

České vysoké učení technické v Praze
Fakulta elektrotechnická



Bakalářská práce

Alfa-beta algoritmus pro umělou inteligenci hry šachy

Josef Suchý

Vedoucí práce: Ing. Adam Sporka

Studijní program: Elektrotechnika a informatika strukturovaný bakalářský

Obor: Informatika a výpočetní technika

Červen 2006

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Josef Suchý**

Studijní program: Elektrotechnika a informatika, strukturovaný, bakalářský

Obor: Výpočetní technika


Název: **Alfa-beta algoritmus pro umělou inteligenci hry šachy**

Pokyny pro vypracování:

Prostudujte princip alfa-beta algoritmu. Pomocí tohoto algoritmu implementujte umělou inteligenci počítače pro hru šachy proti uživateli. Grafické prostředí hry realizujte v knihovně Simple DirectMedia Layer. Rozsah implementace konzultujte s vedoucím práce.

Vedoucí: Ing. Adam Sporka

Platnost zadání: únor 2007


doc. RNDr. Josef Kolář, CSc.
vedoucí katedry počítačů




doc. Ing. Karel Müller, CSc.
proděkan

V Praze dne 23. 2. 2006

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v příloženém seznamu.

Nemám žádný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 29.6.2006

.....

Abstract

There are many computer programs implementing the game of chess with various level of the artificial intelligence. The purpose of this work was to study the techniques of the artificial intelligence of the game of chess and make an own implementation of the game with artificial intelligence. The result is a program with graphics interface and an artificial intelligence at an intermediate level.

Abstrakt

Existuje mnoho počítačových programů pro hru šachy, které mají různou umělou inteligenci. Cílem této práce bylo prostudovat způsoby řešení umělé inteligence pro hru šachy a implementovat šachový program s umělou inteligencí pro tuto hru. Výsledkem je šachový program s grafickým uživatelským rozhraním a umělou inteligencí na úrovni středně zkušeného hráče.

Obsah

	Seznam obrázků	xi
	Seznam tabulek	xiii
1	Úvod	1
2	Studie existujících programů a stanovení požadavků	3
2.1	Existující programy	3
2.2	Požadavky na program	4
3	Definice pojmů a používané algoritmy	5
3.1	Naivní algoritmus	5
3.2	Algoritmus MiniMax	7
3.3	Algoritmus alfa-beta	9
3.4	Redukční algoritmus	11
4	Realizace	14
4.1	Použité vývojové prostředí	14
4.2	Reprezentace šachovnice	15
4.3	Potřebné funkce	18
4.4	Návrh grafického rozhraní	21
4.5	Načítání a ukládání hry	21
5	Zkoušení a testování programu	24
5.1	Ladění	24
5.2	Testování	26
6	Závěr	28
7	Seznam literatury	29
A	Uživatelská příručka	31
B	Obsah přiloženého CD	39

Seznam obrázků

Obr. 3.1.1	Strom hry pro naivní algoritmus	6
Obr. 3.2.1	Strom hry algoritmu MiniMax	8
Obr. 3.3.1	Strom hry pro alfa-beta algoritmus	10
Obr. 3.4.1	Příklad nalezených obrazců převzato z [5]	13
Obr. 4.5.1	Příklad přehrání hry	23
Obr. 5.1.1	Příklad pozice pro ladění	24
Obr. A.1	Hrací plocha	32
Obr. A.2	Pohyb pěšce	33
Obr. A.3	Pohyb věže	34
Obr. A.4	Pohyb jezdce	34
Obr. A.5	Pohyb střelce	35
Obr. A.6	Pohyb dámy	35
Obr. A.7	Pohyb krále	36
Obr. A.8	Malá rošáda	36
Obr. A.9	Velká rošáda	36
Obr. B.1	Obsah přiloženého CD	39

Seznam tabulek

Tab. 3.4.1	Parametry nalezených obrazců z Obr. 3.4.1	12
Tab. 4.2.1	Reprezentace šachovnice maticí 8x8	15
Tab. 4.2.2	Reprezentace šachovnice maticí 10x12	16
Tab. 4.2.3	Znázornění indexace v matici 16x8	17
Tab. 4.2.4	Reprezentace šachovnice maticí 16x8	17
Tab. 4.3.1	Poziční hodnoty polí	19
Tab. 4.5.1	Příklad formátu souboru	23
Tab. 5.2.1	Výsledky testování	27

1. Úvod

Již dlouhou dobu se lidé snaží vytvořit stroj, který by uměl hrát šachy. Ale nejen, aby zvládal pravidla a náhodně přemísťoval šachové kameny, ale aby byl schopen vyhrát a měl nějakou inteligenci. Tato inteligence by měla být dostatečná k tomu, aby dokázal porazit šachové mistry. Šachy jsou považovány za královskou hru a jsou výzvou pro počítačové programátory.

Dějiny šachových strojů však nezačaly až ve dvacátém století. První pokus o počítačový stroj se objevil už v osmnáctém století. Nebyl to šachový stroj v pravém slova smyslu, ale vzbudil mimořádný dojem.

Vůbec za první šachový stroj byl považován Kempelenův Turek. Což byl stroj, který měl podobu orientálně oblečeného muže sedícího před skříňkou. Tento stroj dokázal v naprosté většině her zvítězit a málokdy se stalo, že by někdo dokázal stroj porazit. V té době lidé věřili, že je stroj schopen hrát šachy a že má vlastní inteligenci. Tajemství stroje nebylo nikdy zveřejněno. Postupem času bylo zjištěno, že stroj neměl vlastní inteligenci, ale že byl ovládán nejlepšimi šachovými mistry Evropy.

Dalším pokusem o sestrojení šachového stroje byl Babageův analytický stroj, který měl původně počítat algoritmické tabulky. Řízení tohoto stroje mělo probíhat pomocí děrných štítků. Tento projekt se nezdařil, protože byl na tehdejší dobu velice rozsáhlý a nerealizovatelný. Ale některé vymyšlené strategické otázky, které si měl stroj pokládat v každé pozici, se používají dodnes.

První skutečný šachový stroj byl Torresův automat. Nebyl to automat, který by dokázal sehrát celou šachovou hru, ale dokázal vyřešit jednoduchou šachovou koncovku.

Vůbec první program, který vznikl vymyslel Alan Turing. Tento program přesně popisoval po sobě jdoucí instrukce, které se mají provádět. Ale v té době neexistoval počítač, který by tento algoritmus dokázal realizovat a tak prováděl Turing své výpočty ručně.

V dnešní době už jsou počítače schopné prozkoumat miliony tahů dopředu a také jsou mnohem lepší algoritmy, které dokáží lépe posoudit výhodnost tahu. I když jsou už dnes vymyšlené programy na velice vyspělé úrovni stále se nachází mnoho lidí, kteří se snaží vymyslet program, který by uměl hrát šachy.

Proč se vlastně šachy těší takové oblibě u programátorů? Odpovědí je hned několik.

- Šachy mají ideální předpoklady, protože je zde dána pevná množina objektů (pole, kameny, tahy), mezi nimiž jsou pevně definované vztahy
- Šachy jsou z pohledu programátora dostatečně obtížné
- Pro úspěch nebo neúspěch existují jednoduchá kritéria

Přínosem práce je seznámení se šachovými počítačovými programy, které existují. Prozkoumání některých postupů a algoritmů, které lze při realizaci umělé inteligence počítače použít. Navrhnutí vlastního grafického rozhraní a potřebných funkcí, aby program zvládal pravidla hry šachy a využití některých algoritmů pro realizaci umělé inteligence počítače.

2. Studie existujících programů a stanovení požadavků

Pro stanovení požadavků vlastností a grafického rozhraní jsem se zabýval už existujícími programy a jejich vlastnostmi jsem navrhl požadavky.

2.1 Existující programy

Dnešní počítačové programy mají umělou inteligenci založenou na hrubé síle. To znamená, že každý počítačový program zkouší zahrát většinu možných pozic a takto zahrané pozice se snaží podle nějakých kritérií ohodnotit.

Současné špičkové programy hrají na velmistrovské úrovni a některé poráží i světové hráče. Světovou jedničkou je program Shredder 8 [1], tento program už porazil řadu světových mistrů. Za ním následuje program Deep Fritz 8 [2], ten nemá takovou inteligenci jako Shredder, ale stejně dokázal porazit řadu mezinárodních mistrů. Toto jsou však komerční programy, které je nutno si zakoupit.

Kromě těchto komerčních programů existuje po dlouhou dobu řada amatérských šachových programů. Jejich grafické možnosti a uživatelské přednosti nedosahují takových kvalit jako u komerčních programů, ale jejich úroveň hry porazí i mezinárodní hráče v šachu.

Amatérské programy se původně vyvíjely stejným způsobem jako komerční programy. To znamená, že autor kromě umělé inteligence musel vytvořit i grafiku (šachovnici, figurky, ovládání) a mnoho dalších věcí.

Postupem času se objevil systém Winboard [3], který už v sobě obsahoval základní prvky, jako je šachovnice, figurky, ovládání a další funkce mimo umělé inteligence. Tento systém zjednodušoval práci, protože autor nemusel vytvářet grafické prostředí a potřebné funkce a mohl se soustředit na vývoj samotné umělé inteligence. Nevýhodou takto vytvořených programů byla jejich podobnost a originalita spočívala jen v umělé inteligenci.

Nakonec vznikla tzv. Aréna [4], která má jednu velkou výhodu a tou je, že se dá nastavit jakou dobu se má partie hrát a algoritmus se tomu musí přizpůsobit.

2.2 Požadavky na program

Pro realizaci programu je zapotřebí navrhnout vlastní grafické rozhraní a další funkce potřebné k realizaci. Od programu se požaduje, aby se dal ovládat pomocí myši. Dále, aby byla zobrazena historie tahů, ve které se dá listovat, a zobrazení vyhozených kamenů obou soupeřů. Také je potřeba, aby se dala hra uložit a načíst ze souboru a umět tuto načtenou hru přehrát. Nutnou podmínkou je umělá inteligence. Doba po jakou hraje hráč je z hlediska použití programu zbytečné a proto tato informace nebude zahrnuta v řešení. Také je potřeba, aby se kameny pohybovaly jen podle šachových pravidel a tuto skutečnost pro aktuálně vybraný kámen zobrazit.

3. Definice pojmů a používané algoritmy

Modelem šachovnice je pole 8x8. Za hrací kámen se považuje figurka, která má přesně definované vlastnosti (např. pohyb). Každému typu hracího kamene je pro reprezentaci v poli šachovnice přiděleno jednoznačné číslo, které reprezentuje daný kámen. Pozicí je myšleno konkrétní umístění kamenů na šachovnici. Změnou pozice je myšleno posun kamene z jednoho místa na jiné. Za tah je považován pohyb, který provedou po sobě oba hráči. Za půltah se pak považuje posun kamene jen jednoho hráče.

3.1 Naivní algoritmus

Tento algoritmus je velice jednoduchý a také nevykazuje velkou inteligenci popisuje ho např. [5]. Spočívá jen v zahrání a ohodnocení těch tahů, které lze ze současné pozice zahrát.

Pseudokód:

```
int NejlepsiHodnotaPozice(){ /* Funkce vrátí nejlepší ohodnocení
                             pozice*/

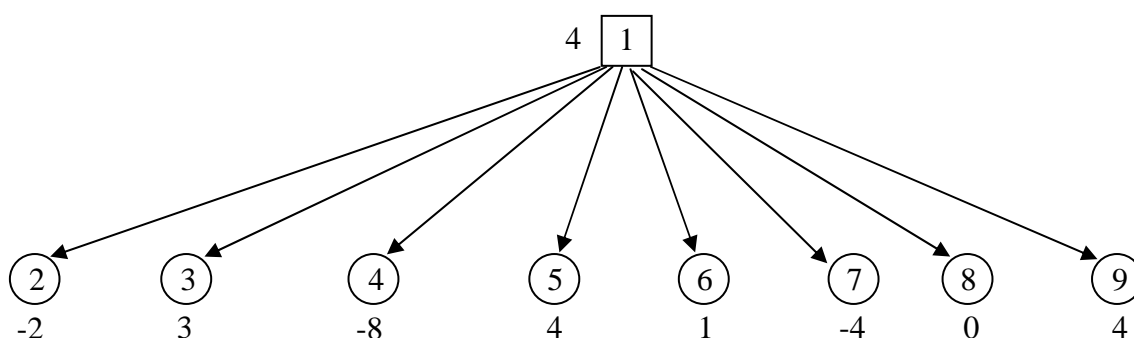
    tahy t;                  /*Proměnná obsahující možné tahy */
    int i, cena, pom;         /*Číselné proměnné*/

    t=GenerujTahy();          /*Generuje všechny tahy, které lze
                             zahrát */

    cena=-nekonecno;

    /* Pamatuji si jak je dobrý tah.*/
    /*Cyklus přes všechny tahy*/
    for(i=0; i<t.pocet;i++){ /*Zahraji tah*/
        Tahni();
        pom=-HodnotaPozice(); /*Zjistím jak je dobrý*/
        if(pom>cena) cena=pom; /*Je-li prozatím nejlepší, zapamatuji
                             si ho*/
        TahniZpet();          /*Zahraji tah zpět*/
    }                          /*konec for cyklu přes tahy*/
    return cena;              /*Vrátím nejlepší hodnotu*/
}
```

Pro znázornění výpočtu a lepšího pochopení budeme uvažovat graf, kde pozice jsou vrcholy a lze-li táhnout z jedné pozice do druhé pak tyto dva vrcholy spojíme orientovanou hranou. Čtverečkem jsou označeny vrcholy, které znázorňují možné tahy protihráče. Vrchol 1 označuje již provedený tah protihráče a zkoumá se jaká je nejlepší pozice pro hráče na tahu.



Obr. 3.1.1 Strom hry pro naivní algoritmus

Výpočet probíhá následovně. Výchozí pozice zde 1 se vygenerují tahy, které lze provést a dostat se do pozic 2 až 9. Algoritmus pak zkouší zahrát jednotlivé tahy a zjišťuje, který z nich je nevýhodnější. Ze všech možných tahů se hledá maximum. Výchozí pozice se přejde do pozice 2 a zjistí se výhodnost této pozice. Pak se vrátí do výchozí pozice a zahraje tah, který vede na pozici 3. Jestliže tato pozice je výhodnější zapamatuje si toto číslo. Takto postupuje přes všechny tahy. Nakonec vrátí hodnotu nejlepší pozice. Příklad stromu hry ukazuje obr. 3.1.1.

3.2 Algoritmus MiniMax

Tento algoritmus není moc složitý. Podobá se naivnímu algoritmu, ale jeho hloubka prohledávání není jedna, ale lze ji stanovit a tím zlepšit vlastnosti určení nejlepší pozice. Je popsán v [5].

Pseudokód:

```
int MiniMax(int hloubka){                                /*Funkce vrátí odhad pozice -
                                                         výsledek propočtu do zadané
                                                         hloubky*/
    tahy t;                                              /*Obsahující pole tahů */
    int i, pom;

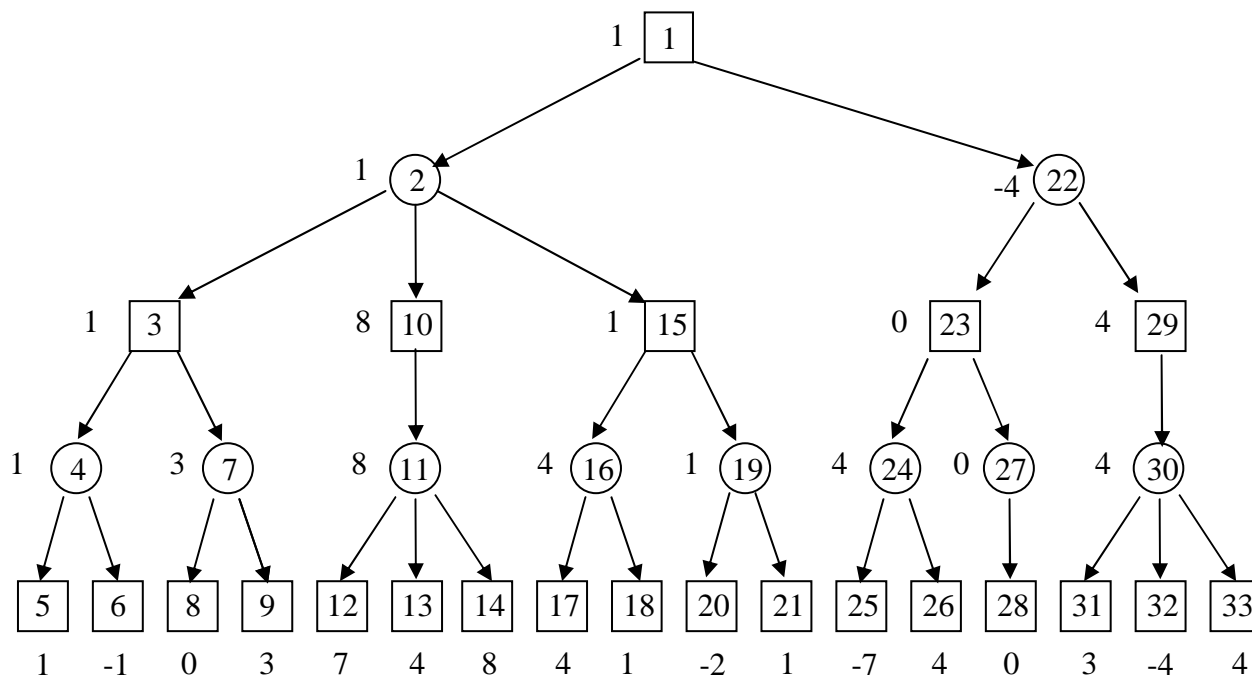
    if(KoncovaPozice()) return MatRemis(); /* Funkce MatRemis vrací 0,
                                             pokud nastává v pozici remíza
                                             a -mnoho,
                                             pokud jsme v matu*/
    if(hloubka==0) return HodnotaPozice(); /* Když jsem na hloubce 0,
                                             končím propočet a vracím
                                             odhad.*/
    t=GenerujTahy();                                     /*Naleznu všechny tahy z
                                                         pozice.*/

    for(i=0; i<t.pocet;i++){                             /*Cyklus přes všechny tahy*/
        Tahni();                                         /*Zahraji tah*/
        pom=-MiniMax(hloubka-1);

        /*Zjistím, jak je tah dobrý propočtem do hloubky o jedničku menší.
        Musím ale návratovou hodnotu vynásobit -1, neboť funkce MiniMax
        posuzovala pozici z hlediska soupeře.*/

        TahniZpet();                                    /* Zahraji tah zpět*/
        if(pom>cena)cena=pom;                             /*Pamatuji si dosud nejlepší
                                                         cenu*/
        TahniZpet();                                    /* Zahraji tah zpět*/
    }                                                    /*konec for cyklu přes tahy*/
    return cena;                                         /*Vvrátím cenu nejlepší
                                                         pozice*/
}
```

Příkladem může být graf na obrázku 3.2.1. Uvažujeme stejnou interpretaci stromu hry jako v kapitole 3.1. Z počáteční pozice 1 lze táhnout do pozice 2 a 22. Z pozice 2 pak může soupeř jít do pozice 3, 10, 15 atd.. Vrcholy na stejné úrovni symbolizují pozice, do kterých se dostaneme po stejném počtu tahů. Výchozí pozici rozvineme do požadované hloubky.



Obr. 3.2.1 Strom hry algoritmu MiniMax

Výpočet probíhá následovně. Generátor možných tahů vygeneruje všechny tahy z aktuální pozice a provede v paměti první z nich. Po přesunu do příslušné pozice (zde 2) počítač vygeneruje možná pokračování soupeře, zase vybere tah z vygenerovaných pro soupeře a v paměti ho provede (zde 3), dále pokračuje zjištěním všech možných svých tahů a pak zase soupeřových a takto pokračuje až se dostane do zadané hloubky (zde pozice 5). Pro tuto pozici se spočítá její výhodnost pro hráče na tahu. Dále vrátí tah zpět a zkusí se zahrát další možný tah (zde 6), opět se pro tento tah zjistí jeho výhodnost. Po prohledání všech možných variant z pozice 4 se určí hodnota této pozice. Pokud bude v této pozici táhnout hráč na tahu, bude hodnota maximum z hodnot pozic do kterých odtud táhnul, bude-li to soupeř tak to bude minimum. Po vytvoření celého stromu hry bude počítač táhnout do pozice s největším ohodnocením. Protože se na jednotlivých hladinách pravidelně střídá minimum s maximumem, nazývá se tento algoritmus MiniMax.

3.3 Algoritmus alfa-beta

Alfa-beta algoritmus vychází z algoritmu MiniMax. Tomuto algoritmu se přidají dva parametry alfa a beta, které udávají interval prohledávání. Vše mimo interval se prohledávat nebude, protože i kdyby se prohledávalo mimo interval bylo by to zbytečné, protože již je nalezená lepší cesta. Je popsán v [5].

Pseudokód:

```
int AlfaBeta(int hloubka, int alfa, int beta){
    /*Funkce vrátí odhad pozice -
    výsledek propočtu do zadané
    hloubky*/
    tahy t;
    int i, pom;

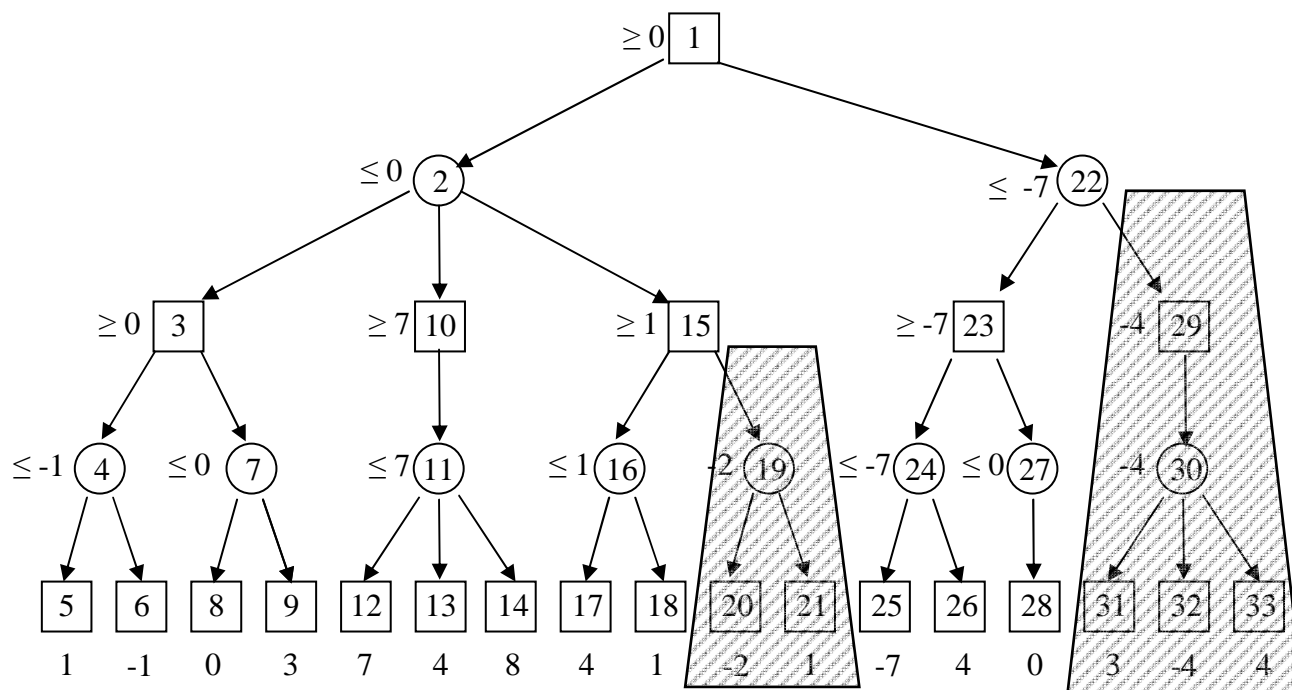
    if(KoncovaPozice()) return MatRemis(); /* Funkce MatRemis
    vrací 0, pokud nastává v
    pozici remíza a -mnoho,
    pokud jsme v matu*/
    if(hloubka==0) return HodnotaPozice(); /* Když jsem na
    hloubce 0, končím propočet a
    vracím odhad.*/
    t=GenerujTahy(); /*Naleznu všechny tahy z
    pozice.*/

    for(i=0; i<t.pocet;i++){ /*Cyklus přes všechny tahy*/
        Tahni(); /*Zahraji tah*/
        pom=-MiniMax(hloubka-1, -alfa, -beta);

        /*Zjistím, jak je tah dobrý propočtem do hloubky o jedničku menší.
        Musím ale návratovou hodnotu vynásobit -1, neboť funkce AlfaBeta
        posuzovala pozici z hlediska soupeře, změní se i intervaly na
        opačné*/

        TahniZpet(); /* Zahraji tah zpět*/
        if(pom>alfa){
            alfa=pom; /*Pamatuji si dosud nejlepší
            cenu*/
            if(pom>=beta) return beta; /*Zde je právě ta úspor. */
        }
    } /*konec for cyklu přes tahy*/
    return alfa; /*Vvrátím cenu nejlepší
    pozice*/
}
```

Příkladem může být graf na obrázku 3.3.1. Uvažujeme stejnou interpretaci stromu hry jako v kapitole 3.1.



Obr. 3.3.1 Strom hry pro alfa-beta algoritmus

Výpočet probíhá následovně. Jako u algoritmu MiniMax se generují možné tahy a ty se zkouší zahrát v paměti. Až se program dostane do pozice 5 zjistí jeho hodnotu a stanoví se horní mez na ≤ 1 , pak se zjistí hodnota pozice 6. Dále se vyhodnotí hodnota pozice 4, kde se zjistí, že je podmínka splněna a tak se tato hodnota změní na ≤ -1 . Tímto jsme prohledali podstromy pozice 4 a dostáváme se výše, kde se změní horní mez na dolní a dostáváme ≥ -1 . Dále algoritmus postupuje stejným způsobem až se dostane do pozice 15. Prohledá levý podstrom, ze kterého získá dolní mez 1. Jelikož se intervaly z pozice 15 a z pozice 2 neprotínají, neexistuje lepší tah a prohledávání zbylých podstromů je zbytečné. To samé se provede u pozice 22.

Je vidět, že pokud by měla pozice 10 více podstromů, tak by byly také odříznuty, protože se zde zase intervaly nepřekrývají.

3.4 Redukční algoritmus

Redukční algoritmus řeší úlohy metodou vyhledávání matových obrazců jak to popisuje [6]. Při optimalizaci se zjišťuje jestli tyto obrazce mohou nastat a tímto získáváme jen některé obrazce. Další fází je zkoumání jestli se dá nalézt konkrétní cesta z počáteční pozice do těchto obrazců. V případě úspěšnosti je nalezeno řešení.

Redukční algoritmus hledá obrazce na základě masek příslušejících jednotlivým typům kamenů. Obrazec odpovídá skupině kamenů, kdy jde o šach a všechna pole kolem černého krále jsou kryta bílými kameny nebo blokována černými kameny. Z celkového počtu bílých a černých kamenů v pozici úlohy jsou v konkrétním obrazci obsaženy jen některé, které se obrazce aktivně účastní. Tyto kameny nazýváme zúčastněné kameny. Zbývající kameny, které se na obrazci nepodílejí, nazýváme nezúčastněné kameny.

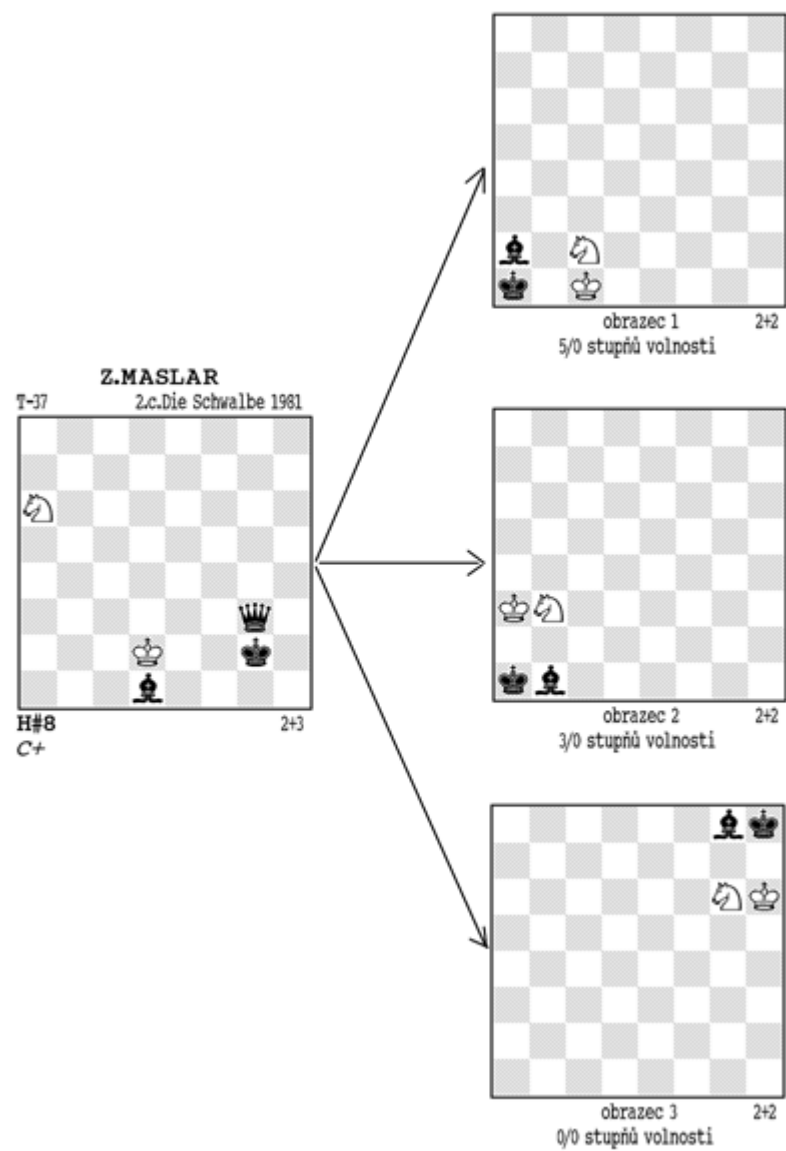
Na základě tabulek dostupnosti polí se provede statický odhad počtu tahů bílého a černého pro dosažení obrazce. Obrazce, kterých není možné dosáhnout v určitém počtu tahů jsou vyloučeny. Celkový počet tahů závisí na obtížnosti úlohy. Počet stupňů volnosti se získá odečtením od počtu tahů počet nutných tahů k dosažení obrazce.

Pro každý obrazec se nejprve zjistí násobnost šachu. Bodový šach může být maximálně jeden. Pokud je počet šachů větší nebo roven dvěma, nelze většinu optimalizací použít. Nemusí jít jen o dvojité šachy. Jelikož jádro redukčního algoritmu “nevidí přes linie”, může být tento počet šachů i větší než 2, ale v tomto případě může jít o vazby černých kamenů. V případě jednoduchých šachů vytvořím seznam kritických polí. Kritická pole jsou pole matujícího kamene a pole na matující líní.

Ve fázi optimalizace se prozkoumává soubor podmínek, které tomu mohou bránit. Tím mohou být bílé zúčastněné kameny, stojící na matové linii, černé zúčastněné kameny, které mohou bodově vstoupit do matové linie nebo na matové pole, atd. Pokud obrazec projde optimalizací, hledá se dynamicky cesta z počáteční pozice do nalezeného obrazce. V této fázi sice už dochází k provádění konkrétních tahů. Jednotlivé tahy však mohou snižovat aktuální počet stupňů volnosti a pokud jsou takto dynamicky tahy některé strany již vyčerpány, všechny dále následující varianty od této úrovně už nejsou uvažovány. Ke zvýšení těchto eliminací přispívá i dynamické volání některých algoritmů (algoritmus pěšců, rozpojovací algoritmus apod. více [6]). Příkladem výstupu programu, který používá tento algoritmus je tabulka 3.4.1. Pro lepší představu je vše vidět na obrázku 3.4.1.

Kd2	Ja6	Kg2	Dg3	Sd1	nutných tahů bílého	nutných tahů černého	stupňů volnosti bílého	stupňů volnosti černého
c1	c2	a1	nezúčast.	a2	3	8	5	0
1	2	6	-	2				
a3	b3	a1	nezúčast.	b1	5	8	3	0
3	2	6	-	2				
h6	g6	h8	nezúčast.	g8	8	8	0	0
4	4	6	-	2				

Tab. 3.4.1 Parametry nalezených obrazců z Obr. 3.4.1



Obr. 3.4.1 Příklad nalezených obrazců převzato z [6]

4. Realizace

4.1 Použité vývojové prostředí

Pro projekt jsem si vybral vývojové prostředí Dev C++ [7]. Tento program už používám několik let a jsem s ním plně spokojen. Pro malé a střední projekty je ideálním řešením a jeho hlavní výhodou je, že je to freewareový program. Nevýhodou tohoto programu je, že neobsahuje veškeré podpůrné prvky jako komerční vývojová prostředí, ale tyto nedostatky se řeší nainstalováním potřebného balíčku, kterých dnes existuje nespočet.

Pro mojí aplikaci bylo nutné doinstalovat balíček, který podporuje práci s SDL knihovnou a doinstalovat také podporu písma ke knihovně SDL. Oba tyto balíčky lze stáhnout z domovských stránek Dev C++. Toto vývojové prostředí je velice jednoduché na ovládání a není složité se tímto softwarem naučit pracovat na rozdíl od komerčních vývojových prostředí.

Jako programovací jazyk jsem zvolil C++, protože je zatím nejrozšířenějším programovacím jazykem a je podporován i vývojovým prostředím, které jsem použil. Pro lepší orientaci a v tomto jazyce při programování jsem použil knižní literaturu [8] a [9].

4.2 Reprezentace šachovnice

Pro reprezentaci šachovnice se nabízí několik možností např. maticí 8x8 ukazuje tabulka 4.2.1, maticí 10x12 ukazuje tabulka 4.2.2 a nebo maticí 16x8 s kódováním 0x88 znázorňující tabulka 4.2.3.

Výběr reprezentace šachovnice je důležitý, protože s touto reprezentací se stále pracuje, a proto je důležité, aby operace testování byly rychlé. Dále je také dobré zohlednit jak dobře se s reprezentací pracuje, aby náhodou na úkor rychlosti nedocházelo ke špatné adresaci šachovnice a tudíž k chybám v programu.

Výhodou reprezentace maticí 8x8 je intuitivní určení souřadnic na šachovnici. Nevýhodou je potřeba testování čtyř okrajových podmínek a to zpomaluje program.

	0	1	2	3	4	5	6	7
0	A8	B8	C8	D8	E8	F8	G8	H8
1	A7	B7	C7	D7	E7	F7	G7	H7
2	A6	B6	C6	D6	E6	F6	G6	H6
3	A5	B5	C5	D5	E5	F5	G5	H5
4	A4	B4	C4	D4	E4	F4	G4	H4
5	A3	B3	C3	D3	E3	F3	G3	H3
6	A2	B2	C2	D2	E2	F2	G2	H2
7	A1	B1	C1	D1	E1	F1	G1	H1

Tab. 4.2.1 Reprezentace šachovnice maticí 8x8

Reprezentace maticí 10x12 využívá toho, že jsou přidány okraje, ve kterých je nějaká hodnota (zde N), která určuje, že už jsme mimo šachovnici. Zde stačí testovat jen jednu podmínku a proto tato realizace zrychluje provádění výpočtu oproti matici 8x8. Nevýhodou je, že souřadnice pole neodpovídají umístění na šachovnici.

Jak je vidět v tabulce 4.2.2 na okrajích je jen jedna hodnota, která udává, že jsme mimo pole. Tak že by bylo logické použít pole 12x12, protože když si vezmeme pohyb jezdce dostáváme, že se může pohybovat až o dvě pole. Vše je úplně v pořádku a stačí reprezentace 10x12, protože pole bývá většinou realizováno lineárním polem a tak lze říci, že sloupec nula a sloupec devět v tabulce 4.2.2 jsou vedle sebe. Takto získáme dvě hodnoty, vedle sebe a i pro jezdce se dá poznat, že je mimo šachovnici.

	0	1	2	3	4	5	6	7	8	9
0	N	N	N	N	N	N	N	N	N	N
1	N	N	N	N	N	N	N	N	N	N
2	N	A8	B8	C8	D8	E8	F8	G8	H8	N
3	N	A7	B7	C7	D7	E7	F7	G7	H7	N
4	N	A6	B6	C6	D6	E6	F6	G6	H6	N
5	N	A5	B5	C5	D5	E5	F5	G5	H5	N
6	N	A4	B4	C4	D4	E4	F4	G4	H4	N
7	N	A3	B3	C3	D3	E3	F3	G3	H3	N
8	N	A2	B2	C2	D2	E2	F2	G2	H2	N
9	N	A1	B1	C1	D1	E1	F1	G1	H1	N
10	N	N	N	N	N	N	N	N	N	N
11	N	N	N	N	N	N	N	N	N	N

Tab. 4.2.2 Reprezentace šachovnice maticí 10x12

Pro reprezentaci maticí 16x8 se využívá vhodného rozmístění čísla sloupce a čísla řádku v jednotlivých bitech indexové proměnné tab. 4.2.3. V bitech 7 a 3 musí být vždy nula. Pokud v těchto bitech nula není pak jsme mimo pole.

Bit	7	6	5	4	3	2	1	0
Obsah	0	číslo řádku			0	číslo sloupce		

Tab. 4.2.3 Znázornění indexace v matici 16x8

Proto se využívá k realizaci pole 16x8, které je znázorněno v tabulce 4.2.4. Pole označená jako X se vůbec nepoužívají a slouží jen jako výplň, aby vzniklo pole s požadovanými vlastnostmi.

Byte	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	A8	B8	C8	D8	E8	F8	G8	H8	X	X	X	X	X	X	X	X
0x10	A7	B7	C7	D7	E7	F7	G7	H7	X	X	X	X	X	X	X	X
0x20	A6	B6	C6	D6	E6	F6	G6	H6	X	X	X	X	X	X	X	X
0x30	A5	B5	C5	D5	E5	F5	G5	H5	X	X	X	X	X	X	X	X
0x40	A4	B4	C4	D4	E4	F4	G4	H4	X	X	X	X	X	X	X	X
0x50	A3	B3	C3	D3	E3	F3	G3	H3	X	X	X	X	X	X	X	X
0x60	A2	B2	C2	D2	E2	F2	G2	H2	X	X	X	X	X	X	X	X
0x70	A1	B1	C1	D1	E1	F1	G1	H1	X	X	X	X	X	X	X	X

Tab. 4.2.4 Reprezentace šachovnice maticí 16x8

Zvolil jsem reprezentaci maticí 8x8, protože je hned na první pohled vidět, s jakým polem se pracuje. Tuto volbu pro reprezentaci šachovnice jsem provedl i s ohledem na zpomalení několika operací, protože se často testuje jestli je šachový kámen mimo hrací pole a zde se testují čtyři okrajové podmínky.

4.3 Potřebné funkce

Bylo zapotřebí implementovat základní funkce:

- určení konce hry, což jsou funkce, které umí detekovat mat a pat
- určení jestli je šach
- funkce udávající pohyb šachových kamenů podle pravidel
- generátor tahů
- algoritmus alfa-beta
- ohodnocovací funkci.

Pro ohodnocovací funkci jsem stanovil materiální hodnoty kamenů takto:

- | | |
|-----------|-----------------------------------|
| • pěšec | 1.0 |
| • jezdec | 3.0 |
| • střelec | 3.0 |
| • věž | 4.5 |
| • dáma | 9.0 |
| • král | 1000.0 (aby, nemohl být obětován) |

Z důvodu, že pro počáteční pozice kamenů vrací ohodnocování funkce velice podobné až stejné hodnoty zavedl jsem pole, které udává poziční výhodu na šachovnici tabulka 4.3.1. Pro každé pole je stanovena hodnota, která udává výhodnost umístění kamene na šachovnici.

	A	B	C	D	E	F	G	H
8	0.00	0.01	0.02	0.03	0.03	0.02	0.01	0.00
7	0.01	0.04	0.04	0.04	0.04	0.04	0.04	0.01
6	0.02	0.04	0.06	0.06	0.06	0.06	0.04	0.02
5	0.03	0.04	0.06	0.08	0.08	0.06	0.04	0.03
4	0.03	0.04	0.06	0.08	0.08	0.06	0.04	0.03
3	0.02	0.04	0.06	0.06	0.06	0.06	0.04	0.02
2	0.01	0.04	0.04	0.04	0.04	0.04	0.04	0.01
1	0.00	0.01	0.02	0.03	0.03	0.02	0.01	0.00

Tab. 4.3.1 Poziční hodnoty polí

Pro zlepšení vlastností ohodnocování funkce jsem zvolil alfa-beta algoritmus, který zkouší zahrát tahy do hloubky 3 půltahů. Tato hloubka je dostačující pro středně vyspělou inteligenci. Tuto hloubku jsem zvolil také z toho důvodu, aby se dal počítač porazit a tento program neodrazoval uživatele s méně zkušenostmi s hrou šachy.

Pseudokód funkce alfa-beta:

```
int AlfaBeta(int hloubka, int alfa, int beta){
    /*Funkce vrátí odhad pozice -
    výsledek propočtu do zadané
    hloubky*/
    tahy t;
    int i, pom;

    if(KoncovaPozice()) return MatRemis(); /* Funkce MatRemis vrací 0,
    pokud nastává v pozici remíza
    a -mnoho,
    pokud jsme v matu*/
    if(hloubka==0) return HodnotaPozice(); /* Když jsem na
    hloubce 0, končím propočet a
    vracím odhad.*/
    t=GenerujTahy(); /*Naleznu všechny tahy z
    pozice.*/

    for(i=0; i<t.pocet;i++){ /*Cyklus přes všechny tahy*/
        Tahni(); /*Zahraji tah*/
        pom=-MiniMax(hloubka-1, -alfa, -beta);

        /*Zjistím, jak je tah dobrý propočtem do hloubky o jedničku menší.
        Musím ale návratovou hodnotu vynásobit -1, neboť funkce AlfaBeta
        posuzovala pozici z hlediska soupeře, změní se i intervaly na
        opačné*/

        TahniZpet(); /* Zahraji tah zpět*/
        if(pom>alfa){
            alfa=pom; /*Pamatuji si dosud nejlepší
            cenu*/
            if(pom>=beta) return beta; /*Zde je právě ta úspor. */
        }
    } /*konec for cyklu přes tahy*/
    return alfa; /*Vvrátím cenu nejlepší
    pozice*/
}
```

4.4 Návrh grafického rozhraní

Pro návrh jsem se zvolil okno velikosti 800x600 pixelů. Šachovnici jsem umístil k hornímu okraji, aby dole zbylo místo na ovládací panel a pole pro zobrazování informací. Zarovnání šachovnice je na střed a po obou stranách je prostor, kde se zobrazují vyhozené kameny. Pro zobrazení historie tahů jsem zvolil pravou stranu spodní lišty a na levou jsem umístil tlačítka pro ovládání. Střed lišty zobrazuje informaci, který hráč je na tahu.

Vzhledem k velikosti okna je jedno políčko šachovnice velké 68x68 pixelů a figurky mají velikost 60x60 pixelů. Pro možnost změny grafického vzhledu figurek jsem se je rozhodl umístit do zvláštního adresáře ve formátu BMP. Stejným způsobem jsem vyřešil i grafické zobrazení šachovnice.

Navrhl jsem si figurky a nakreslil podle požadované velikosti. Figurky jsem umístil na červené pozadí, která je při načítání uvažována jako transparentní.

4.5 Načítání a ukládání hry

Pro ukládání a načítání hry jsem zvolil ukládání do textového souboru, aby se dal editovat v textových programech a tímto dosáhnou možnosti upravení hry a nebo vytvoření vlastní hry, kterou si lze přehrát. Při vytváření hry pro přehrávání je zapotřebí, aby hra začínala od základního postavení kamenů a aby se kameny pohybovali podle pravidel šach, protože přehrávací funkce nekontroluje správnost tahu. Přehrávací funkce předpokládá, že tahy jsou podle pravidel. Tento soubor má následnou strukturu:

- první řádek obsahuje údaj pro funkci pat, je to počet tahů, ve kterém nebylo táhnuto pěšcem a ani nebyl vyhozen kámen
- od druhého řádku jsou pak všechny následující řádky stejné

- jeden řádek udává jeden půltah a jeho struktura je: A B C D E F, kde jednotlivá písmena znamenají:

A – číslo tahu

B – písmenná souřadnice počátečního půltahu

C – číselná souřadnice počátečního půltahu

D – písmenná souřadnice koncového půltahu

E – číselná souřadnice koncového půltahu

F – je nenulové pokud v tomto tahu došel pěšec

nakonec a udává za jaký kámen se proměnil

- hodnoty figur, na které se může změnit pěšec jsou následující:

1 – věž

2 – jezdec

3 – střelec

5 – dáma

Pozn.: kladná čísla značí kameny bílého záporná černého hráče

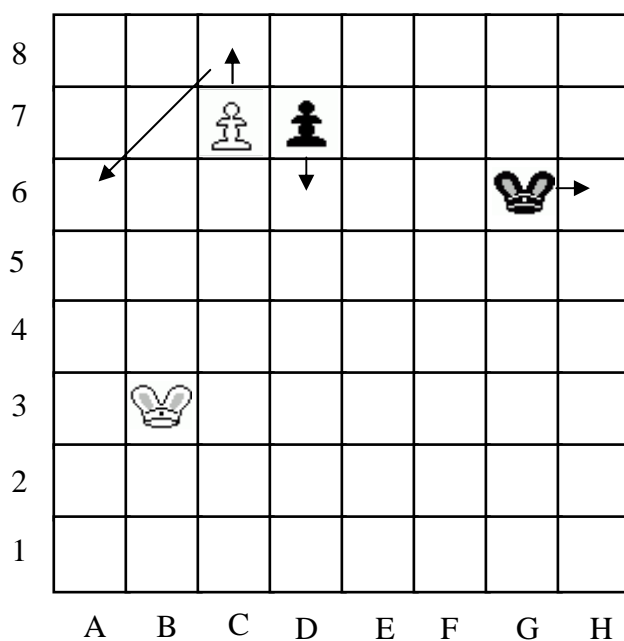
- příklad formátu souboru ukazuje tabulka 4.5.1

10
1 D 2 D 3 0
2 E 7 E 5 0
...
50 D 7 D 6 0
51 C 7 C 8 5
52 G 6 H 6 0
53 C 8 A 6 0
...

Tab. 4.5.1 Příklad formátu souboru

Na obrázku 4.5.1 je znázorněno jak bude probíhat přehrávání podle uložených informací v textovém souboru, které ukazuje tabulka 4.5.1.

- tah 50 – černý pěšec z D7 na D6
- tah 51 – bílý pěšec z C7 na C8 a 5 značí, že se zde změní na dámu
- tah 52 – černý král z G6 na H6
- tah 53 – dáma z C8 na A6



Obr. 4.5.1 Příklad přehrání hry

5. Zkoušení a testování programu

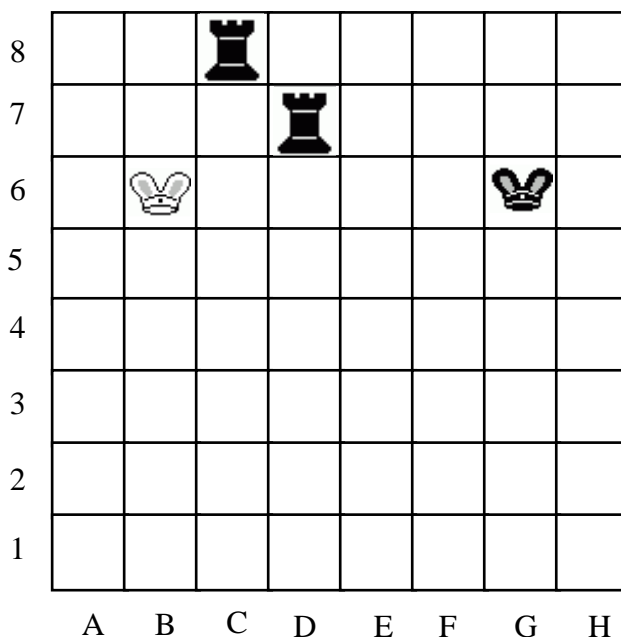
5.1 Ladění

Při testování jsem se setkával s několika problémy, které jsem musel řešit.

Prvním problémem byl generátor tahů, který sám o sobě pracoval správně, ale po přidání do projektu začal generovat nesmyslné tahy. Tento problém jsem vyřešil redundancí pole tahů a vše pak znova otestoval a vše bylo v pořádku.

Druhým problémem byla ohodnocování funkce. V základním tvaru byla její funkčnost bezchybná, ale jelikož jsem chtěl dosáhnout lepšího ohodnocení různých variant pozic, tak jsem vždy nasimuloval nějakou pozici a zkoušel jsem různé varianty ohodnocení.

Jeden z příkladů ukazuje obr 5.1.1, kde je černý na tahu.



Obr. 5.1.1 Příklad pozice pro ladění

Z obrázku 5.1.1 je jasné, že černý má převahu a je vidět, že černý může vyhrát. Ale problémem je, že ať se posune jakákoliv z věží tak, aby dostal bílý hráč šach přijde o ni. Různé možnosti sice existují, ale tento problém se mi nepodařilo vyřešit, protože když už

jsem skoro u cíle, tak jsem zkoušel hrát hru od začátku a intelligence počítače byla znatelně horší. Stejně kdyby se mi podařilo vyřešit tento problém, mohl bych řešit zase jiný, ale takových situací existují stovky a proto by bylo výhodnější pro takové pozice mít databázi koncovek a postupovat podle nějaké koncovky.

Dále jsem se snažil, aby si počítač lépe hlídal své kameny a nevyměňoval je za kameny, které měly menší materiální hodnotu. Když jsem dosáhl cíle, aby si počítač lépe hlídal své figurky tak se zněj stal neagresivní protihráč, který se snaží rozmístit kameny po šachovnici tak, aby měl co nejvíce kamenů hlídaných vlastními kameny. A tak jsem se vrátil k předchozí verzi, kdy byl počítač agresivnější, ale stávalo se, že provedl z logického pohledu nesmyslnou výměnu.

Posledním problémem bylo upravit ohodnocování funkci tak, aby se počítač snažil dát šach a dále mat. To jsem řešil tak, že jsem považoval za výhodnější pozici takovou, kdy má král menší pohyblivost a o málo více jsem zvýhodnil pozici kdy je král v šachu. Tato implementace se mi podařila celkem úspěšně zrealizovat.

5.2 Testování

Po dokončení programu jsem provedl test, který měl ověřit jak je program graficky a inteligenčně zdařilý. Program testovalo pět nezávislých uživatelů.

Test probíhal následovně.

- Nejprve si každý uživatel zahrál jednu hru.
- Pak jsem je nechal napsat hodnocení, ve kterém měli zhodnotit grafické rozhraní, užitečnost a dostatečnost zobrazovaných informací během hry a ovládání programu.
- Dále si každý uživatel zahrál ještě čtyři partie.
- Pak měli za úkol znovu napsat hodnocení.

Test byl záměrně sestaven tak, aby každý uživatel po první hře napsal hodnocení programu a pozbývajících čtyřech toto hodnocení napsal ještě jednou. Z prvního hodnocení se zjistilo jaký je první dojem na vytvořený program.

Podle hodnocení se všem grafická stránka líbila. Inteligenci počítače prohlásili dva lidé za lehčí, a že nebyl problém počítač porazit. Ostatním uživatelům se také podařilo počítač porazit, ale jejich názor byl, že to nebylo zas tak jednoduché. Všichni se shodli, že má program trochu zajímavé ovládání, ale nepovažovali to za nějakou závažnou chybu. Po nějaké době se prý dá na ovládání zvyknout a je to jen otázka zvyku.

Výsledky testu ještě shrnuje tabulka 5.2.1.

Uživatel	Věk	Zkušenosti v šachu	Skóre	Poznámky
U1	20	Středně pokročilý	2 výhra 2 prohra 1 pat	+ Příjemné grafické rozhraní + Možnost přehrání si hry
U2	19	Začátečník	1 pat 4 prohra	+ Zobrazování pohybu kamenů + Zajímavá grafika – Větší obtížnost
U3	23	Středně pokročilý	2 pat 1 výhra 2 prohra	+ Rychlost reakce – Horší zakončování hry počítače
U4	42	Pokročilý	5 výhra	+ Příjemné grafické rozhraní – Slabá inteligence
U5	39	Pokročilý	4 výhra 1 pat	+ Rychlost reakce – Počítač by mohl mít lepší inteligenci

Tab. 5.2.1 Výsledky testování

6. Závěr

Práce se zabývala prostudováním existujících šachových programů a realizací vlastního šachového programu včetně grafického rozhraní, pomocných funkcí a umělé inteligence. Dále shrnula základní algoritmy, které se používají ke tvorbě umělé inteligence pro šachy, ale nejen pro ně. Bylo ukázáno několik reprezentací šachovnice, které lze pro realizaci použít.

Vzniklý program má vlastní znázornění informací, které je potřeba znát během hry. Grafické rozhraní je implementováno pomocí knihovny Simple DirectMedia Layer. V omezeném čase, který jsem měl k dispozici pro realizaci bakalářské práce nebylo možné dosáhnout veliké inteligence počítače.

Program, ale umí načítat a přehrávat šachové turnaje uložené v textovém souboru. Tento soubor, ale musí mít definovanou strukturu, aby bylo možné hru načíst. Lze si uložit rozehranou hru a později v ní pokračovat nebo pokračovat v načtené hře, která neskončila matem nebo patem. Program je plně ovládán myší. Je zobrazováno pět posledních odehraných tahů z historie, ale v historii tahů se dá listovat, tak že není problém najít hledaný tah nebo kombinaci tahů. Program se nevyznačuje vynikající inteligencí, ale je postačující a grafická stránka programu podle uživatelů, kteří program testovali, je postačující.

Možností pokračování práce, aby měl program vyšší inteligenci by mohlo být:

- vylepšení umělé inteligence počítače tak, že by se do programu přidala databáze koncovek, aby se nemuselo počítat jaká je nerychlejší cesta k vítězství
- zavedení databáze počátečních tahů, aby počítač nezačínal stále stejným protitahem na zahajující tah bílého hráče.

Po doplnění databáze počátečních tahů a koncovek by mohl tento program porážet i mítry v královské hře šachu.

7. Seznam literatury

- [1] Domovská stránka programu Shredder
<http://www.shredderchess.com>
- [2] Domovská stránka programu Deep Fritz
<http://www.chessbase.com>
- [3] Stránky o programu winboard
<http://www.tim-mann.org/xboard.html>
- [4] Stránky o programu arena
<http://www.playwitharena.com/directory/download.htm>
- [5] Vysvětlení naivního algoritmu, algoritmu minimax a alfa-beta algoritmu
<http://www.chessmater.wz.cz/alg.php>
- [6] Popis redukčního algoritmu
<http://web.telecom.cz/vaclav.kotesovec/algorit/algoritmy.htm>
- [7] Domovské stránky programu Dev C++
<http://www.bloodshed.net/dev/devcpp.html>
- [8] Pavel Herout. Učebnice jazyka C. Nakladatelství KOPP, České Budějovice, IV. přepracované vydání, 2004
- [9] Pavel Herout. Učebnice jazyka C 2.díl. Nakladatelství KOPP, České Budějovice, II. přepracované vydání, 2004

Příloha A - Uživatelská příručka

Instalace

1. vložte CD do mechaniky
2. v adresáři šachy spusťte hru souborem šachy.exe
pokud nechcete mít CD v mechanice překopírujte celý adresář šachy na disk počítače

Ovládání

LTM – levé tlačítko myši

PTM – pravé tlačítko myši

Výběr figurek provedete LTM nad figurkou, kterou chcete hrát.

Zrušení výběru se provede PTM kdekoliv na hrací ploše.

Přemístění se provede druhým kliknutím LTM po výběru figurky.

Další ovládání viz obrázek A.1.

Hrací Plocha



Obr. A.1 Hrací plocha

- 1 – pole, kde se zobrazují vyhozené figurky
- 2 – takto je označeno kam se může táhnout
- 3 – aktuálně vybraná figurka
- 4 – začne novou hru
POZOR! Pokud máte rozehranou partii přijdete o ni
- 5 – udává kdo je na tahu
- 6 – historie tahů
První sloupec – číslo tahu
Druhý sloupec – počáteční místo tahu
Třetí sloupec – koncové místo tahu
- 7 – Při přehrávání posun vřed a vzad (pokud se nelze pohybovat není zobrazeno)
- 8 – Uloží hru do souboru save/hra.txt
POZOR! Přemaže již existující soubor (pokud chcete hru zachovat přejmenujte soubor hra.txt)
- 9 – Načte hru ze souboru save/hra.txt (pro načtení hry je potřeba, aby zde byl tento soubor, jiný se nenačte)
- 10 – Přepne se do režimu přehrávání

Pravidla

Cíl hry

Dát soupeřovu králi mat.

Hra

Hru začíná vždy bílý, pak táhne černý a dále se hráči střídají.

Hráč na tahu přenese jeden ze svých kamenů na volné pole nebo na pole, které je obsazeno spoluhráčovým kamenem. Ve druhém případě se soupeřův kámen odstraní ze hry.

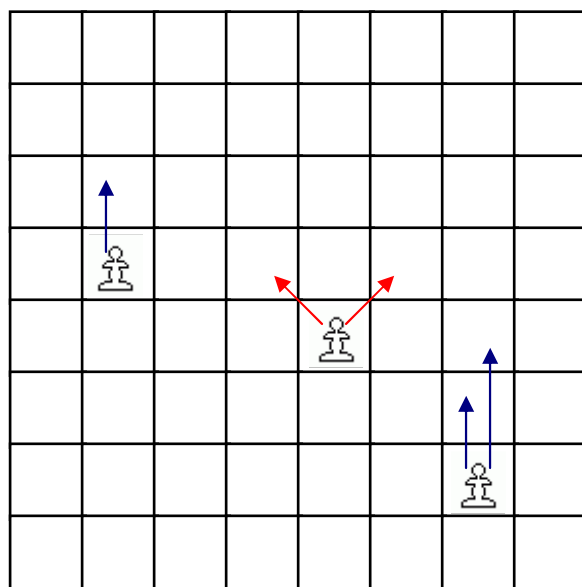
Král je v šachu, jestliže je ohrožen alespoň jedním soupeřovým kamenem, tj. tento kámen by mohl v příštím tahu vstoupit na královo pole.

Král, který se v šachu, musí této hrozbě uniknout v bezprostředně následujícím tahu. A to přesunem na jiné, bezpečné pole, zajištěním soupeřova kamene, který dává jeho králi šach, nebo zrušením šachu kamenem postavením mezi krále a šachující soupeřovu figuru.

Pohyb figur:

Pěšec

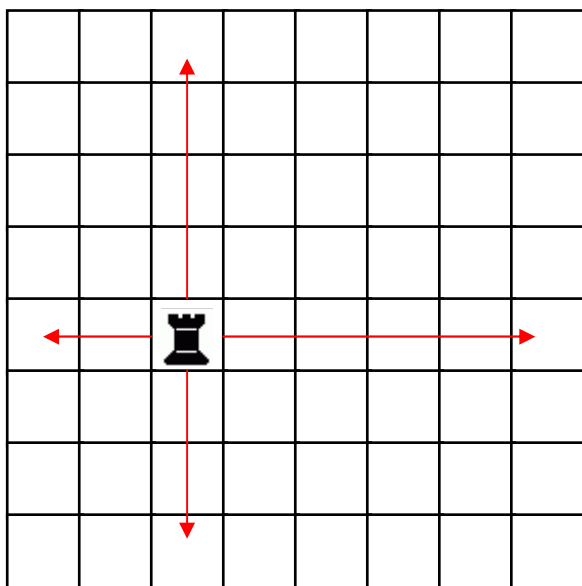
Pěšec se pohybuje jen vpřed o jedno pole. Výjimku tvoří tah z výchozí pozice kdy může táhnout o jedno nebo dvě pole. Druhou výjimkou je braní soupeřova kamene, kdy pěšec může sebrat soupeřův kámen o jedno pole vpřed a šikmo vpravo nebo vlevo. Pokud se pěšcem dostanete na konec hrací plochy smíte si ho vyměnit za libovolnou figurku kromě krále bez ohledu na počet ostatních kamenů ve hře. Takže můžete mít tři a více jezdců, dam, věží nebo střelců. Tato výměna proběhne v rámci dokončení tahu a její působnost je okamžitá.



Obr. A.2 Pohyb pěšce

Věž

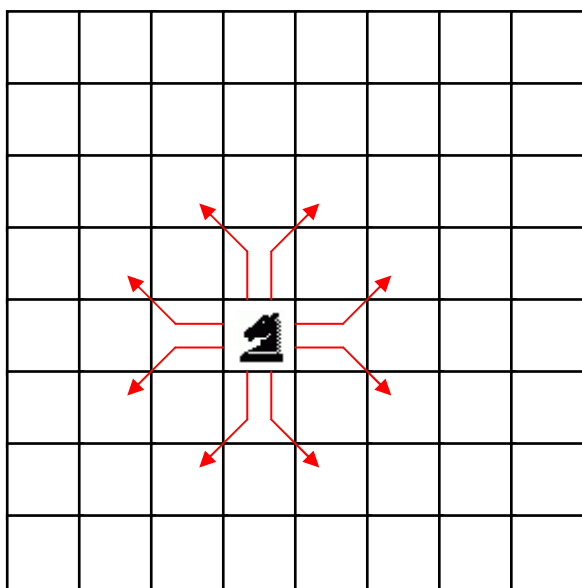
Věž se může pohybovat vodorovně nebo svisle o libovolný počet polí, ale nesmí přeskakovat svoje ani soupeřovi kameny.



Obr. A.3 Pohyb věže

Jezdec

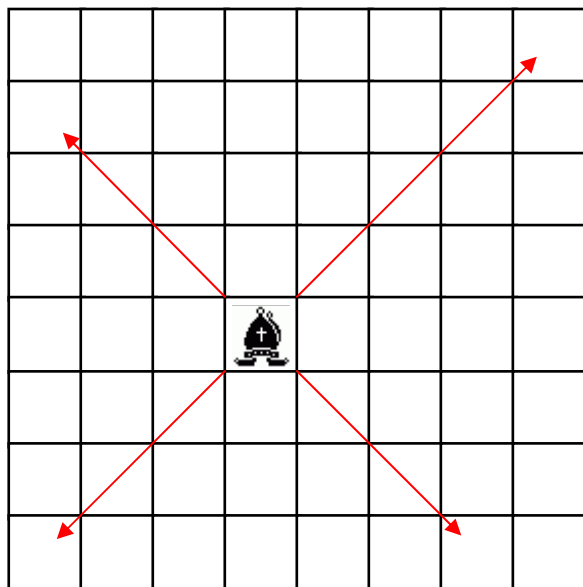
Jezdec se pohybuje zajímavým způsobem. O jedno pole horizontálním nebo vertikálním směrem a bez zastavení jedno pole diagonálním směrem. Od výchozího pole se stále vzdaluje. Jezdec je jediný kámen, který může přeskakovat ostatní kameny.



Obr. A.4 Pohyb jezdce

Střelec

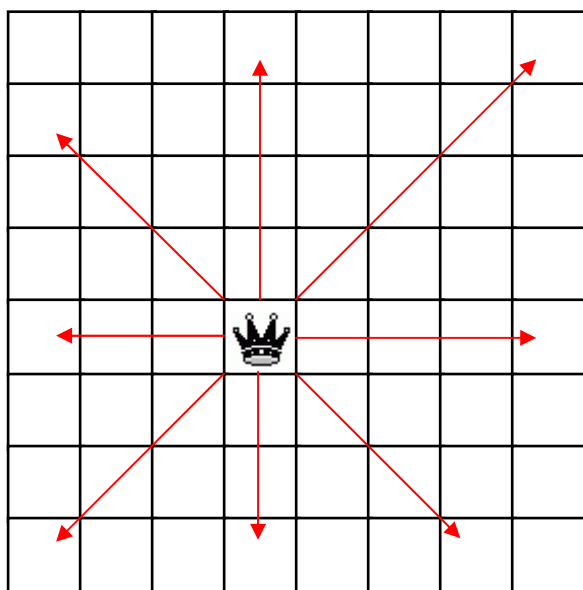
Střelec se může pohybovat diagonálním směrem o libovolný počet políček, ale nesmí přeskakovat svoje ani soupeřovi kameny.



Obr. A.5 Pohyb střelce

Dáma

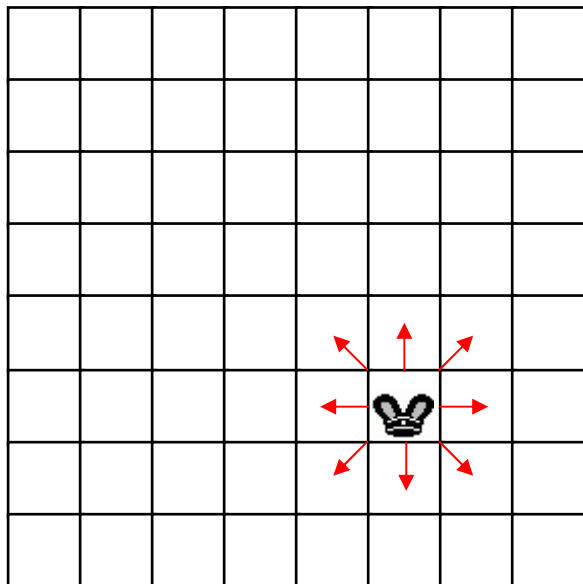
Dáma se může pohybovat horizontálně a vertikálně o libovolný počet polí jako věž a dále se může pohybovat diagonálně všemi směry jako střelec. Opět platí, že při svém pohybu nesmí přeskakovat svoje ani soupeřovi kameny. Díky svému pohybu je dáma nejsilnějším kamenem ve hře.



Obr. A.6 Pohyb dámy

Král

Král může postoupit jen na sousední pole libovolným směru, ale toto pole nesmí být ohroženo soupeřovým kamenem. Výjimku tvoří zvláštní dvojtah zvaný rošáda.



Obr. A.7 Pohyb krále

Rošáda

Rošáda je zvláštní druh dvojtahu, ve kterém smí hráč posunout dva ze svých kamenů – jednu z věží a krále. Počítá se, ale jako tah krále. Máme dvě rošády malou a velkou. Malá je, že se král posune o dvě pole směrem k bližší věži a věž se posune o dvě pole směrem ke králi. Velká rošáda je skoro stejná jako malá, ale provádí se směrem ke vzdálenější věži a věž se posune o tři pole.

Rošáda smí být provedena jen za těchto okolností:

1. Králem ani věží ještě nebylo taženo.
2. Pole, na kterém král stojí, které musí překročit, nebo které má obsadit, není ohroženo soupeřovým kamenem.
3. Všechna pole mezi králem a věží, s nimiž bude rošáda provedena, jsou volná.



Obr. A.8 Malá rošáda



Obr. A.9 Velká rošáda

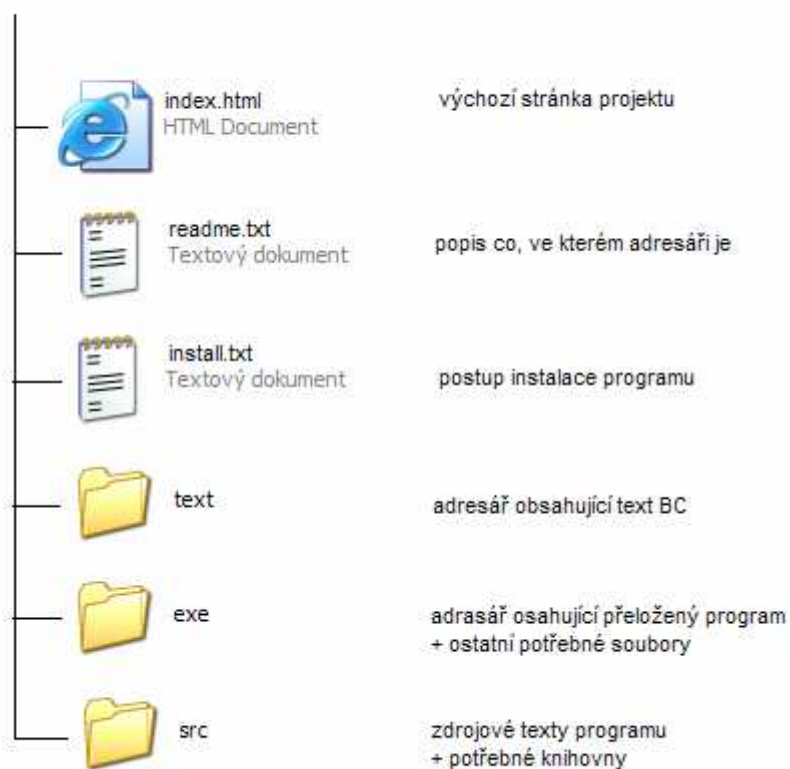
Konec Hry

Nedokáže-li hráč zlikvidovat svým nejbližším tahem ohrožení svého krále a král zůstane dál v šachu, hra končí jeho porážkou. podle ustálené zvyklosti se říká, že dostal mat.

Partie je nerozhodná:

- Pokud král hráče, který je na tahu, není v šachu a hráč nemůže provést žádný přípustný tah. Říká se, že partie skončila patem.
- Pokud oba hráči provedli posledních 50 po sobě následujících tahů, aniž byl brán nějaký kámen nebo bylo taženo pěšcem.
- Když zůstaly ve hře takové kameny, které neumožňují dosáhnout jednomu z hráčů vítězství:
 1. Dva osamělí králové.
 2. Král se střelcem stojí proti králi a střelci a oba střelci jsou buď na bílé nebo na černé diagonále.
 3. Král stojí proti králi a střelci, nebo proti králi a jezdcí.

Příloha B – Obsah přiloženého CD



Obr. B.1 Obsah přiloženého CD