Gymnázium, Praha 6, Arabská 14
předmět Programování, vyučující Daniel Kahoun,

Šachové rozhraní

ročníkový projekt

Vladimír Vávra, 1.E

květen 2019

Rád bych na tomto místě poděkoval panu Ing. Danielu Kahounovi, který byl vedoucím mého ročníkového projektu. Děkuji mu za nasměrování při implementaci klient-server komunikace a především za to, že mě na toto téma nasměroval.

Anotace

Cílem tohoto ročníkového projektu bylo vytvořit funkční, esteticky příjemné uživatelské rozhraní pro hraní světoznámé hry, šachů, které implementuje všechna pravidla, jež této hře náleží. Program podporuje hru mezi dvěma hráči jak na jednom, tak i na více počítačích současně, ale také lze hrát proti šachovému enginu Stockfish podle uživatelem zadané obtížnosti. Je možné hrát i upravenou verzi této hry zvanou Fischerovy šachy.

Abstract

The objective of this year's project was to create a functional, aesthetically pleasing user interface for playing the world-famous chess game, which implements all the rules that belong to this game. This program supports the game between two players on one or more computers simultaneously, but it is also possible to play against the Stockfish chess engine according to the difficulty that was specified by user. It is also possible to play a modified version of this game called Fischer's Chess.

Die Annotation

Das Ziel dieses jährigen Projekts war, eine funktionale, ästhetisch ansprechende Benutzeroberfläche für das weltberühmte Schachspiel zu machen. Dieses Programm unterstützt das Spiel zwischen zwei Spielern auf einem oder mehreren Computern. Es ist jedoch auch möglich, gegen die Stockfish-Schach-Engine zu spielen, je nach den vom Benutzer angegebenen Schwierigkeitsgraden. Es ist auch möglich, eine modifizierte Version dieses Spiels zu spielen, die Fischer's Chess heißt.

Důležité pojmy

- **Parsovat** – analyzovat vstupní data pomocí předem daných pravidel a postupů, na jejichž konci jsou získána data, se kterými je možno určitým způsobem pracovat
- **Protokol** – pravidla pro komunikaci
- **Algoritmus** – přesný postup, jak vyřešit danou úlohu
- **Šachový engine** – počítačový program, kterému je poslán aktuální stav šachovnice a pomocí protokolu je možné si vyžádat nejlepší možný tah při zadané hloubce vyhledávání
- **UCI protokol** – protokol zajišťující komunikaci mezi grafickým rozhraním a šachovým enginem
- **Server** – označení pro počítač nebo počítačový program, který poskytuje nějaké služby
- **Klient** – počítačový program, který poskytuje uživatelské rozhraní
- **FEN** – způsob zaznamenání šachovnice. Řetězec je rozdělen mezerami na jednotlivé informace o hře. První část je rozdělena „/“ na jednotlivé řádky. Písmena značí symbol pro figuru na daném políčku, malá znamenají černého, velká bílého hráče. Čísla označují počet volných políček do další figury, následuje w, nebo b (w = bílý na tahu, b = černý na tahu). Třetí část je možnost rošády (K = rošáda na králově straně, Q = na straně královny, malými písmeny pro černého hráče), další část je buď „-“, nebo pole pro braní mimochodem. Poslední 2 čísla jsou počet tahů od posledního sebrání figury nebo táhnutí pěšce a poslední je číslo tahu (rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1)
- **Hodnota figury**: Číslo udávající důležitost figury dle FIDE pravidel**1** = pěšec, **3** = Jezdec, Střelec, **5** = Věž, **9** = Dáma, **90** = Král
- **matchmaking**: spárování nezávislých klientů tak, aby spolu mohli hrát.

Zadání práce

Povinná část:

Vytvořit rozhraní, které pomocí grafického rozhraní umožňuje hrát hru šachy.

Bonusy:

- Vytvořit klient-server komunikaci a hrát tedy na více počítačích současně
- Vytvořit vlastní šachový engine

Obsah

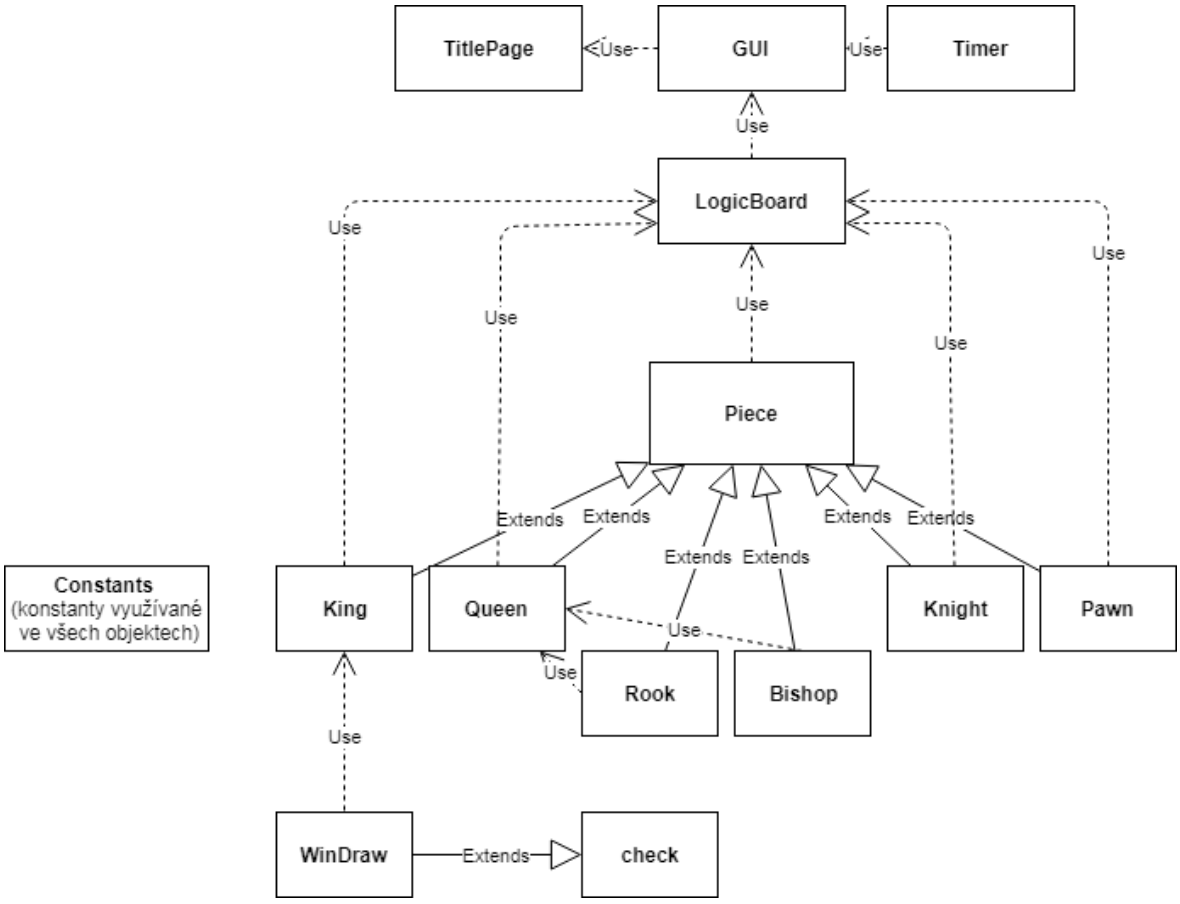
1. Úvod

Jak již je v anotaci popsáno, cílem je vytvořit grafické rozhraní. Práce popisuje každou třídu kromě té s konstantami a podrobně je rozebírá. Je strukturována od svrchních částích obrázku 1 až po ty spodní a ty, které v něm nejsou zakresleny.

Toto téma jsem si vybral proto, že šachy jsou má nejoblíbenější hra a chtěl jsem si tedy vyzkoušet ji naprogramovat. Měl jsem již předtím zvolené téma implementovat sudoku a algoritmus na vyřešení, ale toto téma mi přišlo jako větší výzva.

2. Architektura programu

2.1. Obecná architektura tříd programu



Obrázek 1 – Architektura tříd, které jsou využity pro každý typ protihráče

V tomto diagramu nejsou zahrnuty třídy spjaté se speciálními případy, jako je hra s šachovým enginem nebo hra na více počítačích současně (podrobněji probráno v dalších částech tomu věnovaným). „Use“ značí, že třída, ke které je šipka namířena vlastní objekt třídy, ze které vychází.

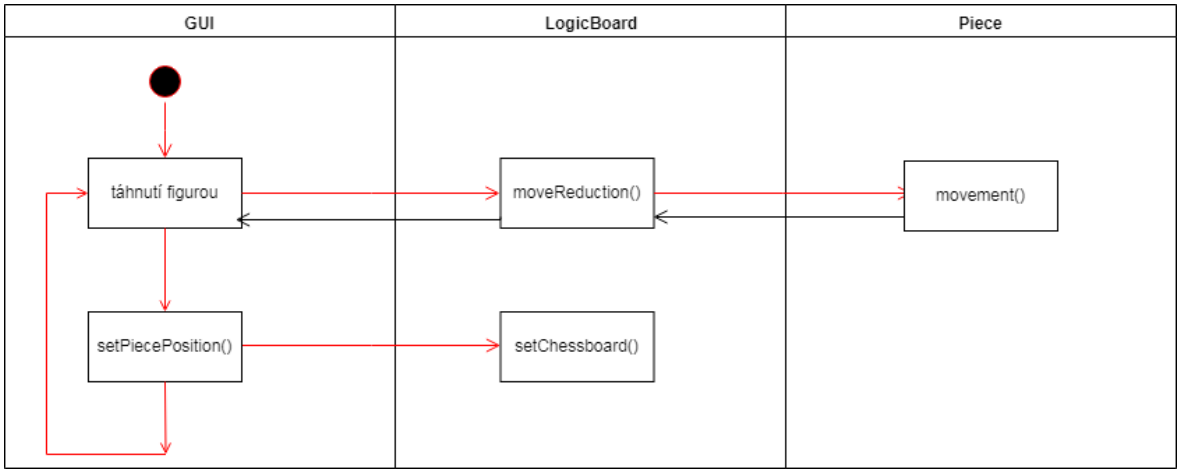
2.2. Obecná architektura běhu programu

Při spuštění se objeví titulní strana, kde si hráč nastaví vše, co se týká hry, jež bude chtít hrát. Po nastavení a odstartování se zobrazí grafické rozhraní.

Program je rozdělen do dvou vrstev. První z nich je grafická (třída GUI). Ta komunikuje přímo s uživatelem, tedy ukazuje mu aktuální stav šachovnice a další věci podrobně ukázané v jiných částech dokumentace. Druhá vrstva je logická, v ní se provádí všechny výpočty.

Poté, co hráč klikne na tlačítko s figurou, se pomocí metody **moveReduction** (redukuje počet všech tahů, které může figura aktuálně udělat na ty, kde se král nedostane do šachu v dalším tahu) z třídy LogicBoard zobrazí možné tahy, které může figura provést. Jakmile hráč táhne, je zavolána metoda **setPiecePosition**, jež provede přípravné práce jako změna časovače, či zobrazení pohybu.

Následně se zavolá metoda setChessboard, která uloží aktuální tah do pole typu Figurka (třída Piece), zneguje proměnnou reprezentující aktuálně hrajícího hráče a další přídavné práce. Následně se celý cyklus opakuje, dokud nenastane konec hry.



Obrázek 2 – architektura tahového mechanismu

2.3. TitlePage

Třída TitlePage je to první, co hráč při spuštění programu uvidí. Vpravo se nachází 4 výběrová pole. První z nich vybírá protivníka, možnosti jsou engine (singleplayer), hráč na stejném počítači (2P/1C), nebo druhý hráč na jiném počítači (2P/2C). Na základě první možnosti se vygenerují možnosti pro další 2 výběrová pole, které jsou rozdílná pro každou možnost. Poslední pole vybírá čas na hodinách.

Obrázek 3 – titulní strana

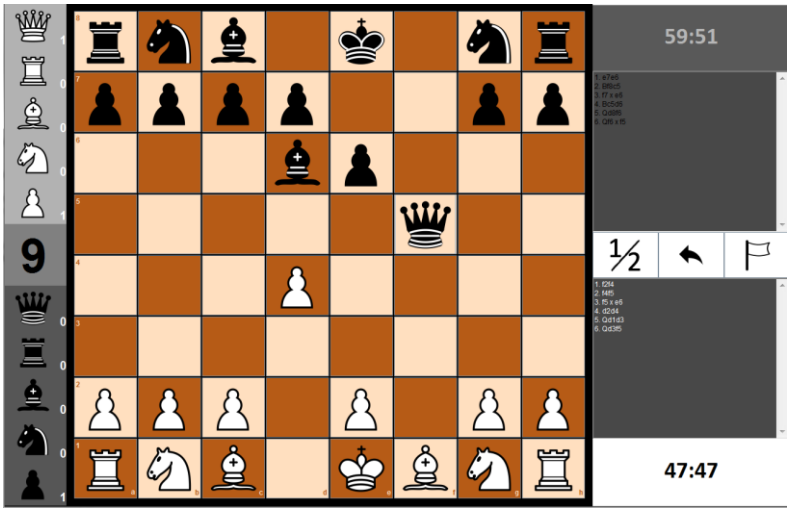
Vlevo se poté nachází 2 tlačítka černé a bílé barvy. Pokud jsou správně vybrána všechna pole vpravo, otevře se po kliknutí v novém okně šachové rozhraní patřící hráči stejné barvy jako je barva zmačknutého tlačítka. Třída dědí z JPanelu a je responzivní, tedy dokáže zareagovat na změnu velikosti tak, že všechen obsah ve stejném poměru zvětší na maximální možnou velikost, aby nic nepřesahovalo JFrame a zarovná na střed jak vertikálně tak horizontálně. Místa, kde nic není, jsou poté vyplněná šedou barvou. Při změně velikosti je použit null layout. Pomocí componentListeneru na JFramu se dokáže program pomocí metody setBounds perfektně adaptovat na novou velikost. Při běhu programu je tedy poměr původní velikosti k původní velikosti JPanelu vynásoben aktuální velikostí JPanelu a dostáváme požadovanou velikost.

Udržování textu na středu výběrového pole zkopírováno (aterai, 2012)

2.4. Třída GUI

Na této části spočívá ta nejdůležitější část programu, tedy samotné šachové grafické rozhraní. Na rozdíl od výše popsané třídy GUI dědí z JFramu, nikoliv JPanelu. Je to proto, že se rozhraní otvírá v novém okně. Důvodem je, že po ukončení jedné hry je poté možné zahájit neprodleně hru další, aniž bychom museli znovu zapínat program. DefaultCloseOperation je proto nastaven na DISPOSE_ON_CLOSE, aby se při zavření nezavřel celý program. Při vytváření objektu přejímá konstruktor od TitlePage parametry: pro zadání času na hodiny, boolean whiteGUI (je-li true, poté toto rozhraní patří bílému hráči), typ protivníka a informace týkající se specifických oponentů probrané dále v dokumentaci. Toto rozhraní je opět responzivní ve stejném stylu jako třída TitlePage. Celý obsah se nachází v jednom hlavním Containeru, ten má nastavený GridBagLayout v poměru šířky 1 (vyhozené figury) :8 (šachovnice) :3. (komponenty vpravo).

Šachovnice je tvořena 8 x 8 polem JButtonů, které jsou umístěny do JPanelu s GridLayoutem. Pokud je souřadnice políčka dělitelná 2, je jeho barva bílá, jinak je hnědá. Figurky jsou reprezentovány jako ikony na tlačítkách. Tyto ikony (Antonsusi, 2012) se získávají ze tříd, které dané figury reprezentují. Pokud klikne uživatel na tlačítko s figurou, zobrazí se všechny možnosti, kam může figura táhnout. Pro provedení tahu musí být vybraná figura přesunuta na platné pole (jedno z těch, které vám rozhraní nabídne). Pokud se král jednoho z hráčů dostane do šachu, což zkontroluje třída Check, zobrazí se královo pole rudou barvou. V případě, že pěšec došel až na konec šachovnice, nastaví se field allowedMove na false a nepůjde udělat žádný tah, dokud si hráč nevybere figuru, která pěšáka nahradí. Seznam těchto figur se zobrazuje místo JPanelu s pohyby.



Obrázek 4 – šachové rozhraní

Na levém sloupci a dolním řádku můžeme vidět čísla řádků a písmena sloupců. Ty jsou reprezentovány jako JLabely přidané na tlačítko, kterému se předtím nastaví GridBagLayout. Ve skutečnosti jsou tyto JLabely i v pravém sloupci a horním řádku, ale jsou nastavené na setVisible(false). Zobrazí se právě tehdy, když se otáčí šachovnice (pokud jsou 2 hráči na jednom počítači, tak se po tazích šachovnice otáčí ve prospěch toho, kdo právě hraje).

Pokud je rozhraní určeno pro černého hráče, je nutné šachovnici otočit. To se provede tak, že nejprve program projde cyklem všechna tlačítka, smaže je z JPanelu a poté je vloží zpět v přesně opačném pořadí, v jakém byly do šachovnice prvně přidány. Na tomto principu funguje otáčení šachovnice zmíněné výše. Vlevo můžeme vidět panel vyhozených figur. Ty jsou opět reprezentovány JLabelem, stejně tak jako číslo pod nimi. Když je zavolána metoda setPiecePosition, tak je z ní zároveň zavolána metoda, která přičte k hodnotě na JLabelu jedničku, pokud byla právě tato figurka vyhozena. Uprostřed je poté hodnota, o kolik vede vzhledem k aktuálnímu materiálu na šachovnici hráč, v jehož barvě je číslo zbarvené. Toto číslo se vypočítá pomocí rozdílu hodnot materiálu vyhozených figur, o který se stará třída LogicBoard.

Napravo leží časovače (viz dále), 3 tlačítka uprostřed a dvakrát TextArea v JPanelu s pohyby obou hráčů. Po každém tahu je opět z metody setPiecePosition zavolána jiná část kódu, která volá metoda setLastMove ve třídě LogicBoard, jež zajistí zapsání posledního tahu do dlouhé šachové algebraické notace. V ní je zapsáno číslo tahu, symbol figury souřadnicemi určené počáteční a cílové políčko, „x“, bere-li při tahu figuru, „+“, dává-li při tahu šach. Do textového pole se zapisuje tak, že se nejprve

vezme jeho aktuální obsah, přidá znak nového řádku a pohyb. Tento prostor může být po určitý čas nahrazen jiným JPanelem, např. promocí pěšáka v dámu, oznámením o výhře/prohře/remíze nebo zamítnutím a přijmutím remízy.

První ze tří tlačítek je návrh remízy. Ten můžete odeslat pouze proti lidskému protihráči. Pokud se tlačítko stiskne, spustí se jeho actionListener, který nastaví field drawOffered na true. V dalším tahu se na místě výše zmíněného textového pole objeví červené a zelené tlačítko, jimiž můžete odmítnout, či potvrdit remízu.

Druhé tlačítko s šipkou vrací tahy. Funguje pouze pro hru s počítačem (nebylo by fair vracet tahy proti druhému hráči). Funguje na principu, že si od třídy LogicBoard vyžádá poslední FEN a ten je zpracován metodou newGame(), ta smaže v cyklu všechny figury a na jejich místo dosadí jiné zapsané ve FEN řetězci. Opraví také počet vyhozených figur a vymaže tahy, které jsou vráceny. Poslední tlačítko (seekpng, nedatováno) je tlačítko vzdání se.

2.5. Timer

Tato třída dědí z Thread. Reprezentuje šachové hodiny s nastavitelným časem od 1 sekundy do 24 hodin. Konstruktor jako argument přebírá počet hodin, minut a sekund do 0, referenci na grafické rozhraní kvůli případu, kdy dojde čas a je nutné zavolat metodu, která zobrazí konec hry. Nakonec přebírá i počet sekund, o kolik každým tahem zvýší počet sekund daného hráče.

Jelikož by bylo složité počítat rovnou s hodinovým časem, převede si ho konstruktor na sekundy. Tento čas je poté v metodě run každou 1 sekundu díky Thread.sleep(1000) snížen o 1. Vždy, když se takto sníží hodnota, se ale musí čas opět převést na ten hodinový.

Samotný časovač, který může uživatel vidět je obyčejný JPanel s JLabelem, ve kterém je uložena textová reprezentace časových hodin.

Časovač se přepíná z metody setPiecePosition zavoláním metody playerChange(), která také změní barvu aktuálně spuštěného časovače.

2.6. LogicBoard

Zatímco předchozí části se staraly pouze o grafický výstup, od této chvíle už bude práce zaměřená pouze na 2. vrstvu programu, tedy tu, ve které probíhají všechny výpočty. Třída LogicBoard si udržuje přehled o tom, kde se aktuálně nachází která figura. To dokáže díky třídnímu poli typu Piece[][] velikosti 8 x 8. Dále třída rozhoduje o tom, kdo je aktuálně na tahu pomocí fieldu typu boolean „whiteMoves“.

Při vytváření objektu konstruktor přebírá řetězec, který říká, jaký typ rozestavení šachovnice se má vytvořit („Fischer“/ „Normal“). V konstruktoru se volá metoda, která vloží do šachovnice objekty figur, tyto pozice jsou již z většiny předdefinované. Do pole se však nevkládá nový objekt na každou figuru na šachovnici, ale pouze jeden objekt na figuru daného typu dané barvy. Tyto objekty jsou uloženy ve svých šesti-prvkových polích, odkud je možné kdykoliv vložit na šachovnici novou figurku tohoto typu.

Nachází se zde podobná metoda jako je setPiecePosition v třídě GUI, tj. setChessboard(). Ta nastaví na základě přejatých argumentů polohu figury na třídní šachovnici, změní hráče, který je na tahu, aktualizuje stav materiálu a zavolá metodu zjišťující, zda není po novém tahu nepřátelský král v šachu. Problém nastává při implementaci speciálních pohybů, jako je rošáda promoce pěšáka nebo braní mimochodem.

Změna pozice funguje tak, že nejprve se na souřadnice, kam se figurka pohnula, vloží objekt té figury a poté se reference na původních souřadnicích nastaví na null.

Pokud tedy bude program řešit braní mimochodem, nejdříve nastaví pozici, kam se pěšec posunul a následně nastaví pozici znovu, tentokrát ale předá metodě souřadnice, kam se má figurka pohnout stejné, jako ty, ze kterých přišla. Zároveň to budou souřadnice vyhozené figury. Toto figuru smaže.

Toto je možné využít i u promoce pěšáka s tím rozdílem, že program musí metodě poslat ještě jeden argument a to objekt figury, ve kterou se pěšák mění.

Co se týče rošády, na tu je napsána speciální metoda, která na základě předaných argumentů jen změní pozici krále a věže. Více o rošádě v kapitole o králi.

Dále třída obsahuje metodu moveReduction(). Ta uživateli říká, kam může táhnout. Její činnost je již zmíněna v kapitole 2.2, ale byla by škoda si o ní nepovědět víc. Metoda totiž nevolá pouze metodu movement(), která je probraná v části věnované figurám, ale také kontroluje, zda může hráč táhnout, tj. zda by po pohybu nevznikl šach. Metoda vrací ArrayList souřadnic všech políček, na který může daná figura přejít Tento ArrayList nejprve obsahuje všechny prvky, které vrací metoda movement(), a poté zůstanou pouze prvky, které má společné s vrácenými ArrayListy metod isCheck() a canMove() ze třídy Check.

Dále třída zapisuje poslední tah.

Poslední a zároveň velice zajímavou funkcí této třídy je archiv hry. Jde vlastně o HashMapu všech FEN stringů, které byly během hry vygenerovány. O jejich generování se stará třída DataParsing probraná dále. Jak klíč je v mapě FEN a jeho hodnotu tvoří počet opakování tohoto rozestavení během hry. Díky této mapě je možné vracet tahy, jak bylo popsáno předtím, dále určovat remízu kvůli trojitému opakování tahů, ale i kvůli 50 tahů bez tahu s pěšcem, či zabrání figury.

2.7. Piece

Piece je abstraktní třída, ze které dědí všechny třídy reprezentující figurky. Je nutné tuto třídu vytvořit, protože se podle ní vytváří pole v třídě LogicBoard.

Každá třída, která z této dědí, má metodu na vrácení symbolu pro danou figurku a metodu movement, která vrací ArrayList všech možných souřadnic, kam může figurka jít. Také mají metody na hodnotu, opravdovou hodnotu (jestliže se z pěšáka stane dáma a já ji vyhodím, nevyhodil jsem dámu, ale pěšáka) a barvu (isWhite()).

2.7.1. Pawn

Třída reprezentující obyčejného pěšáka. Všechna pole, kam může figurka jít, jsou již přednastavená. Jediný zádrhel tkví ve směru chůze pěšáka. Pokud je pěšák bílý, zapne se test předepsaných možností v if bloku, pokud není, provedou se předepsané testy z test bloku. Movement metoda zároveň přebírá řetězec reprezentující poslední zahraný tah, podle něj se tedy dá určit, zda může pěšák provést braní mimochodem (detail v kódu). Za promoci pěšáka třída neodpovídá.

2.7.2. Knight

Naprosto nejjednodušší figurka na implementaci. Všechny pohyby se totiž musely předepsat, neexistuje způsob, jak je nějakým hezkým cyklem efektivně získat. Vždy, když se testuje, zda je pole vhodné pro přesunutí, musí se otestovat, zda je v rozsahu pole. Nemůže se použít try blok, protože je potřeba provést všechny testy a nezastavit se na chybě.

2.7.3. Rook

Movement() volá 4 metody, každá zodpovídá za jednu část, kam se věž může podle pravidel přesouvat. Metody fungují díky for-cyklům, kdy je řídící proměnná inicializována na řádek/sloupec +/-1 (záleží na směru, kam má věž jít). Poté řídící proměnná reprezentující buď řádek, nebo sloupec zvětšuje svoji hodnotu o +/-1 a jakmile narazí na pole, které není null, končí cyklus Poté otestuje políčko, které nebylo null a zkontroluje, jestli figurku na něm stojící nemůže zabrat. Pokud ano, přidá ho do ArrayListu a následně vrátí ArrayList. Movement pak vrací sjednocení ArrayListů

2.7.4. Bishop

Naprosto stejný princip jako věž, akorát má for-cyklus 2 řídící proměnné.

2.7.5. Queen

Vytváří objekt střelce, věže a kombinuje jejich schopnosti

2.7.6. King

Král je nepochybně nejsložitěji implementovatelná figura, protože při svých tazích nesmí dostat šach nebo může provést rošádu. Třída LogicBoard musí přesně vědět, kde král je, aby mohla dobře předat třídě Check souřadnice, z toho plyne, že král musí mít proměnnou se svými koordináty. K nim LogicBoard bude přistupovat přes gettery. Dále musí mít field na první souřadnice a další field na místo výskytu svých věží, aby mohl správně provést rošádu.

Jelikož program umí hrát Fischerovy šachy, rošáda se značně komplikuje. Definujeme ji tedy následovně: Král, který není v šachu se přes nešachované prázdné (věž, se kterou dělá rošádu mu může stát v cestě) musí dostat na g(řádek krále), nebo c(řádek krále) a věž f(řádek krále), nebo d(řádek krále). Takto definujeme rošádu jak v normálních, tak i ve Fischerových šachách. Rošáda je rozdělena na 2 metody a každá z nich postupně všechna pole, přes která král projde a vyhodnocuje, zda jsou šachovaná, či jsou volná.

2.8. WinDraw a Check

Ačkoliv se jedná o 2 rozdílné třídy, tak WinDraw dědí z Check a prakticky využíváme pouze ji. Ve třídě Check se nachází 2 důležité metody. Jedna z nich, isCheck(), kontroluje, zda je hráč v šachu a vrací všechna pole, přes která může figura ohrožující ho projít, nebo vrací -1, pokud král dostal dvojitý šach. Pokud totiž dvojitý šach nedostane, může hráč představit před krále jinou figuru. Metoda funguje na principu, že se vytvoří objekty všech figur a následně předpokládá, že je král na jejích místě. To znamená, že se zavolají metody movement, v jejichž středu je král a pokud v některé z nich narazí na nepřátelskou figuru, která je může ohrozit, tak se zapíše cesta k té figuře.

Druhá metoda kontroluje, zda můžeme určitou figurou táhnout a vrací nám ArrayList všech možností, kam může figura jít. Jako argumenty přebírá mimo jiné i řádek a sloupec naší figury. Pokud tedy nejde o krále, tak ze šachovnice figuru metoda vyjme a poté na ní zavolá metodu isCheck(). Ta uživateli vrátí ArrayList všech souřadnic, přes které by uživatelova krále nějaká figura ohrožovala, kdyby tam ta původní figura nebyla. Pokud vrátí prázdný ArrayList, LogicBoard to vyhodnotí jako, že může táhnout všemi směry.

WinDraw poté obsahuje metody, které identifikují aktuální stav hry – výhra, remíza, neukončené a kontroluje, jakým způsobem tato situace nastala. Např. test, zda jsou možné tahy, se provádí procházením celé šachovnice a voláním metody moveReduction(). Pokud je šach a nejsou možné tahy, je to mat, jinak je to pat. Remíza nedostatkem materiálu se opět provádí pouze čistým procházením šachovnice.

3. Volba protihráče

Doposud dokumentace popisovala části programu, které jsou využity při každém typu hry. Je ovšem rozdíl mezi tím, jak bude implementována hra s počítačem oproti hře s člověkem. Typ protihráče se volí v titulní straně programu.

3.1. DataParsing

Statická třída, která má pouze pomáhat ostatním objektům analyzovat příchozí tahy uložené v textové podobě a graficky je zpracovat nebo naopak. Obsahuje čtyři metody, dvě se týkají FEN řetězců a druhé dvě pohybům uložených v dlouhé šachové notaci.

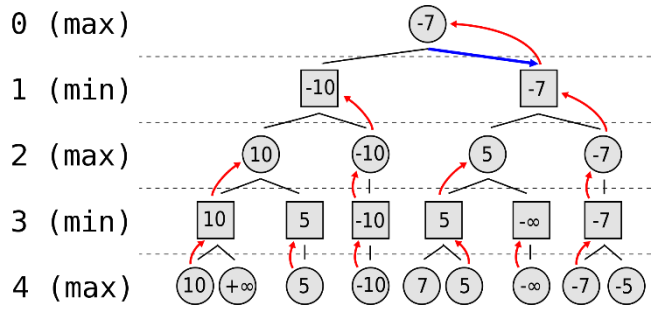
Popis metod:

1. void **inFEN** (String FEN, LogicBoard lb) – metoda přebírá reprezentaci pomocí FEN řetězce a poté upraví data ve třídách LogicBoard a King data tak, aby odpovídala FENU. Nejprve se řetězec pomocí metody split rozdělí. Poté se projde cyklem první část FENU a vytvoří se tak nové pole typu Piece. Když cyklus narazí na krále, uloží se jeho souřadnice do jeho fieldů, to samé pokud se narazí na věž (aby fungovala rošáda). Jestliže není místo pro braní mimochodem v řetězci prázdné, uloží se poslední pohyb tak, aby bylo po tomto tahu možné provést braní mimochodem. Nakonec se uloží zbytek proměnných. Metoda je volána pouze v jednom případě a to když vytvářím novou šachovnici při stisknutí tlačítka zpět.
2. String **outFEN**(LogicBoard board) – Ze zadaného stavu na šachovnici vytvoří FEN řetězec. Funguje na prostém procházení šachovnice a přistupování k fieldům. Volá se, pokud chce uživatel odeslat enginu aktuální stav šachovnice
3. void **inMove**(String move, Piece[][] chessboard, GUI g) – Zobrazí příchozí tah na obrazovku tak, že nejprve naparsuje pohyb a poté zavolá metodu setPiecePosition(„souřadnice“). Volána při komunikaci klient-server
4. String **outMove**(„vše, co přebírá metoda setLastMove v logicBoard“) – Vytváří řetězcovou reprezentaci tahů stejným způsobem jako výše zmíněná metoda setLastMove, akorát nepřidává písmena reprezentující figury atd…

3.2. Šachový engine

V program je pro tento účel využíván Stockfish (Tord Romstad, 2008), což je jeden z nejlepších současných šachových enginů. Při komunikaci s ním je využito UCI protokolu. Každý příkaz v tomto protokolu končí znakem nového řádku. Engine má celkem 20 úrovní síly a od levelu 6 je téměř nemožné ho porazit.

3.2.1. Popis fungování



Obrázek 5 – Minimax algoritmus

Engine na rozdíl od tohoto programu reprezentuje šachovnici pomocí tzv. bitboards. To jsou vlastně reprezentace typu boolean, např. je tam pěšec, není tam pěšec…

Vyhodnocování nejlepšího tahu funguje díky algoritmu minimax. Už z názvu můžeme vidět slova min a max, na čem je algoritmus postaven. Nejprve se na nejnižší vrstvě stromu určí hodnota všech možností šachovnice z aktuálního tahu do určité hloubky, tj. kladná, vyhrává-li bílý a záporná pokud černý. Číslo je tím vyšší, čím vyšší je hodnota, o kterou hráč vede. Poté se do vyššího uzlu vybere buď nejnižší, nebo nejvyšší hodnota ze všech jeho synů. Následně se proces opakuje pro opačný

extrém (táhne druhý hráč a chce co nejvýhodnější pozici), dokud se nedojde k nejlepší hodnotě pro právě hrajícího hráče. Engine samozřejmě tento proces značně optimalizuje. (Flanagancz, 2007)

3.2.2. Implementace komunikace

Důležité příkazy UCI (Dijksman, 2004) protokolu:

- engine –> GUI: **bestmove** „move“ – engine právě našel nejlepší možný tah
- GUI –> engine: **moves** FEN/(startpos moves) „…“ – buď posílám FEN, nebo všechny pohyby od začátku hry
- **go** (depth, nodes, infinite) – aktivuje vyhledávání nejlepšího tahu v zadané hloubce
- **setoption** name Skill Level value „…“ – nastaví sílu enginu (nachází se jen u Stockfish)

Pro hru s šachovým enginem je potřeba zadat do výběrového pole na titulní straně programu Singleplayer. Následně zadá uživatel obtížnost a čas na hru. Nevýhodou hry s počítačem je, že si nemůžete zvolit postavení Fischerovy šachy. Je to kvůli rošádě, kterou tento engine neumí v tomto módu zahrát. Jelikož totiž engine nezavádí pro rošádu zvláštní znak, mohlo by dojít k tahům, které by mohly být považovány za rošádu a zároveň i normální pohyb. Řešením by bylo, aby engine posílal FEN po každém svém pohybu.

Bohužel takto engine nefunguje, nezaznamenává totiž tahy. Při každém táhnutí figurou se totiž musí nejprve odeslat síla enginu, protože si ji nepamatuje, poté je nutné odeslat FEN (jako to dělá tento program), nebo poslat všechny tahy v dlouhé notaci bez písmen reprezentující figurky a úplně nakonec příkaz go. Tento program hledá do stejné hloubky jako je stanovená obtížnost. Zajímavé bylo, že když se nezadal příkaz na stanovení obtížnosti a prohledávalo do hloubky 1, ani tak ho nikdo z testerů nedokázal porazit. Další problém byl zprostředkovat spojení mezi enginem a GUI, o vytvoření procesu se stará následující kód a o přenos dat BufferedReader a BufferedWriter.

```
Stockfish = Runtime.getRuntime().exec(„stockfish-10-win\\Windows\\stockfish_10_x32");
(spojení enginu a popisovaného programu)
```

Jelikož operace, mezi odesláním hráčova pohybu a následnou odezvou enginu může trvat i 6 sekund a swing nedokáže vykreslit správně ikony, dokud neproběhne, je na odesílání a přijímaní informací od enginu vytvořeno nové vlákno. Signalizace, že se má zavolat metoda pro komunikaci je implementována nešikovně (nedostatečná znalost vláken) změnou boolean proměnné na true. Ta se každých 200 milisekund ve vláknu s enginem testuje.

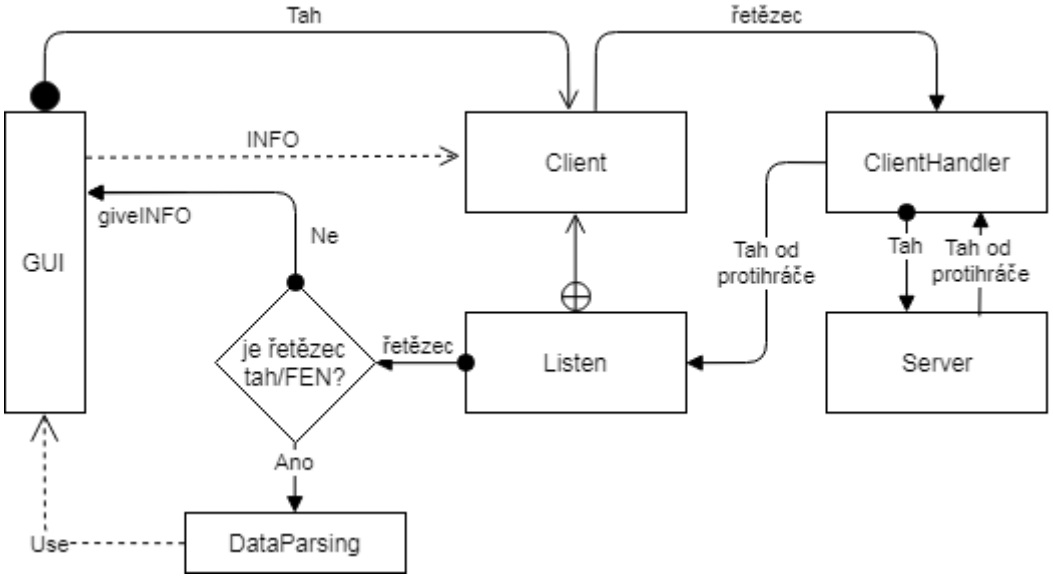
3.3. Hráč na stejném počítači

Tato možnost je realizována pomocí otáčení šachovnice. Když odehraje hráč, zavolá se metoda rotateBoard(). Ta funguje na principu popsaném v kapitole 2.4. K tomu ještě zviditelní JLabely reprezentující sloupce a řádky, které v tu danou chvíli nejsou viditelné a skryje ty aktuální. To je kvůli tomu, aby vždy byla písmena sloupců v dolním řádku a čísla řádků v prvním sloupci. I když je graficky šachovnice obrácená, v logické vrstvě se nic nezměnilo a vše je jako dřív.

Navrhnutí remízy probíhá tak, že po zmáčknutí tlačítka se boolean proměnná reprezentující navrhnutou na true. Každý tah se tato proměnná kontroluje, a pokud je nastavená na true, zobrazí se, že druhý hráč navrhl remízu a možnosti, co s tím udělat.

3.4. Hráč na jiném počítači v lokální síti

Pro volbu tohoto módu uživatel klikne na titulní straně na možnost 2P/2C. Povoleny jsou jak Fischerovy, tak i normální šachy. Komunikace totiž není omezena možnostmi šachového enginu. Stejně jako u něj ale všechny posílané řetězce končí novým řádkem. Jelikož při tazích dochází ke zpoždění, které se po několika tazích okamžitě projeví, je s tahem posílán i aktuální čas na hodinách. Ten je od tahu oddělen „synchronize“. Po přijetí tahu se tedy řetězec rozdělí podle „synchronize“ a na hodiny hráče se zapíše čas ze zprávy. U rošády by zpoždění dokonce bylo 3 sekundy.



Obrázek 6 – Architektura klient-server

3.4.1. Client

Tato třída reprezentuje klienta, ke kterému je přiřazeno právě jedno grafické rozhraní. Nachází se v ní vlákno (Listen), které kontroluje příchozí zprávy od serveru pomocí while cyklu, jenž se provádí každou sekundu 5x, dokud není vlákno vyrušen. O příchozí zprávy se stará BufferedReader a o odesílání serveru zase BufferedWriter.

Seznam možných příchozích příkazů:

- **switchTimer:** Změní časovač hráče. Volá se v případě, kdy se posílá FEN (rošáda)
- **start:** Zobrazí grafické rozhraní a zapne časovač, tedy synchronizuje hru
- **giveFEN:** Okamžitě serveru pošle FEN své šachovnice, použito při rošádě a synchronizování Fischerových šachů (pokud chtějí hráči hrát tento typ a nedošlo by k synchronizaci, byla by rozestavení figurek u obou hráčů úplně odlišná. Rozestavení je totiž čistě náhodné)
- **white:** Okamžitě serveru pošle boolean, zda je aktuální rozhraní pro bílého hráče (volá se jen při matchmakingu)
- **gameType:** posílá buď Normal/Fischer(volá se jen při matchmakingu)
- **time:** posílá čas, který si hráč vybral (volá se jen při matchmakingu)
- **drawOffer:** Zobrazí do GUI, že protihráč nabídl remízu
- **surrender:** Zobrazí do GUI, že se protihráč vzdal
- **acceptedDraw:** Zobrazí do GUI, že protihráč přijal remízu
- **obsahuje-li řezězec „/“:** Jedná se o FEN, je tedy zavolána metoda newChessboard v GUI
- **v jiných případech** se jedná o pohyb v dlouhé šachové notaci a je zavolána metoda DataParsing.inMove(move, chessboard)

Pro komunikaci od GUI směrem k serveru se každý tah v metodě setPiecePosition zavolá zároveň i metoda sendToClientHandler, která pohyb pošle serveru.

V případě, že nastane chyba během odesílání zprávy (server nebo druhý klient terminoval), okamžitě se zavře okno s šachovým rozhraním (metoda end), proudy dat a clientSocket.

3.4.2. ClientHandler

Inspirováno (Mahrsee, 2017)

Každý klient se musí nějak se serverem komunikovat. Jelikož není tento projekt žádná extrémně velká aplikace, můžeme si dovolit přiřadit ke každému klientovi jedno vlákno na straně serveru, které mu bude naslouchat. To je reprezentováno třídou ClientHandler. Pokud klient odešle zprávu, bude to právě tato třída, která ji obdrží a která odešle odpověď.

Třída uchovává informace o typu hry, což jsou všechny popsané ve třídě Client, u kterých je napsáno: volá se jen při matchmakingu (probrán ve třídě server), socket a referenci na server.

V případě, že nastala chyba popsaná v třídě Client, se zavírají proudy dat a socket.

3.4.3. Server

Server se vytváří v případě, že do titulní strany se do 3. výběrového pole odshora zadá pouze port a nic víc. Běží na jednom počítači, kde se zároveň nachází i klient a počet klientů, kteří se na něj mohou připojit je omezen pouze jeho hardwarem.

Třída si udržuje dvě důležité kolekce. Jednou z nich je HashMapa<ClientHandler, ClientHandler>, ta udržuje na pozici klíče vlákno bílého hráče a na hodnotě toho černého. Druhou z nich je ArrayList<ClientHandler> blackHandlers. Ten je plněn ClientHandlery, které nemají přiřazeného partnera. Bílí hráči, kteří ho také nemají, jsou rovnou psáni do HashMapy s hodnotou null.

Po žádosti o připojení od klienta se vytvoří nový ClientHandler, kterému je předán socket a reference na server. Poté je zavolána metoda matchmaking(ClientHandler). Ta přiřadí předanému argumentu vhodného partnera. Ten musí mít stejná nastavení jako předaný ClientHandler.

Na začátku tedy parametru pošle přes BufferedWriter dotazy na barvu figurek, vybraný čas a typ šachovnice. Klient odpoví a data se uloží do proměnných v ClientHandleru. Poté pokud jsou hráčovy figury bílé, se projede cyklem ArrayList blackHandlers a uloží do HashMapy argument a hledaný ClientHandler (pokud takový není, je vložen null). Pokud patří předaný parametr černému, projde se HashMapa a pokud je nalezena vhodný partner, je do HashMapy přidán jako hodnota. V opačném případě se vloží do blackHandlers. Po úspěšném spárování se zavolá metoda startGame, která odešle příkazy start ke klientům.

Při běhu programu se po každém tahu volá metoda sendMoveToClient. Ta jako argument přebírá odesílající ClientHandler. Poté je vyhledán partner v HashMapě a tomu je poslán tah.

4. Závěr

Co se týče povinné části, jednoznačně se ji povedlo naplnit. Původně bylo zamýšleno udělat pouze šachovnici, která se neumí zvětšovat a neumí prakticky nic jiného než táhnout figurou. Toto se naštěstí nestalo a výsledná práce toho může nabídnout mnohem víc.

Bonus klient-server model se také úspěšně podařilo vytvořit, dokonce i pro Fischerovy šachy, se kterými bylo mnoho komplikací, hlavně kvůli implementaci rošády.

Druhou část bonusu, tedy vytvoření vlastního enginu se bohužel nepodařilo realizovat, protože reprezentace šachovnice v tomto projektu by byla značně neefektivní. Na tah by se tedy místo sekundy čekalo i čtvrt minuty.

Původně bylo zamýšleno, že by se ještě implementovala možnost ukládat hry a také zobrazení aktuálních klientů a parametrů jejich šachovnice, pokud správně uživatel zadá adresu serveru. Na řešení těchto problémů však nezbyl čas, i když základy pro ně připraveny již jsou.

Hlavním úskalím už od začátku bylo, že se program implementoval bez předešlého návrhu. To vyústilo v nekvalitní reprezentaci šachovnice (objekt tlačítka, které by uchovávalo zároveň grafickou i logickou reprezentaci šachovnice).

Projekt již nemá cenu rozšiřovat, protože vše možné kromě horních problémů již bylo implementováno. Pro složitější problémy jako vytvoření výše zmíněného enginu, či taktických úloh nebo hry přes internet by se musela předělat třetina projektu.

5. Seznam obrázků

6. Použité zdroje

6.1. Použité technologie:

- Java
- NetBeans IDE 8.2
- UCI protokol
- Pixlr Editor
- Microsoft Word, Publisher
- Stockfish
- Typy souborů: .java, .class, .docx, .PNG, .pub, .pdf, .pptx (prezentace),.jar

6.2. Použitá literatura

Antonsusi. (11. Březen 2012). *Chess piece*. Načteno z Wikipedia: https://en.wikipedia.org/wiki/Chess_piece

aterai. (13. Červen 2012). *How to center items in a Java combobox.* Načteno z stackoverflow: https://stackoverflow.com/questions/11008431/how-to-center-items-in-a-java-combobox

Dijksman, L. (Duben 2004). *wbec-ridderkerk*. Načteno z Description of the universal chess interface (UCI): http://wbec-ridderkerk.nl/html/UCIProtocol.html

Flanagancz. (26. Květen 2007). *Minimax (algoritmus)*. Načteno z Wikipedia: https://cs.wikipedia.org/wiki/Minimax_(algoritmus)

Flis, O. (2017). *Painting Chess PNG image.* Načteno z picpng: https://www.picpng.com/image/chess-png-36105

Mahrsee, R. (16. Červen 2017). *Introducing Threads in Socket Programming in Java.* Načteno z geeksforgeeks: https://www.geeksforgeeks.org/introducing-threads-socket-programming-java/

Pexels. (21. Listopad 2016). *pixabay.* Načteno z battle-chess-checkmate-blur-1846807: https://pixabay.com/photos/battle-chess-checkmate-blur-1846807/

seekpng. (nedatováno). *Surrender Flag - White Flag Icon Png*. Načteno z seekpng: https://www.seekpng.com/ipng/u2q8w7e6y3i1y3e6_surrender-flag-white-flag-icon-png/

Tord Romstad, M. C. (Listopad 2008). *download.* Načteno z stockfishchess: https://stockfishchess.org/download/