Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Implementace algoritmů šachové hry

Plzeň, 2018 Bc. Jan Sehnal

Prohlášení

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

V Plzni dne 16. května 2018

Bc. Jan Sehnal

Abstrakt

Implementace algoritmů šachové hry

Diplomová práce se zabývá technikami používanými pro realizaci šachových programů. Účelem první části, která je věnována analýze šachové hry, je identifikovat obecně platné principy, které je možné převést na programově realizovatelné prvky ohodnocovací funkce šachových pozic. Práce se dále zaměřuje na algoritmy používané k vyhledávání šachových tahů včetně technik používaných k dosažení vyšší hloubky vyhledávacího stromu. Implementační část popisuje tvorbu vlastního šachového programu a využívá postupy z analytické části práce.

Klíčová slova: šachy, šachový algoritmus, statická ohodnocovací funkce, umělá inteligence

Abstract

Chess Algorithm Implementation

This thesis deals with the implementation of a chess program. The first part of the thesis is dedicated to chess game analysis. The main purpose of the first part is to identify general principles of the chess game which can be programatically transformed into components of a positional evaluation function. The analytical part also contains information about chess game search algorithms, including techniques which can be used to achieve greater search depth. The practical part of this thesis describes the implementation of a chess engine by using techniques from the analytical part.

Keywords: chess, chess algorithm, static evaluation function, artificial intelligence

Obsah

1 Úvod

Jeden z hlavních rozdílů mezi způsobem přemýšlení člověka a šachového programu vystihl československý šachový velmistr Richard Réti: „Milovníci šachu, kteří se mě ptají, kolik tahů počítám dopředu, jsou vždy překvapeni, když jim zcela pravdivě odpovídám: ‚zpravidla ani jeden'." Hlavní zbraní lidského šachového hráče jsou jeho zkušenosti a intuice. Schopnost podvědomě propojovat vzorové šachové pozice s příslušnou akcí, kterou je v dané pozici zapotřebí provést. Na druhé straně barikády stojí šachový program. Neúprosný nepřítel, který má k dispozici vysokou výpočetní sílu a soubor pravidel, který mu dodal jeho lidský stvořitel.

Tato diplomová práce navazuje na moji bakalářskou práci na téma „Implementace algoritmu šachové hry". V té byly nastíněny základní způsoby používané při implementaci šachové hry. Cílem této práce je prohloubení znalostí o tvorbě šachových programů. Zaměřil jsem se na techniky potřebné k vytvoření efektivního šachového algoritmu. Mou další snahou bylo pokrýt co nejvíce strategických prvků šachové hry a promítnout je do ohodnocovací funkce vyhledávacího algoritmu. Tato práce se dále zabývá způsoby programových řešení pro počáteční a koncové části šachových partií, protože pro tyto pasáže šachové hry platí speciální pravidla, která neodpovídají obecnému konceptu ohodnocovací funkce.

2 Ohodnocovací funkce

Český velmistr David Navara, který se pohybuje na samotné špičce světové šachové scény, pronesl: „Šachy jsou příliš složitá hra na to, aby bylo možné vytvořit jednoduchý návod na to, jak dosáhnout bezchybné hry.“ Programujeme-li počítač pro hraní šachové hry, návod, resp. soubor pravidel je ale přesně to, co potřebujeme. Ohodnocovací funkce šachového algoritmu je právě tímto souborem pravidel, pomocí kterého šachový program určuje, zda jím provedené tahy vedou k výhodné pozici.

V bakalářské práci byly do ohodnocovací funkce zahrnuty následující faktory: materiálová rovnováha, postavení vůči středu šachovnice, pohyblivost figur, napadení figur, postup pěšců a konec partie (mat, pat). Takto vytvořená ohodnocovací funkce posloužila k zprostředkování základního chování šachového programu, nicméně i mírně pokročilý protivník většinou neměl žádný problém program porazit. V této kapitole rozeberu další faktory, o které je možné ohodnocovací funkci rozšířit, aby se více podobala rozhodovacímu procesu zkušeného hráče šachu.

2.1 Pár střelců

Vlastní-li hráč oba střelce, je to obecně považováno za poziční výhodu. Takováto výhoda je většinou ohodnocována hodnotou přibližně poloviny pěšce. Při zahájení, kdy jsou všichni pěšci ještě na šachovnici, jezdci mají vyšší hodnotu než střelci. Hráč, který při zahájení vymění jezdce za střelce tak získává dočasnou výhodu, která se postupně snižuje v závislosti na počtu pěšců na šachovnici: čím méně pěšců zbývá, tím vyšší hodnoty pár střelců nabývá. Dalším důvodem, proč je výhodné vlastnit oba střelce, je vzájemné doplňování pole působnosti. Každý ze střelců má přístup na opačnou polovinu polí šachovnice. Vlastní-li hráč pouze jednoho střelce, soupeř často zvolí strategii přesunutí svých figur na pole opačné barvy ke hráčově střelci, čímž do značné míry sníží možnosti tohoto střelce. V případě, kdy hráč stále má oba své střelce na šachovnici, tato hrozba odpadá.

2.2 Vazba

Již v mé předchozí práci jsem nastínil problém svázanosti figur neboli vazby. Jedná se o situaci, kdy jsou možnosti pohybu figury omezené, jelikož jejím odtažením by byla vystavena napadení jiná figura, která byla původní figurou kryta. Jev vazby je tedy nežádoucí, jelikož omezuje pohyblivost figur. Rozlišujeme čtyři typy vazeb [1]:

2.2.1 Absolutní vazba

Absolutní vazba je taková vazba, kdy odtažení figury, která se nachází ve vazbě, není možné, protože by se podle pravidel jednalo o nelegální tah. Tato situace nastává, je-li král schován za figurou ve vazbě. Odtažení této figury by vystavilo krále v šach a tudíž je tento tah nelegální.

2.2.2 Relativní vazba

Pro relativní vazbu platí, že je figuru ve vazbě sice možné odtáhnout, ale většinou to není pro hráče výhodné. Za figurou, která je v relativní vazbě, může být umístěna jakákoli figura, která není králem. Zpravidla platí, že figura, která je ve vazbě, má nižší hodnotu než figura, která se nachází za ní, není to však podmínkou.

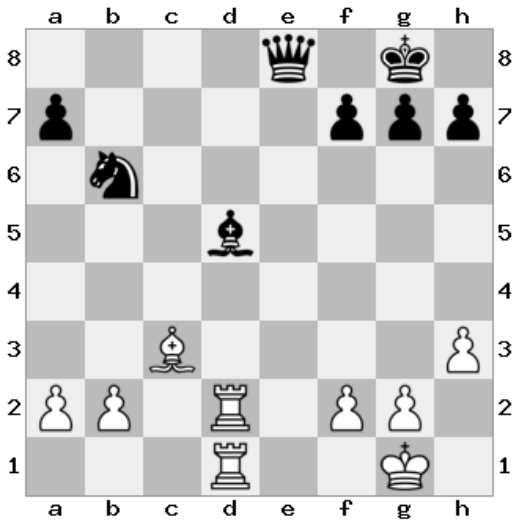
2.2.3 Částečná vazba

Částečná vazba odpovídá případu, kdy je figura sice ve vazbě, ale může se pohybovat po sloupci (dáma, věž) či diagonále (dáma, střelec), jelikož takovým pohybem vazba není porušena. Částečnou vazbu ilustruje obrázek 2.1.

Obrázek 2.1: Bílý střelec se nachází v částečné vazbě

2.2.4 Situační vazba

Situační vazbu je oproti ostatním druhům vazby mnohem složitější detekovat. Jedná se o takový případ, kdy se figura sice nenachází v přímé vazbě, kdy je za ní schovaná nějaká další figura, ale jejím odtažením vznikne situace, která neprodleně vede ke ztrátě materiálu nebo i matu. Situační vazba je vyobrazena na obrázku 2.2.



Obrázek 2.2: Černý střelec se nachází v situační vazbě (hrozbou je Rd8)

2.3 Vidlička

Vidlička je označení pro útok jedné figury na dvě figury naráz. Hráč, jehož figury jsou v této situaci, je většinou nucen cennější z těchto dvou figur přesunout do bezpečí a v důsledku ztratit méně hodnotnou figuru z této dvojice. Stejně jako u vazby se rozlišuje absolutní vidlička (jednou z napadených figur je král a hráč musí bezpodmínečně vyřešit jeho napadení) a relativní vidlička (ani jedna z napadených figur není král). Při vyhodnocování výhodných tahů pro danou partii je vhodné přiřadit vysokou prioritu právě tahům tohoto typu, jelikož je pravděpodobné, že právě tah tohoto typu bude hledaným řešením v dané pozici.

2.4 Nedostatek materiálu

Při ohodnocování šachové pozice je nutné brát v úvahu, zda jsou figury jedné či druhé strany schopny dát soupeřově králi mat. Konkrétně na příkladu: bílý vlastní kombinaci král a střelec, černý má pouze krále. Pokud bychom pozici ohodnocovali pouze podle materiálové výhody, hodnota ohodnocovací funkce by byla přibližně +300 pro bílého. Reálná hodnota pozice je ale 0, tedy jasná remíza, protože se samotným střelcem není možné dát soupeřovu králi mat. K řešení tohoto problému je zapotřebí mít vytvořenou tabulku kombinací, které vedou k remíze.

2.5 Výměna materiálu

V šachové hře je důležité být schopen určit, kdy je výhodné vyměňovat figury se soupeřem. Základní šachovou pomůckou je přiřazení hodnot jednotlivým figurám v násobcích pěšců. Konkrétně: střelec a jezdec mají přibližnou hodnotu třech pěšců, věž má hodnotu pěti pěšců a královna má hodnotu devíti pěšců. Takto stanovené ohodnocení figur ale nemusí přesně odpovídat dané situaci v partii. Jedním z příkladů, kdy je toto ohodnocení zapotřebí poupravit, je výše zmíněný pár střelců. V této podkapitole se zaměřím na další materiálové faktory tohoto typu, které nastínil IM Larry Kaufman [2].

2.5.1 Výměna střelce či jezdce za věž

Jak již bylo zmíněno, věž má přibližně o dva pěšce vyšší hodnotu než střelec či jezdec. Tato hodnota je ale nepřesná a závisí na aktuální pozici. Nebyly-li doposud v šachové partii vyměněny žádné figury, rozdíl mezi ohodnocením střelce a jezdce se od věže liší spíše o 1,5 pěšce. Důvod je zřejmý: jsou-li všechny figury stále na šachovnici, pole působnosti věží je značně omezené. Naopak, chybí-li na šachovnici dáma a věž pro každou stranu, hodnota zbývající věže roste a překračuje hodnotu střelce či jezdce o mírně víc než dva pěšce. Navíc je při vypočítávání síly věže vhodné brát ohled na počet pěšců na šachovnici a zvyšovat ohodnocení věže s ubývajícím počtem pěšců.

2.5.2 Pěšec u věže

Krajové pěšce, tedy pěšce na sloupcích A a H, jsou znevýhodněny oproti ostatním pěšcům, protože tyto pěšce mohou soupeřovy figury brát pouze jedním směrem. Navíc v některých případech je povýšení takového pěšce o něco složitější. Podle [2] je hodnota takového pěšce přibližně o 15% nižší oproti ostatním pěšcům. Tento rozdíl je dostatečně velký, aby bylo pro hráče výhodné upřednostnit sebrání soupeřovy figury právě krajovým pěšcem oproti dobrání jinou figurou. Často je také vedlejším produktem této akce otevření sloupce pro vlastní věž.

2.5.3 Princip nadbytečnosti

Kaufman uvádí, že vlastní-li strana dvě věže, je v jejím zájmu jednu věž vyměnit za odpovídající množství materiálu. Důvodem je, že ve většině případů je velmi obtížné efektivně využít obě dvě věže – často má hráč k dispozici pouze jeden nebo žádný otevřený sloupce pro věž.

2.5.4 Královna versus dvě věže

Podle ohodnocení zavedeného na začátku této kapitoly odpovídá hodnota dvou věží hodnotě královny a pěšce. Podle Kaufmana tato rovnice platí pouze tehdy, nejsou-li na šachovnici již žádné lehké figury. V situaci, kdy každý

z hráčů vlastní alespoň dvě lehké figury, dáma se vyrovná dvěma věžím bez potřeby dodatečné výhody jednoho pěšce.

2.5.5 Královna versus tři lehké figury

Situace, kdy je v partii vyměněna královna za tři lehké figury, již zdaleka není tak častá jako výše zmíněné případy. Nicméně, proběhne-li taková výměna, strana, které zůstaly tři lehké figury, má výhodu přibližně půl pěšce.

2.6 Bezpečnost krále

Bezpečnost krále je dalším z velmi důležitých součástí ohodnocovací funkce. Bezpečnost krále hraje nejdůležitější roli ve středních fázích partie, při zahájení král totiž většinou není v ohrožení a v koncové fázi partie je často vhodné krále aktivně využít a tudíž defenzivní pěšcová hradba pozbývá svého smyslu.

2.6.1 Štít z pěšců

Štít z pěšců je tvořen pěšci, kteří se nacházejí před králem. Účelem štítu z pěšců je ochrana vlastního krále. Štít z pěšců je většinou uvažován až poté, co byla provedena rošáda nebo byl král přesunut ze své základní pozice směrem k jednomu z krajů šachovnice, protože součástí většiny šachových zahájení jsou tahy pěšci, kteří sousedí s královou výchozí pozicí. Hráč tedy získává vyšší poziční ohodnocení, nachází-li se hráčův král na základní řadě a jsou-li jedno až dvě pole před králem rozmístěni hráčovi pěšci.

2.6.2 Pěšcový útok

Pěšcový útok (angl. pawn storm) je šachový strategický prvek, kdy se útočící hráč za pomocí vlastních pěšců snaží prolomit obranu soupeřova krále. Výsledkem takové akce může být porušení soupeřova pěšcového štítu či otevření jednoho ze sloupců poblíž soupeřova krále. Při výpočtu hodnoty ohodnocovací funkce je zapotřebí zavést postih pro každého hráče, jehož král se nachází pod útokem pěšců.

2.6.3 Útok na okolí krále

Jedná se o další pomocné ohodnocení pozice. Podobně jako u pěšcového útoku i útok na okolí krále vyhodnocuje potenciální hrozbu pro krále ze strany nepřátelských figur. Nejdříve je určeno území okolo krále, pro které se bude tato hodnota vypočítávat – například čtverec 3x3. Pole, ve kterém se nachází král, leží v takto vytvořeném čtverci uprostřed od soupeře nejvzdálenější strany. Po stanovení tohoto území přidělíme vlastníkovi krále postih za každou nepřátelskou figuru, která napadá některé z polí v tomto čtverci. Postih je možné zvýšit, pokud některá z útočících figur napadá více polí z okolí krále naráz. Dále je možné upravovat hodnotu postihu podle síly útočících figur.

2.6.4 Ztráta nároku na rošádu

Rošáda je základní defenzivní prvek šachové hry. Bylo-li taženo alespoň jednou ze figurou dvojice král a věž, se kterými chceme rošádu provést, možnost rošády ztrácíme. Pro ohodnocovací funkci je tedy vhodné zavést penalizaci pro ztrátu nároku na rošádu, aby se algoritmus, pokud je to možné, takovým pozicím vyhnul.

2.7 Pohyblivost figur

V bakalářské práci byla nastíněna důležitost pohyblivosti figur. V ohodnocovací funkci figury získávaly bonusy za každé pole, na které se mohly přesunout, a za každou soupeřovu figuru, kterou napadaly. Ve skutečnosti je problém pohyblivosti figur mnohem složitější. V této kapitole se pokusím nastínit některé prvky, které je vhodné brát v úvahu při sestavování ohodnocovací funkce šachového programu.

2.7.1 Různé druhy pohybu

Při vypočítávání bonusů či postihů za pohyblivost figur je zapotřebí brát v úvahu několik faktorů zároveň:

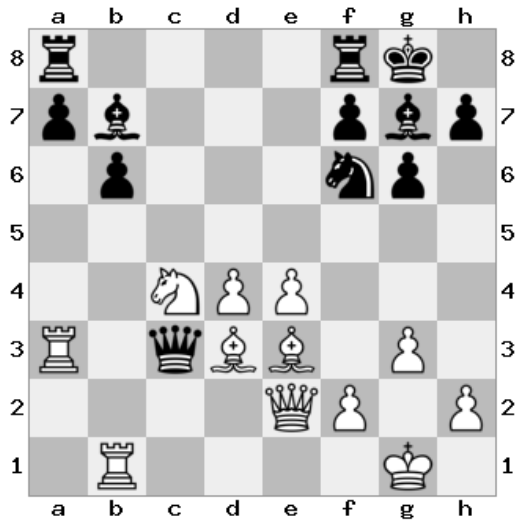
- o jaký druh figury se jedná

- v jaké fázi se šachová partie nachází
- typ možných tahů (tah směrem k soupeři, tah zpět, tah horizontálně, tah vertikálně)

Obecně platí, že v počátečních fázích partie je výhodné se snažit o co nejvyšší mobilitu pro střelce a jezdce a naopak často není vhodné být příliš aktivní s těžkými figurami. Těžké figury přicházejí na řadu později, obvykle ve střední fázi partie. Zejména pro věže jsou upřednostňovány tahy vertikální (volné sloupce). Vhodné využívání jednotlivých figur je podrobněji rozebráno v kapitole pravidel pro jednotlivé figury. Dále je možné přidělovat vyšší bonifikaci za dopředné tahy, jelikož je pravděpodobnější, že právě dopředný tah bude vést ke zlepšení hráčovy pozice. Z bonifikace za pohyblivost je také možné odebrat bonus za všechna pole, na která se sice hráčovy figury mohou přesunout, ale jsou napadena soupeřovými figurami.

2.7.2 Uvězněné figury

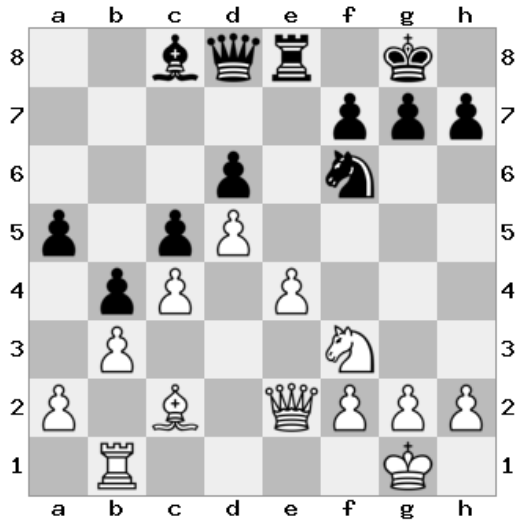
Uvězněná figura je taková figura, která má svou pohyblivost vysoce omezenou a jakýkoli tah touto figurou vede k její ztrátě v příštích tazích. Takovou figuru je často zapotřebí vyměnit za figuru nižší hodnoty, která ji uvěznila (např střelec za pěšce), aby vlastním uvězněné figury získal alespoň nějakou kompenzaci. Příklad uvězněné figury je vyobrazen na obrázku 2.3.



Obrázek 2.3: Černý na tahu, královna na c3 je uvězněna

2.7.3 Špatný střelec

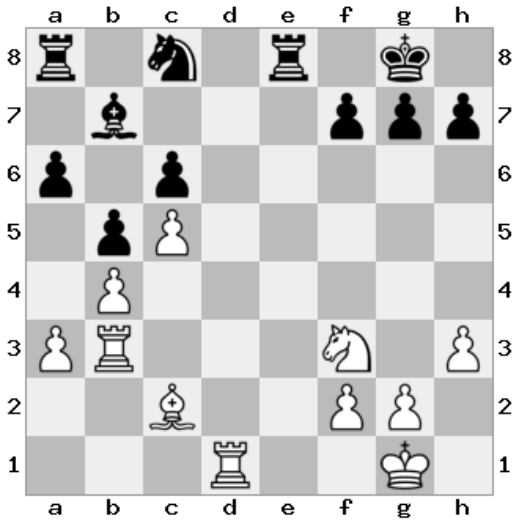
Špatný střelec je střelec, který se pohybuje po polích takové barvy, na které se nachází struktura vlastních či soupeřových pěšců. Jeho pohyblivost je tím značně omezena a je často velmi složité nalézt pro takového střelce vhodnou pozici. Pozice se špatným střelcem je znázorněna na obrázku 2.4.



Obrázek 2.4: Špatný střelec bílého na c2

2.7.4 Pasivní střelec

Jedná se o podobnou situaci jako u uvězněné figury. V tomto případě je střelec „uvězněn“ vlastními figurami, zejména pěšci. Na rozdíl od uvězněného střelce tedy nehrozí jeho ztráta, nicméně ani tento střelec není nijak zvlášť prospěšný. Pohyblivost takového střelce je značně snížena a uvolnění prostoru pro takového střelce je často netriviální, obzvláště v případech, kdy není možné vlastními pěšci táhnout, protože jsou blokováni soupeřovými figurami. Příklad pasivního střelce ilustruje obrázek 2.5.



Obrázek 2.5: Pasivní střelec černého na b7

2.8 Kontrola centra šachovnice

Získání kontroly nad centrem šachovnice je častým cílem mnoha šachových zahájení. V této podkapitole je nastíněno, proč je pro hráče šachu (a tedy i pro šachový algoritmus) výhodné ovládat centrum šachovnice. Je důležité si uvědomit, že strategické výhody, které centralizované figury poskytují, se mohou překrývat s dalšími částmi ohodnocovací funkce. Možným příkladem je pohyblivost figur, která je uvažována jako samostatná komponenta ohodnocovací funkce. Navíc platí, že centralizovaná figura má zpravidla také vysokou mobilitu. Na první pohled by se tedy mohlo zdát, že je bonus za tu samou věc započítán dvakrát, ale není tomu tak. Důvody pro ovládnutí centra šachovnice jsou následující:

- vyšší pohyblivost vlastních figur
- omezení pohyblivosti soupeřových figur
- podpora pro případný útok na soupeřova krále
- nadvláda nad šachovnicí

Ohodnocovací funkci je tedy zapotřebí rozšířit o bonifikaci za kontrolu centrálních polí šachovnice, konkrétně polí *d4*, *d5*, *e4* a *e5*. Dále je možné

zavést sekundární bonus za ovládnutí polí sousedících s poli centrálními. Závěrem je nutné podotknout, že do bonifikace je vhodné započítat bonus nejen za figury, které se v samotném centru nachází, ale i bonus za figury, které centrum napadají či brání.

2.9 Napadání a obrana

Napadání a obrana je rozšířením ohodnocení pro pohyblivost figur. Při hraní šachu je důležité vytvářet hrozby a zároveň odvracet hrozby soupeřovy. Z tohoto důvodu je vhodné v ohodnocovací funkci brát ohled na ofenzivní a defenzivní přínos jednotlivých figur na šachovnici. Je-li vhodně zvoleno ohodnocení za napadání a obranu, šachový program získá schopnost najít některé ze slabin v soupeřově obranné struktuře a této slabiny využít ve svůj prospěch.

Nejjednodušším řešením je započítat do ohodnocení pro danou figuru bonus za všechny přátelské figury, které jsou touto figurou kryty a bonus za každou soupeřovu figuru, kterou tato figura napadá. Toto řešení má zjevné nedostatky. Napadení soupeřova pěšce by mělo stejnou bonifikaci jako napadení soupeřovy královny. Je tedy zapotřebí přidělit vyšší bonus pro útoky na hodnotnější figury. Pro bránění vlastních figur toto pravidlo již neplatí. Hráči není moc užitečné svou královnu bránit mnoha figurami, protože pro soupeře je výhodné jakoukoli ze svých méně hodnotných figur za dámu vyměnit.

2.10 Strategie podle fáze hry

Při vytváření ohodnocovací funkce je programátorovou snahou identifikovat všeobecně platné šachové postupy a tyto postupy posléze ztvárnit do jednoduchých pravidel, která je možné snadno implementovat. Při hledání takových postupů se často stává, že daná strategie funguje velmi dobře v jedné fázi partie a v jiné fázi naopak hráčovu pozici spíše zhoršuje. Vzorovým příkladem by mohlo být například zabezpečení krále. Zbývá-li na šachovnici mnoho figur na obou stranách, není většinou moudré vydat se s vlastním králem na odvážnou výpravu do centra šachovnice. Naopak, pokud dospěla hra do koncové fáze, je zapotřebí, aby i král přiložil ruku k dílu. Účinné strategie pro hraní šachu tedy závisí na aktuálním stavu hry. Rozlišujeme mezi třemi herními fázemi: zahájení, střední hra a koncová hra.

2.10.1 Zahájení

Pojem zahájení označuje přibližně prvních deset tahů šachové partie. Hráčovy cíle v šachovém zahájení jsou odlišné od ostatních fází hry a je tedy zapotřebí náležitě upravit ohodnocovací funkci tak, aby tyto cíle zohledňovala. Základní principy, které bychom se měli snažit při zahájení aplikovat, podle GM Igora Smirnova [3] jsou:

- táhnout centrálními pěšci (*d4*, *d5*, *e4*, *e5*) za účelem vytvoření prostoru pro vlastní figury a zisku iniciativy
- ovládat centrum šachovnice či jej figurami alespoň napadat
- vyvinout lehké figury
- provést rošádu za účelem zvýšení královy bezpečnosti
- netahat dvakrát po sobě stejnou figurou, pokud to není nezbytně nutné
- nepřesouvat zbytečně brzy královnu do pole, stala by se snadným cílem pro soupeřovy lehké figury

2.10.2 Střední hra

V šachové hře nelze triviálně určit konkrétní bod v partii, kdy skončilo zahájení a začala střední hra. Jak již bylo nastíněno, zahájení končí přibližně po deseti tazích, nelze ale zaručit, že hráč provedl tyto tahy efektivně: tah jezdcem dopředu a zase zpět opakovaný pětkrát po sobě dává dohromady také deset tahů, ale hráč se po desátém tahu nachází ve své startovní pozici. Obecně je možné říci, že zahájení končí ve chvíli, kdy se hráčům podařilo naplnit většinu výše zmíněných cílů pro zahájení. Tato kapitola se zaměřuje na hlavní strategické cíle pro střední hru, zmíněné v [4], které je vhodné zohlednit v ohodnocovací funkci. Tyto cíle jsou:

- centralizovat figury
- vyměnit boční pěšce za centrální
- vyhnout se slabinám ve struktuře pěšců
- nevytvářet slabá místa, která může soupeř využít jako opěrný bod

- blokovat soupeřovy volné pěšce jezdci
- umisťovat věže na otevřené sloupce
- pokud možno zachovat pár střelců

2.10.3 Koncová hra

Koncová hra je v šachu velmi odlišná od ostatních fází partie. Mnohá pravidla, která doposud platila, platit přestávají. Optimální strategie pro koncovou hru je přímo závislá na kombinaci figur na šachovnici. Některé pozice představují jednoznačnou výhru pro jednu či druhou stranu, jiné pozice jsou tzv. teoretickou remízou – hra by měla skončit remízou, pokud ani jeden z hráčů neudělá chybu. Detailně popsané strategie pro konkrétní materiálové kombinace jsou dostupné např. v Silmanově knize o koncové hře [5]. V této podkapitole se ale opět zaměřím spíše na obecně uplatnitelné principy, než na strategie použitelné pouze pro konkrétní materiálové kombinace. Obecné principy uplatnitelné pro koncovou hru jsou následující:

- vyměnit se soupeřem věže, pokud vlastním dvě věže a soupeř jen jednu věž
- centralizovat krále
- využít pěšcovou většinu, je-li k dispozici
- využít volných pěšců, jsou-li k dispozici
- podporovat volné pěšce věží
- vyměňovat materiál, pokud máme více pěšců než soupeř
- ponechat si alespoň jednoho pěšce

2.11 Doplňující pravidla pro figury

Tato kapitola se zabývá konkrétními taktickými a strategickými prvky jednotlivých figur, které jsem doposud nezmínil v ostatních podkapitolách. Tyto prvky představují další možná rozšíření ohodnocovací funkce.

2.11.1 Pěšec

Strategické prvky uplatnitelné pro hraní s pěšci jsou následující:

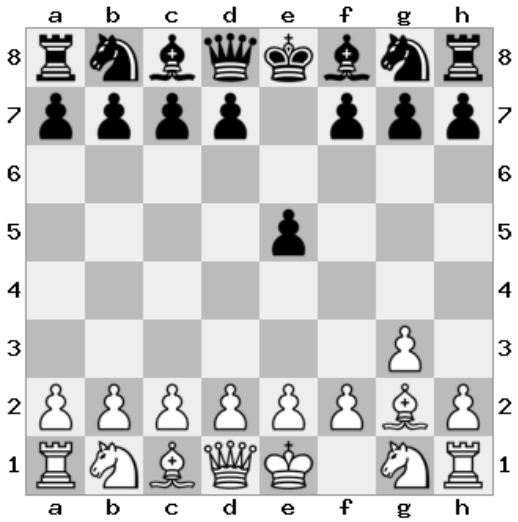
- bonus za volné pěšce
- další bonus za spojené volné pěšce
- postih za zdvojené pěšce
- další postih za ztrojené pěšce
- postih za izolované pěšce

2.11.2 Jezdec

Pro hru s jezdcem je vhodné zavést snižování jezdcovy hodnoty s ubývajícími pěšci na šachovnici.

2.11.3 Střelec

Pro hru se střelcem je možné aplikovat bonus za *fianchetto*, obzvláště, je-li součástí královy obranné struktury. Šachový prvek *fianchetta* je ilustrován na obrázku 2.6.



Obrázek 2.6: Fianchetto na královské straně bílého

2.11.4 Věž

Strategické prvky uplatnitelné pro hraní s věžemi jsou následující:

- bonifikace za věž umístěnou na volném sloupci
- bonifikace za věž umístěnou na sedmé řadě pro bílého a na druhé řadě pro černého
- bonifikace za zdvojení věží na volném sloupci
- bonifikace za zdvojení věží na sedmé řadě pro bílého a na druhé řadě pro černého
- zvyšování hodnoty věže s ubývajícími pěšci na šachovnici
- postih za věž, která je blokována vlastním králem bez možnosti rošády

2.11.5 Královna

Strategické prvky uplatnitelné pro hraní s královnou jsou následující:

- zvýšení ohodnocení královny, pokud má soupeř slabou obrannou strukturu u krále
- bonifikace, jsou-li obě královny na šachovnici a ohodnocení hráčovy obranné struktury u krále je vyšší než ohodnocení soupeřovy obranné struktury (tzn. není výhodné vyměnit královny)

3 Vyhledávací algoritmus

Tato kapitola navazuje tam, kde skončila bakalářská práce [8], kde byly popsány algoritmy *MinMax* a *Alfa-Beta prořezávání*. Tyto algoritmy jsou základním odrazovým můstkem pro většinu přístupů k implementaci šachového programu. Stručný popis algoritmu Alfa-Beta prořezávání je dostupný například zde [23]. Některé ze zmíněných implementačních technik byly inspirovány průzkumem provedeném v oborovém projektu [19] a většina ze zmíněných technik je rozšířením algoritmu Alfa-Beta. V závěru této kapitoly nastíním přístup R^*, který vychází z rozdílného základu.

Algoritmus MinMax je příkladem strategie typu A. Algoritmy typu A ohodnocují všechny dostupné tahy. Nemůže se tedy stát, že by MinMaxu unikl na první pohled zcela absurdní tah, který by v konečném důsledku vedl k vítězství partii. Jedinou podmínkou pro nalezení takového tahu by bylo, aby pozice, která reprezentuje jasně vyhranou partii, mohla být dosažena v dostatečně nízkém počtu tahů, který nesmí překročit hloubku prohledávání. Právě hloubka prohledávání je největší slabinou MinMaxu. Nekontrolovaný růst vyhledávacího stromu znemožňuje prozkoumání pozic do potřebné hloubky. Většina postupů uvedených v této kapitole se bude snažit o zjednodušení vyhledávacího stromu. Některé z těchto postupů zaručují nalezení tahu s nejlepším ohodnocením pro danou hloubku prohledávání, jiné nikoli.

Použitím algoritmu Alfa-Beta je zaručeno, že získaný výsledek přesně odpovídá výsledku získaného MinMaxem. V následující tabulce 3.1 je znázorněn praktický účinek algoritmu Alfa-Beta na počet prohledávaných pozic, uvažujeme-li 40 možných tahů pro každou pozici [12]. Autoři uvádějí, že užitím algoritmu Alfa-Beta je v praxi snížen počet prohledávaných pozic přibližně na $5 \cdot \sqrt{N}$, kde N je počet prozkoumaných pozic.

Půltah	**Počet pozic (N)**	**5 * sqrt(N)**
1	40	-
2	1600	200
3	64 000	1 265
4	2,6 milionu	8 000
5	102 milionů	50 000
6	4,1 miliardy	320 000
7	164 miliard	2 miliony
8	6,5 bilionů	13 milionů
9	262 bilionů	80 milionů
10	10 biliard	512 milionů

Tabulka 3.1: Účinek algoritmu Alfa-Beta

3.1 NegaMax

Algoritmus *NegaMax* vychází z algoritmu MinMax, liší se jen ve formě implementace. Pro úlohu, kde se střídají strany na tahu a kde je možné vyjádřit stav jedním číslem, platí:

$$max(a, b) = -min(-a, -b).$$

NegaMax se používá za účelem zjednodušení implementace, protože jednou metodou je možné pokrýt jak tahy bílého, tak tahy černého. Algoritmus NegaMax tedy produkuje stejné výsledky jako MinMax a uvádím jej, protože některé z později zmíněných algoritmů této kapitoly vychází právě z NegaMaxu. Pseudokód algoritmu NegaMax je zapsán následovně:

```
Function negaMax(depth, whiteToMove):
    if depth = 0 then
        return evalFunction(whiteToMove);
    end
    max = –INF;
    for each object m from Moves do
        m.makeMove();
        int eval = –negaMax(depth – 1, !whiteToMove);
        if eval > max then
            max = eval;
        end
        m.undoMove();
    end
return max
```

Algoritmus 1: Algoritmus NegaMax

3.2 Technika Futility pruning

Technika *Futility pruningu* je jednou ze strategií, kterou je možné zjednodušit vyhledávací strom. Futility pruning se zaměřuje na uzly, které leží ve hloubce o jedna nižší, než je hloubka maximální. Pro každý z těchto uzlů je zkontrolováno, zda se hráč, který je v příslušné pozici na tahu, nenachází v šachu. Dále je zkontrolováno, zda tah, který má být z dané pozice ohodnocován, nevede k šachu či k sebrání figury. Pokud je výsledek všech tří kontrol záporný, je možné přejít k aplikaci Futility pruningu.

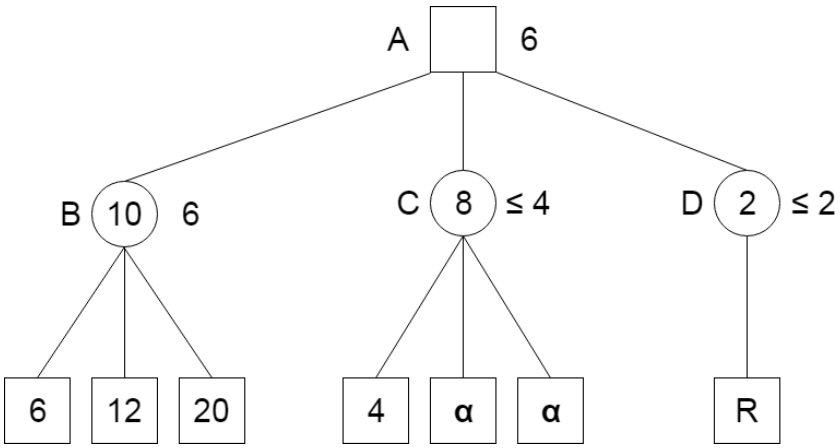
Pro výchozí pozici je vypočteno statické ohodnocení. K tomuto ohodnocení je přičten konstantní bonus v přibližné hodnotě ohodnocení jedné lehké figury. Pokud ohodnocení této pozice po přičtení bonusu nepřesáhne hodnotu alfa algoritmu Alfa-Beta, je další zkoumání takové pozice prohlášeno za zbytečné. Jinými slovy, použitím Futility pruningu je možné odstranit všechny listy vyhledávacího stromu, které neoznačují tah vedoucí k šachu či k sebrání figury, pokud je statické ohodnocení předchozí pozice dostatečně nízké.

S metodou Futility pruningu je možné experimentovat. V prvé řadě je možné měnit hodnotu přičítaného bonusu. Čím vyšší je bonus, tím opatrněji jsou vybírány uzly, které je možné odříznout. Dále je možné na vyhledávací algoritmus šachového programu aplikovat tzv. *Extended futility pruning*, který vychází ze stejné myšlenky, jen je aplikován o jednu úroveň vyhledá-

vacího stromu výše, tedy v uzlech, které leží ve hloubce o dva nižší, než je hloubka maximální. Při použití Extended futility pruningu je zapotřebí použít vyšší toleranci pro ohodnocení výchozí pozice, např. bonus ve výši dvou lehkých figur.

3.3 Technika Razoring

Tato technika vychází z předpokladu, že pro každou pozici platí, že můj soupeř je schopen nalézt alespoň jeden tah, který jeho pozici vylepší. Tento základní předpoklad neplatí v pozicích s vynuceným tahem, kdy každý z dostupných tahů zhorší hráčovo postavení v partii. *Razoring* sice negarantuje nalezení nejlepšího dostupného tahu, ale nalezne tah dostatečně silný. Na následujícím obrázku 3.1 je vyznačeno, jakým způsobem technika Razoringu odřezává větve vyhledávacího stromu [9].



Obrázek 3.1: Aplikace Razoringu a Alfa-Beta algoritmu na MinMax vyhledávací strom

Algoritmus potřebuje prozkoumat tahy B, C a D. Uzly označené kruhem představují úroveň *Min* a uzly označené čtvercem představují úroveň *Max*. Nejdříve je provedena statická evaluace pozic po provedení těchto tahů. Tahy jsou poté seřazeny podle statického pozičního ohodnocení (na diagramu 10, 8 a 2). Nyní je zapotřebí prozkoumat pozice do vyšší hloubky. Tři reakce na tah B mají ohodnocení 6, 12 a 20. Jelikož uzel B leží na úrovni *Min*, je vybrána hodnota 6. Uzel C má statické ohodnocení 8, je tedy prozkoumán, protože platí, že $8 > 6$. První reakce na tah C má ohodnocení 4. Zbylé potenciální

reakce na tah C jsou proto odstraněny Alfa prořezáváním. Tah D má statické hodnocení 2. Podle základní myšlenky Razoringu jakákoli ze soupeřových reakcí na tah D povede k dalšímu snížení ohodnocení této pozice, a proto je celý podstrom pod tahem D odříznut.

3.4 Heuristika Null move

Jedná se o jednoduchý algoritmus používaný za účelem zjednodušení vyhledávacího stromu. Heuristika *Null move* vychází z myšlenky, že pro jakoukoli šachovou pozici není pro hráče výhodné nedělat nic, tedy předat iniciativu soupeři. Při vyhodnocování dané pozice uvnitř stromu je soupeři dovoleno táhnout dvakrát za sebou. Pokud se jeho pozice, resp. skóre statické evaluace nelepší, je výchozí pozice považována za velmi silnou a její statické ohodnocení je vráceno bez dalšího prohledávání. Při aplikaci této heuristiky je zapotřebí ošetřit situace, kdy se šachová partie nachází v nelegálním stavu, jako je např. král v šachu, který nebyl v příštím tahu přesunut do bezpečí. Tato heuristika je efektivní, ale má určité nevýhody. Hlavní nevýhodou jsou situace s vynuceným tahem (stejně jako u Razoringu), kdy by pro hráče bylo výhodnější netáhnout nijak. K vynucenému tahu běžně dochází v koncových fázích partií. V takových situacích totiž neplatí výchozí tvrzení, že nikdy není pro hráče výhodné nedělat nic.

Pseudokód heuristiky Null Move vypadá následovně:

```
Function alphaBeta(alpha, beta, depth, whiteToMove):
    if !isInCheck() then
        whiteToMove = !whiteToMove;                    // null move
        eval = –alphaBeta(–beta, –beta + 1, depth – R – 1,
         whiteToMove);
        if eval >= beta then
            return eval;                                    // cutoff
        end
    end
    // the rest of the search
return alpha
```

Algoritmus 2: Heuristika Null Move

Algoritmus se nejdříve ujistí, zda je provedení Null move legální. Je-li to možné, provede algoritmus prázdný tah a prozkoumá danou pozici v blízkém

okolí hodnoty beta pro hloubku sníženou o R. Nevýhodou heuristiky *Null Move* je přítomnost efektu horizontu. Způsob, kterým tato heuristika zjednodušuje vyhledávací strom, zvyšuje riziko odříznutí posloupnosti tahů, které by měly radikální vliv na vývoj partie. Hodnota R závisí čistě na programátorovi. R určuje, o kolik je snížena hloubka prohledávání při aplikaci Null Move heuristiky. S rostoucím R se zvyšuje riziko přehlédnutí klíčového tahu, ale zároveň se zvyšuje množství ušetřeného času díky prořezávání. Dřívější šachové algoritmy používaly hodnotu $R = 1$, tento přístup je dnes považován za příliš konzervativní. Lépe se jeví $R = 2$ či $R = 3$. Následující tabulky 3.2 a 3.3 uvádí závislost počtu prohledávaných uzlů stromu na hodnotě R [14]. K testování bylo použito 138 testovacích pozic z [15] pro hloubku 9 a 10 půltahů.

Hloubka prohl.	**R = 1**	**R = 2**	**R = 3**
9	1,652,668,804 p.	603,549,661 p.	267,208,422 p.
10	11,040,766,367 p.	1,892,829,685 p.	862,153,828 p.

Tabulka 3.2: Závislost počtu pozic na hodnotě R pro hloubku prohledávání 9 a 10

Hloubka prohledávání	**R = 1**	**R = 2**	**R = 3**
9	64	62	53
10	71	66	65

Tabulka 3.3: Počet vyřešených taktických pozic pro jednotlivá R s hloubkou prohledávání 9 a 10

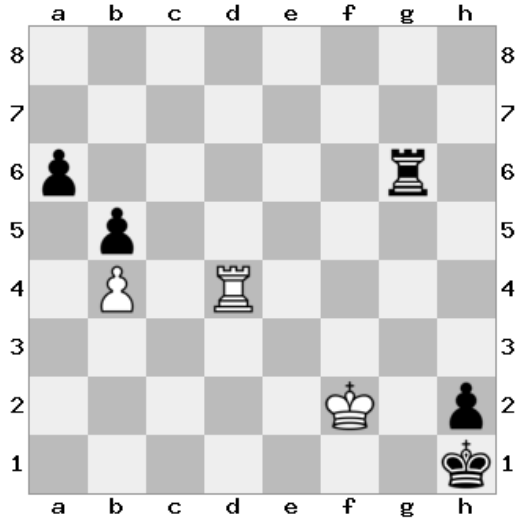
3.4.1 Heuristika Null Move a vynucený tah

Blíží-li se šachová hra do koncové fáze, roste riziko vzniku pozice s vynuceným tahem. Aby mohl šachový program na takové situace patřičně reagovat, navrhuje [12] následující opatření:

- Pokud ve výchozí pozici už žádná ze stran nevlastní figury, pak se pro ně heuristika nulového tahu vypne. V těchto jednoduchých koncovkách silně narůstá účinnost hašovacích tabulek, což poněkud kompenzuje výpadek selektivní vyhledávací komponenty.

- Jestliže uvnitř stromu není materiál strany s právem táhnout roven ani jednomu střelci, pak se rovněž nulový tah neprovádí.
- Ve složitějších koncovkách (např. po dvou věžích a jednom střelci) se jako poslední tah výchozí pozice zkoumá nulový tah. Pokud se ukáže, že nulový tah je lepší než nejlepší z regulérních tahů, pak je odhalena situace s vynuceným tahem. V tomto případě se vymažou hašovací tabulky a hledání pokračuje odpředu po vydatném doušku z nádoby s časem.

První dvě pravidla se jeví jako rozumná opatření. Pokud bychom chtěli být o něco opatrnější, mohli bychom tato dvě pravidla zpřísnit a vypínat heuristiku Null Move i pro vyšší než navrhovaný počet figur na šachovnici. Třetí pravidlo efektivně rozpozná problém vynuceného tahu jen ve výchozí pozici prohledávání. V takovém případě již problém s vynuceným tahem nastal a může být příliš pozdě jej řešit. Následující obrázek 3.2 [12] je příkladem pozice, kde navzdory použití všech výše zmíněných pravidel nebylo předejito prohře v partii v důsledku vynuceného tahu.



Obrázek 3.2: Bílý využije vynuceného tahu k vítězství

Partie na diagramu pokračuje následovně: 1. Rd4-d1+ Rg6-g1 2. Rd1-f1 Rg1×f1 3. Kf2×f1 a6-a5 4. b4×a5... mat. Bílý v třetím tahu vynutí tah soupeřova pěšce na a5 a později proměnou pěšce na a8 dává soupeři mat.

3.5 Řazení tahů

Aby mohl algoritmus Alfa-Beta efektivně odřezávat větve vyhledávacího stromu, je zapotřebí, aby zkoumané tahy byly seřazeny podle předpokládané síly od nejsilnějšího tahu po nejslabší tah. V případě, kdy se podaří přesně odhadnout síla zkoumaných tahů, je maximalizováno prořezávání. Pokud bychom ale chtěli zjistit přesné ohodnocení každého z dostupných tahů, museli bychom do dané hloubky prohledat celý vyhledávací strom. Tímto přístupem bychom ale rovnou nalezli přesné řešení dané pozice a řazení by postrádalo smyslu. Následující podkapitoly se zaměřují na heuristické techniky, kterými je možné bez vysoké výpočetní náročnosti seřadit tahy k budoucímu prohledávání.

3.5.1 Heuristika Killer move

Heuristika *Killer move* je jedním z přístupů používaných pro řazení tahů. Pojmem Killer move je míněn takový tah, který vede k odříznutí větve vyhledávacího stromu algoritmem Alfa-Beta. Prioritně jsou uvažovány uzly ze stejné úrovně vyhledávacího stromu, je ale možné jako Killer move využít i tahy, které způsobily prořezávání v jiných úrovních stromu. Takové tahy jsou průběžně ukládány a při dalším prohledávání upřednostňovány před tahy ostatními. Cílem je zefektivnit prořezávání stromu. Killer Move heuristika podle [13] pracuje efektivněji, pokud je kombinována s metodou transpozičních tabulek. Toto tvrzení vychází z předpokladu, že seřazením tahů a prioritním použitím stejných tahů pro různé hloubky prohledávání docílíme vyššího využití informací uložených v transpoziční tabulce z důvodu častějšího opakování totožných pozic.

3.5.2 Heuristika History

Základní myšlenka *History heuristiky* vychází z předpokladu, že byl-li daný tah dobrý v předchozích pozicích, existuje vysoká pravděpodobnost, že tento tah bude dobrý i v pozici aktuální. Jedná se vlastně o zobecnění heuristiky Killer Move. Dobrým tahem je myšlen takový tah, který buď přímo nabízí nejlepší řešení pro aktuální pozici, nebo alespoň způsobí odříznutí podstromu Alfa-Beta algoritmu. History heuristika využívá faktu, že rozdíl mezi mnoha pozicemi ve vyhledávacím stromu spočívá v různém umístění poměrně

malého počtu figur. Pokaždé, když je nalezen dobrý tah, je zvýšeno interní ohodnocení tohoto tahu. Tahy, které jsou opakovaně dobré, zaujmou vrchní příčky v seznamu tahů History heuristiky. Tahy s vysokým history ohodnocením jsou poté přednostně zkoumány algoritmem Alfa-Beta. Podrobnější informace o History heuristice jsou dostupné z [17].

3.6 Vyhledávání Principal variation

Principal variation je vylepšením Alfa-Beta prořezávání. Principal variation přichází na řadu ve chvíli, kdy už jsou tahy, které mají být zkoumány, předem seřazeny podle jejich očekávané síly.

```
Function PVS(alpha, beta, depth, whiteToMove):
    if depth = 0 then
        return evalFunction(whiteToMove);
    end
    eval;
    for each object m from Moves do
        m.makeMove();
        if m.isFirst() then
            eval = –PVS(–beta, –alpha, depth – 1, !whiteToMove);
             // full search
        end
        else
            eval = –PVS(–alpha – 1, –alpha, depth – 1,
             !whiteToMove);                          // short search
            if alpha < eval AND eval < beta then
                eval = –PVS(–beta, –eval, depth – 1, !whiteToMove);
                 // double back to full search
            end
            alpha = max(alpha, eval); if alpha >= beta then
                break; // cut off
            end
        end
        m.undoMove();
    end
return alpha
```

Algoritmus 3: Principal variation search

První tah je prozkoumán pro maximální velikost okna algoritmu Alfa-Beta. Další tahy jsou zkoumány již pro omezenou velikost okna, aby byla zaručena jistota, že daný tah je opravdu horší, než tah označený jako dosud nejlepší. Potvrdí-li se, že zkoumaný tah opravdu není lepší, došlo k ušetření času při hledání. Zjistí-li se ale, že ohodnocení daného tahu přesahuje hodnotu dosud nejlepšího tahu, je zapotřebí tento nově nalezený tah prozkoumat pro plnou velikost okna Alfa-Beta algoritmu.

Algoritmus tedy předpokládá, že první tah ze seznamu seřazených tahů je pro danou pozici ten nejvýhodnější. Provede tedy pro tento tah prohledávání do plné hloubky. Pro zbylé tahy je již prohledáváno jen pro zmenšené okno Alfa-Beta algoritmu. Zjistí-li se, že některý ze zkoumaných tahů získal pro zmenšené okno vyšší ohodnocení, je zapotřebí tento tah prozkoumat pro plné Alfa-Beta okno. V [16] je uvedeno, že zvýšení výkonu užitím algoritmu Principal variation search je přibližně 10 procent a závisí na zvoleném způsobu řazení tahů. Pokud jsou tahy řazeny neefektivně nebo dokonce náhodně, dojde použitím tohoto algoritmu dokonce ke ztrátě výkonu.

3.7 Algoritmus B*

*Algoritmus B** je příkladem algoritmu, který nepoužívá pouze jedné hodnoty k vyjádření stavu šachové partie. B* ohodnocuje pozice dvěmi čísly, která označují optimistické a pesimistické ohodnocení stavu partie. Během průchodu prohledávacím stromem jsou tyto hranice postupně upravovány. Algoritmus končí ve chvíli, kdy pro nejlepší tah platí, že jeho pesimistické ohodnocení je vyšší nebo rovno než optimistické ohodnocení všech ostatních tahů. Čím jsou od sebe tato dvě ohodnocení vzdálenější, tím vyšší je nejistota konečné využitelnosti daného tahu. Tahy s vysokým optimistickým ohodnocením mohou být označeny jako tahy s vysokým potenciálem pro použití dalšími heuristikami viz pozdější kapitola o obětování figur. Tímto přístupem je redukován *efekt horizontu* (o efektu horizontu bude řečeno více v dalších kapitolách), jelikož narozdíl od přístupu MinMax má algoritmus B* díky neurčitosti nástroj, kterým může potenciální projev efektu horizontu vyjádřit. Další výhodou tohoto přístupu je možnost ukončení vyhledávání, aniž bychom znali přesné ohodnocení dané pozice. Potřebujeme jen vědět, že nalezené řešení je lepší než všechna ostatní dostupná řešení. Podrobněji algoritmus B* popisuje Hans Berliner ve své publikaci [18].

4 Možnosti implementace

4.1 Databáze zahájení

Hlavní výhodou šachového vyhledávacího algoritmu, který používá ohodnocovací funkci, je fakt, že na každou ze vzniklých situací je schopný najít odpověď. Algoritmus z výchozí pozice vytvoří vyhledávací strom a pro danou hloubku prohledávání nalezne tah, který se jeví jako nejvýhodnější. Tato versatilita s sebou ovšem přináší i jistou nevýhodu. Bylo by naivní očekávat, že šachový algoritmus s omezenou hloubkou prohledávání nalezne vždy optimální řešení. Nachází-li se šachová partie v plném proudu s velkým množstvím dostupných tahů pro obě strany, je použití vyhledávacího algoritmu dobrou volbou. Na druhou stranu, je-li šachová partie do značné míry zjednodušena, není na škodu vyhledávací algoritmus nahradit jiným mechanismem. Tímto mechanismem jsou databáze zahájení. Rozlišují se dva hlavní přístupy: sekvenční a poziční.

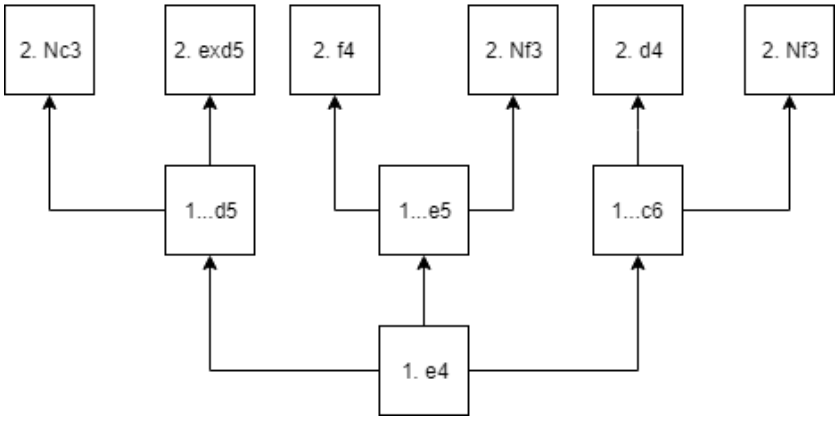
4.1.1 Sekvenční databáze zahájení

Sekvenční přístup vychází z tzv. *ECO* (Encyclopaedia of Chess Openings) standardu [22]. ECO je databáze šachových zahájení, která byla sestavena sběrem dat ze statisíců her mezi šachovými mistry a velmistry. Zahájení jsou rozdělena do pěti hlavních kategorií (A-E). Každému zahájení je pak přiřazen unikátní kód ve formátu kategorie+subkategorie, tedy např. A50. Kategorie A je rozdělena takto [7]:

- white first moves other than 1.e4, 1.d4 (A00–A39)
- 1.d4 without 1...d5, 1...Nf6: atypical replies to 1.d4 (A40–A44)
- 1.d4 Nf6 without 2.c4: atypical replies to 1...Nf6 (A45–A49)

Hlavní myšlenkou sekvenčních databází je vytvoření stromové struktury reprezentující jednotlivé tahy zahájení. Šachový program poté při hledání dalšího tahu jednoduše nahlédne do databáze, zda se ve větvi databázového

stromu, po které se program vydal, nachází další tahy. Nalezne-li další dostupné tahy, jeden vybere (např. náhodně). Pokud již žádné další tahy dostupné nejsou, přebírá kontrolu nad partii vyhledávací algoritmus. Zjednodušený strom šachových zahájení je znázorněn na obrázku 4.1:



Obrázek 4.1: Sekvenční ukládání tahů

4.1.2 Poziční databáze zahájení

Sekvenční způsob je implementačně jednodušší, ale nese s sebou určité nevýhody. Může totiž dojít k transpozici zahájení, kdy dvě totožné pozice odpovídají dvěma různým zahájením, jelikož byly dosaženy různou posloupností tahů. Ze sekvenčního stromu je sice možné duplicity eliminovat, je k tomu ale zapotřebí informace o celé pozici, která v sekvenčním přístupu není k dispozici. Tento problém odstraňují tzv. poziční databáze, jelikož v pozičních databázích nejsou ukládány sekvence tahů, ale výsledné pozice. Jednotlivé pozice takové databáze je možné doplnit o další informace jako např. statistiku výher bílých a černých figur. Tyto informace mohou být využity šachovým algoritmem k výběru nejvýhodnějšího pokračování.

4.2 Databáze pro koncové hry

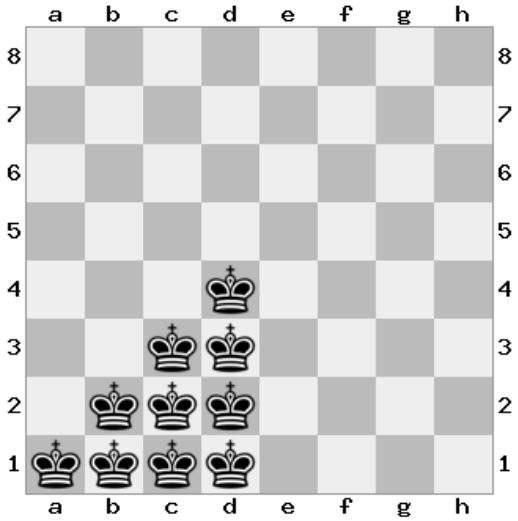
Při zahájení používá šachový program databázi zahájení, pokud je dostupná. Po odehrání posledního tahu dostupného z databáze zahájení přechází program na svůj vyhledávací algoritmus. Pro koncové hry platí podobný princip.

Šachový program si připraví některé z koncových scénářů. Ve chvíli, kdy některý z připravených scénářů nastane na šachovnici, dohraje program partii podle tohoto scénáře. Technika, která se používá k sestavení databáze pro koncovou hru, se nazývá retrográdní analýza.

4.2.1 Retrográdní analýza

Retrográdní analýza je proces, kterým je možné vytvořit databáze koncových her. Každá z databází je vytvářena pro konkrétní kombinaci figur na šachovnici. Jednotlivé tabulkové hodnoty obsahují dvě informace: danou pozici a počet tahů do matu. Postup sestavení takové databáze znázorníme na koncové hře krále a dámy proti králi, viz [10].

Nejdříve určíme počet potřebných pozic v databázi. Černého krále je možné umístit na 64 různých pozic na šachovnici. Využijeme-li symetrie šachovnice, je ve skutečnosti zapotřebí uvažovat pouze 10 různých pozic pro černého krále. Po umístění černého krále je zapotřebí umístit bílého krále a bílou královnu. K dispozici je 64 polí pro každou z těchto figur, překrývání pozic bude ošetřeno později. Celkem je tedy zapotřebí vyhodnotit 10 * 64 * 64 = 40 960 různých pozic. Využití symetrie šachovnice je znázorněno na obrázku 4.2:



Obrázek 4.2: Pozice databáze uvažované pro černého krále

Pro každou z takto vytvořených pozic je nejdříve určeno, zda jsou legální. Nelegální pozice jsou takové, kde se dvě figury nachází na stejném poli nebo kde se dva králové nachází na sousedních polích. Dále je zapotřebí označit pozice, kde se černý král nachází v šachu, a pro každou z nich určit, zda se jedná o mat. Tímto způsobem je vytvořen seznam matových pozic. Každá pozice, ze které je možné jedním tahem získat jednu z matových pozic, je dále označena jako mat jedním tahem. Dokud v databázi zbývají neohodnocené pozice, je pro každou iteraci navyšován čítač pro potřebný počet tahů k matu a zbývající pozice jsou ohodnocovány. Pro tuto konkrétní situaci se rekurze zastaví na desáté iteraci, kde jsou nalezeny nejhorší případy s matem do deseti tahů.

4.3 Přechodné ohodnocení

V šachové partii není jednoduché přesně určit, kdy některá z fází hry končí a jiná fáze začíná. Pro šachový program tato neurčitost herních fází představuje nepříjemné úskalí: program sice má implementováno chování pro jednotlivé fáze, ale neví, který soubor pravidel ve sporné situaci použít. Jedním z možných řešení by bylo přesně definovat hranice jednotlivých fází, například podle součtu hodnot figur obou stran. Nevýhodou tohoto řešení by ale byla neplynulost přechodu mezi jednotlivými fázemi. Docházelo by k radikálním změnám v ohodnocení pozic ve vyhledávacím stromu v místech přechodů mezi fázemi. Elegantnějším řešením je použít *přechodné ohodnocení*. Pro ilustraci použiji zjednodušený příklad. Použijeme faktor F, což je hodnota v rozmezí 0 až 100. Hodnota 0 odpovídá začátku partie a hodnota 100 odpovídá stavu, kdy na šachovnici zbyli pouze dva králové. Rovná-li se F hodnotě 50, přibližně polovina figur byla již vyměněna. Přechody mezi fázemi je pak možné definovat následovně:

$$F < 50 : total\_eval = opening\_eval * (50 - F) + midgame\_eval * F$$

$$F >= 50 : total\_eval = midgame\_eval * (100 - F) + endgame\_eval * (F - 50).$$

Další možností, jak rozlišit herní fáze, je použití počtu odehraných tahů. Tento přístup je zejména vhodný pro určení konce zahájení. Použil-li programátor knihovny zahájení, je navíc možné část ohodnocovací funkce určenou pro zahájení zcela eliminovat a rovnou použít ohodnocování pro střední hru ve chvíli, kdy je odehrán poslední tah obsažený v knihovně zahájení.

4.4 Mapa útoku a obrany

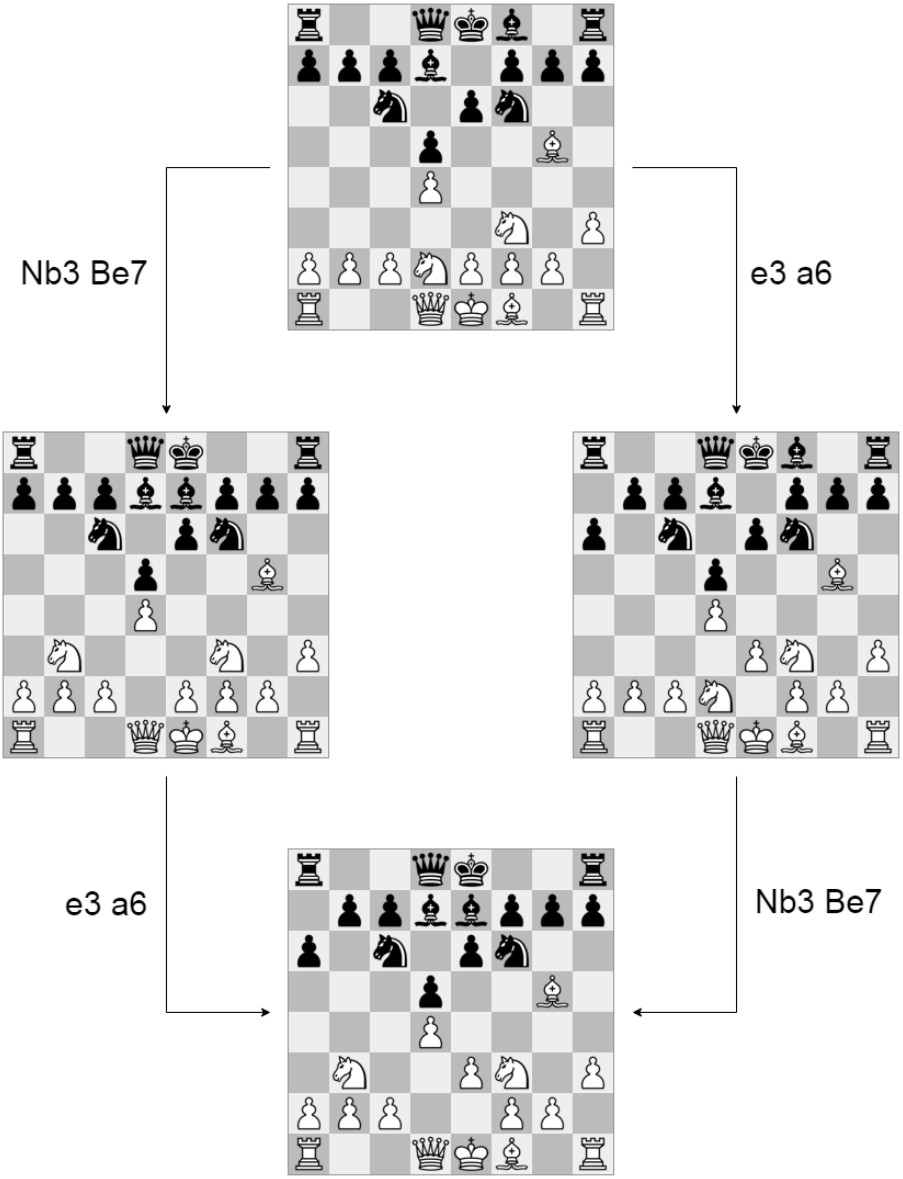
Jedná se o pole hodnot, ve kterém je pro každé pole šachovnice uchováváno, kolika soupeřovými figurami je dané pole napadeno a kolika vlastními figurami je chráněno. Požadujeme-li tuto informaci z pohledu druhého hráče, stačí jen hodnoty pro počet útoků zaměnit za hodnoty pro počet obran. Taková mapa má vysoké využití při výpočtu ohodnocovací funkce, obzvláště při poziční evaluaci jednotlivých figur.

4.5 Mop-up ohodnocení

Mop-up evaluace je užitečná ve chvíli, kdy na šachovnici již nejsou žádní pěšci a jeden hráč má znatelnou materiální výhodu. Jedná se o úpravu ohodnocovací funkce za účelem vyřešení koncové fáze partie. Cílem je zahnat soupeřova krále ke kraji šachovnice a dát mu mat. Výherní strana tedy získává evaluační bonusy za vzdálenost soupeřova krále od středu. V případech, kdy je k matu zapotřebí i král, získává útočící strana bonus k ohodnocení za přiblížení svého krále ke králi soupeře.

4.6 Transpoziční tabulky

Šachový program během vyhodnocování často naráží na totožné pozice dosažené různými posloupnostmi předešlých tahů. Účelem *transpozičních tabulek* je ušetřit výpočetní čas eliminací případů, kdy je stejná úloha řešena vícekrát. Do transpozičních tabulek je ukládáno ohodnocení již zkoumané pozice s informací o tom, do jaké hloubky byla tato pozice zkoumána. Čím výše ve vyhledávacím stromu již ohodnocená pozice leží, tím hodnotnější je pro šachový program si takovou informaci uložit, jelikož k získání takové hodnoty bylo zapotřebí vyhodnotit celý podstrom ležící pod touto pozicí. Ohodnocení pozice, která byla prozkoumána do dostatečné hloubky, je přímo použito v dalším vyhledávání. Informace o ohodnocení pozic s nedostatečnou hloubkou prohledávání mohou být využity pro řazení tahů za účelem zefektivnění prořezávání. Vzniklé transpozice ve vyhledávacím stromu ilustruje následující obrázek 4.3:



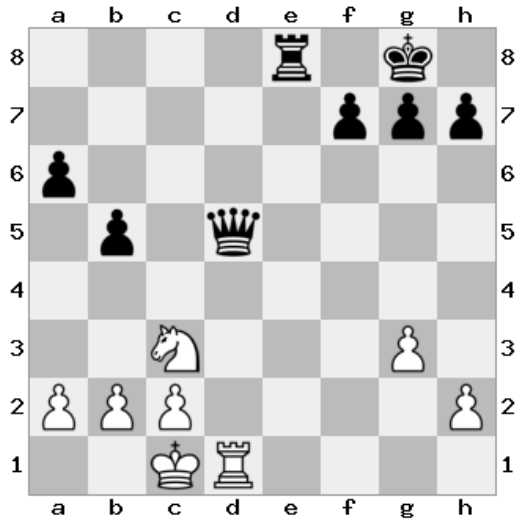
Obrázek 4.3: Transpozice

Informace o pozicích jsou ukládány do hashovací tabulky. Je-li naplněna maximální kapacita tabulky, jsou postupně uvolňovány pozice, které nebyly prozkoumány do dostatečné hloubky, nebo pozice s nízkou frekvencí výskytu. Pokud jsou v transpozičních tabulkách uloženy jen pozice bez dodatečných informací, může dojít k nepřesnostem u pozic, které jsou závislé na předchozím vývoji partie (právo na rošádu, braní mimochodem apod.).

5 Problémy šachových enginů

5.1 Efekt horizontu

Problém *efektu horizontu* společně s technikou *hledání klidu* byl již nastíněn v bakalářské práci [8]. Jedná se o nežádoucí jev, který se projevuje v mnoha úlohách s umělou inteligencí, které jsou postaveny na vyhledávacím algoritmu se stromovou strukturou potenciálních stavů hry. Příčinou problému je omezená hloubka prohledávání šachového algoritmu. Vyhledávací algoritmus ohodnocuje jednotlivé pozice až do předem stanovené hloubky. Prozkoumané scénáře jsou mezi sebou porovnávány a scénář, který se jeví jako nejvýhodnější, je algoritmem vybrán jako řešení aktuální pozice. Efekt horizontu může výrazně ovlivnit rozhodovací proces algoritmu a způsobit zkreslení v ohodnocení jednotlivých scénářů. K efektu horizontu dochází tehdy, pokud se ohodnocení pozice v maximální hloubce prohledávání značně liší od ohodnocení pozice, která leží „za horizontem“, tedy v hloubce o jedna vyšší, než je hloubka maximální. Jednoduchým příkladem by byla obět figury „před horizontem“ za účelem matu jedním tahem „za horizontem“. Šachový program, který by neměl ošetřen efekt horizontu by tuto pozici ohodnotil jako velmi výhodnou pro stranu, která by získala figuru, i když by ve skutečnosti takový scénář vedl k rychlé prohře. Další příklad důsledku efektu horizontu je znázorněn na následujícím obrázku, kde černý ztratí příštím tahem dámu. Pokud by se tato pozice nacházela v listu vyhledávacího stromu, mohla by být ohodnocena jako velká výhoda pro černého. Obrázek 4.3 zachycuje problémovou pozici pro efekt horizontu.

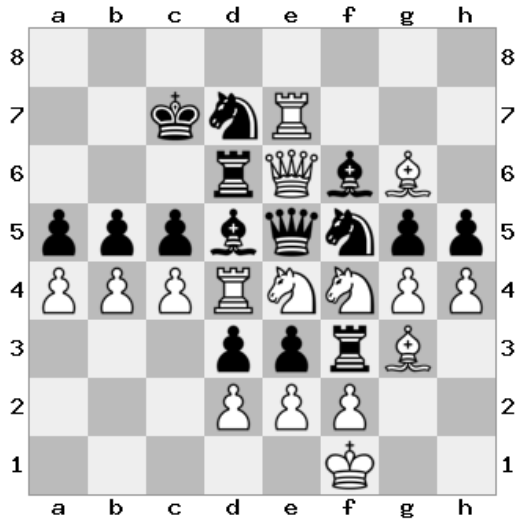


Obrázek 5.1: Bílý na tahu, statické ohodnocení pozice by signalizovalo výhodu černého

Pro řešení problému horizontu je k dispozici několik rozšíření vyhledávacího algoritmu. Tato rozšíření se snaží identifikovat místa, ve kterých hrozí projevy efektu horizontu, a pro tato místa prohlubovat vyhledávací strom.

5.1.1 Hledání klidu

Klidná pozice resp. pozice s *vysokou mírou klidu* je v šachové partii taková pozice, ve které je k dispozici relativně malé množství tahů vedoucích k drastické změně ve vývoji partie. Je to tedy taková pozice, kterou můžeme bezpečně ohodnotit, aniž bychom se museli bát, že toto ohodnocení bude příštím tahem zneplatněno. Opakem klidné pozice je pozice na obrázku 5.2 s *vysokou mírou turbulence* viz [9]:



Obrázek 5.2: Pozice s vysokou mírou turbulence

Jedním z možných přístupů je zavedení ukazatele q pro míru klidu pozic. Nachází-li se vyhledávací algoritmus v jednom z listů vyhledávacího stromu, určí q dané pozice. Tahy vedoucí k sebrání figury a tahy, které vystavují jednoho z králů šachu, jsou hlavními faktory, které je zapotřebí při výpočtu q brát v úvahu. Dále je možné výpočet q rozšířit tak, aby bral v úvahu další šachové prvky, které mohou vést k rychlé změně ohodnocení pozic, jako např. vazby, vidličky či povýšení pěšců. Po sestavení rovnice pro výpočet q je zapotřebí určit maximální hodnotu q, kdy je pozice ještě považována za klidnou. Při překročení takové hranice je zapotřebí v daném listu stromu prohloubit prohledávání, dokud je q vyšší než přípustná hodnota. Konkrétní způsob výpočtu q a jeho maximální přípustná hodnota záleží na konkrétní implementaci. Čím přísněji je hranice pro maximální q navržena, tím více je redukován efekt horizontu. Cenou za potlačení efektu horizontu je ovšem zvýšení doby výpočtu jako důsledek potřeby prohlubování prohledávání.

Šach při hledání klidu

Je-li jedna ze stran v šachu při ohodnocování pozice v listu vyhledávacího stromu, je zapotřebí vyhledávání prohloubit. Podle šachových pravidel je hráč, který je v šachu, na tuto hrozbu nucen neprodleně reagovat. Minimální rozšíření prohledávání je tedy jeden půltah. Takové řešení ale nemusí být dostatečné, jelikož hrozba nemusí být zažehnána ukrytím krále před prvním

šachem. Ve skutečnosti nastávají v šachové hře takové situace, kdy je král nucen čelit posloupnostem opakovaného šachování. Pro řešení takových případů je vhodné prohledávání prohlubovat, dokud není dosažena pozice bez hrozby šachu či dokud není dosažena předem daná horní hranice počtu prohloubených tahů pro případy, kdy se jedná o nekonečnou sekvenci šachování.

Braní figur při hledání klidu

Je-li možné pro pozici v listu vyhledávacího stromu brát figury, je prohledávání prohloubeno. Takové prohlubování může mít negativní dopad na celkovou dobu výpočtu, protože možnost sebrání některé ze soupeřových figur je poměrně častým jevem. K dalšímu zpomalení výpočtu dochází, snažíme-li se pro pozici s vysokou turbulencí propočítat všechny možné kombinace tahů vedoucích k výměně napadených figur. Pro tyto případy je možné použít některou z následujících heuristik.

MVV/LVA

MVV-LVA (Most Valuable Victim / Least Valuable Aggressor) je heuristika, která slouží k zjednodušení pozic s vysokou turbulencí. Heuristika vytvoří seznam všech soupeřových figur, které je v dané pozici možné sebrat, a všech figur, které na soupeřovy figury útočí. V každé iteraci vybere nejhodnotnější figuru ze seznamu napadených figur a pokusí se ji sebrat nejméně hodnotnou figurou ze seznamu útočníků. Výhodou tohoto algoritmu je bezesporu jeho rychlost, nevýhodou je omezení kombinační síly šachového programu. Hlavním nedostatkem tohoto přístupu je ignorování principu krytí figur. Algoritmus tedy upřednostní sebrání kryté figury (např. střelce střelcem) před sebráním méně hodnotné figury (např. pěšce střelcem), i když je zcela nekryta.

Statické ohodnocení výměny

Statické ohodnocení výměny prozkoumává posloupnost výměn figur pro jedno konkrétní pole šachovnice. Po vyhodnocení výměn pro dané pole je jako návratová hodnota vrácena změna ohodnocení partie po provedení posloupnosti výměn figur na tomto poli. Při použití tohoto přístupu není zapotřebí prohlubovat prohledávací strom, protože jsou brány v úvahu výměny figur na

jednom poli. Na rozdíl od MVV/LVA statické ohodnocení výměny nezanedbává některé z dostupných kombinací výměn figur. Pseudokód pro statické ohodnocování výměn vypadá následovně:

```
Function staticExchange(square, whiteMove):
    piece = getLeastValuableAttacker(square, turn);
    if piece NOT null then
        captured = capture(piece, square);
        return max(0, captured – staticExchange(square, !turn));
    end
    else
        return 0;
    end
```

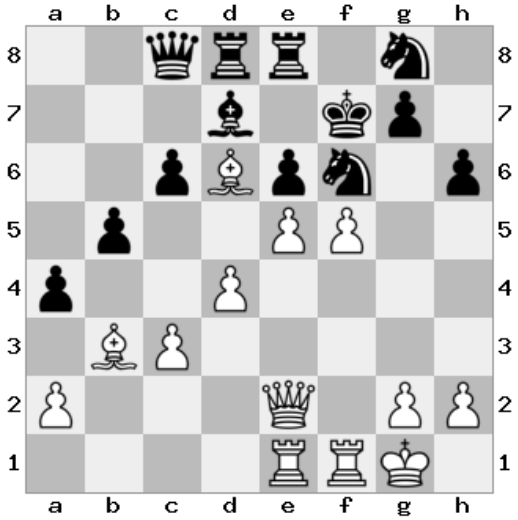
Algoritmus 4: Statické ohodnocování výměny

Algoritmus se pro dané pole pokusí sebrat soupeřovu figuru vlastní figurou s nejnižší možnou hodnotou. Pokud je takový tah dostupný, odečte algoritmus materiálový zisk soupeře v následné posloupnosti výměn od hodnoty takto získané figury. Pokud je návratová hodnota vyšší než nula, jedná se o výhodný tah.

5.1.2 Obět' figur

V kapitole o ohodnocovací funkci bylo popsáno mnoho různých aspektů šachové hry, které dávají hráči výhodu nad soupeřem. Při tvorbě ohodnocovací funkce je ale zapotřebí klást důraz na materiální stránku partie. Ve většině případů ani kombinace pozičních výhod nepřebije ohodnocení za zisk figury. Z tohoto důvodu šachový engine nerozumí konceptu dlouhodobé oběti. Pokud šachový algoritmus nevidí nevyhnutelný zisk za svou investici v rámci vyhledávacího stromu, takový tah jednoduše zavrhne.

Pokud by šachový engine měl provést obět', musel by si být jist výsledkem takové akce. Riziko je totiž příliš velké, protože neúspěšná obět' figury vede většinou k rychlé prohře. Rozdíl mezi správnou a nesprávnou obětí může přitom být minimální, viz [9]. V pozici na obrázku 5.3 je možné obětovat dámu za bílého a posloupností devatenácti půltahů dát černému mat. K vyjádření jednotlivých tahů jsem použil šachovou notaci viz příloha B.



Obrázek 5.3: Bílý obětuje dámu na h6

Kombinace vedoucí k matu vypadá následovně: 1. Qe2-h5+ Nf6×h5 2. f5×e6+ Kf7-g6, 3. Bb3-c2+ Kg6-g5, 4. Rf1-f5+ Kg5-g6, 5. Rf5-f6+ Kg6-g5, 6. Rf6-g6+ Kg5-h4, 7. Re1-e4+ Nh5-f4, 8. Re4×f4+ Kh4-h5 9. g2-g3 cokoli, 10. f4-h4 mat.

Pokud výchozí pozici pozměníme a přesuneme černého pěšce z h6 na h5, není již možné královnu obětovat se stejným výsledkem, protože po třetím tahu skončí série šachů bílého. Tato posloupnost tahů vypadá následovně: 1. Qe2xh5+ Nf6xh5 2. f5xe6+ Kf7-g6, 3. cokoli Kg6-h6. Tahem černého krále na h6 končí série vynucených reakcí a královna tedy byla obětována zbytečně.

Pro šachový program, který používá pouze jednoho čísla k ohodnocení pozic v partii, je velmi obtížné rozlišit mezi správnou a nesprávnou obětí. Leží-li řešení ve stanovené hloubce prohledávání, je konečné řešení nalezeno a figura je obětována. Leží-li konečný výsledek hlouběji, než je program schopen prozkoumat, nelze pro většinu případů pomocí pozičních aspektů ohodnocovací funkce s jistotou říci, jaký bude konečný zisk ze zkoumané oběti figury.

Vyjdeme z předpokladu, že figuru není výhodné obětovat, dokud si nejsme jisti výsledkem této akce. Potřebujeme tedy šachový program přimět, aby při prozkoumávání pozic upřednostnil ty tahy, které mají vysoký potenciál. Pokud by se podařilo takové tahy určit, bylo by možné je nejen prozkoumat před tahy ostatními, ale zároveň by bylo výhodné pro ně prohloubit prohledávání. K tomuto účelu by bylo vhodné použít ukazatel q pro míru klidu

z kapitoly o hledání klidu. Pokud ohodnocujeme pozici s vysokým q, existuje vysoká pravděpodobnost, že se ohodnocení pozice po provedení několika tahů bude značně lišit od ohodnocení pozice aktuální. Je tedy možné dočasně upřednostnit pozici s vysokým potenciálem a důkladně ji prozkoumat a pří neúspěchu se vrátit zpět ke stabilnějším tahům.

6 Implementace

Implementace vychází z programu vytvořeného v rámci bakalářské práce [8]. Cílem diplomové práce bylo rozšířit herní schopnosti šachového programu a nalézt řešení pro objevivší se nedostatky.

6.1 Výchozí stav programu

Program je implementován v programovacím jazyku Java převážně z důvodu osobní preference. Projekt je možné zprovoznit pro Java Runtime Environment *JRE 1.8.0* [20] a novější.

Při vytváření původního programu byly použity následující postupy:

- reprezentace šachovnice variantou 10x12
- indikátory pro právo na rošádu a braní mimochodem
- vyhledávací algoritmus MinMax
- vyhledávací strom o výšce čtyři (výchozí pozice a tři reakce)
- pět složek ohodnocovací funkce: materiál, postavení vůči středu šachovnice, postup pěšců, pohyblivost figur a konec partie
- uživatelské rozhraní s využitím knihovny Java Swing, Ikony figur podle [21]
- zápis partie v uživatelském rozhraní
- možnost exportu tahů odehrané partie do textového souboru

6.2 Změna původní implementace

V původní implementaci byly informace o stavu pozic předávány zkopírováním obsahu polí šachovnice. Tento přístup se prokázal jako vysoce neúsporný, jelikož program musel vytvořit kopii pozice pro každý uzel vyhledávacího

stromu. Problém byl částečně vyřešen implementací metody *unmakeMove()*, kterou je možné vrátit pozici do stavu před provedením metody *makeMove()*. Při vracení tahů bylo nutné zálohovat ty aspekty šachové pozice, ke kterým není možné udělat inverzní operaci, jako např. právo na braní mimochodem či právo na rošádu. Pro případy, kdy bylo zapotřebí uchovat místo posloupnosti tahů celou šachovou pozici, jsem využil hashovací funkci.

6.3 Rozdělení tříd

V této kapitole popíšu stěžejní třídy, které se podílejí zprostředkování funkcionality programu. Tyto stěžejní třídy jsou vyobrazeny na diagramu v příloze I diplomové práce. Pro přehlednost jsem u jednotlivých tříd vypustil většinu vedlejších metod a proměnných.

ChessEngine.java

Tato třída byla již použita v původní implementaci. Třída `ChessEngine` se stará o korektní provádění šachových tahů. V programu je použit tzv. pseudolegální přístup generování tahů. Při generování tahů pro vyhledávací algoritmus jsou nalezeny všechny tahy, které splňují definovaná pravidla pohybu. Konečnou legálnost tahu je ale možné ověřit až po provedení daného tahu, kdy je zkontrolováno, zda se nenachází král strany, která právě provedla tah, v šachu. Pokud by se král této strany v šachu nacházel, je zapotřebí navrátit stav šachovnice do pozice před tahem a označit tento tah jako neproveditelný.

`boolean makeMove(short fromX, short fromY, short toX, short toY)`
Metoda využívaná k provedení tahů partie. Prokáže-li se provedený pseudolegální tah jako neproveditelný, je partie vrácena do původního stavu a vráceno false jako návratová hodnota.

`boolean unmakeMove(short fromX,short fromY, short toX,short toY)`
Jedná se o inverzní operaci k metodě makeMove(). Je zapotřebí brát ohled na nevratné aspekty šachových pozic a ošetřit je odděleně.

ChessBoard.java

Tato třída byla také součástí původní implementace. Účelem této třídy je zprostředkování šachové hry pro uživatele. `ChessBoard` se tedy stará o zahájení hry, zobrazení grafického uživatelského rozhraní a získávání tahů od umělé inteligence.

ChessGameUI.java

Třída `ChessGameUI` byla také v pozměněné podobě využívána v implementaci pro bakalářskou práci. V původní verzi byla využívána metoda `findNextMove()`, která s využitím algoritmu MinMax hledala odpověď umělé inteligence na vzniklou situaci na šachovnici. Vyhledávací algoritmus byl do značné míry pozměněn. Vyhledávání je aktivováno metodou `search()`, detaily implementace popíšu v pozdější části této kapitoly.

PositionEvaluator.java

V této třídě je prováděna část statického ohodnocování pozic. Třída se zaměřuje na takové faktory ohodnocovací funkce, pro které není zapotřebí generovat další tahy. Výpočet ohodnocení pozic bude rozebrán podrobněji v kapitole o implementaci ohodnocovací funkce.

PossibleMovesCalculator.java

Třída `PossibleMovesCalculator` se zabývá složkami ohodnocovací funkce, pro které je zapotřebí brát v úvahu pohyb figur. Ohodnocení pozic je vypočítáváno pro pseudo-legální tahy z důvodu nižší výpočetní náročnosti. Jednotlivé složky ohodnocení budou popsány v kapitole o implementaci ohodnocovací funkce.

OpeningDatabase.java

Jedná se o implementaci databáze pro zahájení. Třída načítá vstupy databáze ze serializovaného souboru a poskytuje šachovému algoritmu tahy spojené s konkrétními pozicemi.

EndingDatabase.java

Databáze koncových her zprostředkovává posloupnosti tahů potřebné k zakončení partií, ve kterých aktuální kombinace figur odpovídají vytvořeným scénářům.

TranspositionTable.java

Transpoziční tabulka je využívána k ukládání vypočtených ohodnocení pro jednotlivé pozice. Pozice jsou prostřednictvím hashovací funkce vkládány do tabulky spolu s patřičným ohodnocením a informace o pozicích jsou posléze využity pro urychlení vyhledávání. Je-li tabulka přeplněna, jsou odstraněny nepotřebné záznamy.

HistoryTable.java

Podobně jako třída `TranspositionTable` je i tato třída použita ke zprostředkování úložiště tabulkových hodnot, tentokrát pro History heuristiku.

6.4 Ohodnocování tahů

Výhodou jednoduchosti ohodnocovací funkce použité v původním programu byla bezesporu výpočetní nenáročnost. Ohodnocovací funkce, která se skládá z malého počtu složek, nabízí kompenzaci ve formě časové úspory a možnosti prozkoumání vyššího počtu pozic. Zjednodušením výpočtu ohodnocení pozic se šachový program ještě více vzdaluje lidskému způsobu uvažování nad partií a naopak se přibližuje k brutální síle výpočtu. Na druhou stranu, je-li použito příliš mnoho složek ohodnocovací funkce, dochází nejen ke zpomalení

výpočtu ohodnocení pozic, ale i k překrývání jednotlivých šachových technik. Příkladem by bylo dilema, zda je výhodnější umístit věž z kraje šachovnice do centra za cenu snížení pohyblivosti. Cílem bylo navrhnout a vyladit ohodnocovací funkci tak, aby výsledné ohodnocení co nejlépe odpovídalo situaci na šachovnici.

K výpočtu hodnoty ohodnocovací funkce je použit lineární přístup. Ohodnocení je vypočítáno jako součet bonusů za příznivé faktory na šachovnici a postihů za faktory nepříznivé. Ohodnocovací funkce dodržuje zásady přístupu NegaMax, kde výsledné ohodnocení označuje výhodu či nevýhodu strany, která je aktuálně na tahu.

V ohodnocovací funkci jsem se rozhodl nezohlednit některé ze složek zmíněných v kapitole o ohodnocovací funkci. Jedním z nepoužitých faktorů je prvek vazby. Hlavním důvodem, proč tento prvek nebyl zahrnut do rovnice pro ohodnocovací funkci, byla výpočetní náročnost potřebná pro detekci vazeb, zejména v případě situační vazby by bylo zapotřebí provádět a posléze vracet celé posloupnosti tahů. Následující seznam obsahuje faktory, které jsem se zahrnul do ohodnocovací funkce. Poziční složky ohodnocovací funkce jsou následující:

- materiálové ohodnocení figur
- bezpečnost krále
- postavení vůči centru šachovnice
- struktura pěšců
- změna ohodnocení jednotlivých figur v závislosti na fázi partie
- změna ohodnocení jednotlivých figur v závislosti na kombinaci figur na šachovnici
- bonusy za výhodné umístění figur (např. věž na otevřeném sloupci)
- detekce nedostatku figur pro zakončení partie

Pohyblivostní složky ohodnocovací funkce jsou následující:

- napadání soupeřových figur s ohledem na hodnotu napadené figury
- bránění vlastních figur s ohledem na hodnotu bráněné figury

- pohyblivost figur s ohledem na druh figury a směr pohybu
- speciální případy zvýšené hrozby (např. vidlička)
- speciální případy neefektivity figur (např. pasivní střelec)

Samotný výpočet statického ohodnocení pozic probíhá takto:

OHODNOCENÍ =
MATERIÁLOVÉ_OHODNOCENÍ_FIGUR +
BONUS_ZA_POHYBLIVOST +
BONUS_ZA_DOPŘEDNÝ_POHYB_U_STŘELCE_A_VĚŽE +
BONUS_ZA_VERTIKÁLNÍ_POHYBLIVOST_PRO_VĚŽE +
BONUS_ZA_KRYTÍ_VLASTNÍCH_FIGUR +
BONUS_ZA_NAPADÁNÍ_SOUPEŘOVÝCH_FIGUR +
BONUS_ZA_PÁR_STŘELCŮ +
BONUS_ZA_POSTUP_PĚŠCŮ +
BONUS_ZA_POSTAVENÍ_U_STŘEDU_ŠACHOVNICE +
BONUS_PRO_VĚŽ_NA_SEDMÉ_NEBO_DRUHÉ_ŘADĚ +
BONUS_PRO_ZDVOJENÉ_VĚŽE_NA_SEDMÉ_ČI_DRUHÉ_ŘADĚ +
BONUS_PRO_VĚŽ_V_KONCOVÉ_FÁZI +
BONUS_PRO_VĚŽ_NA_VOLNÉM_SLOUPCI +
BONUS_PRO_VĚŽ_NA_POLOOTEVŘENÉM_SLOUPCI +
BONUS_PRO_VOLNÉ_PĚŠCE +
BONUS_PRO_SPOJENÉ_VOLNÉ_PĚŠCE +
BONUS_PRO_KRÁLOVNU_VERSUS_VĚŽE (s lehkými figurami) +
BONUS_ZA_VIDLIČKU_NA_CENNĚJŠÍ_FIGURY +
BONUS_PRO_TŘI_LEHKÉ_FIGURY_VERSUS_KRÁLOVNA +
BONUS_PRO_KRÁLE_V_OBRANNÉ_POZICI +
BONUS_ZA_ŠTÍT_Z_PĚŠCŮ_U_KRÁLE +
POSTIH_ZA_KRÁLE_BLÍZKO_STŘEDU_ŠACHOVNICE +
POSTIH_ZA_ZTRÁTU_NÁROKU_NA_ROŠÁDU +
POSTIH_PRO_JEZDCE_PŘI_NÍZKÉM_POČTU_PĚŠCŮ +
POSTIH_ZA_NEAKTIVNÍ_STŘELCE +
POSTIH_ZA_POSTRANNÍ_PĚŠCE +
POSTIH_PRO_VĚŽE_PRO_ZAČÁTEK_PARTIE +
POSTIH_ZA_NADBYTEČNOU_VĚŽ +
POSTIH_ZA_ZDVOJENÉ_PĚŠCE +
POSTIH_ZA_ZTROJENÉ_PĚŠCE +
POSTIH_ZA_IZOLOVANÉ_PĚŠCE +
POSTIH_ZA_PŘEDČASNÝ_VÝVIN_KRÁLOVNY +
POSTIH_ZA_NEVYVINUTÉ_LEHKÉ_FIGURY.

Konkrétní hodnoty pro ohodnocení figur jsem zvolil následovně: 900 pro královnu, 500 pro věž, 300 pro jezdce a střelce a 130 pro pěšce. Výši bonusů a postihů pro ohodnocovací funkci jsem volil v řádu desítek ohodnocovacích bodů tak, aby ani kombinace více pozičních výhod svou hodnotou nepřesáhla hodnotu lehké figury. Konkrétní hodnoty jsou k dispozici v příloze H. Principy uplatňované pro šachová zahájení jsou zahrnuty do obecné ohodnocovací funkce. Nejčastěji používaná šachová zahájení jsou navíc pokryta databází pro zahájení. Pro koncovou fázi partie byla vytvořena vlastní ohodnocovací funkce. Výsledné ohodnocení pozic je pak vypočteno přechodným ohodnocováním takto:

$$total\_eval = general\_eval + midgame\_eval * F + endgame\_eval * (1 - F)$$

Proměnná F označuje fázi hry a nabývá hodnot od 0 do 1. Hodnota F je vypočítána jako součet ohodnocení figur na šachovnici vydělený součtem ohodnocení figur při začátku partie. Samotná funkce pro koncové ohodnocení znázorňuje obecně platné strategie pro koncové části partií jako např. centralizaci krále či postup pěšců. Pro konkrétní případy závěrečných fází partií byla vytvořena databáze koncových her a tzv. Mop-Up ohodnocovací funkce.

6.5 Mop-Up ohodnocování

Pro vyhodnocování pozic v závěrečných fázích partie je použita *Mop-Up ohodnocovací funkce*. Funkce se inspiruje rovnicí, která byla navržena v [27], a upravená verze je použita v programu. Šachový program používá původní ohodnocovací funkci, dokud nejsou splněny podmínky pro aktivaci Mop-Up ohodnocování. Aby mohla být Mop-Up funkce použita, musí být rozdíl materiálového ohodnocení bílého a černého hráče alespoň 400 (čtyři pěšci). Dále součet materiálu strany, která prohrává, nesmí přesáhnout 700. Třetí podmínkou je, že útočící strana musí mít buď alespoň jednu těžkou figuru, nebo žádného pěšce a takovou kombinaci lehkých figur, aby bylo možné dát soupeři mat. Jsou-li tyto podmínky splněny, přechází vyhledávací algoritmus na Mop-Up ohodnocování. Mop-Up ohodnocení je vypočteno rovnicí:

$$eval = 28 * CMD + 5 * (14 - MD)$$

CMD je zkratka pro Center Manhattan Distance [24]. V tomto případě CMD označuje vzdálenost krále bránící se strany od středu šachovnice. Pri-

mární snahou je tedy zatlačit krále, kterému se pokoušíme dát mat, do rohu šachovnice. Zkratka *MD* označuje vzdálenost krále a jezdců útočící strany od soupeřova krále. Již první složka rovnice většinou stačí k tomu, aby figury, které se mohou pohybovat po celé šachovnici, začaly vytlačovat soupeřova krále směrem ke kraji šachovnice. Často ale nejsou pozice prohledány do takové hloubky, aby se i figury jako král a jezdec začaly přesunovat směrem k bránícímu se králi, a právě proto byla zavedena druhá složka rovnice. Nakonec je k Mop-Up ohodnocení je přičteno ohodnocení materiálové, aby si program zachoval motivaci neztrácet své figury a výsledná hodnota je použita ve vyhledávání. Navíc, klesne-li hodnota figur na šachovnici pod 1500 (hodnota 3 věží), je situace dostatečně jednoduchá natolik, že je možné zvýšit hloubku prohledávání. Pro tyto případy je zvýšena hloubka prohledávání o 2.

6.6 Vyhledávací algoritmus

Implementace vyhledávacího algoritmu vychází z algoritmu Alfa-Beta prořezávání. Hlavní snahou bylo zefektivnit ořezávání takových větví vyhledávacího stromu, které mají nízký potenciál pro ovlivnění konečného výsledku. Byly zkombinovány techniky zmíněné v teoretické části tak, aby vyhledávací algoritmus opustil podstrom vyhledávacího stromu ve chvíli, kdy se další prozkoumávání tohoto podstromu jeví jako zbytečná činnost. Nelze zaručit, že použitím některých technik nebude přehlédnuto nejlepší dostupné řešení situace, je to ale přiměřená cena za zisk vyšší hloubky prohledávání. Další snahou bylo řadit zkoumané tahy tak, aby tahy s nejvyšším potenciálem byly zkoumány jako první a tím bylo dosaženo dalšího urychlení algoritmu.

Samotné vyhledávání vychází z *Alfa-Beta* algoritmu zapsaném v *NegaMax* notaci. Již prozkoumané pozice jsou ukládány v *transpoziční tabulce* a uložená ohodnocení jsou opětovně používána, narazí-li algoritmus na pozici, která již byla jednou prozkoumána. Dále byla použita technika *Razoringu* pro odřezávání podstromů ve hloubce 2, tedy dvě úrovně nad listy vyhledávacího stromu. Technika Razoringu je poměrně riskantní, a proto nebyla použita na vyšších úrovních. Na uzly ve hloubce 1 byl aplikován *Futility pruning* za účelem snížení celkového počtu volání ohodnocovací funkce. Dospěje-li vyhledávací algoritmus do hloubky 0, tedy do listu vyhledávacího stromu, jsou prozkoumány všechny tahy vedoucí k sebrání figury *Statickým ohodnocováním výměn*. Hloubka prohledávání je dále zvýšena pro pozice v listu stromu, kdy se král nachází v šachu, a to maximálně o čtyři půltahy. Průchod vyhle-

dávacím algoritmem je popsán podrobně v příští podkapitole.

Řazení tahů je provedeno kombinací *transpoziční tabulky*, *Killer heuristiky* a *History heuristiky*. Jednotlivé položky transpoziční tabulky obsahují údaje o již provedených průzkumech pozic. Pro každou z těchto pozic je uloženo nalezené ohodnocení a hloubka prohledávání. Pokud pro danou položku v transpoziční tabulce platí, že hloubka prohledávání, do které byla tato pozice prozkoumána, je nižší než hloubka maximální, nemůže být tato položka použita jako náhrada plného průzkumu této pozice. Je ale možné ohodnocení z transpoziční tabulky pro takovou pozici využít k řazení tahů. Existuje totiž vyšší šance, že ohodnocení pozice prozkoumané do omezené hloubky bude přesněji odpovídat skutečnosti než statické ohodnocení téže pozice bez jakéhokoli dalšího průzkumu. Killer heuristika je implementována jako dva seznamy tahů. První seznam uchovává primární Killer tahy a druhý seznam uchovává sekundární Killer tahy. Při spuštění vyhledávacího algoritmu jsou oba dva seznamy pročištěny, aby se Killer tahy nepřenášely mezi jednotlivými vyhledáváními. Způsobí-li tah odříznutí větve vyhledávacího stromu, je uložen do seznamu Killer tahů pro hloubku vyhledávání, ve které k odříznutí došlo. Pro jedno konkrétní vyhledávání je maximální kapacita nastavena na dva Killer tahy pro každou úroveň prohledávání. Objeví-li se v dané hloubce více Killer tahů, nově nalezený tah jednoduše nahradí původní Killer tah v seznamu sekundárních Killer tahů. History heuristika na rozdíl od Killer heuristiky uvažuje spojitosti mezi jednotlivými vyhledáváními, je tedy zapotřebí uchovávat informace potřebné pro výkon History heuristiku napříč běhu programu. Jednotlivé položky History tabulky obsahují tahy, které někdy za dobu běhu programu vedly k odříznutí větve vyhledávacího stromu. Pro každý z těchto tahů je navíc uloženo interní ohodnocení, které znázorňuje, jak často tento konkrétní tah způsobil odříznutí větve. Dojde-li k odříznutí větve na jakékoli úrovni vyhledávacího stromu, je History tabulka aktualizována následovně:

```
if alpha >= beta then
    historyTable.add(move, depth * depth);
end
```

Algoritmus 5: Aktualizace History tabulky

Druhá mocnina aktuální hloubky vyjadřuje ohodnocení, které je přičteno k danému tahu v history tabulce. Účelem druhé mocniny hloubky je zmírnit účinek takových tahů, které způsobily odříznutí větve stromu ve spodních patrech, kde se hloubka blíží nule. Způsobuje-li tah opakovaně odřezávání algoritmu Alfa-Beta, vystoupá brzy na první pozice History tabulky. Z důvodu

regulace velikosti History tabulky je postupně ohodnoceni jednotlivých tahů dekrementováno a tahy, které mají v rámci History tabulky podprůměrné ohodnocení, jsou vyřazovány. Samotné řazení probíhá následovně:

```
currentPosEval = evalFunction(engine);
for each object m from Moves do
    m.makeMove();
    if transposTable.hasEntry(engine) then
        m.setEval(transpositionEntry.getEval());
    else if isKiller(m, depth) then
        m.setEval(currentPosEval + 1);
    else if isHistory(m) then
        m.setEval(currentPosEval);
    else
        m.setEval(evalFunction(engine) – ORDERING_PENALTY);
    end
    m.undoMove();
end
sort(Moves);
```

Algoritmus 6: Řazení tahů

Pro ohodnocovanou pozici jsou nalezeny všechny dostupné pseudo-legální tahy. V cyklu je pro každý z tahů vyzkoušeno, zda je možné daný tah provést. Potvrdí-li se, že je tah legální, pokusí se algoritmus najít tah v transpoziční tabulce. Pokud transpoziční tabulka obsahuje záznam pro pozici, která nastane po provedení tahu, který plánujeme řadit, je ohodnocení z transpoziční tabulky přiřazeno k tomuto tahu. Není-li nalezen záznam v transpoziční tabulce, přichází na řadu Killer a History heuristiky. Tahům, které se nachází v seznamu pro Killer tahy, je přiřazeno dočasné ohodnocení o jedna vyšší, než bylo ohodnocení pozice před provedením tahu. Tahy History heuristiky jsou ohodnoceny o jedna méně, než Killer tahy. Nakonec, nebyl-li tah nalezen v žádném ze seznamů, je vypočítáno statické ohodnocení pozice po provedení tahu a od tohoto ohodnocení je odečtena polovina hodnoty lehké figury. Tímto způsobem jsou tahy, které nebyly nalezeny ostatními technikami, znevýhodněny a většina z nich se po sestupném seřazení seznamu tahů zařadí na jeho konec. Výjimkou jsou takové tahy, které vedou k sebrání figury, protože pro takové tahy statické ohodnocení značně naroste a proto se zařadí mezi první pozice seznamu.

6.7 Průchod vyhledávacím algoritmem

Nyní se pokusím podrobněji znázornit, jakým způsobem pracuje vyhledávací algoritmus. Na následujícím diagramu je vyobrazena první část vyhledávacího algoritmu. Diagram pokrývá vyhledávací rutinu od vstupního bodu až po řazení tahů. Algoritmus nejdříve zjistí, zda je možné získat ohodnocení zkoumané pozice, aniž by bylo zapotřebí pozici prohledávat. K tomu slouží databáze zahájení, databáze koncových her a tabulka transpozic.

Dále je zjištěno, kolik legálních tahů je pro aktuální pozici k dispozici. Aby bylo možné potvrdit legálnost tahu, je z důvodu návrhových rozhodnutí nutné tento tah nejdříve provést. Naštěstí je jen zapotřebí zjistit, zda je počet dostupných tahů roven nule, jedné nebo je vyšší než jedna. Není-li k dispozici žádný tah, nachází se pozice buď v matu (král na tahu má šach, ale nemá žádná volná pole) nebo v patu (král sice nemá žádná volná pole, ale není v šachu). Je-li k dispozici pouze jeden tah a nacházíme-li se v kořenu vyhledávacího stromu, je možné aplikovat tzv. *Chopper techniku*. Jedná se o jednoduchou metodu, kdy vybereme jediný dostupný tah jako řešení aktuální pozice a nezkoumáme ohodnocení tohoto tahu, protože je irelevantní.

Protože vyhledávací algoritmus využívá rekurze, je zapotřebí stanovit *ukončovací podmínky*. Touto ukončovací podmínkou je dosažení maximální hloubky vyhledávacího stromu. Dalším požadavkem pro ukončení rekurze je, aby se král v koncové pozici *nenacházel v šachu* (hledání klidu). Pro případy, kdy se král v koncové pozici v šachu nachází, je vyhledávání prohloubeno o další dva půltahy. Maximální míra tohoto prohloubení jsou čtyři půltahy, protože by mohlo dojít k situacím, kdy by měla jedna strana k dispozici tzv. nekonečný šach a rekurze by nikdy neskončila.

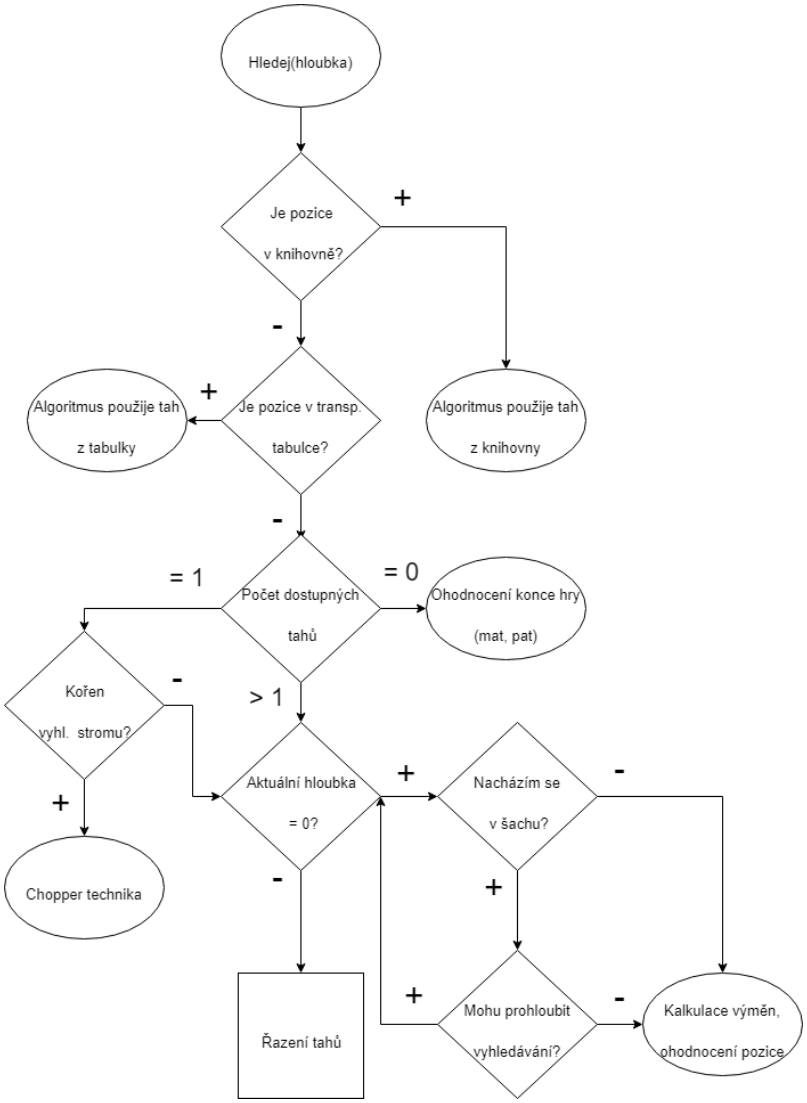
Není-li král v koncové pozici v šachu, je možné přejít ke statickému ohodnocení. Nejdříve jsou prozkoumány všechny tahy vedoucí k sebrání figury *Statickým ohodnocením výměn*, které bylo popsáno v teoretické části. Výhodou tohoto přístupu je zisk dodatečné hloubky prohledávání pro výměny figur, aniž by bylo zapotřebí provádět další tahy. Metoda Statického ohodnocení výměn nalezne maximální možný materiální zisk strany, která je v koncové pozici na tahu, docílený výměnou figur na jednom konkrétním poli šachovnice.

Tento přístup má dva nedostatky. Hlavním nedostatkem je, že přístup Statického ohodnocování nepokrývá případy, kdy jsou v po sobě jdoucích tazích vyměňovány figury na různých polích šachovnice. Aby bylo možné vyřešit

tento typ situací, bylo by zapotřebí spustit nové vyhledávání a prozkoumat všechny kombinace výměn figur. Takové řešení by bylo ale příliš výpočetně náročné vzhledem k faktu, že se nacházíme v listu vyhledávacího stromu a naší snahou je příslušnou pozici ohodnotit a nespouštět další vyhledávací rutinu.

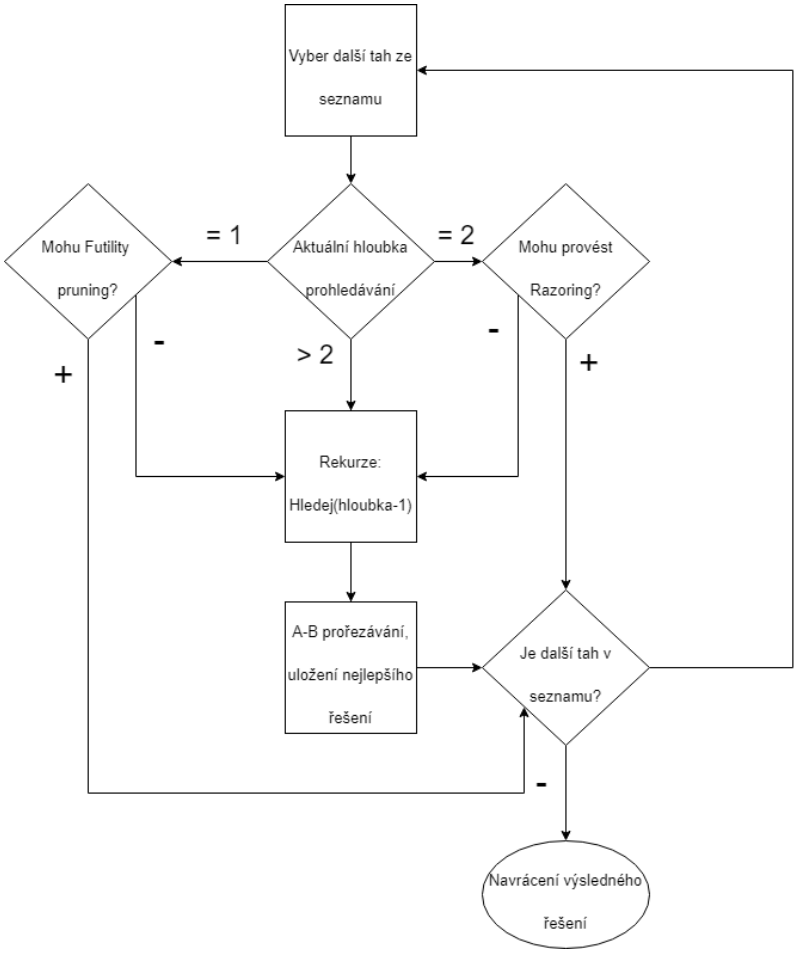
Druhým nedostatkem aktuální implementace je neuvažování odkrytých hrozeb při vyhodnocování výměn na jednom z polí šachovnice. Na začátku metody je vytvořen seznam útočníků a obránců daného pole, ze kterého jsou figury postupně odebírány a výsledné ohodnocení výměny je patřičně upravováno. Při odebírání jednotlivých figur ze seznamu není již zkoumáno, zda přesunutím figury na šachovnici nepřibyl pro dané pole další útočník nebo obránce. Tento nedostatek je námětem pro budoucí rozšíření programu. Pro urychlení ohodnocování výměn je, podobně jako ve vyhledávací rutině, použit algoritmus Alfa-Beta. Ohodnocení výměny, které je nejvýhodnější pro stranu na tahu v ohodnocované koncové pozici, je přičteno ke statickému ohodnocení získanému voláním ohodnocovací funkce a výsledek je navrácen jako návratová hodnota. Zdrojový kód implementace ohodnocování výměn je k dispozici v příloze F a v příloze G.

V případech, kdy není možné použít Chopper techniku a zároveň není možné vyhodnotit koncovou pozici, přechází algoritmus k řazení tahů, které jsem popsal v předchozí podkapitole. Na diagramu 6.1 je znázorněn průchod algoritmem až po řazení tahů, diagram 6.2 pokračuje ve fázi, kdy jsou tahy seřazeny podle očekávané síly.



Obrázek 6.1: První část vyhledávacího algoritmu

V druhé části vyhledávacího algoritmu máme již k dispozici seznam seřazených tahů podle očekávané síly. Pro zjednodušení diagramu jsem vynechal činnosti jako např. provádění a navracení tahů, aktualizaci záznamů v transpoziční, Killer a History tabulce apod. Pro každý tah v seznamu je zjištěno, zda je možné aplikovat Futility pruning pro uzly, které leží jednu úroveň nad listy vyhledávacího stromu, či Razoring pro uzly ležící dvě úrovně nad listy stromu. Pro zbylé případy je opětovně volána vyhledávací rutina pro hloubku o jedna nižší. Nepotřebné větve jsou průběžně prořezávány algoritmem Alfa-Beta a nejlépe ohodnocený tah je ukládán do speciální proměnné.



Obrázek 6.2: Druhá část vyhledávacího algoritmu

6.8 Hashovací funkce Zobrist

Aby bylo možné vytvořit databázi zahájení, databázi koncové hry a transpoziční tabulku, je zapotřebí zvolit způsob, jakým budou reprezentovány jednotlivé pozice partie. Hashovací funkce *Zobrist* představuje jeden z možných přístupů, kterým lze převést informace o pozici na jedno číslo o velikosti 64 bitů. Možné kolize a podrobnější informace o hashovací funkci Zobrist jsou dostupné zde [25].

Zpočátku je zapotřebí náhodně vygenerovat číslo o velikosti 64 bitů pro všechny kombinace figur a polí na šachovnici. V šachové hře je celkem 12

druhů figur a 64 polí, je tedy zapotřebí vygenerovat 12 * 64 = 768 čísel pro zakódování pozice. V implementaci jsem použil dalších 22 hodnot k zakódování práv na rošádu a braní mimochodem a dodatečné číslo k zakódování strany, která je v dané pozici na tahu. Jako generátor náhodných čísel jsem použil knihovní funkce jazyka Java.

Výsledná hodnota klíče hashovací funkce Zobrist je vypočtena provedením XOR operace mezi všemi vygenerovanými čísly, která odpovídají aktuálnímu stavu partie. Pro objasnění způsobu výpočtu klíče hashovací funkce přikládám pseudokód převzatý z [26].

```
white_pawn := 1;
white_rook := 2;
// etc.
black_king := 12;
Function init_zobrist():
    // fill a table of random numbers/bitstrings
    table := a 2-d array of size 64x12;
    for i from 1 to 64 do
        // loop over the board, represented as a linear
           array
        for j from 1 to 12 do
            table[i][j] = random_bitstring();// loop over the
               pieces
        end
    end
Function hash():
    h := 0;
    for i from 1 to 64 do
        // loop over the board positions
        if board[i] NOT empty then
            j := the piece at board[i], as listed above;
            h := h XOR table[i][j];
        end
    end
return h
```

Algoritmus 7: Hashovací funkce Zobrist

Pomocí takto vytvořeného klíče je možné ukládat informace o stavu pozic pro pozdější použití. Další výhodou tohoto přístupu je možnost jednoduché aktualizace klíčů pozic. Potřebujeme-li klíč nové pozice, je možné jej vypo-

čítat využitím klíče pozice původní. Stačí pouze provést operaci XOR pro každou ze změn v rozestavení figur na šachovnici, které daný tah způsobil. Zdrojový kód implementované Zobrist funkce je k dispozici v příloze E.

6.9 Databáze zahájení

Při sestavování databáze zahájení byla k zakódování jednotlivých pozic použita hashovací funkce Zobrist. Na rozdíl od transpoziční tabulky, pro kterou jsou vždy data pro výpočet klíče hashovací funkce vygenerována při jejím vytvoření, bylo v případě databáze zahájení a databáze koncové hry zapotřebí vygenerovat neměnný soubor dat, aby bylo možné k položkám databáze přistupovat při opětovném spuštění programu.

Databáze zahájení je koncipována jako mapa, kde klíčem jsou šachové pozice zakódované funkcí Zobrist a hodnotami jsou objekty obsahující seznam tahů, které je v dané pozici vhodné použít. Takto vytvořená mapa je uložena do serializovaného souboru *opening_book.ser*. Jednotlivé vstupy databáze zahájení jsem vkládal manuálně spuštěním programu v módu, ve kterém jsou ukládány zadané tahy jako knihovní. Databáze aktuálně obsahuje 310 transpozic, ve kterých se může partie v počátečních fázích nacházet. Pro každou z těchto transpozic je uložen seznam doporučených tahů, ze kterého šachový algoritmus jeden náhodně vybere. Při zadávání zahajovacích scénářů jsem se inspiroval veřejně dostupnou databází šachových zahájení [28]. Volil jsem nejčastěji používané scénáře a sekvence tahů jsem vkládal v rozmezí od 4 do 11 půltahů.

6.10 Databáze koncové hry

Pro implementaci databáze koncových her byla použita retrográdní analýza, která byla popsána v teoretické části. Samotná databáze koncové hry je koncipována podobně jako databáze zahájení. Pro každou z kombinací, pro které byla retrográdní analýza provedena, je vytvořen samostatný serializovaný soubor, který obsahuje šachové pozice zakódované funkcí Zobrist. Každá z pozic má přiřazené ohodnocení, které reprezentuje počet tahů potřebných k docílení matu. Na rozdíl od databáze zahájení jsou v databázi koncové hry zakódovány pouze pozice bez dodatečných informací jako např. práva na

rošádu či indikátor strany, která je na tahu. Databáze je vytvořena pouze z perspektivy bílého hráče bez uvažování transpozic. V případech, kdy je zapotřebí řešení koncové hry pro černého, stačí dočasně invertovat barvu všech figur na šachovnici. Jelikož v databázi koncové hry nejsou uloženy pozice, ve kterých je strana, která se brání, na tahu, je zapotřebí pro nalezení vhodného pokračování vždy provést hledání do hloubky dvou půltahů a najít takový tah, který vynutí nepatovou pozici s počtem tahů k matu o jedna nižší.

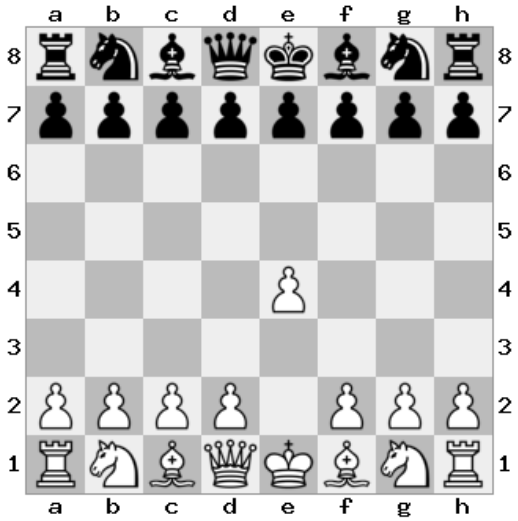
Konkrétní databáze koncové hry je programem načítána pouze tehdy, zůstane-li na šachovnici kombinace figur, pro které je databáze připravena. Druhou podmínkou pro načtení databáze je, že šachový program musí být stranou, která je v partii v převaze a snaží se dát mat. Databázi koncové hry je možné využít i pro obranné účely, pro tyto potřeby ale dostatečně poslouží Mop-Up evaluace, která se snaží bránícího se krále přesunout co nejblíže centru šachovnice, kde je nejtěžší dát králi mat. Databáze koncových her obsahuje tři scénáře: král s dámou proti králi, král s věží proti králi a král se dvěma střelci proti králi.

7 Testování

Pro vyšší přehlednost hraje v testovacích scénářích hráč (člověk) s bílými figurami a protivník (počítač s inteligentními algoritmy) s černými figurami. V adresáři *ChessGame/src/positions* jsou k dispozici pozice každého z testovacích scénářů. Uložené pozice je možné nahrát využitím grafického rozhraní aplikace volbou *Load position*.

7.1 Test databáze zahájení

Výchozí pozice se nachází na obrázku 7.1. Bílý hráč zahajuje partii tahem na 1. e4. Umělá inteligence používá databázi zahájení a nachází 8 knihovních tahů: c7-c5, e7-e5, e7-e6, c7-c6, d7-d6, d7-d5, g7-g6, Ng8-f6 a Nb8-c6.



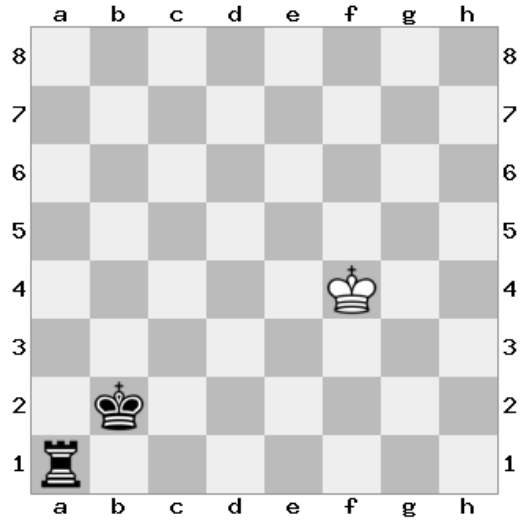
Obrázek 7.1: Pozice č.1: Zahájení

Z dostupných tahů náhodně volí tah 1...c7-c5 (pro náhodnost nelze zaručit, že testovací scénář proběhne totožně). Bílý hráč pokračuje tahem 2. Ng1-f3. Program tentokrát nachází 5 tahů: d7-d6, d7-d5, Nb8-c6, e7-e6 a g7-g6. Náhodně je zvolen tah 2...Nb8-c6. Bílý pokračuje 3. d2-d4 a program nachází v knihovně jediný tah: 3...c5×d4. Bílý pokračuje tahem 4.Nf3×d4. Černý nyní nachází tři dostupné tahy: Ng8-f6, g7-g6 a e7-e5. Černý volí tah 4...e7-

e5, bílý reaguje 5. Nd4-b5. Nyní nachází černý jeden knihovní tah: 5...d7-d6. Po tahu bílého Nb1-c3 již umělá inteligence žádné knihovní tahy nenachází a pokračuje podle vyhledávacího algoritmu. Pozice je uložena pod názvem *opening*.

7.2 Test koncové hry král s věží proti králi

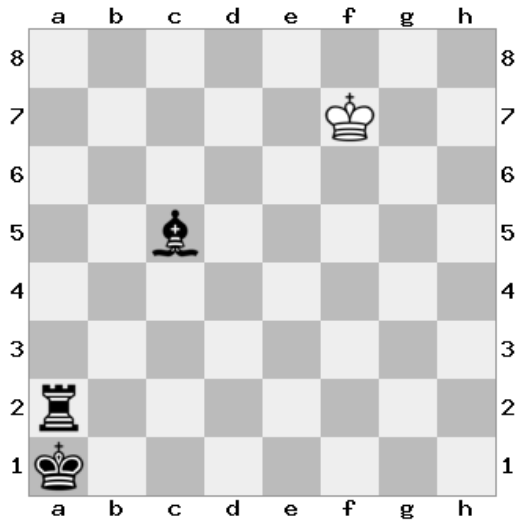
Druhá testovací pozice se zaměřuje na zakončení partie s využitím databáze pro koncovou hru. Umělá inteligence využívá předgenerovaných scénářů z databáze a postupně vytěsní bílého krále ke kraji šachovnice a partii zakončí matem. Partie pokračuje následovně: 1. Kf4-e4 Ra1-e1, 2. Ke4-d3 Kb2-b3, 3. Kd3-d4 Kb3-b4, 4. Kd4-d5 Kb4-c3, 5. Kd5-d6 Kc3-c4, 6. Kd6-c6 Re1-d1, 7. Kc6-b6 Rd1-d6, 8. Kb6-a5 Rd6-c6, 9. Ka5-a4 Rc6-a6 mat. Výchozí pozice se nachází na obrázku 7.2. Pozice je uložena pod názvem *king-rook-vs-king*. Další testovací pozice pro databázi koncové hry je možné najít pod názvy *king-queen-vs-king* (král a dáma proti králi) a *king-two-bishops-vs-king* (král a dva střelci proti králi). Protože pro tvorbu databáze koncové hry dvou střelců bylo nutné vygenerovat podstatně více pozic, je při testování tohoto koncového scénáře zapotřebí počítat s delší dobou načítání databáze.



Obrázek 7.2: Pozice č. 2: Král a věž proti králi

7.3 Test Mop-Up ohodnocování

Ve třetí testovací pozici nemá šachový algoritmus k dispozici databázi s vygenerovanými tahy. Partie se ale nachází v koncové fázi, je tedy možné využít Mop-Up ohodnocování. Primárním cílem algoritmu je donutit soupeřova krále, aby opustil centrum šachovnice. Sekundárním cílem je snížit vzdálenost mezi králi, aby bylo možné partii zakončit matem. Partie na obrázku 7.3 pokračuje takto: 1.Kf7-e6 Ra2-e2, 2. Ke6-d5 Bc5-a7, 3. Kd5-c6 Re2-d2, 4. Kc6-b7 Ba7-e3, 5. Kb7-c6 Ka1-b2, 6. Kc6-c7 Kb2-b3, 7. Kc7-c6 Kb3-a4, 8. Kc6-c7 Ka4-b5, 9. Kc7-c8 Kb5-b6, 10. Kc7-b8, Rd2-d8 mat. Pozice je uložena pod názvem *king-rook-bishop-vs-king*.

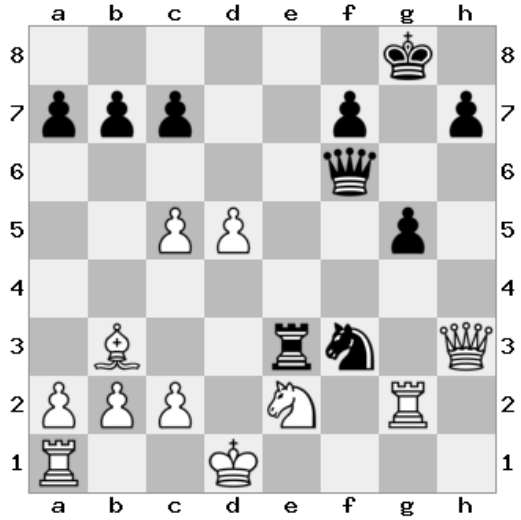


Obrázek 7.3: Pozice č. 3: Král s věží a střelcem proti králi

7.4 Test kombinačních schopností

Poslední test se zaměřuje na vyzkoušení kombinačních schopností programu. Základní hloubka vyhledávání je 5 půltahů. Tato hloubka je navyšována o další 2 až 4 půltahy pro sekvence, které končí pozicí s jedním z králů v šachu. Z tohoto důvodu je algoritmus schopný najít mat v pěti tazích. V pozici na obrázku 7.4 je černý na tahu a dává mat pěti tahy následující sekvencí tahů: 1...Qf6-d4, 2. Kd1-c1 (při 2. Ne2×d4 dává černý tahem Re3-e1 mat) 2... Qd4-d2, 3. Kc1-b1 Qd2-d1, 4. Ne2-c1 Qd1×c1, 5. Kb1×c1 Re3-e1 mat.

Pozice je uložena pod názvem *black-mates-in-five*. Další testovací pozice jsou uloženy pod názvy *black-mates-in-four* a *black-mates-in-three*.



Obrázek 7.4: Pozice č. 4: Černý na tahu dává mat pěti tahy

8 Zhodnocení řešení, možná rozšíření

8.1 Databáze zahájení

Databáze zahájení momentálně obsahuje 310 různých pozic. Každá z transpozic může být dosažena různými posloupnostmi tahů. Některé z těchto posloupností tahů jsou také součástí databáze. Moderní šachové programy používají databáze ve velikostech přesahujících 100 000 uložených pozic. Pro vytvoření databáze zahájení o takových rozměrů by bylo zapotřebí celý proces zautomatizovat.

Možným přístupem k automatizaci zpracování šachových zahájení by bylo vyjít z veřejně dostupných databází šachových partií. Pro tyto partie by bylo vhodné vyčlenit ty, které byly odehrány vysoce hodnocenými hráči na obou stranách a neskončily předčasně z důvodu hrubé chyby jedné ze stran. V takto vytvořené databázi by bylo možné uchovávat další statistiky jako např. percentuální úspěšnost tahů pro každou z pozic.

8.2 Změna reprezentace šachovnice a generování tahů

Při realizaci diplomové práce bylo navázáno na řešení z práce bakalářské. Původní řešení vychází z tzv. Mailbox reprezentace šachovnice. Šachové hrací pole o rozměru 8 x 8 polí je obaleno prázdnými poli ze všech stran tak, že vzniká struktura o rozměrech 10 x 12. Takový přístup je nenáročný na představivost a poskytuje určité výhody při generování tahů a definování pravidel pohybu. Bohužel právě generování tahů a následné ověřování legálnosti se při profilaci projevilo jako úzké hrdlo aplikace. Jako řešení tohoto problému se nabízí implementace nového šachového enginu na bázi bitových polí. V reprezentaci šachovnice bitovými poli jsou tahy definovány logickými operacemi, čímž by bylo možné dosáhnout značného urychlení v operacích nad šachovnicí. Dalším námětem pro rozšíření by byla paralelizace vyhledávacího algoritmu.

8.3 Ohodnocovací funkce

Vytvořená rovnice pro statickou ohodnocovací funkci aktuálně obsahuje 33 složek. Každá z těchto složek reprezentuje obecně platné šachové principy. Přiřazení ohodnocení každému z těchto principů je již bohužel netriviální. Pravá hodnota různých pozičních výhod a nevýhod je předmětem častých diskusí šachových mistrů a velmistrů a názory mohou být velmi rozdílné. Důležitost šachových principů bývá vnímána individuálně a přímo závisí na hráčově herní strategii.

Strategické dovednosti šachového programu jsou do vysoké míry omezeny. Z tohoto důvodu potřebuje šachový program obecně uplatnitelný soubor pravidel. Jednotlivá ohodnocení jsem volil na základě studie popsané v kapitole o ohodnocovací funkci. Bylo-li to možné, inspiroval jsem se názory šachových mistrů a velmistrů. Samotný herní projev programu nasvědčuje, že je program ochotný obětovat materiál (zpravidla pěšce) za účelem zisku domnělé poziční výhody, kterou v příštích tazích není schopný využít. Dále jsem narazil na situace, kdy algoritmus ztratil figuru z důvodu zkreslené představy o provedených výměnách figur na šachovnici. Důvodem tohoto nedostatku je aktuální implementace pro ohodnocování výměn, která vždy uvažuje výměny provedené pouze na jednom poli šachovnice. Práci by bylo možné vylepšit stanovením přesnějších hodnot pro použité bonusy a postihy a doplněním rozšiřujících pravidel pro ohodnocovací funkci.

8.4 Koncové scénáře šachových partií

V diplomové práci jsem vytvořil databázi koncových her, která pokrývá některé ze scénářů, které mohou v závěrečné části partie nastat. Pro případy, kdy je jedna ze stran ve značné materiální výhodě a snaží se soupeřovu králi dát mat, jsem použil Mop-Up evaluaci. Vytvořená implementace ale nenabízí řešení pro situace, kde jsou na šachovnici stále pěšci. Moderní šachové programy řeší tyto případy použitím databázových scénářů, je-li zbývající počet figur na šachovnici dostatečně nízký, nebo speciálními ohodnocovacími technikami, které se zaměřují na povýšení pěšců, prolomení pěšcové struktury apod.

9 Závěr

Tato práce navazuje na původní řešení vytvořené v rámci bakalářské práce. Cílem práce bylo prozkoumat metody používané při implementaci šachových algoritmů. Dále bylo úkolem rozšířit implementaci vybráním vhodných technik a ověřit funkčnost na šachových úlohách. Posledním úkolem bylo vyhodnocení dosažených výsledků a nastínění možných rozšíření.

První část diplomové práce se zabývá analýzou šachové hry. Cílem analýzy bylo nalézt obecně uplatnitelné principy šachové hry a nalezené principy později využít při implementaci funkce pro statické ohodnocování šachových pozic. Prokázalo se, že se faktory statické ohodnocovací funkce dělí na dva základní typy. Prvním typem jsou složky, které je možné použít téměř pro každou ze vzniklých situací na šachovnici. Možným příkladem je součet ohodnocení figur přítomných na šachovnici. Mezi složky druhého typu patří například ohodnocení obranného postavení krále, které je uplatnitelné pouze v první polovině partie.

Součástí diplomové práce je také analýza technik využitelných pro dosažení vyšší hloubky prohledávání. Práce se dále zabývá možnostmi implementace pro různé fáze šachových partií.

Praktická část popisuje vytvořené řešení. Statická ohodnocovací funkce byla rozšířena z původních 5 složek na 33 složek. Hloubka prohledávání byla navýšena z původních 3 půltahů na 5 až 11 půltahů v závislosti na konkrétní ohodnocované pozici. Pro počáteční a koncové fáze šachové partie byly vytvořeny databáze pozic, které doplňují funkcionalitu statické ohodnocovací funkce.

Správnost implementovaného řešení byla ověřena na vzorových pozicích z různých fází partie. Schopnosti šachového programu odpovídají původním představám. Jako hlavní úskalí vytvořeného řešení považuji nevhodně zvolenou reprezentaci šachové úlohy, která zpomaluje proces generování a ověřování tahů. Možná rozšíření programu jsem nastínil v kapitole o dosažených výsledcích.

Literatura

[1] Wikipedia, The Free Encyclopedia. *Pin (chess)*, [online] poslední úpravy 20. 9. 1999 [cit. 18. 4. 2018] Dostupné z: `https://en.wikipedia.org/wiki/Pin_(chess)`

[2] Chess.com. *The Evaluation of Material Imbalances (by IM Larry Kaufman)*, [online] poslední úpravy 17. 11. 2008 [cit. 18. 4. 2018] Dostupné z: `https://www.chess.com/article/view/the-evaluation-of-material-imbalances-by-im-larry-kaufman`

[3] Smirnov, I. *Chess Opening Fundamentals*, [e-kniha]. [cit. 12. 3. 2018] Dostupné z: `https://chess-teacher.com/wp-content/uploads/2016/04/Chess-opening-fundamentals-ebook.pdf`

[4] The Chess World. *7 Most Important Middlegame Principles*, [online] poslední úpravy 5. 18. 2015 [cit. 15. 3. 2018] Dostupné z: `https://thechessworld.com/articles/middle-game/7-most-important-middlegame-principles/`

[5] Silman, J. *Silman's Complete Endgame Course: From Beginner to Master*, Siles Press, Los Angeles, 2006, ISBN 9781890085100

[6] Apronus. *Chess Diagram Editor and Generator*, [online] [cit. 20. 3. 2018] Dostupné z: `https://www.apronus.com/chess/diagram/editor/`

[7] Wikipedia, The Free Encyclopedia. *Encyclopaedia of Chess Openings*, [online] poslední úpravy 12. 5. 2018 [cit. 13. 5. 2018] Dostupné z: `https://en.wikipedia.org/wiki/Encyclopaedia_of_Chess_Openings`

[8] Sehnal, J. *Implementace algoritmu šachové hry*, Plzeň, 2013, 45 s. Bakalářská práce na Fakultě aplikovaných věd Západočeské univerzity v Plzni na katedře informatiky a výpočetní techniky. Vedoucí diplomové práce Prof. Ing. Václav Matoušek, CSc.

[9] Levy, D. *Computer Chess Compendium*, Springer-Verlag, New York, 1988, ISBN 9780387913315

[10] Levy, D., NEWBORN, M. *How Computers Play Chess*, Ishi Press, New York, 2009, ISBN 9784871878012

[11] Wikipedia, The Free Encyclopedia. *Chess notation*, [online] poslední úpravy 10. 5. 2018 [cit. 13. 5. 2018] Dostupné z: `https://en.wikipedia.org/wiki/Chess_notation`

[12] Steinwender, D., Friedel, F. A. *Schach am PC*, Markt & Technik, Buch- und Software-Verlag GmbH, Haar bei München, 1995, ISBN 978-3877915226

[13] Frey, P. W. *Chess Skill in Man and Machine*, Springer, New York, 1983, s. 82-118. ISBN 978-0387908151

[14] *ICGA journal*, Č.25(3) (září 2002). Maastricht: IOS Press, 2002. Vychází jednou za tři měsíce. ISSN 1389-6911

[15] Nejstadt, J. *Test Your Tactical Ability*, B T Batsford Ltd, 1992, s. 110–135. ISBN 978-0713440133

[16] Wikipedia, The Free Encyclopedia. *Principal variation search*, [online] poslední úpravy 26. 6. 2017 [cit. 18. 4. 2018] Dostupné z: `https://en.wikipedia.org/wiki/Principal_variation_search`

[17] *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Č.11(11) (listopad 1989). IEEE Computer Society; Institute of Electrical and Electronics Engineers, 1989, s. 1203 - 1212. Vychází měsíčně. ISSN 0162-8828

[18] *Artificial Intelligence*, Č. 12(1) (květen 1979). Elsevier, 1979, s. 23 - 40. Vychází měsíčně. ISSN 0004-3702

[19] Sehnal, J., *Programově realizovatelné algoritmy šachové hry*, Plzeň, 2014, 9 s. Oborový projekt na Fakultě aplikovaných věd Západočeské univerzity v Plzni na katedře informatiky a výpočetní techniky. Vedoucí oborového projektu Prof. Ing. Václav Matoušek, CSc.

[20] Oracle. *Java SE*, [online] poslední úpravy 16. 4. 2018 [cit. 5. 5. 2018] Dostupné z: `http://www.oracle.com/technetwork/java/javase/overview/index.html`

[21] deviantART. *Chess Piece Brushes*, [online] [cit. 2. 5. 2018] Dostupné z: https://punkdoutkittn.deviantart.com/art/Chess-Piece-Brushes-90810700

[22] 365Chess.com. *ECO Codes*, [online] [cit. 22. 4. 2018] Dostupné z: https://www.365chess.com/eco.php

[23] Hackerearth. *Minimax Algorithm with Alpha-beta pruning* [online] poslední úpravy 31. 3. 2017 [cit. 24. 4. 2018] Dostupné z: https://www.hackerearth.com/blog/artificial-intelligence/minimax-algorithm-alpha-beta-pruning/

[24] Chess Programming Wiki. *Center Manhattan-Distance* [online] poslední úpravy 26. 11. 2016 [cit. 22. 4. 2018] Dostupné z: https://chessprogramming.wikispaces.com/Center%20Manhattan-Distance

[25] Chess Bin. *Transposition Table and Zobrist Hashing* [online] poslední úpravy 19. 2. 2010 [cit. 29. 4. 2018] Dostupné z: http://www.chessbin.com/post/Transposition-Table-and-Zobrist-Hashing

[26] Wikipedia, The Free Encyclopedia. *Zobrist Hashing* [online] poslední úpravy 19. 9. 2017 [cit. 15. 4. 2018] Dostupné z: https://en.wikipedia.org/wiki/Zobrist_hashing

[27] Frey, P. W. *Chess Skill in Man and Machine*, Springer, New York, 1983, ISBN 978-0387908151

[28] 365Chess.com. *Chess Opening Explorer*, [online] [cit. 10. 4. 2018] Dostupné z: https://www.365chess.com/opening.php

[29] NCZOnline. *How to install Apache Ant on Windows*, [online] poslední úpravy 12. 4. 2012 [cit. 4. 5. 2018] Dostupné z: https://www.nczonline.net/blog/2012/04/12/how-to-install-apache-ant-on-windows/

[30] Wikipedia, The Free Encyclopedia. *Pravidla šachů* [online] poslední úpravy 21. 4. 2018 [cit. 15. 4. 2018] Dostupné z: https://cs.wikipedia.org/wiki/Pravidla_šachů

A Slovník použitých pojmů

1, 2, 3, ..., 8 – názvy řad šachovnice
A, B, C, ..., H – názvy sloupců šachovnice
B – střelec (**B**ishop)
Braní mimochodem – speciální šachový tah, kdy je možné vlastním pěšcem sebrat soupeřova pěšce v případě, kdy v předchozím tahu soupeřův pěšec postoupil o dvě pole ze základní linie a tímto tahem přeskočil pole kontrolované vlastním pěšcem. Samotné sebrání soupeřova pěšce je provedeno, jako kdyby soupeřův pěšec táhl jen o jedno pole dopředu.
Dlouhá rošáda – rošáda provedená na dámském křídle
En passant – viz Braní mimochodem
Fianchetto – situace, kdy pěšec na druhém či sedmém sloupci táhne dopředu a uvolní tak výhodnou pozici pro vlastního střelce
GM – šachový velmistr (Grand Master)
IM – mezinárodní mistr (International Master)
Izolovaný pěšec – pěšec, okolo kterého se v ohruhu jednoho sloupce nenachází žádný další pěšec stejné barvy
K – král (**K**ing)
Kompenzace – šachový termín, hráč ztrácí materiál za účelem krátkodobé výhody (iniciativa, útok)
Krátká rošáda – rošáda provedená na královském křídle
Legální pozice – pozice odpovídající šachovým pravidlům
Legální tah – tah odpovídající šachovým pravidlům
Lehká figura – jezdec či střelec
Materiál – figury, šachové kameny na šachovnici
N – jezdec, kůň (K**n**ight)
Obět' – úmyslná ztráta figury za účelem zisku poziční či jiné výhody
P – pěšec (**P**awn)
Pěšcová většina – hráč má pěšcovou většinu na dané straně šachovnice, platí-li pro tuto stranu šachovnice, že součet hráčových pěšců je vyšší než součet pěšců soupeřových
Proměna – dosáhne-li pěšec poslední řady šachovnice, je povýšen na figuru dle hráčova výběru
Půltah – totéž jako tah, používáno k zdůraznění, že je míněn jeden konkrétní tah, nikoli tah včetně protihráčovy reakce
Q – dáma (**Q**ueen)
R – věž (**R**ook)
Řada – pole šachovnice, která mají stejný číselný index

Sloupec – pole šachovnice, která mají stejný písmenný index
Spojené volné pěšce – seskupení pěšců, které spolu sousedí a pro všechny platí, že jsou volné
Těžká figura – věž či dáma
Volný pěšec – pěšec, který má volnou cestu (na sloupci, na kterém se pěšec nachází, ani na sloupcích sousedních, nemá soupeř žádného pěšce, který by stál v cestě proměně)
Výměna – sekvence tahů vedoucí ke ztrátě figur na obou stranách
Vynucený tah – vynucený tah (popř. Zugzwang) představuje situaci, kdy je pro hráče každý z dostupných tahů nevýhodný
Vývin – proces přesunu figur ze základní linie do strategičtějších pozic
Zdvojené pěšce – dva pěšce stojící na stejném sloupci
Zdvojené věže – dvě věže stojící na stejném sloupci
Ztrojené pěšce – tři pěšce stojící na stejném sloupci

B Šachová notace

Pro popsání některých šachových příkladů a posloupností tahů jsem použil šachovou notaci. Jedná se o ustálený způsob zaznamenávání tahů. Jedním z nejrozšířenějších způsobů zápisu šachových tahů je algebraická šachová notace [11]. Pro zkrácenou algebraickou notaci jsou jednotlivé tahy popsány identifikátorem figury a cílovým polem, na které figura táhne. V případě plné algebraické notace je zápis doplněn o původní pole, ze kterého figura táhla. I když je zkrácená notace nejrozšířenější, pro některé případy jsem se v diplomové práci pro přehlednost rozhodl použít notaci plnou, ve stručnosti tedy znázorním oba přístupy.

Figury jsou značeny velkými písmeny. Značení figur vychází z anglických názvů a vypadá následovně: K pro krále, Q pro dámu, R pro věž, B pro střelce a N pro jezdce. Jedná-li se o tah pěšcem, je značení figury vynecháno. Pole šachovnice jsou pojmenována podle příslušných souřadnic šachovnice. Sloupce jsou značeny písmeny od A po H a řady jsou značeny čísly od 1 do 8. Pro přehlednost se pro značení polí v notaci používají malá písmena. Další používaná značení jsou znázorněna v tabulce B.1.

Značení	**Význam**
+	šach
# nebo ++	mat
×	sebrání figury
=	povýšení pěšce
O-O	krátká rošáda
O-O-O	dlouhá rošáda
1-0	výhra bílého
0-1	výhra černého

Tabulka B.1: Znaky šachové notace

S definovaným značením je již možné zapsat tahy šachové partie. Jednotlivé tahy jsou většinou doplněny číslem tahu. Ve většině případů jsou tahy číslovány od začátku partie, není to ale pravidlem. Zejména u šachových diagramů s ukázkami pozic jsou tahy pro přehlednost číslovány od daného výchozího bodu. Při zapisování posloupnosti tahů je možné volit mezi horizontálním zápisem, kde jsou tahy zapsány za sebou na řádce, a zápisem

vertikálním, kde každému tahu přísluší jedna řádka. V následující tabulce B.2 se v levém sloupci nachází ukázka zkrácené vertikální notace a v pravém sloupci se nachází ekvivalentní plná notace.

Zkrácená notace	**Plná notace**
1. e4 e5	1. e2-e4 e7-e5
2. Nf3 Nc6	2. Ng1-f3 Nb8-c6
3. Bb5 a6	3. Bf1-b5 a7-a6

Tabulka B.2: Ukázka šachové notace pro zahájení

C Uživatelský manuál

C.1 Systémové požadavky

Pro spuštění programu je zapotřebí mít instalovaný Java Runtime Environment JRE ve verzi 1.8 [20] a vyšší. Doporučované operační systémy jsou Windows 7 a Windows 10. Doba odezvy programu závisí na výkonnosti stroje, na kterém je program spuštěn, na složitosti zkoumané pozice a na aktuálním obsahu transpoziční tabulky, která je například v případech připravených pozic prázdná a program ji musí nejdříve naplnit.

C.2 Sestavení aplikace

Program je možné sestavit skriptem *build.xml*, který se nachází v adresáři *ChessGame*. K sestavení je zapotřebí mít instalovaný Apache Ant [29]. Následně stačí zkopírovat adresář *ChessGame* z přiloženého CD na disk, navigovat příkazovou řádku do tohoto adresáře a spustit sestavení programu příkazem *„ant“*.

Zdrojové kódy programu jsou umístěny ve složce *ChessGame/src/chessgame*. Ve složce *ChessGame/src* se dále nachází vytvořené soubory databáze pro koncové hry (podadresář *endgame*), soubor databáze zahájení (podadresář *opening*) a připravené testovací scénáře (podadresář *positions*).

C.3 Spuštění aplikace

JAR soubor se nachází v kořenovém adresáři přiloženého CD. Při použití sestavovacího skriptu se výsledný JAR soubor nachází v adresáři *dist*. Aplikace se spouští příkazem

`java -jar <JAR_path>`.

C.4 Hraní hry

Po spuštění programu je zobrazena hrací plocha s počáteční pozicí šachové partie. Uživatel začíná jako bílý. Výměnu stran je možné provést položkou v menu s názvem *Switch sides*. Uživatelův tah je proveden, shoduje-li se s šachovými pravidly. Pokusí-li se hráč provést nelegální tah, je tento tah ignorován. Stručná pravidla šachové hry jsou dostupná v příloze D.

V pravém panelu uživatelského rozhraní je zobrazen zjednodušený zápis partie, který je možné exportovat do textového souboru. Položka menu *Load position* slouží k nahrání připravených testovacích pozic.

D Stručná pravidla

Pravidla jsou převzata z [30].

D.1 Základní informace

Šachovou soupravu tvoří šachovnice a dvě sady kamenů – bílých a černých. Šachovnice je čtvercová deska o velikosti 8 x 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.

D.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.

D.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“.

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.

Dáma

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

Věž

Věž se pohybuje po řadách a sloupcích.

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.

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.

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.

D.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.

E Hashovací funkce Zobrist

```
/**
* Calculates the Zobrist hash key for given game.
*
* @param game Game information.
* @return Hash key
*/
public long getChessBoardKey(ChessEngine game) {
    long key = 0;
    short[][] position = game.getPosition();
    short[] castling = game.getCastlingPrivileges();
    short[] enPassant = game.getEnPassantPrivileges();
    boolean whiteOnTurn = game.getTurn();
    int fieldIndex;
    for(int i = ROWS_END; i >= ROWS_START; i--) {
        for(int j = FILES_START; j <= FILES_END; j++) {
            fieldIndex = (i - ROWS_START) * 8 + j -
               FILES_START;
            switch(position[i][j]) {
              case -1: key ^= BLACK_PAWN[fieldIndex];
                 break;
              case -2: key ^= BLACK_KNIGHT[fieldIndex];
                 break;
              case -3: key ^= BLACK_BISHOP[fieldIndex];
                 break;
              case -4: key ^= BLACK_ROOK[fieldIndex];
                 break;
              case -5: key ^= BLACK_QUEEN[fieldIndex];
                 break;
              case -6: key ^= BLACK_KING[fieldIndex];
                 break;
              case 1: key ^= WHITE_PAWN[fieldIndex];
                 break;
              case 2: key ^= WHITE_KNIGHT[fieldIndex];
                 break;
              case 3: key ^= WHITE_BISHOP[fieldIndex];
                 break;
              case 4: key ^= WHITE_ROOK[fieldIndex];
```

```
                    break;
                case 5: key ^= WHITE_QUEEN[fieldIndex];
                    break;
                case 6: key ^= WHITE_KING[fieldIndex];
                    break;
                default: break;
            }
        }
    }
    for(int i = 0; i < CASTLING_RIGHTS_COUNT; i++) {
        if(castling[i] == 1) {
            key ^= CASTLING_RIGHTS[i];
        }
    }
    for(int i = 0; i < EN_PASSANT_COUNT; i++) {
        if(enPassant[i] == 1) {
            key ^= EN_PASSANT_RIGHTS[i];
        }
    }
    if(whiteOnTurn) {
        key ^= WHITE_TO_MOVE;
    }
    return key;
}
```

F Hledání klidu pro výměny figur

```
/**
 * Takes care of evaluation of captures.
 * Generates available enemy piece captures and
    examines the sequence of captures for given square.
 *
 * @param engine Chess engine used to find moves.
 * @param alpha alpha margin
 * @param beta beta margin
 * @return Final exchange evaluation
 */
private int pieceExchangeQuiescence(ChessEngine engine,
    int alpha, int beta) {
    int eval = evaluator.evalFunction(engine);
    if(eval > alpha) {
       alpha = eval;
    }
    ArrayList<ChessMove> moves = getAvailableMoves(
       engine);
    ArrayList<ChessMove> blackListMoves = new
       ArrayList<ChessMove>(); //not to repeat same
       target of captures, SEE takes care of that
    for(ChessMove m : moves) {
        if(isCapture(m, engine) && !blackListMoves.
           contains(m)) { // only examine captures
           which haven't been examined earlier
            backUpIrreversible(engine);
            short targetFigure = engine.getPosition()[m
               .getToX()][ m.getToY()];
            short originalFigure = engine.getPosition()
               [m.getFromX()][ m.getFromY()];
            boolean legalFlag = false;
            if(engine.makeMove(m.getFromX(), m.getFromY
               (), m.getToX(), m.getToY())) { //check
               the legality of given move
                legalFlag = true;
                blackListMoves.addAll(getBlacklistMoves
```

```
                    (moves, m));
                engine.unmakeMove(m.getFromX(), m.
                    getFromY(), m.getToX(), m.getToY(),
                    targetFigure, originalFigure);
                restoreIrreversible(engine);
            } else {
                discardIrreversible();
            }
            if(legalFlag) {
                int newEval = -staticExchange(engine.
                    getAttackersAndDefenders(m.getToX(),
                     m.getToY()), engine.getTurn(),
                    targetFigure); //recursively
                    evaluate given capture
                newEval += eval;
                if(newEval > alpha) {
                    alpha = newEval;
                }
            }
        }
    }
    return alpha;
}
```

G Statické ohodnocování výměn

```
/**
 * Finds the best exchange evaluation from the
    perspective of player on turn.
 * The Player on turn can choose not to capture the
    enemy piece, resulting in total evaluation of zero.
 *
 * @param pieces List of pieces that attack given
    square.
 * @param whiteOnTurn side to play
 * @param originalPiece Chess piece that originally
    occupied the square that is being evaluated.
 * @return
 */
private int staticExchange(ArrayList<ChessPiece> pieces
   , boolean whiteOnTurn, short originalPiece) {
    int totalEval = 0;
    int originalValue = evaluator.evalPiece(
       originalPiece);
    if(whiteOnTurn) {
        originalValue = -originalValue; //negamax
           fashion
    }
    ChessPiece attacker = getLeastValuableAttacker(
       pieces, whiteOnTurn);
    if(attacker != null) {
        totalEval += originalValue;
        pieces.remove(attacker);
        totalEval += -staticExchange(pieces, !
           whiteOnTurn, attacker.getPiece());
        return Math.max(totalEval, 0);
    } else {
        return 0;
    }
}
```

H Ukázka použitých hodnot ohodnocovací funkce

```
private final int KNIGHT_EVAL_CONST = 300; //the knight
    evaluation constant
private final int BISHOP_EVAL_CONST = 300; //the bishop
    evaluation constant private final int
   PAWN_EVAL_CONST = 130; //the pawn evaluation
   constant
private final int ROOK_EVAL_CONST = 500; //the rook
   evaluation constant
private final int QUEEN_EVAL_CONST = 900; //the queen
   evaluation constant
private final int[] whitePawnAdvanceBonus =
   {0,0,0,0,10,15,20,40,100,0,0,0}; //the white pawn
   advancement bonus
private final int[] blackPawnAdvanceBonus =
   {0,0,0,-100,-40,-20,-15,-10,0,0,0,0}; //the black
   pawn advancement bonus
private final int UNDEVELOPED_LIGHT_PIECE_PENALTY =
   100; //undevelopment penalty
private final int PAWN_SIDE_EVAL_PENALTY = 15; //pawn
   on 1st and 8th rank
private final int PAWN_DOUBLED_PENALTY = 10;
private final int PAWN_ISOLATED_PENALTY = 10;
private final int PAWN_FREE_BONUS = 17; //passed pawn
private final int PAWN_CONNECTED_FREE_BONUS = 25; //
   connected passed pawns
private final int KING_EVAL_CONST = 3000;
private final int STARTING_ROOK_EVAL_PENALTY = 50; //
   decrease rook value in the opening
private final int ENDGAME_ROOK_EVAL_BONUS = 25; //
   increase rook value in the endgame
private final int QUEEN_VERSUS_ROOKS_BONUS = 50; //
   queen vs rooks with light pieces on the board
private final int EARLY_QUEEN_DEVELOPMENT_PENALTY =
   100;
private final int EXTRA_ROOK_EVAL_PENALTY = 25; //
   redundancy principle
```

```
private final int ROOK_SEVENTH_RANK_BONUS = 30;
private final int KNIGHT_EVAL_LESS_PAWNS_PENALTY = 20;
   //decrease knight value with less pawns on board
private final int ROOK_EVAL_LESS_PAWNS_BONUS = 25; //
   increase rook value with less pawns on board
private final int ROOK_SEMI_FREE_FILE_BONUS = 15; //
   semi open file for rooks
private final int ROOK_FREE_FILE_BONUS = 30;
private final int NO_CASTLING_RIGHTS_PENALTY = 30;
private final int THREE_LIGHT_PEICES_VERSUS_QUEEN_BONUS
    = 40;
private final int BAD_BISHOP_PENALTY = 30;
private final int KING_EVAL_POSITION_CONST = 200; //the
    king position penalty constant
private final int BISHOP_PAIR_BONUS = 50;
private final int KING_DEFENSIVE_POSITION = 50; //the
   bonus for king's defensive position
private final int PAWN_SHIELD = 40;
private final int INACTIVE_BISHOP_PENALTY = 30;
private final int FORK_BONUS = 100; //only counts if
   both forked pieces are of higher value than the
   piece that forked them
private final int[] FIGURES_ENDANGERED_TABLE = {0, 2,
   4, 4, 6, 8, 10}; //endangerment table, from pawn (2)
    to king(10)
private final int[] FIGURES_DEFENDED_TABLE = {0, 3, 3,
   3, 2, 1, 0}; //defense table, from pawn (3) to king
   (0)
```

I UML diagram aplikace