VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ŠACHOVÝ PROGRAM PRO VARIANTU HOLANĎANY

CHESS PROGRAM FOR BUGHOUSE VARIANT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAREK STAŇA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/20117/2016/xstana03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Staňa Marek**
Obor: Informační technologie
Téma: **Šachový program pro variantu Holanďany**
Chess Program for Bughouse Variant
Kategorie: Umělá inteligence

Pokyny:

1. Nastudujte šachovou variantu pro čtyři hráče, tzv. Holanďany. Nastudujte metody pro programování umělé inteligence u hry šachy.
2. Navrhněte uživatelské rozhraní pro hru šachy pro čtyři hráče. Navrhněte ohodnocovací funkci a vyberte vhodnou metodu pro prohledávání stavovho prostoru hry. Zahrňte do ní i možnost předávat figurky spoluhráči.
3. Navržený program implementujte a otestujte úroveň hry. Umožněte i vzájemné hraní proti už existujícím programům.
4. Výsledky zhodnoťte a navrhněte případná vylepšení.

Literatura:

- Russell, Stuart J. and Peter Norvig. 2003. Artificial intelligence: a modern approach, ISBN:0137903952

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Rozman Jaroslav, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato práce popisuje postup při tvorbě šachového programu hrajícího variantu holanďany proti člověku i jiným programům. V úvodu jsou vysvětleny rozdíly v pravidlech oproti klasickým šachům, hlavní část se věnuje umělé inteligenci. Porovnává jednotlivé metody, které se používají pro tvorbu šachových programů, a adaptuje je pro variantu holanďany.

Abstract

This thesis describes process of creating a chess program playing Bughouse variant allowing to play against human or other programs. Firstly explains difference in Bughouse rules from classic chess, main part is about artificial intelligence. It compares individual methods used for making chess programs and adapts them to Bughouse variant.

Klíčová slova

Šachy, holanďany, bitboard, alfa-beta, ohodnocovací funkce, umělá inteligence.

Keywords

Chess, bughouse, bitboard, alpha-beta, evaluation function, artificial intellingence.

Citace

STAŇA, Marek. *Šachový program pro variantu Holanďany.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Rozman Jaroslav.

Šachový program pro variantu Holanďany

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jaroslava Rozmana. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Staňa
17. května 2017

Poděkování

Vedoucímu Ing. Jaroslavu Rozmanovi bych chtěl poděkovat za vedení a připomínky při tvorbě této práce.

Obsah

Úvod

Obsahem této práce je vytvořit šachový program, který bude schopný hrát variantu holanďany proti hráči, nebo proti jiným šachovým programů, které jsou schopny tuto variantu hrát. Cílem této práce je vytvořit program, který bude mezi ostatními šachovými programy konkurenceschopný. K tomuto účelu je součástí práce také průzkum výhod a nevýhod používaných algoritmů a struktur, které jsou použity v rámci práce. Názvy kapitol jsou psány v českém jazyce. Na začátku kapitol je uveden i anglický název, který se používá v anglicky psaných zdrojích a ukázkách algoritmů. V práci jsou použity anglické výrazy, pokud nejsou vhodné české protějšky.

Kapitola 1 slouží k seznámení s bughouse šachem a s jeho pravidly, hlavně jeho odlišnostmi oproti šachu klasickému. V této práci se často používá k vysvětlení pozic a tahů šachová notace, která je taktéž přiblížena v této kapitole. Dále pak jsou uvedeny šachové programy, které bughouse šach podporují a do jaké míry ho podporují. Závěr je pak věnovaný základním používaným strategiím hraní, které silně ovlivňují ohodnocování pozic.

Kapitola 2 je věnována reprezentaci šachovnice jakožto základního kamene pro šachový motor (engine), jejími možnými implementacemi a jejich výhodami a nevýhodami. Druhá část kapitoly pojednává o vytváření množiny možných tahů z určité pozice zachycené reprezentací.

V kapitole 3 jsou představeny různé algoritmy, které umožňují prohledávat stavový prostor a jejich rozšíření a vylepšování. V kapitole 4 jsou popsány způsoby, které se používají a byly použity pro ohodnocení pozice na šachovnici. V kapitole 5 je uvedena samotná implementace programu. V kapitole 6 je přiblížena současná situace na poli šachových programů, které podporují bughouse šach, jejich umělé inteligence i grafické uživatelské rozhraní a možnosti propojení těchto dvou částí.

Kapitola 1

Základní informace o holanďanech

Holanďany (používány také názvy bughouse šachy nebo zkráceně Bughouse, v anglicky mluvících zemí Pass-On Chess a Tandem Put-Back) je varianta šachu hraná na dvou šachovnicích ve dvoučlenných týmech. Zatím co původ šachů je velmi starý, holanďany pochází z období 60. let dvacátého století, ovšem přesnější původ je nejasný [16].

1.1 Pravidla Bughouse šachu

V této kapitole je čerpáno z pravidel Bughouse šachu [17]. Pravidla jsou vysvětlována hlavně v porovnání s klasickým šachem. Holanďany se hrají na dvou šachovnicích v týmech po dvou, kde jeden hráč v každém týmu má bílé kameny a druhý černé kameny. Hráči na každé šachovnici proti sobě hrají šachovou partii s upravenými pravidly.

Hra se obvykle hraje v tzv. *bleskovém tempu*, kdy každý hráč má na partii 3 nebo 5 minut bez časových bonusů za tah. Obě dvě šachovnice jsou tudíž opatřeny šachovými hodinami. Pokud se hraje bez časového omezení hráč smí čekat maximálně 3 tahy odehrané na druhé šachovnici. V této práci není časové omezení na partii, ale také se nevyužívá pravidlo čekání.

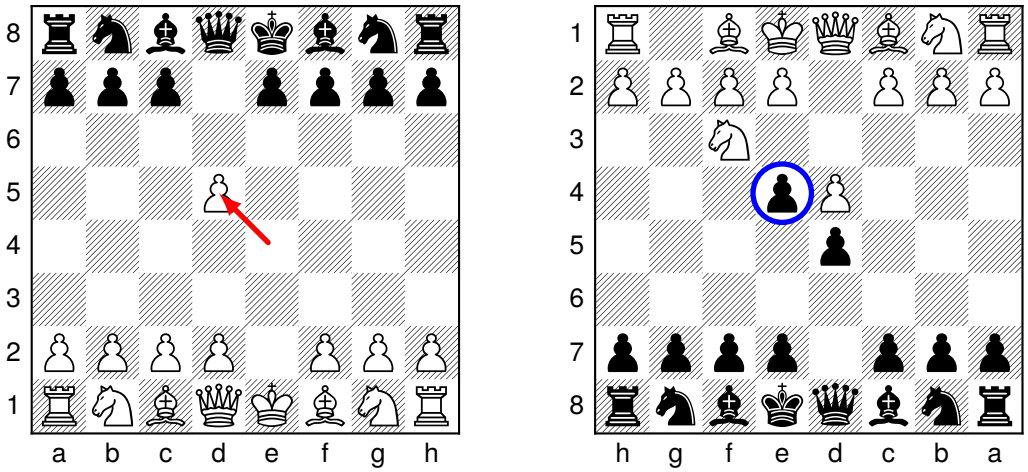
Spoluhráči si smí při partii radit a to i konkrétní tahy. Cílem hry je dát soupeři mat na kterékoliv šachovnici, což vede k okamžitému ukončení hry na obou šachovnicích. Hra může skončit i vypršení času jednoho z hráčů, což znamená prohru jeho týmu. Mat je pouze tehdy pokud hráč nemůže představit žádnou potencionální figuru. Tudíž mat musí být buď kontaktní, jezdcem, anebo dvojšachem. Partie končí remízou, pokud dojde k 3krát opakované pozici na jedné ze šachovnic a do pozice se nepočítá obsah zásoby figur hráčů.

Základní postavení figur a jejich pohyb jsou stejné jako v klasickém šachu.

Když hráč vezme soupeři kámen, předá jej svému spoluhráči, který si jej uchová v *zásobníku* (anglicky *Stuff-place*). Používá se i vyjádření „mít v ruce“. Hráč na tahu může místo pohybu figurou na jakékoliv prázdné pole na šachovnici položit figuru z ruky. Takto vyložený pěšec nemůže být položen na první a osmou řadu. Pěšec položený na druhou resp. sedmou řadu smí táhnout o dvě pole stejně jako pěšci stojící na této řadě od začátku hry. Položená věž může dělat rošádu stejně jako originální věž.

Pro zápis tahů se používá tzv. šachová notace. Šachová notace pro klasický šach se nazývá *Portable Game Notation* (PGN). Pro zápis tahů v této variantě se používá rozšířená verze *Bughouse Portable Game Notation* (BPGN). Pole na šachovnici se v ní označují jako průsečíky sloupců (označené písmeny a až h) a řady (označené čísly 1 až 8). Pro označení figur v šachové notaci se používá jedno, obvykle první písmeno:

Tým B, hráč 1 Tým B, hráč 2



Tým A, hráč 1 Tým A, hráč 2

Obrázek 1.1: Ukázková pozice

K	**K**ing	Král
Q	**Q**ueen	Dáma
R	**R**ook	Věž
B	**B**ishop	Střelec
N	k**N**ight	Jezdec
P	**P**awn	Pěšec

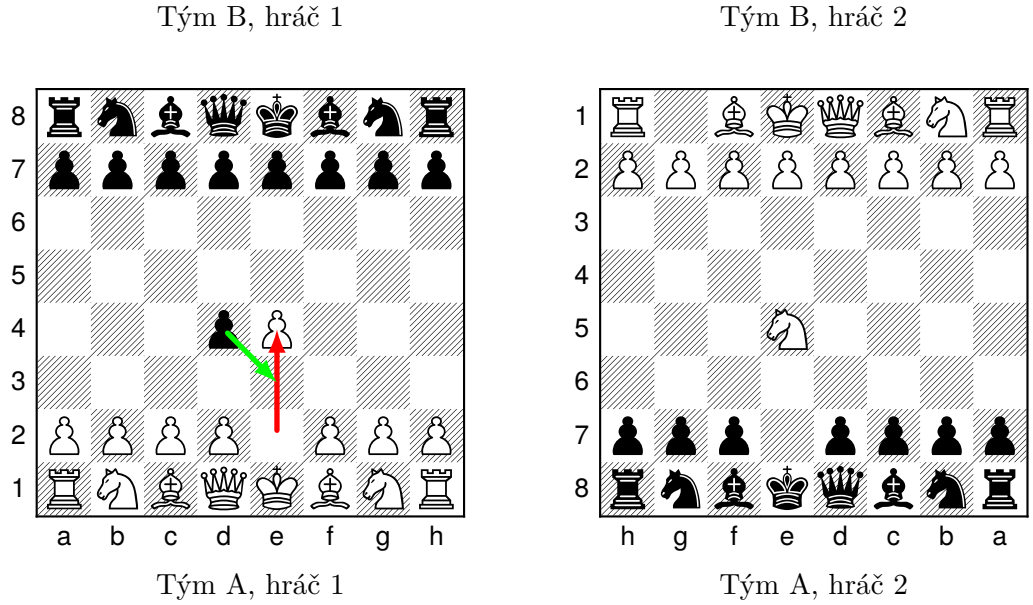
Obvyklý tah se skládá z označení figury a koncového pole. U tahu pěšcem se označení figury vynechává. Pokud na koncové pole může víc figur stejného typu, připisuje se rozdílný sloupec, řada nebo obojí výchozího pole (*Nfe7*). Rošáda se zapisuje jako *0-0* v případě malé rošády a *0-0-0* v případě velké rošády. Braní se značí znakem *x*. Tah *Nxf7* je tah jezdce, který vzal na poli *f7*. Hlavním rozdílem v notaci bughouse šachu je potřeba udržet kontinuitu tahů na obou šachovnicích. Obě partie se tedy zapisují dohromady a každá šachovnice se označuje písmenem. Velké písmeno pro bílého, malé pro černého.

1A. e4	První tah bílého na šachovnici A
1a. Nf6	První tah černého na šachovnici A
1B. Nf3	První tah bílého na šachovnici B
1b. Nc6	První tah černého na šachovnici B

Vykládací tahy se zapisují jako označení figury, znaku @ a koncového pole. Příkladem je text *N@f5*, který se dá číst jako, že jezdec byl položen na pole *f5*.

Příklad 1.1.1 *Na obrázku 1.1 bílý hráč týmu A vzal soupeři černého pěšce na d5 a předal spoluhráči, který jej položil na pole e4.*

Pěšec, který dojde na poslední řadu se promění podle výběru hráče v jezdce nebo dámu. Poté, co je sebrán ze stolu a předán spoluhráči, je proměněn zpět v pěšce. Pěšci, stejně jako v klasickém šachu, mohou brát mimochodem (en passant). Pokud pěšec z druhé resp. ze



Obrázek 1.2: Položený pěšec na *d4* smí vzít mimochodem pěšce na *e4*

	Klasický šach	Sunsetter	Sjeng
Pěšec	100	100	100
Jezdec	290	192	210
Střelec	300	195	230
Věž	500	200	250
Dáma	940	390	450

Tabulka 1.1: Hodnoty figur v klasickém šachu a v šachovým motorech pro holanďany

sedmé řady postoupí o dvě pole, pěšec stojící na sousedním sloupci vedle cílového pole smí v následujícím tahu vzít pěšce způsobem znázorněným na obrázku 1.2. Tento speciální tah může být proveden i pěšci položenými na tato pole v předchozím tahu.

1.2 Hodnota figur

Hodnota figur je základním prvkem pro ohodnocování pozice, která je detailně popsaná v kapitole 4.1. Každá figura má jinou hodnotu, obvykle přepočítávanou na nejslabší figuru ve hře. V šachu je nejslabší figurou pěšec, jenž má nejčastěji hodnotu jedna. Některé výhody mohou mít menší hodnotu než pěšec a proto šachové programy přepočítávají hodnoty figur na centipěšce - setinu pěšce. V tabulce 1.1 je vidět jaké hodnoty jsou používány v klasickém šachu a jaké používají šachové programy pro holanďany. V tabulce není zahrnuta hodnota krále, kterého nemá význam ohodnocovat, protože ho nelze vyměnit ani vzít. Pokud hrozí jeho braní dalším tahem, znamená to mat - konec partie, tudíž se používá maximální možná hodnota.

Kapitola 2

Reprezentace šachovnice a generování tahů

2.1 Reprezentace šachovnice

Reprezentace šachovnice je základním prvkem šachového programu, který udržuje přehled o současném stavu šachovnice. Je vhodné, aby se operace nad pozicí, zejména generování tahů, prováděli co nejrychleji a nejefektivněji. Kromě samotné polohy figur je třeba mít při popisu pozice uložené dodatečné informace o hráči na tahu, možnostech rošády a braní mimochodem, pozicích prosazených pěšců a historii tahů pro pravidla třikrát opakované pozice a pravidlo 50 tahů. Všechny tyto dodatečné informace jsou důležité pro vytvoření všech legálních tahů a pro správné ohodnocení pozice. Existují dva přístupy jakými lze pozici na šachovnici zachytit:

1. Reprezentace orientované na pole
2. Reprezentace orientovaná na figury (viz kapitola 2.1.3 Bitboard)

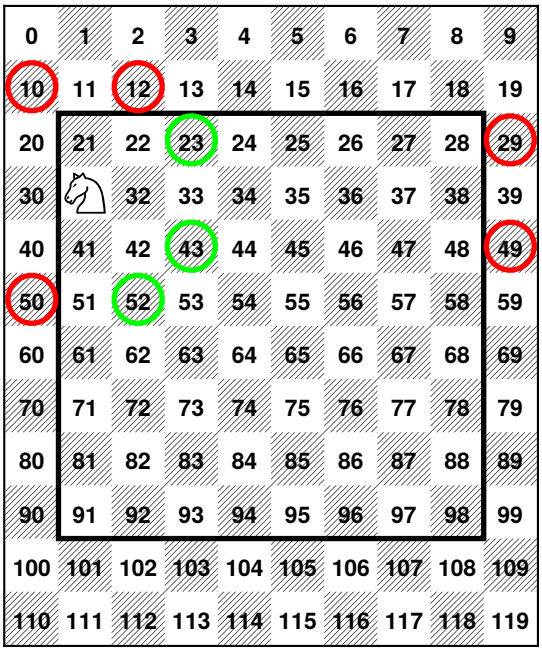
Často je používaná také hybridní verze, která k popisu pozice využívá oba přístupy a tím eliminuje jejich nevýhody.

2.1.1 Pole 8x8

Pole 8x8 je základní zástupce reprezentace orientované na pole. Jedná se buď o dvourozměrné pole znaků nebo čísel označující kameny a prázdné pole, které je indexované sloupci a řadami, nebo častěji o jednorozměrné pole s indexy 0..63. Tento způsob záznamu má výhodu ve velmi rychlém získání informací o tom, která konkrétní figura stojí na daném poli. Naproti tomu nevýhodou bývá generování tahů z této reprezentace. Při každém vygenerovaném tahu je potřeba se ujistit, že figura nestojí na poli mimo šachovnici, což tento proces velmi zpomaluje.

2.1.2 Pole 10x12

Pole 10x12 je matice, která rozšiřuje pole 8x8 o prázdné řady okolo. Originální pole je nahoře a dole ohraničeno dvěma řadami a jedním sloupcem na každém boku. Důvodem pro dvě prázdné řady je pohyb jezdcem, který se pohybuje pouze o dvě řady nebo o dva sloupce. Na bocích stačí pouze jeden, protože levý a pravý okraj se navzájem dotýkají. Výhodou tohoto



Obrázek 2.1: Vygenerování tahů jezdce s vyznačenými legálními tahy - 10x12 board

pole je, že je možné vygenerovat všechny tahy a poté nevalidní tahy odstranit. Odpadá tím nutnost kontrolovat okraje šachovnice při generování jednotlivých tahů.

2.1.3 Bitboard

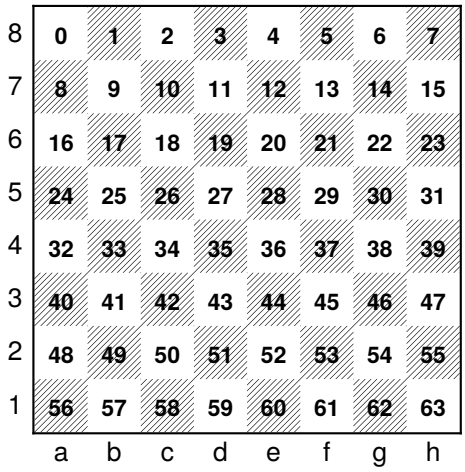
Tato metoda byla poprvé použita v roce 1950 pro hru dáma. Pro šachy se začala používat až o deset let později v programu KAISSA. Nyní tzv. *bitboardy* neboli bitová pole jsou nejčastěji používanou reprezentací šachovnice. Využívají pole 64 bitů. Pro každý typ figury je vytvořeno toto bitové pole, kde každý bit odpovídá jednomu poli na šachovnici. Můžeme tedy například zjistit, jestli je některé pole obsazeno(1), nebo jestli je prázdné(0). Bitboard lze indexovat více směry. V této implementaci označuje nultý bit pole a8 a 63. bit pole h1 viz obrázek 2.2.

Příklad 2.1.1 *Vezměme si bitboard černých jezdců ve výchozí pozici. Jeden stojí na g8 a druhý na b8. Tyto pole odpovídají indexům 6 a 1. Pokud to převedeme na 64-bitové číslo bude výpočet následující.*

$$2^1 + 2^6 = 66_{dec} = \boxed{42_{hex}}$$

Přirozenější je ovšem zápis v šestnáctkové soustavě, kde 4 pole na šachovnici jsou jedním jednotkovým hexadecimálním číslem. Celkem tedy šestnáctimístné hexadecimální číslo. V obrázku 2.2 mají sloupce a,e hodnotu 1, sloupce b,f hodnotu 2, sloupce c,g hodnotu 4 a sloupce d,h hodnotu 8.

$$b8 \rightarrow 2$$
$$g8 \rightarrow 4$$
$$\boxed{42_{hex}}$$



Obrázek 2.2: Způsob indexování bitboardu

Bitboardy jsou velmi jednoduché na vytváření nové pozice pomocí tahu. Tah je také zaznačen nastavenými bity na polích odkud a kam figura táhne. Pro provedení tahu tak stačí provést XOR operaci:

$$(puvodniPozice \oplus bitboardPresunuFigury) \tag{2.1}$$

Pro zjištění, zda je dané pole obsazené, slouží operace:

$$poleZajmu \wedge (bileFigury \oplus cerneFigury) \tag{2.2}$$

A pro detekci jestli je tah proveditelný a neponechává krále v šachu:

$$utocneTahySoupere \oplus poziceKrale \tag{2.3}$$

2.2 Generování tahů

Velkou výhodou bitboardů je možnost v nich ukládat nejen pozice figur, ale také jejich pohyb. Bity v log. 1 jsou příznakem polí, na které figura může vstoupit, bity v log. 0 jsou opakem. Tohoto lze využít tak, že využijeme předpřipravené bitboardy pohybu, které má každá figuru pro každé pole na šachovnici. Tyto bitboardy pohybu vyznačují, kam by mohla figura táhnout, pokud by byla šachovnice prázdná.

Když mluvíme o generování tahů je dobré si definovat dva druhy šachových figur. Tzv. „klouzající“ figury jako střelec, věž a dáma, a ty ostatní (král, pěšec, jezdec). Hlavní důvod pro takové rozdělení je ten, že pro krále, pěšce a jezdce stačí použít jejich bitboardy pohybu a logickými operacemi *XOR* a *AND* s bitboardem figur stejné barvy získáme bitboard polí, na které může v tomto tahu figura táhnout.

$$MoveMask = KnightMask[i] \oplus AllyMask \wedge KnightMask[i]$$

Tento způsob nelze aplikovat na klouzající figury, protože jim mohou v pohybu po řadě nebo diagonále stát v cestě blokující figury. Zde je potřeba použít některý z následujících algoritmů.

2.2.1 Bitový sken

Výstupem algoritmů na generování tahů je 64 bitový bitboard. Ten je ještě potřeba převést na jednotlivé indexy polí. Naivním přístupem by bylo procházet takový vstup bit po bitu a hledat nastavené bity. Tento způsob je velmi vyčerpávající, vzhledem k tomu, že jen několik málo bitů je nastavených, proto by většina z 64 přístupů byla prohlédnuta zbytečně.

Nejméně významný bit Nejméně významný bit resp. nejméně významný nastavený bit (LS1B) je to, co se při algoritmu na získávání nastavených bitů využívá jako bit pro skenování. Získá se jeho index, jeho hodnota se změní na nulu a hledá se znovu. Toto se děje dokud výsledný bitboard není prázdný.

De Bruijnova metoda Pro určení indexu bitu využívá algoritmus posloupnost B(2,6). Abecedu tvoří 2 symboly – 0 a 1, posloupnost má délku 6. Takto pak lze indexovat pole s pozicemi nejméně významných bitů nastavených na 1 [7]. Nejprve je izolován LS1B:

$$number \wedge -number \tag{2.4}$$

Tento bit se pak chová jako bitový posun De Bruinovy posloupnosti.

Algoritmus je efektivní pouze pro řídké bitboardy. Pokud je nastavených víc bitů po sobě, efektivita hledání LS1B se snižuje. Proto je možné implementovat různé kombinace a ty na základě očekávané obsazenosti bitboardu střídat.

```
static private final long deBruijn = 0x03f79d71b4cb0a89L;
static private final int[] magicTable = {
        0, 1,48, 2,57,49,28, 3,
        61,58,50,42,38,29,17, 4,
        62,55,59,36,53,51,43,22,
        45,39,33,30,24,18,12, 5,
        63,47,56,27,60,41,37,16,
        54,35,52,21,44,32,23,11,
        46,26,40,15,34,20,31,10,
        25,14,19, 9,13, 8, 7, 6,
};

static public int bitScanForwardDeBruijn64 (long b) {
    int idx = (int)(((b & -b) * deBruijn) >>> 58);
    return magicTable[idx];
}
```

2.2.2 Pseudo-legální a legální tah

Pseudo-legální tah je takový tah, který byl vygenerován z pozice, ovšem nemusí být platným tahem, pokud ponechává krále v šachu. Kontrola platnosti tahů (generování legálních tahů) je příliš komplexní (braní figury, rošáda, en-passant, pohyb z vazby) a tudíž i náročné. Kontrola platnosti je provedena až při generování tahů v uzlu, do kterého tah vede. V listech stromu, kde je prováděno hodnocení pozice, je třeba ovšem kontrolu učinit ihned a to pomocí útočných polí soupeřových figur.

Legální tah je pseudo-legální tah, který nenechává krále v šachu. Tah, který je nakonec enginem proveden, musí být legálním tahem.

2.2.3 Magické bitboardy

Magické bitboardy je způsob získávání bitboardu pohybu klouzajících figur, který využívá perfektního hashování k získání indexu do databáze. V současnosti asi nejrychlejší a nejrozšířenější přístup pro generování tahů. [14] Kromě databáze pozic je třeba mít předpočítané tzv. magická čísla a hodnoty bitového posunu pro každé pole. Tyto všechny hodnoty se používají pro funkci algoritmu tak, že bitboard blokujících figur daného pole je vynásoben magickým číslem. Výsledek této operace je bitově posunut o hodnotu určenou pro dané pole. Tím jsme získali index do databáze, kde se nachází námi hledaný bitboard.

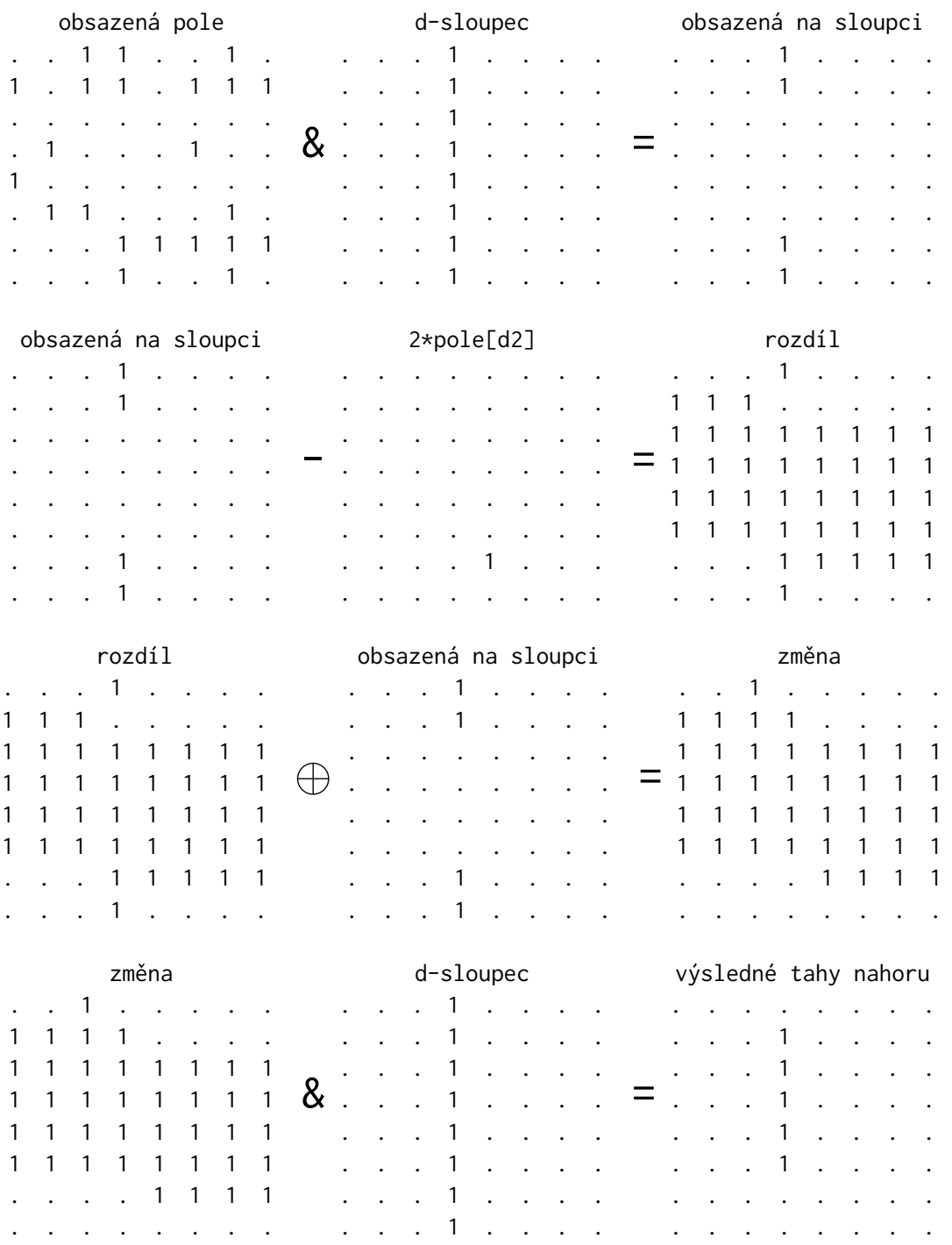
2.2.4 Hyperbola Quintessence

Je metoda, která využívá tzv. *o XOR (o-2s)* trik a obrácené bitboardy. Název této metody vychází z projektu, ve kterém byla představena. Na tabulce 2.1 je vidět algoritmus, který se na bitboard aplikuje pro věž na poli d2. První a poslední krok v tabulce je jen získání skupiny bitů, které nás zajímají, v tomto případě sloupec d. Druhý a třetí krok jsou právě tím

obsazené XOR (obsazené - 2 · pole)

trikem. Tímto způsobem lze získat figury ležící pouze směrem dolů nebo vpravo od pole (v pozitivním směru), ze kterého hledáme tahy. Zde přicházejí ke slovu obrácené bitboardy. Po obrácení bitboardu můžeme stejný algoritmus použít i na pole v negativním směru a oba výsledky spojit.

Hlavní nevýhodou tohoto algoritmu je časté obracení celého bitboardu, což může být velmi pomalé. Programovací jazyk Java ovšem používá strojovou instrukci místo běžné metody pro obrácení bitového pole [1][8]. Vzhledem k tomu, že implementace této práce je tvořena v jazyce Java, byla z experimentálních důvodů použita tato metoda.



Tabulka 2.1: Postup Hyperboly Quintessence

Kapitola 3

Vyhledávácí algoritmy

3.1 Minimax a Negamax

Minimax je algoritmus vytvořený pro maximalizování zisku a minimalizování ztrát při nejhorším možném scénáři v hrách dvou hráčů s nulovým součtem, kde oba hráči mají dokonalou informaci o hře. Prochází stavový prostor do určité hloubky, ze které poté aproximované hodnocení pozice předává uzlům v nižší hloubce. Vychází z předpokladu, že oba hráči budou vybírat nejlepší možné tahy v každé pozici. To znamená, že pokud je v dané hloubce na tahu hráč A, pro kterého algoritmus hledá řešení, tak hledá nejlepší tah, ale pokud je v dané hloubce na tahu hráč B, tak je vyhledáván nejhorší možný tah z pohledu hráče A. Jedná se o rekurzivní algoritmus, ve kterém se střídají metody `max()`, hledající nejlepší z možných tahů, a `min()`, hledající nejhorší z možných tahů.

Negamax je varianta minimaxu založená na faktu, že zisk jednoho hráče je ztrátou soupeře `max() = -min()`. Toho lze využít pro jednodušší implementaci minimaxu. Pro každý tah se hledá záporné maximum z tahů ve vyšší hloubce. `score = -max(depth-1)`. Při používání negamaxu je důležité si uvědomit, jakým způsobem bude volán z kořenového uzlu a že je potřeba upravit skóre, které vrací hodnotící funkce.

3.2 Alfa-beta

Alfa-beta je algoritmus, který vychází z minimaxu. Oproti minimaxu ve svém algoritmu ovšem definuje dvě nové proměnné, které využívá pro určení, které větve bude prohledávat. Tyto proměnné se mění dle už prohledaných částí stromu. Vychází z předpokladu, že pokud už našel tah, který je horší než nejlepší tah z už prohledné části stromu, tak nemá smysl prohledávat zbytek podstromu, protože už našel tah pro soupeře, který bude výhodnější než současná varianta. Přesné znění algoritmů Minimax a Alfa-beta včetně jejich pseudokódů lze najít například v přednáškách předmětu Základy umělé inteligence [18].

Ačkoli i u alfa-bety je časová složitost exponenciální, nárůst této funkce není tak rapidní jako v případě Minimaxu. Toto platí pouze za předpokladu, že nejlepší tah je nalezen ve stromě co nejdříve. V nejhorším případě totiž může být prohledán celý stavový prostor a nejlepší řešení může najít až na konci, čím se časovou náročností neliší od minimaxu. Proto se tahy před vložením do alfa-bety staticky ohodnotí a seřadí. Výsledný alfa-beta strom s perfektním řazením tahů se nazývá *kritický* nebo *minimální* strom [2]. Uzly minimálního stromu jsou kritické proto, že jakýkoliv prohledávací algoritmus musí projít alespoň tyto uzly v případě stejného řazení tahu a stejného výchozího uzlu. Pokud vezmeme hloubku

hloubka	nejhorší případ	ideální případ
H	B^H	$B^{H/2} + B^{H/2} - 1$
0	1	1
1	40	40
2	1,600	79
3	64,000	1,639
4	2,560,000	3,199
5	102,400,000	65,569
6	4,096,000,000	127,999
7	163,840,000,000	2,623,999
8	6,553,600,000,000	5,119,999

Tabulka 3.1: Počet prohledávaných pozic v Alfa-betě při počtu tahů $B = 40$ [9]

prohledávání jako H a počet možných tahů v každé pozici B, tak celková velikost stromu bude B^H. Alfa-beta tuto velikost stromu snižuje až na minimální strom:

$$B^{H/2} + B^{H/2} - 1. \tag{3.1}$$

Ve smyslu časové složitosti kritický strom snižuje $\mathcal{O}(B^H)$ exponenciální velikost stromu na $\mathcal{O}(\sqrt{B^H})$ odmocninu z této hodnoty [5].

3.2.1 Typy uzlů

D. Knuth a R. Moore [5] poprvé klasifikovali tři typy uzlů, které se vyskytují v alfa-beta stromě. Tyto typy tahů se ukládají spolu s tahy v transpoziční tabulce [10].

PV-Node Je takový uzel, který je ohodnocen v rozmezí mezi alfou a betou. Jsou to uzly, které musí být vždy prozkoumány.

Cut-Node Tyto uzly jsou hodnoceny jako větší než β. Protivník už nalezl tah, který je pro něj výhodnější, tím pádem tento podstrom nemá význam dále prohledávat. Tyto uzly způsobují redukci stromu.

All-Node Hodnota těchto uzlů je menší než α. Každý poduzel těchto uzlů je prohledáván.

3.2.2 Řazení tahů

Jak bylo v předchozí kapitole vysvětleno, je potřeba tahy, pro co nejlepší fungování alfa-bety, předpřipravit a seřadit podle jejich kvality. Samozřejmě je téměř nemožné tahy řadit přesně od nejlepšího po nejhorší, vzhledem k tomu, že ke správného ohodnocení potřebujeme právě hloubkové prohledávání, které provádí alfa-beta, ale i přibližné seřazené tahy nám pomohou zmenšit časovou složitost alfa-bety.

Tahy jsou řazeny podle tohoto pořadí:

1. Nejlepší tahy uložené v tabulce z předchozí iterace
2. Výhodné tahy podle MVV-LVA

3. Ostatní tahy podle jejich vylepšení pozice
4. Nevýhodné tahy podle MVV-LVA

3.2.3 MVV-LVA

MVV-LVA (most valuable victim - least valuable attacker) je metoda, která pomáhá řadit tahy braní. Vyšší hodnotu dává tahům, které útočník má nižší hodnotu než oběť.

	P	N	B	R	Q
P	11	108	109	132	322
N	-91	6	8	19	220
B	-93	4	6	27	218
R	-119	-22	-20	1	192
Q	-309	-212	-210	-189	2

Tabulka 3.2: Tabulka řazení (svisle útočníci, horizontálně oběti)

Hodnoty v tabulce jsou vypočítány rozdílem hodnocení figur na šachovnici a na ruce definovaných v kapitole 4.1. Pro útočníka je použita hodnota „ze šachovnice“ a oběť je použita hodnota „na ruce“. Takto je program podporován k výměnám figur.

3.2.4 Kaskádová metoda prohledávání do hloubky

Vzhledem k předpokladu, že přesnost statického hodnocení pozice se zvyšuje s hloubkou prohledání, je žádoucí používat alfa-beta prohledávání s co nejvyšší hloubkou. Protože ovšem není možné předem říct, jak bude prohledávání do určité hloubky časově a prostorově náročné, je vhodné vytvořit nějaké omezení, které bude určovat, jak moc může algoritmus brát časové a prostorové zdroje [6].

Kaskádová metoda (ang. Depth-First Iterative deepening) je algoritmus, která iteruje volání alfa-bety a s každou iterací zvyšuje hloubku propočtu. Díky tomu je možnost algoritmus kdykoliv ukončit a dostat hodnocení, které je s určitou přesností správné, za cenu, že dochází k nárůstu počtu prohledaných uzlů. Tato cena ovšem není nijak velká.

Příklad 3.2.1 *Vezměme si minimax s větvením tahů B 40 a hloubkou propočtu H 3. Z tabulky počtů prohledávaných pozic 3.1 lze vyčíst, že při takovém případu je potřeba projít 64,000 uzlů. Pokud je použita kaskádová metoda nad tímto zadáním, zvýší se počet uzlů o 40 + 1,600 = 1640, což je nárůst o zhruba 2,5%.*

Kaskádová metoda byla do alfa-beta prohledávání přidána jako možnost časově omezovat prohledávání stavového prostoru. Ukázalo se však, že má kupodivu možnost nikoliv navýšit ale snížit časovou náročnost alfa-bety. Toho lze docílit ve spojení s řazením tahů. V každé iteraci se ukládají nejlepší tahy do tabulky včetně jejich hodnocení. Toto hodnocení se v následující iteraci použije při řazení těchto tahů. Tahy, které nejsou v tabulce, se ohodnocují klasickým způsobem.

3.3 Prořezávání vyhledávacího stromu

Prořezávání (ang. Pruning) jsou heuristiky, které zmenšují velikost stromu, aniž by ovlivnily výsledek prohledávání. Vzhledem ke specifičnosti bughouse šachu některé heuristiky tuto

vlastnost ztrácejí, takže nemá smysl je používat. Příkladem toho je *Mate Distance Pruning*. Tato heuristika po nalezení vynucenému matu v některé větvi odřeže větve, které nevedou ke kratšímu matu. Spoluhráč může potřebovat figury, aby sám ubránil svou pozici před matem, je vhodné ponechat možnost prohledání tahů, které mohou najít možnost, jak předat spoluhráči potřebnou figuru a najít případný mat v pozdějším tahu.

3.3.1 Metoda nulového tahu

Metoda nulového tahu (ang. Null-move pruning) je heuristika založená na předpokladu, že v každé pozici je nějaký tah lepší než žádný tah (*null move*). Algoritmus prohledává stavový prostor podstromu s menší hloubkou, který vnikne, pokud se vzdáme svého tahu. Výsledek tohoto prohledávání porovná s β [3]. Pokud je větší dojde k prořezání stromu. Nevýhoda této heuristiky je v koncovce, kdy často nastává pozice, kdy přenechání tahu soupeři je nejlepší možnost a to vede ke špatným výsledkům. Tato nevýhoda se řeší podmínkou pokud je na šachovnici málo figur metoda nulového tahu se přestane používat. Tuto nevýhodu ovšem v holanďanech nemusíme řešit vzhledem k tomu, že koncovka zde nenastává. Tento algoritmus je volán před klasickým alfa-beta průchodem a provádí volání metody podobné metodě alfa-beta do omezené hloubky. Interval, který je v tomto volání použit, je zúžen kolem hodnoty beta a tím je docíleno k velkému množství redukcí. Jak je vidět v algoritmu 3.1, aby šlo tuto metodu použít je třeba, aby pozice splňovala podmínku, že král hráče na tahu není v šachu. Tato podmínka je logická, protože pokud by byl král v šachu a předali bychom tah soupeři, nejlepší soupeřův tah by byl vzít krále, což je ilegální tah.

```
if(allowed_null && !isKingInCheck(color)){
  score = -NullMove(-beta, 1 - beta,depthLeft-null_depth-1);
  if(score > beta)
  return score;
}

int NullMove(unt alpha, int beta, int depthLeft ){
    if( depthLeft <= 0 ) return QuiescenceSearch(alpha, beta);
    MoveOrdering(moves);
    for (all moves){
      makeMove();
      score = -NullMove(-beta, -alpha, depthLeft - 1);
      unmakeMove();
      if (score >= beta) return beta;
    }
    return beta-1;
}
```

Algoritmus 3.1: Metoda nulového tahu

3.4 Dopočet do klidné pozice

Dopočet do klidné pozice (ang. quiescence search) je rozšíření alfa-bety, kde se ohodnocují pouze „klidné“ pozice [13]. Když algoritmus zjistí, že v hloubce stromu, kde obvykle probíhá ohodnocování, není pozice „klidná“, prohledává stavový prostor hlouběji, dokud se nedostane do pozice, kterou má cenu hodnotit. „Neklidná“ pozice je pozice, ve které zrovna

probíhají výměny nebo ve které je některý z hráčů v šachu. V případě šachu je řešením tzv. *Check extension*, které zvýší hlouku prohledání v současné větvi o jedna. Tuto heuristiku klidné pozice využívá téměř každý moderní šachový program [11]. Využívání tohoto dopočtu je nutné pro potlačení efektu horizontu.

```
private int AlphaBeta(int alpha, int beta, int depth) {
  ...
  if (depth == 0) {
    if (isKingInCheck){
      MoveOrdering(moves);
      for (all moves){
        makeMove();
        score = -AlphaBeta(-beta, -alpha, 0);
        unmakeMove();
        if (score >= beta) return beta;
        if (score > alpha) alpha = score;
      }
      return alpha;
    }
    return QuiescenceSearch(alpha, beta);
  }
    ...
    //depth > 0 probiha klasicke alfa-beta prohledavani
}
```

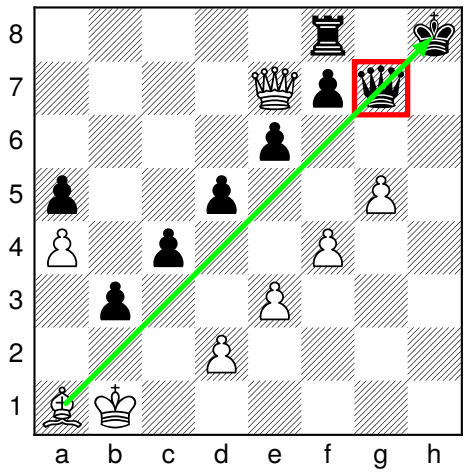
Algoritmus 3.2: Agoritmus alfa-beta - část pro listové uzly

V algoritmu 3.2 je část alfa-bety, kde probíhá kontrola šachu (Check Extension) a prohledávání do klidné pozice.

3.4.1 Efekt horizontu

Zatím nevyřešeným problémem šachových programů je prokládání tahů, které odsouvají nevyhnutelnou ztrátu materiálu až za maximální prohledávanou hloubku - horizont, takže samotná ztráta je pro program skrytá. Efekt horizontu (ang. Horizon effect) je problém, který nastává, když se program rozhodne obětovat materiál navíc proto, aby oddálil nevyhnutelnou ztrátu.

Příklad 3.4.1 *Vezměme si následující pozici:*



V této pozici už na první pohled černý má dámu ve vazbě. Ovšem při ohodnocování pozice programem s žádným, nebo jednoduchým dopočtem do klidné pozice i při prohledání do hloubky 8 tahů (Pb2, Bxb2, Pc3, Bxc3, Pd4, Bxd4, Pe5, Sxe5) pozice stále neukazuje jasně ztrátu dámy a program si nadále myslí, že zachránil dámu za cenu pouze 4 pěšců.

V bughouse šachu tento problém nastává kvůli pokládání figur z ruky častěji než v klasickém šachu, kdy se může program pokoušet vkládat figury do šachu a tím oddalovat nevyhnutelné. Tomuto problému nelze úplně zabránit, ale kvalitní hodnocení klidové pozice ho potlačují.

Kapitola 4

Ohodnocovací funkce

Protože není možné vytvořit variantní strom od začátku až po konec partie, která by nám přesně určila výsledek - výhra, prohra, remíza, je třeba hodnotu současné pozice aproximovat. Obvykle výsledek téhle aproximace zapisuje jako reálné číslo omezené na několik desetinných čísel. Kladné číslo znamená lepší postavení bílého hráče, záporné číslo znamená lepší postavení černého hráče. Pozice ohodnocená nulou je pozice, ve které oba hráči stojí stejně dobře.

4.1 Materiální hodnota figur

Jedná se o naprostý základ u ohodnocování pozice. Nejjednodušším způsobem jakým lze ohodnotit sílu figur je spočítat všechny figury jedné barvy na šachovnici a následně druhé barvy. Hráč, který disponuje větším množstvím materiálu stojí na základě tohoto hodnocení lépe. Ovšem každá figura bývá jinak ceněná. Takže k tomuto jednoduchému hodnocení přidáme ještě hodnotu jednotlivých figur. V klasickém šachu se používají hodnoty 1 pro pěšce, 9 pro dámu, 5 pro věž, 3 pro střelce a 3 pro jezdce.

Tyto hodnoty jsou ovšem relativní. Jak se pozice v klasickém šachu blíží do koncovky, střelec získává na síle na rozdíl od jezdce, který naopak kvůli nízké mobilitě ztrácí. Vzhledem k neustálému kolování figur mezi šachovnicemi je nemožné v holanďanech přejít do koncovky, tudíž se v průběhu hry nemění hodnota figur, tak jak v šachu klasickém.

Hodnoty z klasického šachu ovšem neodpovídají stylu hraní v holanďanech. V holanďanech, kde materiál koluje mezi šachovnicemi, mají figury menší hodnotu, protože jejich ztráta není až tak bolestivá. Je potřeba zohlednit pokládání figurek z ruky na šachovnici. Pěšci v holanďanech získávají na ceně, protože je lze položit do agresivních pozic, naopak dochází k devalvaci silnějších figur jako jsou dámy a věže. Hodnota dámy klesá na méně než polovinu a hodnota věže je téměř na úrovni střelce. Nízká hodnota dámy je dána celkovým zvýšením mobility figur díky dosazováním figur na vhodná pole. V porovnání s těmito tahy nevypadá přirozená mobilita dámy jako dostatečná výhoda.

Do hodnoty figury je potřeba zahrnout taktéž figury, které hráč drží v ruce. Z praktických experimentů z publikace *Learning the Piece Values for Three Chess Variants* [4] je zřejmé, že hlavně věž, střelec a jezdec v ruce mají vyšší hodnotu. Díky této zvýšené hodnotě figur na ruce a tabulkám uvedených v následující kapitole 4.2 můžeme zajistit, že šachový motor bude pokládaní figur hodnotit lépe na polích, kde bude mít figura hodnotu vyšší než na ruce.

Figura	postavená	v ruce
Dáma	420	422
Věž	230	231
Střelec	204	210
Jezdec	202	208
Pěšec	100	111

V této práci se operuje s dvěmi tabulkami hodnot. Jedna tabulka určená pro figury na šachovnici a druhá pro figury v ruce. V tabulce jsou uvedeny hodnoty v centi-pěšcích. Tyto hodnoty byly původně stejné jako v Sunsetteru 1.1, ale v průběhu testování a upravování pozičních tabulek docházelo často k bezúčelným obětem figur za pěšce, tudíž byly hodnoty figur lehce navýšeny.

4.2 Ohodnocení figury na základě její pozice

Do hodnocení pozice se počítá nejen počet a základní hodnota materiálu, ale jeho hodnocení je také počítána z jeho postavení na šachovnici. K zachycení takových rozdílů se používají tabulky polí pro figury. Každá figura má tabulku, kde je každé pole ohodnoceno. Tím dochází k určení, které pole jsou pro danou figuru vhodné.

-50	-40	-30	-30	-30	-30	-40	-50
-40	-20	0	5	5	0	-20	-40
-30	5	10	15	15	10	5	-30
-30	0	15	20	20	15	0	-30
-30	5	15	20	20	15	5	-30
-30	0	10	15	15	10	0	-30
-40	-20	0	0	0	0	-20	-40
-50	-40	-30	-30	-30	-30	-40	-50

Tabulka 4.1: Poziční tabulka hodnocení pro jezdce

V tabulce 4.1 je vidět, že kvalita jezdce je nejvyšší v centru a postupně klesá ke krajům šachovnice. V centru šachovnice má 8 polí pohybu. Naproti tomu jezdec v rohu má pouze 2 pole, proto je jeho hodnocení menší než pro pole uprostřed.

4.3 Hodnocení pozice

Celkové hodnocení pozice je dáno součtem hodnot figur, přičemž u černých figur jsou brány záporné hodnoty. Hodnota figur je navýšena nebo snížena o jejich pozici na šachovnici podle tabulek. Dále do hodnocení pozice je potřeba zahrnout další aspekty pozice jako například bezpečí krále. Nejjednodušeji lze toto hodnotit prostou vzdálenosti figury od krále. Čím větší tato vzdálenost je, tím víc je král v bezpečí.

Kapitola 5

Implementace

5.1 Popis implementace

Aplikace je implementovaná v jazyce Java SE 8. Je vytvořeno grafické uživatelské rozhraní, motor a kontrolér, který obsluhuje herní komunikaci s jinými šachovými motory. Využívá frameworku Swing k vytvoření uživatelského rozhraní, které funguje na principu návrhového vzoru MVC. Tahy hráče jsou obsluhovány jako tzv. *event dispatching thread* a tahy šachových motorů jsou implementovány jako *worker thready*. V případě cizích motorů se spustí daný program a otevřou se input a output streamy, na které se začnou posílat zprávy ve formátu upřesněného v kapitole 6.2. V případě motoru, který je součástí této práce *worker thread* pracuje se stejným modelem pozice jako kontrolér.

Program je rozdělen na 4 hlavní balíky *BoardRepresentation*, *Bot*, *Controllers* a *GUI*.

BoardRepresentation

Součástí tohoto balíku jsou všechny třídy potřebné k udržování informace o šachovnici a generování tahů. Obsahuje následující třídy:

Board Instance této mají za atributy současnou pozici, počet odehraných tahů a seznam odehraných tahů. Chová se jako model (rozšiřuje `Observable`), který při změně pozice vyrozumí objekty (implementující `Observers`), které tato změna zajímá. Změnou pozice je myšleno provedení tahu `makeMove()` nebo předání figury od druhé šachovnice `addToStack()`.

Position Třída, která představuje pozici. Jako atributy má pole figur, zásobníků a další blíže definované v kapitole 2. Obsahuje metody `getMoves()` na generování tahů, `makeMove()`, `unmakeMove()` pro provedení a vrácení tahu.

Move Objekty této třídy jsou generovány pomocí výše zmíněné metody třídy Position `getMoves()`. Ukládá veškeré potřebné informace o tahu.

Bitboard Abstraktní třída, ze které dědí třídy jednotlivých figur, které lze nalézt v balíku *BoardRepresentation/pieces*. Obsahuje metodu `getMoves()`, kterou jednotlivé figury přepisují.

Chesspiece Enum typů figur, které v sobě uchovává ohodnocovací tabulky pro jednotlivé figury.

Bot

V tomto balíku jsou kontroléry motorů. Pouští a zastavují *worker thready*, které se starají o zahrání tahu motoru.

AIController Třída s veškerými metodami týkající se prohledávání stavového prostoru.

XBoardController Třída zajišťující komunikaci s jiným šachovým motorem.

Controllers

Obsahuje třídy, které ovládají celý program.

Game Třída, která vytvoří GUI, poté kontroléry a propojí jednotlivé části programu. Při spuštění nové hry zajišťuje promazání starých dat.

Controller Kontrolér, který zpracovává vstup z grafického uživatelského rozhraní a od motorů. Pouští motory, pokud jsou na tahu, kontroluje konec hry. Sleduje změnu současné pozice v *Board* a na základě této změny dává pokyn ke změně modelu.

GUI

Jak už název balíku vypovídá jsou zde implementovány veškeré viditelné prvky z grafického uživatelského rozhraní. Mimo to obsahuje také model, jehož data jsou použita k vykreslení pozice na GUI. Tento model je reprezentací šachovnice jakožto 8x8 pole.

5.2 Popis uživatelského rozhraní

Součástí uživatelského rozhraní je okno, ve kterém lze najít dvojici šachovnic včetně zásobníků na pokládání figur. Dále je v pravé části historie tahů. Na každé straně šachovnic jsou jména hrajících oponentů a ukazatel hráče na tahu. Pro lepší orientaci je graficky označen poslední tah. Graficky vyznačené jsou i možné tahy, které lze v dané pozici odehrát. Toto vyznačení není použito pro pokládání figur z „ruky“, ale pouze pro tahy na šachovnici.

V hlavní nabídce lze najít možnost zahájení nové hry a nastavení, kde se dají zvolit hrající hráči. Hra není časově omezená a proto se může při hře dvou šachových motorů stát, že se zastaví a budou čekat na dohrání druhé pozice. Tomuto chování nelze zamezit vzhledem k tomu, že čekání před matem je často používaná strategie. Zabránit mu v tom bylo možné v případném vylepšení, kde by byla hra časově omezená.

Mimo grafické uživatelské rozhraní program při spuštění z příkazové řádky resp. terminálu může vypisovat dodatečné informace o tahu na standardní výstup.

Kapitola 6

Hraní proti jiným programům

Hraní proti jiným programům je nejlepší způsob, jak ohodnotit herní sílu programu.

6.1 Současné programy hrající holanďany

Současné šachové programy jsou implementovány jako programy motoru s vlastním gui, nebo častěji pouze motor, ke kterému se lze připojit jakékoliv uživatelské rozhraní, se kterým komunikuje přes protokol nazvaný Chess Engine Communication Protocol. Pro tento protokol se používají i názvy winboard nebo xboard protokol. Zde je název odvozený od grafického uživatelského rozhraní určené pro Linux resp. Windows, které tento protokol používá. Detailnější popis tohoto protokolu v 6.2

Sjeng
: Uvádí se síla 2000 elo[1][15]. Používá základní komunikaci, kterou posílá přes xboard protokol spoluhráči ("Neztrať figuru", "Hrozba matu za n tahů", "Potřebuji tuto figuru"). Je silný hlavně v týmu s člověkem.

Sunsetter
: Neudává přesnou sílu v holanďanech. Vývoj je soustředěn hlavně na Crazyhouse, kde patří k nejlepším na světě, holanďany, které jsou podobné, jsou spíš vedlejší produkt.

TJchess
: Na svých stránkách[2] uvádí poslední elo na 1900. Vytvořený primárně pro hru dvou motorů společně. Je slabší při hře člověka s počítačem v týmu. Motory spolu na přímo komunikují přes TCP/IP. Posílají si informace o času na hodinách, kdo je na tahu. Bohužel v současné verzi není možné hrát proti němu bughouse variantu.

Vzhledem k tomu, že je hráno bez času, často se stane, že hrají-li proti sobě dva motory, zůstanou čekat několik tahů před matem. Partii pak nelze dohrát. Řešením by bylo přidat čas a tím i představit tomuto motoru umělou inteligenci čekání a omezit pro něj čas přemýšlení.

[1] Vzhledem tomu, že není žádná oficiální elo hodnocení hráčů porovnání je použito z klasického šachu. V rámci České republiky by se zařadil na konec horních 20% hráčů. Celosvětově by byl v 39% nejlepších (zdroj z roku 2013 https://en.chessbase.com/portals/4/files/news/2014/topical/paterek/ratings.png)

[2] http://tonyjh.com/chess/tjchess/bughouse/

6.2 Chess Engine Communication Protokol

V této kapitole je čerpáno z manuálu [12]. Chess Engine Communication Protokol je protokol, jakým mezi sebou komunikují šachové motory a uživatelské rozhraní. Šachové motory čtou a zapisují data na standardní vstup a výstup.

Rozhraním je v tomto případě tento program a motorem je jiný program. Po ustanovení komunikace motor odešle úvodní zprávu. Poté čeká na zprávy od rozhraní.

6.2.1 Zprávy od rozhraní motoru

xboard Rozhraní sděluje motoru, že motor nemá vypisovat šachovnici ani žádné jiné přebytečné zprávy na svůj výstup.

protover 2 Zpráva o verzi protokolu (vybraný 2), jakou chce rozhraní komunikovat. Motor by měl odpovědět zprávou *FEATURE*

variant bughouse Vybrání varianty bughouse, pokud ji motor nepodporuje vrací zprávu *Error (unsupported variant): bughouse*, tato zpráva se zobrazí uživateli a hra nezačne.

partner JMENO Přiřadí motoru spoluhráče.

MOVE Poslání tahu motoru ve formátu pole odkud a pole kam (např. e2e4) nebo v případě položení figury P@e4, pokud motor tah nepovažuje za validní vrací zprávu *Illegal move: MOVE*

holding [bily] [cerny] Informace o držených figurách (např. holding [PPQR] [PBBN]).

go Program je na tahu a má zahrát tah.

6.2.2 Zprávy motoru k rozhraní

FEATURE Vypíše možnosti, které lze v programu nastavit.

move MOVE Motor zahrál tah MOVE. Formát tahu je identický jako formát zprávy tahu od rozhraní motoru.

tellics ptell DETAIL Zpráva určená spoluhráči.

Závěr

Cílem této práce bylo pochopit rozdíly mezi klasickým šachem a variantou Holanďany, zjistit, jak ohodnocovat tahy specifické pro tuto variantu a jaké metody prohledávání stavového prostoru použít, a využít tyto znalosti pro vytvoření programu hrající tuto variantu. Program by měl být schopen hrát proti hráči i proti jiným již existujícím programům.

Tyto cíle byly splněny tak, že jsem z uvedených zdrojů zpracoval informace, navrhl a poté implementoval daný program. Program využívá bitboardů s hyperbola quintessence algoritmem pro ukládání informacích o hře a generování tahů. Pro prohledávání stavového prostoru používá alfa-beta algoritmus s řazením tahů, kaskádovou metodou, dopočtem do tiché pozice a s metodou nulového tahu. K hodnocení pozice používá statické hodnocení pozice figur a jejich základní hodnoty.

Program nekomunikuje mezi šachovnicemi, což je z hlediska šachové síly velká slabina. Tuto vadu by šlo odstranit v případném vylepšení, kde by se do generování tahů zavedli tahy pokládání figury, které by bylo možné zahrát, kdyby spoluhráč tuto figuru získal. Vzhledem k tomu, že v současné verzi by takové generování mělo za následek masivní nárůst časové složitosti, je takové vylepšení ve fázi návrhu. Program nepodporuje hraní na čas a má fixní dobu na odehrání tahu. Tento problém úzce souvisí s předchozím a to tím, že vyvstává otázka, jak dlouho je program ochotný čekat, než spoluhráč danou figuru předá. Vyvstává taktéž otázka, kolik času je program ochotný se věnovat hloubkovému propočtu v současné situaci. Odpovědi na tyto otázky nebyly v práci vyřešeny, tudíž nebyla implementována časová kontrola partie.

Nelze určit přesnou šachovou sílu programu, protože program v konečné fázi nebyl testován z hlediska šachové síly proti lidských soupeřům různé šachové síly. Ve hře proti jiným šachovým programů byl testován proti programům Sjeng a Sunsetter. V porovnání s těmito špičkovými programy nebyl úspěšný.

Literatura

[1] Aziz, A.; Lee, T.; Prakash, A.: *Elements of Programming Interviews in Java: The Insiders' Guide*. CreateSpace Independent Publishing Platform, 2015, ISBN 978-1-5174358-0-6, [Online; navštíveno 24.01.2017].
URL https://books.google.cz/books?id=NDQrjgEACAAJ

[2] Bibel, W.; Heinz, E.; Kruse, R.: *Scalable Search in Computer Chess: Algorithmic Enhancements and Experiments at High Search Depths*. Computational Intelligence, Vieweg+Teubner Verlag, 2013, ISBN 9783322901781.

[3] David-Tabibi, O.; Netanyahu, N. S.: Verified null-move pruning. *ICGA journal*, ročník 25, č. 3, 2002: s. 153—-161.

[4] Droste, S.; Fürnkranz, J.: Learning the Piece Values for Three Chess Variants. *ICGA Journal*, ročník 31, č. 4, 2008: s. 209–233.

[5] Knuth, D. E.; Moore, R. W.: An analysis of alpha-beta pruning. *Artificial intelligence*, ročník 6, č. 4, 1975: s. 293–326.

[6] Korf, R. E.: Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, ročník 27, č. 1, 1985: s. 97–109.

[7] Lefler, M.: Chess Programming wiki. [Online; navštíveno 12.05.2017].
URL chessprogramming.wikispaces.com/Bitscan

[8] Lefler, M.: Chess Programming wiki. [Online; navštíveno 24.01.2017].
URL chessprogramming.wikispaces.com/Java

[9] Lefler, M.: Chess Programming wiki. [Online; navštíveno 02.05.2017].
URL chessprogramming.wikispaces.com/Alpha-Beta

[10] Lefler, M.: Chess Programming wiki. [Online; navštíveno 08.05.2017].
URL chessprogramming.wikispaces.com/Node+Types

[11] Lorenz, U.; Tscheuschner, T.: Player modeling, search algorithms and strategies in multi-player games. In *Advances in Computer Games*, Springer, 2005, s. 210–224.

[12] Mann, T.; Miller, H.: Chess Engine Communication Protocol. [Online; navštíveno 05.05.2017].
URL http://www.open-aurec.com/wbforum/WinBoard/engine-intf.html

[13] Marsland, T. A.: A review of game-tree pruning. *ICCA journal*, ročník 9, č. 1, 1986: s. 3–19.

[14] Moreton, C.: Rival Chess - Magic Bitboards. [Online; navštíveno 11.05.2017].
URL http://www.rivalchess.com/magic-bitboards/

[15] Pascutto, G. C.: Sjeng. [Online; navštíveno 08.05.2017].
URL sjeng.org/indexold.html

[16] Pritchard, D.: *The Classified Encyclopedia of Chess Variants*. John Beasley, 2007, ISBN 978-0-9555168-0-1.

[17] Šachový svaz České republiky: *Pravidla Bughouse šachu*. [Online; navštíveno 08.01.2017].
URL www.chess.cz/www/informace/legislativa/zavazne-dokumenty/bughouse.html

[18] Zbořil, F. V.: Přednášky IZU, VUT FIT. [Online; navštíveno 02.02.2017].
URL www.fit.vutbr.cz/study/courses/IZU/private/1617izu_5.pdf

Přílohy

Příloha A

Obsah CD

Přiložené CD obsahuje:

- Technickou dokumentaci - Zdrojový text v jazyce LaTeXa výsledná dokumentace ve formátu PDF.
- Aplikace – Zdrojové kódy napsané v jazyce Java a spustitelný soubor ve formátu JAR.
- Pomocné soubory - obrázky potřebné pro správnou funkci programu.
- ReadMe - s odkazem ke stažení jiných šachových programů a postupem jak zprovoznit hru proti nim.