Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Šachy a umělá inteligence

Karel Najman

vedoucí: Ing. Jan Pavlík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Karel Najman

Systémové inženýrství a informatika
Informatika

Název práce

Šachy a umělá inteligence

Název anglicky

Chess and artificial intelligence

Cíle práce

Tato bakalářská práce je zaměřena na tvorbu šachových enginů a přesněji na hardwarovou náročnost různých algoritmů a metod, na kterých tyto enginy fungují. Hlavní cílem této práce je porovnat efektivitu jednotlivých postupů při tvorbě šachové AI. Dílčí cíle práce jsou:
-naprogramování několika šachových enginů fungujících na různých algoritmech
-porovnání jejich výkonostní náročnosti a přesnosti jejich hry
-nalezení nejefektivnějších metod pro vylepšení kvality šachového enginu

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Praktickou částí práce je vytvoření několika šachových enginů fungujících na známých algoritmech. Jednotlivé enginy budou porovnány na základě jejich výpočetní efektivity, herních výsledků a hardwarové náročnosti. Z výsledku tohoto zkoumání budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

35-50

Klíčová slova

šachy, umělá inteligence, šachové enginy

Doporučené zdroje informací

ATKINSON, George: Chess and Machine Intuition, Intellect, 1998, ISBN: 1871516447
CHERNEV, Irving: Logical Chess : Move By Move: Every Move Explained, Batsford Ltd, 1998, ISBN: 0713484640
KISLIK, Erik: Applying Logic in Chess, Gambit Publications Ltd, 2018, ISBN: 1911465244
LEVY, David; NEWBORN, Monty: How Computers Play Chess, Computer Science Press, 1990, ISBN: 4871878015
PACHMAN, Luděk: Modern Chess Strategy, Dover Publications, 1971, ISBN: 0486202909

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Jan Pavlík

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 29. 7. 2020

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 12. 03. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Šachy a umělá inteligence" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 12. 3. 2021 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Janu Pavlíkovi za vstřícnost a rychlou odezvu během doby, která znemožnila osobní setkání.

Šachy a umělá inteligence

Abstrakt

Tato práce se zaměřuje na zhodnocení několika základních funkcí používaných v šachových enginech na základě jejich hardwarové náročnosti a jejich vlivu na herní sílu šachového AI.

Náplní praktické části této bakalářské práce je vytvoření jednoduchého šachového enginu, následná implementace a úprava jednotlivých funkcí a změření jejich dopadu na šachový program. Východiskem těchto zkoumání jsou poznatky relevantní pro optimalizaci šachových aplikací.

Klíčová slova: šachy, umělá inteligence, šachové enginy

Chess and artificial intelligence

Abstract

This work focuses on evaluation of few basic functions used by chess engines based on their hardware requirements and their impact on the strength of the chess AI.

Content of the practical part of this bachelor thesis is creating a simple chess engine, subsequent implementation and modification of individual functions and measuring their impact on the chess program. Results of these tests are findings relevant to the optimization of chess applications.

Keywords: chess, artificial intelligence, chess engines

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Šachy jsou hrou s dlouhou historií a jejich role často nebyla rázu pouze zábavního, ale mnohdy se tato hra nacházela v popředí dějin. Počínaje legendami o vzniku šachů jako je známá matematická úloha o indickém mudrci jménem Sissa ibn Dahir, který pro krále tuto hru vytvořil. Přes rivalitu mezi Sovětským svazem a USA v 70. letech, která vyvrcholila „zápasem století“ mezi Bobby Fischerem a Borisem Spasským. A končíce, pro tuto práci nejrelevantnější událostí, zápasem mezi Garry Kasparovem a šachovým počítačem Deep Blue, který skončil výhrou Deep Blue, což značilo začátek nové éry, kdy se AI (Artificial intelligence) začalo stávat nezbytné pro fungování naší společnosti. Šachy byly a stále jsou nedílnou součástí vývoje naší civilizace.

Dnešní šachovou scénu je nemožné si představit bez využívání moderních technologií. Může se například jednat o online partie, které se staly nejčastějším způsobem, jakým lidé spolu šachy hrají. Nicméně technologie, která však nyní definuje šachy nejvíce, jsou šachové enginy. Od dob Deep Blue jsme ušli velký kus cesty a dnes je počítačová analýza partií každodenní záležitostí pro šachové mistry a jiné šachové experty. Krom využívání šachových AI na nejvyšší úrovni se však průměrný šachový nadšenec bude setkávat s méně komplexními programy ve svých tabletech a mobilech. Existuje řada postupů a metod, jak naprogramovat relativně silné a rychlé šachové AI. Bylo by proto vhodné, kdyby zde bylo objektivní zhodnocení jednotlivých metod, které by mohlo poté sloužit k vytvoření efektnějšího, ať už časově, či paměťově, šachového enginu.

2 Cíle práce a metodika

2.1 Cíle práce

Tato bakalářská práce je zaměřena na tvorbu šachových enginů a přesněji na hardwarovou náročnost různých algoritmů a metod, na kterých tyto enginy fungují. Hlavní cílem této práce je porovnat efektivitu jednotlivých postupů při tvorbě šachové AI. Dílčí cíle práce jsou:

- naprogramování několika šachových enginů fungujících na různých algoritmech
- porovnání jejich výkonnostní náročnosti a přesnosti jejich hry
- nalezení nejefektivnějších metod pro vylepšení kvality šachového enginu

2.2 Metodika práce

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Praktickou částí práce je vytvoření několika šachových enginů fungujících na známých algoritmech. Jednotlivé enginy budou porovnány na základě jejich výpočetní efektivity, herních výsledků a hardwarové náročnosti. Z výsledku tohoto zkoumání budou formulovány závěry bakalářské práce.

3 Literární rešerše

3.1 Šachy

Původ šachů je nejasný, nejpopulárnější teorií je, že první verze této hry vznikla v Indii kolem roku 600 a byla známá jako *chaturanga*, odsud se pak rozšířila do Evropy a Asie [1]. Existují však i alternativní teorie o původu z Číny, či Persie. Kolem 10. století se hra dostala do Evropy, avšak nejednalo se ještě o šachy, tak jak je známe dnes [2]. Hra nejprve musela projít několika změnami jako je například schopnost pěšáka jít o dvě pole dopředu na prvním tahu. Poslední úpravy pravidel byly provedeny v 15. století, jednalo se o rošádu a braní mimochodem. Než se tato pravidla standardizovala po celé Evropě, tak to trvalo dalších 300 let [2].

Šachy 18. století se už se dají považovat za definitivní verzi hry, která je hrána dodnes. Začala vznikat komplexnější teorie šachu, která se do této doby rozvíjet nemohla vzhledem k neustáleným pravidlům na evropském kontinentě. Roku 1749 vyšla kniha *Analyse du jeu des Échecs* od francouzského autora Francois – Andre Philidora, ve které byla také probrána teorie zahájení, odsud pak pochází Philidorova obrana, zahájení používané dodnes.[1] Roku 1886 byl udělen první titul mistra světa rakouskému hráči, narozenému v Praze, Wilhelmu Steitnitzovi. Po něm následovala celá řada, dnes už legendárních, šachistů, kteří se zasloužili o popularizaci šachu a jeho vývoj. Šlo o hráče jako například Michail Botvinnik, Michail Tal, Bobby Fischer, Garri Kasparov.

Dalším zlomovým rokem byl rok 1997, kdy šachový superpočítač Deep Blue porazil Kasparova, tehdejšího šachového mistra světa.[3] To značilo začátek nové éry, během několika krátkých let se šachové AI vyvinulo natolik, že už nebylo pochyb o nadřazenosti počítačové výpočetní síly nad lidským intelektem.

3.2 ELO

Před tím, než proberu zástupce šachových enginů a metody tvorby a algoritmy šachového AI, tak je nutné vysvětlit číslo Elo. Jde o hodnotu, která definuje míru relativní síly hráče. To znamená, s jakou pravděpodobností porazí jeden hráč druhého. Tato skutečnost je znázorněna v následující tabulce 1.

D	%
+150	29.66
+100	35.99
+50	42.85
0	50
-50	57.15
-100	64.01
-150	70.34

Tabulka 1 – Pravděpodobnost výhry/prohry na základě Ela [4]

D označuje rozdíl mezi hodnotami Elo daných hráčů. Procenta pak znázorňují šanci na vítězství, nebo remízu. Pokud by hráč měl Elo 1400 a hrál by proti soupeři s hodnotou 1300, tak jeho šance na vítězství je 64,01 %.

Podle Ela jsou pak hráči zařazováni do kategorií. Způsob zařazování do kategorií se liší podle toho, která šachová organizace danou soutěž spravuje, každopádně mezinárodní šachová organizace FIDE zařazuje hráče podle tabulky 2.

Elo	Kategorie
2500+	Šachový velmistr (GM)
2400-2500	Mezinárodní mistr (LM)
2300-2400	FIDE mistr (FM)
2200-2300	FIDE kandidát mistr (CM)
2000-2200	Kandidát mistr
1800-2000	Třída A
1600-1800	Třída B
1400-1600	Třída C

1200-1400	Třída D
Pod 1200	začátečník

Tabulka 2 - Šachové úrovně podle FIDE [5]

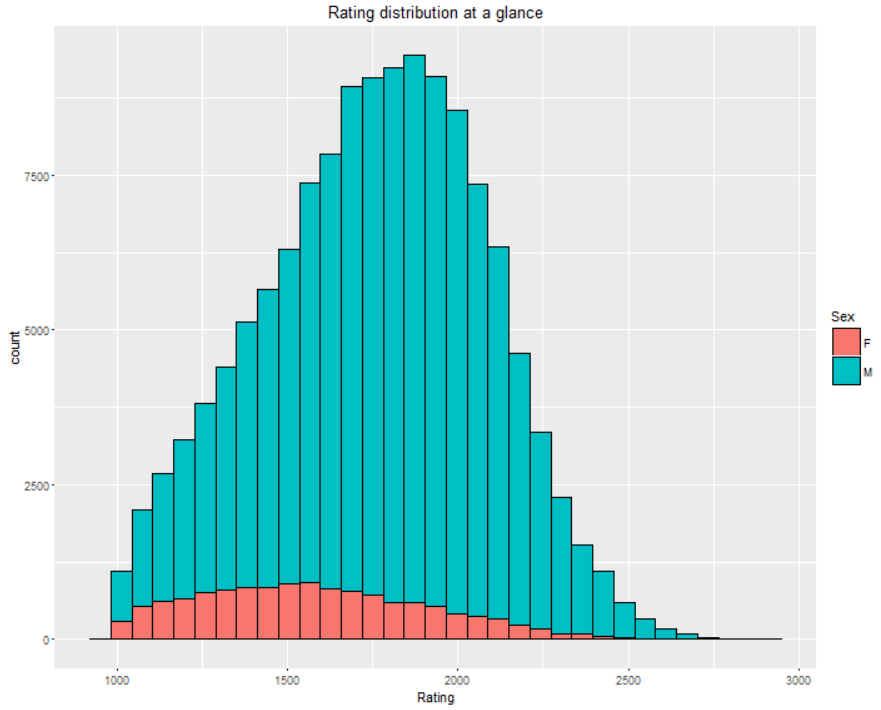
3.2.1 Výpočet

Co se samotného výpočtu Ela týče, tak zde opět záleží na šachové organizaci, která tento výpočet provádí. Zatímco například Fabiano Caruana má ke dni 19. 10. 2020 Elo 2828 na FIDE řebříčku [6], tak ke stejnému dni má na řebříčku USCF (United States Chess Federation) Elo 2908.[7] To je samozřejmě dáno výpočty podle různých rovnic. Původní rovnice, jejichž tvůrcem byl Arpad Elo, měly tento tvar:

$$Ea = \frac{1}{1+10^{\frac{Rb-Ra}{400}}} \qquad Eb = \frac{1}{1+10^{\frac{Rb-Ra}{400}}}$$

Kde Ea je hráč jedna a Ra je hodnota jeho Ela, Eb je pak hráč dvě a Rb je jeho Elo. Výsledkem je pak šance na vítězství daného hráče. Pokud hráč nedosáhne očekávaného skóre, tak systém vyvodí, že jeho hodnocení musí být nižší. Když hráč dosáhne předčí očekávané skóre, tak jeho hodnocení musí být vyšší. Maximální hodnota, o kterou se Elo může změnit, se nazývá K-faktor. Tato hodnota se odvozuje z hodnoty Ela hráčů.[8] Dnešní výpočty Ela jsou však složitější než tato původní verze, což je dáno tím, mimo jiné, že původní výpočty byly prováděny na papíře.

Nezávisle na systému, který k výpočtu Ela je použit, výsledná křivka hodnot hráčů by měla být křivkou normálního rozdělení.



Obrázek 1 - FIDE distribuce Ela [9]

3.2.2 Elo a šachové AI

Síla šachových enginů je také vyjadřována hodnotou Elo. Hlavním a nepřesnějším způsobem určování Ela jednotlivých enginů je skrze turnaje pro šachové AI. Nejznámějším z nich je *Top Chess Engine Championship*. Jde o neustále probíhající turnaj, kde po dobu jedné sezóny, což jsou přibližně 3-4 měsíce, spolu soupeří pouze šachové programy. Vítězem poslední sezóny byl Stockfish. [10] Co se týče Ela jednotlivých šachových enginů, tak opět záleží na způsobu výpočtu, navíc se ale také musí brát v potaz hardware. Podle CCRL (Computer Chess Rating Lists) jsou nejsilnější enginy ke dni 20. 20. 2020 následující:

Jméno	Elo
Stockfish	3522
Komodo	3421
Houdini	3401
Ethereal	3368
SlowChess	3343

Tabulka 3 - Nejsilnější enginy podle CCRL [11]

Elo těchto počítačových programů je ovlivňováno, krom použitého hardwaru, faktory jako časová kontrola, zda běží na 32-bit či 64-bit operačním systému, nebo třeba využité nastavení pro zahájení.

3.3 Šachové enginy

„No computer will ever beat me.“ (Kasparov, 1994)

Pouhých pár let po tomto prohlášení prohrál Garry Kasparov první zápas proti Deep Blue. Jak je vidět, tak vývoj šachových programů byl rapidní a během několika krátkých let se enginy staly nespornými králi šachu. Dnes už se i ti nejlepší hráči spoléhají na počítačovou analýzu a šachové enginy hrají důležitou roli v každé Elo kategorii. V následujících podkapitolách je popsáno několik významných zástupců.

3.3.1 **Stockfish**

Stockfish je open source software, jehož první verze byla vydána v roce 2008. Dnes jde o asi nejznámější šachové AI, což je dáno poměrně dlouhotrvající dominancí v AI turnajích. *Top Chess Engine Championship* měl 19 sezón, z toho 9 z nich vyhrál Stockfish. V posledních letech se ale začaly objevovat enginy, které hrozí sesazením dosavadního šampiona z šachového trůnu. [12]

3.3.2 **Leela Chess Zero**

Jedním z vyzyvatelů je Leela, která se proslavila nejen kvalitou její hry, vyhrála 2 sezóny *Top Chess Engine Championship*, ale také její architekturou. Na rozdíl od tradičních šachových enginů, jejichž zástupcem je například Stockfish, se jedná o software pracující pomocí neuronové sítě. To znamená, že Leela je schopná se sama učit a adaptovat z odehraných her. [13] Toto je koncept, který v šachu poprvé předvedl engine Alpha Zero. Vývojáři Leela Chess Zero pak vycházeli z poznatků nabytých od Alpha Zero. Leela je v šachové komunitě také populární kvůli její relativní ochotě obětovat materiál, čímž se liší od většiny ostatních šachových enginů. [14]

3.3.3 **Rybka**

Rybka je další engine, jenž stojí za zmínku. Jeho tvůrcem byl Vasik Rajlich, který je československého původu. Rybka byla mezi lety 2007–2010 jedním z nejsilnějších enginů,

nicméně roku 2011 byla diskvalifikována z dalších šachových soutěží kvůli nařčení z plagiátorství kódu ostatních šachových softwarů. [15]

3.4 Algoritmizace šachu

Už začátkem druhé poloviny 20. století existovala snaha vytvořit šachové AI. Tento poměrně brzký zájem byl dán, krom populárnosti šachu, relativní jednoduchostí pravidel, absencí neobvyklých herních mechanik, například jako je vracení figurek v shogi, a jednoduše definovatelný cíl, mat, nemožnost předejít ztrátě krále na dalším tahu. Co naopak programování šachu ztěžuje je množství možných tahů. Konečné číslo je předmětem mnoha odhadů, jedním z nichž je Shannonovo číslo, které má hodnotu 10^{123}. Toto je hodnota pouze jen přibližná a její výpočet vychází z průměrných hodnot jako je 30 možných tahů v každé pozici a délky hry o 80 tazích.[16] Každopádně je samozřejmé, že 10^{123} tahů, nebo tomu podobné množství, je nemožné i pro dnešní počítače prozkoumat. Z toho tedy vyplývá, že jsou za potřebí algoritmy, které jsou schopné vyhodnocovat, které tahy se mají dál propočítávat, a které tahy z výpočtů eliminovat. Tyto algoritmy jsou pak kostrou každého šachového AI.

3.4.1 Vyhodnocení pozice

V první řadě je potřeba, aby šachové AI bylo schopné jednotlivé pozice vyhodnotit. Naprostý základ každé hry je materiál, figurky, proto je nutné, aby šachový engine měl k jednotlivým šachovým figurkám přiřazeny hodnoty. Tyto hodnoty vycházejí z šachové teorie. Stejně tak, jak se vyvíjela teorie šachu, tak se vyvíjelo i hodnocení relativní síly jednotlivých figurek. Tradiční ohodnocení vypadalo takto:

Figurka	Hodnota
pěšák	1
kůň	3
střelec	3
věž	5
dáma	9

Tabulka 4 - Hodnoty figur [16]

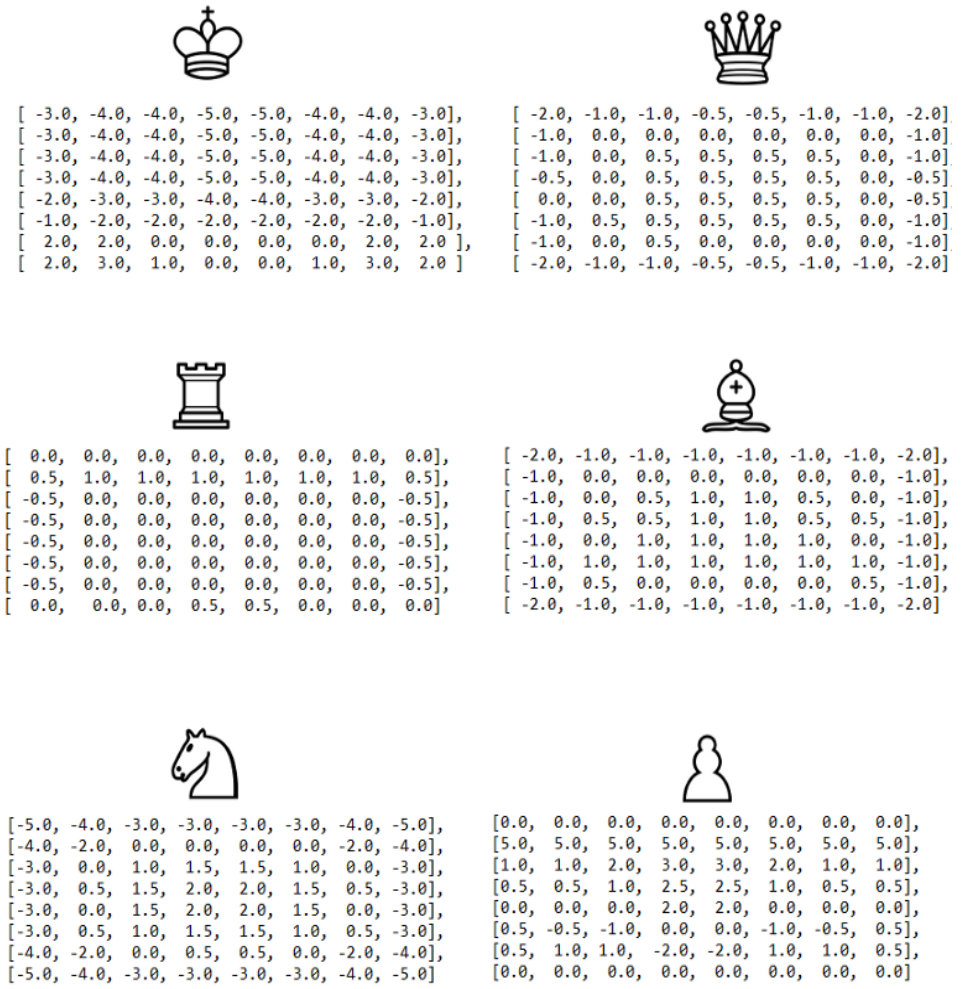
Nicméně v průběhu let se tyto hodnoty vyvíjeli a dnes už se například střelec považuje za figuru mírně silnější, než je kůň. Bobby Fischer přiřazoval střelci hodnotu 3,25, zatímco koni přiřadil hodnotu 3. [17] Dnešní hodnocení, které pracuje s AI analýzou, vypadá takto:

Figurka	Hodnota
pěšák	1,0
kůň	3,2
střelec	3,3
věž	5,1
dáma	8,8

Tabulka 5 - Dnešní hodnocení figur [16]

Důležité je si ale uvědomit, že šachy jsou komplexnější hrou, než aby se pozice daly jednoznačně vyhodnocovat pouze na základě materiálu. Umístění jednotlivých figur hraje roli, stejně tak kombinace figur je relevantním faktorem, například zda má hráč dva střelce, či jednoho střelce a jednoho koně, tudíž abychom dosáhli přesnějšího zhodnocení, tak tyto faktory musíme brát v potaz.

Jak už jsem tedy naznačil, hodnota jednotlivých figur se také odvíjí od jejich umístění, přeci jen, pokud je kůň v rohu šachovnice, tak jeho dopad na hru je nižší, než kdyby byl na aktivnější pozici jako je střed šachovnice. Stejně tak, pěšák, který je v 7. řadě, tedy pouze jedno pole od výměny za královnu, má už také výrazně vyšší hodnotu. O kolik se změní hodnota figur na základě jejich postavení pak vychází opět z šachové teorie. Tyto skutečnosti jsou znázorněny v následujícím obrázku. Každá pozice v matici znázorňuje políčko na šachovnici. Hodnota pozice je poté hodnotou, kterou přičteme k základním hodnotám figur, které jsou v tabulce 5.



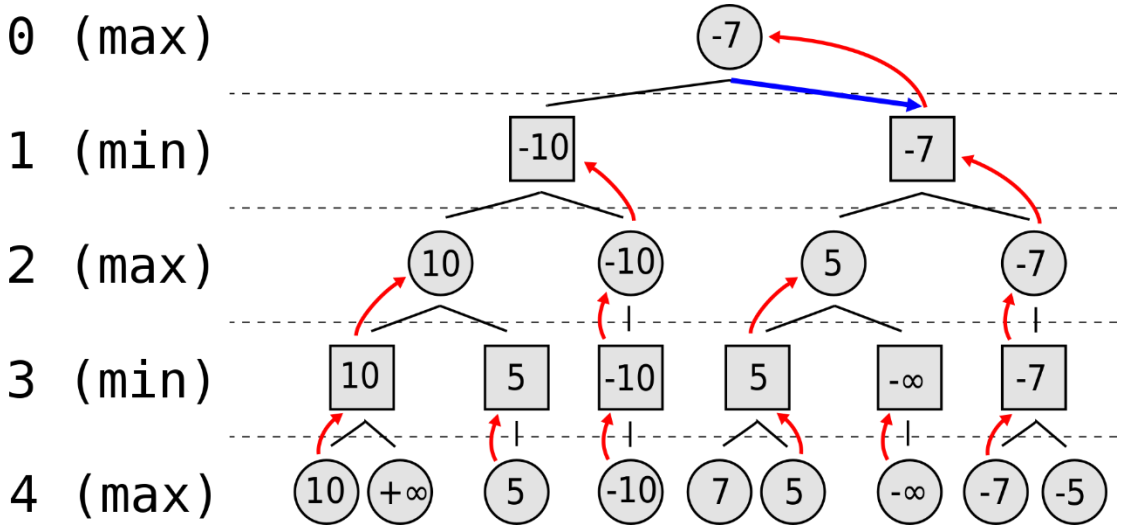
Obrázek 2 - Znázorněné změny hodnot [19]

Samozřejmě říct, že tento způsob hodnocení je dokonalý, by bylo naivní. Ne každá partie vypadá stejně a existují situace, kdy kůň v rohu šachovnice může být silnější figurou, než by byl na středu šachovnice. Zároveň existuje celá řada dalších faktorů, které tímto způsobem hodnocení nejsou zachyceny, struktura pěšáků, zda jsou věže propojené, atd. Nicméně zachytit všechny tyto aspekty spolehlivě je obtížné.

3.4.2 **Minimax**

Jde o rozhodovací pravidlo, které obecně nachází uplatnění u strategických her, při kterých se hráči rozhodují mezi jednotlivými tahy, od nichž se pak odvíjejí další rozhodnutí. Průběh takovýchto her, včetně jejich alternativních pozic, které by nastaly při zvolení jiných tahů, se dá zaznamenat stromem, tak jak je tomu na obrázku 3. Samotným principem

minimax algoritmu je minimalizace maximálních ztrát.[20] V našem případě ztráty a zisky vycházejí z hodnocení pozic, tak jak bylo načrtnuto v předchozí podkapitole.



Obrázek 3 - Minimax strom [20]

Na obrázku je zobrazen minimax strom včetně zvolených tahů. Každý řádek znázorňuje jeden půl tah, tedy jeden tah jednoho z hráčů, v následujícím řádku je pak vidět vybraný tah protihráče. Zatímco maximalizační hráč bude vybírat tahy s nejvyšší hodnotou, jeho oponent, minimalizační hráč, bude chtít vybírat tahy s nejnižším ohodnocením. Na obrázku jsou tahy, které by vybíral maximalizační hráč v kolečku, tahy minimalizačního hráče pak jsou ve čtverci. V této situaci je také vidět, že nejvýhodnějším tahem pro maximalizačního hráče a tedy tah, který zahraje, je ohodnocen -7, což znamená, že jde o tah, který je výhodný pro jeho oponenta, nicméně pro maximalizačního hráče neexistuje lepší alternativa.

Je důležité podotknout, že toto byl pouze zjednodušený příklad, v šachu může hráč v každé pozici učinit 30 tahů a od každého tohoto tahu se odvíjejí nové pozice, kde je možné opět učinit značné množství tahů. Náročnost výpočtů tedy roste s hloubkou exponenciálně. Tento problém pak řeší, aspoň do jisté míry, metoda alpha-beta ořezávání.

Alpha-beta ořezávání

Jde o vylepšení minimax algoritmu, jehož principem je eliminace větví stromu, u kterých je jasné, že jejich evaluace stejně neovlivní vybrání tahu. To spočívá v pozorování, že pokud už existuje lepší půl tah, tak nemá smysl propočítávat alternativní tahy, ani jejich

případné důsledky. Výsledkem využívání této metody je markantní snížení výpočetního času.[21]

Negamax

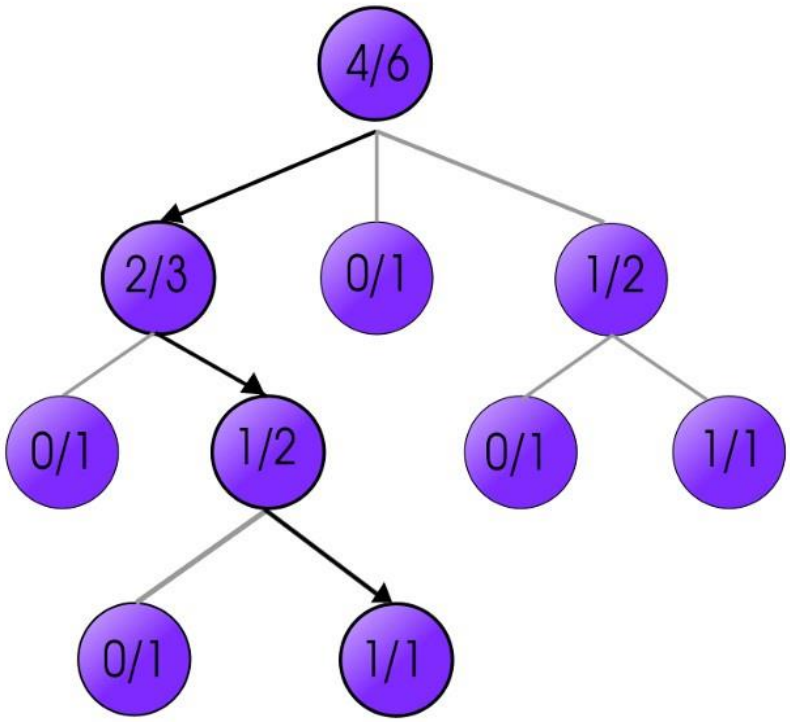
Negamax je variací minimaxu, jejíž výhodou je, že místo implementace dvou minimax metod, jedné pro minimalizačního hráče a druhé pro maximalizačního, vytvoříme pouze jednu. To vychází z faktu, že pozice na šachovnici, a tudíž ohodnocení, je pro oba hráče stejná, tudíž nám stačí pro jednoho z hráčů pouze otočit znaménko. Matematický zápis by vypadal takto.[32]

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

Samozřejmě toto je spíše úprava kosmetického rázu, nicméně stojí za zmínění, poněvadž v praxi se prostá minimax funkce moc neobjevuje.

3.4.3 Monte-Carlo metoda

Krom minimaxu existuje ještě jiný alternativní algoritmus. Jak jsem poukázal v kapitole *Vyhodnocení šachu*, tak napsat přesnou vyhodnocovací funkci šachových pozic, která bude brát v potaz i faktory jako je struktura pěšáků, je obtížné, a tudíž můžeme předpokládat, že v praxi jsou tyto vyhodnocovací funkce často mírně nepřesné. Problém pak je, že minimax algoritmus provádí takové množství výpočtů s těmito drobnými nepřesnostmi, že z malých nepřesností se stávají velké. Výhodou Monte-Carlo tree search algoritmu je, že díky své architektuře není tak náchylný na nedokonalosti vyhodnocovacích funkcí.[22] Principem Monte-Carlo metody je simulace her. Z jakékoliv pozice můžeme nasimulovat tisíce náhodných her, pokud jeden z hráčů, třeba bílý, vyhraje 70% těchto nasimulovaných her, tak můžeme, i bez jakékoliv vyhodnocovací funkce, usoudit, že nynější stav hry favorizuje bílého hráče. Stejně tak můžeme rozpoznat „dobré“ tahy, tedy ty, které povedou k vyhraným hrám nejčastěji. Strom tohoto algoritmu je znázorněn na následujícím obrázku.



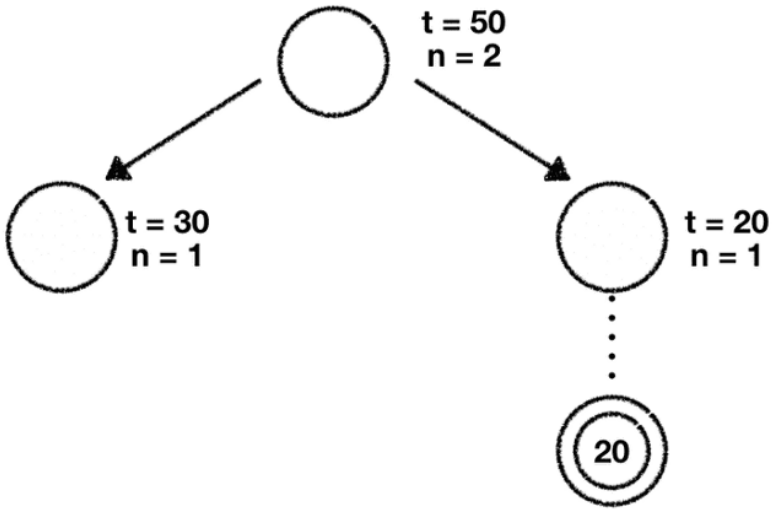
Obrázek 4 - Monte-Carlo strom [23]

Hodnoty reprezentují šanci na vítězství, herní AI bude pak samozřejmě volit tahy a cestu s největší šancí na výhru, tato cesta je na obrázku znázorněna tučnou čarou.

Ještě je důležité zmínit hodnotu UCT (Upper Confidence Bound), která determinuje, v jakém pořadí budou jednotlivé uzly, uzly v našem případě reprezentují tahy, prozkoumávány. Rovnice výpočtu UCT vypadá takto:

$$UCT = x_i + C\sqrt{\frac{ln(N)}{n_i}}$$

Kde x_i je poměr výher, C je konstantní hodnota, která určuje, jak moc se budeme soustředit na jednotlivé uzly. Nízká hodnota C bude znamenat, že se budeme často vracet k už před tím navštíveným uzlům, vysoká hodnota pak znamená, že algoritmus se bude spíše soustředit na neprozkoumané uzly. Hodnota N zaznamenává, kolikrát byl rodičovský uzel (uzel od kterého se odvíjejí další uzle) už navštíven a hodnota n_i je kolikrát byl dceřiný uzel navštíven.[24] Výpočet demonstruji na příkladě.



Obrázek 5 -Výpočet UCT [23]

Hodnota t je totální hodnotou uzlu a vychází ze součtu poměru výher jeho dceřiných uzlů, hodnota n je pak, kolikrát byl daný uzel navštíven. Hodnoty levého uzle (S1) a pravého (S2) dosadíme do rovnice. Musíme také určit hodnotu C, já budu počítat s C = 2. Výpočty by vypadaly následovně:

$$S1 = 30 + 2\sqrt{\frac{\ln(2)}{1}} = 31{,}38 \qquad S1 = 20 + 2\sqrt{\frac{\ln(2)}{1}} = 21{,}38$$

Hodnota S1 je vyšší, tudíž bude tou, kterou algoritmus bude zkoumat dále.

I přes výhody Monte-Carlo algoritmu jde dnes stále o méně populární metodu zkoumání stromů, než je minimax. To je dáno velkou zátěží na paměť. Jde také o mírně nespolehlivý algoritmus, v praxi může přehlédnout větev stromu, což může zapříčinit prohru partie. Posledně je také časově náročný, než je schopen efektivně rozhodnout, která cesta je optimální, tak musí projít celý strom několikrát, minimax na druhou stranu je schopný poskytnout hratelnou možnost takřka okamžitě, byť by šlo pouze o výsledek hledání s malou hloubkou.[25] V posledních letech se ovšem Monte-Carlo algoritmus začal dostávat do popředí šachových AI turnajů díky Alpha Zero, který ho používá.[26]

3.5 Podpůrné nástroje

Tvorba šachových AI je populárním tématem, a tudíž existuje značné množství podpůrných nástrojů. Může jít o knihovny usnadňující samotné programování šachového enginu, například knihovna generující tahy, či knihovny zdokonalující už existující šachové AI, jako jsou knihovny obsahující šachová zahájení. Dále pak existují složitější programy obsahující jak generování tahů včetně UI, tak i jiné šachové enginy, proti kterým se pak může testovat vlastní šachový engine. Na jednotlivé zástupce se blíže zaměřím v následujících kapitolách.

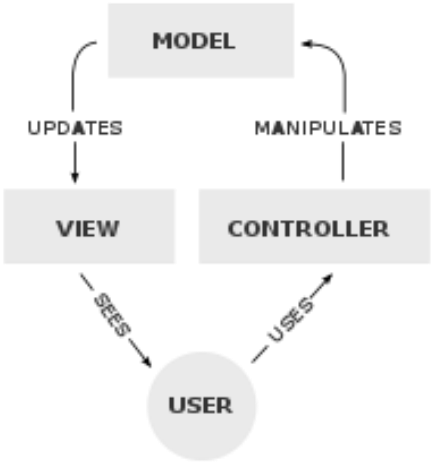
3.5.1 **Knihovny**

Knihovny se dají popsat jako kolekce standartních skriptů a programů, které mohou vývojáři v případě potřeby použít. Obvykle obsahují základní funkcionalitu, nicméně knihovny můžou být používány i jakožto skladiště dat, které pak bude ovlivňovat funkcionalitu hlavního programu.

Během této práce jsem se několikrát odvolával na šachovou teorii, nicméně jsem se ještě nezaměřil na aspekt, který s teorií šachu bývá asociován nejčastěji. Tím je teorie zahájení a koncovek. Dnes existují tisíce knih zabývající se právě těmito tématy, proto by bylo výhodné zužitkovat moudrost nespočtu šachových expertů a zakomponovat jejich znalosti do šachového AI. Toho se dosáhne pomocí knihoven obsahujících sekvence šachových tahů. Nejprve se zaměřím na knihovny zahájení. Výhody využívání těchto knihoven jsou jasné. Šetří čas, šachové AI nemusí počáteční pozice propočítávat a zároveň bude hrát kvalitnější tahy. Přeci jen i sofistikovaný engine nemusí chápat dlouhodobé strategické aspekty, či jiné hlubší faktory. Je celá řada těchto knihoven a jsou volně dostupné na internetu. Jednou z těchto knihoven je například Cerebellum 3, který obsahuje 10932446 pozic.[27] Dále pak existují také knihovny koncovek. Koncovky mohou být vyhrané i při stejném materiálu, což by šachový engine, který nehledá dostatečně hluboko ve stromě, mohl přehlédnout. Za využití těchto knihoven je tedy šachové AI nejen schopno konvertovat složitější koncovky na vítězství, ale zároveň může k těmto koncovkám záměrně směřovat. Množství pozic obsažených v knihovně se odvíjí od počtu figur přítomných na šachovnici, zatímco Syzygy knihovna pro 4 figury obsahuje 125,246,598 pozic, tak Sygzy knihovna pro 6 figur má v sobě uloženo 3,787,154,440,416 pozic, stejně závratně roste pak i zátěž na paměť. [28, 29]

3.5.2 **Grafická uživatelská rozhraní (GUI)**

Tyto softwary řeší interakci mezi hráčem a šachovým programem, popřípadě pouze mezi šachovými programy. Dnes krom základního rozhraní tyto softwary nabízejí celou škálu dalších funkcionalit jako grafová evaluace pozic, editace pozic, či předinstalované šachové enginy, popřípadě možnost přidat vlastní. Grafická rozhraní jsou budovány podle MVC (Model-view-controller) architektury. Diagram této struktury vypadá následovně.



Obrázek 6 - Diagram MCV architektury [30]

Jak je na obrázku vidět, tak se MCV architektura skládá ze 3 komponentů a samozřejmě ještě zahrnuje uživatele, který na základě výstupů (view) vkládá vstupy do řadiče (controller), řadič konvertuje vstupy uživatele na příkazy, které jsou zpracovávány v modelu. Výpočty, změny, atd, provedené v modelu, jsou zobrazovány uživateli v pohledu (view). [31] Konvence, podle kterých jsou data mezi GUI a šachovým AI sdíleny, se nazývají protokoly, nejpoužívanějšími z nich jsou Chess Engine Communication Protocol (CECP) a Universal Chess Interface (UCI). Zástupci grafických uživatelských rozhraní využívající (CECP) jsou například WinBoard a ChessX, zatímco zástupci pro UCI jsou třeba Aquarium a Fritz GUI. Existují ale také grafická rozhraní podporující oba, jako je například Arena.

3.6 Měření hardwarové náročnosti

Existuje celá řada parametrů, které se při testování aplikací, či jiných softwarů, dají měřit. Nicméně v rámci této bakalářské práce se zaměřím na pro nás relevantní parametry. Těmi tedy jsou primárně CPU, pak RAM paměť a samozřejmě celková velikost šachového AI také hraje roli.

3.6.1 Paměť

Co se týče paměti, tak to je poměrně prostou záležitostí. Závisí totiž převážně na velikosti knihoven, které jsou použity. Jak jsem poukázal v kapitole *Knihovny*, tak velikost těchto knihoven se může značně lišit v závislosti na množství pozic, které obsahují. Například 3-4-5 Syzygy knihovna (obsahuje koncovky pro 3, 4, 5 figur) má velikost 939 MB, je tedy otázkou, do jaké míry tyto knihovny ovlivní výslednou sílu šachového enginu, popřípadě, zda se vyplatí takto velké knihovny využívat.

3.6.2 RAM

Jde o krátkodobou paměť počítače. V našem případě je relevantní, poněvadž šachové enginy používají takzvanou hašovací tabulku. To je typ datové struktury optimalizovaný pro potřeby rychlého vyhledávání.[33] Principem je ukládání dat do polí, která jsou potom přístupná pomocí specifického unikátního klíče. Nedostatečná RAM paměť by poté negativně ovlivnila rychlost šachového enginu.

3.6.3 CPU

Právě CPU je limitující parametr dnešních šachových programů. Jak bylo v kapitole *Algoritmizace šachu* naznačeno, tak množství pozic, které je šachové AI schopno prozkoumat, je podmíněno výpočetní kapacitou. Toto znamená, že hloubka průzkumu pozic v určitém časovém úseku se odvíjí od CPU, specifičtěji od hodinové taktu (clock rate), množství jader hraje také roli, ale kvůli výpočetním neefektivnostem není dopad tak markantní.[34]

3.6.4 Měření

Pro měření zátěže CPU a RAM paměti existují různé nástroje. Jedním z nástrojů pro Windows je správce úloh, který zobrazuje základní informace o zátěži počítače, pokud je ovšem zapotřebí detailnější výpis, tak je potřeba stáhnout speciální software. Těchto

softwarů je mnoho, typicky fungují pouze pro jeden operační systém. Zástupci pro Windows jsou například Wise Systém Monitor nebo Rainmeter.

Dalším způsobem je za využití GUI softwaru. Některé z nich, jako například Arena, poskytují totiž technické informace o průběhu hry. Těmi jsou *Total number of nodes,* to je hodnota počtu prohledaných uzlů. Čím větší počet prohledaných uzlů, tím větší výpočetní a časová náročnost enginu. Další potenciálně užitečné hodnoty mohou být například *Nodes per second* a *Hashtable usage of memory*.

4 Vlastní zpracování

Existuje velké množství šachových aplikací, může jít o různé typy, počínaje jednoduchými pomůckami pro začátečníky, přes analytické nástroje, až po klasický herní enginy pro hru s lidmi. Nicméně potřeby jednotlivých aplikací se mohou lišit, a proto by bylo vhodné, kdyby existovalo objektivní zhodnocení různých algoritmů a funkcí, na základě jejich ovlivnění kvality herního enginu a jeho hardwarové náročnosti.

4.1 Cíl

Cílem praktické části je změření dopadu jednotlivých funkcí, algoritmů a podobně na kvalitu hry herního enginu a na jeho hardwarovou náročnost. Výsledkem by měly být poznatky užitečné při tvorbě různých šachových aplikací.

4.2 Postup zpracování

Základem měření této práce je šachový engine, od kterého se zbytek práce odráží. Byl napsán v jazyce C, poněvadž dodnes je v něm stále vytvořena většina návodů, tutoriálů a open source enginů. Základní funkcionalita, jakožto schopnost generovat tahy, či čtení pozic atd., vychází z různých šachových enginů a návodů, aby, krom ušetření práce, bylo také zajištěno správné fungovaní, bez bugů, které by pak mohli negativně ovlivňovat měření. Bližší popis je v následující podkapitole.

Jakožto uživatelské rozhraní je využíván software Arena, s kterou šachové AI komunikuje pomocí UCL protokolu a používá FEN (Forsyth–Edwards Notation) zápis šachových pozic. Využití Areny má tu výhodu, že je zde herní režim *Tournament*, kam může uživatel nahrát enginy a nechat je hrát proti sobě. Dále také existuje v Areně nástroj Elostat, která je z PGN (Portable Game Notation) zápisů her schopna vypočítat hodnotu Elo. Elo hodnota je pro potřeby této závěrečné práce spíše jen orientační a doplňující informací, přeci jen, změření hardwarové náročnosti šachového algoritmu, bez jakéhokoliv ohodnocení dopadu na sílu šachového enginu, by nemělo velkou vypovídající hodnotu o užitečnosti dané funkce.

Metodika testování jednotlivých funkcí a algoritmů je poměrně prostá. Nejprve byl vytvořen základní šachový program, který je schopný odehrát partii, bližší popis tohoto programu bude následovat v další kapitole. K tomuto programu byly pak přidávány další funkce a implementována různá zlepšení, pak následovalo změření efektu těchto změn na sílu šachového programu a na jeho hardwarovou náročnost.

Obrázek 7 - Nastavení turnaje v Areně [autor]

4.3 Základní šachový engine

Šachové AI, které použito pro testy v následujících kapitolách, bylo postaveno na algoritmech a funkcích, které jsou kostrou většiny dnešních šachových enginů, včetně těch nejlepších, jako je Stockfish. To znamená minimax algoritmus a využití alfa-beta ořezávání. Dále obsahuje primitivní evaluační funkci, která bere v potaz jen materiál a pozici jednotlivých figur. Celkově jde o poměrně jednoduchý program, který je sice schopný hrát poměrně decentní šachy, ale který by na silnější lidské protivníky nevystačil.

Z tohoto základu se pak odvíjejí jednotlivé verze šachového enginu, které budou v následujících kapitolách porovnávány. Jednotlivé verze se od sebe budou lišit na základě menších změn jako je například využití, či absence knihovny (viz. kapitola *Knihovna zahájení*) nebo třeba přesnost evaluační funkce (viz. kapitola *Hodnocení pozic*).

4.3.1 Postup tvorby enginu

Nejprve byla vytvořena maticová reprezentace šachovnice, šlo o prozatímní řešení, poněvadž eventuelně je implementováno uživatelské prostředí Areny. Dalším krokem bylo naprogramování jednotlivých figur a jejich legálních tahů, šlo i o tahy jako rošáda, či braní mimochodem. Dále muselo také být implementováno čtení a zápis pozic pomocí FEN notací, což je potřeba pro komunikaci s Arenou a pro Perft testování, které bude popsáno za chvíli. S touto základní funkcionalitou mohlo být přidáno generování tahů. To, zda engine generuje tahy správně se testuje pomocí Perft testu, ten funguje tak, že do daného enginu je nahrána známá šachová pozice, ta v tomto případně byla nahrána ve FEN formátu, a engine ji má prozkoumat a vypsat množství legálních tahů, které mohou být hrány. Pokud engine přijde se správnou hodnotou, tak je to potvrzení správnosti funkce generace tahů.

Po zajištění bezchybného fungování následovalo přidání evaluační funkce, aby byl engine krom generování tahů také schopný vybírat tahy dobré. Když už byl šachový schopný generovat tahy a hodnotit je, tak byl čas na implementaci minimax algoritmu a následného alfa-beta ořezávání. Po implementaci byl engine schopný hrát základní šachy, nicméně k dokonalosti to mělo daleko. Před dalšími zlepšení byl nejdříve dosavadní engine propojený s Arenou pomocí UCL protokolu. Aby s Arenou mohl engine komunikovat, tak musely být přidány další funkce jako schopnost zastavit prohledávání apod.

Posledními změnami byly implementace funkcí známých jako Quiescence a Move Ordering. Obě funkce jsou optimalizace prohledávacího algoritmu a mají za následek výrazné zlepšení kvality šachového enginu. Další úpravy tohoto enginu jsou pak předmětem měření v následujících kapitolách.

Tvorba takovéhoto šachového programu je poměrně složitá a zdlouhavá záležitost a tento projekt mi zabral přes 2 týdny aktivní práce, naštěstí existuje velké množství podpůrných materiálů a na internetových fórech je aktivní komunita, která je schopna s většinou věcí dobře poradit.

```
void SearchPosition(S_BOARD *pos, S_SEARCHINFO *info) {

    int bestMove = NOMOVE;
    int bestScore = -INFINITE;
    int currentDepth = 0;
    int pvMoves = 0;

    ClearForSearch(pos,info);

    if(EngineOptions->UseBook == TRUE) {
        bestMove = GetBookMove(pos);
    }

    if(bestMove == NOMOVE) {
        for( currentDepth = 1; currentDepth <= info->depth; ++currentDepth ) {

            rootDepth = currentDepth;
            bestScore = AlphaBeta(-INFINITE, INFINITE, currentDepth, pos, info, TRUE);

            if(info->stopped == TRUE) {
                break;
            }

            pvMoves = GetPvLine(currentDepth, pos);
            bestMove = pos->PvArray[0];

        }
    }
}
```

Obrázek 8 - Zdrojový kód prohledávací funkce [autor]

4.4 Vlastní měření

Hodnocení herní síly enginu vychází ze souboru několika desítek her, které byly odehrány v Areně a hodnota Elo vypočtena nástrojem ELOstat. Měření hardwarové zátěže budou převážně zaměřena na časovou náročnost. Další relevantní údaj je také velikost enginu, kterou poté značně ovlivní knihovny. Krom těchto jednoduše kvantifikovatelných hodnot se zaměřím ještě na jednu a tou je pracnost implementace, je to informace, která bude pouze slovně popsána. V praxi se totiž tvůrce šachové aplikace může setkat se situací, kde by naprogramování dané funkce mohlo vést k o něco silnějšímu šachovému AI, ale zda by to inkrementální zlepšení stálo za vynaložené úsilí by pak mohlo záležet na situaci, ve které se programátor nachází.

Informace, kterými se práce tedy zabývá jsou:

- CPU náročnost (čas na tah)
- Velikost souboru
- Síla herního programu – Elo
- Pracnost implementace

Výsledky těchto zkoumání jsou zaznamenány a okomentovány v následující části této práce.

4.4.1 Hloubka hledání

Čím hlouběji šachový engine situaci prozkoumává, tím déle mu trvá přijít s tahem, ale tím spíše najde tah silný. Při tvorbě šachového AI jde tedy o kompromis mezi herní sílou a výpočetní/časovou náročností. Předmětem tohoto měření je zjistit, jak hloubka prohledávání ovlivní výslednou Elo hodnotu.

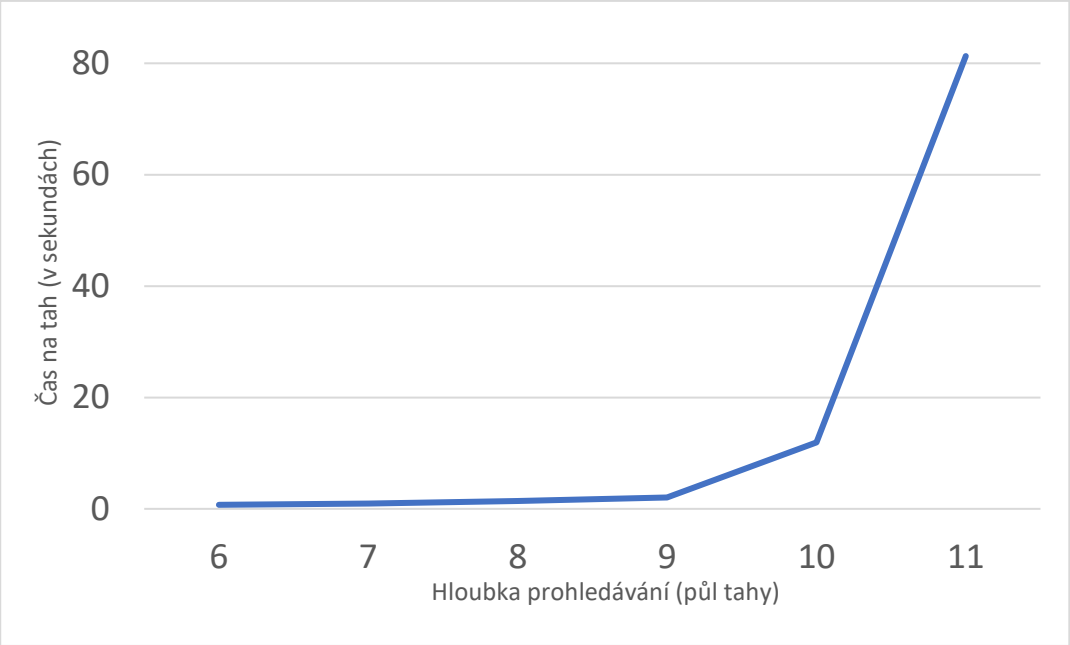
Hloubka a doba na tah

Nejprve bylo změřeno, jak hloubka hledání ovlivní časovou náročnost. Arena umožňuje nastavit *Fixed search depth*, to je proměnná, která určuje, do kolika půl tahů bude engine pozice prozkoumávat. Z hodnot počet tahů a z čas trvání byl vypočten čas na tah. Hry se ukončují po 40. tahu, aby bylo předejito koncovkám, kde už je situace natolik zjednodušená, například jako koncovka jen s králem a pěšákem, které by zkreslovali výsledky. Závěrečné hodnoty byly zaznamenány v Excelu, samozřejmě výsledná doba trvání se odvíjí od použitého hardwaru, nicméně křivka grafu bude mít stejný tvar, nezávisle na něm.

Testy byly prováděny pro hodnoty 6, 7, 8, 9, 10, 11. Pro hloubky 6, 7, 8, 9, 10 bylo provedeno 50 testů, pro hloubku 11 bylo provedeno pouze 10 testů, a to právě kvůli její časové náročnosti. Výsledný graf a tabulka dat vypadá následovně.

Hloubka	6	7	8	9	10	11
Čas na tah	0,725	0,937	1,43	2,038	11,95	81,3

Tabulka 6 - hodnoty pro graf na obrázku 9 [autor]



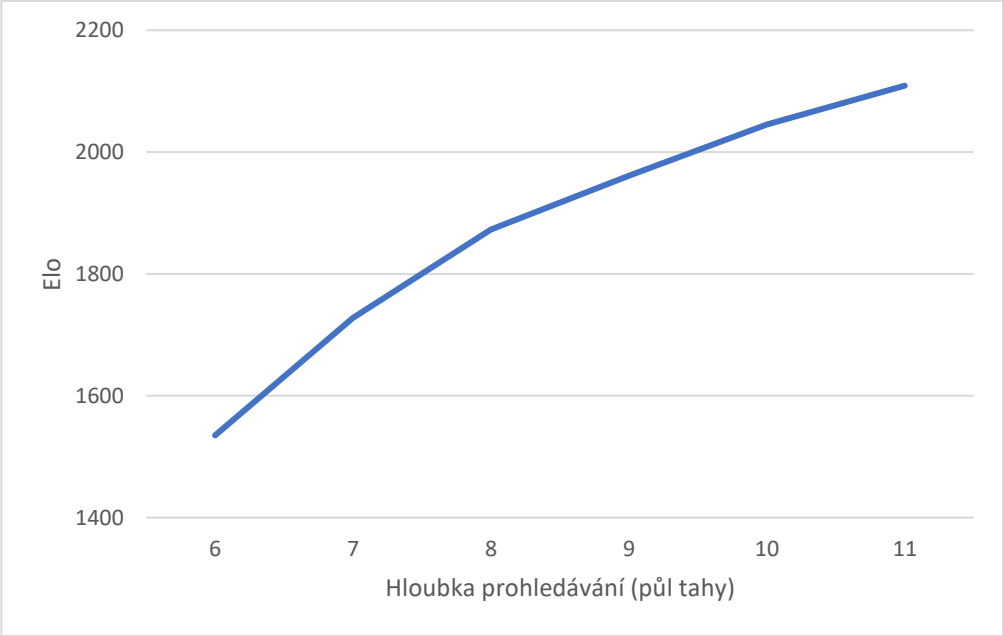
Obrázek 9 - Závislost hloubky prohledávání a času na tah [autor]

Závislost Ela na hloubce hledání

Doba trvání na nalezení tahu není jediná vypovídající informace o kvalitě herního programu. Další je hodnota Elo. Měření budou opět prováděna pro hodnoty 6, 7, 8, 9, 10, 11 hloubky hledání. Výpočet vychází ze souboru 50 her a je pro něj použit nástroj ELOstat. Jen pro hloubku 11 půl tahů byl kvůli časové náročnosti opět použit soubor 10 her. Výsledek je následující.

Hloubka	6	7	8	9	10	11
Elo	1535	1718	1863	1966	2053	2116

Tabulka 7 - hodnoty pro graf na obrázku 10 [autor]



Obrázek 10 - Elo v závislosti na hloubce prohledávání [autor]

Shrnutí

Co se týče hloubky prohledávání a časové/výpočetní náročnosti, tak je vidět, že volit hloubku větší než 9 půl tahů je pro potřeby většiny šachových aplikací neadekvátní přístup. Pokud by programátor potřeboval šachové AI, které prochází herní strom do takové hloubky, tak by pak musely být implementovány hashovací funkce a pozice a jejich ohodnocení by byly skladovány v RAM paměti, tak, jak je tomu u vrcholových enginů jako je Stockfish.

Zároveň je na druhém grafu vidět zpomalující růst koeficientu Elo po každém nárůstu prohledávání o půl tah. Tento výsledek jsou konzistentní s poznatky o zákonu klesajících výnosů (law of diminishing returns). Každopádně hlubší průzkum pozic samozřejmě znamená přesnější hru.

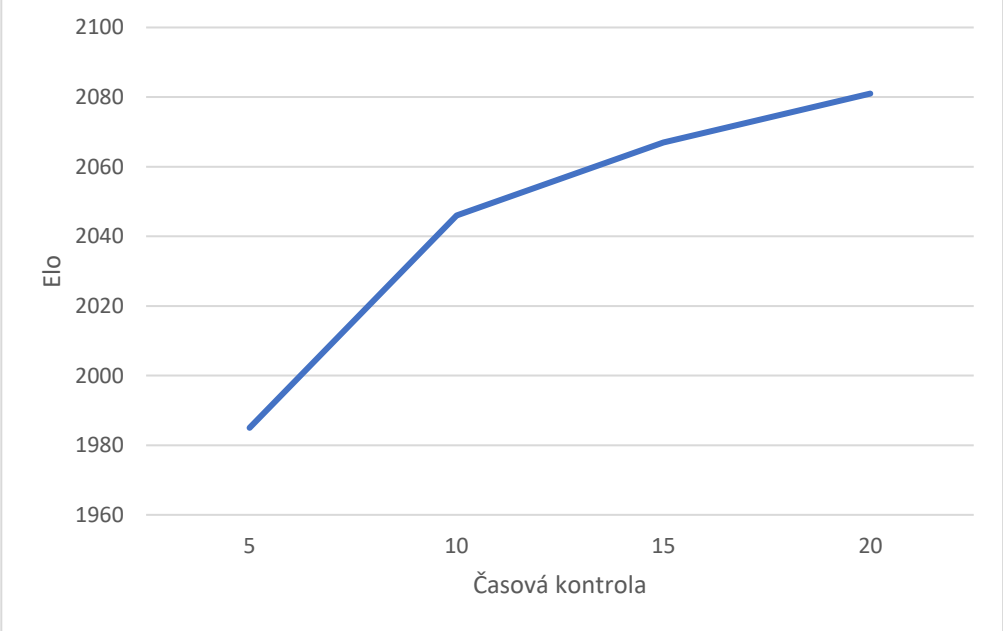
Z těchto dvou měření pak vyplývá, že optimální hloubka prohledávání se pohybuje kolem 7, či 8. U nižších hodnot by sice šachový program hledal tahy rychleji, nicméně by to nebyl markantní rozdíl, člověk by ho možná ani nezaznamenal a zbytečně by pak šachový program přišel o několik desítek Elo bodů. Hlubší hledání by už znamenala značné zpomalení, které by mohlo být problematické.

4.4.2 Časové omezení a Elo

Nejčastěji ale enginy nemají pevně danou hloubku prohledávání, ale jsou omezeny časovou kontrolou. Obvykle se jedná o pětiminutová, desetiminutová, patnáctiminutová omezení, popřípadě dvacetiminutová. Proto bylo změřeno Elo pro tyto časové kontroly. Měření bylo opět provedeno ELOstatem ze souboru 50 her. Výsledek vypadá takto.

Kontrola	Elo
Pětiminutová	1985
Desetiminutová	2046
Patnáctiminutová	2067
Dvacetiminutová	2081

Tabulka 8 - Časové omezení a Elo [autor]



Obrázek 11 - Elo v závislosti na časové kontrole [autor]

Nárůst hodnocení Elo sleduje stejnou křivku jako na obrázku 10. To je dáno opět dáno fenoménem klesajících výnosů. Při všech čtyřech časových omezeních zvládne šachový program projít pozice při zahájení a při střední hře do hloubky 9 půl tahů. Rozdíl pak nastává po zjednodušení, tedy po výměně několika figur. Při koncovce je hloubka prohledávání pro desetiminutové omezení kolem 11 půl tahů. A pro patnáctiminutové a

dvacetiminutové to může být 12, či 13 půl tahů. A při koncovce o dvou figurách se dostanou do hloubky i 19 půl tahů.

Také se zde dá poukázat na skutečnost, že je výrazně jednodušší implementovat fixní hloubku prohledávání, spíše než hloubku odvíjející se od časové kontroly, a podle výsledků měření se Elo hodnoty pro oba algoritmy překrývají, tudíž by se dalo odvodit, že pro většinu šachových aplikací je naprosto dostačující využívat fixní hloubku prohledávání a tím si ušetřit práci.

4.4.3 Hodnocení pozic

Jak bylo probráno v kapitole *Vyhodnocení pozice*, tak vytvořit jednoduchou evaluační funkci je poměrně rychlá záležitost, nicméně vzhledem k tomu, že minimaxový šachový engine volá tuto funkci při prohlížení každého uzlu, tak jakákoliv nepřesnost ve vyhodnocovací funkci může vést k velkým chybám. Proto je dalším testem porovnávání dvou verzí šachového AI. Jednoho, které používá jen základní matici materiálu, jako je na obrázku 2. Druhý pak bere v potaz i faktory jako izolované a volné pěšáky, pozici krále a zda jsou věže, či střelci v otevřených řadách nebo sloupcích. Je jasné, že vyhodnocovací funkce může být ještě dokonalejší a zahrnovat více faktorů, nicméně to by vyžadovalo hlubokou znalost šachové teorie. Samozřejmě implementace těchto dodatečných skriptů se nijak zásadně neprojeví na hardwarové náročnosti. Proto má měření jsou zaměřena jen na změnu síly herního enginu.

Měření první verze enginu proběhlo v předchozí kapitole. Elo druhé verze bylo vypočteno nástrojem Elostat ze souboru 50 her jen s pětiminutovou kontrolou. Další testy pro ostatní časová omezení by postrádala smysl. Výsledek je tento.

Verze	Elo
Verze 1 (základní)	1892
Verze 2 (lepší)	2039

Tabulka 9 - Hodnotící funkce [autor]

Jak je zde vidět, tak jde o nárůst 147 Elo bodů, což je poměrně značné zlepšení. A i přestože implementace lepší vyhodnocovací funkce vyžaduje bližší studium šachové teorie, tak je to určitě něco, na co by se tvůrci šachových aplikací měli zaměřit.

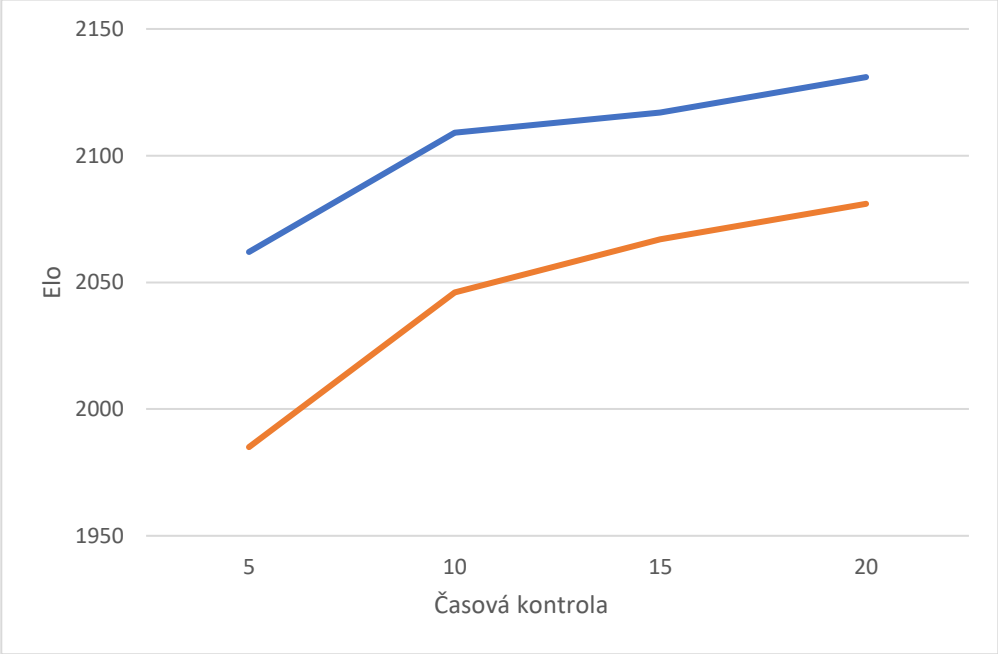
4.4.4 Knihovna zahájení

Implementace, která se už projeví na hardwarovou náročnosti je knihovna. Velikost knihoven se může lišit, ale klasicky se jedná o soubor o velikosti v řádech megabytů. Pro tuto práci byla použita 1,1 MB knihovna. To je knihovna Perfect_2010, jedna ze 3 základních knihoven nainstalovaných v Areně. Předmětem tohoto měření bude nárust hodnoty Elo pro různé časové kontroly. K výpočtu byl použit Elostat a soubor 50 pro každou časovou kontrolu. Výsledky vypadají následovně.

Časová kontrola	Elo
Pětiminutová	2062
Desetiminutová	2109
Patnáctiminutová	2117
Dvacetiminutová	2131

Tabulka 10 - Nárůst Ela při použití knihovny [autor]

Z hodnot byl opět vytvořen graf, do grafu byla taky přidána křivka z grafu na obrázku 8. Červená křivka tedy znázorňuje hodnoty Elo pro engine nevyužívající knihovnu, modrá křivka je pak engine využívající knihovnu.



Obrázek 12 - Časová kontrola a Elo [autor]

Na obrázku je vidět, že nárůst Ela se pohybuje kolem 50 bodů pro všechny časové kontroly a že jde o konzistentní inkrement. Z toho lze usoudit, že knihovny jsou užitečné spíše pro silnější šachové AI hrající aspoň na úrovni 2000 Elo bodů a výše. To proto, že pro dosáhnutí vyššího Ela u slabších enginů stačí nastavit vyšší hloubku hledání, zatímco u silnějších toto není z praktických důvodů možné kvůli časové náročnosti. Tento fenomén klesajících výnosů byl demonstrován v měřeních na obrázcích 9 a 10.

5 Výsledky a diskuze

Byl vytvořen šachový engine, na kterém byla prováděna měření předchozích kapitol. Základ toho šachového programu byl popsán v podkapitole *Základní šachový engine*, k této kostře byly pak implementovány funkce probrané v předchozích kapitolách. Výsledkem je pak klasické herní AI, které hraje konzistentně kolem 2,2 tisíc Elo bodů při pětiminutové kontrole. To z něj činí pro člověka silného protivníka, ale samozřejmě v porovnání s profesionálními šachovými enginy neobstojí. Opět byly provedeny testy pro různé časové kontroly. Pro každou časovou kontrolu byl proveden výpočet Ela ze souboru 50 her.

Časová kontrola	**Elo**
Pětiminutová	2203
Desetiminutová	2225
Patnáctiminutová	2343
Dvacetiminutová	2254

Tabulka 11 - Výsledný šachový engine [autor]

Další krok, který by byl potřeba, pokud by tento engine měl hrát kvalitněji, by byla implementace ukládání pozic a jejich ohodnocení do RAM paměti, dále by pak mohla být evaluační funkce dále optimalizována. Krom knihovny zahájení by také mohla být přidána knihovna koncovek. Nicméně, tak jak existuje šachový engine teď, tak by vystačil pro potřeby většiny šachových aplikací.

6 Závěr

Hlavním cílem této práce bylo měření hardwarové náročnosti jednotlivých funkcí šachových enginů. K tomuto cíli bylo potřeba vytvořit vlastní šachový program, aby byla zajištěna dostatečná kontrola všech faktorů, které by mohly měření ovlivňovat. Jde o poměrně klasický program fungující na známých metodách, což činí na něm prováděné testy užitečné pro velkou škálu šachových aplikací.

Měření byla zaměřena na základní části šachového AI jako jsou hloubka hledání a evaluační funkce, které se vyskytují ve většině šachových enginů. Poznatky vyplývající ze zkoumání této práce by měly najít využití při tvorbě různých šachových aplikací, kde by umožnily volbu správného postupu s ohledem na dané požadavky.

Celkově je tedy přínos této práce spíše praktického rázu, s tím, že měření byla konzistentní se známými fenomény jako je exponenciální nárůst časové a výpočetní náročnosti enginu s rostoucí hloubkou prohledávání a zároveň stále snižujícího se výnosu s rostoucí hloubkou.

7 Zdroje

1) History of chess | From Early Stages to Magnus. *Chess.com* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.chess.com/article/view/history-of-chess

2) Andrew E. Soltis. History. V *Britannica* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.britannica.com/topic/chess/History

3) Patrick Gebhardt. The History of Chess AI. *Paessler* [online]. (navštíveno v říjnu 2020) Dostupné z: https://blog.paessler.com/the-history-of-chess-ai

4) ELO Probability Table. *318chess* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.318chess.com/elo.html

5) What is an Elo rating? *Thechesspiece.com* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.thechesspiece.com/what_is_an_elo_rating.asp

6) Top 100 Players October 2020. *FIDE* [online]. (navštíveno v říjnu 2020) Dostupné z: https://ratings.fide.com/

7) Top Player Lists. *US Chess Federation* [online]. (navštíveno v říjnu 2020) Dostupné z: http://www.uschess.org/component/option,com_top_players/Itemid,371?op=list&month=2010&f=foreign&l=R:Top%20Overall.&h=Overall%20regardless%20of%20Country,%20Residence,%20or%20Federation

8) Jørgen Veisdal. The Mathematics of Elo Ratings. *Medium* [online]. (navštíveno v říjnu 2020) Dostupné z: https://medium.com/cantors-paradise/the-mathematics-of-elo-ratings-b6bfc9ca1dba

9) Aravind Kolumum Raja. Visualizing FIDE chess rating list. *NYC Data Science Academy* [online]. (navštíveno v říjnu 2020) Dostupné z: https://nycdatascience.com/blog/student-works/visualizing-fide-chess-rating-list/

10) *TCEC* [online]. (navštíveno v říjnu 2020) Dostupné z: https://tcec-chess.com/

11) About. *Stockfish Chess* [online]. (navštíveno v říjnu 2020) Dostupné z: https://stockfishchess.org/about/

12) Leandro Álvarez Gonzáles. What is Lc0. *GitHub* [online]. (navštíveno v říjnu 2020) Dostupné z: https://github.com/LeelaChessZero/lc0/wiki/What-is-Lc0%3F-(for-non-programmers)

13) Gerwin de Groot. What is Alpha Zero's Playstyle? *Chess Stackexchange* [online]. (navštíveno v říjnu 2020) Dostupné z: https://chess.stackexchange.com/questions/21781/what-is-alpha-zeros-play-style-including-its-openings-and-its-middlegame

14) Sebastian Anthony. Rybka, the world's best chess engine, outlawed and disqualified. *ExtremTech* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.extremetech.com/extreme/88610-rybka-the-worlds-best-chess-engine-outlawed-and-disqualified

15) Kateřina Helán Vašků. Kolik je v jedné partii možných tahů? Více než atomů v pozorovatelném vesmíru! *100+1* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.stoplusjednicka.cz/kolik-je-v-jedne-sachove-partii-moznych-tahu-vice-nez-atomu-v-pozorovatelnem-vesmiru

16) *Chess for novices* [online]. (navštíveno v říjnu 2020) Dostupné z: http://www.chessfornovices.com/chesspiecevalues.html

17) Dostupné z: http://billwall.phpwebhosting.com/articles/Value.htm

18) Lauri Hartikka. A step-by-step guide to building a simple chess AI. freeCodeCamp [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/

19) Dostupné z: http://en.wikipedia.org/wiki/Image:Minimax.svg

20) Minmax Algorithm in Game Theory | Set 1 (Introduction). *GeeksforGeeks* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

21) Minmax Algorithm in Game Theory | Set 4 (Alpha -Beta Pruning). *GeeksforGeeks* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

22) Philipp Muens. Game Ais with Minimax and Monte Carlo Tree Search. *Towards data science* [online]. (navštíveno v říjnu 2020) Dostupné z: https://towardsdatascience.com/game-ais-with-minimax-and-monte-carlo-tree-search-af2a177361b0

23) Sagar Sharma. Monte Carlo Tree Search. *Towards data science* [online]. (navštíveno v říjnu 2020) Dostupné z: https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa

24) What is MCTS? *Swarthmore* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.cs.swarthmore.edu/~mitchell/classes/cs63/f20/reading/mcts.html

25) Rahul Roy. ML | Monte Carlo Tree Search (MCTS). *GeeksforGeeks* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/

26) Stephan Oliver Platz. Monte Carlo instead of Alpha-Beta? *Chess Base* [online]. (navštíveno v říjnu 2020) Dostupné z: https://en.chessbase.com/post/monte-carlo-instead-of-alpha-beta (navštíveno v říjnu 2020)

27) Dostupné z: https://zipproth.de/Brainfish/download/

28) Kirill Kryukov. NULP. (navštíveno v říjnu 2020) Dostupné z: http://kirill-kryukov.com/chess/nulp/results.html

29) Niklas Fiekas. Syzygy endgame tablebases. (navštíveno v říjnu 2020) Dostupné z: https://syzygy-tables.info/

30) Dostupné z: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#/media/File:MVC-Process.svg

31) Erin Doherty. MVC Architecture. *Educative* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.educative.io/blog/mvc-tutorial

32) Jarkyn. Joys of minimax and negamax. *Medium* [online]. (navštíveno v říjnu 2020) Dostupné z: https://medium.com/@indykidd/joys-of-minimax-and-negamax-ee5e456977e2

33) Hashovací tabulka. *IT network* [online]. (navštíveno v říjnu 2020) Dostupné z: https://www.itnetwork.cz/navrh/algoritmy/algoritmy-vyhledavani/algoritmus-vyhledavani-hashovaci-tabulka

34) CPU clock speed or core count? *TalkChess* [online]. (navštíveno v říjnu 2020) Dostupné z: http://talkchess.com/forum3/viewtopic.php?t=6964