumu figūra dodas. Figūras, ar kuru tiks izdarīts gājiens, laukums un laukums uz kuru dosies figūra tiek atdalīti ar domuzīmi “-”. Gājieni savā starpā tiek atdalīti ar uz priekšu vērstu slīpsvītru “/”.

Visa informācija, kas seko nākamajā līnijā pēc spēles līmenim paredzētajiem gājieniem tiek uzskatīta par spēles līmeņa mājienu.

Pielāgotā FEN notācijas piemērs:

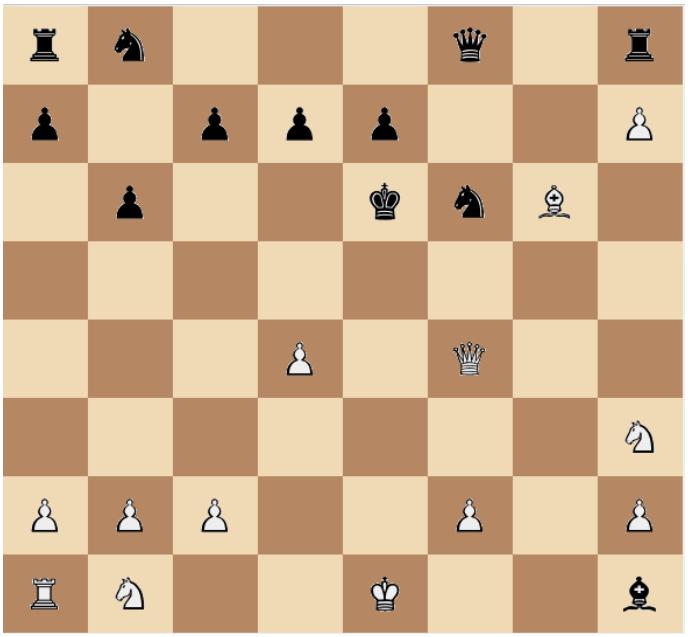
Šaha spēles laukuma izkārtojums, kas redzams attēlā 2 :

rn3q1r/p1ppp2P/1p2knB1/8/3P1Q2/7N/PPP2P1P/RN2K2b b Q -

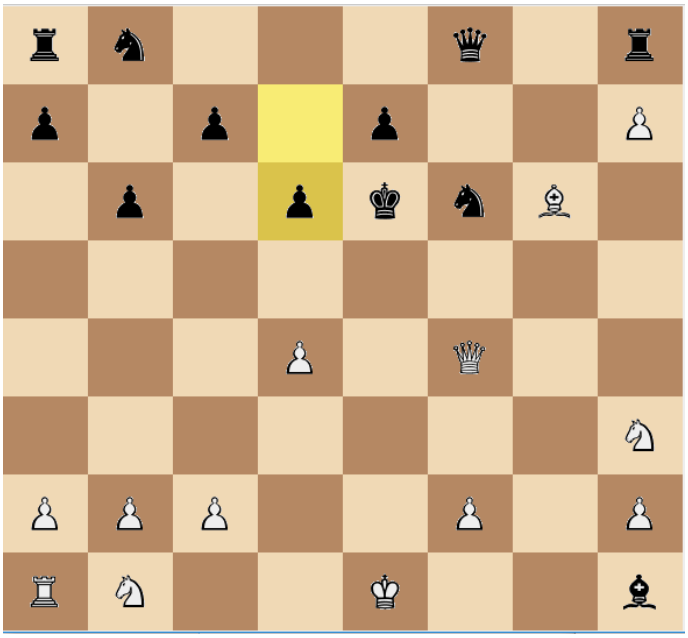
d7-d6/f4-f5

Hint:

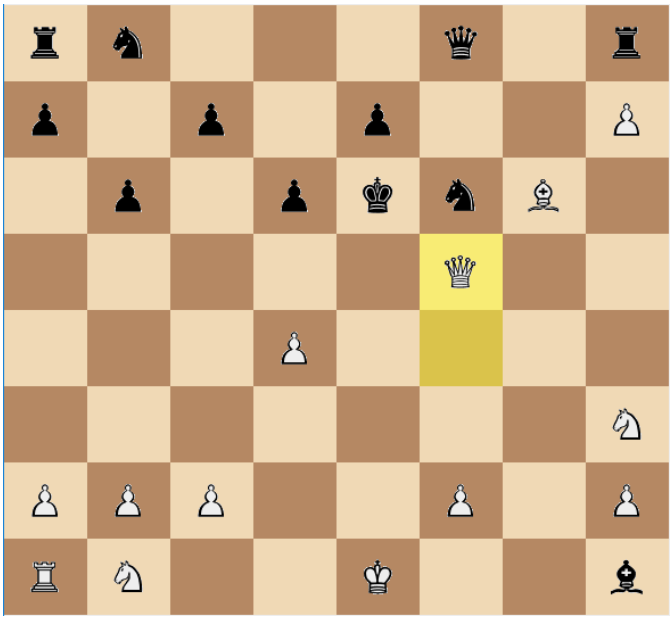
Look for check-mate with queen



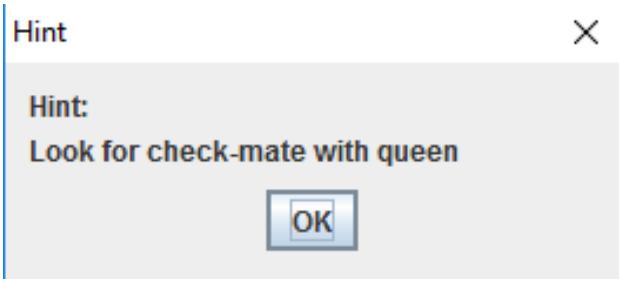
Attēls 2 Pielāgotā FEN piemēra pozīcija



Attēls 3 Pēc pirmā gājiena

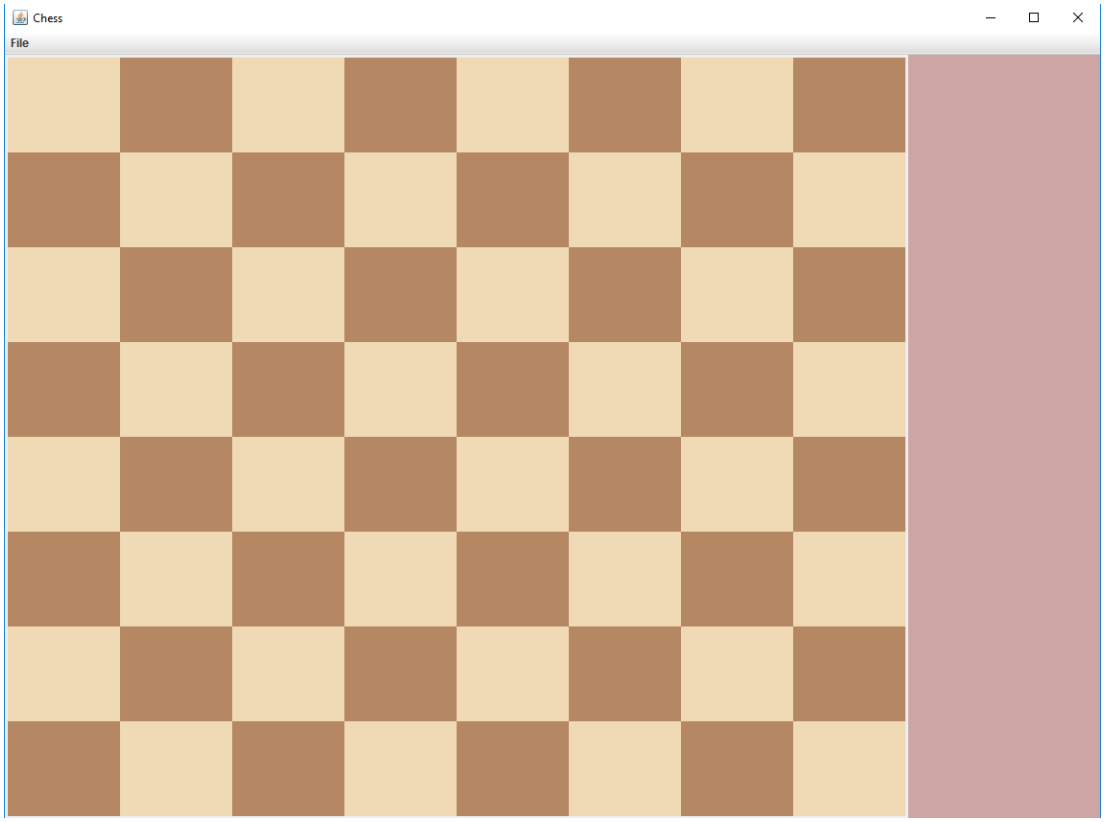


Attēls 4 Pēc 2. gājiena

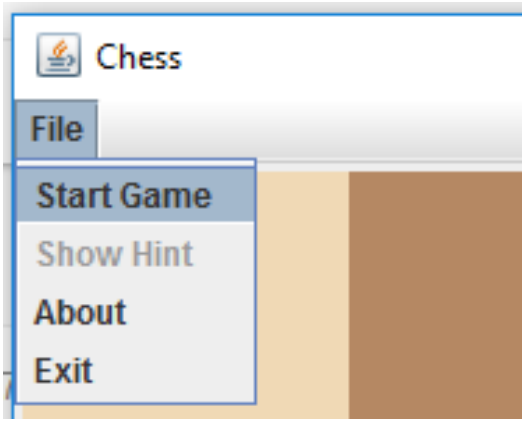


Attēls 5 Mājiena ziņa

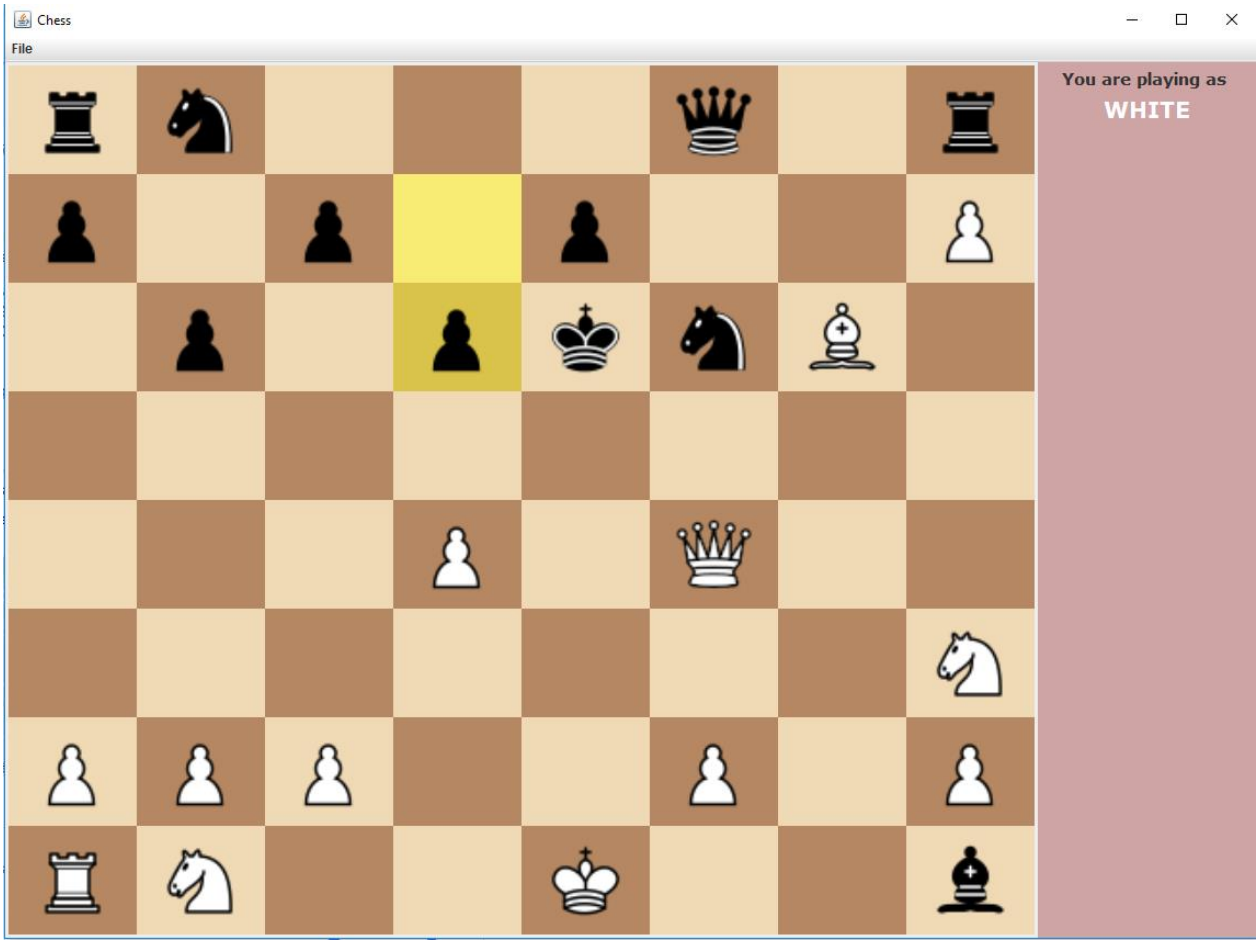
4. Pielikums Grafiskā lietotāja saskarne



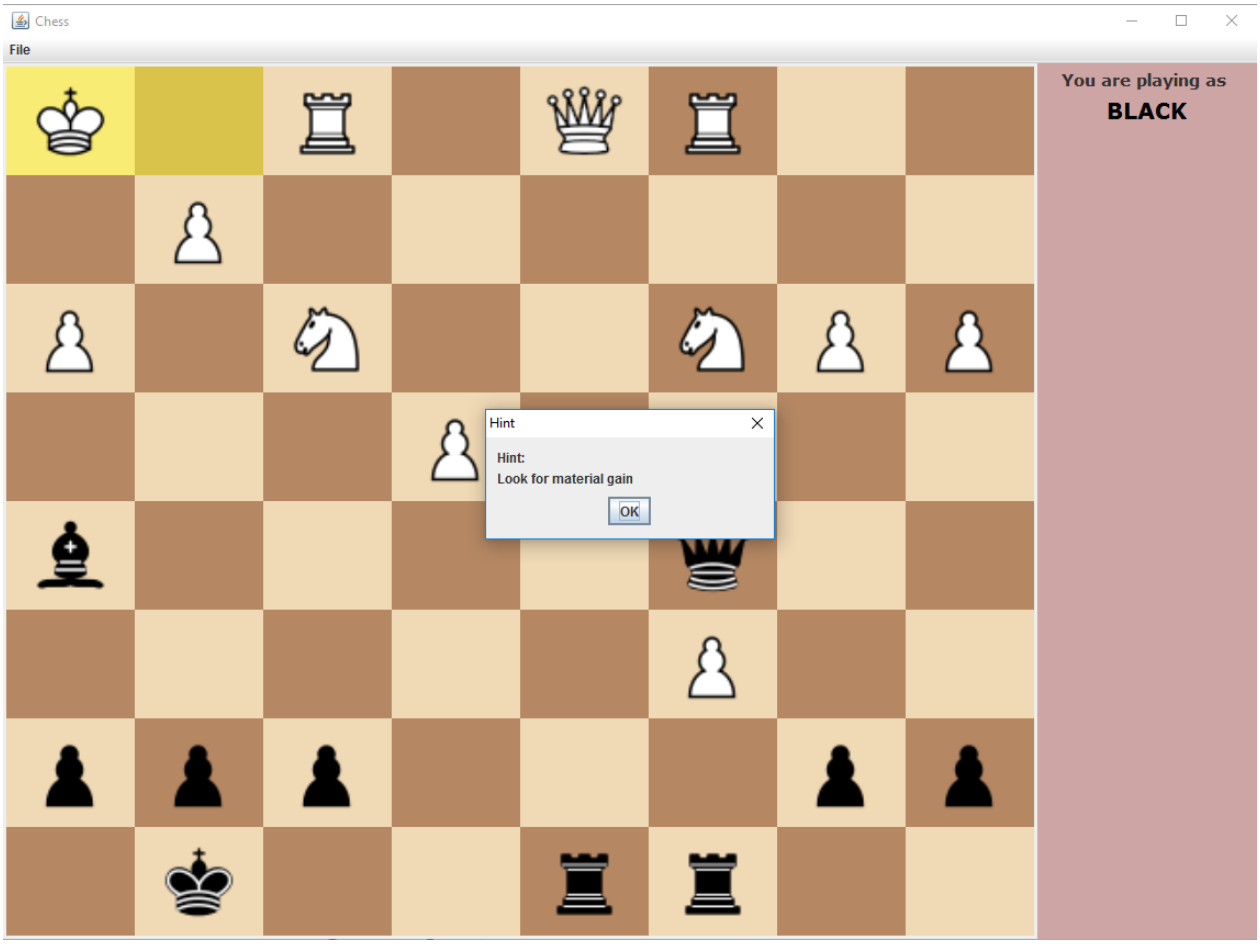
Attēls 6 Sākuma logs, kad atvērta programma



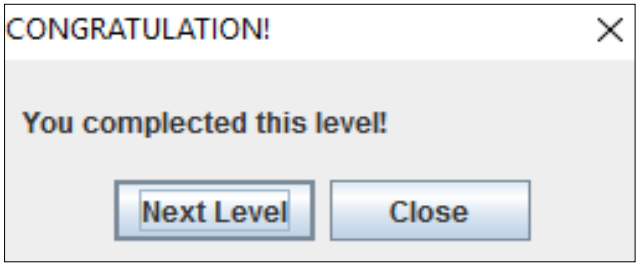
Attēls 7 rīkjosla



Attēls 8 spēles līmenis pēc pretinieka gājiena veikšanas



Attēls 9 spēles līmeņa mājiena parādīšana



Attēls 10 ziņa pēc līmeņa pareizas izpildīšanas

5. Pielikums Koda fragments

```
   public static void newTable() throws Piece.OccupiedSquareException {
        Game.newGame();
        table = new Table();
    }

    private Table() {
        gameFrame = new JFrame("Chess");
        gameFrame.setLayout(new BorderLayout());
        final JMenuBar tableMenuBar = createTableMenuBar();
        gameFrame.setJMenuBar(tableMenuBar);
        gameFrame.setSize(FRAME_DIMENSIONS);
        render();
        currentLevel = 0;
    }

/*-------------------------------------------------------------------
* This method is used for creating all new the components in
* gameFrame and then redrawing them back in frame
-------------------------------------------------------------------*/
    private void render() {
        if (this.boardPanel != null) {
            gameFrame.remove(this.sidePanel);
            gameFrame.remove(this.boardPanel);
        }

        this.boardPanel = new BoardPanel(); // panel for displaying board
        this.sidePanel = new JPanel(); // panel on right side for displaying
information
        this.sidePanel.setBackground(new Color(206, 165, 165));
        this.sidePanel.setPreferredSize(SIDE_PANEL_DIMENSIONS);
        JLabel sideLabel = new JLabel();
        sideLabel.setFont(new Font("Verdana", Font.BOLD, 15));
        JLabel colorLabel = new JLabel();

        if (Game.getHeroe() != null) {
            String color = Game.getHeroe().getColor() == Color.WHITE ? "WHITE"
: "BLACK";
            sideLabel.setText("You are playing as ");
            colorLabel.setText(color);
            colorLabel.setForeground(Game.getHeroe().getColor());
            colorLabel.setFont(new Font("Verdana", Font.BOLD, 20));
            this.sidePanel.add(sideLabel);
            this.sidePanel.add(colorLabel);
        }
        gameFrame.add(this.sidePanel, BorderLayout.EAST);
        gameFrame.add(this.boardPanel, BorderLayout.CENTER);
        gameFrame.setVisible(true);
        levelMoveList = Game.getLevelMoveList();
        iter = 0;
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                sidePanel.repaint();
                SwingUtilities.invokeLater(new Runnable() {
                    public void run() {
                        boardPanel.drawBoard();
                        makeComputersMove();
```

```
                    }
                });
            }
        });

    }

    private void makeComputersMove() {
        if (!levelMoveList.isEmpty()) {
            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                LOGGER.log(Level.SEVERE, e.toString(), e);
                Thread.currentThread().interrupt();
            }
            Game.NextMove move = Game.getLevelMoveList().get(iter); // getting
next right move
            Game.move(Game.getComputer(), move.getFromSquare(),
move.getToSquare(), iter); // making that move
            move.getFromSquare().setToMovedColor();
            move.getToSquare().setToMovedColor();
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    boardPanel.drawBoard();
                }
            });
            iter++;
        }
    }

    private JMenuBar createTableMenuBar() {
        JMenuBar tableMenuBar = new JMenuBar();
        tableMenuBar.add(createFileMenu());
        // additional menu bar items can be added later here
        return tableMenuBar;
    }

    private JMenu createFileMenu() {

        final JMenu fileMenu = new JMenu("File");
        final JMenuItem newGame = new JMenuItem("Start Game");
        final JMenuItem hint = new JMenuItem("Show Hint");
        final JMenuItem about = new JMenuItem("About");
        final JMenuItem exit = new JMenuItem("Exit");

        // setting action listeners for each button for specifying
        // what action will be made after that button is pressed
        newGame.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if(Game.newLevel("level1")) {
                        currentLevel = 1;
                        render();
                        hint.setEnabled(true);
                    }else{
                        showMessage("Could not load level \"level1\"","No
level found !");
                    }
```

```
                } catch (Piece.OccupiedSquareException ex) {
                    LOGGER.log(Level.SEVERE, ex.toString(), ex);
                }
            }
        });

        hint.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                showHint(Game.getHint());
            }
        });
        about.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    showMessage(getAbout(), "About");
                } catch (IOException ex) {
                    LOGGER.log(Level.SEVERE, ex.toString(), ex);
                }
            }
        });
        exit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                gameFrame.setVisible(false);
                gameFrame.dispose(); // getting rid of game frame
            }
        });
        hint.setEnabled(false); // disabling hint button, before any level is
loaded
        fileMenu.add(newGame);
        fileMenu.add(hint);
        fileMenu.add(about);
        fileMenu.add(exit);
        return fileMenu;
    }
```

Kvalifikācijas darbs „ Šaha taktikas treniņprogramma” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Krišjānis Kallings _____ ___.05.2017.

Rekomendēju darbu aizstāvēšanai
Darba vadītājs/a: Dr. dat. Jānis Zuters _____ ___.05.2017.

Recenzents: M.dat. Aleksandrs Zeļenkovs

Darbs iesniegts 29.05.2017.
Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: Darja Solodovņikova_____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē ___.06.2017. prot. Nr. _____

Komisijas sekretārs(-e): _____