Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Implementace algoritmu šachové hry

Plzeň, 2013 Jan Sehnal

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 22. června 2013

Jan Sehnal

Abstract

Chess Algorithm Implementation

This bachelor thesis deals with the artificial intelligence of the chess programming. The thesis is focused on the typical characteristics of the chess engines and describes the difference between the human and computer chess playstyle. The thesis surveys and analyses the techniques which are necessary for implementing a chess program. The description of the chess engine creation is involved including the problems which appeared during the process of creating.

Abstrakt

Implementace šachového algoritmu

Tato práce se zabývá umělou inteligencí šachového programu. Obsah práce je zaměřen na typické vlastnosti šachových motorů, je popsán rozdíl přístupu lidského faktoru a stroje k šachové hře. Součástí práce je průzkum a analýza technik potřebných k implementaci šachového programu. V textu je zahrnut popis tvorby vlastního šachového programu, včetně vzniklých úskalí.

Obsah

1 Úvod

Šachy jsou tradiční deskovou hrou pro dva hráče. Mezi důvody popularity šachů patří bezesporu variabilita pozic, ve kterých se hra může v daný moment nacházet, a také faktor duševního souboje, který spolu dva hráči svádějí. Právě druhý z uvedených důvodů nabyvá zcela jiných rozměrů, je-li jeden z hráčů nahrazen tzv. umělou inteligencí (UI).

Z filozofického hlediska pak nastává další variace odvěkého souboje člověk versus stroj, z hlediska programátorského se jedná o výzvu, kdy se programátor snaží naučit svůj stroj optimálně zareagovat v nastalých herních pozicích.

Vytvoření umělé inteligence schopné hrát šachy je vhodnou programátorskou úlohou, jelikož je hra šachu dostatečně složitá. Existuje průměrně 400 nových pozic, které mohou vzniknout z pozic původních po odehrání jednoho tahu každým z hráčů. Toto číslo po odehrání tří tahů oběma hráči přesahuje hranici 9 milionů možných pozic. Průměrná délka hry je třicet tahů, teoreticky nejdelší hra by pak čítala 5949 tahů. Je tedy nasnadě, že cílem programátora není nalézt tah, který vede k jistému vítězství, ale tah, který se jeví jako nejlepší v nejbližším horizontu událostí.

Programátorovou výhodou je absence neurčitosti ve hře šachu. Každá pozice je přesně dána, neexistují neznámé aspekty, narozdíl od her karetních, kdy je hráči obvykle určitá podmnožina faktorů potřebných k nalezení ideálního tahu zatajena.

Cílem této práce je prozkoumat a zhodnotit existující metody řešení šachové úlohy, poté jednu z jednodušších metod vybrat a implementovat ji. Jednotlivé kapitoly se zabývají následujícími tématy: měřitelný výkon hráčů a stroje, silné a slabé stránky počítačového šachu, reprezentace šachové úlohy, generování tahů, šachové algoritmy, implementace šachového motoru a jeho následné testování.

2 Měřitelný výkon stroje

V této kapitole jsou nastíněny způsoby, kterými se hodnotí herní výkonnost hráčů a šachových motorů.

2.1 Číslo ELO

V šachu je zajisté zapotřebí ohodnotit herní sílu hráče, ať už se jedná o fyzickou osobu, nebo stroj. K tomu slouží tzv. číslo ELO. Nejedná se o zkratku, toto číslo bylo pojemnováno podle amerického fyzika a statistika Arpada Ela. Jedná se o spolehlivý systém hodnocení, který definuje míru relativní herní síly jednotlivých hráčů. Ke změně tohoto čísla jednotlivých hráčů může dojít pouze při autorizovaných turnajích. Čím vyšší ELO číslo hráč má, tím je hráč silnější v poměru s ostatními hráči. Procento úspěšnosti a diference ELO znázorňuje tabulka 2.1 [1].

%	D	%	D	%	D	%	D	%	D
0	-	20	-240	40	-72	60	72	80	240
1	-677	21	-230	41	-65	61	80	81	251
2	-589	22	-220	42	-57	62	87	82	262
3	-538	23	-211	43	-50	63	95	83	273
4	-501	24	-202	44	-43	64	102	84	284
5	-470	25	-193	45	-36	65	110	85	296
6	-444	26	-184	46	-29	66	117	86	309
7	-422	27	-175	47	-21	67	125	87	322
8	-401	28	-166	48	-14	68	133	88	336
9	-383	29	-158	49	-7	69	141	89	351
10	-366	30	-149	50	0	70	149	90	366
11	-351	31	-141	51	7	71	158	91	383
12	-336	32	-133	52	14	72	466	92	401
13	-322	33	-125	53	21	73	175	93	422
14	-309	34	-117	54	29	74	184	94	444
15	-296	35	-110	55	36	75	193	95	470
16	-284	36	-102	56	43	76	202	96	501
17	-273	37	-95	57	50	77	211	97	538
18	-262	38	-87	58	57	78	220	98	589
19	-251	39	-80	59	65	79	230	99	677

Tabulka 2.1: Úspěšnost vzájemných her při dané diferenci čísla ELO

Počet procent znázorňuje šanci na výhru daného hráče, D pak označuje diferenci, neboli rozdíl hráčova a protihráčova čísla ELO. Má-li tedy hráč nižší číslo ELO o 125 než daný soupeř, měl by tohoto soupeře porazit přibližně v každé třetí partii.

Hráčovo číslo ELO stoupá, překoná-li hráč dosaženým výsledkem statistický odhad, a naopak klesá, prohraje-li hráč více partií, než bylo očekáváno. Číslo ELO by mělo dlouhodobě odrážet hráčovy výkony.

2.2 Ohodnocení výkonu stroje

Při ohodnocování výkonu stroje nastávají určité problémy. V prvé řadě, číslo ELO je spojováno s identitou fyzické osoby. Tato osoba se postupem času vyvíjí, stejně tak se vyvíjí její ELO. V tomto případě je přiřazena jedna hodnota ELO právě jedné konkrétní osobě. Problém nastane, pokud bychom chtěli to samé provést s počítačovým programem. Počítačové programy jsou vypouštěny na trh v mnoha verzích, často s identickými názvy.

Bylo by složité přidělovat identitu ekvivalentní lidskému jedinci jednotlivým vydáním programu. Navíc, další neodmyslitelnou složkou, která ovlivňuje výkon šachového programu, je hardware, na kterém tento program pracuje. Chtěli-li bychom tedy přidělovat globální identifikátory šachovým programům, bylo by nutné unifikovat hardware, na kterém by byly všechny tyto programy spouštěny, což by bylo obtížně realizovatelné, navíc vzhledem k neustálé evoluci hardwaru, nevhodné.

Číslo ELO se všemi svými aspekty tedy nebylo zavedeno také pro počítačový šach. Přesto je jeho hodnota používána pro orientační určení výkonnosti šachových motorů.

Podle [2] odpovídá ELO o hodnotě 100 začátečníkovi, který se právě naučil pravidla. Začátečníci na poli bodovaného šachu pak mají ELO v rozsahu 800-1000. Nejlepších 10% turnajových hráčů má ELO o hodnotě 1900. Šachovým mistrům a velmistrům pak odpovídá ELO o hodnotě ~2200, resp. ~2500.

Existuje více způsobů, jak určit přibližné ELO šachového softwaru. Jedním z nich je uspořádávání rozsáhlých kompeticí, kde proti sobě soupeří šachové programy. Tento způsob je nejpřesnější, ale také nejnákladnější. Pro určení hrubého odhadu síly daného šachového motoru je možné použít různé poziční testy. Programu jsou předkládány pozice, následné reakce na tyto pozice jsou pak ohodnocovány. Toto řešení s sebou nese jisté nevýhody. Z důvodu dříve zmíněného množství všech možných šachových pozic není možné pomocí testu pokrýt všechny situace, testy jich mohou obsahovat jen zlomek. Navíc, optimální tah s nejvyšším ohodnocením může být nalezen zcela náhodně, aniž by měl program v době jeho provedení propočítáno, k čemu tento tah povede. Pomocí testů taktéž není možné zmapovat herní strategie.

2.3 Závislost výkonu stroje na hloubce prohledávání

Je nasnadě, že výkon šachového motoru poroste, zvýšíme-li hloubku prohledávání. Při odhadu 35 možných tahů na jednotlivý půltah také ale drasticky roste složitost prohledávacího stromu. Závislost výkonu šachového enginu na hloubce prohledávání zkoumal americký informatik Ken Thompson. Naprogramoval sedm verzí svého šachového enginu Belle. Pojmenoval je v řadě P3, P4, P5, P6, P7, P8 a P9. Číslo v názvu představovalo počet prohledávaných půltahů, tedy hloubku vyhledávacího stromu, který byl v dané verzi vytvořen pro nalezení nejlepšího tahu. Takto modifikované programy pak nechal sehrát 20 partií každý s každým. Výsledky obsahuje tabulka 2.2 [1].

	P3	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**ELO**	**ELO Dif.**
P3	-	8,0	0,0	0,0	0,0	0,0	0,0	1101	0
P4	12,0	-	5,0	0,5	0,0	0,5	0,0	1235	+134
P5	20,0	15,0	-	3,5	3,0	0,5	0,0	1570	+235
P6	20,0	19,5	16,5	-	4,0	1,5	1,5	1826	+256
P7	20,0	20,0	17,0	16,0	-	5,0	4,0	2031	+205
P8	20,0	20,0	19,5	18,5	15,0	-	5,5	2208	+167
P9	20,0	20,0	18,5	18,5	16,0	14,5	-	2328	+120

Tabulka 2.2: Výsledky vzájemných partií mezi šachovými motory

Každá řádka tabulky odpovídá jedné verzi enginu. Jednotlivé buňky řádek vyjadřují počet výher dané verze proti verzím ostatním.

Číslo ELO verze P8 bylo nastaveno jako výchozí, jelikož verze P8 byla nasazena v mnoha turnajích, tudíž byla jeho herní síla již ověřena. Zbylé hodnoty byly dopočítány z výchozí hodnoty.

Test ukazuje, že šachový motor, který umí prohledávat do hloubky ctyř půltahů, by měl být schopen hrát vyrovnaně s mírně pokročilým hráčem. K dosažení herní úrovně velmistrů je zapotřebí (mimo jiné) prohledávání do hloubky nejméně deseti půltahů.

2.4 Slabiny šachových enginů

2.4.1 Materiál

Materiál bývá základním stavebním kamenem ohodnocovacích funkcí. Šachový program generuje pozice, do kterých by se mohla partie vyvinout. Tyto pozice následně ohodnotí a snaží se pokračovat podle takového scénáře, který vede k co nejpříznivějšímu ohodnocení. Ohodnocovací funkce obecně kladou na materiál veliký důraz, vyšší, než na různá poziční kritéria. Z tohoto důvodu je pro šachové programy problematické rozeznat materiálovou oběť za účelem získat poziční nadvládu, přesahuje-li výtěžek z této oběti hloubku prohledávání programu.

2.4.2 Horizont

Efekt horiznotu je situace, kdy šachový engine provede tah, který vede ke ztrátě, i přes to, že v předchozích tazích o špatnosti tohoto tahu věděl. K tomuto efektu dochází v situaci, kdy se tah, vedoucí ke ztrátě, nachází v maximální hloubce prohledávání nebo v její blízkosti. Tento kritický tah je pak posunut až za hranici hloubky prohledávání ve chvíli, kdy se naskytne jiný tah, který tuto ztrátu oddálí.

2.4.3 Koncová hra

V koncové hře platí jiná pravidla než v předchozích částech partie. Neplatí již výhoda obranné pozice pro krále, pěšci získávají drasticky na hodnotě, čím více se přiblíží poslední (resp. první) řadě. Problematické jsou taktéž matové kombinace, jelikož i ve velké převaze počítače tyto kombinace většinou leží mnohem hlouběji, než kam sahá vyhledávací strom.

2.4.4 Strategie

Z důvodu ohodnocování každé pozice jako samostatné jednotky postrádají šachové motory aspekt dlouhodobého plánu. Každý tah je volen tak, aby hodnota získaná aplikováním ohodnocovací funkce byla co nejvyšší. Absence strategie se nejvíce promítne v uzavřených partiích, kde se nenabízí mnoho výměn. Program provádí tahy, které pouze kosmeticky upravují ohodnocení aktuální pozice, zatímco lidský protivník má prostor k přeskupování figur a přípravě na následný útok.

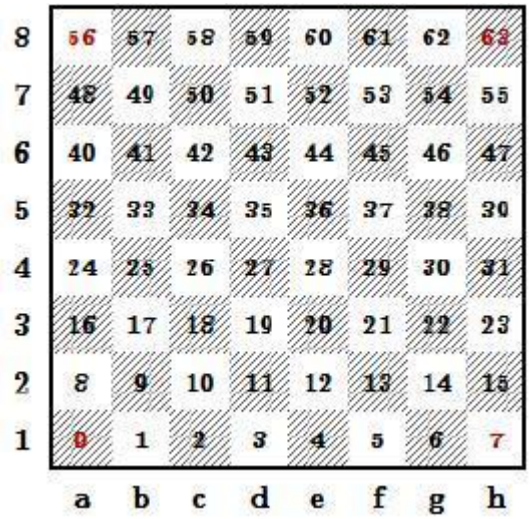
3 Reprezentace úlohy, generování tahů

3.1 Reprezentace šachovnice

Při tvorbě šachového programu je zapotřebí začít „od základu“. V tomto případě je základem reprezentace šachovnice. Jedná se o způsob, kterým je popsána aktuální, a tedy jakákoli možná, pozice v partii. Existuje více způsobů, jak danou pozici zapsat.

3.1.1 Reprezentace 8x8

Pozice je reprezentována maticí, popř. polem o 64 prvcích. Každý z prvků matice odpovídá právě jednomu poli šachovnice. Hodnota každého z prvků vyjadřuje, zda se na poli, které je vyjádřeno daným prvkem, nachází nějaká figura či je pole prázdné. Tento způsob reprezentace je nejpohodlnější pro představivost, nicméně přináší jistá úskalí. Při následném generování možných tahů je nutné manuálně ošetřit situace, kdy figury přeskakují z jednoho okraje na druhý. Například, je-li nadefinován jako jeden z možných tahů koně tah: *nová pozice = aktuální pozice + 17*, je-li *aktuální pozice = 23*, dojde k přeskočení okraje, viz Obr. 3.1 [3].

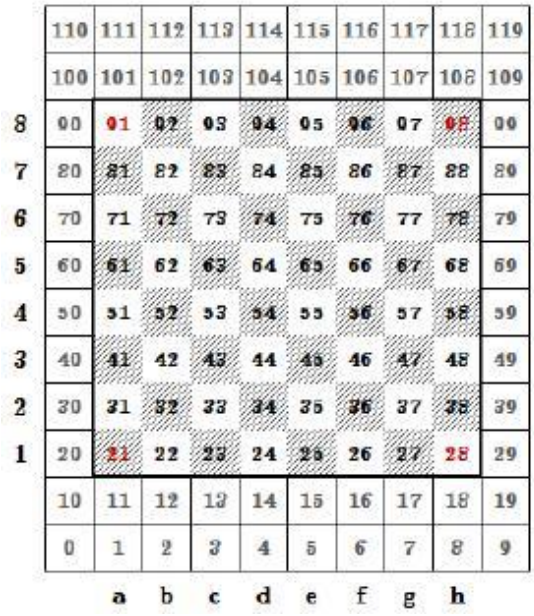


Obrázek 3.1: Reprezentace šachovnice 8x8

3.1.2 Reprezentace 10x12

Tato reprezentace vychází z reprezentace 8x8. Původních 64 polí je obaleno dalšími poli tak, že vznikne struktura o celkovém rozměru 10x12, viz. Obr. 3.2 [4].

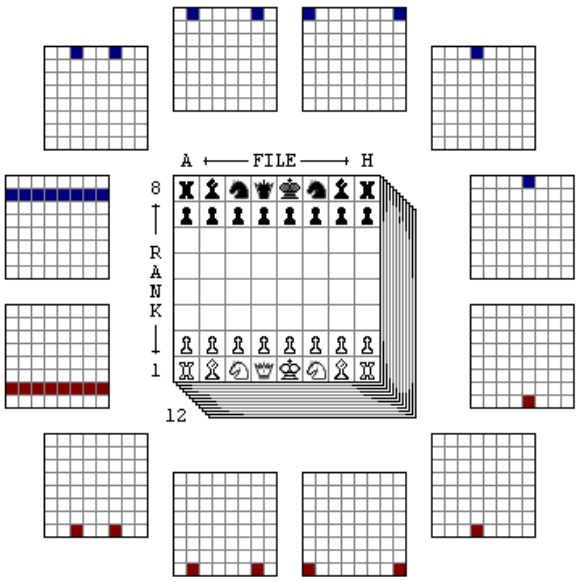
Nově přidaná pole slouží jako nárazníkové pásmo, prostor mimo hru, zakázané území. Pro ošetření všech možností nesprávného přesunu figur přes okraj je zapotřebí dvou rezervních sloupců či řad u každého okraje šachovnice. Jelikož při horizontálním přeskoku je využita rezerva z obou stran, je možné z každé strany jeden rezervní sloupec odebrat.



Obrázek 3.2: Reprezentace šachovnice 10x12

3.1.3 Bitová pole

Narozdíl od výše zmíněných reprezentací, bitová pole nevychází ze samotné šachovnice, ale z jednotlivých figur. Pro reprezentaci celé šachovnice je zapotřebí 12 bitových polí, každé o velikosti 64 bitů: viz Obr. 3.3 [5]. Každé z těchto dvanácti polí odpovídá jednomu druhu figury na šachovnici: král, královna, věž, střelec, kůň a pěšec, a to pro každou barvu. Jednotlivé bity fungují jako logická hodnota, která vyjadřuje prezenci příslušné figury na daném poli šachovnice. Reprezentace bitovými poli vytváří jiný způsob pohledu na generování tahů. Tahy je možné generovat pomocí logických funkcí, vytvořených použitím soustav bitových polí.



Obrázek 3.3: bitová pole

3.2 Generování tahů

3.2.1 Podle legálnosti

3.2.1.1 Pseudo-legální

Vygenerované tahy respektují základní šachová pravidla o pohybu figur, není ale brán ohled na vystavení krále šachu po provedení tahu. Podmínka šachu je následně kontrolována mimo generátor tahů.

3.2.1.2 Legální

Oproti pseudo-legálnímu způsobu jsou vždy generovány pouze legální tahy, je tedy při každém potenciálním tahu ověřováno, zda jeho provedením není král vystaven šachu, a to přímo na úrovni generátoru tahů.

3.2.2 Podle postupu:

3.2.2.1 Speciální generátory

Hlavní generátor tahů může být doplněn dalšími speciálními generátory, které jsou vytvořeny pro specifické události, např. v situaci, kdy se král nachází v šachu, má smysl generovat pouze takové tahy, které šachu bezprostředně zabrání, tedy: posunutí krále na jiné pole, sebrání útočící figury či představení vlastní figury.

3.2.2.2 Etapové generování

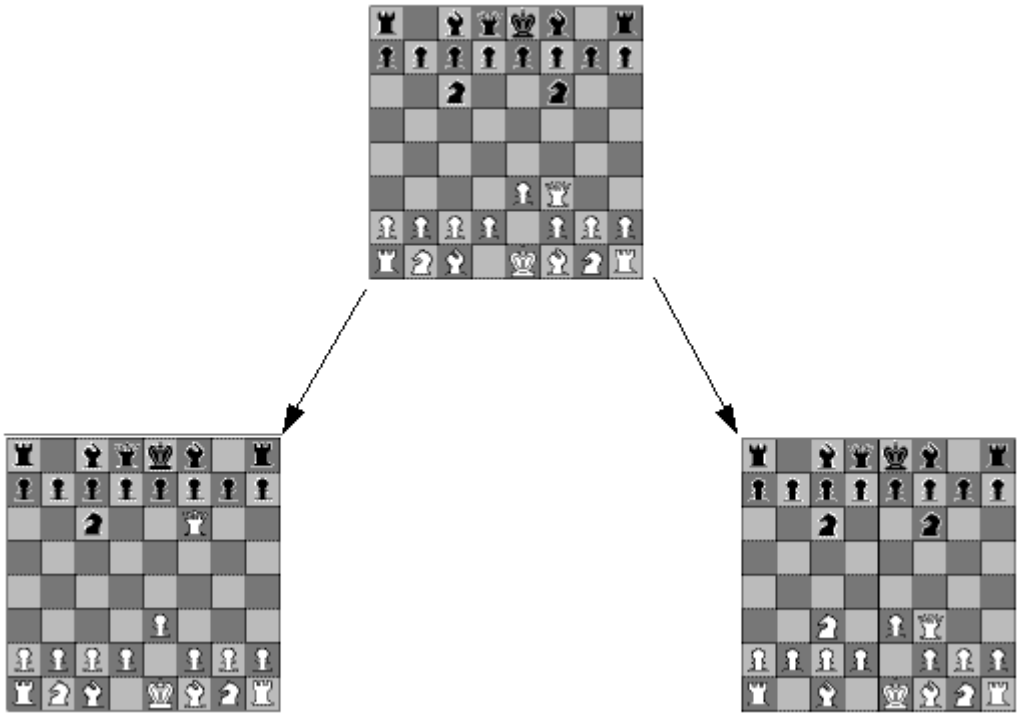
Tahy je možné také rozdělit do různých kategorií. Etapové generování testuje nejdříve takové tahy, u kterých je percentuálně vyšší šance, že se stanou při dané hloubce hledání nejlepším řešením. Potenciálními adepty jsou tahy, kterými je ohrožen soupeřův král, či tahy, kterými je odstraněna některá soupeřova figura.

4 Šachové algoritmy

Úlohou šachového algoritmu je za pomoci ohodnocovací funkce nalézt ideální tah pro danou pozici. Se šachovým algoritmem úzce souvisí hloubka prohledávání, která představuje počet půltahů, které budou prohledány. Některé algoritmy uvažují všechny možné kombinace půltahů, které by mohly být z dané pozice zahrány, jiné algoritmy zužují rozsah prohledávání podle předem zadaných kritérií.

4.1 Základní algoritmus

Základní algoritmus není koncipován k propočítávání tahů do hloubky. Vychází z aktuální pozice a hledá takový tah, který vede k nejvyššímu ohodnocení po zahrání tohoto tahu. Základní algoritmus vygeneruje všechny možné tahy zahratelné z aktuální pozice. Poté odehraje každý z vygenerovaných tahů a ohodnotí nově vzniklou pozici. Je-li nové ohodnocení lepší, nežli dosavadní, je poslední použitý tah uložen jako nejlepší. Následně je pozice navrácena do původního stavu. Algoritmus končí po vyzkoušení všech dostupných tahů, nejlepší nalezený tah je odeslán jako návratová hodnota algoritmu. Část vyhledávacího stromu základního algoritmu je zobrazena na Obr. 4.1 [6]. Ve skutečnosti by tento strom obsahoval přes třicet dalších diagramů vycházejících z diagramu kořenového, hloubka by zůstala ve všech případech rovna jedné.



Obrázek 4.1: Strom základního algoritmu

Pseudokód základního algoritmu vypadá následovně:

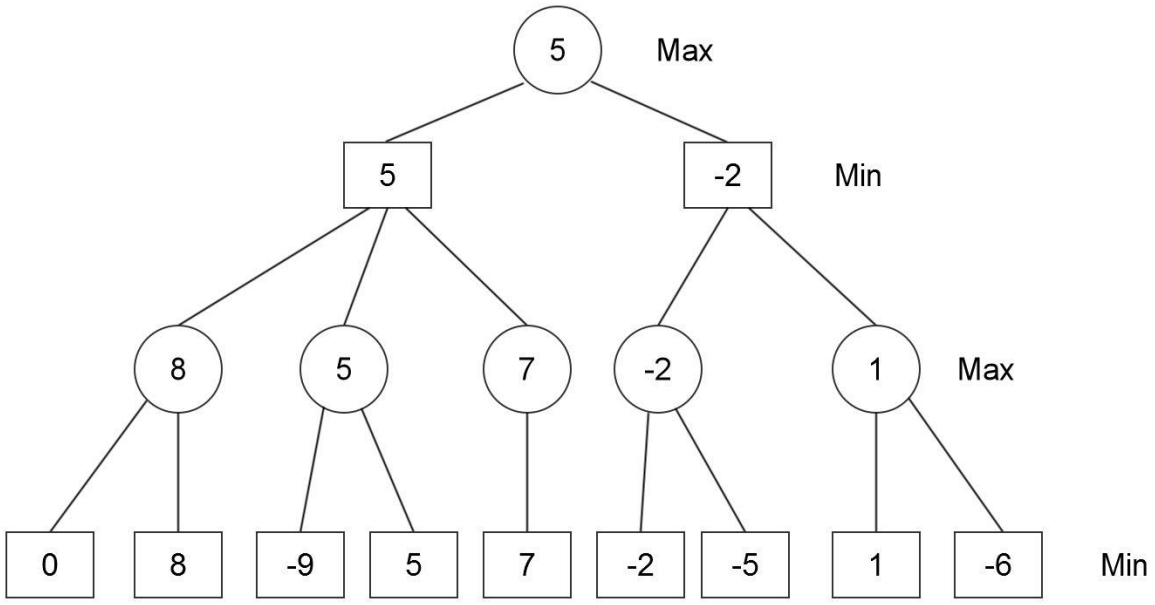
```
najdiNejlepsiTah(){

      Tah tahy[] = generujTahy();
      int nejlepsiHodnota = -INF;
      Tah nejlepsiTah;
      For(Tah t : tahy){
            sachovnice.zahrajTah(t); /* odehrání vygener. Tahu
            int hodnota = sachovnice.ohodnotPozici();
            if(hodnota > nejlepsiHodnota){
                  nejlepsiHodnota = hodnota;
                  nejlepsiTah = t;
            }
            sachovnice.vratTah(t); /* návrat do pův. Pozice
      }
      return nejlepsiTah; /* nejlepsi tah, existuje-li
}
```

Pseudokód 4.1: Základní algoritmus

4.2 Algoritmus MinMax

MinMax je klasickým algoritmem, který se používá ve strategických hrách, kde se hráči střídají na tazích: např. Reversi, Piškvorky, Dáma. Šachový MinMax vychází ze základního algoritmu. MinMax, stejně jako základní algoritmus, generuje pro danou pozici všechny dostupné tahy. Narozdíl od základního algoritmu ale nekončí po vygenerování a prozkoumání první generace půltahů, nýbrž pro každý ze získaných půltahů vygeneruje novou sadu půltahů. Takto pokračuje rekurzivně do hloubky *n*. Princip MinMaxu je znázorněn na Obr. 4.2.



Obrázek 4.2: Algoritmus MinMax

Tento diagram reprezentuje prohledávání do hloubky tří půltahů z výchozího bodu. Každá úroveň diagramu představuje jeden půltah, jednotlivé uzly diagramu obsahují ohodnocení konkrétní pozice. Rozvinutí uzlů do jednotkové hloubky je ekvivalentní výše zmíněnému základnímu algoritmu.

V šachové partii se střídají tahy bílého a černého. Partie je podle zvolené ohodnocovací funkce vyrovnaná, rovná-li se hodnota určité pozice nule. Pokud by chtěl bílý zvrátit partii ve svůj prospěch, potřeboval by dosáhnout co nejvyššího kladného ohodnocení dané pozice. Naopak černý se snaží tuto hodnotu snižovat. MinMax předpokládá, že každý z hráčů v dané pozici zvolí nejvýhodnější dostupný tah. V každé z úrovní *„Max“* je tedy vybráno nejvyšší ohodnocení potomků, naopak v úrovních *„Min“* jsou vybírána ohodnocení nejnižší. Z tohoto postupu mimo jiné vyplývá, že šachový program na principu MinMaxu nebude strojit na svého soupeře léčky, jelikož automaticky předpokládá, že soupeř zareaguje správně. MinMax bývá používán pro svou jednoduchost. Nevýhodou MinMaxového algoritmu je množství uzlů, které jsou vygenerovány. Při rostoucí hloubce prohledávání přestává být klasický MinMax efektivní.

Pseudokód MinMaxu je znázorněn na následující stránce.

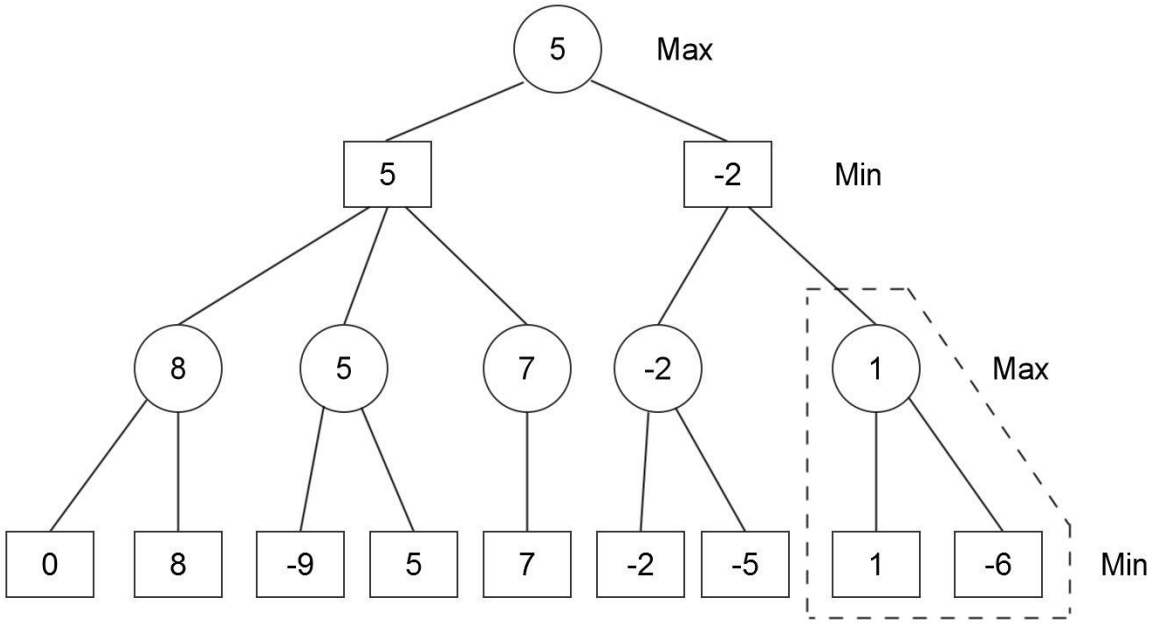
```
max(int hloubka){ /* hledání tahu pro bílého
      if(hloubka == 0){
            return sachovnice.ohodnotPozici();
      }
      Tah tahy[] = generujTahy();
      int maximalniHodnota = -INF;
      For(Tah t : tahy){
            sachovnice.zahrajTah(t); /* odehrání vygener. Tahu
            int hodnota = min(hloubka – 1);
            if(hodnota > nejlepsiHodnota){
                  nejlepsiHodnota = hodnota;
            }
            sachovnice.vratTah(t); /* návrat do pův. Pozice
      }
}

min(int hloubka){ /* hledání tahu pro černého
      if(hloubka == 0){
            return sachovnice.ohodnotPozici();
      }
      Tah tahy[] = generujTahy();
      int minimalniHodnota = INF;
      For(Tah t : tahy){
            sachovnice.zahrajTah(t); /* odehrání vygener. Tahu
            int hodnota = max(hloubka – 1);
            if(hodnota < nejlepsiHodnota){
                  nejlepsiHodnota = hodnota;
            }
            sachovnice.vratTah(t); /* návrat do pův. Pozice
      }
}
```

Pseudokód 4.2: Algoritmus MinMax

4.3 Algoritmus Alfa-Beta prořezávání

Alfa-Beta prořezávání je značným vylepšením MinMaxu. Vychází z faktu, že není zapotřebí prohledávat takové větve vyhledávacího stromu, které na celkový výsledek nebudou mít žádný vliv z důvodu existence dominantnější hodnoty na stejné úrovni tohoto stromu. Větev stromu, kterou je možné odříznout, je vyobrazena na Obr. 4.3.



Obrázek 4.3: Alfa-Beta ořezávání

Označenou větev je možné odříznout, jelikož hodnota uzlu ležícího nad označenou větví vzhledem k pravidlům MinMaxu může být pouze *-2* a nižší, tudíž nebude jeho hodnota na nejvyšší úrovni stromu nikdy vybrána.

Pseudokód AlfaBeta prořezávání vypadá následovně:

```
alfaBetaOrezavani(int hloubka, int alfa, int beta){
      if(hloubka == 0){
            return sachovnice.ohodnotPozici();
      }
      Tah t[] = generujTahy();
      for(Tah t : tahy){
            sachovnice.zahrajTah(t);
            hodnota = -alfaBetaOrezavani(hloubka -1, -beta, -alfa);
            sachovnice.vratTah(t);
            if(hodnota > beta){ /* není zapotřebí prohledávat
                  return beta;
```

```
            }
            if(hodnota > alfa){ /* aktualizace hodnoty
                  alfa = hodnota;
            }
      }
      return alfa;
}
```

Pseudokód 4.3: Algoritmus AlfaBeta ořezávání

V tabulce 4.1 jsou uvedeny nejlepší a nejhorší případy průběhu algoritmu Alfa-Beta ořezávání [7]. Je uvažováno 40 možných tahů pro každou pozici.

Hloubka prohledávání	Počet ohodnocovaných pozic	
	Nejhorší případ	Nejlepší případ
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 4.1: Nejhorší a nejlepší případ Alfa-Beta ořezávání

Nejhorší případ nastane, nebyla-li v celém průběhu odříznuta ani jedna větev. V takovém případě degeneruje algoritmus Alfa-Beta ořezávání na původní MinMax. [1] uvádí, že při použití Alfa-Beta ořezávání je v průměrném případě možné prohledávat do dvakrát větší hloubky, než s algoritmem MinMax.

4.4 Iterativní prohlubování

Účelem iterativního prohlubování je zefektivnit algoritmus Alfa-Beta ořezávání. Aby bylo Alfa-Beta ořezávání efektivní, je zapotřebí, aby partikulární nejlepší řešení vždy ležela co nejvíce v levé části stromu, maximalizuje se tím počet odříznutých nepotřebných větví stromu.

Iterativní prohledávání vychází z poznatku, že je-li určité řešení tím nejlepším ve hloubce *d*, je pravděpodobné, že bude nejlepším i ve hloubce *d+1*. Iterativní prohledávání začíná tedy ve hloubce *1*, nalezená řešení jsou ve stromu seřazena podle hodnot evaluační funkce, od nejlepšího k nejhoršímu. Algoritmus inkrementuje hloubku prohledávání a pokračuje stejným způsobem.

Ukončovací podmínkou algoritmu nemusí být nutně maximální hloubka prohledávání, je možné použít i časový limit. Při použití omezení časem je možné použít výsledek prohledávání z hloubky *d-1*, vyprší-li časový limit pro prohledávání v hloubce *d*.

Pseudokód vypadá následovně:

```
iterativniProhlubovani(){
      int ohodnoceni;
      Tah t[] = generujTahy();
      Cas cas; /* cas pred spustenim algoritmu
      For(int hloubka = 1; ; hloubka++){
            ohodnoceni = AlfaBetaRazeni(hloubka, t); /* vylepseni
            If(aktualniCas – cas > limit){ /*vyprseni limitu
                  break;
            }
      }
      return ohodnoceni;
}
```

Pseudokód 4.4: Algoritmus iterativního prohlubování

V tomto pseudokódu je použit upravený algoritmus AlfaBeta, který během prohledávání zároveň řadí nalezené tahy podle výsledného ohodnocení pro urychlení příští iterace. Nabízí se otázka, zda není ztrátou výkonu, je-li provedeno prohledávání pro každou z hloubek *1..n*, namísto *n*. Tabulka 4.2 obsahuje počty prohledávaných uzlů stromu pro jednotlivé hloubky vyhledávání. Je uvažován začátek partie, počet možných tahů pro každou pozici je tedy o 10 až 15 tahů nižší, než ve střední fázi partie [8].

Hloubka prohledávání	Počet uzlů
1	22
2	97
3	736
4	3 586
5	32 193

Tabulka 4.2: Počet prohledaných uzlů na jednotlivých úrovních stromu

Součet všech pomocných uzlů je roven *4441*, což tvoří přibližně 13,8% uzlů poslední úrovně. Nejedná se tedy o markantní zvýšení doby výpočtu.

4.5 Hledání klidu

Účelem hledání klidové pozice je vyřešit problém horizontu zmíněný v kapitole 2. V šachových algoritmech vycházejících z MinMaxu existuje nebezpečí zisku zkreslených výsledků hledání. Příčinou zkreslení je ohodnocení pozice vzniklé po odehrání zásadního tahu, který se nachází v maximální hloubce vyhledávacího stromu. Jinými slovy, do ohodnocení je započítáno např. sebrání figury či šach, není ale uvažována reakce protihráče. Pro vyřešení tohoto problému je zapotřebí nalézt takovou pozici, kde nehrozí žádné další šachování nebo výměna materiálu. Hledá se tedy tzv. klidová pozice.

Hledání klidové pozice je poměrně těžkým úkolem. Je-li koncový uzel vyhledávacího stromu představitelem tahu, jehož provedením byla sebrána figura, je zapotřebí z tohoto uzlu prohloubit prohledávání. Prohlubování stromu může způsobit značné zpomalení celého algoritmu, obzvláště pro pozice, ve kterých je napadeno mnoho figur.

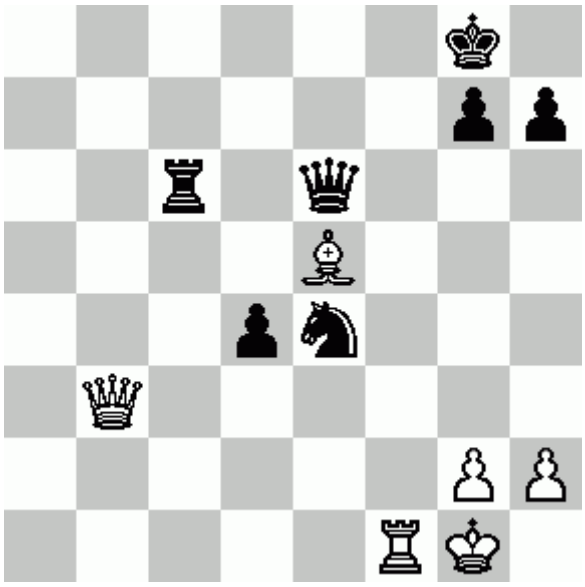
Dalším problémem je možnost šachování. Stejně jako u výměny figur by bylo zapotřebí prohloubit vyhledávání do takové míry, kde by se nenaskytovala žádná příležitost vystavení krále šachu. To ale není vždy možné, jelikož v šachách je při příhodných okolnostech možné dosáhnout nekonečné posloupnosti po sobě jdoucích šachů. Bylo by tedy zapotřebí rozlišovat účelné a bezúčelné šachování, k čemuž by bylo potřeba provést další prohledávání, které by dobu výpočtu jen zhoršilo. Možným řešením je zavedení horní hranice míry prohloubení vyhledávacího stromu, a to jak pro výměny figur, tak pro šachování krále.

4.6 Pravděpodobné tahy

Každý zkušenější hráč šachu je schopen pro jakoukoli pozici na první pohled určit několik tahů, které připadají v úvahu. Kdyby šachový engine měl schopnost vyčlenit tři nejpoužitelnější tahy pro každou pozici, obsahoval by vyhledávací strom o hloubce 12 pouhých 531 441 koncových pozic. Algoritmus MinMax tento počet v průměrném případě překročí již ve hloubce 4.

Doposud uvedené algoritmy jsou reprezentanty *strategií typu A*, tedy takových strategií, které berou v úvahu všechny dostupné tahy za cenu nízké hloubky vyhledávání. Generátory pravděpodobných tahů patří do třídy *strategií typu B*.

Pravděpodobné tahy je obecně obtížné vyčlenit. Největšími adepty jsou tahy, které vedou k sebrání figury, šachu, pokrytí napadené figury, ústupu s napadenou figurou, centralizace figury... Čím více je takových tahů uvažováno a následně začleněno do procesu vyhledávání, tím více je zpomalen proces hledání. Naopak, při zúžení výběru pravděpodobných tahů roste risk, že nebude nalezeno správné řešení. Příklad takové situace je zobrazen na Obr. 4.4, převzato z [9].



Obrázek 4.4: Úskalí generování pravděpodobných tahů

V této pozici je bílý na tahu. Generátor pravděpodobných tahů by nejspíše nabídl tahy jako sebrání černé dámy a následný zisk pěšce či odsunutí dámy na jiné pole, jelikož je momentálně napadena. Nicméně, správným tahem je *Bd6*, tedy střelec na pozici D6**.** Tento tah vede k nevyhnutelnému matu. Sebere-li černý dámu, dostává mat věží. Vezme-li černý střelce věží, *Qb8*, tedy dáma na B8, vede také k matu. Odpoví-li černý sebráním střelce koněm, získává bílý dámu a mat je také nevyhnutelný.

Při použití generátorů pravděpodobných tahů je tedy získána vyšší hloubka prohledávání za cenu vyřazení mnoha tahů bez záruky, že bude vždy nalezeno nejlepší možné řešení. Bezpečnějším využitím generátoru tohoto typu by bylo jeho začlenění do algoritmu AlfaBeta ořezávání s tím, že by byly generátorem navrhované tahy prozkoumány jako první.

5 Implementace

5.1 Programovací jazyk, uživatelské rozhraní

Šachový program je realizován v programovacím jakyku Java. Osobně tento jazyk preferuji. K vývoji jsem použil prostředí Netbeans IDE 7.2.1, dostupné z [10]. Projekt byl vytvářen ve verzi JDK 1.7.0, dostupné z [11].

Ke tvorbě uživatelského rozhraní jsem použil komponenty knihovny Java Swing. Oknu je nastavena fixní velikost 800x600 pixelů. Kromě samotné hrací plochy obsahuje uživatelské rozhraní zápis jednotlivých tahů partie. Rozhodl jsem se použít zjednodušenou notaci, každý tah je zaznamenán ve formátu figura, cílové pole. Figury jsou značeny následovně: Král – K (King), královna – Q (Queen), věž – R (Rook), střelec – B (Bishop), kůň – N (kNight). Tah pěšcem obsahuje pouze cílovou pozici bez speciální značky pro figuru. Tah královny na pole E7 je tedy zapsán takto: *Qe7*. Zápis partie je možné pomocí menu uložit do textového souboru. Menu dále obsahuje funkce pro restart partie a výměnu stran. Nad hracím polem se nachází panel indikující, který hráč je na tahu. Ikony figur jsou převzaty z [12]. Další informace jsou zmíněny v uživatelské příručce.

5.2 Reprezentace

Jako reprezentaci šachovnice jsem se rozhodl zvolit již zmíněnou variantu 10x12. Tato reprezentace poskytuje dobrý přehled o rozestavení figur na šachovnici. Další výhodou je usnadnění definování možných tahů jednotlivých figur díky okrajům okolo hrací plochy.

Jednotlivá pole reprezentace mohou nabýt následujících hodnot, viz Tabulka 5.1:

Obsah pole	**Bílý**	**Černý**
Pěšec	1	-1
Kůň	2	-2
Střelec	3	-3
Věž	4	-4
Dáma	5	-5
Král	6	-6
Prázdné pole	0	
Pole mimo šachovnici	-1	

Tabulka 5.1: Hodnoty polí reprezentace šachovnice

Dosáhne-li pěšec soupeřovy základní řady, je automaticky proměněn v dámu. Společně s obsahem šachovnice je zapotřebí uchovávat a aktualizovat dvě další informace: právo na rošádu a právo na braní mimochodem. Kromě matice 10x12 jsou zavedena dvě pole. První

pole o velikosti 6 prvků uchovává informaci o tom, zda bylo v průběhu partie taženo králem, popř. věžemi. Rošádu je možné provést pouze v případě, nebylo-li s figurami, mezi kterými je rošáda prováděna, doposud taženo. Druhé pole s rozměrem 16 prvků slouží k indikaci možnosti braní mimochodem. Tato informace musí být aktualizována po každém tahu, jelikož brát mimochodem je možné pouze v tahu bezprostředně následujícím po pohybu pěšcem o dva kroky.

5.3 Použité třídy a metody

5.3.1 ChessEngine.java

Tato třída má na starosti vytváření a kontrolu šachových tahů.

`boolean isLegal(short fromX, short fromY, short toX, short toY)`
Klíčová metoda, která zjistí, zda je předaný tah v souladu se šachovými pravidly. V této metodě není kontrolováno vystavení vlastního krále šachu po provedení tahu (a tedy nemožnost provedení tahu).

`boolean makeMove(short fromX, short fromY, short toX, short toY)`
Jedná se o provedení vlastního tahu. Je-li tah pseudo-legální, je proveden. Pokud je král hráče, který právě odtáhl, vystaven šachu, je pozice navrácena do původního stavu a tah označen jako neproveditelný.

5.3.2 ChessBoard.java

Jedná se o reprezentaci hry samotné. Třída mimo jiné zajišťuje započetí nové hry či výměnu stran hráčů.

`void moveFigure(ChessSquare from, ChessSquare to)`
Zde již jsou používány instance třídy *ChessSquare*, které reprezentují jednotlivé čtverce grafické šachovnice. Po zvolení tahu hráčem je informace předána této metodě, která za pomoci instance třídy *ChessEngine* rozhodne, zda je možné zadaný tah provést. Není-li hráč na tahu, provede tato metoda tah, který je umělou inteligencí označen jako nejlepší.

5.3.3 ChessGameUI.java

Zde probíhají výpočty spojené s hledáním nejlepšího tahu umělé inteligence.

```
ChessMove findNextMove(short[][] matrix,
     short[] castlingPrivileges,short[] enPassantPrivileges,
     boolean whiteOnTurn)
```

Tato metoda je implementací vyhledávacího algoritmu. Jsou jí předávány všechny potřebné informace k dané pozici, tedy jak rozestavení figur na šachovnici, tak práva na rošádu, práva na braní mimochodem a barva hráče, který je momentálně na tahu. Za pomoci metod `min()` a `max()` je následně určen výsledný tah.

`int evalFunction(short[][] matrix, boolean whiteOnTurn)`
Ohodnocovací funkce. Pro předanou pozici je vypočteno ohodnocení, které je za průběžného porovnávání s ohodnoceními ostatních pozic použito k výběru nejlepšího dostupného tahu.

5.4 Určení konce partie

Po provedení legálního tahu je zkontrolováno, zda má hráč, který je na řadě, k dispozici alespoň jeden tah. Není-li možné provést žádný tah, je vyhlášen šachmat v případě, stojí-li král tohoto hráče v šachu. Pokud není král v šachu a zároveň neexistuje legální tah, je partie označena za remízu, neboli pat. Po každém tahu je také kontrolováno, zda alespoň jeden z hráčů vlastní matící materiál. Není-li tomu tak, je vyhlášena remíza. Matícím materiálem se rozumí kombinace figur, kterými je možné dát soupeři mat. Matícím materiálem není kombinace král a střelec, král a kůň, či král a dva koně.

5.5 Algoritmus pro umělou inteligenci

Výběr tahů je prováděn algoritmem MinMax. Výhodou MinMaxu je jeho jednoduchost a přehlednost. I přes vysoký faktor expanze vyhledávacího stromu v závislosti na hloubce vyhledávání (na každý půltah se zvýší počet zkoumaných pozic přibližně čtyřicetkrát) poskytuje MinMax při hloubce čtyř půltahů dostatečnou odezvu.

5.6 Ohodnocovací funkce

Na začátku partie je ohodnocovací funkce rovna 0. Snahou bílého hráče je tuto hodnotu navyšovat, u černého je tomu naopak. Započítáním určitého faktoru se rozumí upravení celkového ohodnocení přičtením či odečtením hodnoty odpovídající danému faktoru.

V rámci ohodnocovací funkce jsem zohlednil následující aspekty: materiálovou rovnováhu, poziční ohodnocení, pohyblivost figur, napadení figur, postup pěšců a matovou (či patovou) pozici.

5.6.1 Materiálová rovnováha

Na materiál je v ohodnocovací funkci kladen velký důraz. V šachové partii všeobecně platí, že materiálová převaha je základním kamenem k vítězství. Jednotlivé hodnoty figur jsem zvolil následovně, viz Tabulka 5.2:

Figura	Hodnota
Pěšec	10
Kůň	30
Střelec	30
Věž	48
Královna	95

Tabulka 5.2: Materiálové konstanty pro ohodnocovací funkci

5.6.2 Postavení vůči středu šachovnice

Čím blíže středu šachovnice se figura nachází, tím více roste její potenciální dopad na partii. Z tohoto důvodu jsem zavedl matici obsahující koeficienty bonusů, které jsou přičítány k celkovému ohodnocení, viz Tabulka 5.3. Bonusy závisí na vzdálenosti figury od centra šachovnice. Konkrétní hodnota bonusu je získána vynásobením koeficientu polovinou materiálové konstanty příslušné k dané figuře.

0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2
0,2	0,4	0,4	0,4	0,4	0,4	0,4	0,2
0,2	0,4	0,6	0,6	0,6	0,6	0,4	0,2
0,2	0,4	0,6	0,8	0,8	0,6	0,4	0,2
0,2	0,4	0,6	0,8	0,8	0,6	0,4	0,2
0,2	0,4	0,6	0,6	0,6	0,6	0,4	0,2
0,2	0,4	0,4	0,4	0,4	0,4	0,4	0,2
0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2

Tabulka 5.3: Matice konstant pro poziční hodnocení

Pro krále není matice pozičního hodnocení využívána, prostředek šachovnice není pro krále bezpečná pozice. Naopak, nachází-li se král v obranném seskupení rošády, je přičten bonus k celkovému ohodnocení. Královna získává poziční bonusy stejným způsobem jako ostatní figury s tím rozdílem, že tento bonus není uvažován, nemá-li daný hráč vyvinuty všechny koně a střelce. Důvodem je prevence brzkého opuštění základní linie dámou.

5.6.3 Pohyblivost figur, napadení figur

Celkové ohodnocení je dále upraveno o 1 za každý dostupný tah, který má daná strana k dispozici, a o 2 za každou figuru, kterou tato strana napadá.

5.6.4 Postup pěšců

Čím je pěšec dále od své základní linie, tím více hrozí jeho proměna v královnu. Navíc, udržitelný pěšec v nepřátelských řadách může způsobit mnoho nepříjemností. Zavedl jsem pole obsahující ohodnocovací bonusy pro postupivší pěšce, viz Tabulka 5.4.

Obsah pole	**8. řada bonus**	**2. řada bonus**	**3. řada bonus**	**4. řada bonus**	**5. řada bonus**	**6. řada bonus**	**7. řada bonus**	**8. řada bonus**
Bílý pěšec	-	0	1	2	3	5	10	-
Černý pěšec	-	-10	-5	-3	-2	-1	0	-

Tabulka 5.4: Bonusy pro postup pěšců

5.6.5 Konec partie

Šachmat bílého je hodnocen jako -1000, remíza jako 0, mat černého jako 1000. Algoritmus je tedy schopen za pomoci ohodnocovací funkce nalézt mat, který leží do čtyř půltahů od aktuální pozice.

5.7 Možná rozšíření ohodnocovací funkce

Následující faktory jsem nezahrnul do rovnice pro ohodnocovací funkci. Důvodem je obtížnost určení správných hodnot, kterými jednotlivé šachové faktory ovlivní celkovou hodnotu pozice. Při vyšším množství ohodnocovacích kritérií vzniká riziko, že by tato kritéria převážila základní šachové cíle. Jinými slovy, mohlo by dojít k bezdůvodnému obětování figury pro získání nepatrné poziční výhody. K zahrnutí dalších kritérií by bylo zapotřebí důkladnější analýzy šachové hry.

5.7.1 Pozice věží

Věže vetšinou přichází na řadu v pozdější fázi hry. Hráč by se měl snažit umisťovat své věže na otevřené sloupce či na předposlední řady šachovnice. Faktor otevřeného sloupce je pokryt bonusem za pohyblivost figur. Věž v sedmé řadě (za bílého), resp. v druhé řadě (za černého), je hrozbou pro protivníka, jelikož vě většině případů napadá základní linie pěšců, či „odřezává“ protivníkova krále od zbytku šachovnice. Zdvojené věže zvyšují údernou sílu věží.

5.7.2 Svázanost figur

Svázanost figur, neboli vazba, je šachový termín označující situaci, kdy je pohyblivost jedné figury omezena, jelikož odtažením této figury by byla vystavena napadení jiná, obvykle cennější, figura. Zájmem hráče je takové vazby vytvářet na soupeřových figurách a zároveň jim zabraňovat na figurách vlastních.

5.7.3 Napadení polí, která jsou v bezprostřední blízkosti krále

Napadá-li hráč prostor přilehlý k soupeřovu králi, zvyšuje tím potenciální hrozbu vítězného útoku. Soupeř bývá nucen přeskupit své figury tak, aby byl schopen bránit své linie.

5.7.4 Figury na polích opačné barvy od soupeřova střelce

Vlastní-li soupeř pouze jednoho střelce, je strategické přeskupit figury na pole opačné barvy, než je soupeřův střelec. Soupeřův střelec tak ztrácí možnost zasáhnout do partie.

5.7.5 Zdvojené pěšce, izolované pěšce

Pěšci jsou zpravidla nejsilnější, když tvoří obrannou formaci, kde se kryjí navzájem. Jak zdvojené pěšce, tak i izolované pěšce tuto formaci narušují, je často obtížné takové pěšce udržet, mají tedy záporný efekt na hodnotu ohodnocovací funkce.

6 Testování

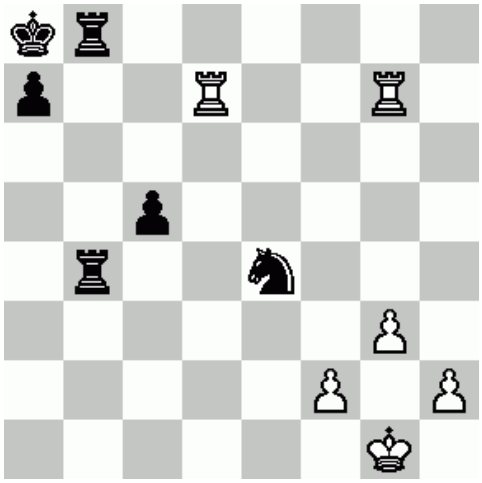
Funkčnost programu jsem testoval na úlohách převzatých z [13] a [14]. Šachovému enginu jsem nastavil testovací pozice určené pro začátečníky a pokročilé a pozoroval jeho reakci. Pro vygenerování vizualizace pozic jsem použil WiseBoard Editor [15]. Veškeré testy byly prováděny pod operačním systémem Windows 7, za použití dvoujádrového procesoru o frekvenci 2,8 GHz na každém jádru. Java heap space byl nastaven na 128 MB.

6.1 Poziční testy

Účelem pozičních testů je ověřit funkčnost vytvořeného programu. Je prověřováno, zda byla šachová pravidla správně naimplementována. Program je testován, zda je schopný správně zareagovat v základních šachových úlohách.

6.1.1 Pozice č. 1

V pozici č. 1 je na tahů bílý, nabízí se mat jedním tahem.

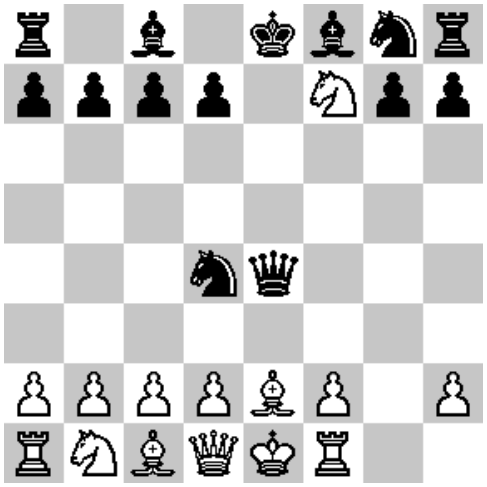


Obrázek 6.1: Testovací pozice č. 1

Program při odezvě *650 ms* odpověděl tahem *Ra7*, který znamenal mat. Tento tah se shoduje s navrhovaným tahem v předloze.

6.1.2 Pozice č. 2

V pozici č. 2 se nabízí černému mat jedním tahem, test je zaměřen na využití vazby střelce.

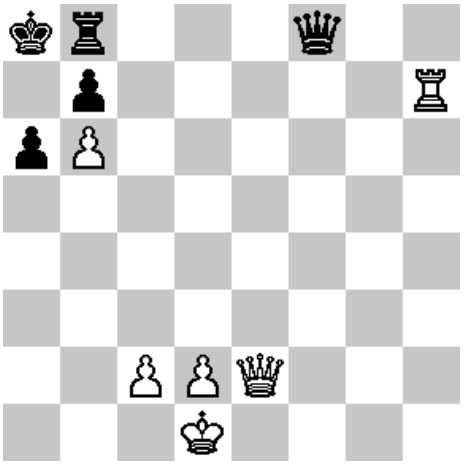


Obrázek 6.2: Testovací pozice č. 2

Program nalezl správný tah, tedy *Nf3,* za *1750 ms*.

6.1.3 Pozice č. 3

Bílý je na tahu, úkolem je nalézt dvoutahovou kombinaci vedoucí k matu.

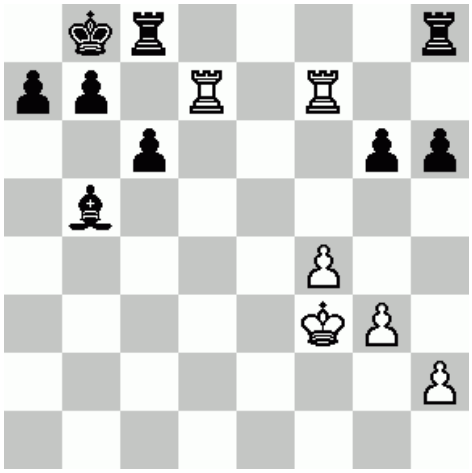


Obrázek 6.3: Testovací pozice č. 3

Řešení *Qa6* bylo nalezeno za *1050 ms*, po jediné možné reakci černého *a6* byl matící tah *Ra7* zvolen po *350 ms*.

6.1.4 Pozice č. 4

Bílý může dosáhnout matu posloupností tří správných tahů.

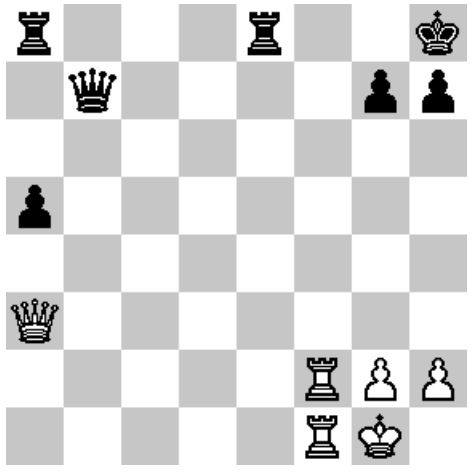


Obrázek 6.4: Testovací pozice č. 4

Engine nachází správnou posloupnost tahů, tedy *1 Rb7 Ka8, 2 Ra7 Ka7, 3 Rb7* v časech *560 ms, 480 ms a 550 ms.* Tahy černého krále jsou vynucené. Je nutné podotknout, že před provedením prvního tahu engine neměl propočítány tahy až do konce partie, nevěděl tedy, že jeho akce nakonec skončí matem. K provedení prvního tahu jej vedl jistý zisk dvou pěšců. V tomto případě tedy byl nalezen správný tah, jelikož byl spojen s materiálovým ziskem. Opačný případ bude nastíněn v následující testovací pozici.

6.1.5 Pozice č. 5

Bílý na tahu, mat ve třech tazích.

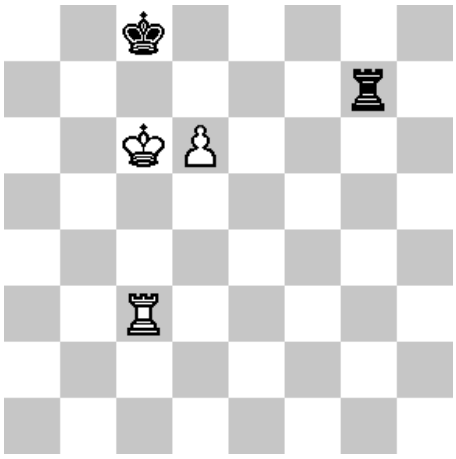


Obrázek 6.5: Testovací pozice č. 5

Program nalezl tah *Qf3* v čase *2800 ms*. Tento tah není optimálním řešením situace. Správnými tahy jsou *Qf8* nebo *Rf8*. Algoritmus nenašel nejlepší možné řešení, protože hloubka prohledávání je rovna čtyřem půltahům. K řešení této pozice je zapotřebí propočet do hloubky šesti půltahů.

6.1.6 Pozice č. 6

Černý na tahu, vynucení patu

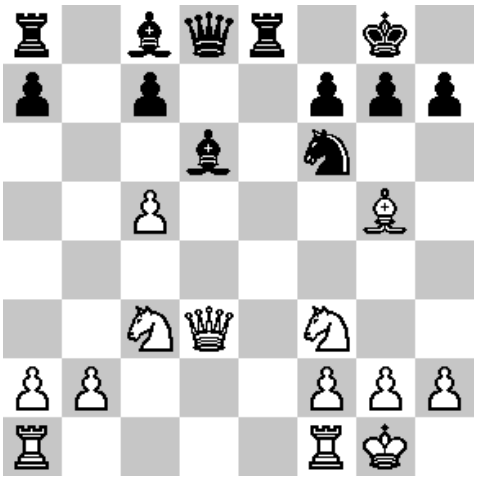


Obrázek 6.6: Testovací pozice č. 6

V tomto případě je správné řešení nalezeno za *120 ms*. *Rc7* je správným tahem, zareaguje-li bílý sebráním věže, nastává pat. Nesebere-li bílý věž, ztrácí věž vlastní a tím i celou partii.

6.1.7 Pozice č. 7

Následující pozice je zaměřena na zisk materiálové výhody. Černý je na tahu.

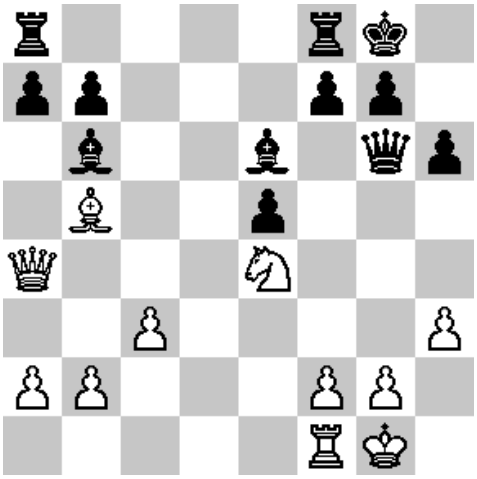


Obrázek 6.7: Testovací pozice č. 7

Program odpovídá správným tahem *Sh2* za *2050 ms*. V příštím tahu získává černý neodvratně královnu.

6.1.8 Pozice č. 8

V poslední testovací pozici je na tahu černý, je hledána vítězná kombince.



Obrázek 6.8: Testovací pozice č. 8

Správný tah, tedy *Sh3,* je nalezen za *1650 ms*. Černý střelec nemůže být sebrán z důvodu vazby na pěšce, mat je neodvratný, zahraje-li bílý *Ng3*, odpoví engine tahem *Dg3*, jelikož i druhý bílý pěšec se nachází ve vazbě.

Ukazuje se, že je program schopen správně zareagovat v šachových úlohách určených pro mírně pokročilé hráče, leží-li řešení do čtyř půltahů. Při složitějších úlohách není možné zaručit ideální řešení.

6.2 Testování průběhu partie

6.2.1 Zahájení

Je možné si povšimnout, že v začátcích partií volí program podobná zahájení. Důvodem je výběr tahů, který je založen na MinMaxu. Po aplikování ohodnocovací funkce je pokaždé vybrán takový tah, který vede k nejvyššímu ohodnocení. Nalezne-li MinMax dva tahy se stejným ohodnocením, je náhodně vybrán jeden z nich, čímž je snížena stejnorodost reakcí na vzniklé pozice.

6.2.2 Střední hra

Implementovaná ohodnocovací funkce je založena převážně na pravidlech, která platí pro střední hru partie. Není-li možné získat materiál, snaží se šachový motor alespoň zvýšit počet

polí, která kontroluje, popř. zvýšit míru napadení soupeřových figur, zatímco je vlastní král uložen v bezpečné pozici. Program nejvíce ztrácí v situacích, kde se nabízí rozsáhlé výměny. Z důvodu nízké hloubky prohledávání není program schopen propočítat výměny až do konce, proto tah, který se zpočátku jeví jako výborný, skončí ztrátou materiálu.

6.2.3 Koncová hra

Šachový motor není primárně na koncové hry nastaven. Nadále respektuje pravidlo materiálové výhody, je naprogramován i pro postup pěšců směrem k soupeřově základní linii. Nicméně, mnohá pravidla, která byla efektivní pro střední hru, přestávají platit. Nachází-li se mat v dosahu hloubky prohledávání, je bez problémů nalezen. Je-li zapotřebí více tahů, má program problémy dovést partii do úspěšného konce.

6.3 Nalezené problémy, možná řešení

6.3.1 Monotónní zahájení

Při opětovném hraní hry proti počítači si uživatel povšimne, že zvolí-li hráč stejné zahájení, zareaguje umělá inteligence ve většině případů stejným způsobem. Tím je uživatel do jisté míry ochuzen o rozmanitost šachové hry. Zdatnější hráč by mohl dokonce nalézt posloupnost tahů, která pokaždé vede k vítězství. Řešením tohoto nedostatku by bylo připojení databáze vhodných zahájení.

6.3.2 Efekt horizontu

Efekt horizontu způsobuje problémy v situacích, kdy se v partii nabízí určitý mezitah, který oddálí neodvratnou ztrátu. Tento efekt je zajisté nežádoucím jevem. K vyřešení tohoto problému by bylo zapotřebí rozšířit algoritmus MinMaxu tak, aby prohloubil prohledávání v případech, ve kterých byly prováděny výměny figur, či šachování krále.

6.3.3 Nedostatečná hloubka prohledávání

Jak již bylo zmíněno, nedostatečná hloubka propočtů je nejčastějším důvodem ztrát v partiích. Aby bylo možné zvýšit hloubku prohledávání, bylo by zapotřebí vylepšit vyhledávací algoritmus. Algoritmus AlfaBeta ořezávání společně s vyhledáváním klidu by mohl poskytnout přibližně dvojnásobnou hloubku prohledávání.

6.3.4 Neúplnost ohodnocovací funkce

Šachy jsou poměrně složitou hrou. Je obtížné nalézt obecná pravidla, která po zakomponování do ohodnocovací funkce povedou pro každou nastavší situaci k nalezení správného řešení. Po nalezení těchto pravidel, která je mnohdy možné uplatnit pouze pro konkrétní fáze partie, nastává problém přidělování váhy jednotlivým pravidlům. K tomu je zapotřebí jak hluboké analýzy šachové hry, tak četných testů.

6.3.5 Absence adaptace pro konce partií

Koncová hra je natolik odlišná od hry střední, že se v mnohých aspektech jedná o řešení zcela jiné úlohy. Bylo by zapotřebí vytvořit zcela nová pravidla pro ohodnocování, např. pravidlo opozice [16] či pravidlo čtverce [17]. Dále by bylo nutné nadefinovat, kdy by měla být tato pravidla uplatněna. Značné vylepšení by představovalo přidání modulů pro matování, jelikož program momentálně neovládá základní postupy k dosažení matu.

7 Závěr

První část práce se zabývá počítačovým šachem v obecném smyslu. Popisuje rozdílnost v přístupu umělé inteligence k šachové partii od přístupu lidského faktoru.

Další částí práce je průzkum technik potřebných k realizaci funkčního šachového programu. Mezi základní pilíře šachového programu patří reprezentace šachovnice, generátor tahů, algoritmus výběru nejlepšího tahu a ohodnocovací funkce.

Reprezentace šachovnice je způsob, jakým jsou zaznamenány jednotlivé stavy partie. Účelem reprezentace šachovnice je zároveň připravit půdu pro generátor tahů, který z reprezentace přímo vychází.

Generátor tahů má na starosti vyčlenění takových tahů, které se shodují se šachovými pravidly. Generátor tahů je využíván jak při kontrole, zda je tah, který hráč odehrál, validní, tak i při hledání nejlepšího tahu ze strany umělé inteligence.

S generátorem tahů úzce souvisí algoritmus výběru tahů. Generátor připraví možné tahy pro danou pozici. Tyto tahy jsou následně algoritmem prohledávány. Důležitým faktorem je hloubka prohledávání, která určuje, kolik možných pozic partie bude prohledáno.

Šachový algoritmus při výběru nejlepšího tahu využívá ohodnocovací funkci. Ohodnocovací funkce je rovnice, která aplikováním obecných šachových principů pro jakoukoli pozici stanoví, která strana se nachází ve výhodě. Pomocí ohodnocovací funkce je zároveň možné vyjádřit míru této výhody. Cílem algoritmu je tedy tuto výhodu maximalizovat.

Práce se dále zabývá implementací vlastního šachového programu. V této části je zmíněna zvolená reprezentace šachové úlohy. Součástí implementační části je popis návrhu vlastní ohodnocovací funkce, spolu s výpisem možných rozšíření této funkce.

Závěrečná část bakalářské práce je zaměřena na testování vytvořeného programu. Vzniklý program je schopen bezchybně řešit šachové úlohy, jejichž řešení leží v nastavené hloubce vyhledávání. Celkový průběh umělou inteligencí odehraných partií by se dal přirovnat k výkonům mírně pokročilého hráče. Jednotlivé nedostatky šachového programu jsou zhodnoceny, pro každý nedostatek je navržen alespoň jeden způsob budoucího řešení.

8 Použité šachové termíny

Braní mimochodem: neboli *en passant* je šachové pravidlo, které hráči umožňuje vlastnímu pěšci sebrat pěšce soupeřova, pokud soupeř v minulém tahu táhl pěšcem ze své základní linie o dvě pole a hráčův pěšec napadá pole, přes které bylo soupeřovým pěšcem v rámci dvojtahu taženo

Koncová hra: pozdní část partie, kdy se na šachovnici nachází méně než 30% původních figur

Izolovaný pěšec: pěšec, okolo kterého se v ohruhu jednoho sloupce nenachází žádný další pěšec stejné barvy

Materiál: figury, šachové kameny přítomné na šachovnici

Mat: konec partie, hráčův král se nachází v šachu, který již nelze odvrátit, tento hráč prohrává

Oběť: přenechání figury za účelem poziční výhody

Otevřený sloupec: slopec, na kterém se nenachází žádné figury

Pat: konec partie, hráč, který je na tahu, nemá k dispozici žádný tah, jeho král zároveň není vystaven šachu, taková partie je vyhodnocena jako nerozhodná

Představení figury: předsunutí vlastní figury před figuru druhou za účelem zamezení soupeřova útoku na tuto figuru

Půltah: totéž jako tah, používáno k zdůraznění, že je míněn jeden konkrétní tah, nikoli tah včetně protihráčovy reakce

Rošáda: šachové pravidlo umožňující vytvoření obranné formace pomocí krále a věže, je uplatnitelné pouze v případě, že nebylo kameny, pomocí kterých je rošáda prováděna, doposud taženo, a není-li daný král (včetně polí ve směru pohybu krále v rámci rošády) vystaven šachu

Střední hra: fáze partie, kdy jsou již figury vyvinuty

Šach: napadení hráčova krále, hráč je nucen toto napadení bezprostředně odvrátit

Šachový motor (engine): jiné vyjádření pro šachový program

Vývin figur: přesun figur ze základní řady za účelem zvýšení jejich působnosti

Zahájení: počáteční fáze partie, obvykle provázena hromadným vývinem figur

Zdvojené pěšce: dva pěšce stojící ve stejném sloupci, ve většině případů se jedná o nevýhodu

Zdvojení věží: dvě věže stojící na stejném sloupci, většině případů se jedná o výhodu

9 Zdroje

[1] STEINWENDER, D., FRIEDEL, F. A. *Schach am PC*. Markt & Technik, Buch- und Software-Verlag GmbH, Haar bei München, 1995. ISBN 3-87791-522-1

[2] The About Group. *Understanding Chess Ratings* [online]. [cit. 12. 5. 2013]. Dostupné z: http://chess.about.com/od/chesscommunities/qt/Ratings.htm

[3] 1000 Projects. *Offset Representation of the Chess Board* [online]. [cit. 13. 5. 2013]. Dostupné z: http://1000projects.org/offset-representation-of-the-chess-board.html

[4] 1000 Projects. *10X12 Chess Game Grids Offset Representation* [online]. [cit. 13. 5. 2013]. Dostupné z: http://1000projects.org/10x12-chess-game-grids-offset-representation.html

[5] The Chess Programming Wiki. *Bitboard Board-Definition* [online]. [cit. 15. 5. 2013]. Dostupné z: https://chessprogramming.wikispaces.com/Bitboard+Board-Definition

[6] University of California, Irvine. *An Introduction to Artificial Intelligence* [online]. [cit. 15. 5. 2013]. Dostupné z: http://www.ics.uci.edu/~pazzani/171.html

[7] The Chess Programming Wiki. *Alpha-Beta* [online]. [cit. 18. 5. 2013]. Dostupné z: http://chessprogramming.wikispaces.com/Alpha-Beta

[8] Mediocre Chess. *Iterative Deepening* [online]. [cit. 18. 5. 2013]. Dostupné z: http://mediocrechess.blogspot.cz/2007/01/guide-iterative-deepening.html

[9] FREY, P. W. *Chess Skill in Man and Machine.* Springer-Verlag, New York, 1983. ISBN 0-387-90790-4

[10] NetBeans IDE. Oracle Corporation. Dostupné z: https://netbeans.org/downloads/

[11] Java Platform. Oracle Corporation. Dostupné z: http://www.oracle.com/technetwork/java/javase/

[12] deviantART. *Chess Piece Brushes* [online]. [cit. 8. 1. 2013]. Dostupné z: http://punkdoutkittn.deviantart.com/art/Chess-Piece-Brushes-90810700

[13] PLISKA, K. *Učebnice šachu pro samouky - začátečníci.* PLISKA, soukromé nakladatelství,Frýdek-Místek, 1999. ISBN 80-85232-42-1

[14] PLISKA, K. *Učebnice šachu pro samouky – středně pokročilí.* PLISKA, soukromé nakladatelství,Frýdek-Místek, 1999. ISBN 80-85232-43-X

[15] Apronus. *WiseBoard Chess Board Editor* [online]. [cit. 12. 6. 2013]. Dostupné z: http://www.apronus.com/chess/wbeditor.php

[16] Wikipedia, The Free Encyclopedia. *Opposition (chess)* [online]. [cit. 19. 6. 2013]. Dostupné z: http://en.wikipedia.org/wiki/Opposition_(chess)

[17] Chess Game Strategies. *The Rule of the Square* [online]. [cit. 19. 6. 2013]. Dostupné z: http://www.chess-game-strategies.com/rule-of-the-square.html

[18] Wikipedie, Otevřená encyklopedie. *Šachy* [online]. [cit. 24. 6. 2013]. Dostupné z: http://cs.wikipedia.org/wiki/Šachy

[19] NCZOnline. *How to install Apache Ant on Windows*. Dostupné z: http://www.nczonline.net/blog/2012/04/12/how-to-install-apache-ant-on-windows/

Přílohy

10 Uživatelská příručka

10.1 Spuštění programu

Program je spustitelný ze souboru *ChessGame.jar*, který se nachází v kořenovém adresáři přiloženého CD. Ke spuštění je zapotřebí mít nainstalován JDK (Java Development Kit) [11] ve verzi nejméně 7.0. Doporučované operační systémy jsou Windows XP a Windows 7.

Program je zároveň možné sestavit pomocí skriptu *build.xml*, který se nachází v adresáři *ChessGame*. K sestavení je zapotřebí mít nainstalovaný Apache Ant. Návod k instalaci a nastavení Antu je dostupný z [19]. Následně stačí zkopírovat adresář *ChessGame* z přiloženého CD na disk, navigovat příkazovou řádku do tohoto adresáře a spustit sestavení programu příkazem *„ant“*.

Zdrojové soubory se nachází v *ChessGame/src*.

10.2 Po spuštění

Po spuštění se zobrazí hrací plocha, je spuštěna nová hra. Hráč je dosazen do pozice bílého hráče, přeje-li si tuto pozici změnit, může tak učinit v menu funkcí *Switch Sides*.

10.3 Hraní hry

K provedení tahu je zapotřebí nejdříve označit figuru vlastní, poté určit, kam by měla být přesunuta. Tah je proveden pouze, shoduje-li se se šachovými pravidly. Šachová pravidla včetně pravidel pro pohyb jednotlivých figur jsou uvedena v příloze textu bakalářské práce. Horní panel indikuje, který hráč je právě na tahu.

10.4 Zápis partie

Uživatelské rozhraní poskytuje zjednodušený zápis partie. Jednotlivé sloupce šachovnice jsou v zápisu značeny v řadě *a,b,c,d,e,f,g,h*, řady jsou v zápisu číslovány *1,2,3,4,5,6,7,8*. Tento zápis je možné kdykoli uložit do textového souboru pomocí menu.

10.5 Nová hra

Novou hru je možné začít pomocí menu, popř. opětovným spuštěním programu.

11 Stručná pravidla

Pravidla jsou převzata z [18].

11.1 Základní informace

Šachovou soupravu tvoří šachovnice a dvě sady kamenů – bílých a černých. Šachovnice je čtvercová deska o velikosti 8×8 polí, střídavě tmavých (označovaných jako černá pole) a světlých (bílá pole), která během hry leží mezi hráči na stole tak, že každý má v rohu po své pravé ruce bílé pole.

11.2 Zahájení partie

Před začátkem hry (šachové partie) se určí, který z hráčů bude hrát bílými kameny. Tento hráč se označuje jako bílý a jeho soupeř jako černý. Kameny se postaví do výchozího postavení.

Bílý partii zahájí, a poté se hráči v tazích pravidelně střídají, nikdo se svého tahu nemůže vzdát. Tah každého hráče sestává z přesunutí jednoho kamene v souladu s pravidly (výjimkou je rošáda, při které se současně přesune král i věž). Žádným tahem se nesmí kámen přesunout na pole, na kterém již je jiný kámen stejné barvy. Tah na pole s kamenem soupeře se nazývá braní; soupeřův kámen je takovým tahem odstraněn ze šachovnice.

11.3 Pohyb figur

Každý druh kamenů se pohybuje jiným způsobem, všechny s výjimkou jezdce se posouvají přímou čarou (po řadách, sloupcích nebo diagonálách) tak, že nesmějí žádný jiný kámen „přeskočit“.

11.3.1 Král

Král se pohybuje o jedno pole v libovolném směru, tzn. na některé z osmi sousedních polí (včetně čtyř sousedících pouze rohem).

Druhým způsobem tahu krále je tzv. rošáda. Pokud se král a některá z věží ještě nepohnuli, král se přemístí o dvě pole směrem k věži a věž přes krále na pole, které král právě přešel. Všechna mezilehlá pole musejí být volná, král nesmí stát před rošádou v šachu a nesmí přejít přes pole ohrožené soupeřem. Žádným svým tahem se král nesmí dostat na ohrožené pole, tj. na pole, na které by se v příštím tahu mohl přesunout soupeřův kámen a tím krále brát.

11.3.2 Dáma

Dáma se pohybuje po sloupcích, řadách nebo diagonálách o libovolný počet polí. Jako taková je nejsilnějším kamenem.

11.3.3 Věž

Věž se pohybuje po řadách a sloupcích.

11.3.4 Jezdec

Jezdec se pohybuje skoky ve tvaru písmene L (dvě pole rovně a jedno stranou, respektive jedno rovně a dvě stranou) bez ohledu na to, stojí-li na okolních polích nějaké kameny. Jezdec při každém skoku změní barvu pole, na kterém stojí: z bílého pole se dostane vždy na černé a naopak.

11.3.5 Střelec

Střelec se pohybuje po diagonálách. Jelikož ty mají na šachovnici stejnou barvu, střelec nikdy nezmění barvu pole, na kterém stojí; proto se v každé sadě hovoří o střelci bělopolném a černopolném.

11.3.6 Pěšec

Pěšec se může posunout o jedno pole vpřed, pokud je toto pole neobsazené (ze základního postavení se může posunout i o dvě pole vpřed, pokud jsou obě prázdná). Navíc může brát soupeřův kámen, který je na úhlopříčně sousedícím poli před pěšcem. Pokud pěšec ohrožuje pole, které soupeřův pěšec přeskočil tím, že z úvodní pozice postoupil o dvě pole, pak ho může tento pěšec vzít, jako by soupeř postoupil pouze o jedno pole. Tento tah, nazývaný braní mimochodem (nebo en passant), lze uskutečnit jen bezprostředně poté, co soupeř svým pěšcem takto táhl.

Pěšec, který postoupil na poslední pole desky (osmou, resp. první řadu), je ve stejném tahu odstraněn z desky a nahrazen na tomto poli dámou, věží, střelcem nebo jezdcem podle okamžité volby hráče (tzv. proměna). Působnost proměněného kamene je okamžitá, tzn. může například dát šach nebo se účastnit matování.

11.4 Šach, konec hry

Pokud hráč nějakým tahem napadne soupeřova krále, tzn. táhne tak, že by příštím tahem mohl krále brát, říkáme, že soupeřovi dal šach. Pokud taková situace nastane, soupeř je povinen hrát tak, aby tuto hrozbu odvrátil. Pokud však neexistuje žádný takový tah, který by hrozbu braní krále odvrátil, znamená to mat, konec hry, vítězství hráče, který takto soupeřova krále napadl. Hra končí vítězstvím také v případě, že se druhý hráč vzdá.

Nerozhodný výsledek, remíza, může nastat dohodou hráčů (jeden remízu nabídne a druhý ji přijme), a dále v situaci patu (hráč není v šachu, ale nemá k dispozici žádný dovolený tah), trojím opakováním stejné pozice, redukcí počtu kamenů tak, že zbylými kameny už nelze dosáhnout matu, a konečně jestliže oba hráči provedli 50 po sobě následujících tahů, aniž by přitom sebrali kámen soupeře nebo táhli pěšcem.

12 Ukázka zdrojového kódu

12.1 Kontrola legálnosti tahu

```
/**
 * Checks whether the inputted move is legal.
 * @param fromX original x position
 * @param fromY original y position
 * @param toX x destination
 * @param toY y destination
 * @return the legality of the move: yes/no
 */
public boolean tryMove(short fromX, short fromY, short toX, short toY) {
        if(!isInBoard(fromX, fromY) || !isInBoard(toX, toY)){
            return false;
        }
        boolean flag = true;
        if(isLegal(fromX, fromY, toX, toY)){
            short tmp = currentPosition[toX][toY];
            currentPosition[toX][toY] = currentPosition[fromX][fromY];
            currentPosition[fromX][fromY] = 0;
            if(whiteOnTurn){
                    if(isWhiteKingInCheck()){ //the move is not legal if the
                              white is on turn and his king is in check
                     flag = false;
                }
            }
            else{
                if(isBlackKingInCheck()){  //the  move  is  not  legal  if  the
                              black is on turn and his king is in check
                     flag = false;
                }
            }
            currentPosition[fromX][fromY]  =  currentPosition[toX][toY];  //
                              rollback
            currentPosition[toX][toY] = tmp;
        }
        else {
            return false;
        }
        return flag;
    }
```

Ukázka zdrojového kódu 12.1: Test legálnosti ve třídě ChessEngine.java

12.2 Ohodnocovací funkce

```
/**
 * Evaluates the position.
 * @param matrix evaluated position
 * @param whiteOnTurn white/black on turn
 * @return evaluation
 */
 public int evalFunction(short[][] matrix, boolean whiteOnTurn){
        int eval = 0;
        eval += evalMaterial(matrix); //adds the material evaluation
        eval += evalPosition(matrix); //adds the positional evaluation
        calculator.setPosition(matrix);
        calculator.calculatePossibleMovesAndBalance();
        eval += calculator.getEvaluation(); //adds the possible moves
                                balance evaluation
        if(noMovePossible(matrix, whiteOnTurn)){
            if(whiteOnTurn && endingConditionsEngine.isWhiteKingInCheck())
                                { //black victory evaluation
                eval = -1000;
            }
            else if(!whiteOnTurn &&
                                endingConditionsEngine.isBlackKingInCheck())
                                { //white victory evaluation
                eval = 1000;
            }
            else{
                eval = 0; //draw evaluation
            }
        }
        return eval;
    }
```

Ukázka zdrojového kódu 12.2: Ohodnocovací funkce ve třídě ChessGameUI.java

12.3 Provedení tahu

```
/**
 * Makes a move.
 * @param from original square
 * @param to destination square
 */
private void moveFigure(ChessSquare from, ChessSquare to){
            if(engine.makeMove((short)from.getRow(), (short)from.getFile(),
                                (short)to.getRow(), (short)to.getFile()))
                                {//checks whether the move can be made
                playerTurn = !playerTurn;
                if(engine.getTurn()){
                    moves.addBlackMove(figureName(from.getFigure()) +
                              fileName(to.getFile()) +
                              String.valueOf(to.getRow() - 1)); //adds a
                              black move to the register
                }
                else{
                    moves.addWhiteMove(figureName(from.getFigure()) +
                              fileName(to.getFile()) +
                              String.valueOf(to.getRow() - 1)); //adds a
                              white move to the register
                }
                status.switchStatus();
                placeFigures(engine.getPosition()); // places the images on
                              the chessboard
                if(!playerTurn){ //makes the next move according to the
                              artificial intelligence
                    ChessMove move =
                              ui.findNextMove(arrayCopy(engine.getPosition()
                              ), engine.getCastlingPrivileges().clone(),
                              engine.getEnPassantPrivileges().clone(),
                              engine.getTurn());
                    ChessSquare fromSquare =
                              squares[move.getFromX()][move.getFromY()];
                    ChessSquare toSquare =
                              squares[move.getToX()][move.getToY()];
                    moveFigure(fromSquare, toSquare);
                }
        }
    }
```

Ukázka zdrojového kódu 12.3: Provedení tahu ve třídě ChessBoard.java