УНИВЕРЗИТЕТ У НОВОМ САДУ
**ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ**

Марко Ђурђевић

# Шах.НЕТ

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, септембар 2023. године

## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

| | | | |
|---|---|---|---|
| Редни број, **РБР**: | | | |
| Идентификациони број, **ИБР**: | | | |
| Тип документације, **ТД**: | | Монографска публикација | |
| Тип записа, **ТЗ**: | | Текстуални штампани документ | |
| Врста рада, **ВР**: | | Завршни (bachelor) рад | |
| Аутор, **АУ**: | | Марко Ђурђевић | |
| Ментор, **МН**: | | Проф. др Драган Иветић | |
| Наслов рада, **НР**: | | Шах.НЕТ | |
| Језик публикације, **ЈП**: | | Српски / ћирилица | |
| Језик извода, **ЈИ**: | | Српски | |
| Земља публиковања, **ЗП**: | | Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | 2023 | |
| Издавач, **ИЗ**: | | Ауторски репринт | |
| Место и адреса, **МА**: | | Факултет Техничких Наука (ФТН), Д. Обрадовића 6, 21000 Нови Сад | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | | |
| Научна област, **НО**: | | Електротехничко и рачунарско инжењерство | |
| Научна дисциплина, **НД**: | | Примењено софтверско инжењерство | |
| Предметна одредница/Кључне речи, **ПО**: | | Шаховска апликација, шаховски програм | |
| **УДК** | | | |
| Чува се, **ЧУ**: | | Библиотека ФТН, Д. Обрадовића 6, 21000 Нови Сад | |
| Важна напомена, **ВН**: | | | |
| Извод, **ИЗ**: | | Шаховска апликација која омогућава кориснику да игра против снажног шаховског програма. | |
| Датум прихватања теме, **ДП**: | | | |
| Датум одбране, **ДО**: | | | |
| Чланови комисије, **КО**: | Председник: | Др Александар Купусинац, ред. проф. | |
| | Члан: | Др Дуња Врбашки, доцент | Потпис ментора |
| | Члан, ментор: | Др Драган Иветић, ред. проф. | |

## KEY WORDS DOCUMENTATION

| | | | |
|---|---|---|---|
| Accession number, **ANO**: | | | |
| Identification number, **INO**: | | | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual material, printed/CD | |
| Contents code, **CC**: | | Bachelor thesis | |
| Author, **AU**: | | Marko Đurđević | |
| Mentor, **MN**: | | Dragan Ivetić, PhD, full professor | |
| Title, **TI**: | | Chess.NET | |
| Language of text, **LT**: | | Serbian / Cyrillic | |
| Language of abstract, **LA**: | | Serbian | |
| Country of publication, **CP**: | | Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2010 | |
| Publisher, **PB**: | | Author reprint | |
| Publication place, **PP**: | | Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | | |
| Scientific field, **SF**: | | Electrical and computer engineering | |
| Scientific discipline, **SD**: | | Power Software Engineering | |
| Subject/Key words, **S/KW**: | | Chess application, chess program | |
| **UC** | | | |
| Holding data, **HD**: | | Library of the Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad | |
| Note, **N**: | | | |
| Abstract, **AB**: | | A chess application that allows the user to play against a strong chess program. | |
| Accepted by the Scientific Board on, **ASB**: | | | |
| Defended on, **DE**: | | | |
| Defended Board, **DB**: | President: | Aleksandar Kupusinac, PhD, full professor | |
| | Member: | Dunja Vrbaški, PhD, docent | Menthor's sign |
| | Member, Mentor: | Dragan Ivetić, PhD, full professor | |

| | | |
|---|---|---|
| | УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**<br>21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Број: |
| | **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА** | Датум: |

*(Податке уноси предметни наставник - ментор)*

| | |
|---|---|
| Врста студија: | ☐ Основне академске студије |
| Студијски програм: | Примењено софтверско инжењерство |
| Руководилац студијског програма: | Доц. Др Алекандар Селаков |

| | | | |
|---|---|---|---|
| Студент: | Марко Ђурђевић | Број индекса: | ПР 78/2018 |
| Област: | Електротехничко и рачунарско инжењерство | | |
| Ментор: | Проф. Др Драган Иветић | | |

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

## НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

**Шах.НЕТ**

## ТЕКСТ ЗАДАТКА:

Проучити шаховске програме и на основу стечених сазнања развити апликацију која омогућава кориснику да игра шах против одабраног шаховског програма.

| Руководилац студијског програма: | Ментор рада: |
|---|---|
| Доц. Др Александар Селаков | Проф. Др Драган Иветић |

Примерак за: o - Студента; o - Ментора

# Садржај

# Списак слика

# Скраћенице

| | |
|---|---|
| *WPF* | *Windows Presentation Foundation* |
| *MVVM* | *Model View View-Model* |
| *GUI* | *Graphical User Interface* |
| *DI* | *Dependency Injection* |
| UCI | *Universal Chess Interface* |

# 1. Увод

Шах се често сматра суштинском игром стратегије која је вековима фасцинирала и привлачила највеће умове света. У последњим годинама, спајање технологије и шаха довело је до нове ере шаховских игара, пружајући љубитељима игре никад пре видљиве могућности за забаву и развој вештина. Ова дипломска теза бави се развојем и анализом апликације за шах, која се у даљем тексту назива Шах.НЕТ. Апликација је дизајнирана да пружи корисницима јединствено искуство играња шаха.

Последњих година, интерес за компјутерски шах нагло расте. Томе је допринела епидемија корона вируса 2020-те године као и јако успешна *Netflix* серија *The Queen's Gambit* која је изашла исте године [1] [2]. Након завршетка епидемије раст се није зауставио, већ је, напротив, настављен. Најпопуларнији веб сајт за играње *Online* шаха *Chess.com* је јануара 2023. године доживео изненадни скок броја активних корисника који је узроковао да сервери веб сајта постану превише нестабилни. Број је неретко прелазио цифру од 10 милиона активних корисника у јануару 2023. године [2]. Данас је на *Chess.com*-у регистровано преко 148 милиона корисника [3].

Примарни циљ ове тезе је да представи свеобухватан преглед Шах.НЕТ апликације, са детаљима о њеним основним карактеристикама, функционалностима и основној технологији, укључујући интеграцију *Stockfish* шаховског програма.

Ова *C# WPF* апликација за шах омогућава корисницима да играју против *Stockfish* шаховског програма, једног од најснажнијих шаховских програма на свету. Пре играња, апликација тражи од корисника да изабере боју са којом ће играти. Након тога корисник мора изабрати јачину *Stockfish* шаховског програма против којега жели да игра. Ако корисник не жели ручно да изабере јачину *Stockfish* шаховског програма, може да игра низ шаховских слагалица како би се проценила потребна тежина *Stockfish* шаховског програма. Слагалице се оцењују на основу тежине, а оцена корисника се израчунава на основу броја тачно решених загонетки.

Када корисник добије оцену, може почети да игра против *Stockfish* шаховског програма. Апликација је дизајнирана да буде свеобухватан алат за шахисте. Може се користити за побољшање шаховских вештина, за тестирање шаховске вештине или једноставно за уживање у изазовној партији шаха.

Апликација је добро дизајнирана и једноставна за коришћење. Такође је скалабилна и може се користити од стране шахиста свих вештина.

# 2. Концептуално решење

У циљу приказа решења апликације у даљим поглављима ће се описати архитектура решења уз помоћ приказа одабраних софтверских образаца и случајева коришћења апликације.

## 2.1 Архитектура система

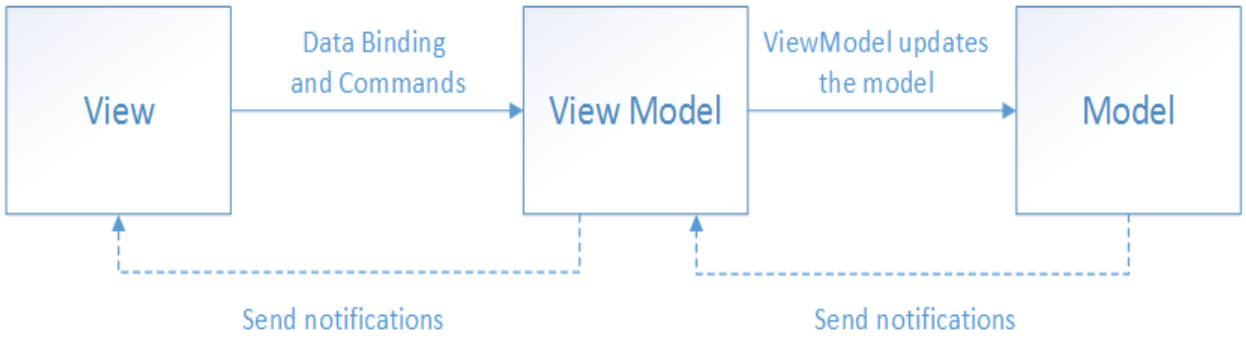
Апликација користи софтверски образац *MVVM*. *MVVM* образац је образац дизајна софтвера који одваја графички кориснички интерфејс *(GUI)* од логике апликације. Ово раздвајање компоненти чини апликацију лакшом за развој, тестирање и одржавање. У *MVVM* обрасцу постоје три основне компоненте:

- модел *(Model)*,
- приказ *(View)*, и
- модел приказа *(View-Model)*.

Сваки служи посебној сврси. Модел представља податке апликације. Он је одговоран за чување и управљање подацима. Приказ *(View)* је одговоран за приказивање података кориснику. Нема никаквог знања о логици апликације. Модел приказа *(View-model)* је одговоран за посредовање између модела и приказа. Он преводи захтеве из приказа у акције на моделу помоћу команди и обрнуто.

*MVVM* образац је добар избор за апликације које имају сложен графички кориснички интерфејс. Може помоћи да се графички кориснички интерфејс учини модуларнијим и лакшим за тестирање. Такође, може помоћи у побољшању перформанси апликације раздвајањем корисничког графички интерфејса од логике апликације [4].

У Шах.НЕТ апликацији, модел представља игру шаха. Чува информације о стању игре, као што је положај фигура на табли, статус тренутне игре и све потребне информације о слагалицама које корисник решава. *View* приказује ове податке на екрану корисника. *View-model* повезује модел и *view*. Он преводи радње корисника у команде које се шаљу моделу. Такође, ажурира приказ најновијим подацима из модела. Употреба *MVVM* обрасца у апликацији олакшава развој, тестирање и одржавање. Такође, чини апликацију модуларнијом и проширивом. Груба представа функционисања *MVVM* обрасца може се видети на слици 1.



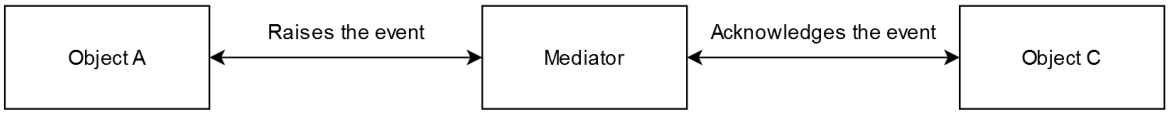

*Слика 1: Дијаграм MVVM обрасца*

Апликација користи класе посредника *(Mediator Pattern)* за комуникацију између модела приказа. Ово је добра пракса, јер омогућава да модели приказа буду одвојени један од другог. Образац посредника је образац дизајна софтвера који одваја објекте један од другог увођењем објекта посредника који координира комуникацију између њих. Ово чини апликацију флексибилнијим и одрживијом.

У Шах.НЕТ апликацији класе посредника омогућавају да модели приказа буду одвојени један од другог. Постоји неколико предности коришћења класа посредника:

- Раздвајање. Класе медијатора раздвајају објекте један од другог, што чини код флексибилнијим и одрживијим.
- Поновна употреба. Класе медијатора се могу поново користити у различитим деловима апликације, што смањује дуплирање кода.
- Тестабилност. Класе посредника чине код пробљивијим јер обухватају комуникацију између објеката. Коришћење класа медијатора за комуникацију између модела приказа је добра пракса која чини код флексибилнијим, одржаванијим, вишекратним и тестиранијим.

Посредничке класе се такође могу користити за имплементацију сложенијих комуникационих образаца, као што су комуникација један-према-више и више-према-више. То их чини моћним алатом за дизајнирање флексибилног кода који се може одржавати [5]. Груба представа функционисања *Mediator* обрасца може се видети на слици 2.




*Слика 2: Дијаграм Mediator обрасца*

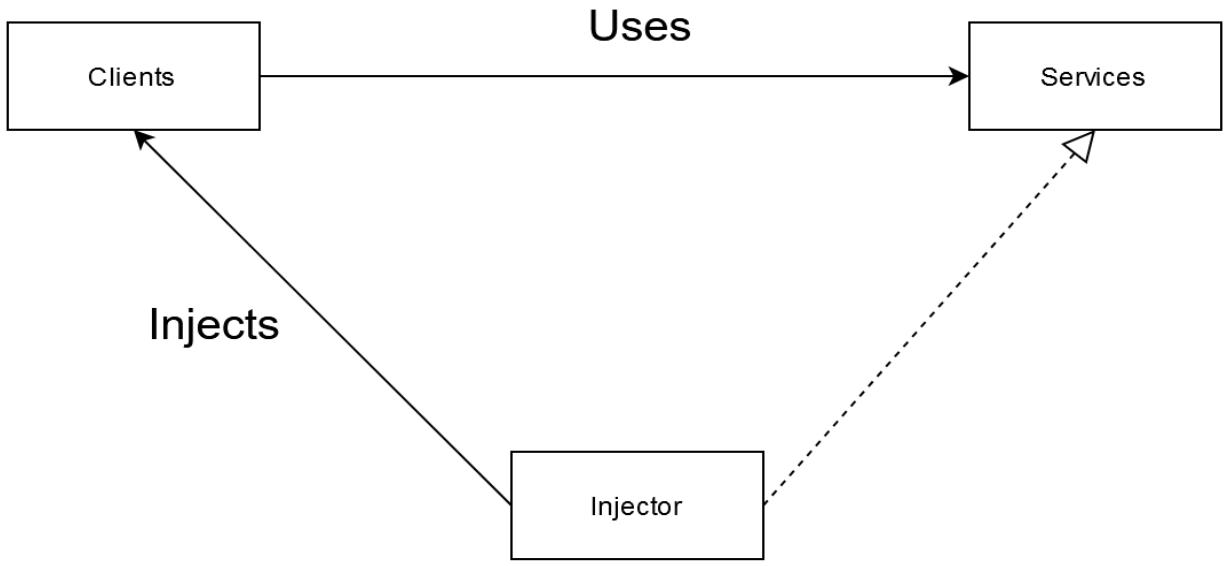
Апликација користи *Dependency Injection* софтверски образац да би убризгала потребне зависности у моделе приказа. Ово је добра пракса јер чини моделе приказа лабавије повезаним и лакшим за тестирање.

Убризгавање зависности је техника програмирања која чини класу независном од њених зависности. Ово је постигнуто тако што се зависности класе прослеђују у њеном конструктору, уместо да буду креиране унутар класе.

Убризгавање зависности чини код лабавије спрегнутим, што га чини флексибилнијим и одрживијим. Убризгавање зависности олакшава поновно коришћење кода јер можете убацити различите зависности у класу у зависности од контекста.

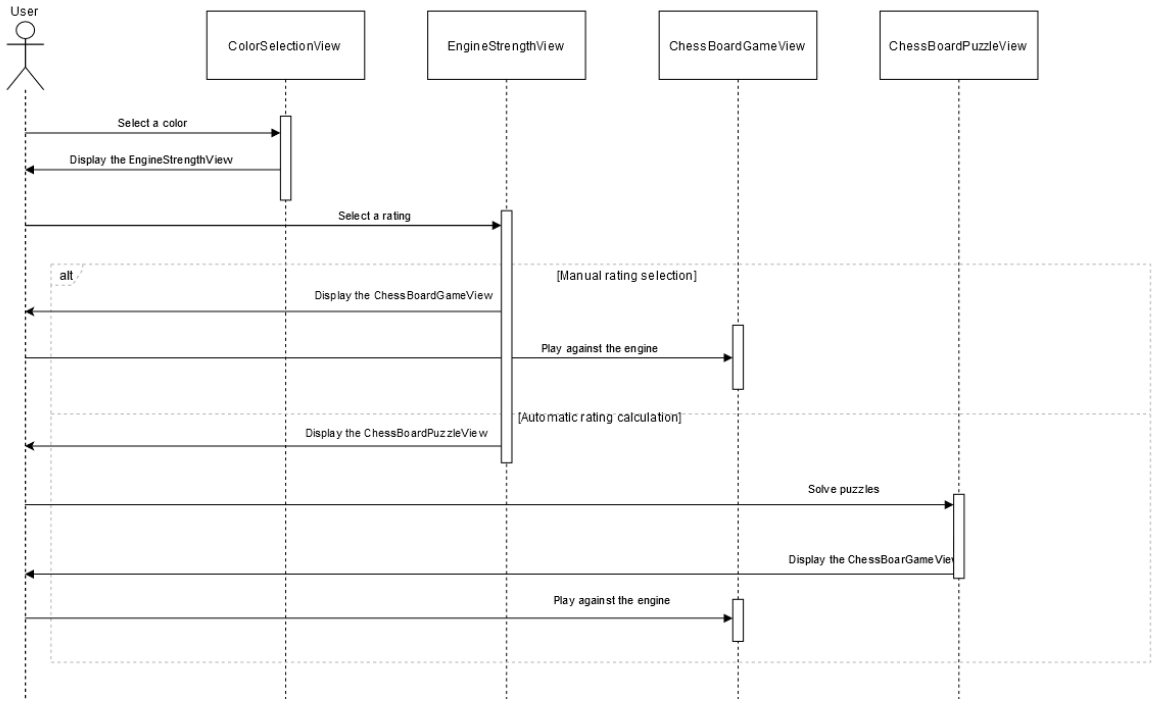
Коришћење убризгавања зависности за убацивање потребних зависности у моделе приказа је добра пракса која чини код лабавије повезаним и лакшим за тестирање [6].

У апликацији је примењен овај образац да би се побољшале могућности одржавања и проширивост. Убацивањем зависности, модели приказа остају лабаво повезани са конкретним изворима података. Не морају да знају како су подаци креирани или одакле долазе. Убризгавање зависности поједностављује тестирање јединица. Лако се могу заменити стварни подаци лажним или лажним имплементацијама током тестирања. Додавање нових података или модела прегледа постаје једноставно. Апликација се може лако проширити без претерано велике модификације постојећег кода. Груба представа функционисања *Dependency Injection* обрасца може се видети на слици 3.




*Слика 3: Дијаграм Dependency Injection обрасца*

Постоје два могућа начина извршавања апликације. Оба почињу тако што корисник одабере боју којом жели да игра против шаховског програма. Потом се испред корисника налазе две опције. Прва је да корисник ручно изабере јачину шаховског програма против којег жели да игра. А друга је да се процени шаховска јачина корисника преко шаховских слагалица која би кориснику најбоље одговарала. Након оба случаја корисник почиње да игра партију шаха против *Stockfish* Шаховског програма са изабраном или прорачунатом јачином. На слици 4 се може видети дијаграм случајева коришћења апликације.




*Слика 4. Дијаграм случајева коришћења апликације*

## 2.2 Одабир шаховског програма

Шаховски програм је рачунарски програм који анализира шаховске позиције и враћа најбољи потез за дату позицију. Да су рачунари шахисти, шаховски програми би били њихов мозак [7].

Историја шаховских програма почела је касних 1940-их и раних 1950-их година прошлог века уз допринос научника Клајда Шенона и Алана Туринга. Клајд Шенон је 1949. године објавио познати рад *Programming a Computer For Playing Chess*, док је Алан Туринг 1951. године развио *TurboChamp* који је сматран првим правим шаховским програмом. Алгоритам *TurboChamp*-а је био „сиров“. Логика је била заснована на најосновнијим шаховских правилима и алгоритам је могао да гледа само два потеза унапред. *TurboChamp* је био способан да игра исправан шах против људи али не да их и победи [8] [9].

Током 1960-их и 1970-их, алгоритми компјутерских програма су значајно побољшани. Основу је поставио научник Џон Вон Нојман, који је развио *MiniMax* алгоритам претраге, савршено прикладан за игру шаха (минимизира резултат једног играча док максимизира резултат другог). У деценијама које су уследиле, MiniMax претрага је побољшана напредним хеуристичким техникама и „итеративним продубљивањем“, што је постепено повећавало дубину MiniMax претраге [8].

Наредне две деценије напредак хардвера и софтвера је за собом повлачио напредак шаховских програма. Шаховски програми су способни и могли су победити јаке противнике, али никада нису могли победити велемајсторе. Све се то променило 1997. године када је један од најјачих шахиста свих времена Гери Каспаров изгубио меч против шаховског програма званог *Deep Blue* развијеног од стране *IMB*-а. У сваком случају, 1997. година је симболично означила крај људске доминације над шаховским програмима [his].

Међутим, наредних година, иако су шаховски програми константно напредовали, нису користили напредне алгоритме, већ су користили такозвану *Brute Force* калкулацију. Односно, гледали су пермутације свих могућих потеза у позицији и на основу најбољег крајњег учинка бирали резултујуће потезе. Шахисти су то брзо схватили и дошли до проналаска такозваних *Anti-Engine* потеза који су у одређеним позицијама збуњивали шаховске програме и понекад им доносили победе против истих [10].

Поменути начин програмирања шаховских програма је прекинут развојем *AlphaZero* шаховског програма од стране *Google*-а. Оно што разликује *AlphaZero* од традиционалних шаховских програма је то што је заснован на неуронским мрежама, што је у суштини начин на који људски мозак функционише. Програм учи кроз међусобно повезане чворове, "неуроне", у процесу познатом као дубоко учење *(Deep Learning)*. Док традиционални шаховски програми уче кроз класично алфа-бета орезивање *(Alpha-Beta Pruning)*, *MiniMax* претрагу и *Brute Force* калкулисање, програми попут *AlphaZero*-а уче кроз поновљено само-играње, понекад играјући стотине милиона игара против себе. Сви најмоћнији шаховски програми су усвојили ову врсту алата за обраду информација и постали још моћнији [10].

Данас су шаховски програми много јачи од људи, а најбољи од њих достижу процењену *ELO* (нумерички број који представља шаховску вештину играча) оцену вишу од 3500 поена. Поређења ради, најјачи шахиста данашњице и вероватно свих времена, Магнус Карлсен, има шаховску оцену од 2839 [11]. Доступни су многи шаховски мотори, али само неколико њих се континуирано појављује у највишим ранговима компјутерских шампионата.

*Stockfish* је тренутно најјачи шаховски програм доступан јавности. Као шаховски програм отвореног кода, читава заједница људи помаже у његовом развоју и побољшању. Као и многи други, *Stockfish* је укључио неуронске мреже у свој код како би направио још боље процене шаховских позиција. *Stockfish* је доступан јавности на свим главним платформама као што су *Windows, Mac OS, Linus, iOS* и *Android* [12]. На слици 5 се може видети табела најбољих шаховских програма у септембру 2023. године.

| Rank | Name | Rating | | | Score | Average Opponent | Draws | Games |
|---|---|---|---|---|---|---|---|---|
| | | Elo | + | − | | | | |
| 1 | Stockfish 20230613 64-bit 4CPU | 3545 | +21 | −21 | 65.2% | −84.1 | 69.3% | 606 |
| 2 | Dragon by Komodo 3.2 64-bit 4CPU | 3526 | +13 | −13 | 62.2% | −67.9 | 74.4% | 1678 |
| 3 | Berserk 11.1 64-bit 4CPU | 3489 | +14 | −14 | 56.2% | −33.5 | 79.1% | 1328 |
| 4 | Chess System Tal 2.00 Elo 64-bit 4CPU | 3485 | +16 | −16 | 55.3% | −27.7 | 82.4% | 930 |
| 5-6 | Ethereal 14.00 64-bit 4CPU | 3482 | +13 | −13 | 54.9% | −25.5 | 83.4% | 1450 |
| 5-6 | Igel 3.5.0 64-bit 4CPU | 3482 | +17 | −17 | 54.5% | −24.3 | 83.9% | 880 |
| 7 | RubiChess 20230410 64-bit 4CPU | 3471 | +14 | −14 | 52.9% | −14.5 | 82.9% | 1270 |
| 8 | Revenge 3.0 64-bit 4CPU | 3468 | +11 | −10 | 52.9% | −16.8 | 81.9% | 2274 |
| 9 | Caissa 1.12 64-bit 4CPU | 3467 | +22 | −22 | 49.8% | −0.1 | 87.8% | 490 |
| 10 | Koivisto 9.0 64-bit 4CPU | 3466 | +13 | −13 | 52.7% | −14.2 | 82.6% | 1552 |
| 11 | Clover 6.0 64-bit 4CPU | 3458 | +17 | −17 | 48.4% | +8.3 | 80.1% | 830 |
| 12 | SlowChess Blitz 2.9 64-bit 4CPU | 3457 | +11 | −11 | 52.3% | −13.7 | 81.4% | 2354 |
| 13-14 | Deep Sjeng 3.6a16 64-bit 4CPU | 3455 | +18 | −17 | 50.4% | −3.0 | 87.7% | 780 |
| 13-14 | Rebel 16.2 64-bit 4CPU | 3455 | +14 | −14 | 50.8% | −4.3 | 82.5% | 1356 |
| 15 | Arasan 24.0 64-bit 4CPU | 3450 | +22 | −22 | 53.6% | −19.9 | 77.5% | 510 |

*Слика 5. Најбољи шаховски програми септембра 2023. године [13]*

Управо, због горе наведених разлога, је одлучено да се у апликацију интегрише *Stockfish* шаховски програм. Међутим, постојао је један проблем. Сам *Stockfish*, као најјачи шаховски програм, нема могућност да се подеси на шаховску јачину слабију од 1300 *ELO* поена. Преко 50% играча шаха су почетници и не поседују велико шаховско знање. То би значило да преко 50% активних шахиста не би имало никакву шансу да победи шаховски програм, односно, не би имали лепо искуство са Шах.НЕТ апликацијом. Да би се овај проблем превазишао, убачен је додатни шаховски програм, односно, *Fairy Stockfish*.

*Fairy Stockfish* је варијанта шаховског програма, изведена из *Stockfish*-а, дизајнирана за подршку варијанти вилинског шаха и за лаку проширивост на више варијанти шаха. Подржава различите регионалне, историјске и модерне варијанте шаха, као и игре са кориснички дефинисаним правилима. Циљ *Fairy Stockfish*-а је да подржава

велики избор игара налик шаху, опремљен моћном претрагом *Stockfish* -а. Такође, може се подесити испод 1300 *ELO* поена [14].

Одлука против којег шаховског програма ће корисник играти је заснована на изабраној јачини програма. Обичан *Stockfish* шаховски програм ће се користити за јачине од преко 1250 *ELO* поена, док ће се за јачине испод 1250 *ELO* поена користити *Fairy Stockfish*. Ово се ради како би се осигурало да сваки корисник, без обзира на његово знање има фер шансе за победу. Поменута имплементација представља фер и уравнотежен начин коришћења два различита шаховска програма у истој апликацији.

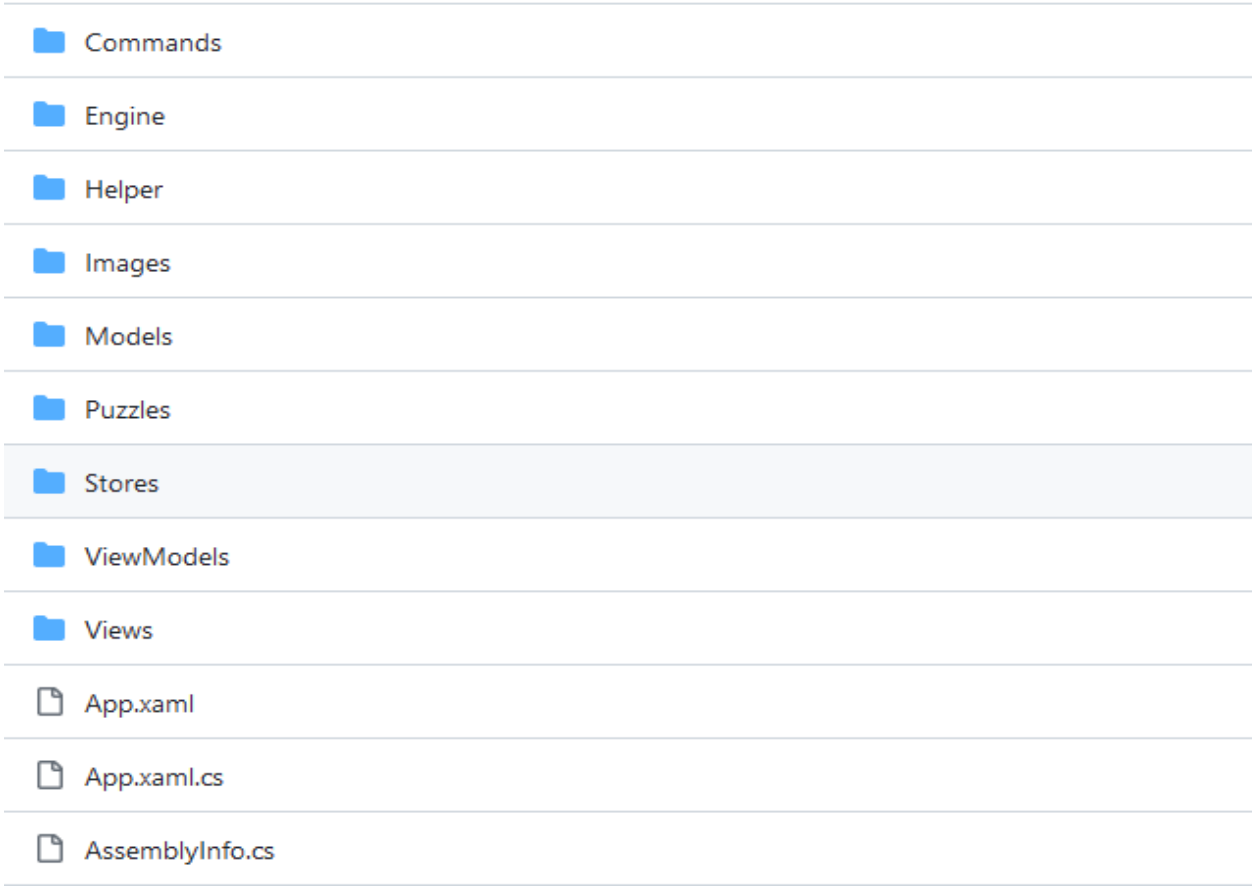
# 3. Имплементација

Програмски језик који је се користио при изради апликације је *C#*. Разлог због кога је одлучено да се апликација развија у овом програмском језику јесте што је *C#* моћан и свестран програмски језик који је веома погодан за развој разних апликација, укључујући и игре. Одабрана је технологија *.NET* јер пружа свеобухватан скуп алата и библиотека за развој и покретање апликација [15]. G*UI Framework* који је одабрат је *WPF (Windows Presentation Foundation)* јер пружа скуп контрола и функција које се могу користити за креирање богатих и интерактивних графичких корисничких интерфејса на једноставан и ефикасан начин [16]. Развојно окружење у којем је се апликација развијала је *Visual Studio 2022*.

Прво је дизајнирам и направљен графички кориснички интерфејс апликације помоћу *WPF*-а. Графички кориснички интерфејс апликације је направљен тако да буде једноставан за коришћење и навигацију. Такође, направљена је и навигација између прозора.

Након тога, имплементирана је логика шаха. Примењена су и испоштована сва шаховска правила као што су прорачунавање легалних потеза, детекција шах-мата, детекција нерешених партија услед недостатка довољно фигура на шаховској табли и детекција пата.

Последњи корак је представљао интегрисање *Stockfish* шаховског програма у апликацију, подешавање истог да игра против корисника и подешавање редоследа играња потеза. На слици 6 је се може видети структура апликације.




*Слика 6. Структура шаховске апликације*

Са слике 6 се могу приметити следеће ставке:

- *View* датотека која садржи све прегледе апликације.
- *Model* датотека која представља податке у току рада апликације.
- *ViewModel* датотека која повезује *View* и *Model*.
- *Command* датотека помоћу које *ViewModel* и *View* комуницирају.
- *Engine* датотека која садржи извршне фајлове шаховских програма као и класе које служе за управљање шаховским програмом и његовим подешавањем јачине.
- *Images* датотека која садржи све потребне слике за изглед апликације као што су шаховске фигуре и иконице.
- *Puzzles* датотека која садржи све потребне слагалице за процењивање шаховске јачине корисника.
- *Stores* датотека која сарджи све потребне *Mediator* класе помоћу којих *ViewModel*-и комуницирају.
- *Helper* датотека која садржи статичке ресурсе као што су конвертовање координата шаховских поља у ред и колону матрице.
- *App.cs* класа која покреће апликацију и убацује све потребне зависности у *ViewModel*-е.

Као што је раније напоменуто, у апликацији је примењен *DI* софтверски образац. Примена *DI* обрасца се може видети на листнгу кода 1 у класи *App.cs*.

```
partial class App : Application
{
   protected override void OnStartup(StartupEventArgs e)
   {
        var colorSelectionStore = new ColorSelectionStore();
        var engineStrengthStore = new EngineStrengthStore();
        var engineStrengthEvaluatedStore = new EngineStrengthEvaluatedStore();
        var newGameStore = new NewGameStore();

        var colorSelectionViewModel = newColorSelectionViewModel(colorSelectionStore);
        var engineStrengthViewModel = new EngineStrengthViewModel(engineStrengthStore);
        var chessBoardPuzzleViewModel = new
                    CheessBoardPuzzleViewModel(engineStrengthEvaluatedStore);
        var chessBoardGameViewModel = new ChessBoardGameViewModel(newGameStore);

        var mainViewModel = new MainWindowViewModel(colorSelectionViewModel,
                    engineStrengthViewModel, chessBoardPuzzleViewModel,
              chessBoardGameViewModel, colorSelectionStore, engineStrengthStore,
              engineStrengthEvaluatedStore, newGameStore);

        MainWindow = new MainWindow()
        {
            DataContext = mainViewModel
        };
        MainWindow.Show();

        base.OnStartup(e);
   }
}
```

*Листинг кода 1. Примена Dependency Injection Обрасца у апликацији*

Као што се може видети са листинга кода 1 све *ViewModel* и *Mediator* класе се инстанцирају у методи *OnStartup* која се извршава тачно једном приликом покретања апликације. Коришћењем *DI* обрасца је осигурано да ће све *ViewModel* и *Mediator* класе бити инстанциране тачно. Једини изузетак је *PromotionWindow ViewModel*, који ће се инстанцирати сваки пут када корисник жели да промовише пијуна у жељену фигуру.

Апликација користи 6 приказа:

- *MainView*. Овај приказ контролише све остале приказе у апликацији. Одговоран за приказ главног менија и навигацију између других приказа.
- *ColorSelectionView*. Овај приказ омогућава кориснику да изабере боју којом жели да игра.
- *EngineStrengthView*. Овај приказ омогућава кориснику да одабере јачину шаховског програма против којег жели да игра. Јачина се може одабрати ручно или се може проценити на основу успешности решавања шаховских слагалица.
- *ChessBoardPuzzleView*. Овај приказ омогућава кориснику да решава шаховске загонетке. Загонетке се оцењују на основу броја решених загонетки корисника.
- *ChessBoardGameView*. Овај приказ омогућава кориснику да игра шаховску партију против шаховских програма. Снага шаховског програма је одређена избором направљеним у *EngineStrengthView* или резултатом шаховских слагалица.

- *PromotionWindowView*. Овај приказ омогућава кориснику да изабере фигуру у коју жели да промовише свог пешака.

Сваки приказ у апликацији је контролисан од стране сопственог модела приказа. Приказа и модел приказа који га контролише деле исто основно име, али модел приказа има суфикс *ViewModel*, док приказ има суфикс *View*. На пример, *ColorSelectionView* приказ има *ColorSelectionViewModel* који га контролише.

Апликација има пет модела:

- *ChessSquare*. Овај модел представља шаховска поље на табли. Чува координате поља и фигуру која се тренутно налази на пољу.
- *Piece*: Овај модел представља шаховску фигуру. Чува опис фигуре која се тренутно налази на шаховском пољу.
- *PuzzleManager*: Овај модел представља све загонетке које корисник може да реши и њихове резултате. Чува слагалице, тежину и резултат корисника.
- *GameStatusDisplay*: Овај модел прати стање игра и ажурира га.
- *FlipBoard*: Овај модел чува оријентацију табле на основу боје којом корисник игра.

Модели су одговорни за чување података апликације. Они су одвојени од приказа и модела приказа, што олакшава њихово тестирање јединица и промену корисничког интерфејса без утицаја на остатак система.

Класа *StockfishManager* је посебна класа која је одговорна за комуникацију са *Stockfish* шаховским програмом. Не наслеђује ниједан од модела или модела приказа. Класа *StockfishManager* је одговорна за следеће задатке:

- Иницијализација *Stockfish* шаховског програма. Класа *StockfishManager* је одговорна за иницијализацију *Stockfish* шаховског програма и постављање његових параметара.
- Слање захтева *Stockfish* шаховског програма. Класа *StockfishManager* је одговорна за слање захтева *Stockfish* шаховског програма, као што је добијање најбољег потеза за одређену позицију.
- Пријем одговора *Stockfish* шаховског програма: Класа *StockfishManager* је одговорна за примање одговора од *Stockfish* шаховског програма, као што је прорачунат најбољи потез за одређену позицију.

*StockfishManager* класа је одвојена од модела и модела приказа и остатка система, што олакшава њено тестирање и промену графичког корисничког интерфејса без утицаја на комуникацију са *Stockfish* шаховским програмом.

Класе *Mediator*-а се користе за раздвајање модела приказа и за комуникацију између њих. Ово се ради пружањем централне тачке моделима приказа да комуницирају једни са другима. Класе посредника су одговорне за рутирање порука између модела приказа. Ово раздвајање олакшава тестирање и одржавање апликације јер модели приказа не морају да знају једни о другима. Такође, олакшава одржавање апликације јер ако се треба променити начин комуникације модела приказа, потребно је само променити класу

посредника. Док моделе приказа није потребно мењати. На додатку А у листингу 2 се може видети примена класи посредника.

На додатку А се могу приметити класа посредника *ColorSelectionStore*, класа *ColorSelectionViewModel*, део конструуктора класе *MainWindowModel* и метода *ColorSelected* класе *MainWindowModel*. Класа посредника *ColorSelectionStore* служи за комуникацију између *MainWindowModel*-а и *ColorSelectionViewModel*-а модела приказа. Класа *MainWindowModel* иницијалицује класу посредника у конструктору и преплаћује методу коју жели да позове када је класа посредника обавести. Када корисник изабаре боју *ColorSelectionViewModel* ће подићи догађај који ће се касније извршити методу *ColorSelected*. У њој се подеси изабрана боја корисника и промени поглед на следећи приказ којим корисник бира јачину шаховског програма.

Начин на који се одређују сви легални потези фигура биће приказан на листингу кода 3.

```
private void UpdateAvailableMoves()
   {
      if (SelectedSquare?.Piece == null) return;

      List<int> original;
      if (EnPassantPossibilty)
      {
         original = SelectedSquare.Piece.GetPossibleMoves(SelectedSquare,
                                                          ChessSquares.ToList());
      }
      else
      {
         original = SelectedSquare.Piece.GetPossibleMoves(SelectedSquare,
                                              ChessSquares.ToList(), EnPassantSquare);
      }

      var final = AreMovesValid(original, SelectedSquare);

      HighlightedSquares.AddRange(final);
      foreach (var t in HighlightedSquares)
      {
         ChessSquares[t].Color = "#964B00";
      }
   }
```

*Листинг кода 3. Алгоритам за добијање легалних потеза*

На листнигу кода 3 се, након што је изабрана жељена фигура корисника, позива функција која израчунава све могуће потезе за изабрану фигуру, неузимајући у обзир сигурност сопственог краља. Након тога, позива се функција *AreMovesValid* која одстрањује потезе који доводе краља у опасност. Алгоритам функије *AreMovesValid* ради тако што за сваки могући потез направи тај потез на копији шаховске табле, провери да ли је краљ у тој позицији у опасности, и ако јесте, одстрани га из листе легалних потеза.

На сличан начин раде алгоритми за детекцију шах-мата и пата, користе исте функције. Једина разлика је у томе што се броји укупан број потеза које играч може да одигра. Ако је укупан број потеза једнак нули и краљ је нападнут тада је дошло до шах-мата, а ако је укупан број потеза један нули и краљ није у опасности онда је дошло до пата у позицији.

На листингу кода 4 се налази функција која провера реми услед недостатка фигура на табли.

```
private bool IsDrawByInsufficientMaterial()
{
   var whitePieces = GetRemainingWhitePieces();
   var blackPieces = GetRemainingBlackPieces();

   if (whitePieces.Count > 3 || blackPieces.Count > 3) return false;

   if (whitePieces.Count == 1 && blackPieces.Count == 1) return true; // K vs K
   if (KingBishopVsKing(whitePieces, blackPieces)) return true; // K B vs K
   if (KingKnightVsKing(whitePieces, blackPieces)) return true; // K N vs K
   if (MinorPieceDraw(whitePieces, blackPieces)) return true; // K M vs K M
   if (KingTwoKnightsVsKing(whitePieces, blackPieces)) return true; // K N N vs K

   return false;
}
```

*Листинг кода 4. Функција која проверава да ли не нерешена партија услед недостатка фигура*

Партија је нерешена услед, недостатка довољно фигура, ако не постоји било каква комбинација легалних потеза који могу довести до шах-мата [17]. Комбинације фигура које доводе до ремија су следеће:

- краљ против краља,
- краљ и ловац против краља,
- краљ и скакач против краља,
- краљ и скакач против краља и ловца,
- краљ и скакач против краља и скакача,
- краљ и ловац против краља и ловца, и
- краљ и два скакаца против краља.

Алгоритам са листинга кода 4 врши проверу после сваког потеза корисника. Врло је брз и једноставан. Прво се добаве све преостале беле и црне фигуре из позиције, и након тога се проверава да ли се комбинација преосталих фигура поклапа са случајевима који задовољавају нерешену партију.

Још једна битна ставка имплементације би биле асинхроне методе. Асинхроне методе у *C#*-у омогућавају да се код истовремено извршава без блокирања позивајуће нити. Ово је посебно корисно у сценаријима у којима се обављају дуготрајне операције, као што су задаци везани за мрежу, а да не доведе до тога да се графички кориснички интерфејс апликације замрзне [18].

У апликацији се асинхроне методе користе приликом комуницирања са шаховским програмом. У зависности од подешене јачине шаховског програма, операције могу трајати јако дуго, понекад више и од десет секунди. Наравно, оволико дуго смрзавање графичког корисничког интерфејса не би било прихватљиво. Пример коришћења асинхроне методе у апликацији се може видети на листингу кода 5.

```
private async Task GetEngineMoveAsync(string playerMove)
{
   string engineMove;

   if (playerMove.Equals(string.Empty)) // prvi potez igra engine
   {
      engineMove = await Task.Run(() => StockfishManager.GetBestMove());
   }
   else // svi ostali potezi
   {
      engineMove = await Task.Run(() => StockfishManager.GetBestMove(playerMove));
   }

   MakeEngineMove(engineMove);

   ...
}
```

*Листинг кода 5. Пример коришћења асинхроне методе у апликацији*

Суштински, направљене су две измене које ову методу чине асинхроном. Прва измена је та да је потпис функције другачији, односно, уместо *void* кључне речи сарджи *async Task*. Ово омогува позивачу да ову методу позове асинхроно са кључном речи await, што значи да позвана метода неће блокирати главну извршну нит програма и графички кориснички интерфејс ће остати респонзиван. Друга измена је направљена у начину позивања методе која добија најбољи потез од шаховског програма. Уместо стандардног начина позивања методе, направљен је *Task* који то ради асинхроно, притом, не блокирајућу остатак кода методе који не зависи од истог позива.

## 3.1 Комуницирање апликације са шаховским програмом

Апликација комуницира са *Stockfish* шаховским програмом преко *UCI (Universal Chess Interface)* протокола. *UCI* протокол је стандардизовани комуникациони протокол који се најчешће користи у компјутерским шаховским апликацијама. Служи за комуникацију између шаховског графичког корисничког интерфејса и шаховског програма, олакшавајући им интеракцију и омогућавајући корисницима да играју шаховске партије против компјутерских противника.

*UCI* обезбеђује заједнички скуп правила и команди којих би шаховски програми и графички кориснички интерфејси требало да се придржавају. Ова стандардизација обезбеђује интероперабилност између различитих шаховских програма и интерфејса.

*UCI* протокол укључује две главне компоненте, а то су шаховски програм и графички кориснички интерфејс. Графички кориснички интерфејс је одговоран за приказивање шаховске табле, руковање корисничким уносом и приказ информација о игри, док шаховски програм израчунава покрете и процењује позиције.

Комуникација између графичког корисничког интерфејса и шаховског програма се се одвија кроз низ текстуалних наредби и одговора. Графички кориснички интерфејс шаље команде шаховском програму, као што су *position* (за постављање жељене позиције) и *go* (да би шаховски програм кренуо да рачуна најбољи потез). Шаховски програм одговара информацијама о својим прорачунима, најбољим потезима и проценама.

*UCI* дефинише скуп стандардних команди и опција које шаховски програми могу разумети и на њих реаговати. То укључује команде за конфигурисање шаховског програма, постављање позиција и анализу игара.

*UCI* користи стандардну алгебарску нотацију за представљање шаховских потеза, што олакшава разумевање и рад са њима и за људе и за шаховске програме.

*UCI* је најпопуларнији и најзаступљенији протокол за развијање шаховских апликација, што га чини пожељним избором за програмере који креирају шаховске апликације [19].

Шаховски програми су често обичне конзолне апликације, које комуницирају помоћу текстуалних команди, као што је случај са *Stockfish* шаховским програмом. На листингу кода 6 се може видети покренути извршни фајл *Stockfish* шаховског програма и пример његовог коришћења.

```
Stockfish 16 by the Stockfish developers (see AUTHORS file)
uci
id name Stockfish 16
id author the Stockfish developers (see AUTHORS file)

option name Threads type spin default 1 min 1 max 1024
option name Hash type spin default 16 min 1 max 33554432
option name Ponder type check default false
option name Skill Level type spin default 20 min 0 max 20
option name UCI_LimitStrength type check default false
option name Use NNUE type check default true
uciok
isready
readyok
setoption name UCI_LimitStrength value true
setoption name Skill Level value 15
position startpos moves e2e4
go depth 5 movetime 100
info string NNUE evaluation using nn-5af11540bbfe.nnue enabled
info depth 1 seldepth 1 multipv 1 score cp -47 nodes 106 nps 17666 hashfull 0 tbhits 0
time 6 pv e7e5
info depth 2 seldepth 2 multipv 1 score cp -47 nodes 187 nps 26714 hashfull 0 tbhits 0
time 7 pv e7e5
info depth 3 seldepth 2 multipv 1 score cp -13 nodes 360 nps 45000 hashfull 0 tbhits 0
time 8 pv g8f6
info depth 4 seldepth 2 multipv 1 score cp -13 nodes 906 nps 100666 hashfull 0 tbhits
0 time 9 pv g8f6
info depth 5 seldepth 3 multipv 1 score cp -14 nodes 1396 nps 139600 hashfull 0 tbhits
0 time 10 pv g8f6 b1c3
bestmove d7d5 ponder e4d5
```

*Листинг кода 6. Пример комуникације са шаховским програмом путем UCI протокола*

Први корак у комуникацији, након што се шаховски програм покрене, је да се шаховски програм подеси на *UCI* протокол. То се ради помоћу *uci* команде. Шаховски програм треба да врати следеће:

- свој јединствени идентификатор,
- списак свих опција које подржава и њихове подразумеване вредности, и
- текст *uciok* који потврђује кориснику да је шаховски програм прешао на коришћење *UCI* протокола.

Након тога потребно је шаховском програму послати команду *isready*, а шаховски програм треба да врати *readyok*. Ова команда се користи за синхронизацију шаховског програма са графичким корисничким интерфејсом и осигурава да су сви унутрашњи параметри шаховског програма подешени.

Следећи корак би чинио подешавање унутрашњих параметара шаховског програма које је шаховски програм навео након *uci* команде. За потребе апликације, од интереса су две команде а то су UCI_*LimitStrength* и *Skill Level*. Прва команда говори шаховском програму да дозволи смањивање шаховске јачине, док друга команда смањује.

Након тога је ред на описивање позиције. То се ради помоћу *position* команде. У конкретном примеру са листинга кода 6, у питању је почетна позиција у којој је одигран један потез e2e4, односно, одигран је први потез и померен је бели пешак са поља e2 на поље e4.

Након тога, све је подешено за прорачунавање потеза, и шаховском програму се може послати команда *go* која му говори да крене у анализирање позиције и тражење најбољег потеза. Команда *go* се комбинује са доста параметара који шаховском програму говоре како да прорачунава дату позицију, али за потребе апликације потребна су само два параметра. Први је *depth* који говори шаховском програму колико потеза да гледа унапред. Други је *movetime* који говори шаховском програму колико времена има за прорачун потеза изражен у милисекундама. Након извршене комаде *go* шаховски програм ће кренути у прорачун о којем ће редовно обавештавати корисника, као што се може видети на листингу кода 6. Ако је постигнута довољна дубина или ако је време за прорачун истекло, шаховски програм ће вратити најбољи потез. Поред најбољег потеза, вратиће још један потез са ознаком *ponder*. То значи да шаховски програм очекује тај потез као одговор, и у позадини прорачунава одговор на тај потез, што потенцијално може довести до тога да се време рачунања доста смањи за следећи потез.

Све што је објашњено на претходном листингу кода 6, требало је извести у програмском коду. За то је заслужна *StockfishManager* класа. Класа *StockfishManager* је прилагођена класа која пружа згодан и ефикасан начин комуникације са *Stockfish* -ом. Она апстрахује детаље *UCI* протокола, што олакшава коришћење *Stockfish*-а у било којој апликацији. Пример иницијализације *StockfishManager* класе може се видети на листингу кода 7.

```
public  StockfishManager(int engineStrength)
{
   _position = "position startpos moves ";
   _engineStrength = engineStrength;
   string chosenEngine = _engineStrength > 1250 ? "Stronger" : "Weaker";

   var currentDirectory = Directory.GetCurrentDirectory();
   var targetFolder = Path.Combine(currentDirectory, "..", "..", "..", "Engine");

   _stockfishProcess = new Process();
   _stockfishProcess.StartInfo.FileName = Path.Combine(targetFolder,
                                                StockfishSetting.Engine[chosenEngine]);
   _stockfishProcess.StartInfo.UseShellExecute = false;
   _stockfishProcess.StartInfo.RedirectStandardInput = true;
   _stockfishProcess.StartInfo.RedirectStandardOutput = true;
   _stockfishProcess.StartInfo.CreateNoWindow = true;

   _stockfishProcess.Start();
   _stockfishInput = _stockfishProcess.StandardInput;
   _stockfishOutput = _stockfishProcess.StandardOutput;

   SendCommand("uci");
   SendCommand("isready");

   SendCommand("setoption name UCI_LimitStrength value true");
   SendCommand(StockfishSetting.Setting[_engineStrength][0]);
}
```

*Листинг кода 7. Иницијализација StockfishManager класе*

Прво се подеси стринг који представља почетну позицију. Након тога, на основу параметара конструктора, који представља јачину шаховског програма, се одређује који шаховски програма ће бити употребљен. Покреће се шаховски програма у виду екстерног процеса који комуницира са апликацијом путем стандардног текста. Након што је шаховски програм покренут, подешава се индентичним командама као у претходном примеру. На листингу кода 8 се може видети метода *GetBestMove* класе *StockfishManager* која добија најбољи потез за дату позицију.

```
public string GetBestMove(string move = "")
{
    string depth = StockfishSetting.Setting[_engineStrength][1];
    string movetime = StockfishSetting.Setting[_engineStrength][2];

    _position += move;
    SendCommand(_position);
    var bestMove = GetMove($"go depth {depth} movetime {movetime}");
    _position += $"{bestMove} ";

    return bestMove;
}
```

*Листинг кода 8. Метода за добијање најбољег потеза шаховског програма.*

Прво се на стринг који описује позицију дода корисников последњи потез. Након тога се ажурира позиција коју памти шаховски програм. Па се пронађе најбољи потез командом *go* која има своје параметре *depth* и *movetime* на основу јачине коју је корисник изабрао. Када се добије најбољи потез, он се налепи на остатак стринга и врати графичком корисничком интерфејсу који касније треба да интерпретира добијени потез.

## 3.2 Процењивање шаховске јачине корисника помоћу слагалица

Врло битна ставка апликације је алгоритам за процену шаховске јачине корисника помоћу шаховских слагалица. Алгоритам динамички прилагођава јачину шаховског програма на основу корисниковог учинка у решавању шаховских слагалица. Заснива се на броју тачно решених загонетки и броју учињених грешака. Рад алгоритма је следећи:

Алгоритам иницијализује јачину корисника на 250, представљајући његову почетну шаховску јачину. Представља кориснику шаховске загонетке које мора да реши, Загонетке су представљене растућим редоследом, са јачином у распону од 250 до 2800 са корацима раста од 50. Задатак корисника је да тачно реши сваку загонетку. Ако корисник успе да реши загонетку, његова шаховска јачина се повећава за 50 поена, а алгоритам представља следећу слагалицу. Ако корисник направи грешку при решавању слагалице, његова оцена остаје непромењена, а алгоритам представља следећу слагалицу. Овај процес се наставља све док корисник или успешно не реши последњу загонетку или не направи три грешке. Након завршетка серије загонетки или достизања границе од три грешке, процена шаховске јачине корисника је завршена.

Алгоритам има неколико предности:

- Прогресивни напредак. Корисник се суочава са загонеткама све веће тежине, омогућавајући постепени напредак у својим шаховским вештинама.
- Тренутна повратна информација: Непосредне повратне информације се пружају након сваке слагалице, подстичући корисника да учи из својих грешака и да се побољша.
- Прилагођена оцена: Алгоритам додељује оцену која тачно одражава способности корисника да решава шаховске слагалице.
- Критеријуми завршетка: Алгоритам се прекида или након завршетка свих загонетки или када корисник направи три узастопне грешке, обезбеђујући фер и свеобухватну процену.

Алгоритам процењивања корисникове јачине је саставни део апликације, који служи за процену и мотивацију корисника док унапређују своје вештине решавања шаховских слагалица. Његова динамична природа, прогресивни изазов и тренутне повратне информације доприносе награђивачком и занимљивом корисничком искуству.

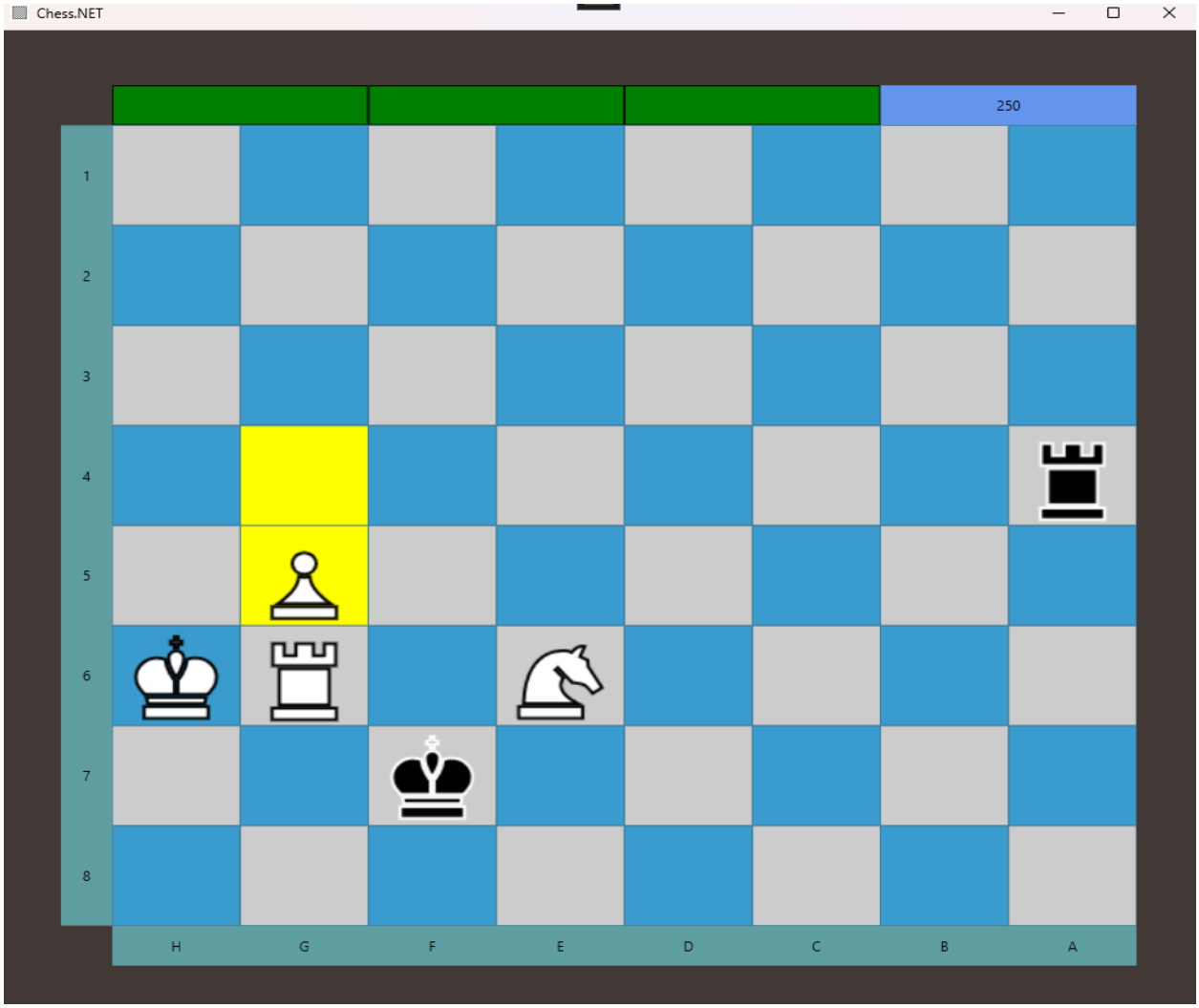
## 3.3 Интерна база података

Апликација не користи конвенционалну базу података за складиштење шаховских слагалица. Уместо тога, користи једноставну текстуалну датотеку. Ово је урађено како би апликација била што више једноставнија и лакша за развијање. Чување шаховских загонетки у текстуалној датотеци је такође релативно једноставно и не захтева пуно меморијског простора. Постоје укупно 52 нивоа јачине шаховски слагалица. У распону од 250 до 2800 са кораком од по 50. Сваки ниво има по три шаховске слагалице, од којих се кориснику насумично изабере једна. Пример чувања шаховских слагалица се може видети на листингу кода 9.

```
[Kh6,Rg6,Ne6,Pg4][Kf7,Ra4][W][g4g5,a4h4][https://www.chess.com/puzzles/problem/685944]
```

*Листинг кода 9. Пример чувања шаховских слагалица у текстуалним датотекама*

Постоје пет делова чувања слагалице, сваки од њих се налази у угластим заградама, и одвојени су доњом цртом ради лакшег парсирања. Прва угласта заграда садржи почетне позиције белих фигура. Друга угласта заграда садржи почетне позиције црних фигура. Трећа угласта заграда указује на то којом бојом је одигран први потез (ако први потез игра бели, онда корисник решава слагалицу као црни и обрнуто). Четврта угласта заграда садржи потезе који треба да се одиграју у алгебарској нотацији. Пета угласта заграда садржи извор слагалице ради лакшег тестирања. Пример слагалице са листинга кода 9 се може видети на слици 123.




*Слика 7. Пример шаховске слагалице са листинга кода 9*

Као што се може приметити са слике 7, бели је одиграо први потез, односно, померен је пијун са поља г4 на поље г5. Сада је на корисника ред да одигра потез као црни. У овом конкретном примеру решење слагалице је шах-мат у једно потезу, односно, померај топа са поља а4 на поље х4.

# 4. Случајеви употребе апликације

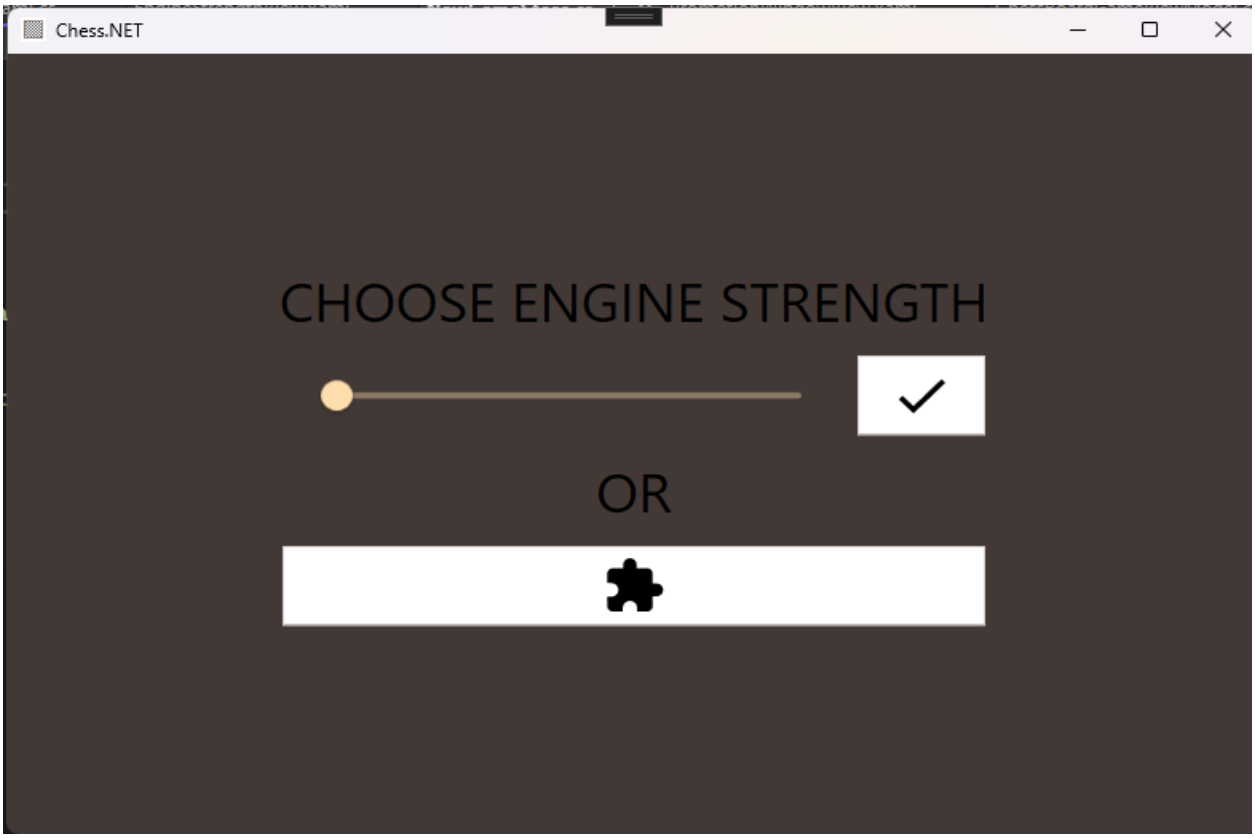
У овом поглављу ће се описати рад програма. Преглед рада апликације ће почети са одабиром боје којом корисник жели да игра.

На слици 8 се може видети почетни прозор за бирање жељене боје.


*Слика 8. Прозор за жељене одабир боје*

Након што корисник изабере боју биће аутоматски преусмерен на прозор за бирање жељене јачине шаховског програма против којег жели да игра.

На слици 9 се може видети прозор за бирање жељене јачине шаховског програма.

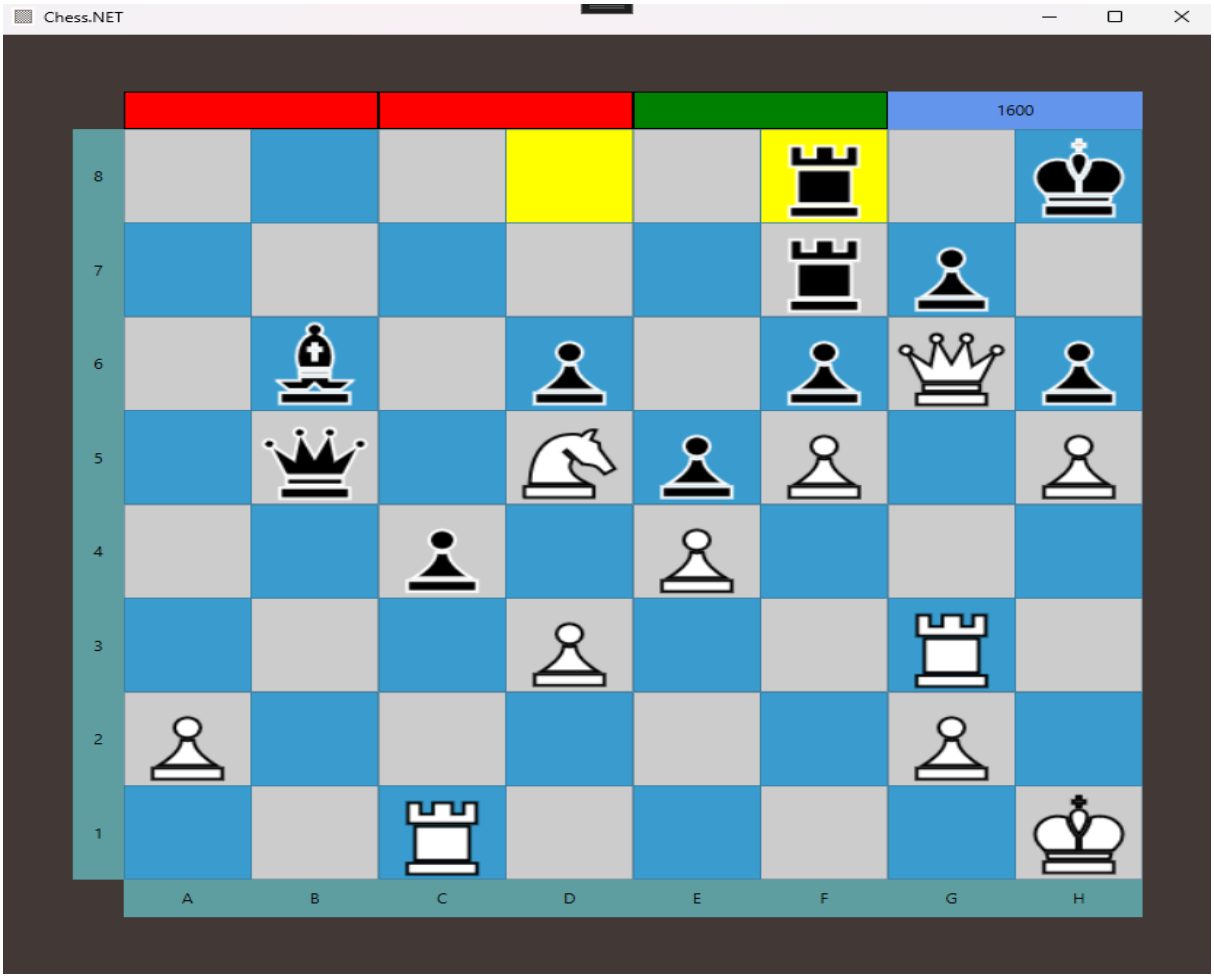



*Слика 9. Прозор за бирање жељене јачине шаховског програма*

На слици се може видети да постоје две могућности. Прва могућност је да корисник ручно изабере јачину шаховског програма против којег жели да игра помоћу клизача, у распону од 250 до 2800. У било ком тренутку корисник може потврдити одабрану јачину шаховског бота кликом дугмета за потврду. Након потврде корисник ће бити преусмерен на прозор где ће партија почети. Друга могућност је да корисник кликом на дугме за шаховске слагалице буде преусмерен на прозор за процењивање јачине корисника помоћу шаховских слагалица.

Пошто оба случаја на крају дана воде до крајњег прозора на којем се игра против шаховског програма прво ће се разматрати прозор на којем корисник решава шаховске слагалице.

На слици 10 се може видети прозор за решавање шаховских слагалица.




*Слика 10. Прозор за решавање шаховских слагалица*

На слици *10* се може приметити шаховкса табла са фигурама и координате свих поља. Поред тога, изнад врха табле могу се приметити четири правоугла. Прва три правоугаоника кориснику говоре колико грешака може да направи. Четврти правоугаоник говори кориснику колику јачину шаховског програма је до сада достигао решавајући шаховске слагалице. Конкретно, на примеру са слике 10 се може видети да је корисник дошао до јачине шаховског програма од 1600 и да је начинио две грешке што нам говоре два црвена правоугаоника. Након треће, уједно, и последње грешке прорачун ће бити завршен и корисник ће бити преусмерен на последњи прозор где ће отпочети партија против шаховског програма. Такође, треба напоменути да ће табла увек бити оријентисана тако да одговара боји фигура којом корисник игра и да први потез увек игра супротна боја од корисника.

На слици 11 се може видети прозор за играње против шаховског програма.


*Слика 11. Прозор за играње против шаховског програма*

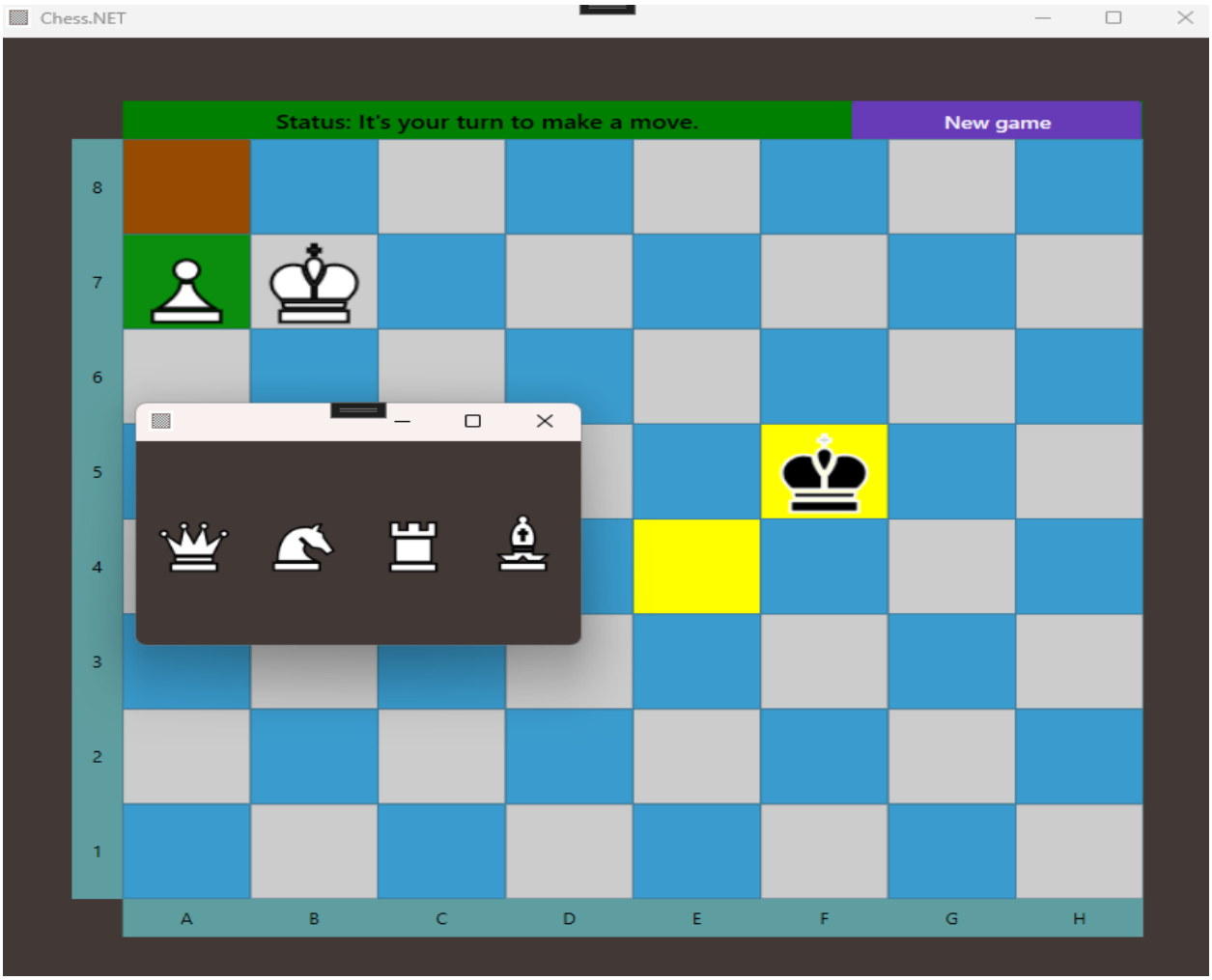
На слици 11 се може приметити, поред фигура и координата, дугме *New Game* које омогућава кориснику да цео процес крене испочетка, и лабела која обавештава корисника о статусу партије. У зависности од ситуације лабела може мењати боју и поруку којом обавештава корисника. Са конретног примера на слици 10 може приметит видети да је корисник померио скакача на поље e4 и тренутно чека одговор шаховског програма.

На слици 12 ће бити објашњен начин померања шаховских фигура.


*Слика 12. Пример померања шаховских фигура*

Када корисник кликне на фигуру, фигурино поље биће зелене боје, док ће сви легални потези које корисник може да одигра бити браон боје. Другим кликом на произвољно браон поље корисник ће извршити потез и фигура ће се померити. Са конкретног прима на слици 12 може се приметити да је шаховски програм одиграо последњи потез, односно померен је топа са поља а3 на поље а8 и кориснику је задат шах. Корисник се одлучио да свој следећи потез одигра тамним ловцем на пољу д6, међутим, једини легални потез у конкретој позицији је да корисник блокира шах  померањем ловца на поље ф8 и зато је једино то поље обојено браон бојом.

На слици 13 се може видети прозор за избор промоције фигуре.




*Слика 13. Пример прозора за промоцију фигура*

Када корисник жели да промовише пијуна у фигуру по избору биће му предтављен дијалог са иконицама фигура у које пијун може да се промовише. Након клика на жељену фигуру извршиће се промоција у изабрану фигуру и потез ће бити окончан.

# 5. Закључак

Циљ овог пројекта је да се развије апликација која ће бити доступна и прилагођена свим љубитељима шаха, без разлике у њиховим шаховским вештинама или нивоу компјутерске писмености.

Када разматрамо ефикасност овог решења, потребно је напоменути да, иако ради прилично добро, оно није најбоље и има пуно могућности за побољшање. Један од изазова са којим се апликација сусреће је зависност јачине шаховског програма од процесора на корисничком рачунару. Ова зависност може резултовати различитим искуствима корисника у зависности од хардвера који користе.

Међутим, начин како би се овај проблем решио је да се апликација подели на клијентски и серверски део. Клијентски део би користили корисници и он би садржао све што садашња апликација садржи сем шаховског програма, док би серверски део чинио шаховски програм који би био покренут на моћнијем рачунару да би свим корисницима пружио једнако искуство. Серверски део би примао шаховске позиције од корисника и враћао најбоље потезе.

Потенцијално унапређење апликације би било чување претходних партија корисника, омогућавајући им да их прегледају. Додатно, корисници би могли користити шаховски програм за анализу својих претходних партија и откривање грешака, што би им помогло да унапреде своје шаховско знање.

Шах.НЕТ апликација представља одличну платформу за вежбање и учење. Играње против *Stockfish*-а је један од најбољих начина да се побољшају шаховске вештине, јер је то веома јак шаховски програм који ће идентификовати и казнити корисникове грешке. Корисник може прилагодити јачину шаховског програма својим тренутним шаховским вештинама и прогресивно подизати јачину шаховског програма како буде напредовао у шаху. Поред тога, шаховске слагалице могу помоћи кориснику да развије своје тактичке вештине и вештине решавања проблема. Такође, апликација би се могла користити у едукативне сврхе нових играча шаха и деце, јер форсира корисника да игра поштујући шаховска правила и јасно приказује све легалне потезе.

# 6. Литература

[1] The New York Times, „How Popular Is Chess”, https://www.nytimes.com/2022/06/17/crosswords/chess/chess-is-booming.html, Нагли скок популарности компјутерског шаха

[2] Chess.com, „Chess Is Booming! And Our Servers Are Struggling”, https://www.chess.com/blog/CHESScom/chess-is-booming-and-our-servers-are-struggling, Нагли скок активних корисника *Chess.com* ваб сајта и нестабилност *Chess.com* сервера

[3] Chess.com, „Members”, https://www.chess.com/members, Укупан број корисника *Chess.com* веб сајта

[4] Microsoft, „Model-View-ViewModel”, https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm#the-mvvm-pattern, *Model-View-ViewModel* софтверски образац

[5] Refactoring.Guru, „Mediator”, https://refactoring.guru/design-patterns/mediator, *Mediator* софтверски образац

[6] Stackify, „Dependency Injection”, https://stackify.com/dependency-injection/, *Dependency Injection* софтверски образац

[7] Chess.com, „Chess Engines”, https://www.chess.com/terms/chess-engine, Шаховски програми

[8] Chessentials.com, „History Of Chess Computer Engines“, https://chessentials.com/history-of-chess-computer-engines/, Историја шаховских програма

[9] History.com, “In 1950 Alan Turing Created a Chess Computer Program That Prefigured A.I“, https://www.history.com/news/in-1950-alan-turing-created-a-chess-computer-program-that-prefigured-a-i, Развитак првог шаховског програма од стране Алана Туринга

[10] Chess.com, „The Evolution of Chess Engines“, https://www.chess.com/blog/TheCheeseDuck/the-evolution-of-chess-engines, Еволуција шаховских програма

[11] Chess.com, „Top Chess Player”, https://www.chess.com/players/magnus-carlsen, Најбољи шахиста данашњице

[12] Chess.com, „Stockfish Chess Engine”, https://www.chess.com/terms/stockfish-chess-engine, *Stockfish* шаховски програм

[13] ComputerChess.org, „CCRL40/15 Rating List-All Engines”, https://www.computerchess.org.uk/ccrl/4040/, Листа најбољих шаховских програма септембра 2023. године

[14] Fairy Stockfish, „Fairy Stockfish”, https://fairy-stockfish.github.io/, *Fairy Stockfish* шаховски програм

[15] Microsoft, „.Net (and .NET Core) – Introduction and Overview”, https://learn.microsoft.com/en-us/dotnet/core/introduction, Увод и преглед *.NET* технологија

[16] Microsoft, „What is Windows Presentation Foundation”, https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-7.0, Опис *WPF Framework*-а

[17] Chess.com, „Draw in Chess”, https://www.chess.com/terms/draw-chess#dead-position, Дефиниција нерешене партије шаха

[18] Microsoft, „Asynchronous Programming”, https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios#important-info-and-advice, Асинхроно програмирање у *C#*-у

[19] Shredder Chess, „UCI Protocol”, https://backscattering.de/chess/uci/, Дефиниција *UCI* протокола

# 7. Додатак А

```
public class ColorSelectionStore
{
    public event Action<Color> ColorSelected;

    public void SelectColor(Color color) => ColorSelected?.Invoke(color);
}

public class ColorSelectionViewModel : ViewModelBase
{
    public ICommand WhiteCommand { get; private set; }
    public ICommand BlackCommand { get; private set; }
    private readonly ColorSelectionStore _colorSelectionStore;

    public ColorSelectionViewModel(ColorSelectionStore colorSelectionStore)
    {
       _colorSelectionStore = colorSelectionStore;

       WhiteCommand = new Command((o) =>
                        _colorSelectionStore.SelectColor(Color.White), (o) => true);
       BlackCommand = new Command((o) =>
                       _colorSelectionStore.SelectColor(Color.Black), (o) => true);
    }
}

public MainViewModel(.....)
{
    ...

    _colorSelectionStore = colorSelectionStore;
    _colorSelectionStore.ColorSelected += ColorSelected;

    ...
}


private void ColorSelected(Color color)
{
    _chessBoardGameViewModel.PlayerColor = color;
    CurrentViewModel = _engineStrengthViewModel;
}
```

*Листинг кода 2. Пример коришћења Mediator софтверског обрасца у апликацији*

# 8. Биографија

Моје име је Марко Ђурђевић. Рођен сам 31.08.1999. у Зрењанину, где сам и одрастао. Средње образовање сам завршио у електротехничкој и грађевинској школи „Никола Тесла“ и Зрењанину. Факултет сам уписао 2018 године на Факултету техничких наука у Новом Саду на смеру Примењено софтверско инжењерство.