FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Zobecnění hry Pac-man z hlediska tahových herních algoritmů
Student:	Bc. Kristián Kuľka
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je navrhnout herní mechanismus pro zobecnění hry Pac-man, která na rozdíl od původní varianty z roku 1980 obsahuje více Pac-manů. Zároveň v rámci tématu na původně arkádovou hru nahlížíme jako na tahovou, podobně jako na šachy. Každý z hráčů, tj. hráč ovládající Pac-many resp. duchy může ve svém tahu pohnout každou ze svých postav o jeden krok, hráči se v tazích střídají. Jako základní přístup k řešení se nabízí použití algoritmu minimax s různými heuristikami. Možné je adaptovat i přístup navržený v rámci systému AlphaGo (Zero) či posilované učení. Úkoly řešitele jsou následující:

1. Prostudujte algoritmy pro řešení tahových her a zhodnoťte jejich aplikovatelnost na tahovou variantu zobecnění hry Pac-man.
2. Na základě provedené rešerše navrhněte nový nebo modifikujte existující herní algoritmus.
3. Navrženou metodu implementujte formou softwarového prototypu a otestujte v relevantních scénářích. Například je možné vyzkoušet různé varianty algoritmů hrající proti sobě.

–

[1] Marika Ivanová, Pavel Surynek, Katsutoshi Hirayama: Area Protection in Adversarial Path-finding Scenarios with Multiple Mobile Agents on Graphs - A Theoretical and Experimental Study of Strategies for Defense Coordination. ICAART (1) 2018: 184-191.

[2] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, Demis Hassabis: Mastering the game of Go without human knowledge. Nat. 550(7676): 354-359 (2017).

[3] Viliam Lisý, Branislav Bosanský, Michal Jakob, Michal Pechoucek: Goal-based Adversarial Search - Searching Game Trees in Complex Domains using Goal-based Heuristic. ICAART 2009: 53-60.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. ledna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Zobecnění hry Pac-man z hlediska tahových herních algoritmů

Bc. Kristián Kuľka

Katedra softwarového inženýrství
Vedúci práce: doc. RNDr. Pavel Surynek, Ph.D.

11. februára 2022

Poďakovanie

Chcel by som sa poďakovať vedúcemu práce, doc. RNDr. Pavlovi Surynkovi, Ph.D., za veľmi zaujímavú tému a vedenie, vďaka ktorému som si rozšíril znalosti. Taktiež moja vďaka patrí rodine a blízkym, za veľkú podporu počas celého štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 11. februára 2022 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na túto prácu

Kuľka, Kristián. *Zobecnění hry Pac-man z hlediska tahových herních algoritmů*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Hra Pac-Man predstavuje zaujímavý problém z pohľadu umelej inteligencie, keďže predstavuje multi-agentný systém, ktorý si vyžaduje koordináciu agentov a dlhodobé plánovanie. Hru navyše komplikuje fakt, že jednotlivé akcie sú častokrát nevýrazné a nemajú signifikantný efekt na stav hry, čo sťažuje rozhodovanie a výber optimálneho ťahu.

V tejto práci zobecním hru Pac-Man z pohľadu ťahových hier, pričom uvažujem väčší počet agentov (duchov aj pacmanov). Predstavím jej základné problémy a zhrniem širšie spektrum prístupov z literatúry. Následne predstavím poupravené implementované metódy, ktoré vychádzali z algoritmu negamax, AlphaZero a z pôvodnej implementácie z roku 1980. Opíšem taktiež ich možné vylepšenia, ktoré môžu adresovať objavené problémy. Na vyhodnotenie, optimalizáciu parametrov a trénovanie týchto stratégií som implementoval obecný modul spoločne s vizualizačnou funkcionalitou, ktorý taktiež detailne opíšem. Prácu zakončím rozborom výsledkov na rôznorodých mapách.

Klíčová slova spätnoväzovné učenie, hlboké učenie, alfa-beta, minimax, Pac-Man, Ms. Pac-Man, multi-agentné systémy.

Abstract

The Pac-Man game represents an interesting problem in the field of artificial intelligence since it belongs to the category of the multi-agent systems which require good dose of cooridnation and long-term planning. This game is also complicated by the fact that immediate actions do not significantly change the state of the game many times, which makes it harder to decide an optimal move.

In this thesis, I will define the general version of the Pac-Man from the perspective of turn-based games with multiple agents (pacmans and ghosts). I will introduce the main problems and summarize some approaches from the literature. Next, I will introduce implemented methods, which were based on the negamax algorithm, AlphaZero and original implementation of the game from the year 1980. I will also mention their possible improvements, which can adress the occurred problems. I have also implemented the visualization module and general framework which is capable of evaluating strategies, optimizing their parameters and training, which will be discussed in its own chapter. I will finish the thesis by analysing results, which were obtained by evaluating strategies on various maps.

Keywords reinforcement learning, deep learning, alpha-beta, minimax, Pac-Man, Ms. Pac-Man, multi-agent systems.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Hry boli od nepamäti súčasťou ľudských civilizácií. Je preto neprekvapivé, že s príchodom počítačov a teda možností ako hry vytvárať, počet hier explodoval. Pre obor počítačových vied a špecificky pre obor umelej inteligencie, sa stali hry zaujímavým poľom na testovanie rôznych algoritmov a to najmä vďaka jednoducho implementovateľným pravidlám, z ktorých sa dal zhotoviť simulátor a taktiež vďaka ich obrovskej „mentálnej“ komplexite.

Azda jednou z najznámejších hier je hra Pac-Man z roku 1980. V nej má hráč za úlohu zjesť tzv. pacdots, ktoré sa nachádzajú v rôznych častiach bludiska. Hráč sa pritom musí vyhýbať duchom, ktorý sa ho snažia chytiť a zjesť. Vzhľadom na dynamiku problému a nutnosť dlhodobého plánovania je tento problém zaujímavým pre obor umelej inteligencie.

Medzi ciele tejto práce patrí zobecniť práve hru Pac-Man, pričom mám uvažovať väčšie množstvo agentov, čo je s spojené s exponenciálne rýchlo rastúcim množstvom ťahov. Situáciu taktiež komplikuje fakt, že ťahy v rámci tejto hry nie sú veľmi výrazné. Inými slovami, to, že sa pacman v jednom ťahu pohne o jedno políčko vľavo, nemá vo väčšine prípadov príliš veľký efekt na výsledok hry. Na signifikantnejšiu zmenu je potrebné väčšie množstvo postupných akcií.

Ďalším cieľom je práve zhodnotiť používané metódy z literatúry, ktorých je vzhľadom na popularitu hry hojne a implementovať niektoré metódy. Tie bude potrebné poupraviť na mieru tomuto problému.

Medzi ciele patrí taktiež implementovať toto riešenie a vyhodnotiť na rôznorodých instanciách hry.

Štruktúra textu

V prvej časti textu som opísal hry z obecného hľadiska. Následne som definoval obecnú verziu hry Pac-Man, rozobral jej problémy a zhrnul možné prístupy. V texte som pokračoval kapitolou s implementovanými metódami,

ich možnými vylepšeniami, popisom implementácie a finálnym zhodnotením metód.

Kapitola 1

Ťahové hry

Množstvo hier si vyžaduje dlhodobé plánovanie, intuíciu, pamäť, rýchle uvažovanie a množstvo ďalších vlastností, ktoré je možno pokladať za inteligentné. Prirodzene sa teda naskytá otázka, ako má všeobecné riešenie hier blízko ku všeobecnej umelej inteligencii?

V tejto kapitole stručne popíšem históriu hier, spoločne s typom riešení, ktoré postupne vznikali a následne budem pokračovať sekciou o definícii hry pre dvoch hráčov, ktorú v tejto práci uvažujem, spomeniem základné vlastnosti týchto hier a problémy, ktoré sa pri ich riešení častokrát objavujú a následne kapitolu zakončím základným prístupom ku hľadaniu optimálnej stratégie.

1.1 História a význam

V histórií riešenia hier bolo množstvo míľnikov. Jeden z azda najznámejších bol systém Deep Blue od spoločnosti IBM [6], ktorý v roku 1997 prvýkrát v histórií porazil vtedajšieho vládnuceho šampióna v šachu Garry-ho Kasparova. Systém bol založený na algoritme minimax s alfa-beta orezávaním a evaluačnou funkciou, ktorá bola postavená na množstve expertných znalostí šachu.

Metódy, ktoré sa v túto dobu používali mali avšak viacero nedostatkov. Jednou znich je najmä fakt, že na vytvorenie rozumných riešení bolo potrebných množstvo expertných znalostí. Taktiež, riešenia boli postavené na prehľadávaní herného stromu, kde hráči dopredu simulujú jednotlivé kroky protihráčov (teda plánujú) a na základe toho volia svoj ťah. Čím hlbšie sa v hernom strome dostanú, tým presnejší ťah hráč odohrá. Avšak s počtom dostupných ťahov rastie veľkosť tohoto stromu exponenciálne (do hĺbky) a teda znemožňuje efektívne prehľadávanie. Pri hrách ako Go, v ktorých počet ťahov je niekoľkonásobne väčší ako pri šachu a herné pozície sú ťažšie z pohľadu evaluácie, sa algoritmom umelej inteligencie ešte dlho nedarilo priblížiť levelu svetového šampióna.

Zlom avšak nastal v roku 2016, kedy systém AlphaGo Lee [4] vyvinutý spoločnosťou DeepMind porazil svetového šampióna s 18 medzinárodnými titulmi v Go. Systém bol založený na algoritme stromového prehľadávania Monte Carlo v kombinácií s neurónovou sieťou, pomocou ktorej dokázal efektívne predikovať silné ťahy v rôznych pozíciach a taktiež evaluovať stav hry. Takto dokázal efektívne redukovať šírku herného stromu, keďže sa sústredil iba na silné ťahy a taktiež nutnosť prehľadávania do vyššej hĺbky, keďže dokázal efektívne odhadovať kvalitu pozície. Herný agent na začiatku tréningu naučil neurónovú sieť imitovať ľudské ťahy profesionálnych hráčov a následne hral množstvo hier sám proti pričom sa učil zlepšovať.

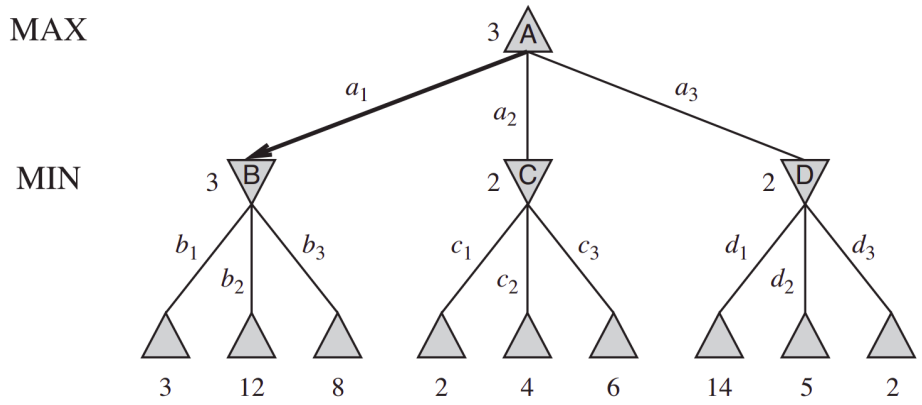
V roku 2019 bol publikovaný komplexný systém AlphaStar [7], taktiež od spoločnosti DeepMind, ktorý dosiahol Grandmaster úroveň v StarCraft II a finálne sa umiestnil nad 99.8 % oficiálne hodnotených hráčov. Náročnosť hry StarCraft II je daná najmä tým, že sa jedná o komplexnú stratégiu reálneho času s množstvom akcií a jednotiek, ktoré dokáže hráč nezávislé na sebe ovládať. Jedná sa navyše o hru s neúplnou informáciou t.j. hráč nemá k dispozícií kompletný stav hry a musí brať v úvahu určitú dávku neistoty. Systém je podobne ako AlphaGo Lee založený na učení neurónovej siete predikovať najlepšiu akciu v danom stave a to z hier, ktoré hraje sám proti sebe, pričom sa kontinuálne zlepšuje. Vzhľadom na enormný počet ťahov v jednom kroku, ktorý je približne rovný 10^26, tento systém vynecháva plánovanie, ktoré je prakticky nemožné.

Jednou zo základných motivácií vytvárať obecné riešenia na komplexné hry je ich schopnosť modelovať rôzne aspekty reálneho sveta. Ako príklad by sa dala uviesť aplikácia metódy AlphaZero (ktorá je všeobecným rozšírením metody AlphaGo Lee) na globálnu optimalizáciu kvantovej dynamiky [8] alebo podobná metóda, ktorá bola aplikovaná na objavovanie nových efektívnych postupností pri organickej syntéze [9].

1.2 Základné definície a vlastnosti

Existuje množstvo pohľadov na hry avšak v tejto práci sa zamerám na ťahové hry 2 hráčov s nulovým súčtom, kde sa hráči po každom ťahu striedajú a na konci hry každý z nich dostane číselnú odmenu. Cieľom oboch hráčov je maximalizovať svoje finálne skóre. Podľa [1] je na hru možné pozerať ako na prehľadávací problém s následujúcimi elementmi:

- S_0: Počiatočný stav v ktorom hra začína.
- PLAYER(s): Hráč, ktorý je na ťahu v stave s.
- ACTIONS(s): Validné ťahy pre daný stav s.
- RESULT(s, a): Prechodný model, ktorý definuje výsledný stav po aplikácii ťahu a v stave s.



Obr. 1.1: Ukážka herného stromu [1]

- TERMINAL-TEST(s): Funkcia, ktorá definuje, ktoré stavy hry sú finálne.
- UTILITY(s, p): Evaluačná funkcia, ktorá určuje výšku odmeny pre hráča p vo finálnom stave s.

Rôzne typy hier môžme rozlišovať na základe oboru hodnôt evaluačnej funkcie pre rôznych hráčov. Hry s nulovým súčtom majú vlasnosť, že súčet finálnych odmien hráčov je rovný nejakej konštante (častokrát práve 0). Napríklad v šachu by sme mohli definovať evaluačnú funkciu z pohľadu bieleho hráča ako -1 ak vyhrá čierny, 0 v prípade remízy a $+1$ ak vyhrá biely. Z pohľadu čierneho hráča je táto funkcia obrátená t.j. vynásobená -1. Vďaka tejto vlastnosti, môžme uvažovať iba jednú evaluačnú funkciu a na hráčov takýchto hier sa prirodzene pozerať ako na MAX a MIN, t.j. hráč ktorý sa snaží maximalizovať skóre alebo naopak minimalizovať. Je jednoduché si povšimnúť, že v hrách s nulovým súčtom platí, že ak jeden hráč niečo získa, druhý hráč musí nutne niečo stratiť. Môžme avšak definovať aj kooperatívne hry, v ktorých výhoda pre jedného hráča znamená výhodu pre ďalších hráčov, poprípade hry s nenulovým súčtom.

Pre analýzu hier je rozumné definovať pojem herný strom. Jedná sa o „rozbalený“ priebeh hry, ktorý vzniká pri doprednom plánovaní. Na jeho vrchole je aktuálny stav s a deti tohoto stavu sú rovné RESULT(s, a) pričom $a \in$ ACTIONS(s). Rekurzívnou aplikáciou tejto definície získame celý rozbalený strom. V listoch obsahuje finálne stavy, pre ktoré platí podmienka TERMINAL-TEST(s) a taktiež ich evaluáciu. Hĺbka stromu d obsahuje všetky stavy, do ktorých sa je možno dostať z ich vrcholu po aplikácií nanajvýš d ťahov. Príklad herného stromu s hĺbkou dva je možné vidieť na obrázku 1.1.

Jednou zo základných vlastností hier je ich exponenciálne veľký herný strom, ktorý má v hĺbke d pri počte ťahov b dokopy b^d stavov. Táto obrovská

komplexita znemožňuje jeho efektívne prehľadávanie naivnými metódami aj do minimálnych hĺbok. V hrách sa ale častokrát objavujú rôzne symetrie alebo cykly, ktoré je možné pri plánovaní využiť. V momente keď sa dostaneme do ekvivalentného ťahu, z ktorého sme už našli optimálny ťah, nemusíme ho analyzovať znova. Avšak ak chceme využiť tejto vlastnosti, je potrebné si tieto výsledky niekde ukladať a vzhľadom na veľkosť herného stromu a konečnej pamäte to môže byť problém.

V hrách s väčším množstvom samostatných jednotiek, ktoré môžme nezávisle na sebe ovládať, je táto komplexita ešte väčšia, keďže samotný počet možných ťahov rastie s počtom jednotiek exponenciálne.

U určitých typoch algoritmov a hier, ktoré sú založené na metóde spätnoväzobného učenia, môže dojsť taktiež k situácií, kedy sa agent naučí hrať hru proti stratégií s ktorou má aktuálne problém ale tým sa automaticky stane slabým voči inej stratégii. Jednou zo základných hier, ktorá ma takýto problém je kameň papier nožnice. Ak má napr. agent problém s hráčom, ktorý zakaždým hraje kameň, tak sa naučí hrať papier ale kvôli tomu začne prehrávať proti hráčovi, ktorý hraje nožnice, následne sa naučí hrať kameň ale tým znova začne prehrávať voči hráčovi, ktorý hraje papier a podobný cyklus naháňa donekonečna, bez zlepšenia.

Ďalším typickým problémom pri hrách sú ťahy, ktorých efektivita sa ukáže až za veľmi dlhú dobu. Plynie to z faktu, že na to aby si naivný agent uvedomil váhu ťahu, musí prehľadať do danej hĺbky herný strom, čo nie je vzhľadom na jeho exponenciálnu veľkosť výpočetne zvládnuteľné. Toto bol výrazný problém pri hre Go, ktorý sa podarilo prekonať až s kombináciou spätnoväzobného a hlbokého učenia.

Nielen tieto vlastnosti znamenajú, že riešiť hry môže byť obecne výzvou. Každopádne rôzne metódy sú častokrát priamo či nepriamo aplikovateľné na široké spektrum hier, ktoré plynú z tej istej obecnej definície a to je možno využiť na vytvorenie robustného finálneho algoritmu.

1.3 Základné metódy

Cieľom hráča je nájsť optimálnu stratégiu, ktorú môžme podľa [10] definovať ako funkciu, ktorá pre každý možný stav hry vráti pre daného hráča jeho ťah. Prirodzene sa naskytá otázka ako takúto stratégiu nájsť. Jedným zo základných algoritmov, ktorý rieši tento problém (pre našu definíciu hry) je minimax, ktorý je definovaný následovne [1]:

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

Na problém hľadania optimálnej stratégie sa môžme pozerať odzadu. Ak sme o jednu hĺbku nižšie oproti poslednému stavu v hernom strome, tak jednoduchou enumeráciou svojich detí môžme nájsť optimálny ťah a teda získať evaluáciu aj pre súčasný stav. V prípade MAX hráča to bude ťah, ktorý vedie do stavu s najväčšou hodnotou evaluačnej funkcie. V prípade MIN hráča to bude zasa opačne. Túto informáciu následne propagujeme až do stavu, z ktorého chceme nájsť optimálny ťah. Toto je v skratke všetko čo robí minimax znázornený vyššie.

Jednou z jeho hlavných výhod (samozrejme okrem optimality) je, že nepoužíva žiadnu doménovú znalosť o probléme a je teda aplikovateľný na akúkoľvek hru, ktorá spĺňa našu definíciu. Každopádne existujú taktiež hry, kde výsledok akcií nie je pevne daný a jednotlivé stavy, ktoré vznikajú po aplikácií RESULT(s, a) nastávajú s určitou pravdepodobnosťou.

Z definície algoritmu minimax je vidieť, že na nájdenie optimálnej stratégie musíme prehľadať celý herný strom. Našťastie v histórií bolo vytvorených množstvo metód ako sa s týmto obmedzením vysporiadať a nájsť efektívne (sub)optimálne stratégie pre konkrétne ale aj obecné hry. Tieto metódy a ich aplikácia na hru Pacman rozoberem v nasledujúcej kapitole.

Kapitola 2

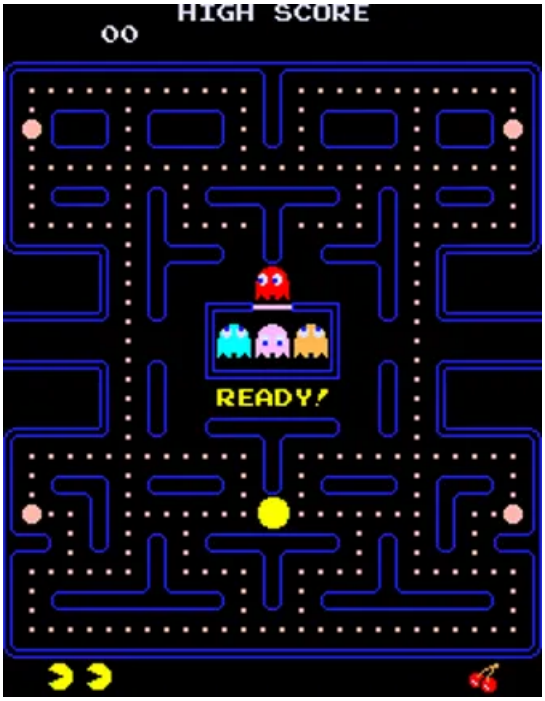
Pac-Man

Hra Pac-Man (s originálnym názvom Puck-Man) pochádza z roku 1980. Bola navrhnutá japonským herným designérom Toru Iwatani, ktorý pracoval pre spoločnosť Namco [2]. Veľmi rýchlo sa stala globálnym fenoménom a pokladá sa za jednu z najznámejších hier vôbec. Preto nie je prekvapujúce, že výskum v tejto oblasti je pestrý.

V tejto kapitole predstavím pôvodnú verziu hry, v skratke opíšem aktívny výskum, predstavím používané prístupy na riešenie hry Pac-Man ale aj obecné prístupy, ktoré sú aplikovateľné na akúkoľvek hru s nulovým súčtom a na záver opíšem ich silné a slabé stránky. Na týchto poznatkoch sú postavené implementované metódy, ktoré predstavím v ďalšej kapitole.

2.1 Originálna verzia

Hra sa odohráva na 2D mape pripomínajúcu bludisko na ktorej sa nachádza pac-man (žltá postavička; ďalej označovaný skrátene pacman), duchovia, tzv. pac-dots (ďalej označované skrátene pacdots; jedná sa o biele bodky) a ovocia, ktoré sa snaží pacman zjesť a vylepšenia (tučné biele bodky), ktoré po zjedení umožnia pacmanovi na krátku dobu duchov zjesť. Duch sa po zjedení presunie na krátku dobu do domčeka a následne začne znova pacmana naháňať. Ovocia sa na mape zjavujú iba raz za čas. Cieľom hráča je ovládať pacmana tak aby maximalizoval svoje skóre, pričom sa musí vyhýbať duchom, ktorý sa ho snažia obkľúčiť a zjesť. Za každú zjedenú pacdot, ovocie alebo s vylepšením taktiež za každého zjedeného ducha, dostáva pacman skóre. Hra pozostáva z niekoľkých levelov, ktoré sa striedajú potom čo pacman zje všetky pacdots na aktuálnej mape. Hráč ma k dispozícií niekoľko životov a počas hry môže nejaký život taktiež získať. Na okrajoch mapy môžu byť taktiež „teleporty“, ktoré pacmana okamžite presunú z jednej strany mapy na druhý. V momente keď pacman bez vylepšenia narazí na ducha, prichádza o život a level začína odznova. Na začiatku levela vychádzajú duchovia postupne jeden za druhým zo svojho domčeka, ktorý sa nachádza v strede mapy. Hra končí s posledným



Obr. 2.1: Ukážka hry Pac-Man [2]

strateným životom. Ukážku hry je možné vidieť na obrázku 2.1 poprípade si je možné zahrať online jej novšiu verziu Ms. Pac-Man na stránke `https://www.mspacman1.com/`.

Stojí za zmienku, že duchovia majú rôzne vlastnosti, červený sa snaží zakaždým dostať čo najbližšie k pacmanovi, ružový sa naopak snaží dostať pred pacmana atp. Taktiež sa z času na čas striedajú módy hry tzv. scatter (rozptýlenie) a chase (naháňanie). V scatter móde sa duchovia snažia rozptýliť a dostať do svojho prideleného rohu mapy. V chase móde sa naopak snažia pacmana chytiť. V prvej verzii hry Pac-Man bolo správanie duchov deterministické, čoho všímaví hráči dokázali využiť. V novšej verzii Ms. Pac-Man autori pridali určitú dávku náhody ale kľúčové charakteristiky duchov ostali nemenné. Znalý hráč to teda dokáže využívať vo svoj prospech. Ako príklad sa dá uviesť napr. správanie oranžového ducha, ktorý sa v chase móde snaží pacmanovi vyhýbať. Tento fakt môže výrazne pomôcť pri plánovaní bezpečnej cesty. Avšak je nutne myslieť aj na už spomenutý fakt, že duchovia nemôžu zmeniť smer v ktorom sa hýbu na opačný a teda ak sa oranžový duch nachádza v dlhej uličke a smeruje ku pacmanovi, tak v danom smere bude naďalej pokračovať až po najbližšiu križovatku [2].

Jednou z kľúčových vlastností hry je plánovanie ťahov pacmana tak aby po vojdení do uličky mohol z danej uličky bezpečne vyjsť, následne plánovanie dlhšej postupnosti ťahov, ktoré ho bezpečne dovedú do rôznych vzdialených

kútov máp, kde mu ostáva nejaké pacdots zjesť a taktiež nutnosť rýchlych reakcií vzhľadom na fakt, že sa jedná o hru reálneho času.

2.2 Zhrnutie výskumu

Za posledných 20 rokov bol výskum v oblasti riešenia hry Pac-Man bohatý čo potvrdzuje aj anketa z roku 2018 [11], ktorá zhrňuje desiatky prác. Medzi najpoužívanejšie metódy patria tzv. pravidlami riadené systémy, algoritmy založené na prehľadávaní herného stromu, evolučné metódy a metódy spätnoväzobného učenia (tie budeme naďalej označovať ako RL metódy, čo je tradične sa vyskytujúca skratka prevzatá z anglického Reinforcement Learning).

Popularite tejto tematike prispieva taktiež fakt, že existujú rôzne súťaže, ktoré porovnávajú implementované metódy. Jedným z výrazných plusov takýchto súťaží je zdieľaná platforma a pravidlá hry, ktoré umožňujú priamu komparáciu prístupov. Avšak v literatúre sa objavuje taktiež množstvo prací, v ktorých si autori mierne upravujú pravidlá hry, čím znemožňujú porovnávanie a teda meranie globálneho progresu v rámci tohoto problému.

V rámci literatúre sa objavujú stratégie pre hráča pacman ale aj pre duchov. Ciele a motivácie autorov sa lýšia, napr. v rámci práce [12] sa autori snažili vytvoriť stratégiu pre duchov, ktorá nebude príliš silná aby odrádzala hráčov a taktiež ani príliš slabá aby zachovali nejakú dávku výzvy. V iných prácach sa naopak snažili vytvoriť čo najsilnejšiu stratégiu pre hráča pacman alebo pre duchov [11].

2.3 Definícia zobecnenej verzie

V tejto sekcii formálne definujem verziu problému, ktorú budem v tejto práci riešiť. Problém zobecním na ľubovoľné množstvo pacmanov a duchov a ľubovoľný typ mapy. Budem sa držať faktu, že na hru Pac-Man (alebo Ms. Pac-Man) sa dá s určitou dávkou abstrakcie pozerať ako na ťahovú hru dvoch hráčov s nulovým súčtom, kde UTILITY(s, p) je pre hráča pacman rovná *score* a pre duchov rovná $-score$. Toto mi umožní používať obecné metódy, ktoré sú aplikovateľné na tento typ hier.

Hra je definovaná ako pätica (w, h, s_0, B, t_{max}) kde $w \in \mathbb{N}$ predstavuje šírku mapy a $h \in \mathbb{N}$ jej výšku, $b_{i,j} \in B$ blok na pozícií (i, j), s_0 počiatočný stav hry a $t_{max} \in N$ maximálna počet ťahov v hre. Mapa sa skladá z políčok, ktoré sú indexované dvojicou $(i, j) \in \mathbb{N}^2$, pričom platí, že $0 <= i < w$ a $0 <= j < h$, teda, prvý index označuje šírku a druhý výšku. Touto dvojicou budem označovať ľubovoľnú pozíciu na mape, ktorú môžme vnímať ako 2D mriežku. Každý objekt, vyskytujúci sa v hre, má pridelenú práve jednu pozíciu. Stav hry je definovaný ako pätica $(P, G, D, t, player)$. kde P je množina pacmanov, G množina duchov, D množina pacdots, t predstavuje aktuálny počet odohraných ťahov

a posledný prvok pätice $player \in \{\, \text{pacman}, \text{player} \,\}$ označuje hráča, ktorý je aktuálne na ťahu. Množinou $O = B \cup P \cup G \cup D$ označím zjednotenie všetkých objektov. Funkciou $pos(o) = (i, j)$ definujeme pozíciu objektu, pričom $o \in O$.

Ďalšie funkcie, typické pre hru dvoch hráčov, ktoré som spomenul v sekcii 1.2 definujem následovne:

- s_0: Počiatočný stav je daný definíciou hry, každopádne zakaždým vňom začína hru hráč pacman a čítač ťahov je nastavený na 0.
- ACTIONS(s): Každý hráč hýbe v jednom ťahu všetkými svojimi jednotkami naraz. Hráči sa po každom ťahu striedajú. Symbolom $a \in A$ označím akciu konkrétnej jednotky, pričom $A = \{\, up, down, left, right, none \,\}$. Ťah hráča definujem ako $\mathbf{a} \in A^n$ kde n predstavuje počet jednotiek hráča. Aplikáciou akcie má za výsledok zmenu pozície jednotky o jedno políčko nahor, dole, vľavo, vpravo alebo v prípade akcie *none* sa jej pozícia nemení. Ak duch stúpi na políčko kde stojí pacman alebo pacman stúpi na políčko kde stojí duch, pacmana odstránime z množiny P po celú nasledujúcu dĺžku hry. Analogicky to funguje v prípade pacmanov a pacdots. Kolízie medzi jednotkami nie sú povolené až na duchov a pacdots. Inými slovami v žiadnom stave nemôže pre ľubovoľnú dvojicu $o_1, o_2 \in O$ nastať, že zdieľajú tú istú $pos(o)$ s výnimkou dvojice $(o_1, o_2) \in (G, D)$ alebo $(o_1, o_2) \in (D, G)$.
- RESULT(s, a): Prechodný model, ktorý definuje výsledný stav po aplikácii ťahu a v stave s. Ten má za následok zmenu pozícií objektov a úpravu množín P a D v prípade niektorej z kolízií.
- TERMINAL-TEST(s): Hra končí v momente kedy počet pacdots alebo počet pacmanov začne byť rovný 0 alebo ak aktuálny počet ťahov v stave s dosiahne maximálnej hodnoty t_{max}.
- UTILITY(s, p): v prípade, že $p =$ pacman tak evaluačnú funkciu definujem ako $|P_{s_0}| - |P_s|$. Inak je rovná obrátenej hodnote $-|P_{s_0}| + |P_s|$.

Narozdiel od originálneho Pac-Man v tejto verzii neuvažujem vylepšenia, ktoré dovolia pacmanom jesť duchov ani ovocia, väčší počet životov, rôzne levely a taktiež ani dom z ktorého duchovia na začiatku hry vychádzajú. Taktiež neuvažujem možné „teleporty“, ktoré dovolia prechádzať pacmanom a duchom z jednej strany mapy na druhú. Uvažujem avšak ľubovoľný počet duchov a pacmanov, ktorý má za následok exponenciálny nárast počtu ťahov čo výrazne sťažuje hru.

2.4 Vlastnosti

Jedna z hlavných „nepríjemných“ vlastností hry Pac-Man je množstvo akcií, ktoré je potrebných vykonať predtým, než sa signifikantne zmení stav hry.

Napr. 2 stavy, ktoré sa lýšia len vtom, že v jednom znich je o 3 pacdots menej než vtom druhom, sú viacmenej ekvivalentné. Podobne to je s pohybom pacmana. Ak sú od neho duchovia príliš ďaleko tak to, že sa pohne vpravo a náhle zmení smer vľavo, nemá nutne príliš veľkú váhu na výsledok. Človek, aj napriek tejto komplikácií, ktorá sťažuje plánovanie, dokáže dosahovať slušné výsledky aj v jeho prvých iteráciách hrania. Dalo by sa hypotetizovať, že to je najmä vďaka rôznym hierarchickým abstrakciám, v ktorých je ľudský mozog geniálny. Pomocou nich môže ignorovať množstvo mini ťahov a plánovať dlhodobo štýlom: „Chcem sa dostať do pravého horného rohu lebo tam vidím množstvo pacdots a málo duchov.“ Avšak stroj nedokáže bez učenia tieto abstrakcia vytvárať a na začiatku mu neostáva nič iné, než pri plánovaní slepo prehľadávať obrovský herný strom, v ktorom sa pri malých hĺbkach môže zdať každý ťah rovnako dobrý. Tento problém ešte zvyšuje kombinácia exponenciálne rastúceho množstva ťahov, vzhľadom na počet jednotiek n, ktoré je rovné $\mathcal{O}(|A|^n)$. To predstavuje signifikantný problém pre dlhodobé plánovanie, kde môže aj rozdiel jedného ťahu znamenať podstatný rozdiel v skóre, napr. v prípade kedy pacman tesne nestihne dôjsť do uličky s množstvom pacdots lebo mu v predchádzajúcom stave cestu zablokoval duch.

Pre kontrast ešte spomeniem hru šach, ktorý týmto problémom netrpí, keďže po odohraní zopár ťahov sa stav hry mení výrazne. Špeciálne to platí pre rozohranú partiu. Stačí sa pozrieť na akýkoľvek zápas aj priemerných šachových hráčov, kde zahodenie jedného alebo dvoch ťahov môže nutne znamenať prehru.

Je jednoduché si povšimnúť, že počet cyklov je v tejto hre obrovský a množstvo stavov sa opakuje. Ako príklad stačí uviesť situáciu kde pacman a duchovia dokola prechádzajú na jedno susedné políčko a späť. Takto sa budú striedať 4 stavy, a aj keby sme prehľadali herný strom do nekonečnej hĺbky, vždy by sme na túto postupnosť ťahov narazili. Je preto veľmi výhodné (v tejto hre by bolo možné povedať, že dokonca až nutné) opakujúce sa stavy po prehľadaní uložiť do cache pamäte a v prípade, že na nich pri plánovaní narazíme znova, použiť predpočítaný výsledok.

O hrách je možno rozprávať ako o symetrických alebo skoro-symetrických, v slova zmysle, že stratégia, ktorá funguje pre jedného hráča sa dá priamo či nepriamo použiť pre hráča druhého. Pri asymetrických hrách táto vlasnosť chýba[1]. V symetrických hrách sa teda stačí naučiť jednu zdieľanú stratégiu, ktorú je následne možné aplikovať pre oboch hráčov. Ak by napr. hod vyváženej mince rozhodoval o tom, kto začne hru, tak symetrickými hrami by boli napríklad šach, Go, dáma alebo japonský šach šógi. Ale aj bez tejto počiatočnej náhody môžme dané hry považovať za skoro symetrické čo zjednodušuje učenie napr. z pohľadu RL systémov[2]. Existujú avšak aj asymetrické hry kde sa

[1] Viac informácií o tejto tématike je možné nájsť v [10], sekcia Symmetric games

[2] Spätnoväzobné systémy reagujú s prostredím, v tomto prípade hrajú hru, a na základe spätnej väzby upravujú svoje akcie. Pri hrách môže spätná väzba predstavovať napr. informáciu či danú hru vyhrali, prehrali alebo veľkosť skóre, ktoré dosiahli.

stratégie a ciele hráčov výrazne lýšia, čo môže sťažovať učenie. Pac-Man spadá do tejto kategórie, keďže to, že sa hráč naučí ovládať pacmana mu negarantuje, že dokáže ovládať taktiež duchov štýlom, ktorým minimalizuje výsledné skóre.

V hre Pac-Man avšak existujú aj jednoduché heuristiky, ktoré sa dajú využiť pri hraní a môžu dosahovať slušných výsledkov. Napríklad hráč pacman si môže zakaždým dávať pozor, či po vojdení do uličky bude môcť relatívne bezpečne vyjsť, čo sa dá jednoducho zistiť pomocou najkratších ciest pacmanov a duchov ku východom uličky. Ak sa tam dostane ako prvý pacman tak je všetko fajn. Ak avšak z uličky neexistuje bezpečný východ, pacman je zablokovaný. Podobnú vlasnosť využívali ako jednu z 10 kľúčových vlastnosti stavu hry pri RL metóde [5] a dosiahli veľmi kvalitných výsledkov.

2.5 Metódy z literatúry

V tejto kapitole predstavím rôzne typy metod, ktoré sa objavujú v literatúre. Zakaždým ich rozdelím do obecnejších kategórií, ktoré skrátene opíšem a následne rozoberem detailne konkrétne prístupy. Z týchto poznatkov som vychádzal pri návrhu stratégií pre zobecnenú verziu hry Pac-Man zo sekcie 2.3.

2.5.1 Pravidlami riadené stratégie

Medzi základne typy stratégií sa radia pravidlami riadené stratégie. V krátkosti by sa dali opísať ako sada *AK podmienka POTOM platí nejaký fakt alebo agent má vykonať nejakú akciu* pravidiel. Tento prístup je následne možné rozširovať napríklad pomocou konečných stavových automatov [13] v ktorých agent počas hry prechádza medzi stavmi, ktoré obsahujú zoznamy pravidiel definujúce správanie pacmana; behaviorálnych stromov [14], ktoré je možné rôzne kombinovať a vytvárať komplexné stratégie alebo fuzzy logiky [15], ktorá do tradičnej logiky pridáva určitú škálu pravdy, pomocou ktorej môžme vytvárať flexibilnejšie pravidlá.

Tieto stratégie sú zväčša veľmi ľahko interpretovateľné a jednoduché na výpočet. Avšak, vyžadujú si podstatnú znalosť problému a pravidlá vytvorené na mieru. To môže byť pri komplexnejších hrách problém. Taktiež sa oveľa ťažšie prepoužívajú naprieč rôznymi hrami, nejedná sa teda o úplne obecné metody ale skôr riešenia šité na mieru.

2.5.1.1 Fuzzy pravidlá

Systémy, ktoré používajú striktné pravidlá ako AK A POTOM B majú nevýhodu, že podmienka A môže byť skoro pravdivá avšak to sa nijak neprejaví na pravdivostnej hodnote B. Tento problém rieši fuzzy logika [16], kde jednotlivé pre-

menné nadobúdajú reálnych hodnôt z intervalu $[0, 1]$ a pravidlá môžu platiť čiastočne.

V práci [15] autori vytvorili sadu fuzzy pravidiel, ktorých parametre optimalizovali prostredníctvom evolučnej stratégie (1+1)ES. Každé pravidlo aplikovali postupne na všetkých duchov a výstup skombinovali do finálnej akcie z množiny $\{ up, down, left, right \}$. Zoznam týchto pravidiel je následovný:

- AK je duch blízko POTOM vyhýbaj sa mu. Toto pravidlo v skratke znamená, že pacman sa bude snažiť v jeho ďalšom kroku maximalizovať vzdialenosť od blízkeho ducha. V prípade, že je týchto duchov viac, tak sa snaží vyhýbať viacerým.
- AK pacman zjedol vylepšenie a duch je blízko POTOM snaž sa ducha dohoniť.
- AK je vzdialenosť pacmana ku najbližšej križovatke, v smere v ktorom sa hýbe, menšia než vzdialenosť ducha k tejto križovatke POTOM choď na križovatku.

V prípade, že nenastane aktivácia žiadneho pravidla, pacman sa vydá cestou ku najbližšej pacdot.

Pri týchto fuzzy pravidlách je potrebné vytvoriť funkcie, ktoré pojmom blízko a ďaleko priradzujú nejakú hodnotu z intervalu $[0, 1]$, v tomto prípade sa jedná primárne o funkciu vzdialenosti. Pre každé pravidlo autori definovali funkciu na mieru s dvoma parametrami, ktorých hodnoty následne zoptimalizovali.

2.5.1.2 Originálna stratégia duchov

Originálna stratégia duchov [2] spadá taktiež do pravidlami riadených hier. V hre Pac-Man sa striedajú 2 módy: scatter a chase. V scatter móde sa duchovia rozptyľujú do svojich rohov (každý znich má pridelený práve jeden roh). V chase móde sa snažia chytiť pacmana. Po určitom počte opakovaní sa avšak zapne iba chase mód, ktorý trvá až do konca hry. V prípade zmeny módu duchovia menia svoj smer na opačný avšak inak sa nikdy nemôžu obrátiť a taktiež nemôžu stáť. Kolízie medzi duchmi sú povolené. Správanie duchov plynie z jednoduchého pravidla - na každej križovatke sa vyber smerom, ktoré minimalizuje vzdušnú vzdialenosť k tvojmu cieľovému políčku, pričom sa nesmieš obrátiť a taktiež nemôžeš stáť. Je ľahké si povšimnúť, že v úzkej uličke bude duch pokračovať stále dopredu, keďže to je jediný smer, ktorým sa môže vydať.

V prípade scatter módu je cieľové políčko ducha fixné po celú dobu hry a závisí na type. Scatter cieľové políčka sú umiestnené v blízkosti rohov. V prípade chase módu je výber cieľového políčka trochu viac komplikovaný a každý typ ducha to má inak:

- Blinky, červený duch, si zakaždým vyberá svoje cieľové políčko ako políčko kde stojí pacman. Toto ma za následok, že Blinky väčšinu času pacmana priamo prenasleduje a častokrát končí tesne za ním, čo donúti pacmana utekať opačným smerom.

- Pinky, ružový duch, si volí svoje cieľové políčko na základe aktuálnej orientácie pacmana a to konkrétne o 4 bunky pred neho. Spolu s Blinky tvoria skvelý pár, keďže Blinky pacmana naháňa zozadu a Pinky sa snaží dostať pred neho a odrezať ho od únikových východov.

- Inky, tyrkysový duch, si volí svoje cieľové políčko na základe aktuálnej orientácie pacmana a pozícií Blinky-ho. V prvom kroku označí stredné políčko, ktoré sa nachádza o 2 políčka pred pacmanom. Následne ho spojí úsečkou s Blinkym a tú zdvojnásobí. Jeho cieľové políčko sa nachádza na opačnej strane úsečky, oproti Blinky-mu.

- Clyde, oranžový duch, strieda 2 módy. Ak jeho vzdialenosť od pacmana nízka, zvolí ako jeho cieľové políčko pozíciu pacmana. Ak táto vzdialenosť prekročí nejakú hraničnú hodnotu, jeho cieľom sa stane scatter políčko. Prekvapivo, sa teda tento duch pacmanovi vyhýba a je najmenej neškodný (pokiaľ sa pacman nenachádza v blízkosti jeho rohu).

Duchom sa taktiež v rámci levelov zvyšovala rýchlosť. V definícií stratégie sa objavuje ešte zopár nuáns ako popis správania duchov v stave, kedy ich dokáže pacman zjesť atp. ale to pre stručnosť nebudem uvádzať. Kompletný zoznam detailov je možné nájsť na [2].

Napriek svojej jednoduchosti a rýchlosti výpočtu sa jedná o relatívne efektívnu a elegantnú stratégiu. Avšak, ak hráč spozná vlastnosti duchov, tak to môže miestami šikovne využiť vo svoj prospech.

2.5.2 Stromové prehľadávanie

Na výber najlepšieho ťahu sa dá pozerať ako na iteratívny proces, ktorý si postupne zlepšuje svoje odhady najsilnejších ťahov pomocou prehľadávania. Toto je základná myšlienka Metod stromového prehľadávania. Medzi ich základné vlastnosti patrí, že čím väčšmi prehľadajú herný strom, tým presnejšie dokážu mať výsledky. Tieto metódy sú veľmi obecné a častokrát aplikovateľné na rôzne problémy s minimom úprav, keďže každú hru z definície 2.3 je možno reprezentovať ako herný strom.

V tejto sekcii spomeniem dve metódy: Alfa-beta orezávanie, ktorá predstavuje základný optimálny algoritmus a obecnejšiu heuristicku metódu MCTS (stromové prehľadávanie Monte-Carlo).

2.5.2.1 Alfa-beta orezávanie

Algoritmus minimax trpí vlastnosťou, že musí prehľadať celý herný strom na to aby našiel optimálny ťah. Avšak ukazuje sa, že množstvo podstromov, môžme pri prehľadávaní úplne vynechať a dosiahneme ten istý výsledok. Toto vylepšenie prináša Metoda minimax s alfa-beta orezávaním [17], ktorá predstavuje základnú optimálnu metódu, pri riešení ťahových hier.

Implementácia minimax s alfa-beta orezávaním je pomerne elegantná a jednoduchá, každopádne v prípade hier s nulovým súčtom sa dá ešte zjednodušiť pomocou algoritmu negamax, ktorý je v tomto prípade s alfa-beta ekvivalentný. Je možné taktiež využiť transpozičné tabuľky, ktoré si ukladajú štatistiky o už prehľadaných stavoch hry [18, 19] čo ušetrí čas v prípade, že pri prehľadávaní herného stromu narazíme na ten istý stav viackrát. To sa pri hre Pac-Man vďaka veľkému množstve cyklov stáva často. Definíciu kompletného algoritmu je možné vidieť na pseudokóde 1.

Je zrejmé, že vzhľadom na exponenciálnu veľkosť herného stromu ho nie je možné prehľadať úplne. Kvôli tomu musíme prehľadávanie v určitej hĺbke zastaviť a namiesto finálnej evaluácie vrátiť evaluáciu danú heuristickou funkciou, ktorá sa snaží čo najviac priblížiť ku skutočnej evaluácií. Heuristická evaluačná funkcia, tvorí základ algoritmu a **výrazne** ovplyvňuje kvalitu výsledku.

Alfa-beta orezávanie rozširuje minimax o 2 hodnoty α a β. Každý uzol v hernom strome si tieto hodnoty vždy nainicializuje od svojho predka a následne ich aktualizuje na základe výsledkov svojich potomkov. V prípade algoritmu negamax α zakaždým predstavuje garantovanú spodnú hranicu aktuálneho hráča a β naopak garantovanú vrchnú hranicu protivníka. V prípade, že aktuálna α prekročí alebo sa vyrovná β, môžme ukončiť prehľadávanie v aktuálnom uzle, keďže náš protihráč by sa do daného podstromu pri optimálnej hre nikdy nevydal, keďže má k dispozícií lepší ťah. Toto vylepšenie dokáže zväčšiť hĺbku prehľadávaného stromu približne o faktor 4/3 oproti základnému algoritmu minimax [20] (za predpokladu rovnakých výpočetných prostriedkov).

Každopádne alfa-beta je citlivé na poradie prehľadávaných ťahov. K tomu, aby sme herný strom čo najviac orezali, je potrebné prehľadávať ťahy počnúc najsilnejšími. Túto informáciu avšak nemáme k dispozícií (inak by sme nemuseli robiť žiadne prehľadávanie). Existuje avšak množstvo heuristík ako ťahy zoraďovať, napr. oplatí sa začať potencionálne „rušivými ťahmi“, ktoré majú za následok výraznú zmenu stavu hry (v šachu by to boli ťahy ako vykopnutie figúrky), poprípade používať iteratívne prehlbovanie v kombinácií s ukladaním aktuálne najlepších ťahov, ktoré bude algoritmus skúšať ako prvé pri hlbších iteráciach [21]. Vzhľadom na to, že pri každom orezaní sa zbavíme exponenciálne veľkého podstromu, radiace algoritmy môžu mať aj mierne väčšiu časovú komplexitu (napr. polynomiálnou) a stále dosiahnuť lepších výsledkov.

Ako som spomenul v predošlých sekciách, používanie transpozičných tabuliek [19] je veľmi dôležité pri hrách s cyklami. Pri algoritme negamax je po-

Algorithm 1 Alfa-beta negamax s orezávaním a transpozičnou tabuľkou

```
procedure NEGAMAX(node, depth, player, α, β)
    alpha_orig := α
    tt_entry := transposition_table.get(node)
    if tt_entry is valid and tt_entry.depth ≥ depth then
        if tt_entry.flag = EXACT then
            return tt_Entry.value
        else if tt_entry.flag = LOWER_BOUND then
            α := max(α, tt_entry.value)
        else if tt_entry.flag = UPPER_BOUND then
            β := min(β, tt_entry.value)
        end if
        if α ≥ β then
            return tt_entry.value
        end if
    end if

    if depth = 0 or TERMINAL-TEST(node) then
        return player * the heuristic evaluation of node
    end if
    value := α
    for each child of node do
        child_value = −NEGAMAX(child, depth − 1, −player, −β, −α)
        value := max(value, child_value)
        α := max(α, value)
        if α ≥ β then
            break
        end if
    end for

    if value ≤ alpha_orig then
        flag := UPPER_BOUND
    else if value ≥ β then
        flag := LOWER_BOUND
    else
        flag := EXACT
    end if
    transposition_table.store(node, player, value, flag)
    return value
end procedure
procedure FIND_BEST_MOVE(root, player, max_depth)
    return  argmax_{c∈root.children}  NEGAMAX(c, max_depth, −player, −∞, ∞)
end procedure
```

trebné pri každom *tt_entry* udržovať flag, či sa jedná o vrchnú, spodnú alebo presnú hodnotu. To závisí na predaných hodnotách α a β od predka uzlu a na fakte, či nastalo prerezanie alebo nie. V prípade kedy alfa-beta prerezanie nastane, dostávame spodnú hranicu, keďže niektorý z neprehľadaných podstromov môže obsahovať väčšiu hodnotu. V prípade kedy je aktuálna *value* menšia než *orig_alpha* dostávame hornú hranicu, keďže to znamená, že sme prehľadali všetkých potomkov súčasného uzlu, avšak v týchto uzloch nastalo prerezanie, čo môže znamenať, že ich finálna hodnota je ešte menšia (v algoritme negamax sa každý uzol pozerá na svojich bezprostredných potomkov ako na hráčov MIN). V prípade, že finálna *value* je v intervale medzi $[alpha\_orig, \beta]$, tak aktuálny výsledok prehľadávania by bol rovnaký aj keby sme nainicializovali hodnoty $\alpha := -\inf$ a $\beta := \inf$, jedná sa teda o presnú hodnotu.

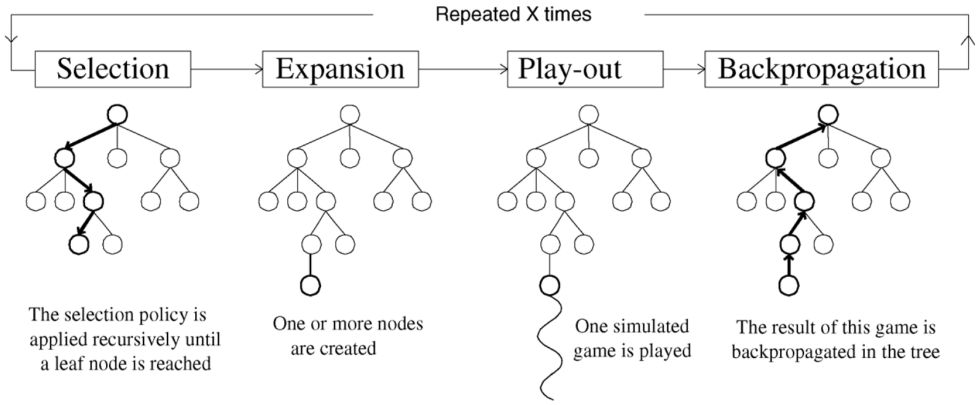
Na to aby sme uložili záznam *tt_entry* do transpozičnej tabuľky, je potrebné zrátať jeho haš kľúč, čo môže byť mierne výpočetne náročné. Avšak, stavy v hrách (špeciálne stolových) sa väčšinou po jednom ťahu lýšia len vo veľmi málo detailoch. K efektívnemu rátaniu haš kľúčov je možné využiť Zobrist hašovanie [22], ktorý tohoto faktu využíva.

Alfa-beta orezávanie je teda obecná a optimálna metoda. Medzi jej nevýhody avšak patrí, že slepo prehľadáva herný strom (čo je výpočetne náročné), môže byť niekedy krátkozraký (napr. v prípade keď ukončíme prehľadávanie v nestabilnom stave, kde sa výhoda hráčov môže s ďalšími ťahmi rýchlo meniť) a jeho kvalita výrazne závisí na evaluačnej heuristickej funkcii, ktorú je taktiež potrebné navrhnúť expertom. Tieto a ďalšie problémy sa výskumníci snažili v posledných desaťročiach rôzne adresovať a vznikali triky, vďaka ktorým je možné dosahovať veľmi sľubných výsledkov pri ťažších hrách aj s malými výpočetnými prostriedkami.

2.5.2.2 MCTS

Ďalšou tradičnou metódou z tejto kategórie je stromové prehľadávanie Monte-Carlo (MCTS). Jedná sa o heuristický algoritmus (nie je optimálny ako alfa-beta), ktorý si nutne nevyžaduje veľký počet expertných znalostí o probléme (narozdiel od evaluačnej funkcie alfa-beta). Je to teda veľmi obecná metóda. Jej ďalšia výhoda spočíva vtom, že sa počas prehľadávania snaží zamerať iba na potencionálne silné ťahy, pričom rozumne kombinuje dva aspekty, typické pri prehľadávaní, a to objavovanie nových ťahov, ktoré môžu byť potencionálne silnejšie než aktuálny najlepší ťah, a spresňovanie odhadov pre silné ťahy [23].

Algoritmus pozostáva zo 4 fáz, ktoré sa niekoľkokrát opakujú, pričom postupne buduje prehľadávaný herný strom. Po vypršaní časového limitu vráti najlepší výsledok. Pre vizualizáciu jednej iterácie algoritmu viď obrázok 2.2. Definícia týchto fáz je nasledovná:



Obr. 2.2: Stromové prehľadávanie Monte-Carlo [3]

- SELECTION: Na základe vopred danej funkcie, ktorá berie v úvahu rôzne štatistiky aktuálneho uzlu ako koľko krát bol navštívený, jeho priemerné skóre atp. vyber potomka. Toto opakuj pokiaľ sa nedostaneš na spodok aktuálne prehľadavaného stromu.

- EXPANSION: Pridaj do herného stromu jedného alebo viacerých potomkov a nainicializuj ich štatistiky.

- SIMULATION: Niekoľkokrát simuluj priebeh hry z expandovaného uzlu až do ukončovacej podmienky, ktorá môže byť napríklad počet odohraných ťahov alebo jednoducho pokiaľ nenarazíme na konečný stav hry. Stratégia, pomocou ktorej hráč vyberá v tejto fáze ťahy, môže byť buď náhodná alebo sa môže jednať o jednoduchú heuristiku. Cieľom je odsimulovať čo najväčší počet hier a napr. spriemerovať ich skóre, čo následne poslúži ako odhad hodnoty stavu. Intuitívne, ak hráč vyhrá 900/1000 simulovaných hier, tak môžme s vysokou pravdepodobnosťou povedať, že je to preňho výhodný stav.

- BACKPROPAGATION: V poslednej fáze sa tento odhad propaguje až do koreňa herného stromu. Počas toho sa uzlom aktualizujú štatistiky (väčšinou sa jedná o priemer). Pri každom vložení uzla do herného stromu, jeho evaluácií prostredníctvom simulácií a následnej spätnej propagácií, teda prinášame novú informáciu do herného stromu, ktorá zlepšuje presnosť štatistík uzlov a teda zlepšuje odhad silných a slabých ťahov.

MCTS sa teda namiesto slepého prehľadávania celého herného stromu snaží zamerať iba na ťahy, ktoré sa zdajú byť silné, pričom to kombinuje s rozumnou dávkou explorácie. V rámci práce [3] autori aplikovali túto metódu na

hru Pac-Man, pričom na každý ťah mali iba 40 ms a aj napriek tomu sa im podarilo dosiahnuť kvalitných výsledkov. Uzly v hernom strome reprezentovali križovatky na mape. Susedné križovatky boli spojené hranou. Pri MCTS teda ťah znamenal vydať sa smerom ku susednej križovatke. Autori kvôli efektívnejšiemu a hlbšiemu prehľadávaniu pri selekcii neuvažovali ťahy z uzlov, ktoré viedli späť do predchodcovskej križovatky. Herný strom sa vytváral iba pre pacmana, pričom pri každom ťahu predpokladali správanie duchov podľa vopred špecifikovanej stratégie. V rámci uzlov si udržiavali tri rôzne čísla, pričom každé reprezentovalo štatistiku nejakej taktiky: ghost, pill alebo survival. Tieto štatistiky boli aktualizované v rámci fázy BACKPROPAGATION. V každom uzle volili ťah do potomka, ktorý maximalizoval rovnicu 2.1. Ľavá strana predstavuje odhad kvality ťahu, pričom pravá strana zabezpečuje dostatočnú exploráciu. Konkrétne, v_i predstavoval výsledok funkcie súčasnej taktiky a štatistík uzlu, C exploračnú konštantu, n_i počet návštev potomka i z aktuálneho uzlu a n_p celkový počet návštev aktuálneho uzlu. So zvyšujúcim sa n_p táto rovnica konverguje ku hodnotám v_i. Kvôli presnejším výsledkom simulácie autori vytvorili heuristické simulačné stratégie pre duchov aj pacmana.

$$X_i = v_i + C * \sqrt{\frac{\ln n_p}{n_i}} \tag{2.1}$$

2.5.2.3 Hierarchická abstrakcia

V niektorých hrách je efekt bezprostredných ťahov na stav hry veľmi malý. Hra Pac-Man v tomto nie je výnimkou. Na to aby sa výraznejšie zmenil stav hry je potrebné vykonať zväčšia postupnosť niekoľkých akcií. Plánovanie na úrovni jednotlivých ťahov môže byť teda neefektívne a v kombinácií s vyšším faktorom vetvenia až prakticky nemožné. Tento problém riešia metody, ktoré stavy hry a ťahy abstrahujú do obecnejších konštruktov, ktorých je výrazne menej a následne plánujú medzi nimi.

V rámci práce [3] autori použili MCTS avšak neplánovali na úrovni jednotlivých ťahov pacmana t.j. hore, dole, vľavo a vpravo ale akcie zobecnili na cesty ku najbližším križovatkám, čo im umožnilo prehľadať strom do väčšej hĺbky a zároveň mali ťahy väčší vplyv na zmenu stavu hry, než napr. jeden krok „pohni pacman vľavo“.

V ďalšej práci [24] autori vytvorili obecný algoritmus, odvodený od max^n v ktorom jednotkám namiesto elementárnych akcií prideľujú abstraktné ciele. Každý cieľ má jasne definovanú postupnosť ťahov, ktorú má jednotka vykonať. Tie sa avšak môžu lýšiť v dĺžke, čo je potrebné adresovať. Algoritmus funguje približne na následujúcich myšlienkach:

- Ak majú aktuálne všetky jednotky pridelený cieľ, extrahuj ďalšiu akciu z ich stratégie a vykonaj ju.

- Ak niektorá z jednotiek dokončí svoj cieľ, vypočítaj pre ňu množinu aktuálne aplikovateľných cieľov, pre každý znich zavolaj rekurzívnu metódu prehľadávania a ako finálny cieľ (a teda ťah) zvoľ ten, ktorý dosiahol najlepších výsledkov.
- Ak počas plánovania narazíš na konečný stav alebo dosiahneš maximálnej hĺbky, zrátaj jeho evaluáciu prostredníctvom evaluačnej funkcie a ukonči prehľadávanie.

Je jednoduché si všimnúť, že pri takomto prehľadávaní sa priemerný vetviaci faktor herného stromu podstatne zníži, keďže vetvenie sa objavuje iba v hladinách v ktorých niektorá z jednotiek dokončí svoj cieľ.

2.5.3 Spätnoväzobné učenie

V spätnoväzobných systémoch (ďalej RL) má agent k dispozícií prostredie s ktorým môže interagovať. V každom čase sa rozhoduje, ktorú z dostupných akcií vykoná. Po vykonaní akcie obdrží zmenený stav prostredia a taktiež odmenu. Na základe týchto odmien následne upravuje svoje správanie. Jeho cieľom je v prostredí dosiahnuť čo najväčší možný súčet bezprostredných odmien až do koncového stavu (tento súčet budem nazývať výsledná odmena).

Jedná sa o veľmi obecnú metódu, pri ktorej stačí mať prístup ku prostrediu alebo jeho simulátoru. V prípade hier je prostredie veľmi „lacná“ záležitosť, keďže je možné simulátor naprogramovať. Následne môže agent s prostredím interagovať mnohokrát a učiť sa. Každopádne nie každé prostredie má takúto vlastnosť. Napr. v robotike je reálna interakcia s prostredím pomalá a v prípade chýb veľmi drahá.

Pri hrách RL agent predstavuje hráča hry a prostredie je samotná hra. Odmena pre prechod do konečného stavu môže byť definovaná ako +1, ak agent vyhrá, alebo -1, ak prehrá, a pre všetky ďalšie prechody je rovná 0. Poprípade môžme používať taktiež bezprostredné odmeny ak napr. agent odohral veľmi silný ťah.

Pri rozhodovaní využíva agent 2 základné funkcie: stratégia $\pi(a|s)$, ktorá prideľuje akciám v stave s pravdepodobnosti a odhad celkovej odmeny $v_\pi(s)$, ktorá predstavuje očakávanú výslednú odmenu z daného stavu pri dodržiavaní stratégie π. Funkcia $v_\pi(s)$ zvykne vo väčšine metod dávať stále menšiu a menšiu váhu budúcim odmenám čo numericky stabilizuváva algoritmus, keďže nie je možné dosiahnuť nekonečnej výslednej odmeny a taktiež odstraňuje chybu, ktorá môže byť spôsobená neistým dlhodobým plánovaním.

V prípade, že agent pozná model prostredia t.j. pozná výsledky akcií a_t zo stavu s_t, môže plánovať. Je dôležité si všimnúť, že nie každé prostredie je deterministické a niekedy môže byť dynamika prostredia pre agenta úplne neznáma, napr. pri hrách s náhodou. V hre Pac-Man to avšak neplatí.

Základná iteratívna metoda na riešenie RL problémov, s prostrediami v ktorých poznáme ich model, spočíva v týchto krokoch [25]:

1. Nauč sa hodnoty $v_\pi(s)$ pre každý stav.

2. Na základe $v_\pi(s)$ zlepši stratégiu pre všetky stavy a to tak, že nová stratégia zakaždým zvolí akciu a, ktorá maximalizuje súčet $R + v_\pi(s')$, kde R je bezprostredná odmena a s' je stav, ktorý vznikne ak aplikujeme a na s.

3. Opakuj predošlé kroky až pokiaľ obe funkcie neskonvergujú.

V reálnych problémoch je počet stavov obrovský, čiže funkcie spomenuté vyššie je nutné aproximovať. Na to sa dajú využiť metódy hlbokého učenia, ktoré s dostatočnou veľkosťou neurónovej siete dokážu aproximovať akúkoľvek funkciu [26]. V kombinácií s RL, ktoré dokáže pri hrách so simulátorom generovať „nekonečne“ veľa dat, sa jedná o veľmi silnú a obecnú metódu. Tomu nasvedčujú aj výsledky výskumu za posledné roky, kedy sa spoločnosti DeepMind podarilo vytvoriť obecnú metódu AlphaZero, založenú na tejto kombinácií a stromovom prehľadávaní [27]. Tá dosiahla nadľudských výsledkov v hrách go, šach a šógi pričom vo všetkých troch prípadoch mala k dispozícií len pravidlá hry a učila sa od nuly.

Ďalším problémom môže byť oneskorená odmena; napr. pri hrách kde agent dostane odmenu až na konci, čo môže byť 50 akcií potom, čo vykonal prvú akciu. Vďaka tomu a taktiež faktu, že agent v týchto metodách začína s náhodnou stratégiou, dokážu byť metody v kombinácií s hlbokým učením veľmi výpočetne náročné pri tréningu. Ako príklad sa dá uviesť spomínaná metóda AlphaGoZero [4], ktorá bola trénovaná po dobu približne 40 dní na silnom hardware a dokopy odohrala približne 29 miliónov hier samej proti sebe.

Základný problém, ktorému agent čelí je taktiež explorácia vs vykonávanie dobrých akcií, ktoré sa už naučil. Podobnú dilemu riešia aj ľudia v každodennom živote, napr. pri výbere jedla v reštaurácií: „Mám si vybrať moje obľúbené jedlo, ktoré som mal už mnohokrát alebo skúsim novú dennú špecialitu, ktorá môže byť ešte lepšia?“ Medzi týmito dvoma aspektmi je nutné rozumne balansovať.

2.5.3.1 SARSA

V rámci práce [5] autori použili metódu SARSA, ktorá sa učí tzv. Q hodnoty (očakávanú výslednú odmenu) stavov a ich príslušných akcií, ktoré agent následne používa na správanie v prostredí. Vzhľadom na vysoký počet stavov v hre Ms. Pac-Man použili abstraktnejšiu a kompaktnejšiu reprezentáciu stavu. Každý stav bol opísaný číselným vektorom s dimenziou 10:

- Prvé 4 čísla predstavovali pravdivostné príznaky s oborom hodnôt buď 0, v prípade nepravdy, alebo 1. Reprezentovali prítomnosť stien v bezprostrednej blízkosti pacmana v 4 smeroch: hore, vľavo, dole a vpravo.

Udalosť	Odmena	Popis
Krok	-0.5	Pacman sa pohol na prázdne políčko
Prehra	-35	Pacman bol zjedený duchom
Stena	-100	Pacman narazil do steny
Duch	+1.2	Pacman zjedol s vylepšením ducha
Pacdot	+1.2	Pacman zjedol pacdot

Tabuľka 2.1: Odmeňovacia schéma pre rôzne udalosti hry [5]

- Piate číslo predstavovalo preferovanú orientáciu pacmana, ktorá je nastavovaná na základe vopred definovanej logiky, pričom bere v úvahu rôzne situácie hry. Obor hodnôt bol v tomto prípade rovný 4 číslom v rozmedzí 0 až 3 (pre každý smer jedno).

- Ďalšie 4 čísla boli znova pravdivostné príznaky, pre každý smer jedno. Sú zapnuté ak sa z daného smeru blíži k pacmanovi duch, ktorý je bližšie než 5 políčok.

- Posledný príznak bol taktiež pravdivostný a bol zapnutý v prípade, že sa pacman nachádza v uličke ale neexistuje preňho bezpečný východ, inými slovami ak je zo všetkých strán obklopený duchmi.

Je jednoduché si uvedomiť, že počet možných stavov je len $2^9 * 4 = 2048$. Tento fakt im dovolil vytvoriť RL agenta, ktorý je výpočetne veľmi efektívny na učenie, keďže si dokáže uložiť všetky stavy do pamäte (napr. pomocou haš tabuľky) a následne im aktualizovať Q hodnoty bez nutnosti vytvárať obecnú aproximáciu (tá by bola potrebná ak by sme napr. nedokázali prehľadať všetky stavy, poprípade v situácií ak by počet stavov prevyšoval dostupnú pamäť).

Odmeňovacia funkciu je možné vidieť na tabuľke 2.1.

V rámci RL terminológie je epizóda považovaná za sériu postupností stavov, akcií a odmien, ktorá je zakončená konečným stavom. V tomto prípade teda epizóda predstavuje jednu odohranú hru. Je pozoruhodné, že tento agent dokázal skonvergovať so svojou stratégiou (z pohľadu SARSA algoritmu) už po 800 epizódach. Metodu otestovali taktiež na nevídaných mapách a vo všetkých prípadoch dosiahli kvalitné výsledky, čo nasvedčuje robustnosti použitej metódy.

2.5.3.2 AlphaZero

'= AlphaZero predstavuje silnú a taktiež obecnú RL metódu, ktorá na to, aby sa naučila hrať široké spektrum hier, potrebuje vedieť iba ich pravidlá.

Základným problémom pri prehľadávaní herného stromu je veľký vetviaci faktor, kvôli ktorému strom rastie do hĺbky exponenciálnou rýchlosťou a potreba robustnej evaluačnej funkcie. Ak by sme mali k dispozícií našeptávaciu funkciu, ktorá by v každom stave vrátila zoznam silných ťahov a funkciu,

ktorá by vedela predikovať výsledok pre ľubovoľný stav hry, mohli by sme výrazne zmenšiť šírku prehľadavaného stromu a taktiež ukončiť prehľadávanie v ľubovoľnej hĺbke. Aproximácie týchto dvoch funkcií sa snaží metoda AlphaZero naučiť pri hraní hier samej so sebou, vďaka čomu môže následne vykonávať efektívne stromové prehľadávanie [27]. V následujúcich odstavcoch opíšem základné chovanie algoritmu.

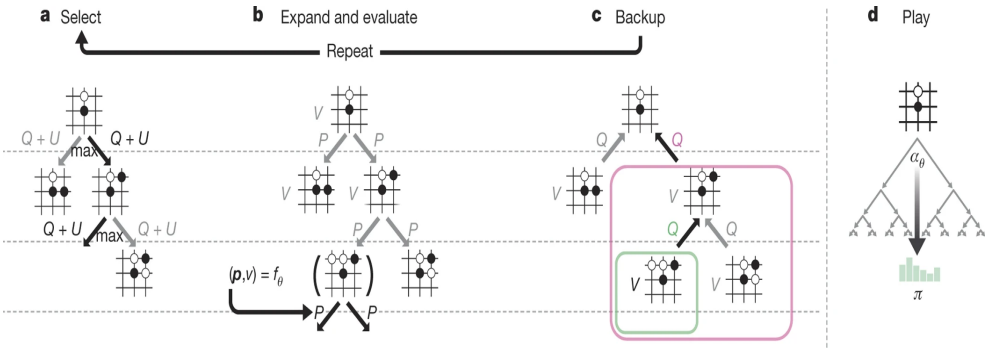
Pri výbere ťahu AlphaZero niekoľkokrát spustí iteráciu MCTS[3] z aktuálneho stavu. Každý stav si drží následujúce štatistiky:

- $N(s)$: Celkový počet návštev stavu s vrámci MCTS iterácií.
- $N(s,a)$: Celkový počet situácií, kedy vrámci SELECT fáze vykonalo akciu a zo stavu s.
- $W(s,a)$: Súčet spätne propagovaných evaluácií pre akciu a zo stavu s.
- $Q(s,a)$: Tzv. akčná hodnota, ktorá je rovná $W(s,a)/N(s,a)$.
- $P(s,a)$: Pravdepodobnosť akcie a v stave s.

Každá AlphaZero simulácia [27] začína v koreňovom stave s_0 a končí ak narazí na posledný uzol herného stromu s_L v čase L. V každom kroku $t < L$ volí v rámci fáze SELECT akciu, ktorá maximalizuje rovnicu $a_t = \text{argmax}_a(Q(s_t,a) + U(s_t,a))$, kde $U(s,a)$ je rovné $U(s,a) = C(s) * P(s,a) * \sqrt{N(s)}/(1+N(s,a))$. $C(s)$ predstavuje exploračný pomer a je rovný $C(s) = \log(1 + N(s) + c_{base})/c_{base} + c_{init}$. AlphaZero následne expanduje koncový uzol s_L a vyhodnotí pomocou neurónovej siete $(\mathbf{p}, v) = f_\theta(s_L)$, kde θ predstavuje jej aktuálne parametre, $\mathbf{p}$ pravdepodobnostný vektor akcií a v je rovná očakávanej evaluácii stavu s_L. Následne nainicializuje jeho štatistiky $\{N(s_L) = 0, N(s_L,a) = 0, W(s_L,a) = 0, Q(s_L,a) = 0, P(s_L,a) = p_a\}$, kde p_a označuje pravdepodobnosť akcie a vo vektore $\mathbf{p}$. Potom propaguje hodnotu v späť do koreňového stavu s_0 pričom aktualizuváva navštívené stavy $t \leq L$ následovne $\{N(s_t,a_t) = N(s_t,a_t) + 1, W(s_t,a_t) = W(s_t,a_t) + v, Q(s_t,a_t) = W(s_t,a_t)/N(s_t,a_t)$. Priebeh MCTS simulácie je možné vidieť na obrázku 2.3.

V každej globálnej iterácií AlpaZero (viď pseudokód 3) odohrá niekoľko epizód (viď pseudokód 2), z ktorých si uloží získané štatistiky do tzv. *replay_buffer*. Konkrétne, pre každý navštívený stav s ukladá trojicu $(s, \mathbf{p}, z)$, kde z reprezentuje finálny výsledok hry (napr. -1, 0 a +1, v prípade prehry, remízy a výhry), $\mathbf{p}$ predstavuje pravdepodobnostný vektor akcií, ktorý bol získaný MCTS prehľadávaním a pre každú akciu a je rovný $N(s,a)/N(s)$. Následne z *replay_buffer* vyberie rovnomerne náhodne určitý počet vzorkov, ktoré použije na tréning neurónovej siete. Pri tréningu sa snaží optimalizovať parametre θ, tak aby minimalizoval stratovú funkciu l, definovanou v rovnici

[3]Iterácia MCTS je v tejto práci označovaná ako simulácia, čo je rozdiel oproti typickému MCTS názvosloviu, kde simulácia predstavuje predposlednú fázu algoritmu.



Obr. 2.3: MCTS simulácia algoritmu AlphaZero [4]

Algorithm 2 AlphaZero epizóda

```
procedure AZ_EPISODE(state, game, model, num_sim)
    samples := empty_array()
    while game.is_not_finished(state) do
        mcts := MCTS()
        node := mcts.expand(state, model)
        add_dirichlet_noise(node)
        for each sim of num_sim do
            node := mcts.run_simulation(node, game, model)
        end for
        policy := sample_policy(node)
        samples.append(policy, state)
        state := game.execute_move(state, get_move(policy))
        return samples, evaluation(state)
    end while
end procedure
```

2.2, kde c predstavuje regularizačnú konštantu veľkosti váh. Priebeh epizódy a trénovania je možné vidieť na obrázku 2.4.

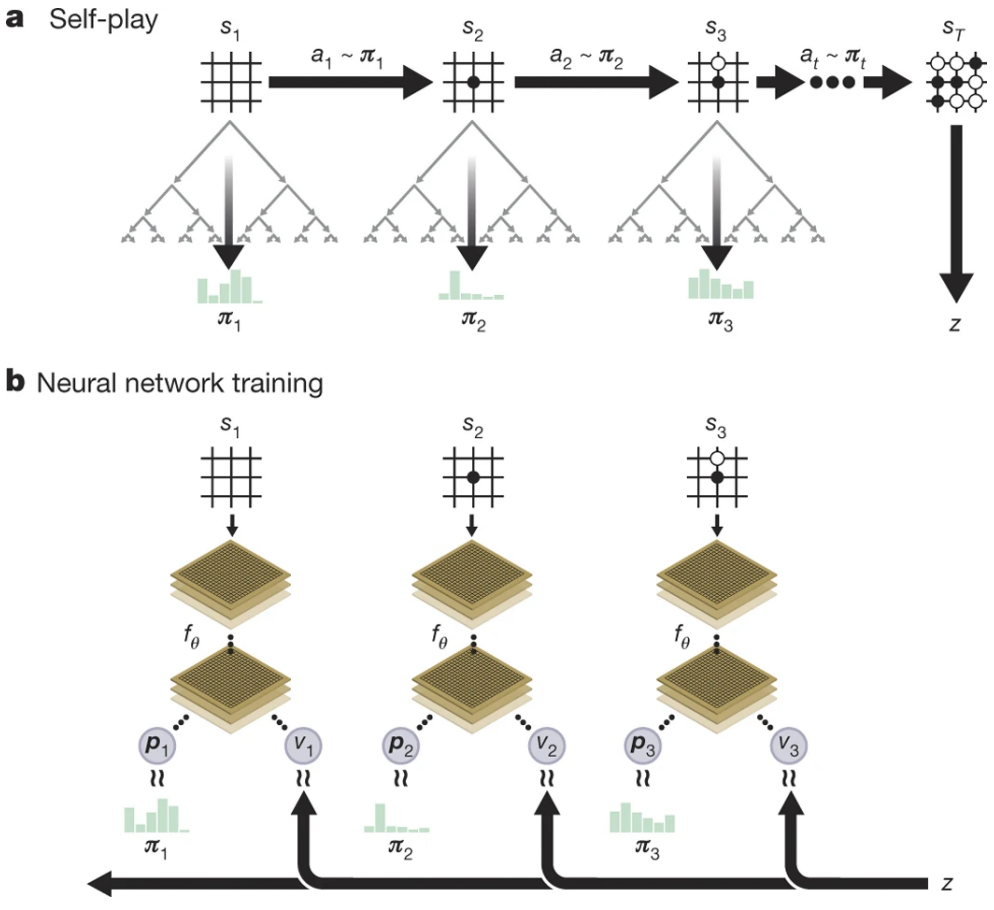
$$(\mathbf{p}, v) = f_\theta(s),\ l = (z\ - v)^2 - \boldsymbol{\pi}^T * \log \mathbf{p} + c * ||\theta||^2 \tag{2.2}$$

Pri tréningu navyše pridávali k pravdepodobnostiam ťahov taktiež dirichletov šum aby garantovali dostatočné množstvo explorácie počas celej fázy tréningu. Taktiež v prípade tréningu bolo niekoľko prvých ťahov vybraných náhodne, vzhľadom na získané pravdepodobnosti $\mathbf{p}$ zo stromového prehľadávania Monte-Carlo. Pri finálnej evaluácií algoritmu avšak vyberali akciu s najväčšou hodnotou $N(s, a)$.

Intuícia, prečo AlphaZero funguje je, že v každej iterácií agent zlepšuje odhady $\mathbf{p}$ pomocou prehľadávania a taktiež spresňuje odhady výsledkov hier

Algorithm 3 AlphaZero iterácia

procedure AZ_ITERATION(*init_nn_weights, game, num_iter, num_sim*)
  nn := initialize_neural_network(*init_nn_weights*)
  replay_buffer := empty_array()
  for each *iter* of *num_iter* **do**
    curr_samples := execute episodes for n games with *num_sim*
    replay_buffer.append(curr_samples)
    training_targets := get_random_samples(*replay_buffer*)
    nn.train(training_targets)
    discard old sampels from *replay_buffer*
  end for
end procedure



Obr. 2.4: AlphaZero epizóda a tréning [4]

v pri dodržiavaní aktuálnej stratégie, čo je princíp iteratívnej metódy spomenutej v úvode sekcie 2.5.3 - zlepšiť odhad výsledku a na základe toho zlepšiť stratégiu. Následne sa snaží tieto funkcie naučiť pomocou neurónovej siete, ktorá sa vo výsledku naučí generalizovať a stane sa teda použiteľnou aj na nevídané stavy.

Jednou z nevýhod počas tréningu (hlavne na jeho začiatku) je, že pre všetky stavy, vrámci epizódy, sa snaží metoda naučiť ten istý očakávaný výsledok z. Môže sa vňom skrývať teda priveľa šumu, hlavne pre úvodné stavy. Na stabilizáciu si teda vyžaduje vysoké množstvo výpočetného času. Efektívnejším riešením by mohlo byť použitie pred-trénovanej neurónovej siete na dostupných dátach (ak to je teda možné). Tento prístup bol použitý pri metóde AlphaGo [28], predchodca algoritmu AlphaZero, avšak nakoniec dosahoval horších výsledkov.

AlphaZero obsahuje taktiež niekoľko dôležitých hyperparametrov. Pri niektorých problémoch sa stačí riadiť predvolenými parametrami použitými vrámci práce [27]. Každopádne môže sa hodiť zjednodušiť problém (kvôli výpočetnej efektivite) a hyperparametre zoptimalizovať pomocou Metod ako Bayesovská optimalizácia [29].

2.5.4 Celulárne automaty

Celulárny automat je systém, ktorý pozostáva z priestorovo usporiadaných buniek, ktorú dokážu spolu interagovať a vymieňať si informácie, avšak len lokálne, t.j. medzi blízkymi susedmi. Aktualizácia stavu bunky je teda funkciou jej stavu a stavu susedov. Bunky sú aktualizované iteratívne v časových intervaloch.

Tieto systémy disponujú veľmi zaujímavou vlastnosťou. Aj pri malom počte jednoduchých aktualizačných pravidiel, dokážu vytvárať veľmi zaujímavé a komplexné správanie [30]. V rámci práce [30] sa autori snažili natrénovať pomerne malú neurónovú sieť, ktorá reprezentovala prekresľovacie pravidlo, pričom ich cieľ bol aby z jedného pixelu, po niekoľkých iteráciach, vznikol cieľový obrázok. Každá bunka reprezentovala svoj stav pomocou 16 reálnych čísel. Neurónová sieť mala teda na vstupe stav bunky a jej okolie a na výstupe nový stav bunky. Každá bunka používala tú istú neurónovú sieť a teda aj to isté pravidlo. Dosiahli veľmi zaujímavé výsledky, ktoré mali rôzne vlastnosti ako regeneráciu, relatívne inteligentnú kombináciu počiatočných buniek a stabilitu. V rámci práce vznikla taktiež aplikáciu, ktorá demonštruje toto správanie. Tá je dostupná na odkaze `https://distill.pub/2020/growing-ca/`.

Keďže tieto systémy pracujú iba s lokálnou informáciou, sú vysoko robustné voči zmenám na vstupe napr. voči zmene veľkosti prostredia. Toto je rozdiel oproti neurónovým sieťam, ktoré majú krehké predpoklady o fixne veľkom vstupe a výstupe [31]. V prípade multi-agentných systémov (do ktorých

spadá aj moja definícia hry Pac-Man), kde sa môže počet agentov meniť, sa javí táto metóda ako veľmi atraktívna.

V práci [32] autor aplikoval túto metódu na hru Ms. Pac-Man. Každá bunka si udržovala svoj stav a sadu pravidiel, ktoré určovali či bude dominovať susedné bunky (predávať na nich svoj stav) alebo nie. Bunky si držali taktiež váhu stavu, ktorá klesala so vzdialenosťou od zdrojovej bunky. Tento proces sa iteroval niekoľkokrát. Pacman sa následne rozhodoval len na základe stavov susedných buniek, ich váh a jeho cieľov.

2.6 Zhodnotenie opísaných metód

Pravidlami riadené systémy predstavujú interpretovateľné a častokrát výpočetne efektívne metody, avšak vyžadujú si expertnú znalosť problému a nedajú sa jednoducho preniesť na ďalšie hry, poprípade sú veľmi citlivé na rôzne zmeny hry. Metódy stromového prehľadávania sú na tom po obecnej stránke podstatne lepšie. Vyžadujú si ale viac výpočetného času a niekedy taktiež expertnú znalosť napr. v prípade alfa-beta a jej evaluačnej funkcie. Obecne pri nich platí, že s väčším výpočetným časom zlepšujú svoj odhad. Ako ďalšie som spomenul RL metódy. Tým vo väčšine prípadov postačí len simulátor prostredia, ktorý im v každom kroku ukazuje aktuálny stav prostredia a získanú bezprostrednú odmenu. Avšak aj tie dokážu byť v niektorých prípadoch veľmi výpočetne náročné. Na konci som taktiež spomenul zaujímavú metódu, založenú na celulárnych automatoch, ktoré si dokážu postačiť aj so zopár jednoduchými lokálnymi pravidlami. Tieto poznatky som využil pri návrhu implementovaných metód.

Kapitola 3

Implementované metódy

V rámci tejto kapitoly detailne opíšem implementované stratégie na ovládanie duchov a pacmanov, akým problémom som pri nich čelil a ako som sa snimi vysporiadal. Stratégie boli testované a optimalizované na 5 rôznorodých mapách s rozmermi 17x20 (šírka x výška), pričom pri niektorých mapách som viackrát zmenil počiatočnú polohu pacmanov a duchov. Medzi každými 2 políčkami, ktoré neobsahujú blok, existuje cesta. Jedna z máp predstavuje originálny level z hry Pac-Man avšak bez teleportov, ktoré sa nachádzali po stranách. Mapy som sa snažil navrhovať tak aby pokrývali pestré množstvo prostredí a situácií: krátke alebo dlhé uličky, viac uzavreté alebo otvorené mapy, asymetrické mapy, situácie kde pacmani začínajú v strede hry alebo na jej krajoch atp. Dokopy som teda používal 10 rôznych hier, ktoré sa líšili buď v mape alebo v počiatočnej pozícií objektov. Všetky počiatočné pozície a mapy používaných hier je možné nájsť v prílohe. Pacdots boli zakaždým rozmiestnené na všetkých voľných políčkach. Počet duchov som zvolil ako 4 a počet pacmanov ako 2. Tieto hodnoty som sa snažil navrhovať tak aby bola hra vybalancovaná. Maximálna dĺžka hier bola zväčša v stovkách krokov.

3.1 Originálna stratégia duchov

Jedná sa o poupravenú stratégiu, ktorú som popísal vrámci sekcie 2.5.1.2. Aj tu platilo, že duchovia sa nemohli len tak otočiť. Každopádne v niekoľkých aspektoch sa tieto stratégie líšili:

- Na začiatku hry sú všetci duchovia von zo svojho domčeka, pričom sú otočený v smere svojho scatter políčka.
- Duchovia nemôžu stáť v rovnaký čas na rovnakom políčko. Majú taktiež povolenú akciu stáť.
- Vzhľadom na to, že v zobecnenej verzii môže existovať viacero pacmanov si musia duchovia zakaždým vybrať, ktorého pacmana budú prenasle-

dovať v chase móde. Ich cieľ je rovný pacmanovi, ktorého priemerná vzdušná vzdialenosť, ku všetkým duchom, je najmenšia. Cieľ duchov sa nemení až pokiaľ daného pacmana nechytia alebo nenastane zmena módu, keďže to aby jedného pacmana chytili si vyžaduje určitú dávku koordinácie (napr. jeden duch sa musí dostať pred pacmana alebo za) a ak by častokrát striedali svoj cieľ tak by to pre nich mohol byť problém.

- Dĺžky scatter a chase modov boli poupravené vzhľadom na rozmery mapy. Scatter mode trvá dokopy 10 ťahov a chase mód 84 ťahov. Opakujú sa dokopy 3 krát. Potom duchovia nasleduje už len chase mód.

- Akcie „stáť“ a „krok späť“ sú povolené, avšak nie sú preferované. Môžu sa zísť ak by na mapách existovali slepé uličky.

- Keďže v zobecnenej definícií nie je povolené aby dvaja duchovia stáli na tom istom políčku, je nutné duchov pri výbere akcií usporiadať. Svoje akcie si následne volia postupne tak aby nedošlo ku žiadnej kolízií a to v tomto poradí: Blinky, Pinky, Inky a Clyde.

- Keďže uvažujem ťahovú alternatívu hry, tak sa duchom prirodzene nezvyšuje rýchlosť. V každom ťahu sa pohnú buď na susedné políčko alebo ostanú stáť.

- V pôvodnej implementácií sa duchovia vždy rozhodli, ktorým smerom sa z križovatky vydajú, už 1 políčko pred ňou. V mojej implementáciu si vyberajú smer až na križovatke.

- V prípade, že sa najbližšie vzdialenosti k cieľovému políčku rovnajú pre viacero susedných políčok, duchovia preferujú akcie v poradí pôvodnej implementácie a to hore, vľavo, dole, vpravo avšak rozširujú to ešte o akciu stáť.

- V rámci implementácie bolo potrebné vyriešiť taktiež zopár technických otázok ako napr. predvolená orientácia pacmana, ktorá bola vľavo, alebo orientácia pacmana po akcii stáť, ktorá sa rovnala jeho predošlej orientácií, ak bola definovaná, alebo predvolenej hodnote vľavo, atp.

Táto metóda predstavovala dobré zrovnávacie kritérium, pomocou ktorého som sledoval progres ostatných stratégií. Takto poupravená avšak nepredstavuje až tak silnú stratégiu, keďže napr. duchom sa nezväčšovala rýchlosť. Taktiež tomu napomáha fakt, že sa jedná o ťahovú hru, kde má hráč čas na rozmyslenie.

Scatter mód je pri tejto stratégií pomerne dôležitý, keďže sa častokrát stávalo, že duchovia po nejakej chvíli v móde chase, vytvorili „vláčik“, ktorý sa im vzhľadom na logiku výberu ďalšieho ťahu nedarilo roztrhnúť; špeciálne ak bol pacman hneď pred ním. Je jednoduché si rozmyslieť, že vláčik o dĺžke

4 pacmana nikdy nechytí, keďže mu nedokáže zablokovať východy v uličkách z oboch smerov.

3.2 Negamax s alfa-beta orezávaním a transpozičnou tabuľkou

Pri tejto metóde som použil optimálny stromový algoritmus negamax[4] (vzhľadom na to, že Pac-Man je hra s nulovým súčtom) s alfa-beta orezávaním a transpozičnou tabuľkou. Pseudokód algoritmu je viacmenej totožný so pseudokódom 1, avšak líši sa v niekoľkých aspektoch, ktoré popíšem v nasledujúcich odstavcoch. Hlavnou motiváciou na použitie tejto metódy bola jej jednoduchosť a potencionálne silná kvalita pri dobrej evaluačnej funkcii.

Prvá vec, ktorú som musel adresovať je časový limit stratégie. Problém pri negamax prehľadávaní je, že na nájdenie ťahu musí prehľadať celý strom do určitej hĺbky a nedá sa jednoducho vopred odhadnúť, ktorá hĺbka tomu zodpovedá. Taktiež v rôznych situáciach pri fixnej hĺbke môže vďaka alfa-beta orezávaniu prehľadávanie trvať kratšie alebo dlhšie. Základné riešenie na tento problém, ktoré som taktiež použil v mojej práci, je metóda iteratívneho prehĺbovania, ktorá postupne spúšťa celý algoritmus negamax s postupne sa zvyšujúcou hĺbkou prehľadavaného stromu. V prípade dosiahnutého časového limitu, vráti najlepší ťah z predošlej iterácie. Mohlo by sa zdať, že algoritmus zbytočne prehľadáva tie isté stavy dokola ale v kombinácií s ukladaním najlepších nájdených ťahov (vrámci celého herného stromu) z predošlých iterácií, môže byť aj táto metóda dosť efektívna a výrazne pomôcť pri alfa-beta orezávaní [33].

3.2.1 Znižovanie časovej komplexity

Každý objekt na mape má na výber až 5 akcií, hráči majú teda v prípade duchov k dispozícií až $5^4 = 625$ ťahov a v prípade pacmanov $5^2 = 25$. Teda aj pri malých hĺbkach je veľkosť prehľadavaného stromu priveľká. V rámci práce som musel teda implementovať rôzne vylepšenia, ktoré túto komplexitu dokázali znižovať.

Alfa-beta orezávanie funguje najlepšie ak sú silné ťahy prehľadávané ako prvé. Ich poradie je teda dôležité. V prípade duchov som volil ako prvé ťahy tie, ktoré končili v stave s najmenším množstvom pacmanov, potom ťahy s najmenšou priemernou vzdialenosťou duchov k **jednému pacmanovi** a nakoniec ťahy s najmenšou priemernou maximálnou vzdialenosťou duchov k **jednému pacmanovi**. Motiváciou bol fakt, že stavy s menším počtom pacmanov znamenajú, že duchovia pacmana zjedli, čo predstavuje skoro zakaždým dobrý

[4] V kóde a potenciálne v ďalších sekciách budem používať alfa-beta a negamax ako označenie pre tú istú metódu.

ťah. Stavy s malou priemernou vzdialenosťou k jednému pacmanovi znamenajú, že duchovia sú blízko a môžu pacmana potencionálne čoskoro dohoniť a zjesť. Stavy s malou maximálnou vzdialenosťou k jednému pacmanovi, znamenajú, že duchovia sú blízko ku všetkým pacmanom, čo je pre duchov taktiež pozitívne.

V prípade pacmanov som ťahy zoraďoval na základe podobných úvah, akurát z opačného konca: čo najväčší počet pacmanov, čo najväčšia minimálna priemerná vzdialenosť duchov k jednému pacmanovi, čo najmenší počet pacdots a čo najväčšia maximálna priemerná vzdialenosť duchov k jednému pacmanovi.

Vzhľadom na fakt, že hra Pac-Man obsahuje veľa cyklov, bolo nutné implementovať taktiež transpozičnú tabuľku, ktorá si navyše ukladala zakaždým doteraz najlepší nájdený ťah, ktorý bol pri ďalšom prehľadávaní vždy vyskúšaný ako prvý.

Počas prehľadávania herného stromu som si pre každý stav uložil najlepší nájdený ťah a ten som skúšal ako prvý ak som na daný stav narazil znova. Vstupom do transpozičnej tabuľky bol zahašovaný tzv. kanonický stav, ktorý obsahoval len pozície pacmanov, duchov a pacdots. Neobsahoval teda čítač ťahov a ani bloky, ktoré boli nemenné po celú dĺžky hry. Z pamäťových dôvodov som neukladal celý kanonický stav, avšak aby som sa vyhol niektorým potencionálnym kolíziám pri dotazovaní haš tabuľky, pridal som ku výstupným hodnotám taktiež 3 kontrolné čísla: počet pacmanov, počet pacdots a počet duchov. Tie sa po každom dotázaní kontrolovali a vprípade, že sa tieto informácie nezhodovali s hodnotami vo vstupnom kanonickom stave, tak sa vrátil prázdny výsledok. Motiváciou prečo nepoužívať číslo aktuálneho ťahu ako súčasť kanonického stavu bol fakt, že to sa mení pri každom ťahu a nemá nutne výrazný vplyv na zmenu stratégie hráča; dalo by sa dokonca hypotetizovať, že na začiatku hry na to nemá žiadny vplyv (pri dostatočne veľkej maximálnej dĺžky hry). Takto teda viacero reálnych stavov malo rovnakú kanonickú reprezentáciu a mohlo využívať predpočítané výsledky z transpozičnej tabuľky. Na rátanie hašu som používal Zobrist hašovanie, avšak bez efektívnej implementácie bezprostredného hašovania medzi následujúcimi stavmi.

3.2.2 Evaluačná funkcia

Aj keby by mal takto implementovaný negamax k dispozícií vysoké výpočetné kapacity, nedosahoval by dobrých výsledkov, ak by nepoužíval kvalitnú evaluačnú funkciu. Tú je možné reprezentovať rôzne, avšak na základe inšpirácie z práce [34] som zvolil jednoduchú lineárnu kombináciu predom definovaných n príznakov:

$$f(s,p) = \sum_{i=0}^{n-1} w_i * f_i(s,p) \tag{3.1}$$

Vzhľadom na fakt, že vážené príznaky jednoducho sčítavam, predpokladám

ich nezávislosť, ktorá nemusí nutne platiť. Lepšia forma evaluačnej funkcie môže niektoré príznaky kombinovať napr. násobiť ale to prichádza s nevýhodou vyššej komplexnosti, ktorá môže byť ťažšia na optimalizáciu a interpretáciu.

Príznaky, ktoré som v práci použil, vychádzali primárne z mojich znalostí o hre. Miestami som sa taktiež inšpiroval prácami spomenutými v kapitole 2.5. Optimalizáciu ich váh som opísal v nasledujúcej sekcii 3.2.3.

Väčšina príznakov bola normalizovaných tak aby ich obor hodnôt bol z intervalu $[0, 1]$. To bolo dôležité, keďže príznakom som nechcel zvyšovať váhu len kvôli škále ich hodnôt. Niektoré príznaky boli sčítavané podľa počtu objektov, pre ktoré platili. V prípade, že v danom stave nebolo možné príznak zrátať, nebral sa v úvahu pri rátaní finálnej evaluácie. Rôzne hraničné hodnoty, ktoré sa objavujú v definícií príznakov som vyberal na základe heuristických odhadov a manuálneho testovania. Ich zoznam je následovný[5]:

- *are_pacmans_about_to_die*: Počet pacmanov, ktoré majú garantované, že zomrú v tomto alebo ďalšom ťahu. Napr. ak je na ťahu duch a stojí vedľa pacmana alebo ak je na ťahu pacman a v každom možnom smere stojí vedľa neho duch.

- *is_nearest_ghost_close_to_one_pacman*: Príznak, ktorý má obor hodnôt z intervalu $[0, 1]$. V prípade, že sa najbližší duch nachádza k nejakému pacmanovi v menšej alebo rovnej vzdialenosti než 10, je tento príznak rovný normalizovanej obrátenej vzdialenosti[6], ktorá je definovaná následovne $max(0, min(1, 1 - (d - 1)/d_max))$, kde d predstavuje vzdialenosť dvoch objektov a d_{max} maximálnu vzdialenosť, pre ktorú sa príznak počíta; v tomto prípade platí $d_{max} = 10$. Inak sa tento príznak rovná 0. Túto funkciu budem v následujúcom texte skrátene označovať ako *rev_dist*.

- *is_second_nearest_ghost_close_to_one_pacman*: Definícia tohoto príznaku je totožná s *is_nearest_ghost_close_to_one_pacman*, avšak v úvahu sa bere druhý duch s minimálnou vzdialenosťou a $d_{max} = 20$.

- *is_min_avg_distance_of_ghosts_to_one_pacman_small*: *rev_dist* minimálnej priemernej vzdialenosti duchov k jednému pacmanovi s $d_{max} = 20$.

- *is_max_avg_distance_of_ghosts_to_one_pacman_small*: *rev_dist* maximálnej priemernej vzdialenosti duchov k jednému pacmanovi s $d_{max} = 20$.

- *are_pacmans_close_to_nearest_pacdot*: Priemer normalizovaných obrátených vzdialeností všetkých pacmanov k ich najbližšej pacdot s $d_{max} = 40$. Príznak slúži primárne na motiváciu pacmanov ísť taktiež k odľahlejším pacdots.

[5]Pre konzistenciu s názvoslovím v kóde a v priložených materiáloch, používam jednotné názvy v angličtine.

[6]Funkcia má obor hodnôt z intervalu $[0, 1]$, pričom maximálnej hodnoty nadobúda ak objekty na sebe stoja alebo sú bezprostredne vedľa seba. Minimálna hodnota nastáva naopak v prípade, že sú objekty od seba príliš vzdialené.

- *are_pacmans_close_to_each_other*: *rev_dist* priemernej vzdialenosti všetkých unikátnych dvojíc pacmanov s $d_{max} = 12$. To, že sú pacmani blízko seba napomáha duchom, keďže nemusia rozdeľovať svoje sily naprieč celou mapou.

- *are_ghosts_close_to_each_other*: Počet všetkých unikátnych dvojíc duchov, ktorý stoja vedľa seba na susednom políčku. Pri tomto príznaku som vychádzal z pozorovania, že ak sú duchovia veľmi blízko seba, častokrát majú problém uväzniť v pacmana nejakej uličke, keďže na to aby ho uväznili musia do danej uličky prichádzať z rôznych smerov.

- *are_all_pacmans_dead*: Príznak je rovný 1 ak sú všetci pacmani zjedený. Inak je rovný 0. Tento príznak slúži primárne na normalizáciu evaluácie finálneho stavu hry, v ktorom nie je možné zrátať množstvo príznakov, čo môže spôsobovať rôzne nepomery evaluačných hodnôt medzi koncovými a nekoncovými stavmi.

- *score*: Počet zjedených pacdots.

- *relative_pacmans_count*: Normalizovaný počet pacmanov. Príznak je rovný podielu aktuálneho počtu pacmanov a počtu pacmanov na začiatku hry.

- *are_pacmans_in_alley_and_in_danger*: Počet pacmanov, ktorý sú v dlhšej uličke t.j. ich vzdialenosť k najbližšej križovatke je aspoň 2 a zároveň sú v „ohrození“, čo znamená ich vzdialenosť ku najbližšiemu duchovi je menšia alebo rovná 7. Idea tohoto príznaku spočíva v tom, že ak pacman vchádza do dlhšej uličky, je to preňho riskantná situácia lebo mu bude trvať dlhší čas znej vyjsť, čoho môžu využiť duchovia a východy zablokovať.

- *are_pacmans_close_to_safe_pacdots*: Počet pacmanov, ktorý majú vo vzdialenosti 3 aspoň jednu pacdot, na ktorú sa vedia dostať skôr, než akýkoľvek duch.

- *can_pacmans_safely_reach_crossroad*: Počet pacmanov, ktorý sa dokážu dostať k ich najbližším križovatkám bez toho, aby boli zjedený duchmi. Podľa výsledkov evaluácie sa jedná o veľmi dôležitý príznak, čo sedí s mojou intuíciou, keďže príznak reprezentuje situáciu, kedy je pacman uväznený v uličke a nedokáže z nej bezpečne vyjsť.

- *are_pacmans_not_in_close_danger*: Počet pacmanov, ktorý sa vedia dostať aspoň na jedno políčko vo vzdialenosti presne 5 skôr, než akýkoľvek duch. V prípade, že to pre nejakého pacmana neplatí, tak to znova môže predstavovať zablokovanú situáciu, z ktorej sa nebude môcť bezpečne dostať von.

3.2.3 Optimalizácia parametrov

V predošlej sekcii som predstavil niekoľko príznakov, ktoré používam pri evaluácií stavov, avšak každý znich je spojený s nejakou váhou, ktorú treba rozumne nastaviť, inak metóda nebude vôbec fungovať. Prvotnú konfiguráciu týchto parametrov som zvolil pomocou rôznych odhadov a následne ladil. Prekvapivo, už po chvíli ladenia, stratégia začala dosahovať kvalitných výsledkov a niekedy bola výzvou aj pre ľudského hráča. To vyzeralo sľubne ale stále som pozoroval zopár situácií, ktoré sa zdali byť nelogickými, napr. v prípade, že boli pacmani pod väčším tlakom a v ďalších ťahoch ich čakala jasná smrť, tak sa niekedy nerozptýlili, čo by malo za následok mierne kratšie prežitie atp.

Pri optimalizačnom prístupe som sa inšpiroval metódou, ktorú použili autori vrámci práce AlphaStar [7]. Počas tréningu si udržiavali ligu agentov s rôznorodými stratégiami, ktorú postupne zlepšovali. Hlavný agenti boli optimalizovaný voči celej lige naraz. Týmto sa snažili adresovať problém cyklov, ktorý sa môže pri učení stratégií v RL objaviť. Ten je zrejmý napríklad pri hre kameň, papier, nožnice, kde stratégia hraj kameň začne prehrávať voči stratégií hraj papier; agent sa následne naučí hrať nožnice ale týmto začne znova prehrávať voči stratégií hraj kameň s ktorou predtým remizoval. Ak by tento problém neadresovali, agent by dokola menil stratégie ale obecne sa nemusel zlepšovať, keďže by mohol zabúdať to, čo sa naučil predtým.

V lige existovali rôzne typy agentov. Pár znich boli hlavnými, ktorých sa snažili naučiť stratégie, ktoré fungovali proti celej lige a následne kontinuálne zlepšovať. Ďalším typom agentov boli tzv. „exploiters“, ktorých funkcia nespočívala vničom inom, než nájsť aktuálne slabiny hlavných agentov. Vďaka tomu boli hlavný agenti stále motivovaný adresovať svoje slabiny. V práci [7] autori použili taktiež ďalšie typy agentov ale tie pre stručnosť už nebudem opisovať.

V rámci optimalizácie som si podobne ako pri metóde AlphaStar držal ligu agentov avšak len s dvoma typmi: hlavný agenti a ich exploiters, pre každého hlavného agenta jeden. Optimalizácia fungovala na iteratívnom princípe. V rámci každej iterácie som sa snažil nájsť kombináciu váh, ktorá dosahovala najlepšie priemerné skóre proti celej lige. Agenta s týmito váhami som následne pridal do ligy spoločne s jeho exploiter-om, u ktorého som sa snažil nájsť najlepšiu kombináciu váh tak, aby nový hlavný agent voči nemu dosahoval čo najmenšie skóre. Liga mala fixnú veľkosť, pričom si udržovala len najnovších agentov. Bola inicializovaná originálnou stratégiou duchov a stratégiou negamax s heuristicky odhadnutými parametrami.

Vzhľadom na väčšiu výpočetnú náročnosť som pri finálnej optimalizácii použil len zopár hier a vzhľadom na podstatu optimalizačného problému (nederivovateľná stratová funkcia, vysoká výpočetná náročnosť pri testovaní kvality jednej kombinácií parametrov atp.), som ako optimalizačnú metódu zvolil Bayesovskú optimalizáciu [29], ktorá je na tento typ problémov stavaná. Iniciálnymi vzorkami v následujúcich iteráciach, v rámci Bayesovskej optima-

lizácie, bola posledná najlepšia kombinácia parametrov agenta a vzhľadom na fakt, že sa väčšinou jedná o dobrú kombináciu parametrov, môže toto malé vylepšenie urýchliť konvergenciu.

3.2.4 Zhrnutie

Negamax s transpozičnou tabuľkou, optimalizovanou evaluačnou funkciou a ďalšími spomenutými vylepšeniami dosahoval veľmi kvalitných výsledkov už aj pri malých hĺbkach prehľadavaného stromu, konkrétne s hĺbkou 2. Obecne, časová komplexita s väčšou hĺbkou rástla signifikantne a prakticky nebolo možné používať väčšiu hĺbku než 4 alebo 5. Inak sa na ťah čakalo v niektorých situáciach radovo v minutách. Počas optimalizácie som vzhľadom na veľkú výpočetnú náročnosť metody používal maximálnu hĺbku 2, veľkosť ligy 3, počet rôznych hier 7, pričom maximálna dĺžka hry bola rovná 300 ťahom. Počet vzoriek, ktoré sa vrámci Bayesovskej optimalizácie rátali bol rovný 100 pre hlavného agenta a 15 alebo 20 pre exploiter-a. S výpočetnými zdrojmi, ktoré som mal k dispozícií ,trvala jedna iterácia približne 15 až 18 hodín.

3.3 AlphaZero

Ďalšou metódou, ktorú som implementoval je AlphaZero opísaná v sekcii 2.5.3.2, ktorá bola aplikovaná na hry šach, go a šógi. Všetky tieto hry sú avšak do určitej miery symetrické, majú jasný výsledok (prehra, remíza alebo výhra) a v každom ťahu ovládajú iba jednu figúrku, čo je podstatný rozdiel oproti zobecnenej definícii hry Pac-Man.

V ďalších sekciách opíšem ako som tieto problémy adresoval. Stojí avšak za zmienku, že metóda AlphaZero obsahuje v kombinácií s hlbokým učením viacero dôležitých parametrov a obecne ju rozumne nakonfigurovať, môže vyžadovať väčšie množstvo experimentácie a výpočetných zdrojov. Každopádne, niektoré parametre som nastavoval podľa predvolených hodnôt práce AlphaZero [27].

3.3.1 Odmena

Prvým kľúčovým rozdielom bol rozdiel vo finálnej odmene, keďže výsledok hry Pac-Man je rovný skóre, ktoré sa pacman snaží maximalizovať. Hra teda nutne nemá víťaza a porazeného.

Finálny výsledok hry z je rovný pomeru počtu zjedených pacdots k počiatočnému množstvu pacdots. Jedná sa o číslo z intervalu $[-1, 1]$. Neurónová sieť sa v každom stave snaží predikovať predpokladaný relatívny počet zjedených pacdots avšak vzhľadom na aktuálny stav. T.j. ak sme na konci hry, v ktorej sa nachádzajú už iba 2 pacdots, tak ak je výsledok predikcie rovný -1 znamená to, že náš model neočakáva, že pacman zje nejaké pacdots, ak by bol rovný 0, model očakáva, že pacman zje práve jednu pacdot a ak by bol rovný 1 tak

všetky pacdots, čo je v tomto prípade 2. Výsledok predikcie sa teda stále viaže na vstup a namiesto tradičného intervalu $[0, 1]$ je normalizovaný na interval $[-1, 1]$.

Výsledná odmena, ktorá sa propaguje vrámci MCTS a počíta pri expandovaní listov herného stromu, je avšak rovná predpokladanému relatívnemu finálnemu skóre hry, ktoré je definované ako:

$$v_{s_L} = (d_{L,eaten} + d_{L,expected})/d_{initial}, \tag{3.2}$$

kde $d_{L,eaten}$ predstavuje počet zjedených pacdots v stave s_L, $d_{L,expected}$ predikciu neurónovej siete (NN) a $d_{initial}$ počet počiatočných pacdots v hre.

Tento prístup zlepšuje presnosť predikcie, keďže vrámci hlbších simulácií spresňuje odhad pomocou $d_{L,eaten}$ a taktiež zjednodušuje prácu NN, ktorá na to aby dokázala počítať finálne relatívne skóre, musela by mať na vstupe taktiež počiatočný počet pacdots a tento fakt sa pri tréningu naučiť.

Jednou z nevýhod pri AlpaZero, hlavne na začiatku tréningu je, že všetky stavy po skončení hry majú priradený ten istý výsledok z, ktorý sa následne používa pri tréningu NN. Ten môže byť výrazne skreslený pre úvodné stavy hry. Taktiež v trénovacich dátach je limitovaná variabilita vzhľadom na fakt, že z môže nadobúdať len 2 hodnôt, čo môže sťažovať tréning (napr. ak by v súčasnej iterácií boli samé výhry, tak všetky trénovacie vzorky by mali to isté z). Tento problém bol spomenutý taktiež v práci [35]. Môj prístup ho do určitej miery adresuje, keďže finálne z sú rátané pre každý stav samostatne a sú rovné:

$$z_s = (d_s - d_{last})/d_s, \tag{3.3}$$

kde d_s predstavuje počet pacdots v stave s a d_{last} počet pacdots v poslednom stave.

Pre zefektívnenie predikcie a výkonu stratégie som implementoval taktiež pravidlá ako, ak sú pacmani mŕtvy alebo všetky pacdots sú zjedené, $d_{L,expected}$ je vždy rovné nule atp.

Predikované hodnoty boli vždy z pohľadu pacmana. Vzhľadom na to, že sa jedná o hru s nulovým súčtom, duchom som priraďoval obrátenú hodnotu $1 - v$. Takto bola propagovaná v stále z intervalu $[0, 1]$ pre pacmanov a taktiež pre duchov.

3.3.2 Transpozičná tabuľka

Hra Pac-Man obsahuje množstvo cyklov a opakovaných stavov. Pre efektívne prehľadávanie herného stromu je veľmi výhodné použiť transpozičnú tabuľku (ďalej ako TT), do ktorej sú ukladané medzi-výpočty pre ekvivalentné stavy. V prípade, že pri prehľadávaní herného stromu narazíme na nejaký stav, najprv sa pozrieme či sa nenachádza v TT, ak áno, použijeme uložený medzivýsledok, ak nie vyhodnotíme stav a uložíme ho do TT. V práci [36] autori analyzovali rôzne stratégie používania TT vrámci MCTS. Jeden z problémov,

ktorý sa pri tejto kombinácií objavuje je, že s TT môže mať uzol vrámci MCTS viacero rodičov. Otázka teda je, ako a ktorým rodičom bude propagovaná v vrámci poslednej fáze BACKPROPAGATION algoritmu MCTS. V tejto práci autori rozobrali viacero prístupov, avšak aj jednoduchá stratégia ako: propaguj v rodičovi, ktorý si ťa zvolil pri fáze SELECT, dosahovala lepších výsledkov. Túto stratégiu som taktiež použil v mojej práci.

Vstupom do TT bol v prípade MCTS celý kanonický stav aj napriek väčším pamäťovými nárokom (oproti alfa-beta, kde sa ako kľúč ukladal len haš), keďže kolízia by mohla výrazne narušiť kvalitu algoritmu pri učení. Ukladané hodnoty predstavovali MCTS uzly so všetkými štatistikami ako $Ns, N_{s,a}, ...$, pričom ukladané prechody obsahovali kanonickú formu ťahu, ktorá bola definovaná ako zoznam dvojíc: pozícia a pridelená akcia.

Pri tomto používaní TT môže dôjsť ku nekonečným cyklom napr. v prípade kedy v rámci fáze SELECT uzol A zvolí prechod do B a B následne zvolí prechod do A. Aby som tento problém vyriešil, pri každej MCTS simulácií si ukladám pri každom navštívenom stave príznak *visited*, ktorý si drží informáciu o tom, či som daný uzol navštívil. Ak pri fáze SELECT narazím už na navštívený uzol s, fázu SELECT zastavím a začnem propagovať jeho hodnotu v, ktorá bola spočítaná pri jeho expanzii, späť do koreňa. Následne príznaky *visited* pre každý navštívený stav vyčistím a znova spúšťam MCTS simuláciu. Takto vrámci cyklu propagujem najhlbšiu hodnotu, čo mi prišlo ako rozumné riešenie.

Aby som zachoval level explorácie pri tréningu, z TT na začiatku každého ťahu vymažem všetky hodnoty.

3.3.3 Dirichletov šum

Pre garantovanie dostatočnej explorácie počas tréningu, je na koreňové stavy aplikovaný Dirichletov šum, ktorý je parametrizovaný parametrom α. V práci [27] používali rôzne parametre α pre rôzne hry, avšak tie boli fixné pre každý stav. Platí, že ak sme v hre, ktorá má v priemere väčšie množstvo ťahov, tak vzhľadom na konečné výpočetné zdroje, je možné prehľadať do určitej hĺbky iba zopár znich. V tomto prípade je α menšie. Naopak ak sme v stave s menším počtom dostupných ťahov, je možné nastaviť α na väčšiu hodnotu, čo ma za následok viac uniformný šum a teda väčšiu exploráciu.

Vzhľadom na podstatný rozdiel v počte ťahov medzi duchmi a pacmanmi som namiesto fixnej α používal funkciu, ktorej vstupom bol počet validných ťahov v pozícií a jej formu som odhadol pomocou exponenciálnej regresie a rôznych parametrov α, ktoré boli aplikované v práci [27] postupne na hry šach, go a šógi. Jej výsledný tvar bol:

$$alpha(x) = max(0.01, 0.3902 * (0.9897 * *x), \tag{3.4}$$

kde x predstavoval počet validných ťahov v aktuálnom stave.

3.3.4 MCTS

Bezprostredné spätné ťahy vrámci MCTS simulácií som na základe práce [3] zakázal aby som dosiahol väčších rozdielov propagovaných hodnôt. To je špeciálne dôležité pri hrách s veľkým množstvom cyklov a nevýraznými ťahmi (jeden ťah nemá veľmi výrazný efekt na koniec hry). Kvôli väčšej variabilite a jednoduchšiemu učeniu som zakázal ťahy „stáť", ktoré vo väčšine prípadov agent tak či tak nepoužíva.

Pri evaluácií používam časový limit a to musel MCTS zohľadniť. V tomto prípade sa jednalo o triviálnu úpravu, MCTS pokračuje v MCTS simuláciach pokiaľ mu nedôjde čas.

Počas tréningu som použil počet simulácií rovný 300, kvôli obmedzeným výpočetným zdrojom. V originálnej práci AlphaZero [27] to je avšak 800. Jednoznačne najužším hrdlom pri výpočte simulácie bol predikovaný výstup neurónovej siete. Konštanty c_init som používal rôzne pre pacmanov a duchov, kvôli rozdielnemu počtu ťahov. Po manuálnom ladení sa rozumnými hodnotami zdali byť 2 pre pacmanov a 2.25 pre duchov.

AlphaZero počas tréningu kvôli vyššej explorácií vykonáva n prvých ťahov náhodne, proporčne k ich $N_{s,a}$. V práci autori použili $n = 30$. Vzhľadom na vyšší počet ťahov v hre Pac-Man som použil $n = 45$.

V prípade prvej návšteve uzlu, sa ťahy vyberajú len na základe ich pravdepodobností, keďže hodnoty $Q_{s,a}$ a N_s (a teda aj $U_{s,a}$) sú všetky rovné 0.

3.3.5 Architektúra neurónovej siete

Hry ako šach, go a šógi majú do určitej miery symetrické stratégie. NN sa teda potrebuje naučiť hrať hru iba z pohľadu jedného hráča a pri predikcii pre rôznych hráčov „invertujeme farby ich figúrok". V práci AlphaZero [27] mala NN pri každej hre jednu hlavu pre v a jednu pre $\mathbf{p}$. Avšak v hre Pac-Man majú duchovia a pacmani výrazne odlišné stratégie. Z tohoto dôvodu som pridal ešte jednu hlavu pre duchov. Dokopy výstup NN reprezentoval trojicu $(\mathbf{p}_p, \mathbf{p}_g, v) = f_\theta(s_L)$. Pri tomto návrhu som vychádzal z práce [35]. To si vyžadovalo zbierať trénovacie vzorky pre oboch hráčov v každom stave zároveň (nezávisle na tom, ktorý hráč je na ťahu), čo zdvojnásobovalo časovú komplexitu ťahu.

NN architektúra bola rovnaká ako v prípade AlphaZero až na hlavy pre stratégie pacmanov a duchov. Jednalo sa o konvolučnú sieť, ktorej hlavné telo pozostávalo z 19 reziduálnych blokov. Základný problém, ktorý som pri návrhu riešil, je ovládanie rôzneho počtu agentov. Reprezentácia vstupu a výstupu musela byť dostatočne malá, aby som NN dokázal natrénovať a ideálne taktiež odolná voči permutáciám pozíc agentov na vstupe. Reprezentovať výstup ako označenie štartovacieho a finálneho políčka (chyť figúrku a polož ju na iné políčko), nebolo vzhľadom na počet agentov prakticky možné, keďže veľkosť

výstupu s počtom agentov rástla enormne rýchlo, čo prakticky znemožňovalo tréning NN.

Výstup pre stratégie som reprezentoval následovne. Pre každé políčko som predikoval vektor pravdepodobností, ktorých súčet bol 1. Dĺžka vektoru bola rovná celkovému počtu akcií jedného agenta, v mojom prípade teda 4: hore, dole, vľavo a vpravo (akciu stáť som neuvažoval). Súčasťou vstupu teda nebolo aktuálne číslo ťahov z podobných dôvodov, ktoré som spomenul v predošlých sekciách a taktiež kvôli jednodušiemu učeniu. Pravdepodobnosť zdieľaného ťahu **a** medzi všetkými agentmi som rátal následovne:

$$\pi(\mathbf{a}|s) = \Pi_{g \in grids} \pi_g(a_g|s) + e, \tag{3.5}$$

pričom g predstavuje políčka, kde stoja agenti aktuálneho hráča, π_g reprezentuje vektor pravdepodobností políčka g a a_g akciu a e predstavuje malý prídavok 0.001, ktorý garantoval, že každý ťah bude mať nenulovú pravdepodobnosť (z exploračných dôvodov). Ťahy boli pred používaním vždy normalizované aby súčet validných ťahov v danom kroku bol rovný 1. Pri tejto reprezentácií som vychádzal z práce [37], kde rovnaký prístup použili na ovládanie viacero agentov pri hre StarCraft II.

Základnou nevýhodou tohoto prístupu je, že predpokladá nezávislosť ťahov medzi rôznymi políčkami, čo rozhodne neplatí. Avšak napriek tomu autori dosahovali sľubných výsledkov. Je dobré si uvedomiť, že veľkosť výstupu je pri tejto reprezentácií striktne viazaná iba na veľkosť mapy a počet akcií, čo znamená, že toto riešenie sa dá použiť na ľubovoľný počet agentov, ktorý sa na mapu zmestí. Taktiež je odolné voči permutáciám pozíc agentov na vstupe, čo je ďalšie veľké plus.

Pri reprezentácií vstup som vychádzal z práce AlphaZero. Reprezentoval som ho ako sériu matíc, každá mala priradené rozmery mapy. Matice boli binárne a boli priradené konkrétnym typom objektov. Na pozícii (i, j) mali 1 v prípade, že sa na danej pozícií nachádzal daný typ objektu, napr. pacman. V opačnom prípade tam bola 0. Používal som jednu maticu pre každý typ objektu: duch, pacman, pacdot a blok. Ku vstupu som ešte pridal maticu, ktorá predstavovala aktuálneho hráča na ťahu. Bola rovná samým 1-tkám ak bol na ťahu pacman, inak bola rovná všade 0. Vstup mal teda rozmery 17x20x5.

Stojí za zmienku, že kombinácia konvolučných NN a hrách, ktoré sa hrajú na tabuľkových mapách, dokáže efektívne prenášať znalostí naprieč rôznymi časťami mapy, keďže ten istý filter sa aplikuváva postupne naprieč celou vstupnou maticou a to čo sa NN naučila v ľavom hornom rohu, bude platiť aj pre pravý dolný roh ak je teda rovnaký.

Konkréte hlava výstupu pre stratégiu pozostávala z jedného reziduálneho bloku, ktorý bol použitý v tele NN, následne konvolučnej vrstvy so 128 filtrami s rozmermi 1x1 a krokom 1, po stranách bola doplnená 0 aby sa zachovali rozmery vstupu a výstupu, nasledovala dávková normalizačná vrstva,

ReLU a finálna konvolučná vrstva so 4 filtrami (jeden pre každý akciu) s rozmermi 1x1, krokom 1, znova po stranách doplnená 0 aby sa zachovali rozmeri vstupu a výstupu a s aktivačnou vrstvou softmax, ktorá normalizovala finálne hodnoty na pravdepodobnostné vektory pre každú bunku mapy (jednalo sa o poupravenú architektúru, ktorá bola použitá vrámci práce [37]).

Architektúra pre výstup predikovaného výsledku bola totožná ako v práci AlphaZero.

Počet filtrov, ktoré boli použité v tele NN a taktiež v prvej vrstve medzi vstupom a telom, som kvôli výpočetným zdrojom znížil z 256 (ktoré boli použité v AlphaZero) na 128. Stratovú funkcia som poupravil aby obsahovala člen $-\boldsymbol{\pi}^T * \log \mathbf{p}$ pre pacmana aj duchov. Očakávané pravdepodobnosti buniek, ktoré sa používali pri tréningu, boli rátané z marginálnych pravdepodobností ťahov, ktoré vypadli pri MCTS. Počet epoch som zvolil ako 2 a mieru učenia ako 0.002. Tieto hodnoty boli odhadnuté manuálnymi ladením, avšak určite je pri nich priestor na zlepšenie. Ostatné parametre boli prevzaté z predvolených nastavení AlpaZero.

3.3.6 Tréning

Pri tréningu AlphaZero si metóda udržovala v tzv. „replay buffer“ všetky hry za posledných 20 hier. Tento a podobné parametre som nemenil. Avšak vrámci jednej iterácie moja verzia odohrá len 30 hier (paralelne), následne si náhodne z „replay buffer“ vyberie 4096 vzorkov, ktoré použije na tréning NN. Jedná sa teda o synchronný proces. Počet vzorkov tzv. „batch size“, z ktorých sa rátal gradient pri tréningu NN bol rovný 32.

Pri tréningu AlphaZero som poupravil počiatočné pozície tak, aby boli viac dynamické a nestabilné. Pacmanov a duchov som dal bližšie k sebe. Vychádzal som z hypotézy, že takto sa pacmani a duchovia dokážu učiť a objavovať silnejšie ťahy skôr, než v prípade kedy by im len trvalo x krokov pokiaľ by sa k sebe na mape priblížili. Maximálnu dĺžku hry som skrátil na 125. Využíval som dokopy všetky mapy a na niektorých som párkrát zmenil počiatočné pozície. Výsledný použitý počet hier bol 10. Každú hru som vrámci iterácie odohral 3 krát.

Počas hrania hier sa častokrát opakovali ťahy, avšak vďaka náhodným ťahom a Dirichletovému šumu mali vo výsledku iné cieľové hodnoty, ktoré som sa následne snažil NN naučiť. Je jednoduché si uvedomiť, že optimálna predikcia v tomto prípade je priemer týchto hodnôt. Aby som teda zefektívnil tréning tak som rovnaké ťahy spájal o jedného a ich trénovacie ciele sprieme-roval.

3.3.7 Zhrnutie

Pri prvotnej implementácií bez väčších modifikácií sa táto metóda mala veľký problém niečo naučiť. To som sa snažil adresovať rôznymi vylepšeniami, ktoré

som vrámci sekcie spomínal avšak nič ztoho nemalo výrazný vplyv na výsledok. Bohužiaľ vzhľadom na časové možnosti a časovú komplexitu experimentácie, som tento prístup nestihol vylepšiť. V nasledujúcich odstavcoch opíšem čo podľa mňa včom spočíval primárny problém, spomeniem finálny experiment, ktorý predstavoval svetielko nádeje pri učení a sekciu zakončím možnými zlepšeniami, ktoré verím, že by túto metódu mohli dotiahnuť k rozumným výsledkom.

Hypotetizoval som, že základným problémom, ku ktorému dochádzalo, bolo, že po určitej iterácií sa hodnoty ťahov $Q_{s,a}$ začali viacmenej rovnať. Pacmanovi sa teda zdal každý ťah rovnaký silný, čo je pri prehľadávaní do malej hĺbky viacmenej pravda. Na to aby sa naučil skutočné hodnoty $Q_{s,a}$ a začal medzi ťahmi detekovať väčšie rozdiely, potreboval vykonať dlhšiu sekvenciu ťahov a taktiež vylepšiť samotný proces učenia. To s pôvodným nastavením a pri mojich dostupných výpočetných zdrojoch[7] nedokázal prekonať.

Situáciu ešte komplikoval fakt, že po inicializačnej fáze, kedy začína algoritmus vyberať ťah deterministicky podľa najvyššej $N_{s,a}$, nastávali situácie, kedy pacmani začali striedať susedné pozície dokola. Podobne to bolo s duchmi. To sa môže v kombinácií s rovnými $Q_{s,a}$ všetkých ťahov stať jednoducho. Malo by stačiť to, že opačné akcie pri susedných políčkach majú najvyššie pravdepodobnosti vrámci všetkých ťahov.

Nakoniec som skúsil ešte výraznú zmenu a pridal som ku finálnym evaluačným hodnotám taktiež tzv. pseudo-odmeny[8]. Tie pacmana a duchov výrazne odmeňovali po lokálnej stránke. Ak sa napr. duchovia priblížili ku pacmanom alebo pacmana zabili, dostali väčšiu odmenu. Pacmani dostávali taktiež extra odmenu za zjedenú pacdot. Výsledná odmena bola rovná súčtu pôvodnému predikovanému v a týchto pseudo-odmien (ktoré boli v prípade výhody pre duchov záporné). To malo za následok podstatne väčšiu variáciu medzi $Q_{s,a}$ hodnotami pri MCTS. Keď som tento prístup nasadil a začal trénovať, zaznamenal som taktiež progres pri učení. Stratégiu som evaluoval voči originálnej stratégií duchov, ktorú som opísal v sekcii 3.1. Po prvej iterácií bolo priemerné skóre na dynamických a ťažších mapách rovné 0.255. Počas tréningu sa toto skóre vyšplhalo na 0.386, avšak medzi niektorými iteráciami dosť kolísalo, nejednalo sa teda nutne o najstabilnejší tréning a miestami to vyzeralo, že konštanty, ktoré som zvolil pre tréning NN je nutné vrámci tejto úpravy ešte odladiť.

Každopádne tento prístup bol len experimentom. Pseudo-odmeny a ich váhy som nastavil ad hoc aby som otestoval, či sa bude vôbec niečo diať. Taktiež výsledná odmena sa pre jednoduchosť implementácie stále rátala voči koreňovému uzlu v MCTS, teda z listového uzlu sa propagovala tá istá pseudo-odmena pre každý uzol až do koreňa. To nie je korektný prístup, keďže nižšie

[7] Jedna iterácia trvala približne 2-3 hodiny.

[8] Jedná sa o pseudo-odmenu, keďže nutne nie je spojená so skutočnou odmenou, ktorú sa pacman snaží maximalizovať, čo je skóre.

uzly by mali mať inú bezprostrednú pseudo-odmenu, vzhľadom na platnosť markovskej podmienka t.j. ak napr. v predošlom ťahu zomrie pacman, nemá sa to prejaviť na odmene nižšieho ťahu, keďže to nemá žiaden vplyv.

Napriek tomu, že takto upravená stratégia dosahovala nejakých úspechov, stále sa nejednalo o rozumné správanie a jednoduchá proti-stratégia by toho vedela využiť. Problém s učením lepšej stratégie mali najmä duchovia, pacmani javili aspoň nejakú formu mierne rozumnejšieho správania. Každopádne bez kontinuálneho zlepšovania oboch hráčov táto metóda nemôže fungovať.

Stojí ale za zmienku, že tieto pozorovania som robil len po pár iteráciach tréningu, čiže môžu byť veľmi skreslené a v skutočnosti pri dlhšom tréningu to mohlo dosahovať podstatne odlišných vlastností. Pri finálnej evaluácií bol počet výsledných iterácií približne 2 krát väčší.

V evaluácií budem používať naučené váhy ztohoto experimentu ale vrámci MCTS nebudem uvažovať spomenuté pseudo-odmeny. Pre kompletnosť som pridal rýchlu ad hoc implementáciu spomenutých odmien do prílohy v zložke *pseudo_rewards_changes*. Konkrétne sa jednalo o poupravené zdrojové kódy modulov *neural_network* a *alpha_zero*. Vrámci konfigurácie som pri tomto experimente zmenil taktiež hodnoty c_{init} pacmanom aj duchom na 4 a 4.25 aby som vyvážil veľkosť exploračnej časti v SELECT rovnici.

3.3.8 Možné zlepšenia

Lepší prístup, v prípade pseudo-odmien, by mohol spočívať pri korektnom používaní bezprostredných odmien. Tie by boli rátané osobitne pre každý stav stým, že budúce odmeny z nadchádzajúcich ťahov, ktoré sú propagované vrámci MCTS, by boli v každom kroku násobené nejakým faktorom z interavalu $[0, 1]$. To je štandardný prístup pri RL, ktorý rieši problém s nekonečnými odmenami, pričom zároveň taktiež zmenšuje váhu budúcim odmenám, ktoré môžu byť neisté. V literatúre sa označuje pod pojmom „discount factor“. Pri tomto zlepšení by bolo potrebné taktiež zoptimalizovať váhy vo finálnej SELECT rovnici vrámci MCTS a veľkosť exploračných konštánt. Pseudo-odmeny adresujú problém s nemennými $Q_{s,a}$ v prvých iteráciach. Ak by sa agent týmto štýlom naučil mikro stratégie t.j. ako sa hýbať v bezprostrednej blízkosti, pričom by uvažoval iba zopár susedných políčok, je možné skúsiť tieto pseudo-odmeny vypnúť a nechať agenta aby pokračoval v učení.

Ďalším potencionálnym zlepšením je využiť prioritizovaný replay buffer [38], ktorý sa snaží zefektívniť proces učenia tým, že preferuje vzorky, pri ktorých je väčšia šanca, že agenta niečo naučia (vzorky, ktoré mali najväčšiu odchýlku od predikovaného výsledku). To môže pomôcť špeciálne v tomto prípade, keďže počet stavov v ktorých pacman pri každej hre zomrie je len zopár, čo môže byť slabým signálom pre ducha aby modifikoval svoje správanie a niečo sa naučil (keďže v replay buffer je takýchto vzorkov málo).

Agenta je možné začať učiť na menších a ťažších hrách, kde sa musí naučiť základy ako sa správať v bezprostrednej blízkosti k rôznym objektom t.j. zbie-

rať pacdots a vyhýbať sa duchom, v prípade pacmanov a zabíjať pacmanov prípade duchov atp. a následne mapu postupne zväčšovať.

Namiesto pseudo-odmien, ktoré by agenta mohli naučiť mikro správanie a základné stratégie, je možné použiť taktiež predtrénovanie modelu aby imitoval inú relatívne kvalitnú metódu (v mojom prípade by sa jednalo napríklad o negamax).

Ďalším zlepšením môžu byť abstraktnejšie ťahy, ktoré majú signifikantný efekt na zmenu hry, napr. ako to bolo v práci [3]. Týmto by sa mohol odstrániť problém pri učení s uviaznutím pri nemenných odmenách. Je tu ale otázka ako by sa to zakomponovalo do algoritmu AlphaZero.

Ďalším potencionálnym zlepšením by mohlo byť vyskúšať iné reprezentácie vstupu a výstupu, poprípade iné typy architektúr neurónovej siete.

Kapitola 4

Implementácia

Najdlhšiu časť práce zabral práve vývoj riešenia, ktoré som implementoval v programovacam jazyku Python 3.8. Kód pozostáva z troch základných modulov:

- `algorithms`: obsahuje implementované metódy a logiku pre spúšťanie a vyhodnocovanie hry.
- `visualization`: jedná sa o vizualizačný modul, ktorý dokáže vykresľovať celý priebeh hry.
- `common`: modul, ktorý obsahuje triedy základných objektov a utilít.

Oba moduly `algorithms` a `visualization` sú závislé na `common`. Modul `algorithms` je taktiež závislý na `visualization`, keďže som chcel podporovať vizualizáciu prebiehajúcej hry.

Pri všetkých týchto moduloch som si dal záležať na kvalite a čitateľnosti kódu, ktorý je zdokumentovaný, základné časti algoritmu otestované a v prípadoch, ktoré si to vyžadovali, taktiež dostatočne rozšíriteľný a obecný aby bolo možné jednoducho pridať, zoptimalizovať a vyhodnotiť novú stratégiu, poprípade rozšíriť vizualizačný modul o nové animácie alebo pridať novú hru Pac-Man s mierne upravenými pravidlami. V ďalších sekciách priblížim architektúru týchto modulov, spomeniem používané knižnice a taktiež spôsob, akým som implementované metódy trénoval a vyhodnocoval v CloudFIT[9].

4.1 Zdieľaný modul

Obsahuje základné zdieľané datové triedy ako `PerformanceEvaluation`, `Map` alebo `GameStateSnapshot`. V rámci utilít tu je implementovaný algoritmus BFS, menšie funkcionálne utility. Modul obsahuje taktiež základnú konfiguráciu logger-u, ktorý posiela logy na štandardný výstup a v prípade chyby

[9]Výpočetné zdroje, ktoré mi boli pridelené fakultou

taktiež ešte email. To bolo dôležité z monitorovacieho hľadiska, keďže tréning niektorých implementovaných metód trval niekoľko dní.

4.2 Vizualizácia

Jednou zo základných používaných architektúr pri vývoji hier je tzv. ECS (systém entít a komponent) [39], ktorý adresuje problém obrovskej dynamiky objektov s množstvom vlastností, ktoré sú pri hrách typické. Objavujú sa pri ňom 2 základné typy objektov:

- Entita: objekt, ktorý si drží data, referenciu na systém alebo scénu a komponenty, avšak nijak snimi priamo neinteraguje. Jedná sa teda skôr o „kontajner“. Data si objekt ukladá v dynamických štruktúrach ako napr. slovník.

- Komponenta: objekt, v ktorom sú implementované rôzne funkcionality ako animácie, reakcia na vstup, fyzika objektov atp. Komponenty menia stav objektu, na ktorý sú pripojený a sú v každom kroku aktualizované systémom.

Základnou výhodou tejto architektúry je fakt, že komponenty je možné ku entitám dynamicky pridávať a takto rozširovať ich správanie. Napr. ak máme objekt `Pacman` a chceme mu pridať možnosť pohybovať sa, jednoducho k nemu pridáme komponentu `Move`, ktorá reaguje na vstup a patrične upravuje objektu `Pacman` jeho pozíciu, ktorá je vždy uložená pod tým istým jednotným kľúčom vrámci dáta slovníkov entít. Ak by sme následne chceli k nemu pridať animáciu, znova k nemu stačí pridať novú komponentu animácie. Ak by sme chceli to isté správanie pridať k objektu `Duch`, znova stačí tieto komponenty naňho nalepiť a systém bude fungovať.

Takýto systém je teda veľmi flexibilný k zmenám, rôzne funkcionality sú enkapsulované vrámci samostatných komponent a dáta vrámci entít. Komponenty majú obecne referenciu len na entitu, medzi sebou teda kominukujú prostredníctvom správ na ktoré si môžu dynamicky prihlásiť odber. Odosielanie správ má na starosti systém/scéna.

Vzhľadom na podstatu implementovaného problému som sa rozhodol pre tento typ architektúry. Systém/scéna je v tomto prípade `VisualizationManager`. Tá v každom kroku prijíma rôzne udalosti na ktoré reaguje (ako vstup z klávesnice), taktiež aktualizuváva komponenty a reaguje na správy. Trieda bola implementovaná tak, aby bola bezpečná z pohľadu paralelného výpočtu. Podporuje plynulé vykresľovanie hry a taktiež základné ovládanie ako: zastaviť/spustiť prehrávanie (tlačítko medzerník), prejdi na začiatok predošlého ťahu (šipka vľavo), prejdi na začiatok následujúceho ťahu (šipka vpravo), posuň prehrávanie o 5 ťahov dopredu (šipka nahor) a posuň prehrávanie o 5 ťahov dozadu (šipka dole). Veľkosť okna sa počíta automaticky na základe veľkosti mapy

a rozlíšenia monitoru. Ukážku vizualizácie je možné vidieť na obrázku 4.1. Žlté veľké bodky predstavujú pacmanov, biele menšie bodky pacdots, ostatné duchov (v prípade originálnej stratégie duchov ich farby reprezentujú ich typ) a modré útvary predstavujú steny.

V prípade zmeny stavu hry rozposiela `VisualizationManager` správu, na čo následne rôzne komponenty reagujú. Napr. objektom sa postupne mení pozícia aby plynule prešli na susedné políčko, mení sa taktiež čítač skóre atp. Na vytváranie objektov s ich patričnými komponentami využíva `VisualizationManager` triedu `GameObjectFactory`. Implementovanú architektúru systému je možné vidieť na obrázku 4.2.

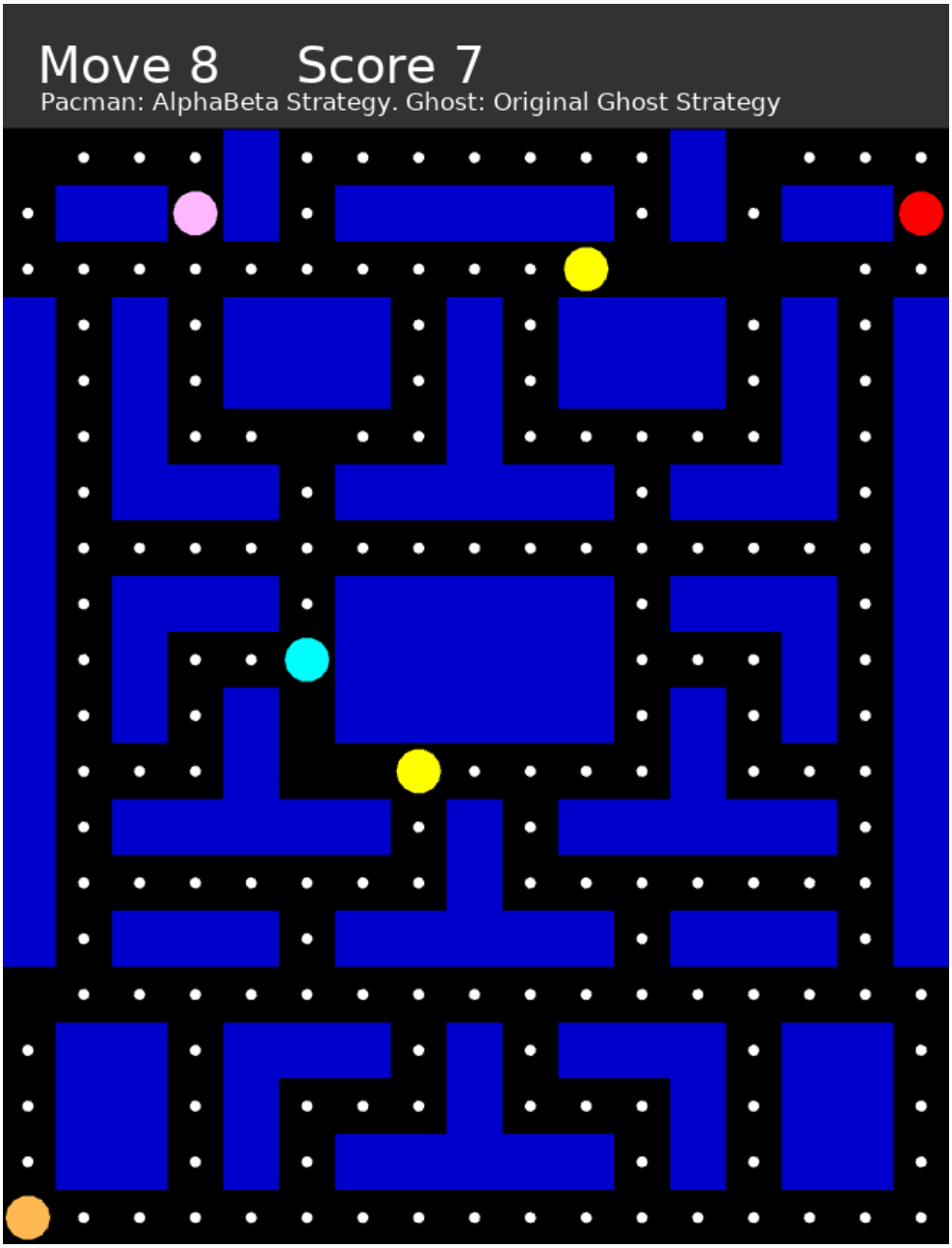
4.3 Algoritmy

Modul obsahuje kompletne implementované metódy, ktoré som spomenul vrámci kapitoly 3 vrátane logiky pre spúšťanie hry, zbieranie dat, optimalizáciu parametrov a finálnu evaluáciu stratégií.

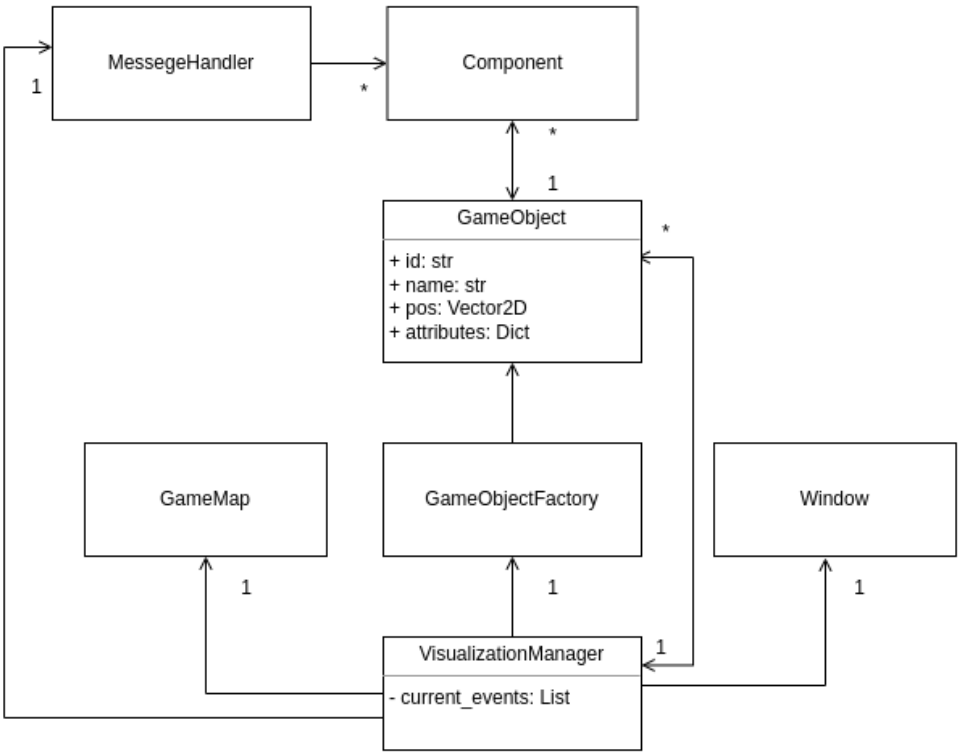
Pri metóde AlphaZero som použil na vytvorenie a tréning neurónovej siete populárny framework Keras [40]. Pre distribuovaný tréning a evaluáciu framework Ray [41].

Základným objektom je `GameManager`. Ten dokáže odohrať sériu hier medzi jednotlivými hráčmi. Na začiatku hry nainicializuje hráčov a ich stratégie, taktiež objekty typu `GameObserver`, ktorý majú rôzne funkcionality ako zbieranie dat, poprípade vizualizáciu hry. V každom kroku sa spýta na ďalší ťah aktuálneho hráča `Player` a informáciu o danom ťahu a novom stave odošle všetkým `GameObserver`. Ťah hráča obsahuje taktiež tzv. `strategy_metadata`, do ktorých môže stratégia vypĺňať ľubovoľné informácie pre `GameObserver`.

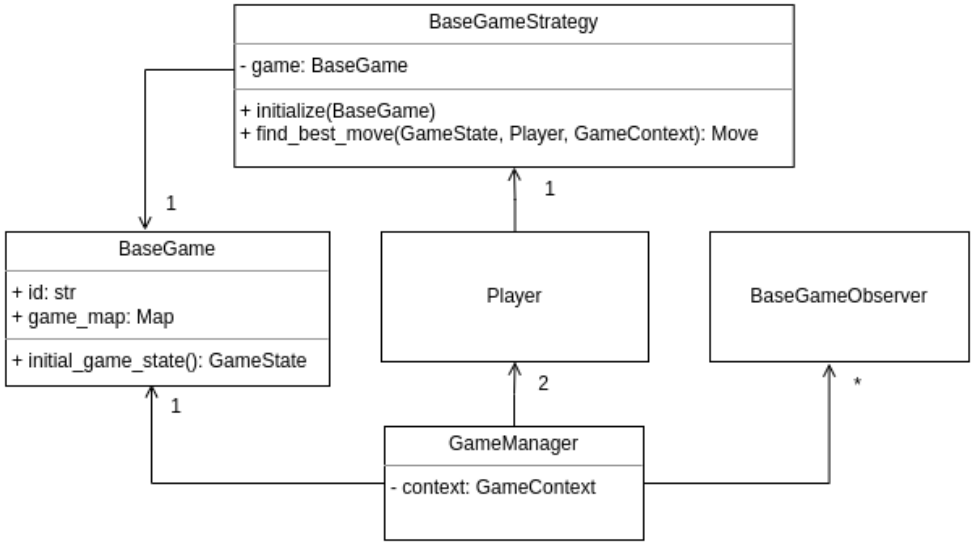
Hráč v metóde prijíma aktuálny stav hry a herný kontext, ktorý obsahuje n posledných ťahov a stavov hry (to používa napr. originálna stratégia duchov na určenie orientácie pacmana). Po skončení hry vráti z metódy finálny stav, ktorý obsahuje základné informácie ako skóre, čítač ťahov a pozície všetkých objektov. `GameManager` nie je závislý na konkretných implementáciach, je teda jednoducho prepoužiteľný pre akýkoľvek typ hry alebo hráčov a ich stratégií, ktoré dodržiavajú definované rozhranie. `GameObserver` je flexibilná trieda, ktorú je možné využívať na všemožné účely (v AlphaZero to bolo napríklad zbieranie trénovacich dat pomocou atribútu `Move.strategy_metadata`). V triede `BaseGame` je definované základné rozhranie, ktoré má poskytovať hra. Konkrétne sa jedná o metodu `initial_game_state` poskytujúcu počiatočný stav hry, metoda `is_move_valid`, ktorá robí validáciu ťahu v konkrétnom stave, metoda `make_move`, ktorá prijíma stav a ťah a vracia nový stav hry a metoda `is_finished`, ktorá kontroluje, či daný stav predstavuje konečný stav hry. V rámci kódu sa ku `GameState` správam ako ku konštantnému objektu, metóda `make_move` vracia teda nový objekt. Táto architektúra je znázornená na obrázku 4.3.



Obr. 4.1: Ukážka vizualizácie hry



Obr. 4.2: Architektúra vizualizačného modulu



Obr. 4.3: Architektúra základnej časti algoritmického modulu

Podmodul `optimization` obsahuje obecnú logiku na optimalizáciu akýchkoľvek parametrov akejkoľvek stratégie v kombinácií s Bayesovskou optimalizáciou a ligov agentov, ktorú som opísal v sekcii 3.2.3. Na Bayesovskú optimalizáciu využívam modul Tune [42] frameworku Ray. Ďalším dôležitým podmodulom je `evaluator`, ktorý dokáže odohrať sériu distribuovaných zápasov na viacerých procesoch, medzi n stratégiami pre pacmanov aj duchov a na ľubovoľnom počte máp. Výsledky hier a ich štatistiky následne vracia v jednotnom objekte, ktorý je možné jednoducho skonvertovať do formátu JSON [43]. Na distribuovaný výpočet využívam taktiež funkcionality Ray. Implementácie stratégií sú uložené v module `strategies`. Evaluátor dokáže taktiež uložiť celý priebeh hry, ktorý je následne možné zvizualizovať.

Definície hier sú načítavané zo súboru. Obsahujú informácie ako id hry, ktoré musí byť unikátne, ich popis, rozmery, maximálnu dĺžku hry a následne definíciu mapy s počiatočnými rozmiestnením objektov. Ich formát je popísaný vrámci modulu `game_loaders`.

Modul `alpha_beta` obsahuje triedy, ktoré sa používajú pri stratégií negamax. Jedná sa o transpozičnú tabuľku, samotné prehľadávanie a funkcionalitu na radenie ťahov.

Modul `alpha_zero` obsahuje podobne ako `alpha_beta` triedy, používané pri stratégií AlphaZero. Jedná sa taktiež o poupravenú transpozičnú tabuľku, triedu `AlphaZeroLearner`, ktorá implementuje distribuovaný tréning a globálnu iteráciu AlphaZero s replay buffer, triedu `NeuralNetwork`, ktorá obsahuje architektúru neurónovej siete a logiku vytvárania vstupu a výstupu a nakoniec `NeuralNetworkWrapper`, ktorý obaľuje triedu `NeuralNetwork` a plní funkcionalitu cachovania predikcií (čo môže podstatne urýchliť tréning, keďže predikcia NN bola výrazne najpomalšou časťou pri MCTS v metóde AlphaZero) a normalizovania predikovaných hodnôt (filtrovanie nevalidných ťahov, úprava v na očakávané finálne skóre atp.).

Stratégie poskytujú taktiež funkcionalitu na cachovanie medzi-výpočtov medzi rôznymi ťahmi. V prípade MCTS to je napríklad prepoužívanie herného stromu, ktorý je uložený v transpozičnej tabuľke, v prípade negamax, taktiež prepoužívanie transpozičnej tabuľky.

Aby som mohol otestovať kvalitu stratégií proti ľudského hráčovi, vytvoril som taktiež stratégiu `KeyboardInputStrategy`, ktorá dokáže ovládať objekty v hre prostredníctvom vstupu z konzole a písmen: a (vľavo), w (hore), s (dole), d (vpravo) a n (stoj). Objekty, ktoré ovláda hráč sú zoradené podľa osy x a následne osy y (v oboch prípadoch vzostupne) a prvé písmeno na vstupe je priradené prvému objektu, druhé druhému objektu atp. Ľavý dolný roh mapy má súradnice $x = 0, y = 0$. Ťah pre 2 objekty by mohol teda vyzerať napr. ako äw".

Na základné testovanie som vytvoril taktiež stratégiu `RandomUniform`, ktorá vyberá náhodný ťah rovnomerne náhodne a taktiež stratégiu `PredefinedStrategy`, ktorá má dopredu dané aké ťahy zvolí.

4.4 Distribuovaný tréning a evaluácia

Distribuovaný tréning bežal na výpočetnom clustri CloudFIT. K dispozícii som mal 64 CPU, 2 GPU NVIDIA A100-PCIE-40GB a 128 GB RAM. Pre jednoduché nasadenie, monitorovanie a celkový manažment, som používal technológiu Docker [44]. Konkrétne som vytvoril 3 Dockerfile: optimalizácia negamax, tréning AlphaZero a evaluácia stratégií. Na monitorovanie mi taktiež poslúžil tzv. Ray dashboard, ktorý bol súčasťou knižnice Ray.

Optimalizácia algoritmu negamax bežala v poslednej iterácií na 21/22 CPU po dobu približne 10 dní. Miestami bola prerušená a spustená znovu s mierne poupravenou konfiguráciou a poslednými parametrami, napr. kvôli tomu, že optimalizácia začala konvergovať a použitý užší obor hodnôt niektorých parametrov (kvôli rýchlejšej konvergencii) sa zdal byť nedostačujúci. V tomto prípade som rozšíril možné obory hodnôt parametrov a optimalizáciu spustil odznova. Výsledný počet iterácií bol 16.

Tréning AlphaZero bežal na 30 CPU a 2 GPU (na každú GPU bolo priradených rovnomerný počet CPU). Vzhľadom na vysokú experimentáciu, posledná úprava sa trénovala len pár dní s výsledným počtom iterácií 20.

Aplikáciu je možné ovládať pomocou high level skriptov, ktoré sa nachádzajú v module `scripts`. Tie je možné spustiť po vytvorení virtuálneho Python prostredia a nainštalovania závislosti pomocou príkazu, ktorý sa spustí z vrchnej zložky so zdrojovými kódmi, `pip install -r requirements.txt`. Skripty je následne možné spúšťať z konzole so správnou nastavenou `PYTHONPATH`, do ktorej je potrebné pridať taktiež adresár so zdrojovými kódmi (rovnaká úroveň ako adresár `src` v prílohe).

Jedná sa o následovné skripty:

- `visualize`: Spúšťa vizualizáciu hry, ktorú dostal na vstupe ako parameter. Ten reprezentuje cestu ku uloženému priebehu hry, ktorý vznikol aplikovaním príkazu pickle [45] na objekt typu `GameSnapshot`.

- `arena`: Spustí evaluáciu všetkých stratégií, ktoré som vrámci tejto práce vytvoril, proti sebe. Hry, ktoré sa budú pri evaluácií brať v úvahu sú všetky hry z adresára `data/games/` Na vstupe prijíma nasledujúce parametre: počet procesov, pracovný adresár, cesta ku váham neurónovej siete metódy AlphaZero, cesta ku váham metody evaluačnej funkcie metódy negamax, časový limit na jeden ťah a príznak, pomocou ktorého je možné zapnúť vizualizáciu priebehu každej hry po skončení evaluácie. Vizualizácia sa spúšťa pre každú hru samostatne a postupne. Ak chce užívateľ prejsť na ďalšiu hru, je potrebné zavrieť aktuálne okno. Ak užívateľ nechce pokračovať vo vizualizácií, je potrebné ukončiť proces.

- `alpha_beta_optimization`: Spúšťa distribuovanú optimalizáciu parametrov metódy negamax. Vstupom sú parametre ako pracovný adresár, veľkosť ligy, počet procesov, ktoré sa majú na optimalizáciu použiť atp.

V prípade, že sa nejaký parameter nešpecifikuje, použije sa jeho predvolená hodnota.

- `alpha_zero_training`: Spúšťa distribuovaný tréning metódy AlphaZero. Vstupom sú parametre ako pracovný adresár, počet simulácií, ktoré sa majú vrámci jedného ťahu vykonať, počet procesov, ktoré sa majú na tréning použiť atp. V prípade, že sa nejaký parameter nešpecifikuje, použije sa jeho predvolená hodnota.

Skripty je po nastavení virtuálneho prostredia Python možné spúšťať následovne `./scripts/arena.py --param1=value1 --param2=value2`. Na vytváranie týchto skriptov som použil technológiu python-fire [46]. Viac o formáte predávaných hodnôt je možné nájsť na `https://github.com/google/python-fire`.

Kapitola 5

Evaluácia

Na evaluáciu algoritmov som používal všetkých 10 pôvodných máp, ktoré je možné nájsť v prílohe v zložke `data/games`. Používal som turnajovú metódu každý proti každému. Finálne parametre, použité pre evaluačnú funkciu algoritmu negamax je možné nájsť v súbore `data/alpha_beta_params/main.json`. Finálneho váhy pre NN sú v súbore `data/nn_model/nn_iteration_20.h5`.

Popri optimalizácii parametrov metódy negamax, rástol výkon proti predvoleným negamax parametrom, originálnej stratégií duchov a poslednému exploiter, stabilne. Priemerné skóre začalo na hodnotách 0.447 so štandardnou odchýlkou 0.226 a po 16 iteráciach skončilo na hodnotách 0.516 so štandardnou odchýlkou 0.181, čo podporuje hypotézu, že táto metóda fungovala. Je možné, že pri väčšom výpočetnom čase a potencionálne ešte lepšej konfigurácií optimalizátora, by tento výkon pokračoval v rastúcom trende.

Parameter maximálna hĺbka pri metóde negamax som volil dostatočne veľký a rovný 100 aby keďže som používal časový limit.

Najväčšie váhy optimalizácia pripisovala príznakom ako „je najbližší druhý duch blízko pacmana“, „pacman sa nedokáže bezpečne dostať na žiadnu susednú križovatku“ alebo „pacman sa nedokáže bezpečne dostať na žiadne políčko vo vzdialenosti 5“, čo sedí s mojou intuíciou.

Ako som už zhodnotil v predošlých sekciách metóda AlphaZero mala počas učenia s experimentom pseudo-odmien nestabilný tréning ktorý miestami dosť kolísal. Avšak priemerné skóre proti originálnej stratégií duchov začalo na 0.251 so štandardnou odchýlkou 0.089 a po 20 iteráciach skončilo na 0.314 so štandardnou odchýlkou 0.12. Najvyššie dosiahnuté priemerné skóre bolo po 9 iterácií a bolo rovné 0.386 so štandardnou odchýlkou 0.131. Nejaký rastúci trend učenia sa tam teda vyskytoval nebolo to stabilné ako v prípade optimalizácie negamax. V rámci tejto evaluácie som použil váhy z tréningu s pseudo-odmenami avšak tie som neuvažoval pri evaluácii.

Finálne hyperparametre metódy AlphaZero pri evaluácií boli následovné $c_{base} = 19652$, $pacman_{c_{init}} = 2$, $ghost_{c_{init}} = 2.25$ a vzhľadom na časový limit som zvolil $num_simulations = 8000$ dostatočne vysoký aby to nemohol

		Pacman							
		Main AB		**Default AB**		**AlphaZero**		**Random**	
		avg	std	avg	std	avg	std	avg	std
Duch	**Main AB**	0.38	0.22	0.37	0.20	0.08	0.04	0.04	0.02
	Default AB	0.36	0.17	0.44	0.18	0.09	0.04	0.04	0.02
	AlphaZero	0.65	0.19	0.53	0.20	0.21	0.03	0.33	0.08
	Random	0.86	0.09	0.74	0.19	0.17	0.02	0.18	0.09
	Original ghost	0.66	0.20	0.69	0.19	0.14	0.07	0.05	0.02

Tabuľka 5.1: Výsledky evaluácie (Main AB predstavuje optimalizovaný negamax a Default AB negamax s predvolenými parametrami)

ovplyvniť.

Okrem optimalizovanej stratégií negamax a AlphaZero, som ku evaluácií pridal taktiež originálnu stratégiu duchov, náhodnú stratégiu a stratégiu negamax s predvolenými parametrami, ktoré som heuristicky odhadol a manuálne ladil. Pre účely porovnávania stratégií medzi sebou, má náhodná stratégia zakaždým rovnako nainicializovaný pseudonáhodný generátor.

Stratégie mali na výpočet jedného ťahu 3 sekundy. Maximálny počet ťahov v hre bol nastavený na 575. Niektoré mapy začínali vo výhodnejších pozíciach pre duchov inokedy to bolo viac výhodné pre pacmanov. Maximálne skóre 1.0 znamená, že pacman zjedol všetky pacdots. To avšak nie je pri mierne rozumnej stratégií nie je úplne možné, keďže by duchom stačilo zablokovať vchod do nejakej uličky a po celú zvyšnú dĺžku hry tam stáť.

Štatistiky o výsledkoch hry som zhrnul v tabuľke 5.1. Detailnejšie výsledky je možné nájsť v prílohe v zložke `arena_final`.

Pri interpretácií výsledkov je potrebné brať do úvahy maximálny počet ťahov, ktorý odmeňoval rýchlejších hráčov a taktiež mohol miestami znemožňovať vyzbieranie všetkých pacdots vrámci hry. Ďalším aspektom sú rôzne mapy, ktoré mali rôznu náročnosť.

Z výsledkov je vidieť, že zďaleka najefektívnejšou stratégiou boli negamax metódy. Podľa očakávaní, AlphaZero nedosahovalo silnejších výsledkov, avšak prekvapivo z experimentu so pseudo-odmenami sa dokázala niečo naučiť, keďže v pozícií ducha mala miestami o dosť lepšie správanie než náhodná stratégia.

Negamax stratégie mali proti sebe podobné výsledky. Hlavná stratégia mala avšak lepšiu štandardnú odchýlku. Prekvapivo negamax s predvolenými parametrami dosiahol proti sebe lepšieho výsledku než hlavná optimalizovaná negamax stratégia ale v prípade slabších hráčov ako AlphaZero a náhodnej stratégie, dosahovala optimalizovaná metóda výrazne lepších výsledkov a najmä stabilnejších s približne polovičnou štandardnou odchýlkou proti náhodnej stratégii. Zdá sa teda, že optimalizácia aj po pár iteráciach, zrobustnila danú metódu.

Jeden z problémov pri negamax stratégiách, ktorý som si pri vizualizácií všimol je, že niekedy sa stávalo, že pacmani ostali v určitej sekcii mapy ako keby zaseknutý. Dalo by sa hypotetizovať, že to bolo najmä kvôli menšej prehľadávanej hĺbke herného stromu. Avšak v tomto mohli úlohu zohrať taktiež možné kolízie medzi príznakmi v evaluačnej funkcii, čo by sa mohlo dobudúcna vylepšiť, poprípade nejaké príznaky pridať.

Záver

V rámci tejto práce som definoval zobecnenú ťahovú hru Pac-Man, ktorá uvažuje väčšie množstvo duchov a pacmanov, čo jej výrazne pridáva na komplexite. V úvodných kapitolách som priblížil kľúčové problémy tejto hry, opísal možné kategórie prístupov objavujúcich sa v literatúre a spomenul ich výhody a nevýhody.

Následne som opísal implementované metódy, spoločne s ich vylepšeniami a modifikáciami. Jednou znich bola základná metóda na ovládanie duchov, ktorá bola priamo prevzatá z hry Pac-Man a poupravená aby dokázala fungovať pre väčšie množstvo agentov. Druhou metodou bol negamax ku ktorému som pridal alfa-beta orezávanie, používanie transpozičných tabuliek a mierne komplexnejšiu metódu pre optimalizáciu váh evaluačnej funkcie. Treťou metódou bola AlphaZero, metoda založená na kombinácií stromového učenia a hlboké spätnoväzobného učenia, pri ktorej som musel vzhľadom na podstatu problému modifikovať viacero častí a skombinovať rôzne prístupy. Vzhľadom na komplexitu tejto metódy, ju bolo podstatne náročnejšie doladiť. Podľa výsledkov posledného experimentu som pravdepodobne odhalil kľúčovú príčinu, ktorú som rozpísal a opis metódy zakončil možnými vylepšeniami.

V tejto práci som implementoval taktiež obecný vizualizačný modul a obecný framework na používanie a distribuované vyhodnocovanie stratégií, tréning a optimalizáciu parametrov.

Implementované metódy som vyhodnotil na rôznorodých mapách s rôznymi počiatočnými pozíciami a následne tieto výsledky zanalyzoval. Ukazuje sa, že zoptimalizovaná metóda negamax dokáže fungovať robustne a „inteligentne“. Pri ďalšej metóde AlphaZero je jej výkon nižší, čo môže byť ale adresované spomenutými vylepšeniami.

Pri tvorbe tejto práce som použil široké spektrum znalostí, ktoré som sa na fakulte naučil a to najmä znalosti zo softwarového inžinierstva, umelej inteligencie, rôznych oblastí matematiky ako lineárna algebra, analýza, pravdepodobnosť a štatistika, a distribuovaných systémov.

Medzi hlavné prínosy práce patrí zanalyzovanie zobecnenej verzie hry

Pac-Man, súhrn používaných metód v literatúre, popis kľúčových problémov, najmä v kombinácií s metodou AlphaZero, opis možných vylepšení, obecná implementácia vizualizačného modulu a modulu pre vývoj a vyhodnocovanie stratégií a vyhodnotenie implementovaných metód.

Literatúra

[1] Russell, S.; Norvig, P.: *Artificial Intelligence*. Upper Saddle River: Pearson Education, třetí vydání, 2010, ISBN 978-0-13-207148-2.

[2] Pittman, J.: The Pac-Man Dossier. [online], 2009, [cit. 2022-02-03]. Dostupné z: `https://www.gamedeveloper.com/design/the-pac-man-dossier`

[3] Pepels, T.; Winands, M. H. M.; Lanctot, M.: Real-Time Monte Carlo Tree Search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, ročník 6, č. 3, 2014: s. 245–257, doi: 10.1109/TCIAIG.2013.2291577.

[4] Silver, D.; Schrittwieser, J.; Simonyan, K.; aj.: Mastering the game of go without human knowledge. *nature*, ročník 550, č. 7676, 2017: s. 354–359.

[5] Tziortziotis, N.; Tziortziotis, K.; Blekas, K.: Play ms. pac-man using an advanced reinforcement learning agent. In *Hellenic Conference on Artificial Intelligence*, Springer, 2014, s. 71–83.

[6] Campbell, M.; Hoane Jr, A. J.; Hsu, F.-h.: Deep blue. *Artificial intelligence*, ročník 134, č. 1-2, 2002: s. 57–83.

[7] Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; aj.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, ročník 575, č. 7782, 2019: s. 350–354.

[8] Dalgaard, M.; Motzoi, F.; Sørensen, J. J.; aj.: Global optimization of quantum dynamics with AlphaZero deep exploration. *npj Quantum Information*, ročník 6, č. 1, 2020: s. 1–9.

[9] Wang, X.; Qian, Y.; Gao, H.; aj.: Towards efficient discovery of green synthetic pathways with Monte Carlo tree search and reinforcement learning. *Chemical science*, ročník 11, č. 40, 2020: s. 10959–10972.

[10] Von Stengel, B.: Game theory basics. *Lecture Notes, Department of Mathematics, London School of Economics, Houghton St, London WC2A 2AE, United Kingdom*, 2008.

[11] Rohlfshagen, P.; Liu, J.; Perez-Liebana, D.; aj.: Pac-Man Conquers Academia. *IEEE Transactions on Games*, ročník 10, č. 3, 2018: s. 233–256, ISSN 2475-1502, doi:10.1109/TG.2017.2737145. Dostupné z: `https://ieeexplore.ieee.org/document/8207594/`

[12] Choi, T.; Na, H.-S.: Making Levels More Challenging with a Cooperative Strategy of Ghosts in Pac-Man. *Journal of Korea Game Society*, ročník 15, č. 5, 2015: s. 89–98.

[13] Gallagher, M.; Ryan, A.: Learning to play Pac-Man: An evolutionary, rule-based approach. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, ročník 4, IEEE, 2003, s. 2462–2469.

[14] Lim, C.-U.; Baumgarten, R.; Colton, S.: Evolving behaviour trees for the commercial game DEFCON. In *European conference on the applications of evolutionary computation*, Springer, 2010, s. 100–110.

[15] Handa, H.; Isozaki, M.: Evolutionary fuzzy systems for generating better Ms.PacMan players. In *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*, 2008, s. 2182–2185, doi:10.1109/FUZZY.2008.4630672.

[16] Zadeh, L. A.: Fuzzy logic. *Computer*, ročník 21, č. 4, 1988: s. 83–93.

[17] Knuth, D. E.; Moore, R. W.: An analysis of alpha-beta pruning. *Artificial intelligence*, ročník 6, č. 4, 1975: s. 293–326.

[18] Schaeffer, J.: The history heuristic and alpha-beta search enhancements in practice. *IEEE transactions on pattern analysis and machine intelligence*, ročník 11, č. 11, 1989: s. 1203–1212.

[19] Breuker, D. M.: Memory versus search in games. 1998.

[20] Pearl, J.: The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the ACM*, ročník 25, č. 8, 1982: s. 559–564.

[21] Schaeffer, J.: The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 11, č. 11, 1989: s. 1203–1212, doi:10.1109/34.42858.

[22] Zobrist, A. L.: A new hashing method with application for game playing. *ICGA Journal*, ročník 13, č. 2, 1990: s. 69–73.

[23] Browne, C. B.; Powley, E.; Whitehouse, D.; aj.: A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, ročník 4, č. 1, 2012: s. 1–43, doi:10.1109/TCIAIG.2012.2186810.

[24] Lisỳ, V.; Bošanskỳ, B.; Jakob, M.; aj.: GOAL-BASED ADVERSARIAL SEARCH.

[25] Szepesvári, C.: Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, ročník 4, č. 1, 2010: s. 1–103.

[26] Hornik, K.: Approximation capabilities of multilayer feedforward networks. *Neural networks*, ročník 4, č. 2, 1991: s. 251–257.

[27] Silver, D.; Hubert, T.; Schrittwieser, J.; aj.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[28] Silver, D.; Huang, A.; Maddison, C. J.; aj.: Mastering the game of Go with deep neural networks and tree search. *nature*, ročník 529, č. 7587, 2016: s. 484–489.

[29] Frazier, P. I.: A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[30] Mordvintsev, A.; Randazzo, E.; Niklasson, E.; aj.: Growing neural cellular automata. *Distill*, ročník 5, č. 2, 2020: str. e23.

[31] Ha, D.; Tang, Y.: Collective Intelligence for Deep Learning: A Survey of Recent Developments. *arXiv preprint arXiv:2111.14377*, 2021.

[32] Darer, A.; Lewis, P.: A Cellular Automaton Based Controller for a Ms. Pac-Man Agent. *arXiv preprint arXiv:1312.5097*, 2013.

[33] Schaeffer, J.: The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 11, č. 11, 1989: s. 1203–1212, doi:10.1109/34.42858.

[34] Cardona, A. B.; Togelius, J.; Nelson, M. J.: Competitive coevolution in ms. pac-man. In *2013 IEEE Congress on Evolutionary Computation*, IEEE, 2013, s. 1403–1410.

[35] Goldwaser, A.; Thielscher, M.: Deep reinforcement learning for general game playing. In *Proceedings of the AAAI conference on artificial intelligence*, ročník 34, 2020, s. 1701–1708.

[36] Childs, B. E.; Brodeur, J. H.; Kocsis, L.: Transpositions and move groups in Monte Carlo tree search. In *2008 IEEE Symposium On Computational Intelligence and Games*, IEEE, 2008, s. 389–395.

[37] Han, L.; Sun, P.; Du, Y.; aj.: Grid-wise control for multi-agent reinforcement learning in video game ai. In *International Conference on Machine Learning*, PMLR, 2019, s. 2576–2585.

[38] Schaul, T.; Quan, J.; Antonoglou, I.; aj.: Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[39] Entity Component System. [online], [cit. 2022-02-10]. Dostupné z: `https://docs.unity3d.com/Packages/com.unity.entities@0.17/manual/ecs_core.html`

[40] Chollet, F.; aj.: Keras. 2015. Dostupné z: `https://github.com/fchollet/keras`

[41] Moritz, P.; Nishihara, R.; Wang, S.; aj.: Ray: A Distributed Framework for Emerging AI Applications. 2018, `1712.05889`.

[42] Liaw, R.; Liang, E.; Nishihara, R.; aj.: Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118*, 2018.

[43] Introducing JSON. [online], [cit. 2022-02-10]. Dostupné z: `https://www.json.org/json-en.html`

[44] Docker. [online], [cit. 2022-02-10]. Dostupné z: `https://github.com/docker`

[45] pickle — Python object serialization. [online], [cit. 2022-02-10]. Dostupné z: `https://docs.python.org/3/library/pickle.html`

[46] pythonfire. [online], [cit. 2022-02-10]. Dostupné z: `https://github.com/google/python-fire`

Dodatok **A**

Zoznam použitých skratiek

RL Spätnoväzobné učenie

DL Hlboké učenie

MCTS Stromové prehľadávanie Monte-Carlo

NN Neurónová sieť

TT Transpozičná tabuľka

Dodatok **B**

Obsah priloženého USB

- `Readme.txt` ..... stručný popis obsahu USB
- `Multi-Pacman` ..... zdrojové kódy implementácie a dáta
- `Run_results` ..... výsledky evaluácie
- `Thesis` ..... text práce
  - `thesis.text` ..... zdrojová forma práce vo formáte LaTeX
  - `thesis.pdf` ..... text práce vo formáte PDF