VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ŠACHY A UMĚLÁ INTELIGENCE
CHESS AND ARTIFICIAL INTELLIGENCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Miloslav Macůrek

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Radomil Matoušek, Ph.D.

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Miloslav Macůrek**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Šachy a umělá inteligence

Stručná charakteristika problematiky úkolu:

Cílem práce bude proniknout do problematiky tvorby algoritmů umělé inteligence pro hru šachy a implementovat vlastní řešení umělého hráče využívajícího jak standardní heuristiky tak metody strojového učení (neuronové sítě).

Cíle diplomové práce:

1. Základní rešerše šachových programů využívajících heuristiky a případové usuzování.
2. Rešerše AI šachových programů obzvláště AlphaZero.
3. Implementace rozhraní pro připojení hráče přes UCI protokol (např. Fritz, Arena).
4. Implementace AI šachového programu.
5. Vyhodnocení výsledků z realizovaných partií.
6. Elektronická prezentace výsledků.

Seznam doporučené literatury:

BIRD, H. E. Mistrovské perly. Přeložil Antonín ČÍŽEK. Ostrožská Nová Ves: Galerie Dolmen, 2017. ISBN 978-80-87303-39-9.

POLGÁR, László. Šachy: 5334 úloh, kombinací a partií. Přeložil Pavel MATOCHA. Praha: Slovart, [2016]. ISBN 978-80-7529-184-4.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2018/19

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Práce se zabývá algoritmy umělé inteligence pro hru šachy a tvorbou programu, který je implementuje. Rešerše obsahuje základy šachové hry a její historii se zaměřením na počítačový šach, klasické metody v šachovém programování a základní shrnutí neuronových sítí s možnostmi jejich aplikace. Vybrané algoritmy jsou implementovány v navrženém šachovém programu „Beast“.

ABSTRACT

This thesis will cover the topic of artificial intelligence algorithms in the game of chess and their implementation in computer chess program. The research contains the basics of the chess game and its history with a focus on computer chess, classical methods in chess programming and basic summary of neural networks and possibilities of their application. Selected algorithms are further implemented in chess program “Beast”.

KLÍČOVÁ SLOVA

Šachy, umělá inteligence, neuronové sítě, prohledávací algoritmy.

KEYWORDS

Chess, artificial intelligence, neural networks, search algorithms.

BIBLIOGRAFICKÁ CITACE

MACŮREK, Miloslav. *Šachy a umělá inteligence* [online]. Brno, 2019 [cit. 2019-05-22]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/117479. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Radomil Matoušek.

PODĚKOVÁNÍ

Děkuji doc. Ing. Radomilu Matouškovi, Ph.D. za poskytnutou odbornou pomoc a vedení a Ing. Petru Šoustkovi za cenné rady.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením doc. Ing. Radomila Matouška, Ph.D. a s použitím literatury uvedené v seznamu literatury.

V Brně dne 24. 5. 2019 .....

Bc. Miloslav Macůrek

OBSAH

ÚVOD

Šachy jsou již od středověku považovány za hru, jejímž ovládnutím hráč prokazuje svou inteligenci a schopnost logického a kreativního myšlení. Proto se hra také stala vyhledávaným soupeřem pro autory systémů a umělých inteligencí, jejichž cílem bylo prokázat schopnosti podobné lidskému rozhodování při řešení složitých problémů, které pouhá výpočetní síla není schopna zvládnout.

Rozmach počítačového šachu byl umožněn nástupem stolních počítačů, jejichž univerzálnost a jednoduchost použití usnadnila vývoj těchto systémů. Klasické metody využívané s velkým úspěchem prohledávají stavový prostor hry na základě heuristik a omezení pro snížení výpočetní náročnosti. Nárůst výpočetního výkonu otevírá nové možnosti aplikace moderních algoritmů, které v minulosti nebyly využitelné. Konkrétně se jedná o neuronové sítě, které v posledních letech způsobily revoluci nejen ve světě počítačového šachu, ale také v mnoha dalších odvětvích.

Tato práce si dává za cíl zpracovat problematiku počítačového šachu a běžně využívaných prostředků při budování šachových programů, spolu s přehledem moderních přístupů se zaměřením na neuronové sítě. Tyto poznatky budou aplikovány a demonstrovány v originálním šachovém programu Beast, který bude prezentovat jak jednoduchou klasickou heuristiku, tak heuristiku naučenou neuronovou sítí.

Práce je dělena do 8 kapitol, které zahrnují rešeršní část, popisnou teoretickou část návrhu a vlastní popis programové implementace v podobě programu Beast. První kapitola *Šachy* pojednává o dávné i moderní historii hry, kapitola 2 *Počítačový šach* se věnuje historii a zástupcům umělých, především počítačových systémů pro hraní šachu. Ve třetí kapitole *Klasické metody* jsou přiblíženy běžně používané algoritmy a přístupy počítačového programování. Kapitola 4 obsahuje poznámky k neuronovým sítím, kapitola 5 pojednává o testování herní síly šachových programů. Kapitoly 6 a 7 popisují tvorbu programu Beast, jeho testování a hodnocení dosažených výsledků.

1 ŠACHY

O vývoji šachu máme záznamy za posledních 1500 letech. Hra prodělala mnoho menších či větších změn, než se její pravidla relativně ustálila kolem 15. století v podobě, ve které je víceméně známe dodnes. Hra se nicméně nadále vyvíjí a přizpůsobuje trendům a adaptuje na nové technologie či překážky, které moderní doba přináší. Šachy lze zařadit do žánru „válečných her“, neboť jejím hlavním cílem je zvítězit v bitvě vedením jedné ze dvou armád a porazit soupeře, který vede armádu druhou. Detailní pravidla hry zde rozebírána nebudou, neboť jsou většině lidí dobře známá díky vysoké oblibě hry po celém světě. V další části textu bude pouze zevrubně shrnuta jejich historie a změny, kterými pravidla prošly v průběhu času.

Hra spadá do kategorie her jednoduchých na naučení, složitých na ovládnutí. Naučit se šachy je otázkou jednoho odpoledne, ale jejich ovládnutí a například získání mistrovského titulu udělovaného mezinárodní šachovou federací FIDE, nehledě na titul mistra světa, je těžká práce na celý život, která vyžaduje nejen odhodlání a pracovitost, ale i talent. [1, 2]

1.1 Dávná historie

Vznik hry šachy se datuje do 5. století, ze kterého máme první záznamy o její existenci z Indie, i když někteří experti předpokládají prapůvodního předka z Číny. Čuturanga, či *chuturanga,* již obsahovala všechny základní figury známé ze současné hry, byť s odlišným názvem a vzhledem (pěchota, jízda, sloni, válečné vozy, král a jeho poradce). Odtud se hra dostala do Persie, odkud se, po jejím dobytí muslimy v polovině 7. století, pod názvem šatrandž či *chatrang* rozšířila po celém tehdejším muslimském světě. Evropa poprvé spatřila šachy v 9. století a hra byla převzata jak ze zemí dnešního Ruska, tak z jižních muslimských států. Zde hra prodělala poslední podstatné změny a v 15. století vzniká moderní šach tak jak ho známe dodnes.

Změny probíhaly nejvíce na území dnešního Španělska a Itálie. Nejzásadnějšími byly změny v pohybu figur. Původní poradce krále, který byl omezen pouze na pohyb jedno pole po diagonále, byl nahrazen moderní dámou, která se pohybuje o neomezenou vzdálenost diagonálně po řadách i sloupcích. Střelec se nadále pohybuje pouze po diagonálách, ale jeho pohyb už také není omezen na původní 2 pole. Pěšci zase pro rychlejší zahájení hry dostali možnost skoku z původního postavení o 2 pole místo jednoho, což také vedlo k zavedení braní mimochodem, či *en passant*, pravidla, které dodnes způsobuje rozhořčení mezi začínajícími šachisty, kteří se s ním setkávají poprvé – ve specifické pozici kdy se pěšec pohne o dvě pole kupředu může být následný (ale pouze tento) tah brán soupeřovým pěšcem (ale pouze pěšcem) i jako kdyby se posunul pouze o pole jedno. Toto pravidlo bylo nutné pro řešení problémů v koncovkách, kde by pěšci skákající o dvě pole mohli uniknout obraně soupeře. Zavedena byla rošáda a sjednocena pravidla ukončení hry, především patu. [1, 2]

1.2 Moderní historie

Do moderní historie můžeme zařadit dobu od 15. století po současnost, tedy od ustálení pravidel moderního šachu do podoby velmi podobné tomu, jak je známe v dnešní době. Z 15. století je také zachována nejstarší známá tištěná šachová kniha, zabývající se teorií šachové hry. Autorem je Luis Ramírez de Lucena, podle kterého se jmenuje známá technika dosažení remízy v pozici, kde má hráč pouze krále a věž, kdežto jeho soupeř krále, věž a pěšce, kterého se snaží protlačit do dámy a vyhrát tak hru díky výhodě silné figury navíc.

Spolu s Lucenou se teorii věnovalo mnoho velikánů šachové historie, kteří zde nebudou detailně rozebíráni. Mnoho jejich příspěvků do teorie zahájení, koncovek, či technik útoků a matových manévrů nese dodnes jména jejich objevitelů. Z těchto byl nejznámější španělský biskup Ruy López de Segura, který prosadil dodnes nejhranější a nejoblíbenější šachové zahájení, tzv. Španělskou hru, či anglicky *Ruy Lopez* nebo *Spanish game*.

Šachové turnaje a zápasy se staly velmi populárními mezi širokou veřejností od 19. století. Počátkem trendu byly zápasy LaBourdonnais-McDonnel v roce 1834. Pro soutěžní utkání však bylo potřeba rozšířit pravidla o povolený čas ke hře. Pro vážné partie se ustálil formát daného času na daný počet tahů, který se s malými změnami používá dodnes.

Běžně užívaný časový formát v např. českých ligách byl až do nedávna 2 hodiny na 40 tahů + 1 hodina na dohrání partie pro každého hráče – partie tedy mohla trvat maximálně 6 hodin. V současné době se formát mírně mění ve prospěch systému s přidaným časem za každý tah, což snižuje počty partií prohraných na čas – moderním formátem je 90 minut na 40 tahů + 30 minut na dohrávku partie, ale v průběhu celé partie je každému hráči přidáváno 30 sekund za každý provedený tah. Celková doba partie se tak výrazně nemění a každý z hráčů má minimálně 30 sekund na každý tah. Prohra na čas je tak méně pravděpodobná i při špatném rozložení času v průběhu partie. Zavedení časomíry podnítilo také vznik nových variant šachu podle množství času, který hráči mají na odehrání partie. K vážným partiím popsaným výše se přidaly tzv. Rapid (10-60 minut pro každého hráče), Blitz (3-10 minut) a Bullet (pod 3 minuty) hry. Tyto časové kontroly jsou dnes velmi populární především pro hraní online.

V 19. století má také počátky titul mistra světa v šachu, nebo anglicky *World Chess Champion title*. V Praze narozený Wilhelm Steinitz odehrál v roce 1886 proti Johannesi Zukertortovi zápas, který je považován za první mistrovství světa a přesvědčivým vítězstvím se stal prvním mistrem světa. Tradice těchto zápasů se udržuje dodnes, mezi nejznámější mistry světa patří např. José Raúl Capablanca, Bobby Fischer, Garry Kasparov či Viswanathan Anand. Současným mistrem, který drží pomyslnou korunu od roku 2013 je Nor Magnus Carlsen, považovaný za jednoho z nejlepších šachistů historie.

20. století znamenalo obrovský rozmach šachu a přineslo mu vysokou popularitu mezi širokou veřejností. V roce 1924 byla založena světová šachová federace

FIDE (*Fédération Internationale des Échecs*), která rozhoduje o úpravě pravidel, sjednocuje jednotlivé národní federace, uděluje mistrovské tituly a pořádá mistrovství světa. Nově zavedenými mistrovskými tituly jsou Velmistr (*Grandmaster*, GM), Mezinárodní mistr (*International Master*, IM), Mistr FIDE (*FIDE Master*, FM) a Kandidát mistra (*Candidate Master*, CM). Tyto tituly jsou velmi prestižní ohodnocení šachových schopností a získat je není snadné.

Druhá polovina 20. století byla pod nadvládou sovětských šachistů, kteří díky výborným šachovým školám a skvělé výchově mládeže překonali celý zbytek světa. Každému šachistovi je tak známý legendární zápas o titul mistra světa z roku 1972, do kterého se mladý Američan Bobby Fischer dokázal dostat přes těžkou sovětskou opozici a sesadil z pomyslného šachového trůnu Borise Spasského v tzv. „zápase století“. Stal se tak prvním nesovětským mistrem světa od roku 1948. Do dalších zápasů však Fischer už nenastoupil a stáhl se do ústraní. Dokázal tím nicméně motivovat a nadchnout celou novou americkou generaci pro hru a díky jeho vlivu se šachy staly v USA velmi populárními.

Sovětská nadvláda pak pokračovala až do rozpadu SSSR a do roku 2000 byl mistrem světa hráč z nově vzniklé Ruské Federace. Velmi vlivnou postavou ovlivňující vývoj šachového světa především v 90. letech a začátku 21. století je původně sovětský, později ruský šachista Garry Kasparov. Titul mistra světa získal v roce 1985 po dlouhých legendárních bitvách s krajanem Anatoly Karpovem, se kterým se pravidelně utkával o titul až do roku 1993, kdy byl Karpov poražen v turnaji kandidátů Nigelem Shortem ze Spojeného Království. Ten tak získal právo vyzvat Kasparova místo Karpova. Oba, Kasparov i Short, se však vymezili proti organizaci FIDE, kterou vinili z korupce a neprofesionálního přístupu. Založili tak novou organizaci PCA (*Professional Chess Association*) se kterou organizovali své vlastní mistrovství světa. Dodnes se vedou diskuse o „pravém“ mistru světa v letech 1993-2005.

V roce 2006 se obě strany dohodly a uspořádaly společné mistrovství světa, které opět tento prestižní titul sjednotilo pod organizací FIDE. Tento zápas vyhrál Vladimir Kramnik z Ruska nad Veselinem Topalovem.

Kramnika však hned v roce 2007 sesadil Viswanathan Anand z Indie, který následně ztratil titul ve prospěch Magnuse Carlsena v roce 2013. Carlsen je současným mistrem světa. Tab. 1 obsahuje mistry světa od Wilhelma Steinitze až po rozdělení v roce 1993, Tab. 2 uvádí mistry světa v přechodném období a Tab. 3 novodobé mistry po obnovení jednotného mistrovství světa. [1, 2]

Tab. 1: Mistři světa v letech 1886-1993 [3]

Pořadí	Jméno	Roky	Národnost
1	Wilhelm Steinitz	1886–1894	Rakousko-Uhersko
2	Emanuel Lasker	1894–1921	Německo
3	José Raúl Capablanca	1921–1927	Kuba
4	Alexander Alekhine	1927–1935	Francie
5	Max Euwe	1935–1937	Nizozemsko
(4)	Alexander Alekhine	1937–1946	Francie
6	Mikhail Botvinnik	1948–1957	SSSR
7	Vasily Smyslov	1957–1958	SSSR
(6)	Mikhail Botvinnik	1958–1960	SSSR
8	Mikhail Tal	1960–1961	SSSR
(6)	Mikhail Botvinnik	1961–1963	SSSR
9	Tigran Petrosian	1963–1969	SSSR
10	Boris Spassky	1969–1972	SSSR
11	Robert James Fischer	1972–1975	USA
12	Anatoly Karpov	1975–1985	SSSR
13	Garry Kasparov	1985–1993	SSSR

Tab. 2: Mistři světa v době rozpolcení, 1993-2005 [3]

Organizace	Jméno	Roky	Národnost
PCA	Garry Kasparov	1993–2000	Rusko
PCA	Vladimir Kramnik	2000-2006	Rusko
FIDE	Anatoly Karpov	1993–1999	Rusko
FIDE	Alexander Khalifman	1999–2000	Rusko
FIDE	Viswanathan Anand	2000–2002	Indie
FIDE	Ruslan Ponomariov	2002–2004	Ukrajina
FIDE	Rustam Kasimdzhanov	2004–2005	Uzbekistán
FIDE	Veselin Topalov	2005–2006	Bulharsko

Tab. 3: Mistři světa po sjednocení, 2006-současnost [3]

Pořadí	Jméno	Roky	Národnost
14	Vladimir Kramnik	2006–2007	Rusko
15	Viswanathan Anand	2007–2013	Indie
16	Magnus Carlsen	2013+	Norsko

2 POČÍTAČOVÝ ŠACH

2.1 Historie

Historie počítačového šachu je stejně zajímavá a rozmanitá jako historie šachu samotného. Hra po dlouhou dobu představovala výzvu, doménu lidského myšlení. A proto také přitahovala a stále přitahuje badatele v oblasti umělé inteligence. První automatizovaný šachový systém se objevil už v 18. století a jmenoval se *The Turk*, tedy Turek. Šlo o robota hrajícího šachy proti vyzyvatelům a způsobil velkou senzaci. Nakonec však byl odhalen (po dlouhých 84 letech) jako podvod, uvnitř mechanického těla robota byl člověk, šachový mistr.

Opravdové první šachové systémy vznikly až ve 20. století. Teoretické základy k nim položilo mnoho světoznámých vědců z oblasti automatizace a informatiky, např. Alan Turing, John von Neumann či Claude Shannon. Teprve nárůst výpočetního výkonu však umožnil vznik opravdu silných šachových systémů založených na *brute-force* prohledávání stavového prostoru hry, propojeného s propracovanými heuristikami a *alpha-beta pruning* algoritmem, který je dnešním standardem.

Historický průlom přišel až v roce 1996, kdy systém *Deep Blue* společnosti IBM dokázal zvítězit jednu ze 6 partií v zápase proti tehdejšímu mistru světa Garry Kasparovovi. Přestože tento zápas nakonec prohrál 2-4, poprvé v historii byl šachový systém schopen zvítězit ve vážné partii proti mistru světa. O rok později, v květnu 1997, se zápas opakoval a *Deep Blue* byl výrazně vylepšen. Výsledkem bylo jeho těsné vítězství 3,5-2,5 a první porážka mistra světa v šachu automatickým systémem.

Výsledkem byl velký nárůst zájmu o šachové systémy, které s úspěchem začaly využívat klasické stolní počítače, jejichž nadále narůstající výkon tento vývoj umožnil. Přesto trvalo dalších 10 let než i počítačové šachové programy přesáhly úroveň lidských velmistrů. Nejznámější programy této doby, které s větším či menším úspěchem bojovaly proti lidským mistrům, byly např. Fritz (Frans Morsch a Mathias Feist), Shredder (Stefan Meyer-Kahlen), Junior (Amir Ban a Shay Bushinsky) či Hiarcs (Mark Uniacke).

Mezi lety 2007 a 2010 zcela dominoval program Rybka Čechoameričana Vasika Rajlicha. Tento se dostal do podvědomí mnoha dnešních šachistů jako nejsilnější na světě především díky velké propagaci německé firmy Chessbase, vydavatele šachových prostředí, se kterými se Rybka, Fritz a další silné enginy (z angl. *engine*, jedná se o výpočetní část šachového programu) prodávají. Dominance Rybky byla ukončena kontroverzí – autor byl obviněn z plagiátorství dvou, v porovnání s Rybkou velmi slabých, open-source šachových programů Fruit a Crafty. Přestože vina nebyla nikdy prokázána nad veškerou pochybnost, protože Vasik Rajlich nezveřejnil zdrojový kód Rybky, jeho engine byl vyřazen z mistrovství světa pořádaných organizací ICGA a byly mu zpětně odebrány veškeré získané tituly. To Rajlicha na dlouhou dobu odradilo od aktivního vývoje a Rybka se postupně propadla v žebříčku a byla překonána nově nastupujícími silnými programy.

Vládu Rybky převzal belgický Houdini (Robert Houdart), ale jeho dominance nikdy nebyla úplná, přestože od roku 2010 do 2013 vyhrával většinu titulů. Spolu s Houdinim začaly stoupat další dva velmi silné programy, se kterými až do nedávna tvořil tzv. Velkou trojku světa počítačového šachu – Komodo (Don Daily, Mark Lefler, GM Larry Kaufman) a Stockfish (open-source, Marco Costalba, Joona Kiiski, Gary Linscott, Tord Romstad a kolektiv). Patovou situaci nakonec přiklonil Stockfish na svou stranu a v roce 2016 překonal své dva hlavní soky, kteří se víceméně neúspěšně pokouší dohnat výkonnostní rozdíl až dodnes.

Druhým historickým milníkem, po porážce Kasparova systémem Deep Blue, byl příchod AlphaZero od DeepMind (David Silver, Thomas Hubert, Julian Schrittwieser a kolektiv). AlphaZero byla novinkou v šachovém programování, neboť místo klasické sčítací heuristiky integruje neuronovou síť a prohledává stavový prostor hry Monte-Carlo Tree Search algoritmem. Článek zevrubně popisuje funkci programu a předkládá výsledky 100 her proti Stockfish 8, v době vydání článku 9 měsíců staré verze aktuálního vládce žebříčků herní síly. Výsledkem bylo drtivé 64-36 ve prospěch neuronové sítě. Netřeba zdůrazňovat, jak obrovské nadšení to vyvolalo. Přes veškerou kritiku herních podmínek (nevhodná časová kontrola a špatné nastavení Stockfishe) bylo nezvratně prokázáno, že neuronové sítě už v dnešní době mohou být použitelným nástrojem pro tvorbu systémů hrajících šachy. AlphaZero však nadále zůstala interním projektem nepřístupným široké veřejnosti a pouze 10 her z historického zápasu bylo publikováno. Vydání způsobilo zvýšené nasazení autorů Stockfishe, který v roce 2018 překonal veškerou konkurenci rozdílem třídy, a vznik zcela nového open-source projektu Leela Chess Zero (lc0), do kterého mnoho autorů Stockfishe také přispívá. Leela je aplikací principů AlphaZero.

Lc0 nemá finanční ani hardwarové zázemí, které Google poskytuje kolektivu DeepMind, nicméně se velmi rychle ujal a díky nadšení, které jeho příchod vyvolal, zafungovala podobná strategie, kterou už roky využívá Stockfish pro interní testování nových verzí – uživatelé a dobrovolníci poskytují výpočetní výkon svých vlastních počítačů pro učení neuronových sítí a jejich následnou aplikaci v lc0. Po roce vývoje tak na začátku roku 2019 lze lc0 považovat za přímého konkurenta Komoda a Houdiniho o pozici 2. nejsilnějšího enginu na světě, 1. místo nadále patří enginu Stockfish, konkrétně *development* (vývojové) verzi budoucího Stockfish 11.

V prosinci 2018 byl publikován druhý článek od DeepMind, přinášející nové detaily o jejich systému AlphaZero a nový, lépe připravený zápas se Stockfishem. V zápase o 1000 hrách proti několika verzím Stockfishe (jak nové development verzi, tak Stockfish 8) zvítězila AlphaZero 155 her a prohrála 6. Tab. 4 zobrazuje současný stav šachové scény dle uznávaného žebříčku CCRL 40/4 (*Computer Chess Rating Lists*).

Tab. 4: Stav žebříčku CCRL 40/4 (k 12.5.2019) [4]

Pořadí	Název	Rating
1	Stockfish 10	3546
2	Houdini 6.03	3519
3	Komodo 11.2	3502
4	Lc0 0.21.1 JH.T6.532	3486
5	Fire 7.1	3425
6	Komodo 12.3 MCTS	3409
7	Xiphos 0.5	3400
8	Ethereal 11.25	3388

Rating (hodnocení herní síly, detailněji rozebíráno v kapitole 5) šachových enginů neodpovídá hodnotami elo ratingu lidských hráčů, nicméně od éry Rybky lze šachové programy považovat za silnější i než nejsilnější velmistři a v dnešní době si málokdo dokáže představit analýzu partie bez přispění výpočtu některého z nich, zvlášť když ten nejsilnější, Stockfish, je zdarma ke stažení. Úroveň domácí přípravy především top velmistrů se tak rapidně zvýšila a narostla i kvalita obrany, ve které enginy objevily mnoho nových, silných přístupů. Negativní stránkou je však nárůst počtu remíz, především krátkých partií sehraných přesně dle domácí přípravy obou hráčů. Enginy tak stále způsobují kontroverzi, většina hráčů je však využívá. [5, 6]

2.2 Turnaje a tituly

K šachovým systémům patří také testování herní síly a kompetice o prestižní tituly nejlepšího programu na světě. Testování herní síly se bude detailněji věnovat kapitola 5 této práce.

Po dlouhou dobu organizovala nejprestižnější turnaje a zápasy o titul mistra světa počítačového šachu (*World Computer Chess Champion*) organizace ICCA (*International Computer Chess Association*) založená roku 1977, přejmenována na ICGA (*International Computer Games Association*) v roce 2002. Pod jejich vedením získaly své tituly legendární programy jako Fritz (1995), Shredder (1999, 2003, 2009), Junior (2002, 2004, 2006, 2011, 2013) či Hiarcs (2008).

Veškeré tituly mezi lety 2007 a 2010 však původně patřily programu Rybka. Velká kontroverze v roce 2010 Rybku o tyto tituly připravila a autorovi byla odepřena další účast v soutěžích. To však mělo vliv nejen na Vasika Rajlicha, který na dlouhé roky od šachového programování upustil (až do roku 2015, kdy se ujal dalšího vývoje vlajkového enginu společnosti Chessbase – Fritz), ale také na reputaci samotného *World Computer Chess Championship* (WCCC). Vyloučení Rybky je dodnes považováno za nevhodné řešení problému a vyvolalo bojkot od autorů mnoha šachových programů, mimo jiné v obavě z podobného incidentu. Zastaralý formát WCCC (účastníci používají vlastní hardware a turnaj neprobíhá jak je v dnešní době zvykem na jednotném silném

hardwaru, velmi malý počet her v soutěži atp.) a poškozená reputace ICGA tak způsobily snížení zájmu o WCCC, čehož využil Martin Thorensen založením nového turnaje TCEC (původně *Thorensen's Chess Engine Competition*, po převzetí společností Chessdom v roce 2015 *Top Chess Engine Competition*), který rychle získal mnoho příznivců a dnes je považován za nejprestižnější turnaj, jehož vítěz se těší stejné reputaci jako původní mistři světa ICGA před rokem 2010.

První ročník TCEC se odehrál na konci roku 2010 a nově nastupující engine Houdini v něm porazil Rybku a odstartoval tak nadvládu Velké trojky, která trvala až do konce roku 2017. S příchodem lc0 a odskokem Stockfishe na rozdíl třídy, sbírajíce všechny dostupné tituly, lze éru Velké trojky považovat za ukončenou, a i když někteří mluví o rozšíření na Velkou čtyřku o lc0, dá se víceméně říci, že současnost je pod nadvládou Stockfishe. V Tab. 5 jsou uvedeni vítězové jednotlivých sezón TCEC.

Tab. 5: Vítězové turnajů TCEC [7]

Sezóna	Ukončení	Vítěz	Vyzyvatel
1	úno. 2011	Houdini 1.5a	Rybka 4
2	dub. 2011	Houdini 1.5a	Rybka 4.1
3	kvě. 2011	-	-
4	pro. 2013	Houdini 3	Stockfish 250413
5	kvě. 2013	Komodo 1142	Stockfish 191113
6	pro. 2014	Stockfish 170514	Komodo 7x
7	lis. 2014	Komodo 1333	Stockfish 141214
8	pro. 2015	Komodo 9.3x	Stockfish 021115
9	pro. 2016	Stockfish 8	Houdini 5
10	pro. 2017	Houdini 6.03	Komodo 1970.00
11	dub. 2018	Stockfish 260318	Houdini 6.03
12	čvc. 2018	Stockfish 180614	Komodo 12.1.1
13	lis. 2018	Stockfish 18102108	Komodo 2155.00
14	úno. 2019	Stockfish 190203	LCZero v20.2-32930

WCCC nebylo ukončeno, a přestože se Stockfish, Houdini a další silné programy neúčastní (ať už kvůli bojkotu ICGA či finanční náročnosti způsobené nutností fyzické účasti operátora, který provádí tahy za engine na fyzické šachovnici), je nadále pořádáno. Tituly z let 2016, 2017 a 2018 drží Komodo, které se jako jediné z Velké trojky soutěže účastní. Turnaj velmi trpí malým množstvím odehraných her, což způsobuje velký vliv náhody na výsledky turnaje (i nejlepší engine může jednou za čas prohrát se slabším protivníkem, kterého jinak ve většině případů porazí), přesto Komodo bez vážných problémů vítězí a v turnajích nemá většího soka. V roce 2019 se bude WCCC konat v Macau v Číně.

V květnu 2019 se odehrál TCEC Cup 3, který je speciálním formátem turnaje předcházejícím velkému superfinále sezóny TCEC. V tomto vyřazovacím poháru lc0 přesvědčivě postoupila až do finálového zápasu proti programu Stockfish a zvítězila

se skóre 5,5-4,5. Nárůst herní síly lc0 je rychlý a spekuluje se o nástupu nové éry počítačového šachu v režii neuronových sítí. Superfinále TCEC Season 15 se odehraje mezi stejnými protivníky, oba programy proti sobě tradičně odehrají 100 her. Vítězství lc0 by mohlo znamenat překonání současného mistra a po dlouhé době změnu na první pozici rating listů. [7, 8]

2.3 Korespondenční šach

Přestože využívání šachových programů, databází atp. není v klasické hře dvou lidských hráčů běžně povoleno, existuje disciplína, ve které to možné je – korespondenční šachy.

Historicky se korespondenční šachy hrály poštou, každý z hráčů měl dostatek času na analýzu pozice doma a odeslání své odpovědi soupeři. S příchodem moderních technologií však poštovní lístky nahradila komunikace přes počítače. A s užíváním počítačů a rostoucí herní silou šachových programů vznikl obrovský problém, jak udržet hru čistou od podvodníků využívajících programy místo svých vlastních schopností k získání výhody. Nakonec však *International Correspondence Chess Federation* (ICCF) rozhodla, že kontrola možná není, a místo zákazů a chytání provinilců použití počítačů povolila. To z korespondenčního šachu udělalo zcela jinou disciplínu vyžadující místo hrubé kalkulace, kterou počítače hravě zvládnou, nutnost koordinovat poziční a strategickou stránku hry spolu s korektním používáním dostupných prostředků. Tato revoluce z přelomu tisíciletí stála korespondenční šachy nejen velkou část hráčské základny, která se odmítla změně přizpůsobit a raději hru opustila, ale také reputaci v očích velké části šachového světa, kterou jen pomalu získává zpět.

Ztráta části hráčské základny byla postupem času vykompenzována přílivem nových hráčů, kteří nacházejí zálibu v novém formátu. Co v mnoha případech hráče přitahuje je možnost sehrát co nejlepší šachovou hru za využití všech dostupných prostředků moderní doby. Nemají totiž pravdu ti, kdo tvrdí, že korespondenční šach je dnes pouze doménou programů a lidé už nemají výsledné partii co nabídnout. Počítač sám bez vedení člověka ve většině případů neobstojí. Lidský faktor se projevuje především ve výběru zahájení, strategických rozhodnutích ve střední hře a vedení v koncovkách.

Zahájení jsou po mnoha letech velmi dobře zanalyzována a často v nich není třeba mrhat výpočetním časem. Správnou volbu zahájení tak provádí člověk, velmi často na míru proti konkrétnímu soupeři a po důkladném studiu jeho odehraných her a analýze jeho silných a slabých stránek. Ve střední hře už přichází ke slovu výpočetní výkon, ale nelze ho používat bezhlavě, ale spíše jako ověření bezpečnosti vlastních myšlenek a plánů. Šachové programy jsou výbornými nástroji při výpočtu možných variant pokračování hry, ale často nehodnotí dobře uzavřené pozice, nebo vhodnost ponechání jednoho typu figury na úkor jiného pro pozdější fáze hry, do kterých se program nebyl schopen dopočítat. Koncovky jsou nejslabší stránkou šachových programů, a ty tak často hodnotí zcela nesprávně i pozice, které jsou člověku zcela zřejmé. Velkým problémem jsou např. pozice střelců nestejné barvy, které jsou velmi blízko remíze i při výhodě pěšce

či dvou na straně některého z hráčů kvůli nemožnosti podpory pěšců přes pole opačné barvy, nebo třeba věžové koncovky.

Organizace ICCF hráčům uděluje mistrovské tituly po vzoru klasického šachu. Lze získat tituly CCE (*Correspondence Chess Expert*), CCM (*Correspondence Chess Master*), IM (*International Master*), SIM (*Senior International Master*) a GM (*Grandmaster*), každý se svými specifickými požadavky. Každoročně také startuje cyklus *World Correspondence Chess Championship* o titul *World Correspondence Chess Champion*, tedy mistra světa. Tyto cykly díky povaze korespondenčního šachu trvají velmi dlouho a úřadujícím mistrem z cyklu, který začal v roce 2015 a byl dokončen v roce 2018, je Aleksandr Surenovich Dronov z Ruska. [9]

3 KLASICKÉ METODY

Klasickými jsou nazývány algoritmy využívané v klasických šachových programech jako je Stockfish, Komodo, nebo Houdini. V této kapitole budou rozebrány především *brute-force search* a *alpha-beta pruning*, neboť byly aplikovány v programu Beast prezentovaném v pozdější části této práce.

3.1 Reprezentace šachovnice

Reprezentací šachovnice a figur reálně začíná tvorba každého šachového programu či systému a je nutná pro uchování veškerých potřebných informací o pozici pro následné ohodnocení, tvorbu stromové struktury a hledání nejlepších možných tahů. Reprezentace také zahrnuje generování možných tahů a tvorbu pozic následujících, které tvoří jednotlivé uzly grafu. Rozlišujeme strukturu zaměřenou na figury, pole a struktury hybridní.

Figurově zaměřená struktura udržuje v paměti několik podstruktur (např. *list*, nebo *array*), které zachycují pozice jednotlivých figur na šachovnici. Velmi populárním typem je *bitboard*, který udržuje několik datových typů *word* o délce 64 bitů, z nichž každý vyjadřuje přítomnost daného typu figury na jednotlivých polích šachovnice. Výhodou je možnost pracovat s informací o všech polích současně a využívat bitové operace, které mohou urychlit generování nových pozic a také jejich ohodnocování. Nevýhodou je pomalá odpověď na dotaz, která figura se nachází na daném poli a obecně horší vizualizace stavu šachovnice. Nutnost použití 64bitového *word* pro každou figuru či typ figury také dělá problematickým využití 32bitové (a méně bitové) architektury starších CPU, kde je nutno informace dělit do dvou a více *word*, což dělá veškeré operace složitější a pomalejší.

Struktura zaměřená na pole má přesně opačný přístup, drží v paměti podstrukturu obsahující informace o jednotlivých polích, zda je na nich figura a pokud ano, jaký typ. Nejpopulárnějším typem je *mailbox*, který je v podstatě *array* kódů jednotlivých figur a znaků pro prázdná pole. Výhodou je velmi jednoduchá vizualizace šachovnice, rychlé odpovědi na typ figury nacházející se na daném poli, a tedy i rychlé a jednoduché změny v pozici samotné. Nevýhodou je však nutnost pracovat se složitějšími datovými typy, neboť figury již nelze kódovat jedním bitem, jako tomu bylo u figurově zaměřené struktury. To v mnoha případech prodlužuje dobu provedení algoritmů, nicméně při současném dostupném výpočetním výkonu a při rozumné optimalizaci jsou rychlostní a časové ztráty při výpočtu zanedbatelné.

Výhody obou přístupů se snaží kombinovat hybridní řešení, která na úkor paměti využívají jak strukturu 8x8 šachovnice, tak podstruktury s informací o jednotlivých figurách. To zrychluje operace nutné provádět přes pole (nastavení pozice, informace o figuře na daném poli atp.), tak operace jednodušeji proveditelné přes figury samotné bez nutnosti skenovat šachovnici pro jejich umístění (generování možných tahů, kontrola

legálnosti tahu atp.). V podstatě všechny v současnosti vyvíjené šachové programy využívají hybridní přístup.

3.2 Ohodnocení

Kritickou součástí, která především rozhoduje o herní síle výsledného programu, je heuristika neboli ohodnocení dané pozice. V ideálním případě by se program propočítal k výsledku hry, který by potom stačilo ohodnotit např. hodnotami z množiny $\{1, 0, -1\}$ podle toho, jestli hráč na tahu, pro kterého je kalkulace prováděna, v daném koncovém uzlu vítězí, remizuje, nebo prohrává. To však není výpočetně možné, neboť počet pozic, které je nutné prozkoumat, narůstá s hloubkou propočtu rychleji než exponenciálně.

Proto je nutné vytvořit sofistikovanou heuristiku, která pomůže odhadnout výsledek hry, pokud by pokračovala z daného ohodnocovaného uzlu. Stejně jako lidé učící se hrát šachy, i klasická sčítací heuristika začíná prostým součtem hodnoty figur na šachovnici. Každá figura má svou hodnotu, běžně udávanou v počtu pěšců. Pěšec samotný má tedy hodnotu 1, jezdec a střelec např. hodnotu 3, věž hodnotu 5, dáma hodnotu 9. Základní hodnota krále se do sčítání nezapočítává, neboť bez něj hra nemůže pokračovat a končí. Dále se započítávají poziční faktory, kterých je u kvalitních programů velmi mnoho. Je nutné zohlednit např. umístění figur (jezdec na okraji šachovnice nemá takové pole působnosti jako v jejím středu, a proto je za něj penalizace, mnoho figur kolem soupeřova krále naopak dává možnost útoku a matu, tedy je ohodnoceno kladně), kontrola centra šachovnice, bezpečí krále, typ figur vzhledem k soupeřovým (koncovka střelců opačné barvy bez dalších figur je obecně velmi blízko remíze, a to i v situacích, kdy jeden z hráčů má pěšce či dokonce dva navíc) a další.

Standardním výstupem ohodnocení každého uzlu je celé číslo, udávající hodnotu pozice v *centipawns* (cp, setiny pěšce), ale šachová prostředí (GUI) běžně tuto informaci předávají uživateli v jednotkách pěšců. Praxí je také udávání hodnoty pozice ne z hlediska hráče na tahu, ale aby nedošlo k nedorozumění vždy z pohledu bílého, přestože komunikace GUI a programu probíhá z hlediska hráče na tahu. Kladná hodnota tedy značí výhodu bílého hráče, záporná výhodu černého a 0 vyrovnanou pozici. Pro všechny špičkové programy platí, že výhodu nad 3 pěšáky promění ve svou výhru ve většině případů, a to proti jakékoliv opozici.

Každý heuristikou ohodnocený uzel je přidán do stromové struktury prohledávacími algoritmy, které potom vyberou nejvhodnější pokračování hry pro hráče na tahu. Příklad základní sčítací heuristiky bude detailněji popsán později v této práci, neboť byl aplikován v prezentovaném programu Beast. [10]

3.3 Prohledávání

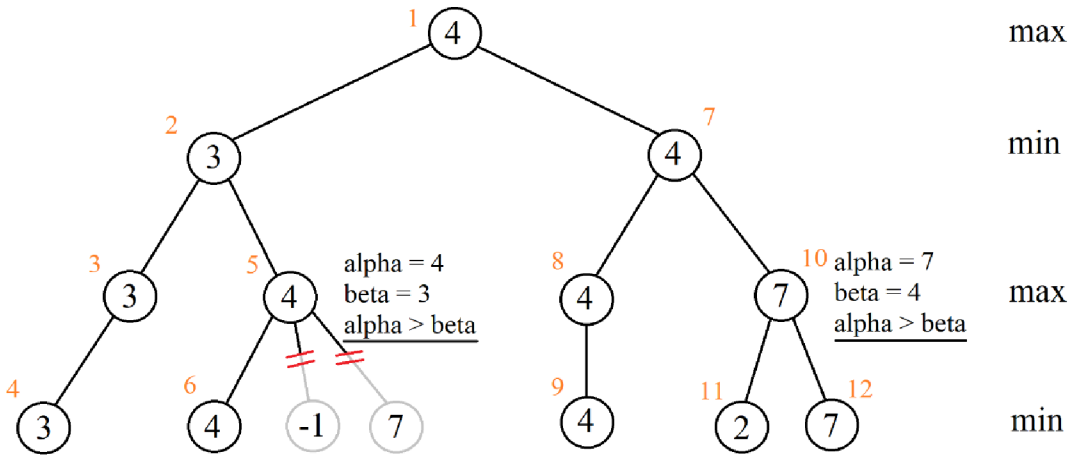
Prohledávání je spojeno s heuristikou a expanzí stromu stavového prostoru hry, rozšiřuje strom stavového prostoru hry a vybírá z jeho ohodnocených uzlů nejlepší pokračování hry pro hráče na tahu. Teoretické podklady k prohledávání položil Claude Shannon, který definoval Typ A a Typ B.

Kvůli nedostatečnému výpočetnímu výkonu byl v začátcích šachového programování využíván především Typ B, často také nazýván selektivním prohledáváním (*selective search*). Ten spočíval v prohledávání pouze uzlů, které byly heuristikou vybrány jako slibné. Ty ostatní byly vyřazeny za účelem zefektivnění a urychlení výpočtu. Nevýhodou tohoto přístupu však byla velká pravděpodobnost vyřazení z výpočtu varianty, která mohla být lepší než varianta nakonec zvolená, a to jak možnosti nalezení výhry, tak minutí soupeřova silného tahu, který později může vést k prohře.

S pokrokem v oblasti hardwaru se však výpočetní možnosti navyšovaly a Deep Blue, pokořitel mistra světa Kasparova, už využíval algoritmus typu A, jinak známý jako *brute-force search*. Ten spočívá jednoduše v prohledávání všech možných pokračování hry a výběru nejlepšího možného z nich. K úspěchu *brute-force* však přispělo především spojení s *alpha-beta pruning* algoritmem a iterativním prohledáváním. [10]

3.4 Alpha-beta pruning

Alpha-beta pruning je evolucí prohledávacího algoritmu *minimax*, umožňující při prohledávání stromu stavového prostoru ořezat některé jeho větve, které nadále nebudou rozšiřovány a prohledávány, bez nebezpečí, že mine nejlepší možnou variantu. K tomuto účelu je využíváno horní (*beta*) a dolní (*alpha*) meze. Stejně jako minimax je postaven na prohledávání do hloubky a pokračuje, dokud nedosáhne uzlu, který nelze nadále rozšiřovat, nebo maximální povolené hloubky. V šachovém programování se používá výhradně ve formě iterativního prohlubování a po každé iteraci prohloubení se předává GUI informace a výsledcích výpočtu.



Obr. 1: Příklad alpha-beta pruning

V Obr. 1 lze poukázat na dvě charakteristiky alpha-beta pruningu – možnost ořezání částí stromu, které nemohou přinést lepší výsledek, a důležitost pořadí rozšiřování. Oranžové číslování uzlů znázorňuje pořadí jejich prohledání.

Schopnost ořezání znázorňuje levá část stromu. Algoritmus se nejprve přes uzly 1, 2 a 3 dostane do listového uzlu 4, ze kterého získá hodnotu 3. Ta se stává horní hranicí beta, pro uzel 5. Jako dolní hranice alpha do něj vstupuje nejprve hodnota 4 po prohledání jeho prvního následníka, uzlu 6. Následně se spíná beta ořezání, neboť hodnota meze alpha právě přesáhla hodnotu meze beta. Další následníky uzlu 5 nemá cenu prohledávat, neboť maximalizace v tomto uzlu vždy vybere hodnotu minimálně 4, kterou nalezlo v uzlu 6. Nicméně to také znamená, že minimalizace v uzlu 2 nevybere žádnou hodnotu, kterou by uzel 5 mohl nalézt, neboť bude mít k dispozici vždy nižší hodnotu z uzlu 3 s hodnotou 3. Jedna hodnota je tedy schopna zastavit celé prohledávání podgrafu za uzlem 5, který by mohl být velmi rozsáhlý a zabral by jinak dlouhý čas prohledat celý např. algoritmem minimax.

Velmi podobná situace nastává při prohledávání pravé části grafu v uzlu 10. Zde je opět aktivována podmínka beta ořezání díky hodnotě z uzlu 12, nicméně díky pořadí uzlů není aktivována ihned, ale nejdříve je prohledán uzel 11, který podmínku neaktivoval. V případě opačného pořadí prohledávání by algoritmus začal s prohledáním nejdříve uzlu 12, kde by byla aktivována podmínka beta ořezání a prohledání uzlu 11 by nebylo nutné, čímž by se ušetřilo potenciálně mnoho výpočetního času. V alpha-beta pruning algoritmu tedy záleží na pořadí prohledání uzlů, které může celý výpočet zkrátit. V nejhorším případě však vždy bude alpha-beta stejně rychlá jako její předchůdce minimax. [10]

Pro lepší představu následuje pseudokód 1 zachycující alpha-beta pruning ve variantě *negamax*, která je implementována v praktické části této práce.

Pseudokód 1 – alpha-beta pruning

```
def alphabeta( node, alpha, beta, depthleft ):
    # koncový uzel
    if( depthleft == 0 ):
        return quiescence( node, alpha, beta )

    # další expanze a prohledávání
    for ( všichni následníci ):
        score = -alphabeta( node, -beta, -alpha, depthleft-1 )

        # ořezání
        if( score >= beta ):
            return beta
        if( score > alpha ):
            alpha = score

    return alpha
```

3.5 Quiescence search

Nepříjemnou vlastností heuristiky je zavádějící hodnocení například v situacích, kdy některý z hráčů právě sebral „materiál“ (v praxi často používaný termín pro figury) svému protivníkovi. V naprosté většině případů není tento materiál opravdu získán, ale soupeř v dalším tahu dobírá materiál zpět. To však program nevidí, protože při iterativním prohlubování ukončil prohledávání uprostřed dané kombinace. Tedy např. hodnocení +3 po dobrání soupeřova jezdce, které plyne z dočasné výhody extra figury na šachovnici, vyjadřuje podstatu pozice zcela špatně, protože v dalším tahu po dobrání jezdce se hodnota vrátí zpět na 0. Obr. 2 zachycuje toto nebezpečí.

Obr. 2: Průběh braní chráněné figury

Příklad z Obr. 2 ukazuje průběh braní chráněné figury. Pokud by prohledávací algoritmus ukončil prohledávání po 4.Sxc6, hodnotil by pozici kladně pro bílého, neboť má oproti svému soupeři navíc střelce, který má hodnotu 3 pěšce. Bezesporu by tedy tuto variantu vybral jako nejlepší možnou pro bílého hráče, neboť takto velká výhoda by jistě vedla k výhře. Co už by však nevzal v potaz je, že bílý střelec je okamžitě dobrán tahem 4…bxc6 a materiál se opět vyrovná. V mnoha případech by tak algoritmus preferoval variantu, která ve skutečnosti výhodu nepřináší, oproti variantě, která by ji přinést mohla, jen ne tak velkou jako zdánlivá výhoda zisku figury.

Tento problém řeší *quiescence search*. Jde o navazující algoritmus na alpha-beta pruning, se kterým má mnoho společného a přebírá od něj hodnoty alpha a beta pro ořezávání neužitečných variant. Pro každý listový uzel prohledávání alpha-beta je nadále prohledáván každý tah, který je braním. Při dostatečné rychlosti prohledávání je toto kritérium rozšiřováno o šachy a další tahy, které autor považuje za nutné. Výsledné hodnocení navrácené z quiescence search je potom přiřazeno listovému uzlu a předáno zpět hlavnímu prohledávacímu algoritmu. Odhaduje se, že 50 až 90 % uzlů je prohledáno právě ve quiescence search. Následuje pseudokód, který tento přístup zachycuje.

Pseudokód 2 – quiescence search

```
def quiescence( node, alpha, beta ):
    # ohodnocení uzlu
    stand_pat = heuristics( node )

    # ořezání
    if( stand_pat >= beta ):
        return beta
    if( alpha < stand_pat ):
        alpha = stand_pat

    # další expanze a prohledávání
    for( new in všechny tahy braní a šachů ):
        score = -Quiesce( new, -beta, -alpha )

        # ořezání
        if( score >= beta ):
            return beta
        if( score > alpha ):
           alpha = score

    return alpha
```

Quiescence search je také často rozšiřován o další podmínky ořezávání za účelem zkrátit výpočet variant, které nepřinesou lepší výsledek, jako např. *delta pruning*. [10]

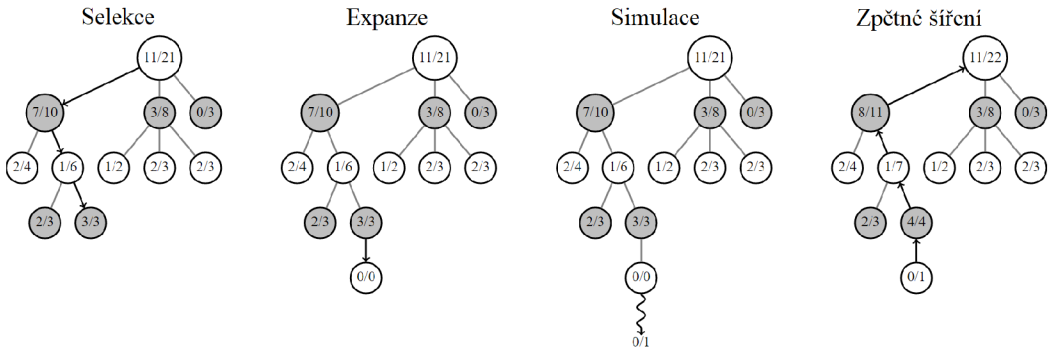
3.6 Monte-Carlo Tree Search

Moderním přístupem, který se objevuje na šachové scéně je Monte-Carlo Tree Search (MCTS). Na rozdíl od alpha-beta pruning se jedná o verzi prohledávání do hloubky založeným na náhodném prohledávání stavového prostoru hry a nepotřebuje heuristiku. Za využití výsledků z prohledávání vytváří v paměti strom hry a s rostoucím množstvím prohledaných uzlů a odehraných her se odhad ohodnocení jednotlivých herních pozic stává přesnějším.

Prohledávání spočívá v sérii simulovaných her z daného startovního uzlu. MCTS operuje ve 4 krocích – selekce, expanze, simulace a zpětné šíření. Tyto kroky se opakují, dokud není výpočet ukončen např. časovým limitem.

Selekce spočívá ve výběru koncového uzlu, který bude nadále expandován. Tento uzel je volen na základě toho, jak často byl v minulosti navštíven, jaké má současné ohodnocení atp. Expanze tento uzel expanduje a jeden (nebo i více, záleží na autorovi) jeho potomek je podroben simulaci, která spočívá v náhodném dohrání hry do konce a ohodnocení daného potomka hodnotou z $\{0; 0{,}5; 1\}$ podle výsledku náhodně odehrané hry. Zpětné šíření informuje předchůdce právě ohodnoceného uzlu o výsledku dané hry a navýší jejich skóre o tuto hodnotu. Zároveň je také navýšen počet her odehraných v dané

linii uzlů vedoucích k expandovanému. Každý uzel tak získává pravděpodobnostní ohodnocení, které vyjadřuje odhadovaný výsledek z něj pokračované partie. Obr. 3 zobrazuje schéma MCTS na konkrétním příkladu. [11]



Obr. 3: Schéma MCTS (upraveno dle [12])

3.7 Databáze koncovek

Důležitým prvkem moderních programů jsou tzv. databáze koncovek (*endgame tablebases*). Pro pozice koncovek s méně než určitým malým počtem figur (běžně méně než 6) lze těchto databází využít jako náhradu heuristiky – obsahují totiž informaci o hodnocení pozice – zda je vyhraná, prohraná, či remíza. Tato informace byla získána propočtem všech možných pokračování hry a program k ní přistupuje čtením souborů databáze z disku. Výhodou je, že není nutné pokračovat v prohledávání následníků pozice, která byla ohodnocena databází koncovek, neboť tato informace je spolehlivá a pokud se do této pozice program dostane, potom vždy s pomocí databáze dosáhne výsledku, který mu byl předán. Nevýhodou je nutnost využití dalšího místa na disku pro databáze, které není zanedbatelné, jak bude ukázáno v Tab. 6. Stejně tak velmi záleží na rychlosti čtení, které je disk schopen dosáhnout. Pro databáze 6 a více kamenů už je nutné použít velmi rychlých SSD disků, protože běžné HDD disky velmi zpomalují výpočet při nutnosti nahlížení do databází.

Základními typy databází koncovek jsou *Syzygy* a *Nalimov*. Syzygy jsou dnes standardem. Až do nedávna jediné s informací o 7 kamenných koncovkách byly speciální *Lomonosov tablebases*, které ale nejsou využívány klasickými programy, ani volně dostupné. Až v roce 2018 byla podpora 7 kamenných koncovek přidána také pro Syzygy.

Nalimov tablebases nejsou nejstaršími vytvořenými databázemi, ale jednalo se o první ze standardů moderní doby. Vytvořil je v roce 1998 Eugene Nalimov a jejich nespornou výhodou, která je dělá v některých situacích stále relevantními, je informace o vzdálenosti k matu v pozicích, které jsou pro jednu ze stran vítězné. Tato informace není podstatná z hlediska výsledku hry, ale v situacích, kdy si hráč přeje hru ukončit co nejdříve, jako například v korespondenčním šachu, je velmi výhodná. Avšak použité kódování a také obsah této informace znamená, že zabírají mnohem více prostoru na disku než jejich nástupce Syzygy.

Prvními pokusy o náhradu Nalimov byly databáze *Scorpio* (2005) a *Gaviota* (2008), které svého času využívány byly, ale nikdy se příliš nerozšířily. To se změnilo až s příchodem Syzygy od autora Ronald de Man v roce 2013. Syzygy neobsahují informaci o vzdálenosti k matu, nicméně berou v potaz nově zavedené pravidlo 50 tahů, které jejich předchůdci v době jejich vzniku nemuseli řešit. Jejich zásadní výhodou jsou také menší požadavky na místo na disku. V roce 2018 byl dokončen vývoj kódu na generování informace o 7 kamenných koncovkách, které jsou však i přes značnou kompaktnost těchto databází stále nedostupné běžným uživatelům kvůli potřebě neskutečných 17 TB. Nicméně například server *lichess* si na svých serverech pro ně místo udělal a jsou tak dostupné široké veřejnosti zdarma.

Lomonosov vznikly na Moskevské univerzitě v roce 2012 a autory jsou Vladimir Makhnychev a Victor Zakharov. K jejich vytvoření bylo použito superpočítače Lomonosov. Do roku 2018 byly jedinými existujícími databázemi pro 7 figur a jsou dostupné pouze online přes produkty společnosti ChessOK. [10]

Tab. 6: Databáze koncovek [13]

Databáze	**Rok vydání**	**5 figur**	**6 figur**	**7 figur**
Nalimov	1998	7.1 GB	1.2 TB	-
Syzygy	2013/2018	939 MB	150.2 GB	17 TB
Lomonosov	2012	-	-	140 TB

4 POZNÁMKY K NEURONOVÝM SÍTÍM

Umělé neuronové sítě jsou velmi populárními nástroji pro modelování a řešení složitých problémů. Excelují především v analýze obrazu, nebo regresních a klasifikačních úlohách různého typu. Skládají se z mnoha jednoduchých výpočetních jednotek (umělých neuronů) propojených do složitějších síťových struktur. Tyto struktury lze učit změnou vnitřních parametrů, které zajišťují přizpůsobení sítě konkrétní aplikaci a zbavují nutnosti ruční tvorby rozhodovacích pravidel a heuristik. Jejich hlavními výhodami jsou schopnost učení, nelinearita, robustnost, schopnost zpracování zašuměných informací a tolerance chyby.

Umělý neuron napodobuje funkci biologického neuronu, ale stále je pouze jednoduchým matematickým modelem, přičemž nedosahuje úrovně složitosti, kterou mají opravdové biologické neuronové buňky. Nejjednodušším modelem biologického neuronu je tzv. *perceptron*. Jednotlivé perceptrony jsou v neuronových sítích navzájem propojeny ve vrstvách, které tvoří tzv. *neuronovou síť*. Perceptron provádí váženou sumaci vstupů, které porovnává s vlastní prahovou hodnotou. Výsledek porovnání je argumentem aktivační funkce definované např. vztahem (1). Uvedený vztah vyjadřuje matematickou operaci uvnitř perceptronu pro n vstupů, kde y je výstupem z perceptronu, x_i jsou vstupy, w_i jsou váhami jednotlivých spojení a b je prahovou hodnotou. V okamžiku překročení prahové hodnoty dochází k tzv. aktivaci neuronu.

$$y = \begin{cases} 1, \ if \ \sum_{i=1}^{n} w_i x_i \geq b \\ 0, \ if \ \sum_{i=1}^{n} w_i x_i < b \end{cases} \tag{1}$$

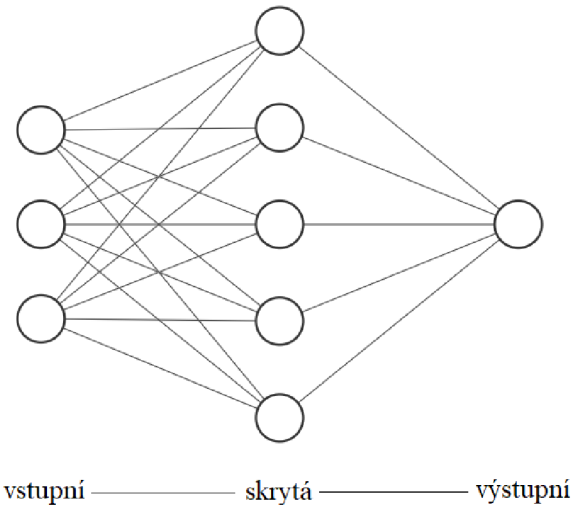
V tomto vztahu hodnota 1 představuje aktivaci neuronu, naopak 0 klidový stav. Dle hodnoty vah dochází k zesílení či zeslabení signálu, který vstupuje do perceptronu. [14]

4.1 Typy umělých neuronových sítí

Neuronových sítí rozlišujeme mnoho typů. Základními jsou dopředné neuronové sítě, které staví na základech prvních neuronových sítí složených z perceptronů. Rekurentní sítě nadále rozšiřují ty dopředné o cyklické struktury uvnitř grafu sítě. Konvoluční neuronové sítě, používané především ke zpracování obrazu, staví na zpracování vstupu konvolučními filtry, které berou v potaz vzájemnou polohu pixelů. Pro účely této práce jsou důležité především dopředné a konvoluční sítě, protože byly následně aplikovány v praktické části.

4.1.1 Dopředné neuronové sítě

Tzv. *dopředné sítě* (*feed-forward neural network, FFNN*) patří v současnosti k frekventovaně používaným. Jejich velkou výhodou je relativní jednoduchost struktury i vlastní implementace. Z teoretického hlediska je dostatečné užití pouze třívrstvé sítě neuronů, přičemž. tzv. skryté vrstvy mají volně definovaný počet neuronů. Vstupní a výstupní vrstva obsahuje počet neuronů odpovídajících definici úlohy. Obr. 4 zobrazuje jako příklad třívrstvou, resp. dvouvrstvou dopřednou neuronovou síť. [14]



Obr. 4: Třívrstvá dopředná neuronová síť, resp. dvouvrstvá síť perceptronů

Učení probíhá s učitelem, řízeně. Příprava dat k učení tedy zahrnuje nejen přípravu vstupních dat, ale také výstupních dat, které ukazují síti, jakých výsledků má při daných vstupech dosáhnout. Učení probíhá v jednotlivých epochách a v každé z nich jsou upravovány hodnoty vah spojení. V každé epoše jsou síti předána všechna dostupná data, ale v tzv. *batch*, dávkách o volitelném počtu vstupních dat. Toho se využívá pro menší paměťové zatížení při učení sítě.

Na rozdíl od původních perceptronových sítí, moderní dopředné sítě nepředávají pouze 1, nebo 0 podle toho, zda byl neuron aktivován nebo ne, ale prakticky jakékoliv reálně číslo. Z toho důvodu je vhodné využívat normalizace hodnot. Pro učení vícevrstvých sítí se používá zobecněné *delta rule* pro perceptron, které upravuje váhy spojení i pro skryté vrstvy neuronů. Nejprve se určí aktuální chyba sítě $E = f(x,d)$, kde x je hodnotou vstupu, d požadovaným výstupem, y reálným výstupem, w aktuální konfigurací vah, p počtem vzorů, a Y množinou neuronů výstupní vrstvy. Výsledná chyba E je potom sumou chyb jednotlivých vzorů a vyjádřena vztahy (2).

$$E(w) = \sum_{k=1}^{p} E_k(w)$$
$$E_k(w) = \frac{1}{2} \sum_{j \in Y} (y_j - d_j)^2 \quad (2)$$

Zobecněné pravidlo pro perceptrony se používá minimalizaci chyby a adaptaci vah, které popisují vztahy (3), kde w^t je novou vahou spojení, $w^{(t-1)}$ vahou původní a Δw^t změnou váhy.

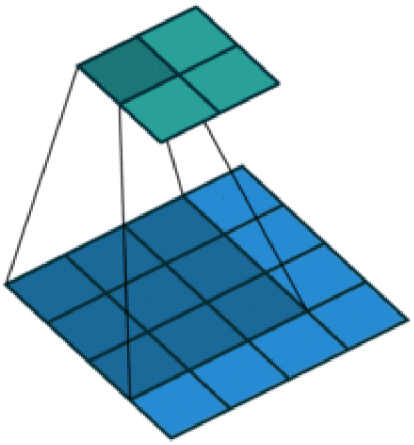
$$w^t = w^{(t-1)} - \Delta w^t$$
$$\Delta w^t = \frac{\delta E(w)}{\delta w} \quad (3)$$

Pro svou univerzálnost jsou dopředné sítě využívány pro predikci, klasifikaci, modelování a další. [15]

4.1.2 Konvoluční neuronové sítě

Konvolučními sítěmi (*Convolutional Neural Networks, CNN*) jsou nazývány takové dopředné sítě, které ve své struktuře využívají konvolučních filtrů. Konvoluce je nejvíce využívána při zpracování obrazu a využité filtry jsou tedy 2D, nicméně lze pro specifické aplikace využít i konvolučních filtrů 1D (např. klasifikace spekter z *Laser-Induced Breakdown Spectroscopy*), nebo více dimenzionálních.

Základním principem je aplikace většího množství konvolučních filtrů určené velikosti (např. 2D filtry 3x3 pixely) na vstupní obrazová data. Po aplikaci získáváme nová data, jejichž počet se rovná počtu použitých filtrů, ale která jsou menší velikosti než data původní. V Obr. 5 je znázorněno schéma aplikace konvolučního filtru. Výsledná hodnota na dané pozici je součtem násobků mezi hodnotou konvolučního filtru a pixelu vstupních dat na odpovídající pozici dle vztahu (4), kde $g(x, y)$ je filtrovaným obrazem, $f(x, y)$ původním obrazem a $h(x, y)$ filtrem.



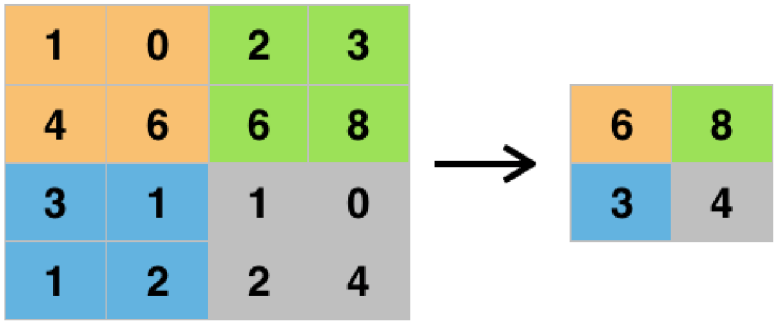
Obr. 5: Schéma aplikace konvolučního filtru [16]

$$g(x,y) = f(x,y) \cdot h(x,y) \tag{4}$$

Speciálním případem je diskrétní konvoluce, ve které je operováno pouze s celými čísly. Obecný vzorec (4) je potom možno rozšířit jako (5), kde *k* je rozměrem obrazu.

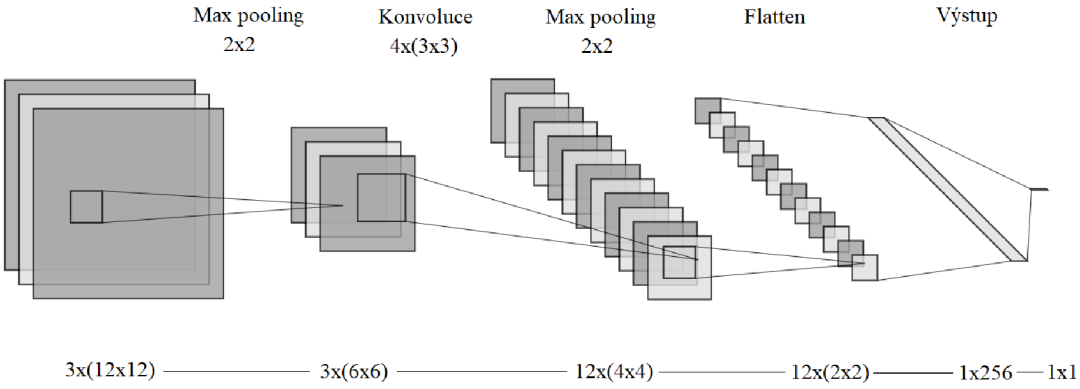
$$g(x,y) = f(x,y) \cdot h(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j) \cdot h(i,j) \tag{5}$$

Učení sítě probíhá úpravou hodnot jednotlivých aplikovaných filtrů a filtrované obrazy vytvořené po jejich aplikaci zachycují významné charakteristiky analyzovaného obrazu. S úspěchem je také v kombinaci s konvolučními filtry využíván tzv. *pooling*, metoda redukce dimenze zpracovávaných dat. Při velikosti jádra 2x2 je dimenze snížena na polovinu. Metoda spočívá v kombinaci několika pixelů vstupního obrazu do jednoho, který dále tuto skupinu reprezentuje. Používá se především *max pooling*, který jako reprezentanta vybírá nejvyšší možnou hodnotu z redukované skupiny pixelů, a *average pooling*, který reprezentuje prostým aritmetickým průměrem hodnot. Obr. 6 znázorňuje příklad *max pooling*.



Obr. 6: *Max Pooling* (upraveno dle [17])

Při aplikaci konvolučních neuronových sítí se využívá obou metod, které se navzájem střídají. Finálním krokem je potom převod do 1D vektoru (*flatten*), který je buď rovnou výstupem ze sítě, nebo může být vstupem i do několika vrstev dopředné neuronové sítě, která na konvoluční část navazuje. Obr. 7 znázorňuje schéma konvoluční sítě aplikované na vstupní data s max pooling (jádro 2x2), 4 konvolučními filtry (jádro 3x3), znovu max pooling (jádro 2x2), flatten, vrstvou 256 neuronů skryté vrstvy a jedním výstupním neuronem. Popisky v horní části znázorňují jednotlivé provedené operace, ve spodní části potom aktuální tvar dat procházejících sítí. [18]



Obr. 7: Schéma konvoluční neuronové sítě

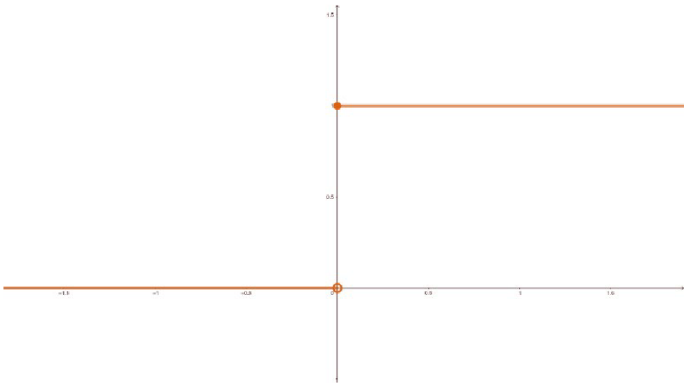
4.2 Aktivační funkce

Aktivační funkce jsou nedílnou součástí neuronových sítí a hodnota jejího výstupu je předána následujícím neuronům. Běžně se dělí na lineární a nelineární aktivační funkce.

Ostrá unipolární nelinearita

Ostrá unipolární nelinearita (*unit step*) je nejjednodušší aktivační funkcí a má schopnost předávat pouze diskrétní hodnoty 1 a 0. Výstup se řídí dle již popsaného vztahu (1), kde

hodnota prahu *b* je nulová, a znázorněn na Obr. 8. Této funkce bylo využíváno především u perceptronu (Pitts, McCulloch 1943). [19]

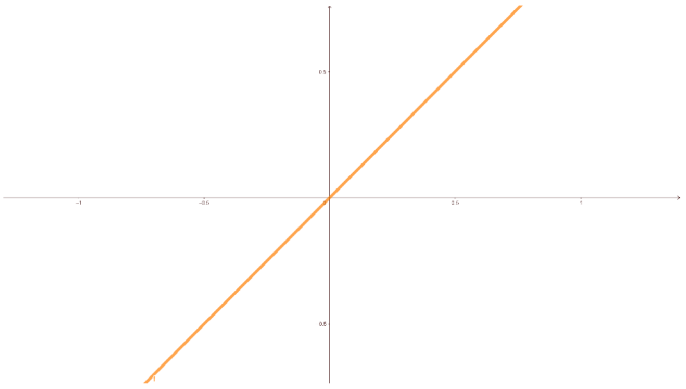


Obr. 8: Jednotkový skok

Lineární funkce

Výstupní hodnota aktivovaného neuronu je přímo úměrná jeho vstupu, řídí se dle vztahu (6) a znázorněna na Obr. 9. Použití lineární aktivační funkce není vhodné pro komplexní data, nebo data s mnoha parametry. Použita byla už v *ADALINE* (*ADAptive LInear Element*, Widrow a Hoff, 1960). [19]

$$f(x) = a \cdot x \tag{6}$$

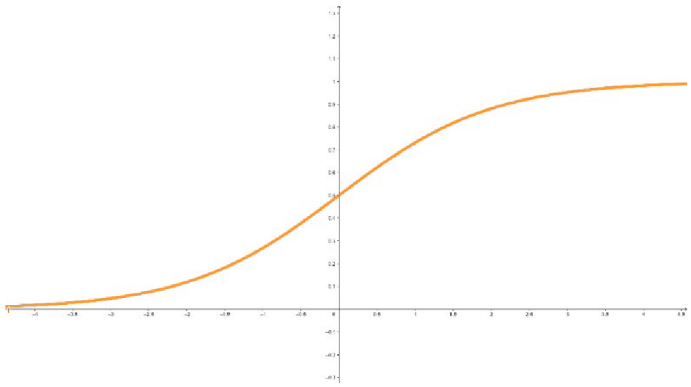


Obr. 9: Lineární funkce

Sigmoida

Sigmoida je základní používanou aktivační funkcí neuronových sítí. Její hlavní výhoda oproti unit stepu je v diferencovatelnosti při zachování podobné nelineární charakteristiky. Její rozsah je pouze v kladných hodnotách $\langle 0,1 \rangle$, který se především využívá pro výstup z neuronové sítě ve smyslu pravděpodobnosti. Sigmoida je popsána vztahem (7) a vykreslena v Obr. 10. [19]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

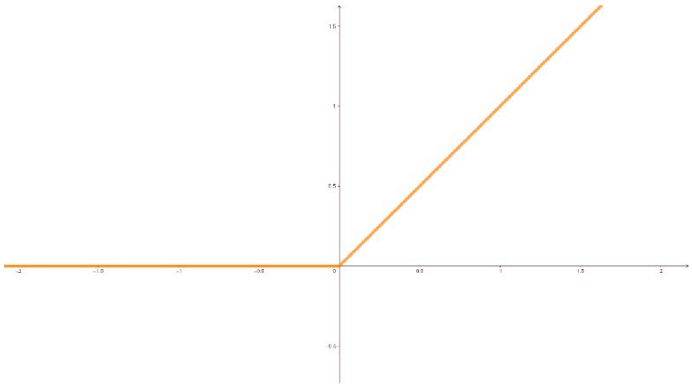


Obr. 10: Sigmoida

Rektifikovaná lineární jednotka

V kontextu tzv. hlubokých neuronových sítí (*Deep Neural Networks, DNN*) jde v současnosti o nejrozšířenější variantu aktivační funkce. Rektifikovaná lineární jednotka (ReLU) je využívána v CNN a je zobrazena v Obr. 11 a vyjádřena vztahem (8).

$$f(x) = \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases} \tag{8}$$



Obr. 11: ReLU

ReLU se využívá jak v základní variantě, tak jako tzv. *Noisy ReLU*, *Leaky ReLU*, nebo *ELU*. Noisy ReLU je rozšířena o Gaussovský šum a s úspěchem byla implementována např. v *Restricted Boltzman Machine*. Leaky ReLU se liší od své klasické varianty pro $x < 0$, kde má místo konstantního rostoucí charakter s nízkým gradientem.

Exponenciální lineární jednotka ELU se opět liší pouze pro záporné x, kde je vývoj nahrazen exponenciálou $\propto \cdot (e^x - 1)$, kde $\propto$ je hyper parametrem nutným ladit.

Výhodou aktivačních funkcí ReLU oproti lineárním je eliminace velmi nízkých hodnot na výstupu z funkce, které často nejsou žádoucí. V některých případech je však úplná eliminace nevhodná, a proto je využito některé varianty ReLU, která tento problém řeší. [19]

Softmax

Pro řešení klasifikačních úloh se jako aktivační funkce výstupní vrstvy využívá *softmax*. Výstupem funkce je pravděpodobnost příslušnosti vzoru do dané třídy, v čemž se velmi podobá sigmoidě, která se nicméně hodí pouze pro binární klasifikaci. Softmax je definován dle (9), kde k značí počet tříd a i právě hodnocenou třídu. [19]

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}}, \qquad i = 1,2,3,\dots,k \tag{9}$$

4.3 AlphaZero

Obrovský zájem o neuronové sítě jako nástroj pro hru šachy způsobily články kolektivu DeepMind společnosti Google vydané v prosinci 2017 a 2018. Tyto představily nový systém AlphaZero, který nahradil specifickou heuristiku hlubokými neuronovými sítěmi s obecným učícím a prohledávacím algoritmem. AlphaZero je vylepšeným a zobecněným systémem AlphaGo, který DeepMind v minulosti připravil pro hru Go, ve které klasické prohledávací programy nenašly velkého úspěchu a lidští hráči stále měli převahu.

AlphaZero je interním systémem nepřístupným veřejnosti, proto vznikla jeho open-source adaptace Leela Chess Zero, která využívá stejných technik popsaných v článcích DeepMind.

Bez jakékoliv specifické znalosti, kromě pravidel hry, se systém učí sériemi zápasů sám proti sobě. Neuronová síť přijímá šachovnici jako vstup a výstupem je vektor pravděpodobností každého tahu (*policy*) a hodnocení pozice. V kombinaci s natrénovanou sítí AlphaZero využívá MCTS algoritmu společně s policy vektorem pro zúžení prohledávaného stavového prostoru pouze na tahy s větší hodnotou pravděpodobnosti.

Neuronová síť se skládá ze vstupní vrstvy, těla a dvou výstupních hlav. Vstupem do sítě je *stack* (označení nemá nic společného se zásobníkem, v tomto kontextu bude používáno pro označení vstupních vrstev do neuronové sítě) 19x19x17, který se skládá ze 17 matic reprezentujících vrstvy šachovnice. 8 a 8 je dedikováno pozicím bílých a černých figur spolu s historií hry pro 7 pozic předcházejících tahů za účelem posílení informace o vývoji hry. Poslední vrstva je tvořena maticí obsahující samé 1 nebo 0 podle toho, jestli je na tahu bílý, nebo černý. Tělo sítě tvoří jedna rektifikovaná normalizovaná konvoluční vrstva a 19 residuálních bloků. Každý blok je tvořen dvěma rektifikovanými

normalizovanými konvolučními vrstvami a vynechaným spojením. Každá konvoluční vrstva aplikuje 256 filtrů s jádrem velikosti 3x3. Policy hlava přidává další rektifikovanou normalizovanou konvoluční vrstvu a zakončuje 73 konvolučními filtry. Hlava ohodnocení aplikuje jednu rektifikovanou normalizovanou konvoluční vrstvu s jedním filtrem velikosti 1x1 následovanou skrytou vrstvou 256 neuronů s aktivací ReLU a jedním neuronem s aktivací hyperbolickým tangens.

Učení probíhalo v 700 000 krocích s velikostí batch 4096 a začínalo s parametry nastavenými náhodně. Využito bylo 5 000 TPU první generace pro generování her a 64 TPU druhé generace pro učení sítě. TPU (*Tensor Processing Unit*) je specializovaný hardware vyvinutý společností Google pro strojové učení.

V článku z roku 2017 byla AlphaZero postavena proti tehdejšímu vedoucímu programu v rating listech – Stockfish. Odehráno bylo 100 her, ve kterých AlphaZero zvítězila s 28 výhrami, 72 remízami a žádnou prohrou. Tento výsledek byl však velmi diskutován kvůli podmínkám zápasu. Použitá verze Stockfishe byla už více než půl roku stará, přestože k dispozici byly novější. Hardwarové podmínky také vzbuzovaly pochyby, mnoho expertů odhadovalo velkou výpočetní výhodu AlphaZero, přestože vykazovala nižší počet prohledaných uzlů. Velikost paměti RAM pro *transposition tables* (tabulky transpozic) Stockfishe byl jen 1 GB, což je při vysoké rychlosti výpočtu na 64 jádrovém výkonném procesoru nedostatečné a může to způsobit problémy s uchováním výsledků předchozích výpočtů. Časová kontrola byla 1 min na každý tah, což zcela eliminuje potřebu rozložení času, zkrácení výpočtu tam kde není třeba a jeho prodloužení v kritických pozicích. Ale i přes tyto pochyby a kritiky byl výsledek fenomenální a prokázal nové možnosti pro vývoj šachových systémů.

Plný článek z roku 2018 napravuje nedostatky při testování. Nová verze AlphaZero byla nasazena v 1 000 hrách proti různým verzím Stockfishe (Stockfish 8, nový Stockfish 9 a z něj odvozený Brainfish s knihovnou Cerebellum), zvítězila ve 155 hrách a prohrála jen 6. Zbylé hry skončily remízou. Hardwarové podmínky kopírovaly konfiguraci TCEC (Season 9). [20]

AlphaZero se neúčastní žádného nezávislého testování ani významných turnajů jako TCEC a není dostupná veřejnosti. Zda se to časem změní není jasné.

5 TESTOVÁNÍ HERNÍ SÍLY

Herní síla je obecně vyjadřována hodnocením Elo, nebo novějším Glicko a Glicko 2. Elo je nadále používáno organizací FIDE pro výpočty hodnocení hráčů šachu a většina národních organizací se drží stejného systému. Je přepočítáváno po delších časových intervalech od jednoho měsíce až po jeden rok. Glicko je používáno především herními servery online, kde je hodnocení přepočítáváno v reálném čase hned po ukončení hry.

5.1 Elo

Prvotní systém Elo byl vytvořen v polovině 20. století profesorem fyziky Arpadem Elo jako systém hodnocení výkonosti hráčů šachu a řídí se vztahem (10), kde R_{new} a R_{old} jsou nové a staré hodnocení, K je násobící faktor (FIDE používá 20), W je výsledek hry a W_e je očekávaný výsledek hry, vypočítaný z R_A a R_B, což jsou hodnocení soupeřů. [21]

$$\begin{aligned} R_{new} &= R_{old} + K(W - W_e) \\ W_e &= \frac{1}{1 + 10^{(R_B - R_A)/400}} \end{aligned} \tag{10}$$

5.2 Glicko

Glicko je novější systém autora Marka Glickmana. Novinkou je zavedení *rating deviation* (RD), které spolu s rozdílem hodnocení soupeřů určuje, o kolik bodů se každému hráči hodnocení změní. Vyšší RD znamená vyšší změnu a funguje jako ukazatel nejistoty hodnocení. RD narůstá v průběhu času, pokud hráč nehraje. Glicko je vypočítáváno ve 3 krocích.

Prvním krokem je aktualizace RD dle formule (11), kde RD je nová nejistota a RD_0 stará, c je volená konstanta (běžně 20-40, ale lze volit libovolně) a t je počet period uplynulých od poslední hry. Tato perioda může být libovolně dlouhá, od minut až po měsíce. 350 je doporučená hodnota počáteční nejistoty nových hráčů a je horním limitem velikosti nejistoty.

$$RD = \min\left\{\sqrt{RD_0 + c^2 t}, 350\right\} \tag{11}$$

Druhým krokem je aktualizace samotného hodnocení dle formulí (12) po odehrání m her, kde r je nové hodnocení, r_0 staré hodnocení, r_i je hodnocení jednotlivých soupeřů a s_i je výsledek jednotlivých her (1 je výhra, 0,5 remíza a 0 prohra). [22]

$$r = r_0 + \frac{q}{\frac{1}{RD^2} + \frac{1}{d^2}} \cdot \sum_{i=1}^{m} g(RD_i)\left(s_i - E(s|r, r_i, RD_i)\right), kde$$

$$g(RD_i) = \frac{1}{\sqrt{1 + \frac{3q^2(RD_i^2)}{\pi^2}}}$$

$$E(s|r, r_i, RD_i) = \frac{1}{1 + 10^{\left(\frac{g(RD_i)(r - r_i)}{-400}\right)}} \tag{12}$$

$$d^2 = \frac{1}{q^2 \sum_{i=1}^{m} \left(g(RD_i)\right)^2 E(s|r, r_i, RD_i)(1 - E(s|r, r_i, RD_i)}$$

$$q = \frac{\ln(10)}{400}$$

Třetí krok reprezentovaný vzorcem (13) je opět aktualizace RD. V kroku jedna šlo o aktualizaci zvyšující RD, tentokrát naopak je po odehraných hrách RD pro aktivitu sníženo. [22]

$$RD = \sqrt{\left(\frac{1}{RD_0^2} + \frac{1}{d^2}\right)^{-1}} \tag{13}$$

5.3 Rating listy

Výkonost šachových programů zachycují rating listy, žebříčky seřazené dle hodnocení herní síly. Na rozdíl od žebříčků pro lidské hráče je možné, aby programy hrály každý s každým a často se toho využívá. Pro sestavení žebříčku se tedy sehraje velký turnaj mezi programy za daných podmínek a po skončení turnaje se vypočítá hodnocení a určí pořadí dle herní síly.

Vlastní žebříček si může vytvořit kdokoliv, nicméně pro jeho relevantnost je nutné dodržet několik základních pravidel – stejný hardware pro všechny účastníky, rozumná časová kontrola, korektní nastavení jednotlivých programů a dostatečné množství odehraných her jsou jen některými z nich. Dostatečné množství odehraných her není přesně definováno, ale běžně za dostatečné považuje kolem 1000 odehraných partií každým programem, což vytváří velký časový problém. Ten je často řešen krátkou časovou kontrolou pro hrající programy a silnějším hardwarem pro rychlejší výpočty.

Většina rating listů tak prokazuje herní sílu programu v krátkých partiích. Nejznámějším je *Computer Chess Rating List* (CCRL), který na velmi silném hardwaru vytváří žebříček v časových kontrolách 40 tahů za 40 min (40/40) a za 4 min (40/4).

Stejně jako ostatním žebříčkům mu po dlouhou dobu vládne *Stockfish*. Výňatek v Tab. 4 kapitoly 2.1.

Vlastní žebříčky si však udržují i turnaje. TCEC, považován za neoficiální mistrovství světa, kvůli velmi dlouhým časovým kontrolám za účelem co nejkvalitnějších partií nemá obrovské množství podkladových her. Žebříček však, i přes větší předpokládanou odchylku od reálné herní síly, vypovídá o schopnosti programů hrát kvalitnější šachy, pokud k tomu dostanou dostatek času. Jeho část je uvedena v Tab. 7.

Tab. 7: Stav TCEC rating listu (k 5.5.2019) [23]

Pořadí	**Engine**	**Rating**
1	Stockfish	3592
2	LCZero	3590
3	Houdini	3549
4	Komodo	3542
5	Komodo MCTS	3482

6 APLIKACE – BEAST

Algoritmy a přístupy popsané v předchozí části práce byly aplikovány v šachovém programu Beast. Ten je univerzálním šachovým enginem kompatibilním s většinou dostupných šachových GUI (např. Fritz, Chessbase, ChessOK Aquarium, Hiarcs Chess Explorer, či Arena) přes UCI protokol a byl vytvořen v programovacím jazyce Python 3 za využití volně dostupné knihovny *python-chess*, *keras* a dalších. Pod *Windows* všechna GUI očekávají engine jako *executable* soubor. Tento problém byl vyřešen volně dostupnou knihovnou *auto-py-to-exe*, která převede skript do požadovaného typu souboru.

Využitými prohledávacími algoritmy jsou alpha-beta pruning ve variantě *negamax* a quiescence search s ořezáváním delta. Program využívá 3 typy heuristiky – klasickou sčítací, neuronovou síť ve dvou variantách (klasifikační a regresní) a generátor pseudonáhodných čísel pro simulaci náhodné hry. Nedílnou součástí je *time management* (schopnost hrát s časovým limitem) a podpora dalších typů příkazů k výpočtu nepodmíněných časem – prohledávání do dané hloubky, prohledání daného počtu uzlů a další.

Pro koncovky, ve kterých na šachovnici zbývá pouze málo figur, byla přidána podpora databází koncovek Syzygy, které jsou v dnešní době standardem. Díky nim jsou tyto pozice sehrány do nejlepšího možného výsledku. Beast také rozeznává pravidlo remízy 3x opakováním stejné pozice a pravidlo 50 tahů – pokud po 50 tahů není učiněn tah pěšcem, nebo neproběhlo braní, hra končí remízou. V lidských partiích musí některý z hráčů remízu požadovat, ale v hrách počítačových programů je aplikováno automaticky.

Beast je primárně *proof of concept* a je proto stále co rozšiřovat a zlepšovat, především v oblasti rychlosti výpočtu a heuristiky. Zdrojový kód je dostupný z [24].

6.1 UCI komunikační protokol

Spolu se starším komunikačním protokolem *Winboard* je *Universal Chess Interface* (UCI) základní metodou komunikace mezi výpočetním programem a *Graphical User interface* (GUI). Tato komunikace probíhá přes komunikační kanál, po kterém jsou předávány a přijímány příkazy a informace pomocí standardních vstupů a výstupů.

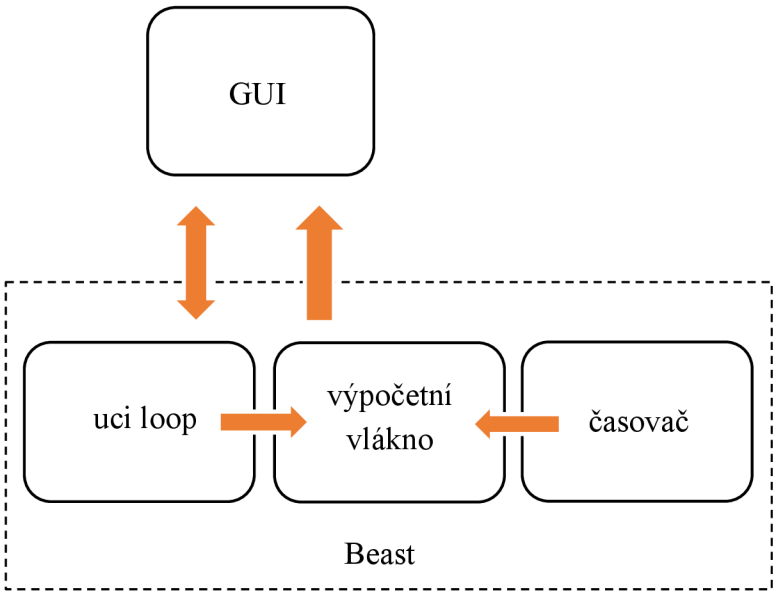
Jednotlivé zprávy jsou předávány každá zvlášť jako jeden řádek textu, který si každý účastník komunikace sám dešifruje. První slovo je vždy klíčové a následují parametry, pokud jsou potřeba. Detaily jsou dostupné z [25], zde bude uveden pouze příklad komunikace, ve kterém GUI (modré) načte engine (červený), změní jeho parametr a přikáže počítat do hloubky 2.

```
uci

id name Beast 0.09
id author M. Macurek
option name Threads type spin default 1 min 1 max 1
option name TimeFlex type spin default 10 min 0 max 1000
option name SearchAlgorithm type combo default AlphaBeta var AlphaBeta
option name Quiescence type check default True
option name SyzygyPath type string default <empty>
option name SyzygyProbeLimit type spin default 6 min 0 max 7
option name Syzygy50MoveRule type check default True
option name Heuristic type combo default Classic var Classic var
NeuralNetwork var Random
option name Network type combo default Regression var Regression var
Classification
option name ModelFile type string default <default>
uciok

setoption name Quiescence value False
isready

readyok

position startpos
go depth 2

info depth 1 score cp 29 nodes 20 nps 619 time 32 pv e2e4
info depth 2 score cp 0 nodes 234 nps 727 time 322 pv e2e4 e7e5
bestmove e2e4
```

Komunikace začíná dotazem GUI na parametry programu *uci*, na který program předá svůj název, autora a parametry, odpověď ukončuje odesláním *uciok*. Následuje příkaz vypnout *quiescence search* přes *setoption* a dotaz *isready*, zda je program připraven hrát, na který je kladnou odpovědí *readyok*. Příkaz *position startpos* znamená, že pozice pro hru je startovní pozice hry a *go* s parametrem *depth 2* požaduje výpočet do hloubky 2 půl tahů. V průběhu výpočtu engine předává aktuální informaci o výpočtu klíčovým slovem *info* s dalšími parametry o současném stavu. Výpočet je ukončen buď příkazem GUI, ukončen přes *time management*, nebo jako v tomto případě po splnění původního příkazu, tedy dopočítání se do hloubky 2. Výpočet končí předáním nejlepšího nalezeného tahu klíčovým slovem *bestmove* s tahem v UCI notaci jako parametrem, který je vyjádřen předáním startovního a finálního pole tahu. [25]

6.2 Struktura programu

Pro správnou funkci programu je nutné, aby byl ve všech okamžicích připraven přijmout příkaz od GUI a reagovat na něj. V programu tedy běží 2 až 3 samostatná vlákna. Jedno pouze čeká v tzv. *uci loop* a komunikuje s GUI, druhé provádí samotný výpočet, jehož výsledky předává komunikačním kanálem přímo GUI a třetí je aktivováno přes *time management* a jeho účelem je zastavit výpočet po uplynutí daného časového intervalu. Tato komunikace je znázorněna v Obr. 12.



Obr. 12: Schéma komunikace

Beast využívá *event*, proměnnou umožňující předání informace mezi vlákny. Výpočetní vlákno kontroluje stav této proměnné, která je nastavena na *true* po celou dobu průběhu výpočtu. Pokud je přepnuta na *false*, ať už příkazem GUI přes *uci loop*, nebo vypršením časovače, výpočet okamžitě končí a současný výsledek je odeslán GUI.

Na výpočetním vláknu běží cyklus, který čeká na příkaz. Když ho dostane, předá parametry hlavní funkci *main*, která začíná výpočet. Její struktura je znázorněna následujícím pseudokódem 3.

Pseudokód 3 – hlavní funkce main

```
def main( parametry, nastavení ):
    # inicializace proměnných
    kořen = parametry.kořen

    # time management + časovač
    start = ( čas počátku výpočtu )
    čas = ( čas na tah )
    if čas > 0:
        (spuštění časovače)

    # hlavní smyčka iterativní expanze
    while splněny podmínky:
        # prohledávání
        ohodnocení = alphabeta( kořen, hloubka, -100000, 100000 )

        print(informace o prohledávání)

    # nejlepší nalezený tah
    print(nejlepší tah)
```

Nejprve jsou převzaty parametry a nastavení od *uci loop*, následně spuštěn časovač, pokud jde o výpočet časově omezený. V iterativním prohledávání je algoritmem *alphabeta* nalezen nejlepší tah dle dané heuristiky a vypsány aktuální informace. Po ukončení iterací je předána informace o nejlepším nalezeném tahu a funkce končí. Výpočetní vlákno se vrací zpět do vyčkávacího módu, dokud nedostane další příkaz k výpočtu.

6.3 Prohledávací algoritmy

Vzhledem k výpočetní náročnosti MCTS a nízké rychlosti propočtu, které Beast dosahuje kvůli využití pythonu a python-chess knihovny, byl jako prohledávací algoritmus zvolen alpha-beta pruning, jehož teoretické základy byly popsány v kapitole 3.4. Aplikována byla varianta negamax, která přistupuje k hodnocení každého uzlu z pohledu hráče na tahu. Zásadní úpravou bylo ohodnocení koncových uzlů algoritmem quiescence a navýšení počtu vracených informací, neboť mimo hodnocení pozice a nejlepšího tahu je pro hráče šachu zajímavá také celá hlavní nalezená varianta (*principal variation*, pv), do jaké hloubky se algoritmus propočítal a kolik u toho prohledal uzlů. Všechny tyto informace jsou v průběhu prohledávání vyhodnocovány a předávány GUI pokaždé, když je dokončena iterace prohloubení. V kombinaci s časem výpočtu je tak možné získat informaci i o rychlosti prohledávání.

Quiescence search je zcela nutnou součástí šachových programů jak bylo popsáno v kapitole 3.5. Jeho základní forma byla rozšířena o delta pruning dvojího typu – plný

a částečný – které pomáhají omezit počet prohledaných uzlů. I přes toto přidané ořezávání v některých pozicích trvá velmi dlouho, než je možné prohledání dané hloubky dokončit, především se jedná o komplikované pozice ve střední hře s mnoha figurami na šachovnici a mnoha možnostmi braní.

Základní myšlenkou delta ořezání je ukončení prohledávání varianty, která má jen malou či nulovou šanci na zlepšené situace hráče na tahu. Aktuální informaci o nejlepší možné variantě udržuje dolní mez alpha, proti které je současný tah testován. Quiescence search prohledává pouze tahy, ve kterých je brán materiál. Než je současná prohledávaná pozice o tyto tahy rozšířena, aplikuje se plný delta pruning, při rozšiřování jednotlivých tahů se aplikuje částečný delta pruning. Pokud aktuální hodnota pozice vrácená heuristikou sečtená s konstantou delta nepřesahuje alpha, nemá cenu tento uzel nadále rozšiřovat, neboť je malá naděje, že by pomohl zlepšit situaci. Například pokud je situace velmi špatná a hráč na tahu prohrává o celou věž, němá cenu prohledávat variantu braní pěšce, protože to situaci velmi pravděpodobně nezachrání. Avšak braní lehké figury (jezdec, střelec) cenu má, neboť je šance, že spolu s kompenzací na pozici by to stačit mohlo. Volba konstanty delta záleží na autorovi, Beast používá pro plný pruning hodnotu dámy 1000 cp a pro částečný hodnotu 200 cp. Ideu delta pruning vyjadřuje i následující pseudokód 4.

Pseudokód 4 – delta pruning

```
def quiescence( node, alpha, beta ):
    ...

    delta = 200
    hodnota = hodnota_sebrane_figury + delta
    if ( heuristika + hodnota < alpha ):
        return alpha

    ...
```

6.4 Heuristika

V kapitole 3.2 byl popsán vliv kvalitní heuristiky na herní sílu. Pro Beasta byly vytvořeny 3 typy heuristiky – základní klasická sčítací heuristika, neuronové sítě (regresní a klasifikační úloha) a náhodná hra. Všechny tyto heuristiky využívají stejný prohledávací algoritmus a další součásti programu.

Náhodná hra je tvořena prostým přiřazením pseudonáhodného čísla každému z uzlů první hloubky prohledávací iterace a prohledávání je v této hloubce ukončeno. Algoritmus alpha-beta vybere tah, kterému bylo přiřazeno nejvyšší náhodné číslo a tento je zahrán. Zbývající dva typy jsou komplikovanější a zaslouží si vlastní podkapitoly.

6.4.1 Klasická heuristika

Klasická sčítací heuristika vychází ze součtu hodnoty figur na šachovnici. Hodnoty jednotlivých figur, které Beast využívá, jsou zachyceny v Tab. 8 a vychází z hodnot prezentovaných GM Larrym Kaufmanem. Králi žádná hodnota přiřazena není, protože bez něj hra končí. [26, 27]

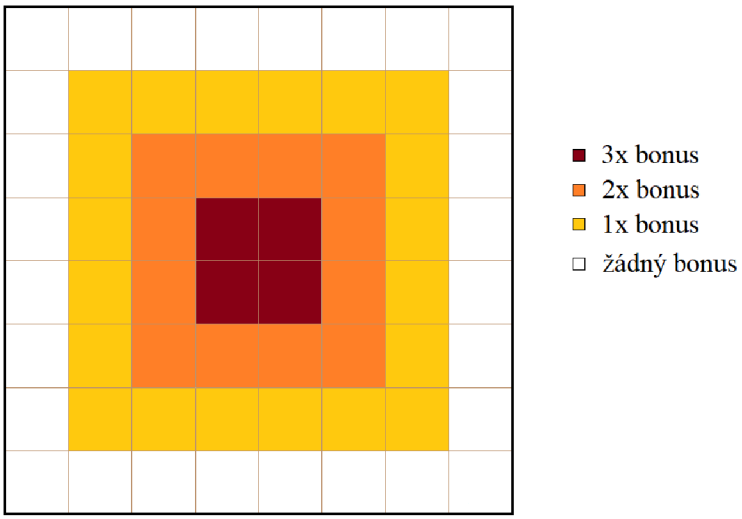
Tab. 8: Hodnoty figur na šachovnici

Figura	Hodnota [cp]
Pěšec	100
Jezdec	350
Střelec	350
Věž	525
Dáma	1000

K této hodnotě jsou připočítány poziční faktory. V Beastovi jsou aplikovány následující.

Pěšci mají 3 poziční faktory, které zohledňují jejich umístění na řadách, sloupcích a zda okupují centrální pole. V potaz je také brána vzdálenost od soupeřova krále. Za každou řadu, o kterou se pěšec posune kupředu je mu k jeho hodnotě přičteno 7 cp. Hodnota centrálních pěšců se považuje za vyšší než hodnota krajních pěšců, a proto je každému pěšci mimo dvou centrálních pěšců na sloupcích *d* a *e* odečteno 5 cp za každý sloupec vzdálenosti od těchto dvou centrálních. Cílem je především motivace dobírání, pouze při kterém pěšec mění svůj sloupec, směrem k centru, kde jsou pěšci i figury běžně užitečnější než na kraji šachovnice. Pokud pěšec stojí na některém ze 4 centrálních polí d4, e4, d5 a e5, je mu k hodnotě přičteno dalších 5 cp. To podporuje boj o centrum, základní princip, který se učí každý začínající šachista. Poslední bonus pěšec získává se vzdáleností od soupeřova krále. Za každé pole vzdálenosti v duchu Manhattanské metriky je mu hodnota modifikována o 5 cp. To podporuje využití pěšců jako útočných figur.

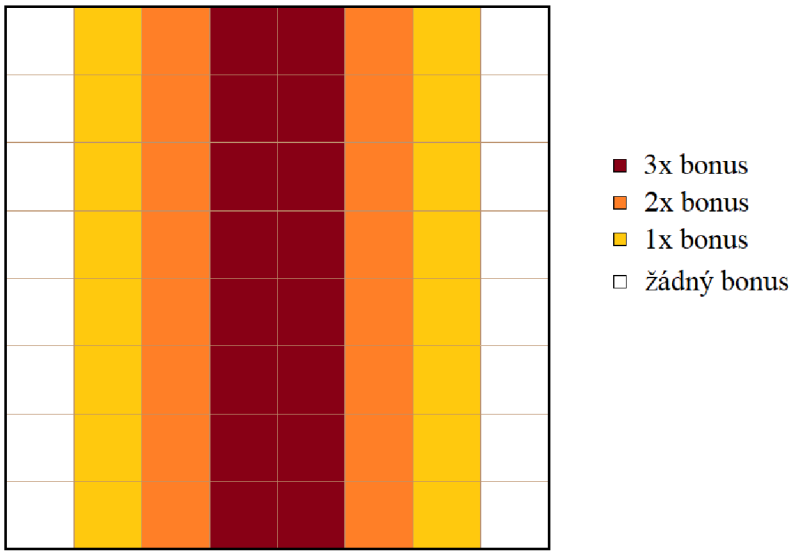
Hodnotu jezdců ovlivňují dva parametry. Opět se jedná o vzdálenost od soupeřova krále, implementována stejně jako v případě pěšce, pouze s rozdílem velikosti váhy vzdálenosti, která byla navýšena na 8 cp. Druhým parametrem je bonus centralizace. Jezdci jsou známí svou potřebou operovat kolem centra šachovnice, kde jejich síla roste. Naopak na okrajích šachovnice jsou velmi omezení a snižuje se jak počet jejich možných tahů, tak jejich užitečnost. Proto se jejich bonus s vahou 7 cp váže právě na jejich centralizaci. Vyznačené oblasti v Obr. 13 vyjadřují velikost bonusu, který jezdci dostávají.



Obr. 13: Bonus pro jezdce

Střelci dostávají opět bonus pro vzdálenost od soupeřova krále o stejné velkosti jako jezdci, 8 cp, a bonus 7 cp pokud okupují centrální pole.

Věže mají vzdálenostní bonus o velikosti 5 cp a navíc bonus 8 cp pokud se nachází na sloupcích blíže centru, podobně jako bonus pro centralizaci jezdců, jak zachycuje Obr. 14.



Obr. 14: Bonus pro věže

Dáma je velmi univerzální figurou a je užitečná především v útoku na soupeřova krále. Její bonus pro centralizaci je principem stejný jako pro jezdce, ale o velikosti pouze 2 cp. Důležitější je vzdálenostní bonus, který je 8 cp.

Přestože král nemá základní hodnotu vyjádřenou v cp kvůli jeho nepostradatelnosti, jeho poziční faktory jsou velmi důležité. Král má vzdálenostní bonus 5 cp za účelem boje proti soupeřovu králi v koncovkách a centralizační bonus o velikosti 8 cp a jezdcova typu jako v Obr. 13. Dokud je však na šachovnici soupeřova dáma, tak je bonus převrácený, aby se král pokoušel schovat mimo dosah soupeřových figur

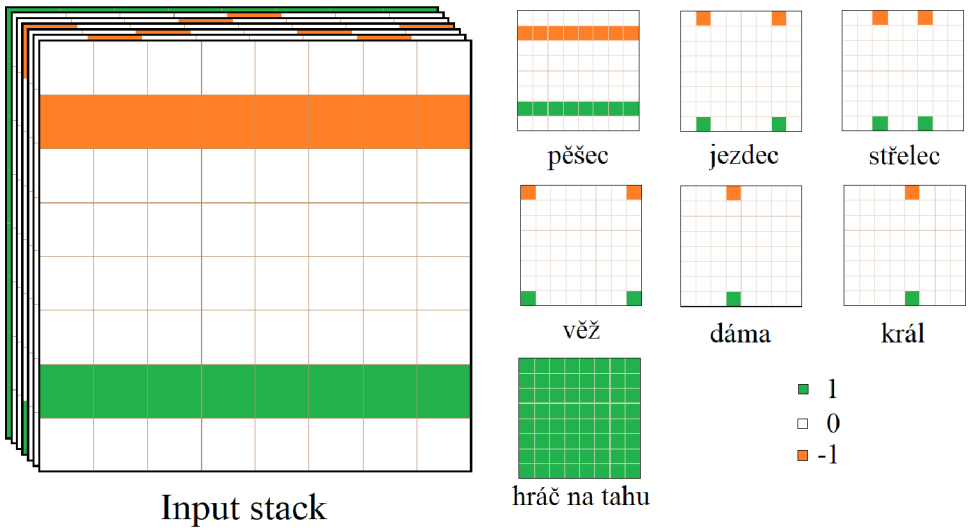
a nepokoušel se bezhlavě útočit v pozicích, kde to je nebezpečné. Jakmile se dámy vymění, bonus se opět obrátí a stane pozitivním, aby král v koncovkách kontroloval centrum, které je velmi důležité.

Výstupem heuristiky je tedy ohodnocení pozice v cp, které následně využívá prohledávací algoritmus pro nalezení co nejlepšího možného tahu. Beast implementuje pouze základní sčítací heuristiku, kterou je možno mnohonásobně vylepšit, nicméně pro prezentaci funkce je dostatečná. Herní síla bude rozebírána v kapitole 6.6.

6.4.2 Heuristika neuronovou sítí

Díky nárůstu výpočetního výkonu a prohloubení vědomostí a porozumění je v dnešní době možno jako heuristiku aplikovat neuronové sítě. Za tímto účelem byly vytvořeny 3 modely aplikující 2 základní úlohy řešené neuronovými sítěmi – regresi a klasifikaci.

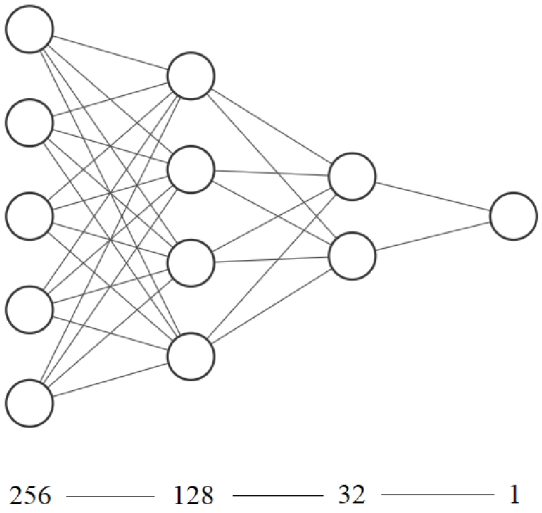
Všechny modely využívají stejný vstup do neuronové sítě, a tím je 7 matic o velikosti šachovnice 8x8. Každá z prvních 6 vrstev reprezentuje umístění jednoho typu figury (král, dáma, věž, střelec, jezdec, pěšec), kdy hodnota 1 znamená postavení bílé figury, -1 černé a 0 nepřítomnost figury daného typu figury nenachází. Poslední vrstva je ponechaná velmi důležité informaci o tom, který z hráčů je právě na tahu – pro bílého je vrstva naplněná hodnotami 1, pro černého -1. Tento *stack* vrstev, prezentovaný v Obr. 15 pro startovní pozici hry, je nadále zpracován neuronovou sítí dle jejího typu.



Obr. 15: Input stack

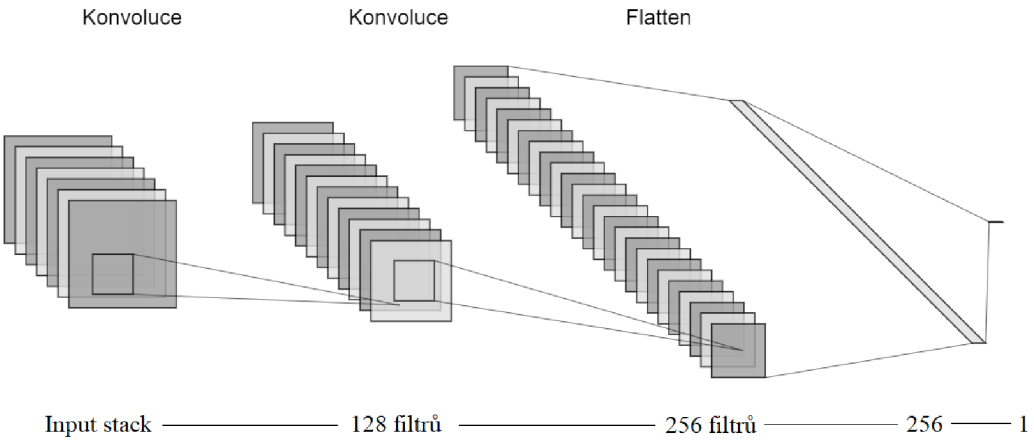
Model 1 je klasická 3 vrstvá dopředná plně propojená neuronová síť (FFNN). Jedná se o sekvenční model vytvořený v Kerasu, přijímající jednotlivé vstupy ve tvaru (7,8,8). Skryté vrstvy mají 256-128-32 neuronů. Součástí každé z nich je normalizace do intervalu hodnot $\langle -1, 1 \rangle$ předcházející ReLU aktivační funkci a 30% dropoutem (vynecháním daného množství neuronů z výpočtu a úprav vah) pro snížení pravděpodobnosti přeučení sítě. Výstup z neuronové sítě zajišťuje jediný neuron

bez aktivační funkce. Jedná se o regresní úlohu. Architekturu přibližuje schéma v Obr. 16.



Obr. 16: Schéma neuronové sítě modelu 1

Model 2 je opět regresní úlohou, ale využito bylo konvolučních neuronových sítí (CNN). Vstup tvaru (8,7,7) prochází dvěma vrstvami o 128 a 256 konvolučních filtrech s rozměry jádra 3x3, v obou vrstvách je nejprve aplikována normalizace a následně *ReLU* aktivace. Výsledné filtry jsou funkcí *Flatten* převedeny do jednoho vektoru, který je vstupem do 256 neuronů skryté vrstvy s 50% dropoutem. Výstupem je opět jeden neuron bez aktivace. Schéma je zachyceno v Obr. 17.



Obr. 17: Schéma modelu 2

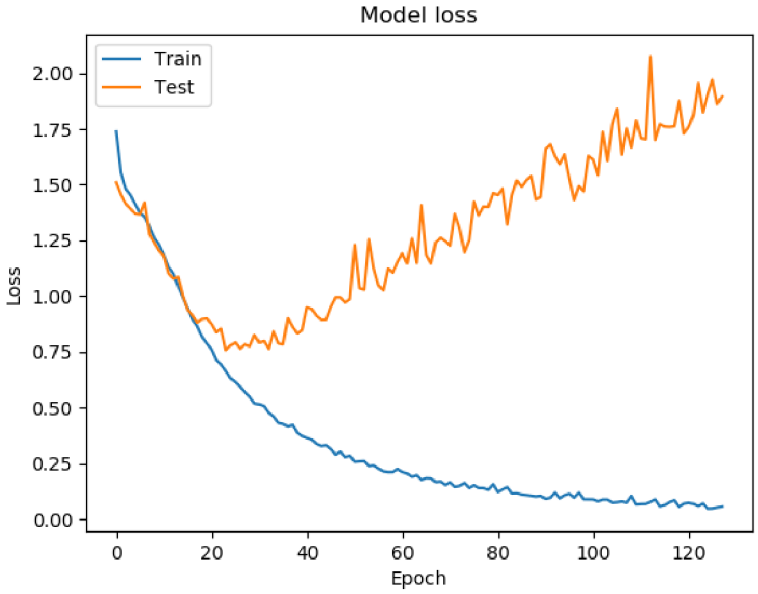
Model 3 je strukturou identický modelu 2 s jediným rozdílem, že výstupní vrstvu tvoří 7 neuronů se *softmax* aktivací. Tento model je klasifikační úlohou, jejíž cílem je zařadit vstupní šachovou pozici do jedné ze 7 kategorií – vyrovnaná pozice, bílý nebo černý má výhodu, bílý nebo černý má velkou výhodu, bílý nebo černý vítězí.

Zařazení do jednotlivých kategorií však znesnadňuje úlohu prohledávacího algoritmu, protože mnoho pozic má stejné ohodnocení, které po úpravě tvoří celé číslo z intervalu $\langle -3, 3 \rangle$. Ke každému hodnocení je tedy přičteno pseudonáhodné reálné číslo z intervalu $\langle -0{,}5; 0{,}5 \rangle$, které nezpůsobuje překrývání jednotlivých kategorií a zároveň pomáhá vybrat náhodný tah ze všech dostupných tahů se stejným ohodnocením.

6.5 Učení neuronové sítě

Všechny neuronové sítě byly učeny řízeně, s učitelem. Vstupními daty byly jednotlivé šachové pozice extrahované z 50, respektive 100 tisíc šachových partií dvojího druhu (v obou setech bylo 30 % vysoce kvalitních korespondenčních partií z databáze ICCF a 70 % partie sehrané v domácích ligách hráči všech úrovní od začátečníků po velmistry) převedených do korektního tvaru (7,8,8) jako v Obr. 15. Cílem při výběru dat bylo zajistit přítomnost jak vysoce kvalitních her, ze kterých je možno získat znalost o vyrovnaných pozicích a korektní hře, tak hry nižší kvalitativní úrovně pro identifikaci pozic nevyvážených s výhodou některého hráče. Všechny tyto pozice byly ohodnoceny programem Stockfish 10 a výstupní data byla omezena do intervalu $\langle -2000, 2000 \rangle$ cp, protože jakákoliv pozice s výhodou některého hráče přes 20 pěšců bude tímto hráčem zcela jistě vyhrána a není tedy třeba mezi nimi rozlišovat.

Dataset byl rozdělen na učící a testovací část náhodným výběrem v poměru 80:20. Z učící části datasetu byla síť učena s cílem minimalizovat ztrátovou funkci (*loss*). Testovací dataset byl použit pro testování výsledku učení a také pro zastavení procesu učení (*early stopping*) před dosažením maximálního počtu epoch, pokud by se síť začala přeučovat (*overfitting*). Přeučení sítě je patrné právě z vývoje ztrátové funkce na testovacím datasetu, jejíž hodnota by přestala nadále klesat, jako tomu je z počátku učení, ale naopak by její stagnace či nárůst signalizovaly pravděpodobnou ztrátu generalizace. Obr. 18 zachycuje tento trend.



Obr. 18: Přeučení neuronové sítě

Učení probíhalo za využití frameworku Keras, který je populární open source knihovnou pro tvorbu neuronových sítí. Keras je pouze nástavbou nad některou z knihoven strojového učení a vývoj probíhá v jazyce python. Knihovnou zvolenou pro tvorbu samotných neuronových sítí byla TensorFlow. Pro učení byly využity dva optimalizátory z knihovny Keras s následujícími parametry.

$$Adam\ (lr = 0.0001, beta\_1 = 0.9, beta\_2 = 0.999, epsilon = None, \\ decay = 0.0, amsgrad = False)$$
$$SGD\ (lr = 0.0001, momentum = 0.9, decay = 0.0, nesterov = True).$$

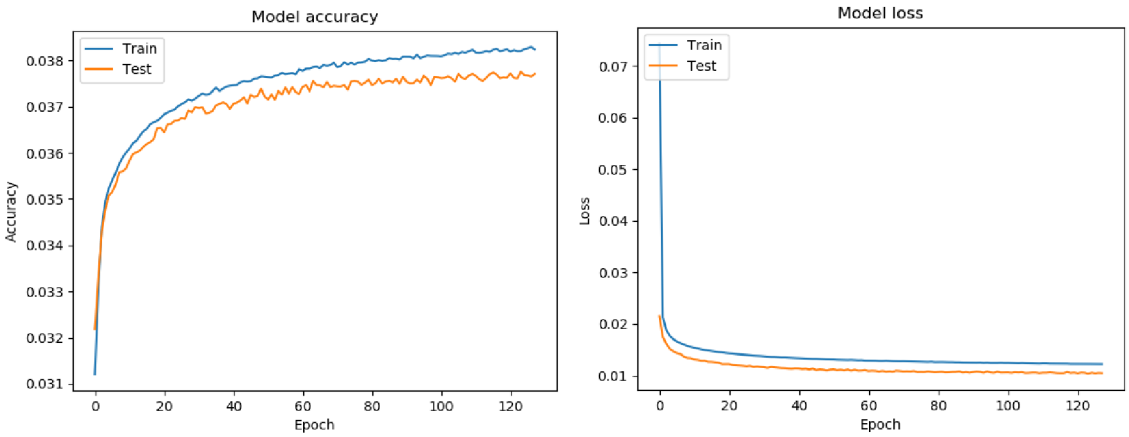
Pro oba byl snížen parametr rychlosti učení pro snížení výkyvů učící křivky a u SGD bylo navíc oproti výchozím hodnotám zvoleno momentum o velikosti 0,9 a použit Nesterov. Ztrátová funkce, podle které se učení řídí, byla pro každou využitou aplikaci odlišná. Pro regresi byl použit *mean squared error* (mse) a pro klasifikaci *categorical crossentropy*. Hlavní metrikou pro regresi potom byla právě finální dosažená hodnota mse a pro klasifikaci úspěšnost zařazení pozice do správné skupiny v procentech.

Učení je nejvhodnější provádět na grafické kartě s jádry CUDA, paralelními procesorovými jednotkami, které Keras dokáže velmi efektivně využít pro urychlení učícího procesu. Použita byla grafická karta nVidia GTX 1080Ti na velmi výkonném stroji zapůjčeném Ústavem Automatizace a Informatiky Vysokého učení technického v Brně. I přes takto silný hardware bylo potřeba mezi 10 a 20 hodinami pro naučení každé sítě. Tab. 9 zobrazuje jednotlivé naučené sítě a jejich použité parametry. Konvence pojmenování modelů je volena jako Model následovaný číslem modelu sítě dle kapitoly 6.4.2, počtem her v tisících, ze kterých byl učen, a pořadím, ve kterém byl připraven, pokud je nutné nadále rozlišovat.

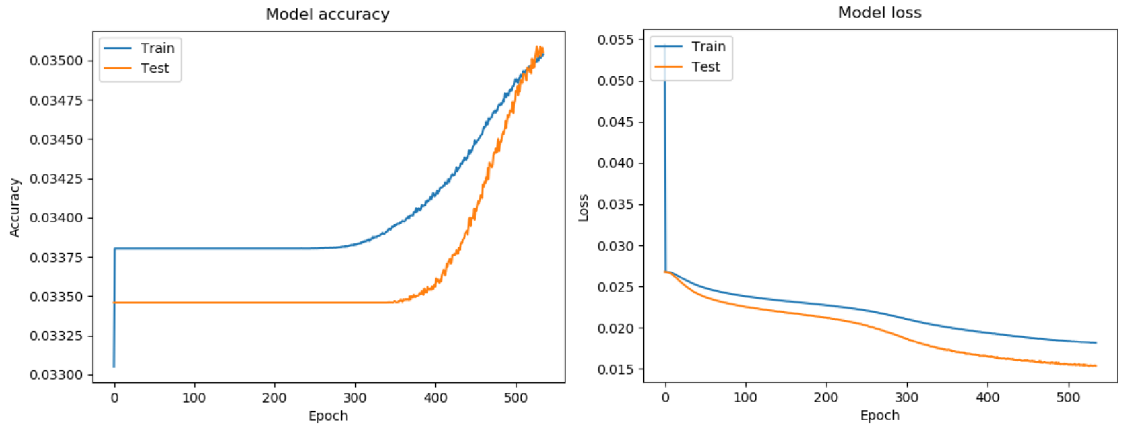
Tab. 9: Naučené modely pro heuristiku

Síť	Úloha	Počet her	Optimalizátor	Počet epoch	Metrika
Model 1-50	Regrese	50 000	Adam	128	0,01
Model 1-100-1	Regrese	100 000	SGD	535	0,015
Model 1-100-2	Regrese	100 000	Adam	128	0,014
Model 2-50	Regrese	50 000	Adam	82	0,0087
Model 2-100-1	Regrese	100 000	SGD	263	0,013
Model 2-100-2	Regrese	100 000	Adam	203	0,008
Model 3-50	Klasifikace	50 000	Adam	55	72,1 %
Model 3-100	Klasifikace	100 000	SGD	223	68,3 %

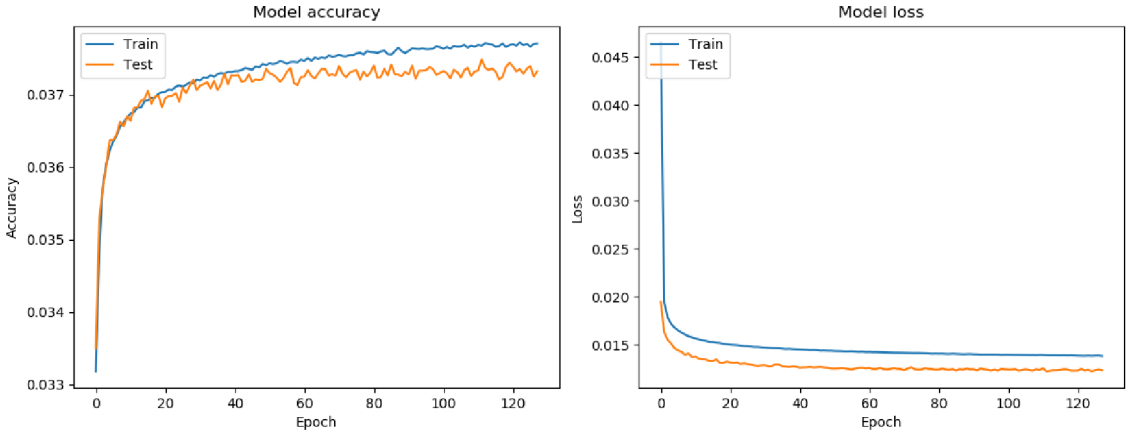
V Obr. 19-26 jsou vyneseny grafy učení jednotlivých sítí, a to obou metrik – *acc* (*accuracy*, přesnost) a *loss* (hodnota ztrátové funkce).



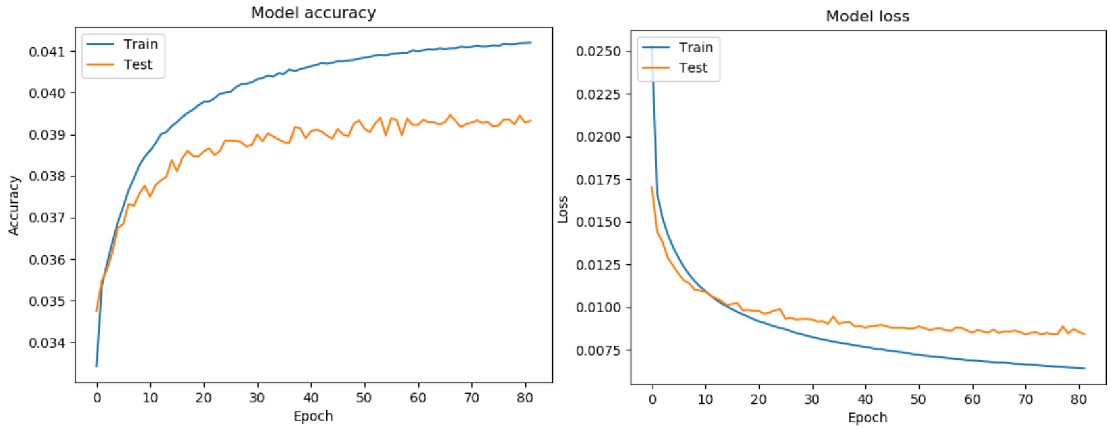
Obr. 19: Model 1-50



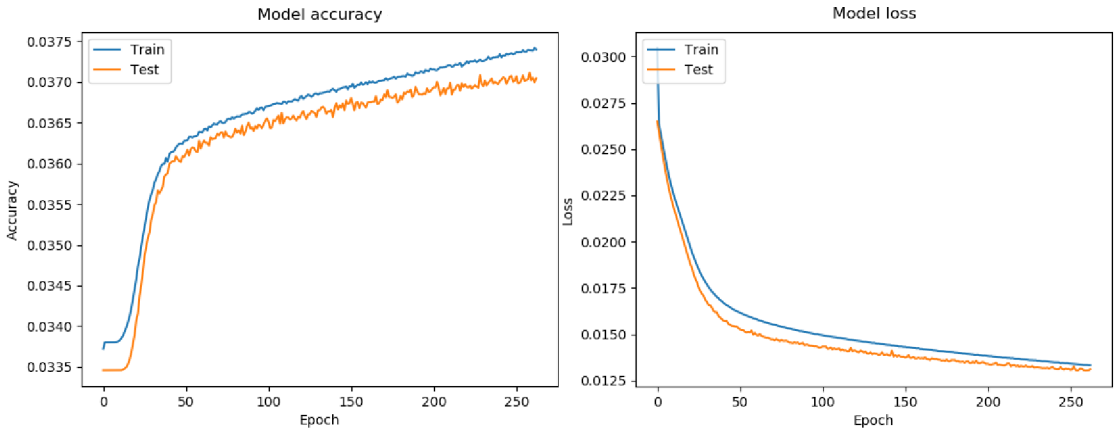
Obr. 20: Model 1-100-1



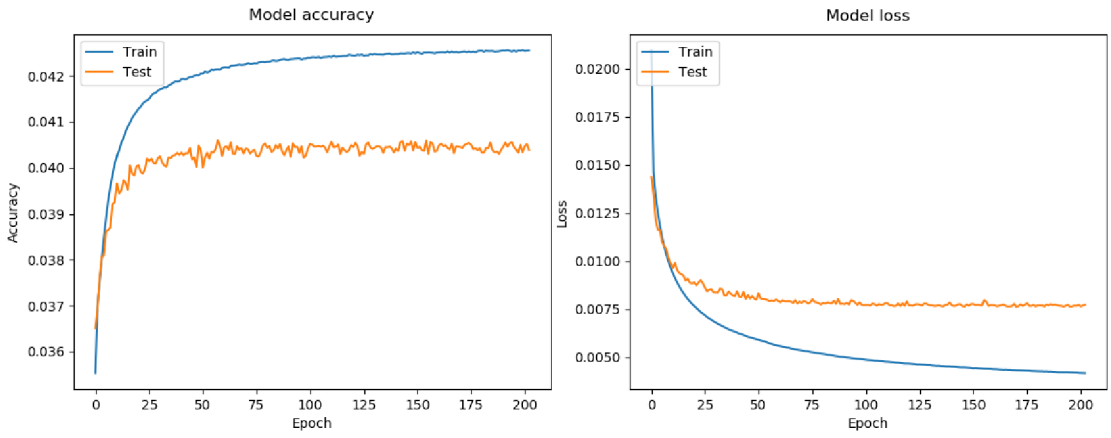
Obr. 21: Model 1-100-2



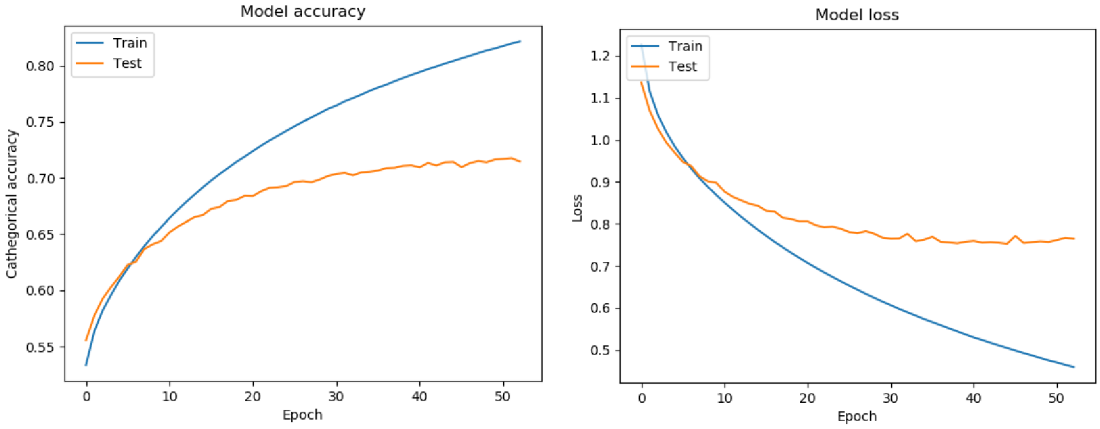
Obr. 22: Model 2-50



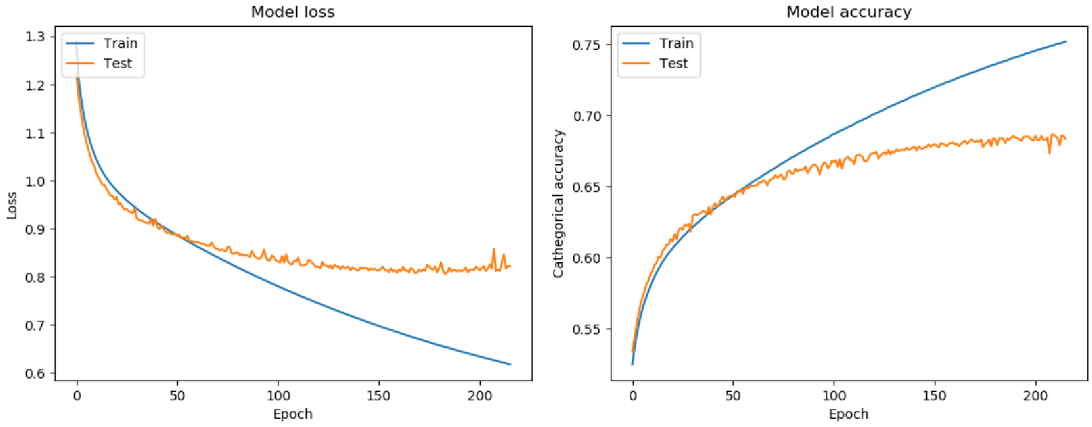
Obr. 23: Model 2-100-1



Obr. 24: Model 2-100-2



Obr. 25: Model 3-50



Obr. 26: Model 3-100

Z výsledných hodnot i grafů můžeme konstatovat, že učení bylo úspěšné a sítě byly naučeny. Nicméně informaci o reálné herní síle přinese až následující kapitola. Regresní úlohy vykazují velmi nízkou acc, nicméně to pouze znamená, že proložená funkce má

jistou míru generalizace, která jí znemožňuje procházet přímo danými hodnotami. Hodnota mse je však dostatečně nízká a lze tak předpokládat malou odchylku od požadovaných hodnot, acc zde není směrodatnou metrikou.

6.6 Testování síly programu

Beast má některá omezení, která je pro testování herní síly nutno brát v potaz. Jedná se především o omezení rychlosti propočtu, která je díky využití programovacího jazyka python a pomalejších knihoven nižší, než je běžné u ostatních šachových programů, pro které byla použita některá varianta jazyka C, moderního standardu. Z tohoto důvodu byly zvoleny parametry testování v Tab. 10, které zajišťují ohodnocení kvality heuristiky bez problému různých rychlostí propočtu. Nejzásadnějším rozdílem oproti běžnému testování je nepoužití časové kontroly, ale omezené hloubky propočtu. Zbylá nastavení jsou běžná a neliší se od nastavení obecně užívaných v uznávaných rating listech.

Tab. 10: Parametry testování

Nastavení	Hodnota
Typ hry	Omezená hloubka 2
Typ turnaje	2x každý s každým
CPU	Intel Core i7-7700HQ
RAM	128 MB
OS	Windows 10 Pro x64
GUI	Arena 3.5.1

Zvolená hloubka propočtu byla z časových důvodů zvolena 2 půl tahy, tedy hloubka, do které je schopen se dopočítat i začínající šachista bez větších problémů. Vzhledem k předpokladu, že herní síla Beasta se pohybuje kolem úrovně amatérských šachistů se zdá tato volba vhodnou. Typ turnaje byl zvolen 2x každý s každým, kdy druhou partii budou mít soupeři opačné barvy. Použitý hardware nehraje velkou roli, neboť při omezené hloubce propočtu by na pomalejším hardwaru bylo dosaženo stejného výsledku, pouze pomaleji. Všechny programy měly k dispozici 1 jádro notebookového procesoru Intel Core i7-7700HQ. Pro takto krátký a nenáročný výpočet není potřeba velké paměti RAM a programy tak dostaly 128 MB, které jsou dostatečné a zároveň to neklade velké požadavky na program, aby operoval s větším množstvím. Turnaj probíhal ve volně dostupném softwaru Arena 3.5.1 pod operačním systémem Windows 10.

Turnaje byly sehrány dva. Prvním bylo interní testování všech verzí heuristik Beasta, ve druhém byly vybrané verze postaveny proti programům ostatních autorů, které jsou volně dostupné a účastnily se testování CCRL. V Tab. 11 jsou jednotlivé verze prezentovaného programu Beast a v Tab. 12 ostatních 11 zúčastněných programů, jejich autoři a CCRL elo (pokud ho mají). Jednotlivé varianty programu Beast jsou označovány následovně – cBeast je klasickou sčítací heuristikou, nBeast značí heuristiku neuronovou

sítí a následuje číslo vážící se na model z kapitoly 6.5, rBeast je heuristikou provádějící náhodné tahy.

Tab. 11: Verze programu Beast

Program	Heuristika
cBeast	Klasická sčítací
nBeast 1-50	Model 1-50
nBeast 1-100-1	Model 1-100-1
nBeast 1-100-2	Model 1-100-2
nBeast 2-50	Model 2-50
nBeast 2-100-1	Model 2-100-1
nBeast 2-100-2	Model 2-100-2
nBeast 3-50	Model 3-50
nBeast 3-100	Model 3-100
rBeast	Náhodná

Tab. 12: Soupeři

Program	Autor	Národnost	Rating CCRL
ACE 0.1	Adam Adair	USA	374
AcquaD 3.2.26	Giovanni Di Maria	Itálie	607
ChessPuter r7	Evgeniy Korniloff	Rusko	800
Iota 1.0	Daniel White	Spojené království	1019
LaMoSca 0.10	Pietro Valocchi	Itálie	326
NSVChess v0.14	Nicolas Velin	Francie	945
Pos 1.20	F. J. J. van Heusden	Nizozemsko	443
Ram 2.0	Harm Geert Muller	Nizozemsko	553
Safrad 2.2.40.360	David Šafránek	Česká republika	1013
Saruman 2019.05.01	Terry Bolt Conor Griffin Darragh Griffin	-	1623
Stockfish 10 level0	T. Romstad M. Costalba J. Kiiski G. Linscott	Mezinárodní	-

Programy byly voleny dle dostupnosti ze spodní části tabulky CCRL tak, aby byly výkonnostně přijatelnými soupeři. K těmto byl přidán Stockfish 10 omezený na nejnižší výkonnostní stupeň 0 ze 20. Všechny zúčastněné programy jsou volně dostupné z [6].

Interní testovací turnaj BeastTour se skládal z 90 her, každý program odehrál 18 partií. Výsledky jsou prezentovány jako tabulka turnaje Tab. 13 a dle vzoru CCRL jako Tab. 14. První z tabulek obsahuje umístění jednotlivých programů po ukončení turnaje, jejich dosažené body, pomocné hodnocení Sonneborn–Berger (S-B) a informaci

o vzájemných zápasech (1 odpovídá výhře, ½ remíze a 0 prohře). Pomocné hodnocení je využíváno pro řešení situací se stejným počtem bodů dvou či více účastníků turnaje a je součtem bodů poražených protivníků a poloviny bodů soupeřů, se kterými bylo dosaženo remízy. Za prohru se pomocné hodnocení nezvyšuje. Druhá tabulka obsahuje relativní elo rating vůči aktuálnímu setu protihráčů s náhodnou heuristikou rBeast fixovanou na rating 0, interval 95% jistoty hodnoty ratingu (hranice + a –) a skóre v procentech. Tyto hodnoty byly počítány programem bayeselo, který využívá i CCRL. [28]

Tab. 13: Výsledková tabulka BeastTour

#	Program	Body	S-B	1	2	3	4	5	6	7	8	9	10
1	cBeast	18,0/18	144	--	11	11	11	11	11	11	11	11	11
2	nBeast 2-100-2	13,0/18	86,0	00	--	00	11	11	10	11	11	11	11
3	nBeast 2-50	12,0/18	82,0	00	11	--	01	00	11	11	11	01	11
4	nBeast 1-100-2	12,0/18	74,0	00	00	10	--	11	01	11	11	11	11
5	nBeast 1-50	11,0/18	66,0	00	00	11	00	--	11	01	11	11	11
6	nBeast 2-100-1	8,0/18	45,0	00	01	00	10	00	--	10	01	11	11
7	nBeast 1-100-1	6,0/18	29,0	00	00	00	00	10	01	--	01	10	11
8	nBeast 3-50	6,0/18	22,0	00	00	00	00	00	10	10	--	11	11
9	nBeast 3-100	4,0/18	18,0	00	00	10	00	00	00	01	00	--	11
10	rBeast	0,0/18	0,0	00	00	00	00	00	00	00	00	00	--

Tab. 14: Bayeselo rating BeastTour

#	Program	Elo	+	-	Skóre [%]
1	cBeast	960	311	177	100
2	nBeast 2-100-2	664	164	145	72
3	nBeast 2-50	620	159	145	67
4	nBeast 1-100-2	616	156	143	67
5	nBeast 1-50	570	151	142	61
6	nBeast 2-100-1	434	144	147	44
7	nBeast 3-50	352	140	155	33
8	nBeast 1-100-1	350	142	157	33
9	nBeast 3-100	248	147	173	22
10	rBeast	0	179	312	0

Přesvědčivým vítězem turnaje je klasická sčítací heuristika cBeast bez ztráty bodů, následovaná konvoluční sítí nBeast 2-100-2. Diskuse výsledku následuje v kapitole 7 této práce. Pořadí v turnaji není shodné s pořadím dle elo listiny, což je dáno systémem bayeselo, který bere v potaz nejen bodové výsledky, ale také barvu figur, se kterými jich bylo dosaženo. Hrát s černými figurami se považuje za drobnou nevýhodu, protože bílý

začíná hru. Proto je vítězství černými hodnotnější pro elo rating než vítězství bílými. Tento princip není aplikován pro hodnocení lidských hráčů.

Pro snížení počtu účastníků a větší přehlednost výsledků byly do následujícího turnaje zařazeny pouze 3 verze Beasta – vítěz interního turnaje cBeast, nejlepší konvoluční síť nBeast 2-100-2 a nejlepší dopředná síť nBeast 1-100-2. Klasifikační úloha aplikovaná v modelu 3 se neprokázala vhodnou a žádná její varianta nebyla dále testována. V kompletním testovacím turnaji GrandTour bylo odehráno 182 her mezi 14 účastníky s maximálním možným počtem dosažených bodů 26. Výsledky jsou opět prezentovány formou turnajové tabulky Tab. 15 a rating spočítán v Tab. 16 programem bayeselo s referenčním enginem cBeast fixovaným na hodnotu 960, které dosáhl v BeastTour.

Tab. 15: Výsledková tabulka GrandTour

#	Program	Body	S-B	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	cBeast	22,5/26	247,00	--	00	1½	11	11	10	11	11	11	11	11	11	11	11
2	Saruman 2019.05.01	22,0/26	241,50	11	--	0½	01	10	½1	11	11	11	11	11	11	11	11
3	Iota 1.0	22,0/26	238,25	0½	½1	--	01	10	11	11	11	11	11	11	11	11	11
4	nBeast 2-100-2	19,0/26	188,00	00	10	10	--	11	10	00	11	11	11	11	11	11	11
5	Stockfish 10 level0	18,0/26	180,50	00	01	01	00	--	11	11	10	11	01	11	11	11	11
6	ChessPuter r7	17,0/26	151,50	01	½0	00	01	00	--	0½	11	11	11	11	11	11	11
7	Safrad 2.2.40.360	16,0/26	137,75	00	00	00	11	00	1½	--	10	1½	11	11	11	11	11
8	nBeast 1-100-2	12,5/26	88,75	00	00	00	00	01	00	01	--	½1	1½	½1	11	11	11
9	NSVChess v0.14	8,5/26	50,25	00	00	00	00	00	00	0½	½0	--	11	½0	½½	11	11
10	AcquaD 3.2.26	8,0/26	49,25	00	00	00	00	10	00	00	0½	00	--	½½	½1	11	11
11	LaMoSca 0.10	6,0/26	37,25	00	00	00	00	00	00	00	½0	½1	½½	--	½½	½0	1½
12	Ram 2.0	4,0/26	23,50	00	00	00	00	00	00	00	00	½½	½0	½½	--	½½	½0
13	ACE 0.1	3,5/26	16,00	00	00	00	00	00	00	00	00	00	00	½1	½½	--	½½
14	Pos 1.20	3,0/26	12,50	00	00	00	00	00	00	00	00	00	00	0½	½1	½½	--

Tab. 16: Bayeselo rating GrandTour

#	Program	Elo	+	-	Skóre [%]
1	cBeast	960	169	139	87
2	Iota 1.0	936	157	134	85
3	Saruman 2019.05.01	927	159	135	85
4	nBeast 2-100-2	810	147	134	73
5	Stockfish 10 level0	777	143	134	69
6	ChessPuter r7	735	132	125	65
7	Safrad 2.2.40.360	695	130	126	62
8	nBeast 1-100-2	559	124	124	48
9	NSVChess v0.14	435	121	127	33
10	AcquaD 3.2.26	409	123	130	31
11	LaMoSca 0.10	344	118	128	23
12	Ram 2.0	277	121	134	15
13	ACE 0.1	236	128	148	13
14	Pos 1.20	217	132	156	12

V GrandTour zvítězil cBeast, konvoluční síť nBeast 2-100-2 získala 4. místo a dopředná síť nBeast 1-100-2 skončila na 8. místě.

Poslední ze série testování byla hra proti lidským protihráčům. Vybrána pro prezentaci byla verze nBeast 2-100-2 jako nejsilnější vytvořená heuristika neuronovou sítí. Podmínky zápasu byly pro program stejné jako při předchozím testování, lidský hráč dostal tolik času k propočtu, kolik uznal za vhodné. Soupeři byli tři amatérští hráči – Radek Š. Jan M. a Jakub U. (jména byla zkrácena pro dodržení GDPR). Tab. 17 prezentuje výsledky tohoto krátkého zápasu.

Tab. 17: Hry proti lidským hráčům

Hráč	Body	nBeast	Radek	Jan	Jakub
nBeast 2-100-2	4,5/6	--	½1	11	01
Radek Š.	0,5/2	½0	--	--	--
Jan M.	0,0/2	00	--	--	--
Jakub U.	1,0/2	10	--	--	--

7 ZHODNOCENÍ A DISKUZE

Kapitola 6 popsala prezentovaný šachový program Beast, jeho strukturu, využité algoritmy a heuristiky, výsledky interního testování a prezentovala testování proti opozici programů ostatních tvůrců i lidské.

Beast je univerzálním šachovým programem fungujícím pod různými GUI přes UCI komunikační protokol. Jeho jediným funkčním omezením je rychlost propočtu, která je v porovnání s ostatními existujícími systémy nízká. To je způsobeno využitím programovacího jazyka python 3, který není známý pro svou rychlost, a také některých pomalejších knihoven, především knihovny *python-chess*. Tento problém je možné překonat přepisem programu bez využití *python-chess* knihovny, nicméně rychlostní deficit oproti opozici je možné eliminovat pouze převodem do jiného programovacího jazyka jako např. C/C++ nebo Rust, které jsou pro dosažení vyšší rychlosti výpočtu vhodnější volbou.

Nabídka funkcí pro hru a analýzu je dostatečná a překonává většinu programů podobné výkonnostní úrovně. Program je schopen hrát jak s využitím časové kontroly, tak se speciálními podmínkami propočtu do omezené hloubky, nebo maximálního počtu prohledaných uzlů. Beast je také jednoduše použitelný pro tzv. nekonečnou analýzu, která je základním nástrojem při hodnocení šachové pozice a spočívá v neomezeném propočtu, který je zastaven až rozhodnutím uživatele.

Beast integruje jak základní klasickou sčítací heuristiku, tak originální heuristiku učenými neuronovými sítěmi. Programy využívající neuronové sítě ještě nedávno vůbec neexistovaly, a i dnes je jich velmi málo. Tento přístup byl inspirován systémy AlphaZero a Lc0, které jsou v současné době jedinými opravdu úspěšnými projekty. Struktura sítě, tvar vstupních dat a jejich příprava jsou originálními výtvory, které nebyly aplikovány v žádném jiném programu.

7.1 Interní testování

Interní testování herní síly přineslo poznatky o vlivu aplikace jednotlivých modelů sítě, optimalizátorů, použitého typu úlohy i počtu her, ze kterých byly sítě učeny. Základním poznatkem je, že veškeré aplikované heuristiky nehrají náhodně. Heuristika aplikující náhodnou hru, rBeast, byla ve všech hrách poražena. To prokazuje, že všechny heuristiky neuronovými sítěmi byly úspěšné a jejich využití zvyšuje herní sílu programu. Následující shrnutí vyjadřuje dále komentované výsledky interního testování.

- vyšší počet her k učení zlepšuje herní výkon
- optimalizátor Adam dosáhl lepších výsledků než SGD
- konvoluční neuronová síť je vhodnější než dopředná síť

Sítě učené s optimalizátorem Adam dosáhly lepších výsledků než sítě s SGD a lze tedy konstatovat, že Adam je vhodnější pro tento typ komplexní úlohy. Větší počet použitých

her k učení očekávaně přináší lepší výsledky hry, nicméně modely učené na méně datech s optimalizátorem Adam stále překonávají modely s více daty, ale používající SGD. Regresní úloha se prokázala lepší volbou než úloha klasifikační, jejíž výsledky jsou velmi špatné.

Typ modelu hrál také podstatnou roli. Konvoluční síť překonala ve výsledné herní síle jednoduchou dopřednou. Předpoklad důležitosti vzorů tvořených specifickým seskupením figur a důraz na jejich vzájemnou polohu se tak potvrdil a konvoluční sítě jsou tento trend schopny zachytit a využít při hodnocení pozice.

Nejsilnější aplikovanou heuristikou je klasická sčítací, která prokázala vyšší úroveň hry proti všem heuristikám neuronovými sítěmi. To je pravděpodobně způsobeno vhodností použití sčítací heuristiky s prohledávacím algoritmem alpha-beta, který nemusí nutně být ideálním ve spolupráci s neuronovými sítěmi. AlphaZero i Lc0 s úspěchem využívají upravenou variantu MCTS, která by mohla být vhodnějším řešením i v tomto případě. Dalším faktorem je malé množství her k učení, jehož navýšení by mělo pozitivní vliv na zvýšení herní síly. Nejsilnější vytvořenou sítí je nBeast 2-100-2.

7.2 Herní výsledky

Ve velkém turnaji GrandTour, kde byl Beast nasazen proti programům dalších autorů, dosáhly jeho hrající varianty dobrých výsledků. Klasická sčítací heuristika cBeast ovládla tabulku, což prokazuje korektní aplikaci prohledávacích algoritmů a vhodné parametry uvnitř heuristiky samotné. Umístění verzí využívajících neuronové sítě na 4. a 8. místě ze 14 účastníků je velmi dobré a ukazuje, že možnost využít neuronové sítě pro komplexní hru typu šachu je už v dnešní době možné.

Výsledky testovacího turnaje trpí na menší množství odehraných her, nicméně odehrání většího množství nebylo z časových důvodů vhodné a pro ilustraci herní síly je toto množství zcela dostatečné. Všechny heuristiky Beasta je možno nadále vylepšovat. Sčítací heuristiku lze posílit přidanými podmínkami a bonusy pro jednotlivé figury, neuronové sítě se mohou učit z většího množství partií. Všem heuristikám navíc může pomoci zvýšení rychlosti výpočtu, jak bylo vysvětleno v předchozí části tohoto textu.

Zápasy s lidskými protivníky potvrdily předpoklad, že herní síla programu se pohybuje na úrovni amatérského šachisty.

7.3 Herní styl

Diskutovat herní styl je velmi obtížné a často velmi subjektivní, ať už se jedná o lidské hráče, nebo šachové programy a systémy. Přesto zde bude uvedeno několik pozorování, které autor provedl při sledování jednotlivých partií.

Klasická sčítací heuristika hraje na pohled zvláštně. Prvořadým cílem je materiál, pro jehož zisk či udržení je schopna poškodit svou pěšcovou strukturu, nebo dostat do nebezpečí své vlastní figury. Umisťování figur se řídí pouze statickými pravidly a není

tak žádnou zvláštností např. střelec blízko centra, ale uzavřený vlastními pěšci, což v praxi snižuje jeho hodnotu výrazným způsobem. Heuristika vítězí pomocí lepšího propočtu a využití získaného materiálu, prohrává při korektní poziční hře soupeře, nebo přílišné aktivitě jeho figur.

Neuronové sítě naopak nepřikládají dostatečnou váhu materiálu. Figury a pěšce staví velmi dobře a jejich pozice nemají přílišných slabin. Pokud však stojí před volbou umístění figury do špatné pozice, často ji raději obětují za pseudo-poziční výhodu, která se v pozici reálně nenachází. Velkou váhu kladou na aktivní hru figur a útok proti soupeřovu králi. Vyhrávají zajímavými matovými kombinacemi, prohrávají především kvůli časté ztrátě materiálu.

8 ZÁVĚR

V úvodu práce je představena historie šachu od jeho vzniku, přes ustálení v moderní podobě, až po dnešní dobu. V kapitole o počítačovém šachu byla rozebrána jeho historie a nejsilnější šachové programy v průběhu let. Popsány byly turnaje mezi programy a boje o titul nejsilnějšího programu na světě.

Dále byla rozebrána struktura klasického šachového programu, popsána reprezentace šachovnice, prohledávací algoritmy *alpha-beta pruning* a *Monte-Carlo Tree Search*, *quiescence search* a databáze koncovek.

Z moderních metod, které se začínají využívat při budování šachových programů, byly rozebrány neuronové sítě. Popsány byly jejich teoretické základy, a to jak základních dopředných, tak sítí s využitím konvoluce. Představeny byly také hlavní aktivační funkce v neuronových sítích. Detailněji byl popsán také první úspěšný šachový systém na bázi neuronových sítí, AlphaZero.

Teoreticky rozebrané principy byly dále aplikovány v šachovém programu Beast, vytvořeném v programovacím jazyce python. Beast je univerzálním šachovým enginem a komunikuje s grafickými uživatelskými prostředími přes protokol UCI. V programu byl aplikován prohledávací algoritmus *alpha-beta pruning*, *quiescence search* a *delta pruning*. Vytvořeny byly 3 typy herních heuristik – klasická sčítací, heuristika neuronovou sítí a náhodná hra. Neuronové sítě aplikují dvě základní úlohy – regresní a klasifikační. Vytvořeno a natrénováno bylo 8 modelů neuronových sítí a jejich herní síla byla porovnána v interním turnaji BeastTour. Nejlepší sítě byly postaveny do turnaje GrandTour, kde se utkaly s programy ostatních autorů. Výsledky byly velmi dobré, klasická heuristika v turnaji zvítězila a neuronové sítě obsadily 4. a 8. místo ze 14 účastníků.

Hra proti lidským hráčům potvrdila předpoklad, že herní síla neuronových sítí je na úrovni amatérského šachisty, což je při malém množství her použitých k učení velmi dobrý výsledek. Herní sílu je nadále možné zvyšovat učením z většího množství šachových partií. Aplikace neuronových sítí na hru šachy je velmi komplexní a náročná. Principy předvedené v této práci jsou pouze náhledem možností, které tato moderní metoda nabízí.

SEZNAM POUŽITÉ LITERATURY

[1] MURRAY, Harold James Ruthven. *A History Of Chess*. Oxford University Press, 1978. ISBN 0198274033.

[2] HOOPER, David a Ken WHYLD. *The Oxford companion to chess*. 2nd ed. New York: Oxford University Press, 1992. ISBN 0198661649.

[3] *World Chess Federation* [online]. [cit. 2019-05-12]. Dostupné z: https://www.fide.com/

[4] CCRL TEAM. CCRL 40/4. *Computer Chess Rating List* [online]. 2005-2013 [cit. 2019-05-14]. Dostupné z: http://ccrl.chessdom.com/ccrl/404/

[5] HSU, Feng-hsiung. *Behind deep blue: building the computer that defeated the world chess champion*. Princeton: Princeton University Press, c2002. ISBN 0-691-09065-3.

[6] Mastering the Game: A History of Computer Chess. *Computer History Museum* [online]. [cit. 2019-05-12]. Dostupné z: https://www.computerhistory.org/chess/

[7] *Top Chess Engine Competition* [online]. [cit. 2019-05-12]. Dostupné z: https://tcec.chessdom.com/

[8] *International Computer Games Association* [online]. [cit. 2019-05-12]. Dostupné z: https://icga.org/

[9] *Internation Correspondence Chess Federation* [online]. [cit. 2019-05-12]. Dostupné z: https://www.iccf.com/

[10] *Chessprogramming* [online]. [cit. 2019-05-12]. Dostupné z: https://www.chessprogramming.org/

[11] BROWNE, Cameron B., Edward POWLEY, Daniel WHITEHOUSE, et al. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2012, 4(1), 1-43 [cit. 2019-05-14]. DOI: 10.1109/TCIAIG.2012.2186810. ISSN 1943-068X. Dostupné z: http://ieeexplore.ieee.org/document/6145622/

[12] MCTS (English) - Updated 2017-11-19.svg. In: *Wikimedia Commons* [online]. 2017 [cit. 2019-05-14]. Dostupné z: https://commons.wikimedia.org/wiki/File:MCTS_(English)_-_Updated_2017-11-19.svg

[13] Endgame Tablebases. *Chessprogramming* [online]. [cit. 2019-05-12]. Dostupné z: https://www.chessprogramming.org/Endgame_Tablebases

[14] BASHEER, I.A a M HAJMEER. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods* [online]. Elsevier B.V, 2000, 43(1), 3-31 [cit. 2017-04-22]. DOI: 10.1016/S0167-7012(00)00201-3. ISSN 0167-7012.

[15] ŠNOREK, Miroslav. *Neuronové sítě a neuropočítače*. Praha: ČVUT Praha, 2004, 156 s. ISBN 80-01-02549-7.

[16] Convolution arithmetic - No padding no strides.gif. In: *Wikimedia Commons* [online]. 2017 [cit. 2019-05-14]. Dostupné z: https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_No_padding_no_strides.gif

[17] Max pooling.png. In: *Wikimedia Commons* [online]. 2017 [cit. 2019-05-14]. Dostupné z: https://commons.wikimedia.org/wiki/File:Max_pooling.png

[18] O'SHEA, Keiron a Ryan NASH. An Introduction to Convolutional Neural Networks. *CoRR*. 2015, (abs/1511.08458).

[19] GUPTA, Dishashree. Fundamentals of Deep Learning – Activation Functions and When to Use Them?. *Analytics Vidhya* [online]. [cit. 2019-05-12]. Dostupné z: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activationfunctions-when-to-use-them/

[20] SILVER, David, Thomas HUBERT, Julian SCHRITTWIESER, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* [online]. 2018, 362(6419), 1140-1144 [cit. 2019-05-14]. DOI: 10.1126/science.aar6404. ISSN 0036-8075. Dostupné z: http://www.sciencemag.org/lookup/doi/10.1126/science.aar6404

[21] GLICKMAN, Mark E a Albyn C JONES. Rating the chess rating system. *CHANCE-BERLIN THEN NEW YORK-*. SPRINGER INTERNATIONAL, 1999, (12), 21-28.

[22] GLICKMAN, Mark E. *The Glicko system*. Boston University, 1995.

[23] TCEC Rating List. *Top Chess Engine Competition* [online]. [cit. 2019-05-12]. Dostupné z: https://tcec.chessdom.com/ordo.txt

[24] MACŮREK, Miloslav. Beast. *GitHub* [online]. GitHub, 2019 [cit. 2019-05-14]. Dostupné z: https://github.com/maelic13/beast

[25] KAHLEN, Stefan-Meyer. UCI protocol. *WBEC Ridderkerk* [online]. [cit. 2019-05-14]. Dostupné z: http://wbec-ridderkerk.nl/html/UCIProtocol.html

[26] Talkchess Forum. *Talkchess* [online]. [cit. 2019-05-14]. Dostupné z: http://www.talkchess.com/forum3/viewtopic.php?topic_view=threads&p=487051&t=45545

[27] Point Value. *Chessprogramming* [online]. [cit. 2019-05-14]. Dostupné z: https://www.chessprogramming.org/Point_Value

[28] COULOM, Rémi. *Bayesian Elo Rating* [online]. [cit. 2019-05-20]. Dostupné z: https://www.remi-coulom.fr/Bayesian-Elo/

[29] BIRD, H. E. *Mistrovské perly*. Přeložil Antonín ČÍŽEK. Ostrožská Nová Ves: Galerie Dolmen, 2017. ISBN 978-80-87303-39-9.

[30] POLGÁR, László. *Šachy: 5334 úloh, kombinací a partií*. Přeložil Pavel MATOCHA. Praha: Slovart, [2016]. ISBN 978-80-7529-184-4.

[31] MACŮREK, Miloslav. *Aplikace neuronových sítí.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2017. 66 s. Vedoucí bakalářské práce prof. RNDr. Ing. Jiří Šťastný, CSc.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH