VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

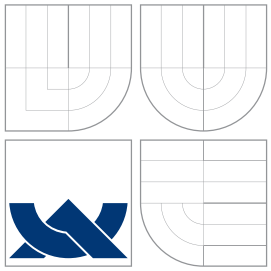
ŠACHY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

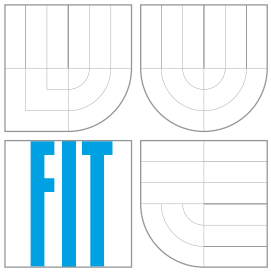
AUTOR PRÁCE
AUTHOR

VÍT VALSA

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ŠACHY
CHESS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÍT VALSA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN

BRNO 2009

Abstrakt

Hlavním úkolem bakalářské práce bylo vytvořit počítačovou hru šachy s grafickým uživatelským rozhraním a navrhnout a naprogramovat algoritmus šachového počítače schopného se učit. Také bylo zapotřebí vytvořit rozhrání pro manažer deskových her. Z počátku v textu pojednávám o šachových pravidlech a již existujících šachových programech a specializovaných šachových počítačích. Po tomto bloku následuje část zaměřená na algoritmus mini-max a z něj vycházející algoritmy, taktéž jejich možná vylepšení. Stěžejní částí práce je popis návrhu a realizace vytvořené aplikace. Dále popisuji ovládání a funkčnost aplikace. V závěru jsou zhodnoceny dosažené výsledky a popsány další možné směry vývoje programu.

Abstract

The main purpose of my bachelor's thesis was to create the chess computer game software with graphic user interface and to design and develop an algorithm of chess computer with learning ability. It was also necessary to create an interface to the board games manager. In the first part of my work I present the chess game rules and already existing chess game software and specialised chess computers. The next part discusses the mini-max algorithm and other derived algorithms and their possible improvements. The main part of the work deals in description of design and realization of the developed application. The user interface and functionality of the application is also described. In the conclusion the achieved results are summarized and the possible directions of further development of the application are described.

Klíčová slova

Šachy, umělá inteligence, grafické uživatelské rozhraní, JAVA, manažer deskových her, software, hardware, FIDE.

Keywords

Chess, artificial intelligence, grafic user interface, JAVA, manager board games, software, hardware, FIDE.

Citace

Vít Valsa: Šachy, bakalářská práce, Brno, FIT VUT v Brně, 2009

Šachy

Prohlášení

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

.....
Vít Valsa
19. května 2009

Poděkování

Chtěl bych poděkovat Ing. Jaroslavu Rozmanovi, za jeho vedení a za jeho ochotu a rady. Také děkuji za výborný přístup k vzájemné spolupráci.

Obsah

Kapitola 1

Úvod

Šachy jsou zajisté jednou z nejzajímavějších deskových her dvou hráčů. Mají jasně daná, poměrně jednoduchá a již delší dobu neměnná pravidla. Jedná se o hru, kde oba hráči mají úplnou informaci o tom, co má jejich soupeř k dispozici a do výsledku nezasahuje náhoda na rozdíl od většiny karetních her jako je například pokr. Díky těmto vlastnostem se šachy staly velmi oblíbenou metodou jak potrápit mozky přírodní i mozky umělé.

Po dlouhou dobu se lidé snaží vytvořit stroj, který by uměl hrát šachy. První pokus byl mechanický stroj z konce 18. století známý jako Turek. Jednalo se o figurku Turka, která hrála partie proti lidským protihráčům, nebo předváděla jezdcovou procházku, což je úkol ukládající hráči objet jezdcem celou šachovnici tak, aby na každém políčku stanul právě jednou. Jednalo se ovšem o velmi propracovaný podvod, jelikož zařízení ukrývalo skutečného šachistu. Více se o tomto stroji dozvíte na [10].

První seriózní pokusy se datují, až do století dvacátého, kde díky rozvoji počítačů sloužily šachy jako test jejich inteligence. Díky vývoji šachových algoritmů a zlepšování hardwaru začaly šachy postupně porážet jak úplné začátečníky, tak i pokročileší hráče, až se nakonec dostaly na úroveň mistrů. V roce 1997 se stalo to, co každý šachista považoval za nemožné, když DeepBlue porazil šachového velmistra Garryho Kasparova. Byla to pouze první vlaštovka, která přiletěla poněkud předčasně, avšak dnes už je zřejmé, že za několik málo desetiletí nebude člověk hře počítačů rozumět.

Oficiální zadaní práce znělo:

- Seznamte se s pravidly hry šachy a se stávajícími šachovými programy.
- Seznamte se s algoritmy minimax a alfa-beta používanými při programování her.
- Navrhněte program na hraní šachů, zamyslete se nad možností učení z databáze už odehraných her.
- Navržený program implementujte včetně grafického rozhraní a rozhraní pro Manažer pro deskové hry.

Tyto požadavky jsem se snažil splnit co nejlépe jsem mohl a pokoušel jsem se o doplnění pomocných funkcí pro usnadnění orientace ve hře šachy.

V následující kapitole stručně popisuji šachová pravidla, věnuji se již existujícím šachovým programům, jejich srovnání se specializovaným šachovým hardwarem a kombinací sofistikovaného šachového softwaru a hardwaru. Závěr kapitoly je věnován soutěžím pro umělé inteligence ve hře šachy.

Třetí kapitola je stručným shrnutím základních algortimů pro umělou inteligenci a jejich možným vylepšením, na kterou navazuje kapitola o samotném návrhu a realizaci projektu.

Následující pátá kapitola se zaměřuje na možnosti učení šachových počítačů obecně a je zde popsáno jakým způsobem je učení implementováno v tomto projektu.

Předposlední kapitola popisuje ovládání programu, všech jeho funkcí a možných nastavení. V závěru hodnotím dosažené výsledky a nastiňuji možnosti dalších možných směrů vývoje programu.

Kapitola 2

Exitující šachové programy a specializovaný hardware

V dnešní době existuje mnoho šachových programů, z nichž pouze několik málo se může srovnávat s nejlepšími světovými hráči v šachu. Pro pochopení umělé inteligene pro hru šachy stojí za to, alespoň stručně zmínit její pravidla.

2.1 Stručná šachová pravidla

Tato podkapitola byla převzata z [11].

Šachová souprava se skládá z šachovnice a dvou sad figur – bílých a černých. Šachovnice je čtvercová deska o velikosti 8x8 polí, střídavě tmavých (označovaných jako černá pole) a světlých (bílá pole), která v průběhu hry leží mezi hráči na stole, takovým způspbem, že v rohu po své levé ruce má každý hráč tmavé pole.

Před začátkem hry (Šachové partie) se určí, který z hráčů bude hrát bílými figurami. Tento hráč je označován jako bílý a jeho soupeř jako černý. Figury 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 jedné figury v souladu s pravidly (výjimkou je rošáda, při které se současně přesune král i věž). Žádným tahem se nesmí figura přesunout na pole, na kterém již je jiná figura stejné barvy. Tah na pole s figurou soupeře se nazývá braní. Soupeřův kámen je takovým tahem odstraněn ze šachovnice.

Každý druh figur se pohybuje jiným způsobem. Všechny s výjimkou jezdce se posouvají přímou čarou (po řadách, sloupcích nebo diagonálách) tak, že nesmějí žádnou jinou figuru přeskočit.

- Král se pohybuje o jedno pole v libovolném směru, přímo i po diagonále. 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 mohla přesunout soupeřova figura a tím krále brát.
- Dáma se pohybuje po sloupcích, řadách nebo diagonálách o libovolný počet polí.
- Věž se pohybuje po řadách a sloupcích.

- 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 mezilehlých polích nějaké figury. Jezdec při každém skoku změní barvu pole, na kterém stojí: z bílého pole se dostane vždy na černé a naopak.

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

- 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 soupeřovu figuru, která 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. proměn). Působnost proměněné figury je okamžitá, tzn. může například dát šach nebo se účastnit matování.

Králové, dámy, věže, střelci a jezdci se označují slovem figury. Pojmem těžké figury se rozumí dámy a věže, zatímco jezdci a střelci se nazývají lehké figury.

Pokud hráč nějakým tahem napadne soupeřova krále, tzn. táhne tak, že by příštím tahem mohl krále brát, říkáme, že soupeři 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.

V soutěžním šachu je hra limitována časem, pro jehož měření se používají šachové hodiny. Ty každému z hráčů měří čas samostatně v době, kdy je na tahu a přemýšlí. Hráč, který překročil stanovený čas, partii prohrává, pokud soupeř v té době ještě čas nepřekročil a má na šachovnici dostatek sil umožňujících dát soupeři mat při nejhorší možné soupeřově hře. Časy na soutěžní partii obvykle bývají v řádu hodin, existuje však i takzvaný rapid šach (15 až 60 minut na partii pro každého hráče) a bleskový šach (méně než 15 minut na partii pro jednoho hráče, typicky 5 minut).

Oficiální znění pravidel schvaluje a vydává Mezinárodní šachová federace (FIDE). Jejich úplnou verzi naleznete zde [4].

2.2 Programy

Dnešní šachové programy jsou založeny na dvou různých přístupech. Prvním je metoda hrubé síly, kdy počítačový program zkouší zahrát většinu možných pozic a takto zahrané pozice následně podle daných kritérií ohodnotí. Výsledný tah je určen z nejvýhodnější pozice. Druhým přístupem je využívání databáze již odehraných partií, kde se pomocí

komplikovaných metod vyhledává nejlepší možný tah vedoucí k vítězství. Nejlepší světové programy většinou používají kombinaci těchto dvou přístupů.

V současné době je za nejsilnější šachový program považován projekt Rybka, jehož autorem je Čechoameričan a šachový mezinárodní mistr Vassik Rajilich. Program vyhrál řadu turnajů proti ostatním programům, také i porazil řadu světových mistrů. Hlavní síla Rybky spočívá v jeho schopnosti hrát strategicky. Je schopný provést výměnu těžké figury za lehkou pro získání poziční výhody, což většina ostatních šachových programů není schopna detekovat. Zdrojový kód Rybky nebyl nikdy zveřejněn. Díky jeho převaze nad ostatními programy je proto předmětem řady spekulací. Více se dozvíte na [12] nebo [15].

Dalším velice slavným programem je Deep Fritz, který již mnoho let fascinuje šachový svět. Tento program hrál proti šachovým velmistrům i mistrům světa. Program obsahuje knihovny s velikým množstvím již odehraných partií a jeho kvality jsou bezesporu ohromné. Více se dozvíte na [13].

Existuje mnoho dalších šachových programů. Kromě komerčních, existuje i řada amatérských programů na velmi vysoké úrovni, které jsou schopny porazit i hráče s mezinárodní šachovou zkušeností. Kvality těchto programů nemohou dosáhnout na kvality výše zmíněných komerčních produktů. Podporou pro vývoj amaterských umělých inteligencí se stala prostředí, která programátorovi poskytnou základní funkce, jako jsou grafické uživatelské rozhraní, ovládání a pravidla. Díky tomu, se může vývojář plně soustředit na vývoj umělé inteligence. Mezi tyto programy patří například WinBoard [3], popřípadě program Arena [14].

2.3 Hardware

Druhým, méně rozšířeným, ale hystoricky starším druhem počítačového šachisty, jsou specializované hardwarové prostředky. U hardwarového šachisty nejsou tahy a oceňovací funkce generovány sekvenčním softwarovým propočtem, ale složitým jednoúčelově navrženým kombinačním obvodem. Takovýto obvod je zobrazen na obrázku 2.1. Průchod elektrického sig-

Obrázek 2.1: Hardwarovy šachista

nálu kombinačním obvodem je velmi rychlý (v řádu nanosekund), což samo o sobě znamená značnou výhodu proti softwarovému řešení. Naprosto klíčovou předností je ovšem možnost libovolně rozšiřovat oceňovací funkci. Tím sice roste složitost, velikost a cena kombinač-

ního čipu, samotné oceňování však probíhá paralelními větvemi a jeho rychlost se proto prakticky nemění.

Průkopníkem hardwarového šachového stroje je legendární Ken Thompson. V roce 1960 se Ken podílel na vývoji Unixu a jazyka C. O deset let později postavil speciální stroj Belle, který k velkému překvapení vyhrál mistrovství světa 1980 před sálovými superpočítači. Ken toto zařízení dále nevyvíjel, ale pro šachy udělal mnoho dalších užitečných věcí. Jako první dal počátkem devadesátých let řadovému uživateli databázi důležitých pětikamenových konstelací na čtyřech CD s dokonalým řešením každé pozice. Na přelomu století zopakoval tento výkon i se šesti kameny, dotazovací server je umístěn na webu. Druhou hardwarovou legendou je DeepBlue zobrazen na obrázku 2.2, který vyvinul u IBM Feng-hsiung Hsu. V roce 1989 získal titul mistra světa, ale vyvrcholením kariéry je nepochybně vítězství v regulérním zápase s Kasparovem v roce 1997. Více se dozvíte na [16].

2.4 Turnaje Umělých inteligencí

Mezi šachovými počítači se pořádá mnoho turnajů a vzájemných zápasů, aby se otestovalo, komu se podařil pokrok ve vývoji umělé inteligence pro hru Šachy. Tyto turnaje a vzájemné zápasy jsou sdruženy v několika žebříčcích: CCRL, GEGT, SSDF. Tyto tři zmíněné žebříčky vede program Rybka, který si tím drží svoji doposud neotřesitelnou pozici. Také se pořádají mistrovství světa, kde se utkávají jak hardwaroví šachisté tak softwaroví.

Obrázek 2.2: Deep Blue

Kapitola 3

Základní algoritmy pro umělou inteligenci

Při tvorbě této kapitoly bylo čerpáno z [1], [17] a [6].

Tato kapitola je věnována problematice základních algoritmů pro umělou inteligenci při hraní her. Hry se díky přesné formulaci úlohy staly jistou ověřovací aplikační oblastí metod umělé inteligence.

V každém případě je hraní her docela přirozeně spojeno s představou prohledávání stavového prostoru, kde stavy i přípustné operátory bývají zcela přesně vymezeny. Zatímco jednoduché hry je možné řešit bez hlubších znalostí o daném problému, pomocí neinformovaného prohledávání stavového prostoru, z hlediska umělé inteligence jsou zajímavé především hry, kde ani nejrychlejší počítače s prohledáváním stavového prostoru bez použití znalostí neuspěly.

Ideálním příkladem jsou právě šachy, kde v každém tahu připadá v úvahu průměrně 35 tahů a v jednotlivých partiích zahraje každý z obou hráču přibližně 50 tahů. Z toho vyplývá, že stavový prostor, který bychom v průměrné partii museli prohledat by měl 35^{100} stavů. Je zřejmé, že v tomto případě se bez použití znalostí obejít nelze.

Metodou jak předejít nekončícímu prohledávání stavového prostoru je určit maximální hloubku prohledávání, přičemž každý ze stavů je ohodnocován statickou ohodnocovací funkcí, která vyjadřuje veškerou postupnou informaci o daném stavu. Hodnota statické ohodnocovací funkce odhaduje, jak je pravděpodobné, že daný stav povede k cíli, tj. povede k vítězství ve hře.

Při prohledávání stavového prostoru v případě hry dvou hráčů, musíme počítat s tím, že rozdílná strategie bude použita v případě, že rozhoduje hráč A, nebo rozhoduje hráč B, tedy protihráč. Za předpokladu, že hráči A jde o maximalizaci výhry (ve smyslu cíle hráče A), hráči B jde právě o minimalizaci výhry (z hlediska cílů hráče A), dostáváme se k nejběžnější metodě prohledávání stavového prostoru v případě dvou hráčů, a to k takzvané metodě minimaxu.

3.1 Metoda minimaxu

Metoda minimaxu je metodou prohledávání do hloubky s omezenou hloubkou prohledávání. Jejím základem je rekurzivní procedura, která se zavolá pro aktuální stav hry hráče který je na tahu. Procedura vrací ohodnocení uzlu, který je pro daného hráče nejvýhodnější a tah vedoucí k tomuto uzlu, tj. tah, který je v daném stavu hry pro hrajícího hráče nejvýhodnější.

Algoritmus předpokládá omezení hloubky prohledávání, jinými slovy omezení povolených iterativních aplikací algoritmu minimax.

Pseudokód metody minimaxu:

```
int minimax(Sachovnice s, int hloubka) {
  Tahy t;
  int indexNejTahu, hodnotaPozice;

  // V případě koncové pozice nebo nulové hloubky propočet ukončíme
  if(jeMat(s)) return -MAT;
  if(jeRemiza(s) return 0;

  // Pokud dosáhneme požadované hloubky ohodnotíme pozici statickou
  // ohodnocovací funkcí
  if(hloubka <= 0) return ohodnotPozici(s);

  t = generujTahy(s);
  nejlepsi = -NEKONECNO;

  for(int i = 0; i < t.pocet(); i++) {
    provedTah(t[i]);
    cena = -minimax(s, hloubka - 1);
    vratTah(t[i]);

    if(cena > nejlepsi) {
      nejlepsi = cena;
      indexNejTahu = i;
    }
  }

  return nejlepsi;
}
```

Je zřejmé, že pro co nejlepší vyhledání tahu je zapotřebí kvalitní statickou ohodnocovací funkci.

3.2 Prořezávání alfa-beta

Prořezávání alfa-beta je vylepšením metody minimaxu díky zavedení mezních hodnot, které nám pomáhají zavrhovat řešení evidentně horší než doposud nalezená. V případě dvou hráčů, musíme pracovat se dvěma mezními hodnotami ohodnocení. Tyto meze se nazívají alfa a beta.

- Alfa představující dolní mez ohodnocení uzlu, v němž je na tahu hráč A (tj. uzlu, v němž je maximalizováno ohodnocení).
- Beta reprezentuje horní mez ohodnocení uzlu, odpovídajícího tahu hráče B (tj. uzlu, v němž je ohodnocení minimalizováno).

Pseudokód alfa-beta prořezávání:

```
int alfaBeta(Sachovnice s, int hloubka, int alfa, int beta) {
  Tahy t;
  int indexNejTahu, hodnotaPozice;

  // V případě koncové pozice nebo nulové hloubky propočet ukončíme
  if(jeMat(s)) return -MAT;
  if(jeRemiza(s) return 0;

  // Pokud dosáhneme požadované hloubky ohodnotíme pozici statickou
  // ohodnocovací funkcí
  if(hloubka <= 0) return ohodnotPozici(s);

  t = generujTahy(s);

  for(int i = 0; i < t.pocet(); i++) {
    provedTah(t[i]);
    cena = -alfaBeta(s, hloubka - 1, -beta, -alfa);
    vratTah(t[i]);

    if(cena > alfa) {
      alfa = cena;
      if(cena >= beta) {
        // Zde je ořezání
return beta;
      }
    }
  }

  return nejlepsi;
}
```

Při prvním volání metody alfa-beta je alfa nastavena na nejnižší možnou hodnotu a naopak beta na nejvyšší možnou.

Účinnost prořezávání alfa-beta do značné míry závisí na pořadí, v němž jsou větve stromu expandovány. Pokud by se náhodou stalo, že budou nejdříve vyšetřovány nejhorší možné cesty, k prořezávání by téměř nedocházelo. Pokud by nejlepší možná cesta byla expandována jako první, nemuselo by dojít k vyšetřování jiných uzlů než mimo tuto cestu.

3.3 Možná vylepšení

Základní myšlenka algoritmu alfa-beta prořezávání byla dále vylepšována a zdokonalována.

3.3.1 Řazení tahů

- Nejjednodušším způsobem vylepšení metody alfa-beta prořezávání je řazení tahů způsobem, aby se na začátku pole tahů nacházely tahy, které mají vysokou pravděpodob-

nost být úspěšné. Tyto tahy se rozpoznávají za pomocí heuristik zaměřenych na konkrétní problematiku, například sežer co můžeš. Pokud by tah vedl k razantní změně materiálu na šachovnici (sebrání, výměna figury, proměnění pěšce za jinou figuru) budeme tah upřednostňovat.

- Prořezávání neperspektivní části stromu je metoda prořezávající větve prohledávaného stromu i v případě, že bylo dosaženo pouze mírně lepšího hodnocení pozice, než v dosavadním průbehu.

3.3.2 Metody prohlubování

- Iterativní prohlubování je algoritmus, který byl vynalezen a poprvé zmíněn v oblasti her. Užívá se především u her s časově omezenou dobou tahu. V takovém případě nelze dost dobře odhadnout jak velkou hloubku prohledávání nastavit. Algoritmus interativního prohlubování může být kdykoliv zastaven, přičemž po zastavení jsou k dispozici odhady statické ohodnocovací funkce dosažené systematickým prohledáváním do maximální hloubky.

- Dopočet do tiché pozice patří u šachů k nejjednodušším a zároveň nejdůležitějším vylepšením alfa-beta prořezávání. Pokud se v metodě alfa-beta dostaneme na nulovou hloubku prohledávání, nevoláme statickou ohodnocovací funkci, nýbrž variantu alfa-beta prořezávání, kde se generují pouze tahy, které mění hodnotu materiálu na šachovnici (tahy beroucí figuru, proměny pěšce). Vzhledem k tomu, že hráči neumožňujeme udělat všechny ostatní tahy (takzvané tiché tahy), musíme mu ponechat možnost nehrát vůbec, jinak by mohlo docházet k nevýhodným výměnám. Touto metodou lze řešit případy nedopočtených výměn figur, kdy při základní hloubce prohledávání nezjistíme, který z hráču výměnu vyhrává.

- Druhotné prohledávání je technikou založenou na principu prohlubování prohledávání u zvláště nadějných variant. Pokud dojdu do nulové hloubky prvotního prohledávání, na jehož základě se vybere nejslibnější tah, provedeme sekundární kontrolní expanzi dosud vybrané optimální cesty, doplněné o prohledání nově vzniklého podstromu do větší hloubky. Tímto způsobem se ujistíme, že ani po větším množství tahů nenarazíme na nějaké úskalí.

Kapitola 4

Návrh a realizace aplikace

V této kapitole je popsán způsob jakým byl samotný projekt realizován a co všechno bylo zapotřebí navrhnout, aby výsledný produkt pracoval nejlépe bezchybně. Jsou zde zmíněny nejdůležitější funkce a třídy, zajišťující hlavní běh programu a obsluhu jak umělé inteligence, tak lidských hráčů, popřípadě manažeru deskových her.

Pro realizaci projektu jsem si vybral oběktově orientovaný jazyk Java^{TM}[7] od společnosti Sun Microsystems, za použití vývojového prostředí NetBeans 6.1[5]. Využíval jsem standardních knihoven. Tento jazyk jsem si vybral vzhledem k jeho všestranosti a přenositelnosti, také jsem přihlížel k dobrému editoru pro grafické uživatelské rozhraní.

Samotná aplikace je řízena třídou SachyView.java, obsaženou v hlavním balíku šachy. Tato třída je vyvolána po spuštění programu hlavní třídou SachyApp.java, komunikující s operačním systém a jejím hlavním úkolem je obsluha grafického uživatelského rozhraní.

4.1 Reprezentace šachovnice

Pro reprezentaci šachovnice existuje více možností z pohledu její velikosti. První z nich je klasická, jakou jí známe ze skutečných šachů, tedy pole 8x8. Tato možnost má však základní nevýhodu, kterou při hře skutečných šachů nevidíme a její kontrolu provádíme zcela automatiky. Jedná se o kontrolu, zda jsme se nedostali s figurkou mimo šachovnici. Při použití pole 8x8, tedy musíme pro každý tah a okraj kontrolovat, jestli nejsme mimo šachovnici, což zpomaluje program. Další možností je pole o rozměrech 10x10, ve kterém již máme dodány okraje, bohužel to problém stále neřeší, jelikož figurka jezdce je schopna táhnout o dvě políčka libovolným směrem tudíž i mimo šachovnici.

Z těchto důvodů jsem si pro reprezentaci šachovnice vybral dvourozměrné pole o rozměrech 12x12. Pro jednotlivé prvky virtuální šachovnice (šachovnice používaná pro potřeby programu, může být rozdílná od té, kterou uživatel vidí na obrazovce) jsem vytvořil třídu Figurka, reprezentující veškeré informace o políčkách:

```
public class Figurka {

    // druh figurky {kral, dama, vez, ...}
    private String druh = null;
    // zda s figurkou bylo hýbáno relevantni pouze pro krále a
    // veř z důvodu rošády
    private boolean uzPohnuto = false;
    // hodnota materiální důležitosti figury
```

```
    private int hodnota = 0;
    // pozice na šachovnici
    private int poziceX = 0;
    private int poziceY = 0;
    // barva figury {B - bíla, C - černá, N - prázdné pole)
    private String barva = ,,N‘‘;

    public Figurka(String druh, boolean uzPohnuto, int poziceX,
                   int poziceY, int hodnota, String barva)
    {
        this.druh = druh;
        this.uzPohnuto = uzPohnuto;
        this.poziceX = poziceX;
        this.poziceY = poziceY;
        this.hodnota = hodnota;
        this.barva = barva;
    }

    Funkce pro práci s třídou (getters, setters, ...)
}
```

V tabulce 4.1 je zobrazen obsah pole šachovnice po jeho základní inicializaci, kde O – značí okraj, N – prázné pole, K – krále, D – dámu, V – věž, S – střelce, J – jezdce a P – pěšáka. V průběhu hry se mění obsah pouze v rozmezí indexů 2 – 9 a z okrajových políček se pouze čte.

11	O	O	O	O	O	O	O	O	O	O	O	O
10	O	O	O	O	O	O	O	O	O	O	O	O
9	O	O	V	J	S	D	K	S	J	V	O	O
8	O	O	P	P	P	P	P	P	P	P	O	O
7	O	O	N	N	N	N	N	N	N	N	O	O
6	O	O	N	N	N	N	N	N	N	N	O	O
5	O	O	N	N	N	N	N	N	N	N	O	O
4	O	O	N	N	N	N	N	N	N	N	O	O
3	O	O	P	P	P	P	P	P	P	P	O	O
2	O	O	V	J	S	D	K	S	J	V	O	O
1	O	O	O	O	O	O	O	O	O	O	O	O
0	O	O	O	O	O	O	O	O	O	O	O	O
	0	1	2	3	4	5	6	7	8	9	10	11

Tabulka 4.1: Virtuální šachovnice po spuštění nové hry

4.2 Zjištění validity tahu

Jinak řečeno zajištění, aby se figurky pohybovali dle platných pravidel hry šachy. Za validní tah se považuje tah takový, jenž s danou figurkou pohybuje pouze způsobem příslušným figurce a zároveň nesmí způsobit ohrožení krále. V programu je kontrola validity

tahu zajištěna pomocí třídy Sachovnice.java, která obsluhuje veškeré události na virtuální šachovnici. Obsahuje potřebné obslužné rutiny jako jsou provedení tahu, vrácení tahu a podobně. Dále komunikuje s generátory tahů, umělou inteligencí, kontrolou šachu, matu, patu a grafickým uživatelským rozhraním. Třída Sachovnice.java obsahuje veškeré potřebné informace pro řízení samotné šachové partie ať mezi dvěma lidskými hráči, tak mezi člověkem a počítačem.

```
public class Sachovnice {

    // virtuální šachovnice
    private Figurka sachovnice[][];
    // reprezentuje zda je na tahu člověk nebo počítač
    private boolean jsemNaTahu = true;
    // reprezentuje zda hráč 1 je počítač nebo člověk
    private boolean hrac1 = true;
    private boolean hrac2 = false;
    // reprezentuje který hráč je na řadě {B = bílý)
    private String hraje = ,,B‘‘;
    // informace o políčku ze kterého se táhlo v minulém tahu
    private Figurka odMinTah = new Figurka();
    // informace o políčku do kterého se táhlo v minulém tahu
    private Figurka kamMinTah = new Figurka();
    // informace o políčku v případě rošády nebo braní mimochodem
    // v minulém tahu
    private Figurka speslMinTah = new Figurka();
    // informace o druhém políčku v případě rošády
    private Figurka speslMinTah2 = new Figurka();
    // pro informační label grafického uživatelského rozhraní
    private String hlaska = ,,‘‘;
    // indikátor braní mimochodem
    private boolean braniMinmochodem = false;
    // indikátor rošády
    private boolean rosada = false;
    // konstanta určení hloubky zanořování alfa-beta řezů
    private int hloubka = 3;
    // počítadlo kolik půltahů již bylo v partii odehráno
    private int pultahu = 0;
    // indikator zda je uroven nastavena na experta
    private boolean expert = true;

    // inicializace pole šachovnice a nastavení okrajů
    public Sachovnice() {
        sachovnice = new Figurka[12][12];
        for(int x = 0; x < 12; x++) {
            for(int y = 0; y < 12; y++) {
               sachovnice[x][y] = new Figurka();
            }
        }
    }
```

```
    Funkce pro práci s třídou (jeTahMozny, provedTah, getters,...)

}
```

Samotné zjištění validity tahu probíhá dále zmíněným postupem. Z grafického uživatelského rozhraní dostaneme souřadnice figurky, jakou chceme táhnout a souřadnice kam máme v úmyslu táhnout. Následně je vyvolána funkce jeTahMozny(), kde pokud není šach, je spuštěn generátor tahů pro příslušný druh figurky. Pokud vygenerované tahy obsahují souřadnice, zaslané od grafického uživatelského rozhraní, funkce vrátí hodnotu true, značící proveditelnost tahu.

V případě šachu je postup obdobný, s tím rozdílem, že místo obyčejného generátoru tahů pro danou figurku, je spuštěn generátor tahů pro šach popsaný v další podkapitole.

Posledním krokem je ověření zda po provedení tahu nenastal šach. Pokud ne, tah je validní a může se vykreslit na grafickém uživatelském rozhraní.

4.3 Kontrola šachu, matu a patu

Pro kontrolu šachu, matu a patu slouží třída SachMatPat z balíku sachy.pravidla. Tato třída obsahuje statické funkce potřebné pro detekci jednotlivých situací.

Zjišťování šachu probíhá následujícím způsobem. Nejdříve se na šachovnici vyhledá pole, kde se nachází král hrajícího hráče, u kterého se zjišťuje, zda by některá ze soupeřových figurek, mohla na pole, kde se nachází táhnout. Pokud se takováto figurka nalezne nastává šach.

Detekce matu je již poněkud složitější a využívá řadu pomocných funkcí a generátorů. Samozřejmostí je, že se mat detekuje pouze v případě, že nastal šach. Nejprve se vygeneruje seznam všech možných tahů pro šach hrajícího hráče mimo tahů králem. Tyto tahy se generují pomocí procházení jednotlivých políček od ohroženého krále k poli, kde stojí figurka ohrožující krále (včetně). Pokud existuje figurka, která by na jedno z těchto polí mohla táhnout, muže tudíž matu zabránit a mat tedy nenastává. V případě, že neexistuje takový tah, jsou vygenerovány dvě pole. První obsahuje místa kam ohrožený král může táhnout. Druhé obsahuje místa kam může hrát soupeř. Pokud soupeř může táhnout na všechna pole, kam může táhnout ohrožený král, nastává mat. V opačném případě je nutné ověřit, jestliže zahraji tah králem, který soupeř zahrát nemůže, zda nenastane šach. Odůvodnením této kontroly je fakt, že v místech kolem krále se může nacházet soupeřova figurka, kterou král je schopen sebrat. Je však nutné ověřit, jestli tato figurka byla krytá. V případě šachu na veškerých polích, kam král směl táhnout nastává mat.

Detekce patu je obdobná detekci matu. Pat musíme kontrolovat pokaždé, když nenastane šach. Vygenerujeme seznam všech možných tahů hrajícího kráče kromě krále. Pokud je tento seznam neprázdný, mat nemohl nastat. Dalším krokem je kontrola, zda se na šachovnici nenachází kombinace figur neumožňující dát mat (dva králové a jeden jezdec). Následně se vygenerují tahy pro krále hrajícího hráče a všechny tahy pro soupeře. Pokud je alespoň jedno políčko, kam může táhnout hrájící král ale soupeř ne nenastal pat. V opačném případě pat nastává.

4.4 Umělá inteligence

Umělá inteligence je v projektu realizována pomocí alfa-beta řezů (viz. 3.2) s proměnnou hloubkou zanořování a pomocí databáze odehraných partií s učením (viz. 5). Základem jsou ohodnocovací funkce, generátor tahů a funkce pro výběr nejlepšího tahu. Všechny tyto funkce jsou obsaženy v balíku sachy.ui třída UI.java.

4.4.1 Generování tahů

Generátor tahů prochází stavový prostor (virtuální šachovnici) a pokud narazí na figurku hrajícího hráče, vygeneruje pro ni všechny možné tahy, které přidává do seznamu. Každý z tahů je reprezentován speciální třídou Tah.java, která obsahuje informace o políčkách přísušného tahu. Výsledný seznam je následně setříděn podle heuristiky sežer co můžeš (viz. 3.3.1), tak aby tahy měnící materiální hodnotu na šachovnici byly na začátku seznamu.

4.4.2 Ohodnocovací funkce

Statická ohodnocovací funkce je nejpodstatnější součástí umělé inteligence. Slouží k odhadování ceny jednotlivých situací na šachovnici. Bývá předmětem dlouhého výzkumu, za účelem nejlepších výsledků v co možná nejkratším čase.

V tomto projektu je ohodnocovací funkce pojata vcelku jednoduše, jelikož šlo hlavně o její rychlost. I přesto si myslím, že její výsledky jsou ucházející. Jsou v ní implementována tato pravidla hodnocení pozic:

- Záporně se hodnotí brzké rozvinutí královny z důvodu špatné strategické pozice lehkých figur.
- Kladně se hodnotí věže na soupeřově předposledním řádku. Předpokládá se, že na posledním řádku stojí soupeřův král.
- Kladně se hodnotí věže na otevřených a polootevřených sloupcích
- Záporně se hodnotí střelci a koně na zadních pozicích svědčící o špatné strategické rozvynutosti figur.
- Kladně se hodnotí koně na centrálních polích (D4, D5, F4, F5) a střelci na hlavních diagonálách.
- Hodnotí se míra zablokování střelců.
- Kladně se hodnotí rozvíjení pěšců na centrálních pozicích a velmi významně je hodnoceno pokud pěšec dojde na druhou stranu šachovnice.

Podle těchto pravidel funkce určí strategickou cenu stávající pozice a vrátí ji funkci alfa-beta prořezávání.

4.4.3 Výběr tahu

Výběr tahu je závislý na zvolené urovni obtížnosti, která určuje hloubku prohledávání stavového prostoru při alfa-beta řezech a na tom, zda se využíva databáze učení. Při nejvyšší úrovni je využito veškerých implementovaných prostředků, což se projevuje na rychlosti hry počítače.

Pokud je výběr tahu pouze na umělé inteligenci alfa-beta prořezávání (úroveň začátečník a pokročilý), je tento algoritmus navržen dle pseudokódu v 3.2. V případě, že se nacházíme v kořeni stromu, je propočet mírně odlišný od výše zmíněného pseudokódu. Je to především z důvodu, že v kořeni stromu nás zajímá především tah samotný, nikoli pouze jeho cena, a také v kořeni hrajeme pouze na jednoho hráče, proto zde není žádná beta, pouze alfa. Algoritmus pro kořenový propočet vypadá následovně:

```
public static Tah nejlepsiTah(Sachovnice s, int hloubka) {
  List<Tah> t = new ArrayList<Tah>();
  int nejlepsi = 0, cena, alfa;

  t = generujTahy(s, s.getHraje());
  alfa = Integer.MIN_VALUE;

  for(int i = 0; i < t.size(); i++) {
    if(provedTah(t.get(i), s)) {
      cena = -alfaBeta(s, hloubka, Integer.MIN_VALUE, -alfa);
      cena = dalOdMatu(cena);
      vratTah(t.get(i), s);
      if (cena > alfa) {
        alfa = cena;
        nejlepsi = i;
      }
    }
  }

  return t.get(nejlepsi);
}
```

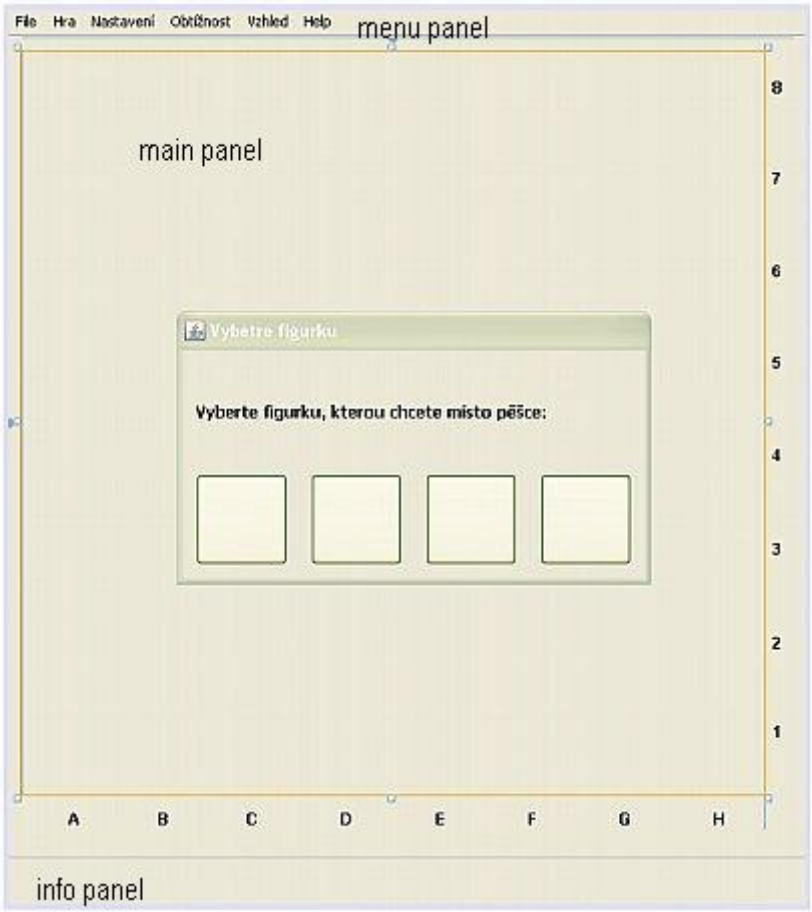
4.5 Grafické uživatelské rozhraní

Pro tvorbu grafického uživatelské rozhraní jsem využil knihovnu Java Swing [9], která poskytuje obrovské množství nejrůznějších komponent a v prostředí NetBeans kvalitně udělaný editor. Na obrázku 4.1 jsou zobrazeny komponenty hlavní obrazovky grafického uživatelského rozhraní programu. Tato obrazovka je součástí řídící třídy SachyView.java, která se stará o řízení aplikace. Tato třída zajišťuje veškeré uživatelské vstup–výstupní operace, překresluje hrací plochu po provedení tahu a vyvolává potřebné komponenty k příslušné funkci programu. Zdrojové obrázky pro grafické uživatelské rozhraní se nacházejí v balíku sachy.resources.

- Main panel je základem zobrazení aplikace. Obsahuje hrací plochu (GridLayot rozdělený na 64 polí, kterým jsou přiděleny Listenery pro uživatelské vstupy z šachovnice). Druhou položkou je JInternalFrame, který je po většinu času běhu aplikace schovaný, jelikož slouží k výběru figurky místo pěšce, pokud dojde na druhou stranu šachovnice. Poslední položkou jsou popisky šachovnice, tvořené z devíti JLabelů. Všechny vodorovné popisky obsahuje jeden JLabel, zatímco pro každý svislý bylo nutné použít vlastní.

- Menu panel obsahuje položky typu JMenu popřípadě JMenuItem, sloužící k obsluhování implemantovaných funkcí hry.
- Info panel je poslední součástní a je to pouze JLabel sloužící k informování uživatele o stavu hry, popřípadě spojení s manažerem deskových her.

Více informací o typu použitých komponent se dočtete zde [9].



Obrázek 4.1: Grafické uživatelské rozhraní

4.6 Rozhraní pro manažer deskových her

Rozhraní pro manažer deskových her byl jedním z požadavků v zadání a slouží jako prostředník mezi šachovým programem a manažerem. Je implementován z důvodu, aby umělá inteligence tohoto projektu mohla popřípadě poměřit své síly s jinými počítačovými šachisty. V projektu je rozhraní obsaženo v balíku sachy, třída Manager.java. V této třídě jsou implementovány veškeré potřebné rutiny, jako jsou konverze tahu z formátu použitelného pro šachový program na formát známý pro manažer (šachy využívají pouze číselné souřadnice v intervalu 2 – 9, zatímco manažer používá klasické souřadnice jako například A1, E3) a naopak. Zajištění komunikace je realizováno pomocí socketů a vstup–výstupních

streamů (ServerSocket, Socket, Stream viz.[7]). Pro samotnou komunikaci slouží balík zpráv, mezi které patří:

- hello msg – Dotaz, zda jsme připraveni komunikovat. Zasláním odpovědi 'hello ok' je dána najevo připravenost přijímat další zprávy.
- com msg – Dotaz na komunikační protokol.
- game [hra] – Informace o hře, pokud se jedná o šachy odpovíme 'game ok', v jiném případě 'game failed'
- board [číslo] – Informace o velikosti hrací plochy. V našem případě vždy osm.
- player [číslo] – Manažer se dotazuje na jméno hráče. Je zaslána odpověď s jménem UI počítače ('name bubakUI').
- timeout [číslo] – Informace o době, v jejímž rozmezí je nutné zaslat odpověď. Pokud je timeout nižší než tři sekundy, odesílá se chybové hlášení 'timeout failed', jinak 'timeout ok'.
- move [tah] – Informace o tahu protivníka (nutné zkonvertovat na srozumitelný tvar). Pokud obsahuje slovo 'start' začínáme partii a je na nás zaslání tahu ve stejném formátu, jako byl tah příchozí.
- new game – Oznámení o nové hře. Provede se inicializace šachovnice.
- quit msg – Informace o ukončení spojení mezi manažerem a programem.

Více se o způsobu komunikace můžete dočíst zde [2].

Kapitola 5

Učení

Tato kapitola je věnována poslednímu z dosud nezmíněných úkolů. Cílem bylo prozkoumat možnosti učení umělé inteligence pomocí databáze již odehraných parií.

5.1 Základní myšlenka

Představa počítače schopného se učit mi připadala poměrně smělá. Promýšlel jsem různé možnosti jak tento problém vyřešit, ale po důkladném zvážení, jsem došel k závěru, že i člověk se nejlépe ponaučí ze svých chyb. Druhou věcí, která mě napadla a pomohla mi v problematice učení bylo: proč bychom dělali něco co vede k požadovanému výsledku jinak než předtím, jen proto, abychom zjistili, zda to tam také vede? Tyto dvě myšlenky mě přivedly na základní stavební kámen učení v tomto projektu.

Základem učení umělé inteligence bude určitý druh databáze již odehraných partií. Po každém tahu nám na šachovnici nastane nová situace, u které je možné přezkoumávat jestli již nenastala. Figurky na šachovnici se utřídí podle daných pravidel do předdefinované struktury, která stav na šachovnici bude reprezentovat. Tento stav lze reprezentovat jednotně, ať je na tahu bílý, nebo černý hráč. V databázi již odehraných partií se pak snažíme najít co nejdelší schodu. V ideálním případě neshodu pouze v jednom, popřípadě dvou prvcích. Pokud je nalezen takový záznam, znamená to, že jsme schopni dostat se do situace, kterou jsme již hráli dříve. Pokud nalezená situace vedla k vítězství, zahrajeme příslušný krok, o kterém se z dosavadní zkušenosti domníváme, že byl správný, neboť nemáme důvod myslet si něco jiného. Ovšem, jestliže víme, že takováto situace vedla k prohře, nebo remíze, nedosáhli jsme z ní požadovaného výsledku (vítězství), tudíž se takové situaci snažíme vyhnout.

Jinými slovy, pokud nastane situace, u které máme zkušenost, že vedla k prohře, pokračujeme v prohledávání databáze. Při dosažení konce databáze bez nalezení schody vedoucí k vítězství se pokusíme provést odlišný tah (vybraný pomocí jiné metody), který by k vítězství mohl vést. Tímto způsobem se počítač je schopný poučit ze svých chyb.

Díky možnosti zaznamenávat situace na šachovnici nezávisle na barvě hrajícího hráče, tento přístup znamená, že se umělá inteligence je schopná poučit i z chyb svého soupeře, neboť v databázi nebude zaznamenáno, kým byl daný tah proveden a i kdyby bylo, nemělo by to relevantní význam.

Tento způsob je velmi jednoduchý, neboť se může stát, že situace, která vedla k prohře v jednu chvíli, ve chvíli další může přinést výhru a naopak, ale to nejsme schopni detekovat. Ve chvíli, kdy budeme znát veškeré možné tahy z dané situace a všechny povedou k prohře,

odehrajeme i tento tah a pokusíme se z pro nás dosavadní proherní situace přinést výhru. Tato situace by šla zlepšit pomocí výběru tahu, který budeme zkoumat, když by se pomocí statické ohodnocovací funkce nalezla z proherních situací ta nejnadějnější (s největší strategickou cenou).

Základním problémem takovéhoto algoritmu, bude jeho nevšestranost. V šachách máme nepřeberné množství situací které mohou nastat a bylo by lepší vyhledávat méně přesné obrazce na šachovnici, podle kterých by se dal určit následující tah. Tento problém je však velmi složitý a i v současné době se jím zabývá řada světových odborníků v souvislosti komerčních umělých inteligencí používajících databáze odehraných partií (např. Deep Fritz). Dalším možným zlepšením by mohlo být odhalování situace, kdy nemá smysl se cokoliv učit, neboť je prohra nevyhnutelná. Tato detekce by však byla silně závislá na úrovni protivníka, neboť pokud bychom hráli proti úplnému začátečníkovi tak beznadějná není každá pozice, což se při hře proti mistrovi říci nedá.

5.2 Realizace

Při realizaci, jsem se snažil o zjednodušení z důvodu rychlosti aplikace, a také z důvodu celkové složitosti problému. Snažil jsem se vytvořit nástin způsobu, jakým by mohlo učení umělé inteligence probíhat.

Pro databázi odehraných partií používám externí soubor databaze.dat obsažený v herním balíku. Funkce zajišťující práci s tímto souborem jsou umístěny v balíku sachy.ui třída Uceni.java. Pro práci se souborem jsem použil třídu File pro uchovávající ukazatele na daný soubor a pro vstup/výstupní operace jsou použity, jako v případě komunikace s manažerem, streamy.

Základní kostra funkce pro práci se souborem:

```
public static void hledejTah() {
  BufferedReader br = null;
  FileWriter fw = null;

  try {
    File f = new File(,,databaze.dat‘‘);
    br = new BufferedReader(new FileReader(f));
    fw = new FileWriter(soubor, true);
  } catch(IOException e) {
    System.out.println(,,Soubor nenalezen‘‘);
  }
}
```

Vyhledávání v databázi tahů probíhá stejně jak je popsáno v předchozí podkapitole 5.1. Při vkládání nového záznamu do databáze (učení) se k tomuto záznamu přidá značka vypovídající o dosavadní neznalosti, zda tento krok vedl k výhře či prohře (remíze). Po dokončení partie se všechy tyto záznamy projdou a značka se změní podle jejího výsledku. Pokud by partie byla ukončena předčasně záznam zůstane označen jako nerozhodnut. Takovéto záznamy se automaticky vymažou při nastavení obtížnosti na experta, aby zbytečně nezabíraly místo a nemusely se brát vůbec v potaz při hledání shodné situace.

Formát záznamu jsem zvolil velmi jednoduchý. Jedná se o textový řetězec o délce šedesáti pěti znaků. První znak je značka zda pozice vedla k výhře (V – z této pozice program

vyhrál, X – prohra, nebo remíza, N – nerozhodnutá hra) a ostatních šedesát čtyři znaků reprezentuje obsach jednotlivých polí šachovnice. Figurky jsou zastoupeny písmeny jako v případě virtuální šachovnice (viz. 4.1), s tím rozdílem, že velká písmena zastupují figurky černého hráče a malá písmena figurky bílého hráče. Každý záznam je na novém řádku pro snadnější čtení. V následujícím kódu jsou vypsány příklady záznamů herních situací ze souboru databaze.dat.

```
VnnnvknnvpppnnpppnnjdsjnnnnnpDnnnnsPnPnnnnnJSnJnnPPnPnPPPVnSnKnnV
XnnnvnnknppnnnpppnnjnnnnnnnnnnnnnnnssnJnnnnPnnnJnnPnnPnPPPVnSnKnnV
VnVnnnnnknnnnnnppnnnnnnnnnpnnnnpnnnnPjJnnnnnnnnnnnnPnnJnPPPnnnnKnnV
NvjsdksjvpppppppppnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnPPPPPPPPVJSDKSJV
```

Musíme počítat s tím, že při častém používání databáze se bude objem dat v ní uložených zvětšovat a tím pádem čas potřebný na její prohledání bude narůstat.

Kapitola 6

Ovládání aplikace

V této kapitole je přiblíženo, jakým způsobem se s programem Šachy pracuje. Jsou tu představeny jeho funkce a je popsáno, jakým způsobem se navazuje komunikace s Manažerem pro deskové hry.

6.1 Ovládání a funkce hry Šachy

6.1.1 Požadavky a spouštění aplikace

Program byl vyvíjen především pro systém Windows. Pro jeho spuštění pod tímto operačním systém, nejsou žádné velké požadavky. Jediné co je potřeba, je volně šiřitelný Java Runtime Enviroment, ke stažení například zde [8]. Teoreticky, by měl program fungovat i pod jinými operačními systémy jako je Linux, to ale nebylo testováno.

Pro samotné spuštění programu stačí v hlavní složce Šachy spustit soubor Sachy.jar, nebo v příkazovém řádku vstoupit do domovského adresáře hry šachy a napsat příkaz: java -jar „Sachy.jar“.

6.1.2 Ovládání

Hra se ovládá intuitivně myší, popřípadě klávesnicí. Po kliknutí na figurku hrajícího hráče, se tato figurka označí červeným rámečkem. Pro odehrátí tahu klikneme na políčko, kam chceme tah provést. Pokud se tah neodehraje, znamená to, že se snažíme zahrát tah, který je v rozporu s pravidly hry Šachy (viz. 2.1). Pokud chceme zahrát jinou než již označenou figurkou, klikneme na označenou figurku podruhé, čímž ji uvolníme a můžeme vybrat figurku jinou. Hra také obsahuje sadu klávesových zkratek, které jsou zmíněny níže.

6.1.3 Funkce hry šachy

Veškeré uživatelsky přístupné funkce jsou umístěny v horním panelu s menu. Některé z funkcí mají klávesové zkratky a většina z nich obsahuje upřesňující popisek. Pro jeho zobrazení stačí myší chvíli počkat nad danou položkou. Některé položky jsou přístupné pouze v určitých fázích, nebo nastaveních hry.

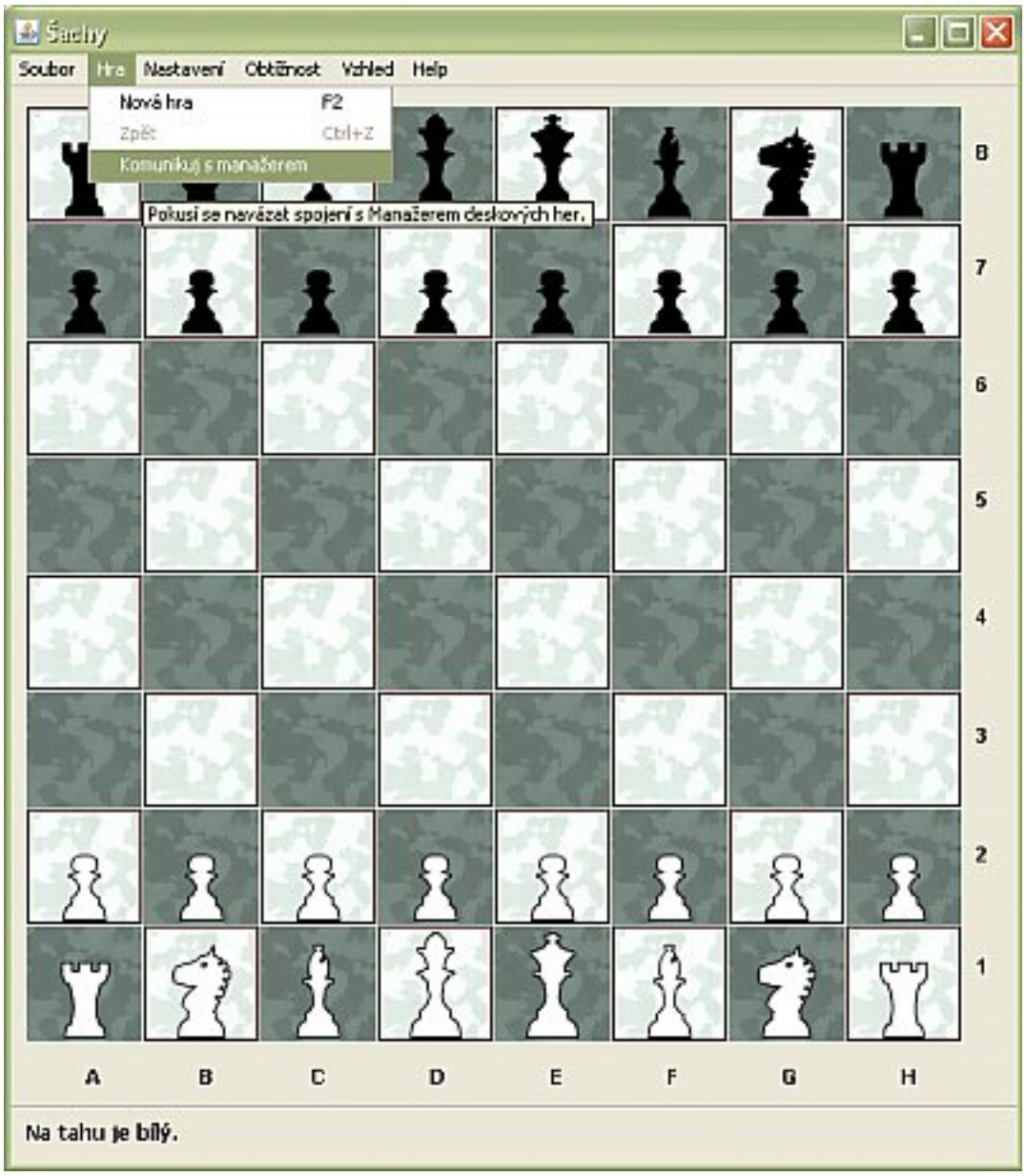
- Soubor/Konec – ukončí běh programu (klávesová zkratka CTRL+Q).
- Hra/Nová hra – inicializuje všechny proměnné na počáteční hodnoty a spustí novou hru (klávesová zkratka F2).

- Hra/Zpět – vrátí právě provedený tah (klávesová zkratka CTRL+Z). Funkce je přístupná pouze v případě hry dvou lidí, neboť při hře hráče proti počítači postrádá smysl, díky okamžitému zahájení generování tahu umělou inteligencí. Existuje jedna výjimka, při hře dvou lidí. Pokud provedeme braní mimochodem (viz. 2.1) a následně tento tah vrátíme, již nemůžeme provést braní mimochodem znovu. Tento problém je dán závislostí možnosti zahrát braní mimochodem na předchozím soupeřově tahu. V případě, že zahraji braní mimochodem, tak se tento tah stane i minulým tahem a po jeho vrácení jím stále zůstává, tudíž není splněna podmínka minulého tahu.
- Hra/Komunikuj s manažerem – viz. 6.2
- Nastavení – zde si můžeme zvolit, zda chceme hrát proti počítači a za jakou barvu. Po zvolení je spuštěna nová hra.
- Obtížnost – nastavení obtížnosti umělé inteligence. Lze ji měnit i v průběhu hry. Začátečník využívá pouze alfa-beta prořezávání s mírou zanoření tři půltahy. Pokročilý, také používá pouze alfa-beta prořezávání, avšak s mírou zanořování 4 půltahy. Expert využívá jak alfa-beta prořezávání (4 půltahy), tak i databázi již odehraných partií s učením.
- Vzhled – nastavení jednoho z dvou možných vzhledů šachovnice a hracích figur (viz. obrázky 6.1 a 6.4).
- Help – obsahuje základní informace o programu, kompletní šachová pravidla FIDE a nápovědu pro snadnou orientaci v programu.

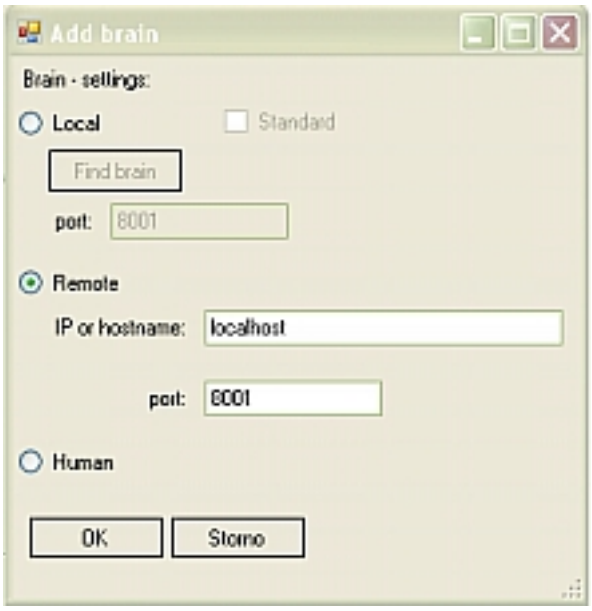
6.2 Komunikace s manažerem pro deskové hry

V této části je popsáno jakým způsobem navázat spojení s manažerem pro deskové hry, který je k projektu přiložen, kvůli možnému testování. Manažer se spouští souborem manager.exe v adresáři manager.

Pro navázání spojení ze strany šachů stačí v menu Hra stisknout Komunikuj s manažerem (viz. obrázek 6.1). Program začne naslouchat na portu 8001. Pokud neobdrží do dvaceti sekund odpověď, naslouchání se ukončí. Po dobu čekání na odezvu program nereaguje na uživatelské vstupy. Pro utvoření spojení ze strany manažeru je třeba po spuštění programu přepnout pomocí tlačítka v horní části na hru šachy. Dalším krokem je navázání brainů (hráčů), pomocí dvojice tlačítek (BRAIN 1 BRAIN 2) pod šachovnicí. Tato tlačítka nám vyvolají dialogové okno (viz. obrázek 6.2), kde volíme jaký druh hráče si přejeme. Pokud chceme aby hráčem byl program Šachy, vyplníme okno následovně. Zvolíme možnost Remote, hostname nastavíme na localhost a číslo portu bude 8001 (viz. 6.2) a stiskneme tlačítko OK. Musíme mít na paměti, že spojení se naváže pouze v případě, že program Šachy naslouchá. Druhého hráče nastavíme obdobným způsobem. Pro potřeby testování doporučuji zvolit možnost Human, kde následně budete vyzváni k zadání jména hráče. Poté už stačí stisknout tlačítko start a hra může začít. Tahy lidského hráče se provádějí pomocí manažeru a jeho šachovnice. Na obrázku 6.3 je zobrazen Manažer deskových her přepnutý na hru šachy. Ukázka z komunikace mezi hrou Šachy a manažerem je vykreslena na obrázku 6.4.

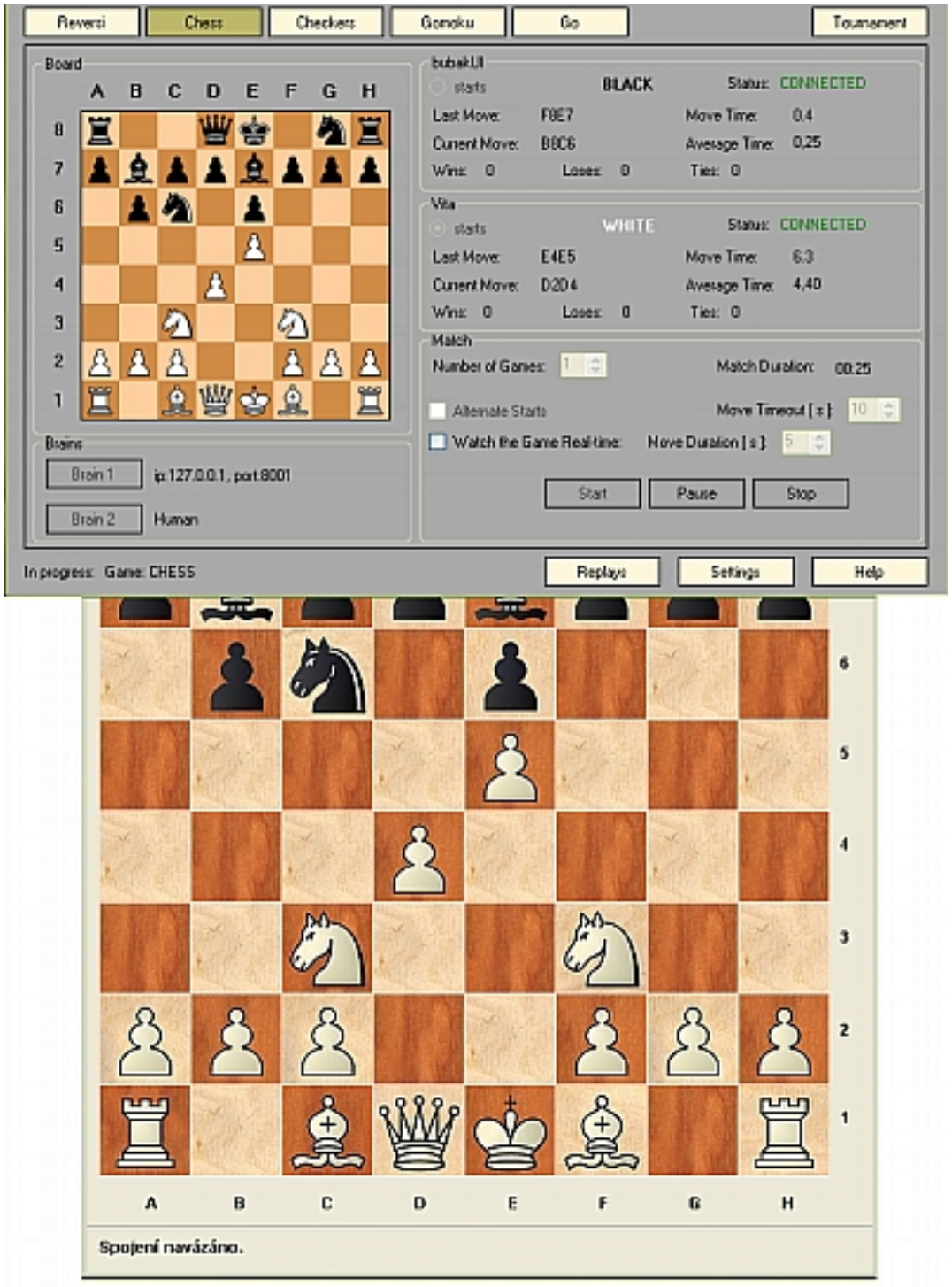


Obrázek 6.1: Klasické zobrazení



Obrázek 6.2: Přidat brain

Obrázek 6.3: Manažer deskových her



Obrázek 6.4: Moderní zobrazení a komunikace s Manažerem pro deskové hry

Kapitola 7

Závěr

Práce se věnuje problematice umělé inteligence pro hru šachy. Popisuje existující metody a jejich možná vylepšení. Shrnuje již existující šachové programy a specializovaný šachový hardware.

V rámci této práce se podařilo vytvořit aplikaci pro hraní hry šachy jak dvou lidských hráčů, tak hraní proti počitači. Bylo vytvořeno vlastní grafické uživatelské rozhraní, definovány veškeré potřebné funkce, mimo jiné umělá inteligence založená na metodě alfa-beta prořezávání, dosahující obstojných výsledků proti méně zkušeným hráčům.

Dále bylo implementováno rozhraní pro Manažer deskových her umožňující poměření sil umělé inteligence aplikace s jinými šachovými počítači, zapojování do šachových turnajů a získávání výkonostních bodů.

Byly zde diskutovány možnosti učení umělé inteligence na základě databáze již odehraných partií. Některé závěry plynoucí z této diskuse byly použity v programu. Implementovaný algoritmus učení není nijak zvlášť propracovaný, zasloužil by mnohá vylepšení, avšak nastiňuje možnosti, kterými by se tento směr mohl ubírat

Z pohledu programátora bylo hlavním přínosem důkladné seznámení s teorií umělé inteligence pro hraní her a s historií a směřováním dalšího vývoje v této oblasti. Dále bylo přínosem zdokonalení v programovacím jazyce Java, zvláště ve tvorbě uživatelských rozhraní, které bylo oceněno od testujících uživatelů jako velmi příjemné.

Dalšímu vývoji aplikce v rámci umělé inteligence by prospělo vylepšení alfa-beta prořezávání některou z možných pomocných funkcí, popřípadě dodání knihoven pro koncovky a začátky hry, které jsou kritickou fází v šachu. Dále by pomohla lepší vyhledávací funkce pro nalezení odpovídající pozice z databáze odehraných partií.

Literatura

[1] Kolektiv autorů: *Umělá inteligence (1)*. Academia, 1993, iSBN 80-200-0496-3.

[2] Kouřil, J.: *Manažer deskových her, bakalářská práce*. Brno, FIT VUT v Brně, 2009.

[3] Mann, T.: XBoard and WinBoard. [online], Naposledy navštíveno 16.4.2009.
URL http://tim-mann.org/xboard.html

[4] Mozek.cz: Pravidla Šachu FIDE. [online], Naposledy navštíveno 15.4.2009.
URL http://mozek.cz/info/sachy-pravidla

[5] netbeans.org: Welcome to NetBeans. [online], Naposledy navštíveno 17.4.2009.
URL http://www.netbeans.org/

[6] Němec, J.: Šachové myšlení (1). [online], Naposledy navštíveno 16.4.2009.
URL http://www.linuxsoft.cz/article.php?id_article=1109

[7] Sun microsystems: JAVA^{TM}2 Platform Standard Ed. 5.0. [online], Naposledy navštíveno 16.4.2009.
URL http://java.sun.com/j2se/1.5.0/docs/api/

[8] Sun microsystems: Java Downloads for All Operating Systems. [online], Naposledy navštíveno 17.4.2009.
URL http://www.java.com/en/download/manual.jsp

[9] Sun microsystems: Trail: Creating a GUI with JFC/Swing. [online], Naposledy navštíveno 17.4.2009.
URL http://java.sun.com/docs/books/tutorial/uiswing/

[10] Wikipedia: Turek(stroj). [online], Naposledy navštíveno 15.4.2009.
URL http://cs.wikipedia.org/wiki/Turek

[11] Wikipedia: Šachy. [online], Naposledy navštíveno 15.4.2009.
URL http://cs.wikipedia.org/wiki/Šachy

[12] Wikipedia: Rybka. [online], Naposledy navštíveno 16.4.2009.
URL http://cs.wikipedia.org/wiki/Rybka

[13] www.chesslady.com: Šachové programy. [online], Naposledy navštíveno 16.4.2009.
URL http://www.chesslady.com/

[14] www.playwitharena.com: Free chess graphical user interface Arena for chess engines. [online], Naposledy navštíveno 16.4.2009.
URL http://www.playwitharena.com/

[15] www.rybkachess.com: Rybka - for the serious chess player. [online], Naposledy navštíveno 16.4.2009.
URL http://www.rybkachess.com/

[16] www.vlasak.biz: Brutus. [online], Naposledy navštíveno 16.4.2009.
URL http://www.vlasak.biz/brutus.htm

[17] Zbořil, F.: *Základy umělé inteligence, studijní opora*. FIT VUTBR, 2006.