Umělá inteligence pro České šachy

Vojtěch Šrámek

2023/2024

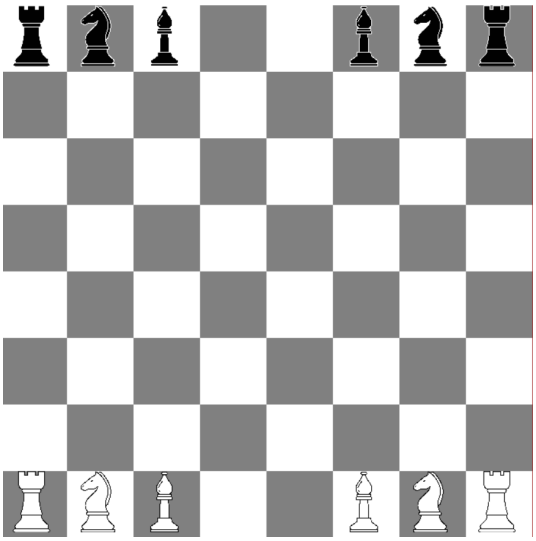
Úvod

Jelikož jsem nadšený šachista, vždy jsem si chtěl vyzkoušet vytvořit šachový motor, proto jsem nad volbou projektu příliš neváhal. Jelikož ovšem pro klasický šach existuje již velká spousta kvalitních motorů, rozhodl jsem se vytvořit AI pro málo známou variantu šachu, pro kterou zatím neexistuje motor (alespoň mě není takový znám) - České šachy.

1 České šachy

České šachy (také nazývané folky) jsou varianta šachu, která byla poprvé představena veřejnosti v roce 2019 na šachovém festivalu Czech Open v Pardubicích (kde jsem se s nimi také seznámil). Jejich autorem je Petr Doubek z Jaroměře.[1]

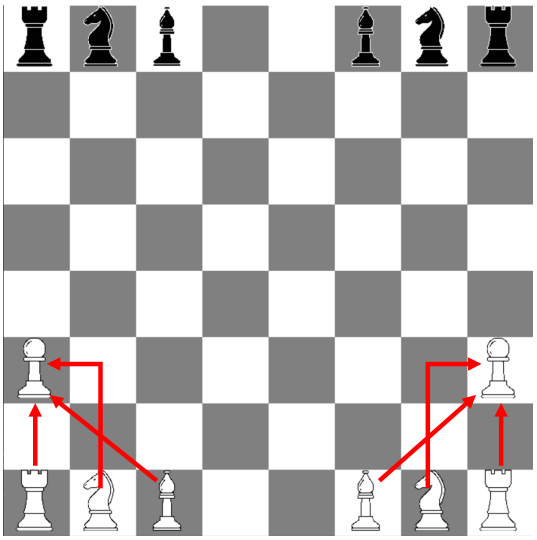
V Českém šachu jsou ve hře přítomny pouze následující figurky - věže, jezdci, střelci a pěšci (dáma a král se v této hře nevyskytují). Počáteční rozestavení věží, jezdců a pěšců je stejné jako v klasickém šachu a je znázorněno na obrázku 1. S těmito figurami je možné pohybovat tak jako v



Obrázek 1: Základní pozice Českých šachů.

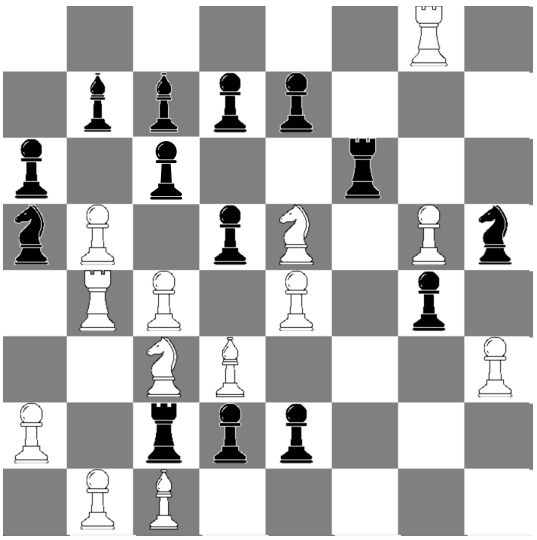
klasickém šachu. Nepřátelské figury se ovšem oproti klasickému šachu nemohou navzájem sebrat.[1]

Pěšce je možné postavit na takové políčko, na které naše figury útočí třikrát. Taková situace je zobrazena na obrázku 2. Pěšci nemohou provést žádný tah, nemohou být ani sebráni. Cílem hry je



Obrázek 2: Ukázka možného umístění pěšců v základní pozici. Bílý může umístit na šachovnici pěšce na jedno ze dvou polí (h3 nebo a3), jelikož na tyto pole působí tři jeho figury.

postavit na šachovnici osm pěšců.[1] Zakončení hry je zobrazeno na obrázku 3. Partie může skončit také remízou. Remíza nastane pokud je libovolná pozice, při které je na tahu stejná strana, třikrát zopakována, případně obdobně jako v klasickém šachu by partie mohla skončit remízou pokud se na ni hráči domluví.

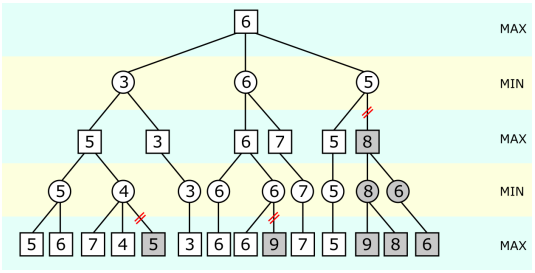


Obrázek 3: Závěr jedné z partií. Černý zvítězil, jelikož jako první postavil osm pěšců.

2 Popis použitých metod

Kostru řešení tvoří algoritmus minimax. Minimax pracuje následujícím způsobem. Pokud je uzel listový (bylo dosaženo maximální hloubky nebo koncového stavu) vypočítej hodnotu ohodnocovací funkce dané pozice a vrať ji jako výstpuní hodnotu. Pokud uzel není listový vrací se postupně maximální (bílý) nebo minimální (černý) hodnota dané vrstvy. Algoritmus funguje jako prohledávání do hloubky.[2]

Dále bylo aplikováno jedno z možných vylepšení minimaxu tzv. alfa-beta prořezávání. Při této proceduře jsou ukládány dvě proměnné navíc - α, což je dolní mez ohodnocení uzlu v němž probíhá maximalizace a β, což je horní mez ohodnocení uzlu v němž probíhá minimalizace. Tyto dvě proměnné jsou inicializovány na $-\infty$ (α) a ∞ (β). Tyto proměnné jsou následně aktualizovány maximem (α) nebo minimem (β) z jejich současné hodnoty a hodnoty ohodnoceného uzlu. Pokud v nějakém uzlu nastane situace, že $\alpha >= \beta$, již se neprohledávají další možné uzly ve vrstvě.[2] Minimax algoritmus s alfa-beta prořezáváním je znázorněn na obrázku 4.



Obrázek 4: Ukázka minimax algoritmu s alfa-beta prořezáváním. Převzato z [3]

Pro další urychlení byla využita tzv. transpoziční tabulka. V transpoziční tabulce jsou uloženy předchozí již ohodnocené pozice (takže je není potřeba znovu prohledávat). Pozice jsou uloženy pomocí hashů.[5]

Využil jsem tzv. Zobrist hashování. Zobrist hashování nejprve přiřadí figurám a všem jejím pozicím náhodné čísla. Hash dané pozice se vypočte pomocí operace XOR, která je provedena mezi všemi takto náhodně vygenerovanými čísly, na kterých je přítomna nějaká figura. Po provedení tahu je možné hash jednoduše aktualizovat provedením operace XOR s náhodným číslem dané figury, která táhla na dané pole. Díky toho je Zobrist hashování pro šachy velice efektivní.[4]

Alfa-beta prořezávání je možné zlepšit vhodným pořadím tahů, které jsou prohledávány jako první. Vhodné pořadí tahů může zefektivnit alfa-beta prořezávání až 10x oproti náhodné volbě tahů.[5] Experimentoval jsem s různými možnostmi výběru pořadí tahů (např. aby prvně byly prohledávány tahy, které již jsou v transpoziční tabulce), nejlépe mně ovšem fungovalo původní pořadí prohledávaných tahů, sestavené pouze jednoduchou logickou úvahou. Nejprve se prohledají možné umístění pěšců (velká šance, že nebudou špatné), pak tahy věží (mají velkou pohyblivost), následně střelců a nakonec jezdců (nejmenší pohyblivost, větší šance, že stojí dobře po pár tazích). V klasickém šachu se často nejprve prohledávají tahy, které dávají šach nebo berou nějakou figuru, což v případě Českých šachů není možné. Existuje však i mnoho jiných heuristik pro výběr správného pořadí tahů, které by bylo možné využít např. „zabijácká"heuristika (upřednostňuje tahy, které způsobily alfa-beta ořezání) apod.[6] To by mohlo být jedno z budoucích vylepšení použitého postupu.

Jedním ze základních stavebních kamenů klasického šachového motoru je využití vhodné heuristiky pro hodnocení pozice. V klasickém šachu se používá celá řada heuristik. Některé, například

jednu ze základních heuristik - materiál [7], však není možné v případě Českých šachů použít.

Pro výhru v Českém šachu je potřeba stavět pěšce. Z tohoto důvodu byl jako jedna z heuristik zvolen počet postavených pěšců. Dalším důležitým aspektem je pohyblivost, a to zejména proto, že se figury nemohou brát, ani přeskakovat (kromě jezdců) a tak můžeme umístěním pěšce své figury velmi znehodnotit. Například bílý si může v základní pozici na obrázku 1 postavit jednoho ze dvou pěšců, významně si tím však omezí pohyblivost své věže, která aby se dostala opět dohry, musí táhnout alespoň dvakrát. Při větším množství pěšců se může dokonce stát, že se nějaká figura nebude moci ani pohnout. Z těchto důvodů byla pohyblivost figur zvolena jako druhá heuristika.

3 Popis implementace a jednotlivých souborů

Šachový motor byl implementován v jazyce python. Využil jsem tzv. bitboards k reprezentaci šachovnice, kde je pozice všech figur uložena pomocí bitových map, díky čehož je možné využívat bitové operace (zejména shift a XOR) pro provádění a generování tahů. Tento způsob by měl být rychlejší než prostá implementace pomocí souřadnic. Byla snaha bitové operace ještě více zrychlit užitím různých knihoven (numpy, bitarray a gmpy2), nicméně implementace s python integery byla rychlejší. Rovněž byla snaha zrychlit prohledávání pomocí knihovny numba, nicméně ani tento pokus nebyl příliš úspěšný.

Vytvořil jsem i jednoduché GUI pomocí knihovny pygame, takže je možné si s motorem pohodlně zahrát. Stručný popis přiložených souborů:

a) **Board.py** - Skript, v němž je implementováno AI (minimax, evaluační funkce apod.) vše ve třídě Board. Skript je využit pro generování tahů ve skriptu Settings.py, je však možné ho samostatně spustit a zjistit, jak dlouho trvá prohledávání pozice při zadané hloubce.

b) **chessboard.py** - Skript, v němž jsou implementovány třídy jednotlivých figur. Skript je využit v Board.py pro generování přípustných tahů v dané pozici.

c) **chessboard_pygame.py** - Skript, v němž jsou implementovány třídy jednotlivých figur a šachovnice pro pygame. Původně jsem měl i šachové AI v souřadnicové reprezentaci, to jsem ovšem pak v zájmu zrychlení generování tahů upravil. Pro samotné GUI je však souřadnicová reprezentace daleko příjemnější, proto jsem ji ponechal. Při hraní s AI existují tedy dvě šachovnice - jedna pro GUI a druhá pro AI.

d) **game_fucntions.py** - Skript, v němž jsou implementovány funkce podstatné pro pygame GUI. Dochází zde také k přepočítávání vygenerovaného tahu AI pro GUI reprezentaci a naopak.

e) **Czech_chess.py** - Hlavní spustitelný skript, který spustí GUI a v němž je možné hrát proti motoru.

d) **Settings.py** - Skript, který slouží k základnímu nastavení pygame GUI (velikost šachovnice apod.), vše ve třídě Settings.

Závěr

V tomto projektu jsem vytvořil jednoduchý motor pro České šachy. Motor je založený na Minimax algoritmu a alfa-beta prořezávání. Sílu motoru je možné otestovat pomocí GUI. Přestože je motor stále poměrně neefektivní (pro rychlou odpověď jsem nastavil hloubku pouze tři půltahy), hraje již poměrně dobře (alespoň tedy mě poráží pokud nad hrou příliš nepřemýšlím, nejsem nicméně příliš trénovaný v Českých šachách). Počet půltahů, při kterém dojde k prohledání v ještě relativně rozumném čase je (alespoň na mém počítači) zhruba 6 půltahů, při současném nastavení.

Možná budoucí zlepšení motoru by kromě již zmiňovaného lepšího pořadí prohledávání tahů mohlo být např. iterativní prohledávání (bylo by možné omezit čas), prohledávání na soupeřův čas, Quiescence ořezávání atd. Jak jsem zjistil optimalizace šachového AI je prací téměř nekonečnou... Závěrem bych chtěl poděkovat za výuku předmětu.

Reference

[1] DOUBEK, Petr. *Folky - O hře*. Online. FOLKY - šachy jinak. Dostupné z: `https://www.folky.cz/czech-chess`. [cit. 2024-05-07].

[2] DVOŘÁK, Jiří. *Metody hraní her, Přednáška*. [cit. 2024-05-07].

[3] JEZ9999. *alfa-Beta pruning example*. Online. Dostupné z: `https://commons.wikimedia.org/wiki/File:AB_pruning.svg`. [cit. 2024-05-07].

[4] *Zobrist Hashing*. Online. Dostupné z: `https://www.chessprogramming.org/Zobrist_Hashing`. [cit. 2024-05-07].

[5] MILLINGTON, Ian. *Artificial intelligence for games*. Online. Elsevier, 2006. ISBN 978-0-12-497782-2. [cit. 2024-05-07].

[6] *Killer Heuristic*. Online. ChessProgramming Wiki. Dostupné z: `https://www.chessprogramming.org/Killer_Heuristic`. [cit. 2024-05-07].

[7] *Evaluation*. Online. ChessProgramming Wiki. Dostupné z: `https://www.chessprogramming.org/Evaluation`. [cit. 2024-05-07].