Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Učenie posilňovaním v doméne symetrických hier

Bakalárska práca

2024
Patrik Filipiak

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Učenie posilňovaním v doméne symetrických hier

Bakalárska práca

Študijný program:	Aplikovaná Informatika
Študijný odbor:	Aplikovaná Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	doc. Ing. Peter Lacko, PhD.

Bratislava, 2024
Patrik Filipiak

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:
Študijný program:

Študijný odbor:
Typ záverečnej práce:
Jazyk záverečnej práce:
Sekundárny jazyk:

Názov:

Anotácia:

Vedúci:
Katedra:
Vedúci katedry:

Dátum zadania:

Dátum schválenia:

garant študijného programu

.....

študent

.....

vedúci práce

Poďakovanie: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstrakt

ABSTRAKT - TODO

Kľúčové slová:

Abstract

ABSTRACT - TODO

Keywords:

Obsah

Zoznam obrázkov

Úvod

TODO

Kapitola 1

Strojové učenie

Umelá inteligencia je definovaná ako schopnosť počítačov riešiť problémy a robiť rozhodnutia podobnou formou ako by och robila ľudská myseľ. Je to jedna z najrýchlejšie rozvíjajúcich oblastí súčasného sveta. Skôr či neskôr, nejakým spôsobom ovplyvní každé existujúce povolanie. Z odvetví umelej inteligencie sa v tejto práci budeme venovať práve strojovému učeniu. Strojové učenie je skupina výpočtových algoritmov, ktoré sa pokúšajú napodobniť fungovanie ľudskej mysle. Tieto algoritmy riešia problémy bez toho, aby boli na také situácie priamo naprogramované. Toto robia tak, že sa pred samotným riešením problému natrénujú rôznymi spôsobmi, aby vedeli v každej situácii vybrať najlepšiu možnú akciu. [2]

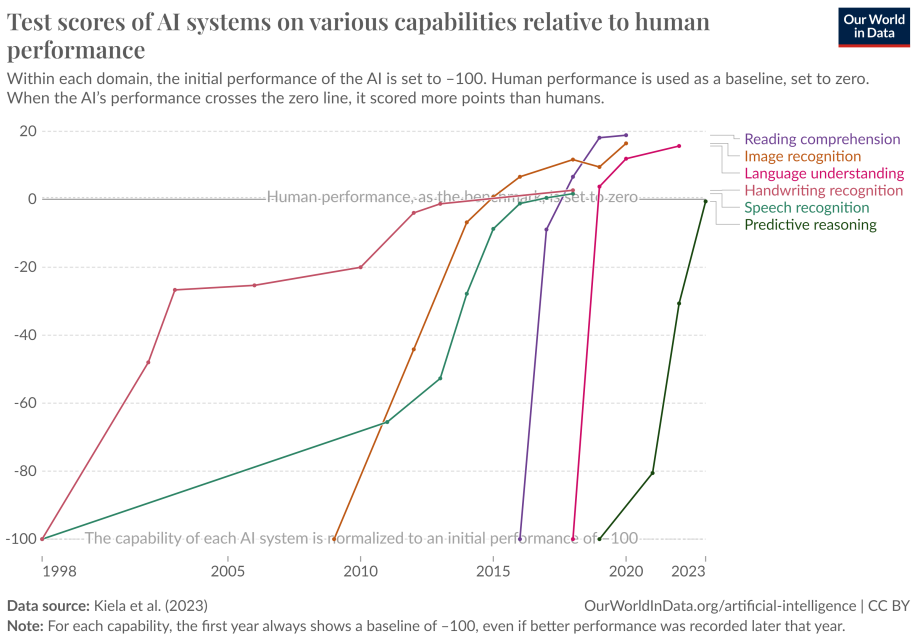
1.1 Význam

Ľudia sa začali zaujímať o strojové učenie, lebo určitú prácu vedia stroje spraviť lepšie, rýchlejšie alebo lacnejšie ako človek. V grafe 1.1 vidíme ako sa vyvíjala umelá inteligencia a kedy sa človeku vyrovnala v aktivitách ako čítanie s porozumením, rozpoznávanie reči alebo obrazu a iných. Aj keď sa v tejto práci budeme sústrediť na učenie stolových hier, prínos strojového učenia ďaleko presahuje hranice tieto hranice. Ďalšie využitia zahŕňajú napríklad:

1. Medicína - umelá inteligencia sa ukázala ako efektívny spôsob diagnostiky symptómov rôznych ochorení.

2. Automobilová doprava - všetky vyvíjané samo-riadiace autá sú postavené na použití umelej inteligencie, ktorá posudzuje vnemy z okolia aby vedela vykonať správnu akciu.

3. Chatboti - Odkedy sa na scéne zjavil ChatGPT, začali sa objavovať rôzne verzie chatbotov od rôznych firiem. Chatbot ponúka ľuďom možnosť vysvetliť problema-



Obr. 1.1: Porovnanie výkonu umelej inteligencie s človekom v rôznych spôsoboch vnímania dát [3]

tickú tému spôsobom akým by ju podal iný človek, aká je veľa krát jednoduchšia na pochopenie.

1.2 Delenie

Strojové učenia sa delí do troch hlavných kategórií, ktoré sa hlavne delia tým aké dáta vstupné dáta potrebujú na učenie. Každá kategória má vlastné algoritmy a výhody pri použití na rôznych problémoch.

1.2.1 Učenie s učiteľom

Hlavná zásada algoritmov využívajúcich tento postup, je zistiť mapovanie vstupov k želaným výstupom. Jedna zo štandardných formulácií úlohy vhodnej na riešenie pomocou učenia s učiteľom, je: učený model sa snaží zreplikovať (alebo sa to mu čo najbližšie priblížiť) chovanie funkcie, pomocou niekoľkých príkladov vstupu a správneho výstupu. [7] Čím viac takýchto príkladov pri trénovaní poskytneme, tým lepšie bude náš model reprezentovať požadovanú funkciu.

Výhody

Trénovanie na dostatočnom množstve kvalitných dát vyústi v model s veľmi presnou kategorizáciou nových dát. Feedback z učenia je ľahko intepretovateľný a teda nájsť

spôsob ako zlepšiť efektivitu je pri tomto prístupe ľahšie ako pri iných. Taktiež, rýchlosť učenia je väčšinou značne lepšia ako pri modeloch bez výstupných dát.

Nevýhody

Tento prístup má aj viaceré nevýhody. Prvou, celkom očividnou je, že na učenie treba čo najviac vstupno-výstupných dát, ktoré sú dostatočne rôznorodé, čo pri niektorých situáciách je ťažké dostať. Avšak, aj keď sa nám podarí zhromaždiť veľké množstvo trénovacích dát, ešte nemáme vyhrané. Tieto výstupné dáta treba totiž všetky označiť správnym výsledkom aby bolo učenie možné, čo môže byť pri toľkých dátach často príliš drahé ak nie nemožné. Navyše niektoré s výstupných dát nemusia mať úplne jasné označenie, čim sa ten proces ďalej zkomplikuje. [5]

1.2.2 Učenie bez učiteľa

Učenie bez učiteľa je menej používané ako učenie s učiteľom, ale aj tak sa nájdu situácie, pri ktorých je jeho použitie výhodnejšie. Hlavným rozdielom týchto dvoch metód je, že učenie bez učiteľa sa nesnaží generovať čo najlepšie predikcie vzhľadom na vstupné dáta, ale snaží sa vstupné dáta pospájať do väčších celkov podľa ich spoločných čŕt.

Výhody

Obrovská výhoda je, že učenie bez učiteľa nevyžaduje k trénovaniu žiadne ručne ohodnotené dáta, ktoré v situáciách z reálneho života často ani nie sú k dispozícii. Ďalšia výhoda takéhoto učenia je, že nájde medzi vstupnými dátami prepojenia, ktoré by ušli človeku, a teda by ich učenie s učiteľom tiež nikdy neobjavilo.

Nevýhody

Kvalita výsledkov učenia bez učiteľa je často veľmi variabilná a nestála. Toto učenie niekedy vstupné dáta rozdelí do úplne iných skupín ako by sme chceli, lebo mu nedávame žiadne dodatočné informácie.

1.2.3 Učenie posilňovaním

Metóda strojového učenia, ktorej sa budeme prevažne v tejto práci venovať, sa volá Učenie posilňovaním. Tento typ učenia závisí od priamej interakcie agenta a prostredia. Agent na začiatok dostane iba problém ktorý sa snaží riešiť a model prostredia kde ho riešiť bude. V tom prostredí sa agent postupne učí metódou pokus omyl ktoré ťahy prinesú lepší výsledok. [12]

Výhody

Toto učenie je veľmi flexibilné, lebo na jeho začatie je potrebný iba model prostredia. Vďaka tomu sa používa v rôznych odvetviach ako napríklad robotika, hry, financie, and zdravotníctvo. Okrem toho sú takýto agenti schopný sa naučiť stratégie, ktorých manuálny dizajn by bol príliš náročný. Agentom učeným posilňovaním aj menej prekáža zmena prostredia.

Nevýhody

Hlavná nevýhoda tohto prístupu je, že na učenie potrebuje agent veľmi veľa interakcií s prostredím. Toto spôsobuje vysokú časovú náročnosť učenia, preto nemusí byť táto voľba vhodná pre jednoduchšie problémy.

1.3 Dôležité pojmy pri učení posilňovaním

Učenie posilňovaním je veľmi rozsiahla téma a pred tým ako sa jej začneme vo väčšej miere venovať, musíme si vysvetliť niekoľko základných pojmov. Tieto pojmy zahŕňajú názvy priamych účastníkov učenia, ako aj pomenovania princípov a vylepšení ktoré sa pri učení posilňovaním používajú.

1. **Agent** je základný komponent strojového učenia. Je to entita, ktorá aktívne mení stav prostredia, robí rozhodnutia a učí sa.

2. **Prostredie** je systém v ktorom sa nachádza agent. Tento agent môže v prostredí vykonať akciu, ktorá môže prostrdie zmeniť, čo ovplyvní vyberanie akcií v budúcnosti.

3. **Akcia** je spôsob, akým ovplyvňuje agent prostredie. Akciu si agent vyberá z akčného priestoru prostredia. Rôzne prostredia môžu mať iný druh akčného priestoru. Zvyčajne sa delia na tri druhy: diskrétny priestor, spojitý priestor alebo diskrétno-spojitý hybrid. [16] V tomto delení záleži najmä na počte možných akcií v každom bode učenia.

4. **Odmena** je signál, ktorý poskytuje prostredie agentovi, po tom ako vykoná nejakú akciu. Táto odmena hovorí ako blízko je agent k celkovému cieľu po práve vykonanej akcii, takže cieľ agenta je dospieť k čo najväčšej hodnote tejto odmeny.

5. **Pozorovanie** je informácia o stave prostredia, ktorú si môže agent od prostredia vypýtať. Táto informácia pomáha pri výbere budúcich akcií.

6. **Stratégia** je mapovanie stavov prostredia k najlepším možným akciám v týchto momentoch. Práve stratégia definuje správanie sa nášho agenta a preto je dôležité vybrať tú správnu pre náš problém.

7. **Pomer prieskumu a využívania** je definovaná hodnota pravdepodobnosti, ktorá určuje, ako často si agent namiesto vykonania najlepšej akcie z doposiaľ preskúmaných možností, vyberie trochu horšiu, aby preskúmal novú časť stavového priestoru. Toto v správnej miere zabezpečuje, aby sa pri učení agent nezasekol na lokálnom maxime.

Kapitola 2

Umelá inteligencia a stolové hry

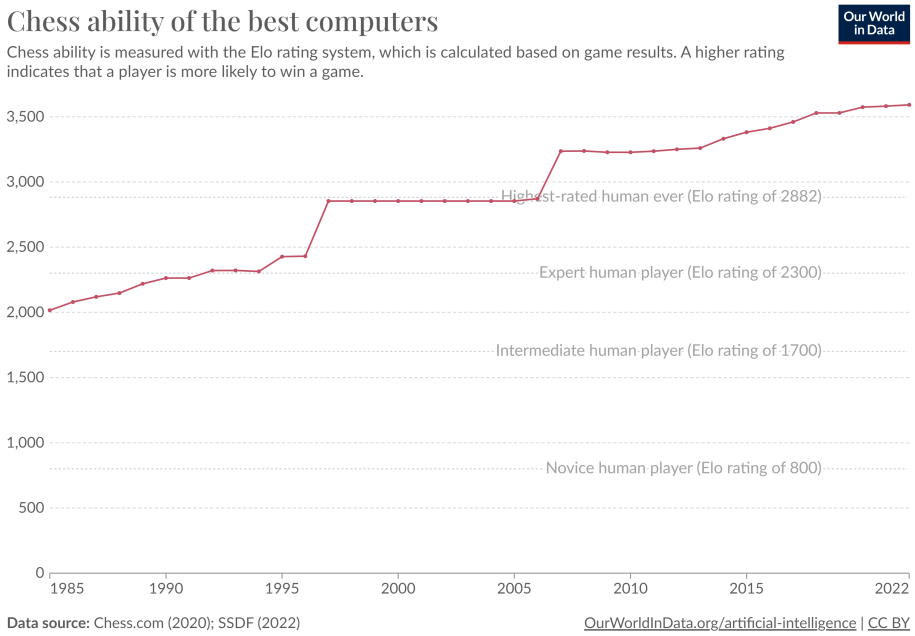
Stolové hry sú už tisícky rokov jedna z najzákladnejších foriem trávenia voľného času. Okrem toho prinášajú pre človeka stolné hry viaceré benefity. Mnohé z nich bohato prispievajú k rozvoju strategického myslenia, zdravej súťaživosti či tvorbe medziľudských vzťahov (avšak niektoré, ako napríklad Monopoly, môžu mať v určitých prípadoch opačný dôsledok). Medzi hrami ako šach, ktoré sa s ľudstvom ťahajú ohromne dlho, a modernou dobu kedy neustále vychádzajú nové zaujímavé originálne hry si každý nájde to svoje. Avšak ako mnohé iné odvetvia, aj toto zažíva zmenu od kedy sa svetu ukázala umelá inteligencia vo svojej súčasnej podobe. Pred tým ako do hier vstúpila umelá inteligencia, boli stolové hry výlučne aktivita na ktorej sa podieľali dvaja alebo viacerý ľudia. Toto sa ale zmenilo ako náhle sa stroje začali učiť a veľmi rýchlo prekonali level priemerného človeka v hre na ktorú sa použili. Na obrázku číslo 2.1 vidíme ako sa vyvíjala umelá inteligencia na hranie šachu a kedy bola dosť dobrá na to aby porazila priemerného človeka, ale aj doposiaľ najlepšieho hráča šachu na svete.

2.1 Šach

Šach bol pre ranný vývoj umelej inteligencie výborný kandidát pre viacero dôvodov. Prvým bol fakt, že je to hra s perfektnou informáciou, čo znamená, že agent má k dispozícii všetky informácie podľa ktorých vyberá ďalší ťah a je v nej dobre definovaný súbor pravidiel. Toto pomohlo umelej inteligencii pochopiť dôsledky jej akcií. Okrem toho bol šach odjakživa spojený s ľudskou inteligenciou, pretože vyžadoval určitý level strategického myslenia, logiky a schopnosti plánovať svoje kroky dopredu.

2.1.1 Počiatočné nápady

Záujem o vývoj počítača schopného hrať šach vzrástol po konci druhej svetovej vojny. Najväčšie zásluhy o tento vzrast mali dva články napísané v tej dobe. Prvým bol "Programming a Computer for playing Chess", ktorý napísal Claude Shannon v roku 1950.



Obr. 2.1: Porovnanie kvality umelej inteligencie na hranie šachu za posledných 40 rokov [3]

Na hranie šachu v tejto práci prvý krát použili minmax algoritmus. Minmax funguje tak, že sa pozrie na celý herný strom a všetky koenčné stavy ohodnotí číslom, napríklad ak vyhrá hráč jedna, tak tam bude jednotka, ak hráč dva tak tam bude dvojka, inak tam bude nula. Teraz sa rozhodne, že jeden hráč bude takzvaný maximalizujúci a druhý bude minimalizujúci. Potom pôjde program od tých konečných stavov naspäť po strome hore a podĺa toho, koho kolo práve je, vyberie buď maximálny alebo minimálny ťah v danom stave. Tento algoritmus je veľmi pomalý, lebo potrebuje prejsť celý herný strom aby dostal najlepšiu cestu. Avšak, ako sa neskôr zistilo, tomuto problému sa dá predísť. V roku 1956 bolo vynájdené takzvané Alfa-Beta orezávanie, ktoré pomôže programu predčasne vyhodnotiť do ktorých vetiev hraného stromu nemá zmysel sa vnárať a tým to ušetrilo kopu času. Druhý dôležitý článok napísal Alan Turing a dôležitý koncept ktorý tam predstavil boli takzvané mŕtve pozície. Pozíciu označil mŕtvou, ak počas nasledujúcich niekoľkých ťahov nie je možné ani pre jedného hráča spraviť významný pokrok k víťazstvu. Z takýchto pozícií sa netreba ďalej vnárať, ale namiesto toho je možné ich vyhodnotiť pomocou funkcie, ktorá sa používa rôzne parametre ako počet ostávajúceho materiálu hráčov, štruktúra pešiakov alebo bezpečnosť kráľa. [4]

2.1.2 Prvé programy

Úplne prvý signifikantnejší program na hranie šachu vznikol v roku 1957 a bol schopný vykonávať 42000 operácií za sekundu. Tento program neprehľadával celý strom, ale obsahoval heurisitku, ktorá mu vybrala sedem zdanlivo najlepších ťahov a vyberal

spomedzi nich. Nasledujúci program, ktorý dosiahol vyššiu úroveň hry, vyšiel v roku 1968 so znalosťami hry na úrovni človeka s 1500 elo. Kvôli slabému hardvéru tej doby ani tento program neprehľadával celú šírku stromu, ale len jeho časť. S programom, ktorý by vedel kontrolovať celú šírku stromu sa objavil až v roku 1973. Táto udalosť započala vo svete inú fázu vývoja programov na hranie šachu. Ľudia sa začali viac zaoberať zlepšovaním hardvérov ako softvérov. Toto vyústilo v osemdesiatych rokoch dvadsiateho storočia na spustenie doposiaľ najlepšieho programu na šach (Blitz) na najrýchlejšom počítači na svete (Cray) a spolu sú schopný preskúmať 20,000 - 30,000 vrcholov hracieho stromu. [4]

2.1.3 Deep Blue

Doposiaľ však neboli vyvinuté programy schopné postaviť sa profesionálnym ľudským hráčom šachu. Platilo to ale až dokým z dielne IBM nevyšiel v deväťdesiatych rokoch počítač na hranie šachu s názvom Deep Blue. Programátori, ktorí vyvíjali Deep Blue si boli jeho schonosťami tak istý, že v roku 1996 vyzvali momentálneho majstra sveta v šachu, Garryho Kasparova na hru. Aj napriek faktu, že ich počítač Kasparova porazil len v dvoch hrách zo šiestich, to považovali za úspech, lebo to bolo bolo prvý krát v histórii kedy sa počítaču podarilo zdolať majstra sveta. Výhra na seba ale nenechala dlho čakať a už o rok na to, pri opätovnom strete Deep Blue a Kasparova nad šachovou plochou, vyšiel ako víťaz práve superpočítač a to s troma výhrami oproti Kasparovým dvom. Túto udalosť bola pre vývoj umelej inteligencie jeden z najväčších milníkov v histórii. Experti v tejto oblasti na to mali ale rôzne názory. Niektorý to označili ako katastrofu lebo to znamenalo koniec ľudskej intelektuálnej nadradenosti strojom, iní považujú túto udalosť za dôkaz vynikajúceho ľudského programovania a inžinierskych schopností [11]. Celkovo, víťazstvo Deep Blue znamenalo zlomový bod. Umelá inteligencia v šachu pomaly prešla z výpočtu hrubej sily na strategickejší a jemnejší prístup aký poznáme dnes. Do šachovej scény dominantne vstúpilo učenie posilňovaním až s príchodom AlphaZero, o tom ale povieme viac v ďalšej kapitole. Momentálne je za najlepší program na hranie šachu považovaný Stockfish, čo je stále aktívne podporovaný open-source projekt.

2.2 Go

Go je čínska stolová hra pre dvoch hráčov pri ktorej im ide o zabratí čo väčšej časti hracej plochy ako ich súper. Štandardná veľkosť hracej plochy je 19x19, ale existujú varianty hry hrané na plochách veľkých 9x9 alebo 13x13. Dvaja hráči, čierny a biely, striedavo umiestňujú kameň vlastnej farby na prázdne križovatke na hracej ploche, pričom začína čierny. Pravidlo zachytávania uvádza, že ak boli kamene jednej farby

úplne obklopené kameňmi druhej farby tak, že žiadne horizontálne alebo vertikálne susediace miesto nie je prázdne, sú odstránené z dosky. Sila hráča v Go sa určuje podľa toho koľko majú "kyuä "dan". Kyu sú považované za študentské úrovne (30-1, pričom 1 je najlepšia) a dan sú zase majstrovské (1-7 sú pokročilé a potom 1-9 profesionálne).

2.2.1 Hrubá sila

Hra Go bola dlho považovaná za najnáročnejšiu z klasických hier pre umelú inteligenciu, kvôli jej obrovskému vyhľadávaciemu priestoru a ťažkostiam pri vyhodnocovaní pozícií a ťahov na hracej ploche. Aj keď sa ľudia pokúšali vytvárať programy hrajúce Go už v šesťdesiatych rokoch, hlavný nárast o záujem bol až v osemdesiatych rokoch. Hlavne sa tak stalo preto, lebo v tejto dobe začali byť počítače dostupnejšie pre väčší počet ľudí. Okrem toho sa začali usporadúvať turnaje so sponzormi a peňažnými cenami. Z tejto iniciatívy vzniklo niekoľko programov, žiadne však nestačili na to aby sa vyrovnali profesionálnym hráčom Go. V 1991 vyhral program Goliath prvý krát proti trom dobrým hráčom s hendikepom 17 kameňov. Ďalšie programy ako Handtalk, Go4++ a KCC Igo tiež dosiahli určitý úspech v hre bez hendikepu proti ľudským hráčom, ktorý sa blížia k sile úrovne dan [6].

2.2.2 Monte Carlo tree search

Veľká zmena nastala v roku 2006 pri vynájdení algoritmu Monte Carlo tree search. Monte Carlo tree search (skrátene MCTS) je rozhodovací algoritmus, ktorý sa používa v hrách s veľkým prehľadávacím priestorom. Tento algoritmus ale nehodnotí každý možný ťah ako to je pri minmax algoritme. Namiesto toho si pomocou danej stratégie vyberie sľubné vetvy podľa toho ako sú ohodnotené. Avšak dôležitá časť algoritmu je aj pomer medzi prehľadávaním a využívaním, vďaka ktorému si niekedy namiesto najlepšej možnosti vyberieme zatiaľ nepreskúmanú aby sme zamedzili možnosti že sa náš program zasekne na lokálnom minime. Po výbere vetvy program bude hru simulovať a zároveň sa bude snažiť vybrané ťahy ohodnotiť podľa toho aký konečný výsledok pomocou nich dosiahol. Použitie tohto algoritmu okamžite zlepšilo kvalitu programov zo 14 kyu, na 5 dan, čo je pokročilý level, ale stále nie profesionálny [13].

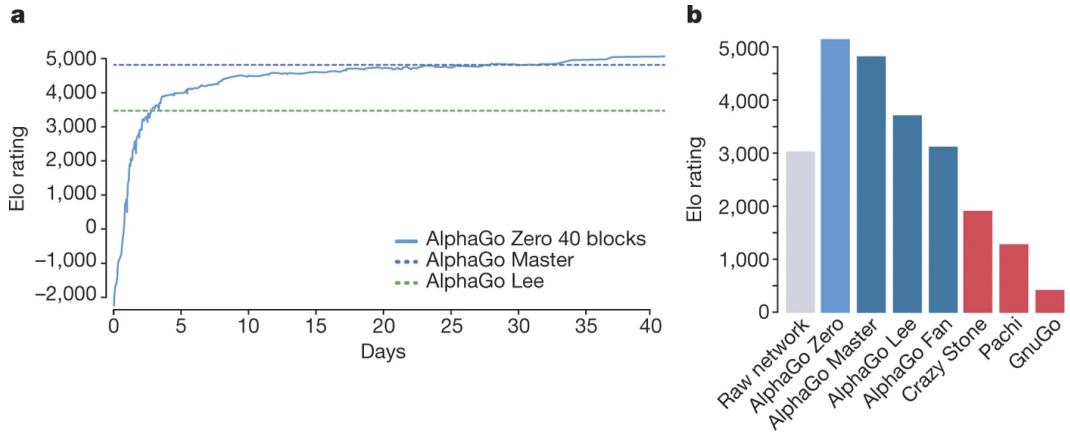
2.2.3 Neurónové siete

Spoločnosť DeepMind prišla v roku 2014 so svojím programom zvaným AlphaGo. Vo vývoji AlphaGo boli použité dve hlboké neurónové siete, jedna mala na starosti stratégiu a vydávala pravdepodobnosti ťahov, zatiaľ čo druhá bola sieť hodnôt, ktorá poskytuje ohodnotenie momentálnej plochy. Trénovanie sietí prebiehalo tak, že najprv

podstúpili učenie s učiteľom kde dostali odohrané hry expertov, a následne boli doladené učením posilňovaním. Natrénované siete boli potom spojené s Monte Carlo tree search, ktorý používal naučenú stratégiu a pomocou získaných pravdepodobností a vyberanie správnych vetiev a sieť hodnôt na vyhodnotenie získaných pozícií. Táto verzia programu sa volala AlphaGo Fan bola schopná v októbri 2015 poraziť súčastného majstra Európy. Legendárny hráč Go a 18-násobný majster sveta, Lee Sedol, bol porazený pomocou o niečo vyladenejšej verzie AlphaGo v marci 2016. Tento výsledok šokoval celú komunitu a ešte viac vzbudil vo svete záujem o umelú inteligenciu. Bolo zjavné, že za úspechom AlphaGo stálo okrem iného aj učenie posilňovaním. Toto pochopili aj programátori z DeepMind a preto ich ďalší projekt AlphaGo Zero to posunul ešte o krok ďalej. AlphaGo Zero sa od AlphaGo odlišovalo vo viacerých veciach, ale tou najdôležitejšou bolo, že na jeho trénovanie sa nepoužili žiadne odohrané hry z minulosti. Celý proces trénovanie sa spoliehalo na učenie posilňovaním, takže program začal tak, že robil úplne náhodné ťahy a po každej hre si upravil váhy podľa jej výsledku. Výhodou tohto postupu je, že agent môže počas trénovania nájsť nové a kreatívne stratégie, ktoré sa v hrách odohraných človekom ešte nevyskytli. Ďalší rozdiel je, že zatiaľ čo AlphaGo používa osobitné neurónové siete na stratégiu a na hodnotu, AlphaGo Zero má iba jednu. Na toto je prispôsobený na následný algoritmus prehľadávania stromu, kde sa už nepoužíval pôvodný Monte Carlo tree search, ale len jednoduchšia verzia. Tento model bol trénovaný od úplne náhodného správania bez ľudského zásahu približne tri dni, počas ktorých stihol vygenerovať 4.9 milión hier samého proti sebe. Po trénovaní postavili AlphaGo Zero proti tej istej kópii AlphaGo Lee, ktorá porazila Leeho Sedola pri rovnakých hernách podmienkach pri akých prebiehal ich zápas. Aj napriek tomu, že AlphaGo Zero používal jeden stroj so štyrmi jednotkami na spracovanie tenzorov, zatiaľ čo AlphaGo Lee bol distribuovaný na mnohých strojoch a používal 48 jednotiek TPU, AlphaGo Zero porazil AlphaGo Lee vo všetkých 100 hrách ktoré hrali [10]. Na obrázku číslo 2.2 môžeme vidieť ako sa schopnosti AlphaGo Zero zlepšovali podľa toho ako dlho bola trénovaná. Na grafe sú znázornené aj hranice schopnosti AlphaGo Lee a AlphaGo Master (program založený na algoritme a architektúre podobnej AlphaGo Zero, ale využívajúci ľudské dáta a funkcie, ktorý v januári 2017 porazil najsilnejších ľudských profesionálnych hráčov 60–0 v online hrách.)

2.3 Všeobecné AI

DeepMind neprestalo s vývojom pri AlphaGo Zero. Novoobjavenú technológiu sa rozhodli presunúť za hranice hry Go. Ich nový projekt AlphaZero mal byť technologicky veľmi podobný ako AlphaGo Zero, ale zároveň mal byť viac generalizovaný aby zvládal hranie viacerých stolových hier. Pôvodný cieľ bol zovšeobecniť program na hranie



Obr. 2.2: Krivka schopnosti AlphaGo Zero hrať Go [10]

Game	White	Black	Win	Draw	Loss
Chess	*AlphaZero*	*Stockfish*	25	25	0
	Stockfish	*AlphaZero*	3	47	0
Shogi	*AlphaZero*	*Elmo*	43	2	5
	Elmo	*AlphaZero*	47	0	3
Go	*AlphaZero*	*AG0 3-day*	31	–	19
	AG0 3-day	*AlphaZero*	29	–	21

Obr. 2.3: Tabuľka výsledkov hier medzi doterz najlepšími programami pre dané hry a AlphaZero [9]

šachu a japonskej hry shogi. Tieto hry sú ale menej vhodné na architektúru neurónových sietí použitú v AlphaZero kvôli viacerým faktorom. Pravidlá týchto hier sú totiž závislé na stave plochy (napríklad pešiak v šachu sa môže hýbať o dva políčka dopredu ak je to jeho prvý pohyb) a sú nesymetrické (pešiak sa môže hýbať len dopredu). Všetky tieto dodatočné informácie komplikujú architektúru sietí v programe, lebo už na učenie nemusí stačiť sieti poslať binárnu reprezentáciu momentálnej plochy, ale aj iné informácie. Aj napriek nepríjemnostiam tento program spravili v roku 2017 a výsledok bol prekvapivý [9]. V šachu AlphaZero prekonal Stockfish už po 4 hodinách tréningu, v shogi AlphaZero prekonal Elmo po menej ako 2 hodinách a v Go prekonal AlphaZero AlphaGo Lee po 8 hodinách. V tabuľke na obrázku 2.3 môžete vidieť koľkokrát vyhral AplhaGo v jednotlivých hrách zo 100 odohraných partií. AlphaZero reprezentoval obrovský pokrok v umelej inteligencii, lebo ukázal aký veľký potenciál má učenie posilňovaním, aj pri komplexných rozhodovacích problémoch.

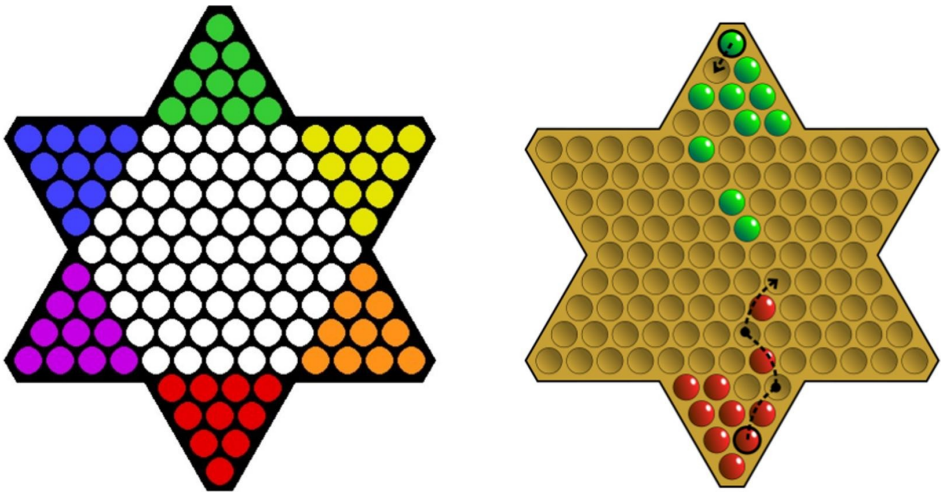
Kapitola 3

Halma

Ako prostredie môjho projektu som si vybral stolovú hru halma. Dôvodov na toto rozhodnutie je viac. Po prvé, halma je dobrý kandidát na aplikáciu učenia posilňovaním. Je to hlavne kvôli tomu, že je to hra s celou informáciou (obidvaja hráči majú k dispozícii všetky informácie, ktoré potrebujú na vyhodnotenie svojho ďalšieho ťahu) a jasne definovanými pravidlami. Okrem toho je v nej veľký priestor na hraciu stratégiu, ktorú sa môže agent naučiť. Druhým dôvodom prečo som si halmu vybral, je fakt že v tomto spektre nie je veľmi skúmaná. Existuje len zopár článkov zaoberajúcich sa aplikácii umelej inteligencie na hranie halmy. Posledný dôvod je osobný. V mladosti som sám hral túto hru doma a tak som sa rozhodol s ňou pracovať aj v tejto práci.

3.1 O hre

Halma je značne menej populárna a aj skúmaná hra ako šach či go. To na jednej strane otvára možnosti o výskum a prípadný pokrok v tejto oblasti, ale ne druhú stranu to znamená, že dostupnej dokumentácie o hre je veľmi málo. Halmu si ľudia často spájajú s čínskou dámou. Aj keď sú si tieto dve hry veľmi podobné, odlišuje ich jeden veľmi očividný prvok. Čo sa týka halmy, tá sa hrá na ploche v tvare štvorca, kde hráči začínajú s figúrkami v jednom s rohov. Táto hra je určená pre dvoch alebo štyroch hráčov. Na druhú stranu čínska dáma má plochu v tvare šesťcípej hviezdy a dá sa hrať v 2-6 hráčoch. Figúrky hráčov začínajú v jednotlivých cípoch (ako to vyzerá si môžete pozrieť na obrázku 3.1). Rok vzniknutia halmy nie je známy, ale čínska dáma je jej variant, ktorý vznikol v roku 1892 v Nemecku pod názvom sternhalma (stern je po nemecky hviezda). Názov čínska halma táto hra dostala až neskôr ako marketingový plán v Amerike [15]. V našej práci sa ale budeme sústrediť na pôvodnú verziu halmy so štvorcovou hracou plochou, konkrétne verzii pre práve dvoch hráčov. Plocha v pôvodnej halme je veľká 16 $\times$ 16 políčok (obr. 5.2).



Obr. 3.1: Ukážka plochy na čínsku dámu [15]

3.2 Pravidlá

Pravidlá hry sú pomerne jednoduché. Ako sme spomínali v predošlej podkapitole, hra sa začína tak, že majú hráči všetky figúrky v jednom z rohov hracej plochy. Pri hre dvoch hráčov musia hráči začínať oproti seba. Cieľom hry je dostať všetky svoje figúrky na druhú stranu plochy skôr, ako sa to podarí protivníkovi. Na splnenie toho sú k dispozícii dva spôsoby pohybu figúrok:

- **Krok** nastáva, keď sa figúrka pohne na nejaké susedné voľné políčko. Po tomto pohybe kolo momentálneho hráča končí a na rade je iný.
- **Skok** je, keď má figúrka vedľa seba postavenú inú figúrku (môže byť aj spojenecká aj protivníkova) a diagonálne za ňou je voľné miesto. V takom prípade sa môže na toto miesto figúrka presunúť a z novej pozície má možnosť takto skočiť ďalej. Za jedno kolo sa dá s figúrkou urobiť neobmedzene veľa skokov.

Kapitola 4

Predchádzajúca práca

Kapitola 5

Implementácia

Implementačná časť práce spočívala vo vytvorení prostredia hry Halma v ktorom by sa agent mohol učiť. V tomto prostredí by sme potom odskúšali agentov využívajúcich rôzne algoritmy a rôzne funkcie odmeňovania. Výsledky týchto kombinácií učenia neskôr porovnáme aby sme vedeli z dostupných algoritmov vybrať taký ktorý najlepšie funguje s naším problémom.

5.1 Výber technológií

Na prácu sme sa rozhodli použiť verejne dostupné knižnice s predprogramovanými prostrediami a algoritmami.

5.1.1 OpenAI Gym/Gymnasium

Najprv nám bolo treba získať funkčné prostredie na hru Halma. Najprv som hľadal na internete či náhodou niekto nespravil voľne dostupný simulátor Halmy ktorý by som mohol len použiť, ale nič čo som našiel nevyhovovalo mojím požiadavkám. Potom som prišiel na knižnicu OpenAI Gym, ktorá ponúkala viacero predprogramovaných prostredí na učenie posilňovaním. Ako som však rýchlo zistil, samotnú Halmu táto knižnica neponúkala a dokonca v svojej ponuke nezahŕňala žiadne prostredie s viacerými agentmi. Aj keď na to knižnica ponúkala nejakú podporu, rozhodol som sa hľadať ďalej. Jeden z dôvodov prečo som nakoniec OpenAI Gym nepoužil bol aj fakt, že knižnica už od 2021 nie je ďalej rozvíjaná [1]. Namiesto toho sa úsilie OpenAI presunulo na veľmi podobnú knižnicu s názvom Gymnasium. Táto knižnica má ale rovnaký problém ako Gym a to taký že sama o sebe neponúka žiadne prostredia kde sa navzájom ovplyvňujú viacerý agenti.

5.1.2 PettingZoo

Po troche hľadania som ale objavil niečo čo vyššie spomínaný problém vyriešilo. Knižnica s názvom PettingZoo sa zaoberá prostrediami pre učeni posilňovaní, ktoré obsahujú dvoch alebo viacerých agentov. Ide vlastne o multi-agent verziu knižnice Gymnasium, čo bolo presne to čo som hľadal [14]. Toto, rozsiahla dokumentácia a živý repozitár boli faktory vďaka ktorým sme sa nakoniec rozhodli použiť pre naše prostredie túto knižnicu aj keď sama o sebe neponúkala plochu na hranie Halmy.

5.1.3 Stable-Baselines3

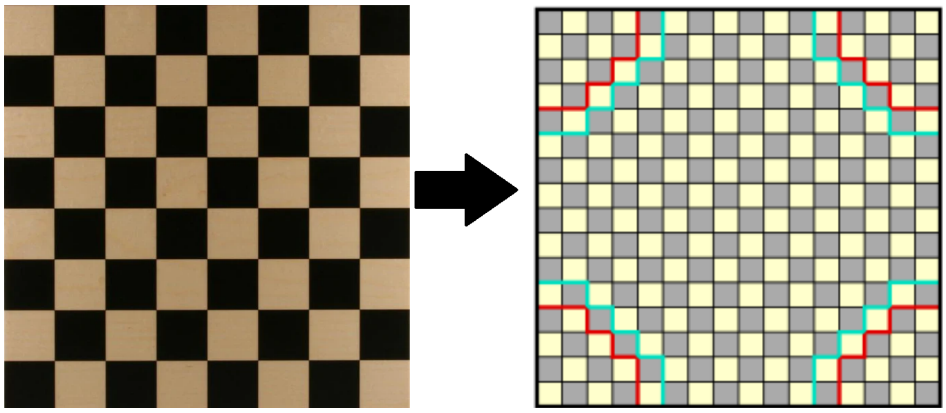
Teraz, keď sme si vybrali knižnicu v ktorej bude prostredie, ešte nám treba zabespečiť agenta. Keďže je OpenAI, a teda aj knižnica PettingZoo, taká populárna, existujú ďalšie knižnice s ponukou algoritmov kompatibilných s danými prostrediami. Hlavné tri medzi ktorými sme sa rozhodovali boli Stable-Baselines3, RLlib a Tianshou. Stable-Baselines3 je knižnica, ktorá si zakladá na poskytnutí rozhrania, ktoré je veľmi ľahké na používanie aj pre ľudí ktorý sú v tomto smere noví. Čo sa týka Stable-Baselines3 to vyzeralo sľubne, ponúkalo to veľa rôznych algoritmov a aj dokumentácia a komunita boli veľmi rozsiahle [8]. Aj keď táto možnosť ponúkala veľa príkladov kódov príkladov v rôznych prostrediach, bolo zjavné, že si zakladajú na prostrediach, kde je iba jeden hráč. Ďalšia vec, ktorá nebola ideálna bola tá, že posledný pohyb na ich oficiálnom githube bol v čase keď som to kontroloval takmer rok dozadu. Toto mohol byť problém, lebo nedostatočná starostlivosť o takúto knižnicu môže vyústiť o zastarané komponenty ktoré nebudú fungovať s viac obstarávanými komponentami.

5.1.4 RLlib

Druhý potenciálny kandidát na knižnicu z ktorej budeme čerpať predpripravené algoritmy bola knižnica RLlib. RLlib je postavený na frameworku Ray, ktorý uľahčuje používateľom paralelizáciu procesov na viacero počítačov. Tento benefit prenecháva aj knižnici RLlib, čo z nej robí veľmi dobrú voľbu pre komplexe projekty a učenie vo veľkej miere. Avšak, toto zároveň spôsobuje to, že práca s tým je značne náročnejšia a menej používateľsky príjemná. Keďže je náš projekt relatívne malý a nenáročný, rozhodli sme sa použiť iné alternatívy.

5.1.5 Tianshou

Posledná knižnica ktorú sme zohľadňovali pri výbere algoritmov bola knižnica Tianshou. Táto knižnica je asi najmenej známa zo všetkých a teda aj dokumentácia je najmenej rozvinutá. Napriek tomu však tiež ponúka bohatý výber algoritmov na učenie posilňovaníma používateľsky príjemné rozhranie. Veľká výhoda tejto možnosti bola aj



Obr. 5.1: Zmena z plochy na hranie šachu na halmu

to, že repozitár je stále konštantne vylepšovaný a aj to, že to ponúka jednoduché spojenie s prostrediami vytvorenými pomocou PettingZoo. Tieto dôvody boli dosť na to aby sme sa rozhodli použiť Tianshou na zabezpečenie testovacích algoritmov do nášho prostredia.

5.2 Vytváranie prostredia

Keďže sme nenašli vytvorené prostredie na Halmu museli sme si ho zaobstarať sami. Avšak namiesto toho aby sme celú hru programovali od začiatku, rozhodli sme sa použiť prostredie šachu, ktoré nám ponúka knižnica PettingZoo a prerobiť ho pre potreby hry Halma. Toto prerábanie vyžadovalo vymazanie veľkej časti pravidiel, ktoré sa v šachu nachádzajú, ale v halme už nie. Príklad takýchto pravidiel je napríklad rôzne druhy figúrok, vyhadzovanie figúrok, rôzne podmienky ústiace v remízu, a mnoho iných vecí.

5.2.1 Hracia plocha

Okrem toho bolo treba aj samotnú hraciu plochu, keďže šach sa hrá na ploche 8x8 a halma na ploche 16x16 (obr. 5.1). Na tejto ploche sú taktiež figúrky rozmiestnené v rohoch na rozdiel od šachu, kde figúrky začínajú v radoch za sebou. V našej verzii hry budú hráči začínať v ľavo hore a v pravo dole, pričom cieľom každého hráča bude protiľahlý roh.

5.2.2 Akčný priestor

Po hracej ploche sme sa vo vytváraní nášho prostredia pozreli na akčný priestor halmy. V tomto kontexte je je akčný priestor definovaný ako množina všetkých možných ťahov, ktoré môže hráč v danom momente vykonať. Jeden ťah znamená presunutie figúrky na prázdne políčko hneď vedľa, alebo políčko na ktorom figúrka ostane po sérii skokov.

Keďže v halme hrá každý hráč s trinástimi figúrkami, pričom každý z nich má väčšinou viacero legálnych pohybov, jej akčný priestor je pomerne veľký. Tento fakt znamená, že počas trénovania pravdepodobne nebude najlepšia voľba použiť niektoré druhy algoritmov, ako napríklad klasické Q-učenie, ale namiesto neho skúsiť hlboké Q-učenie. O tom prečo to nastáva rozvedieme v kapitole na to určenej. V našom programe, pri inicializácii prostredia každej možnej akcii na ploche priradíme číslo podľa ktorého sa na ňu bude agent odkazovať.

5.2.3 Priestor pozorovaní

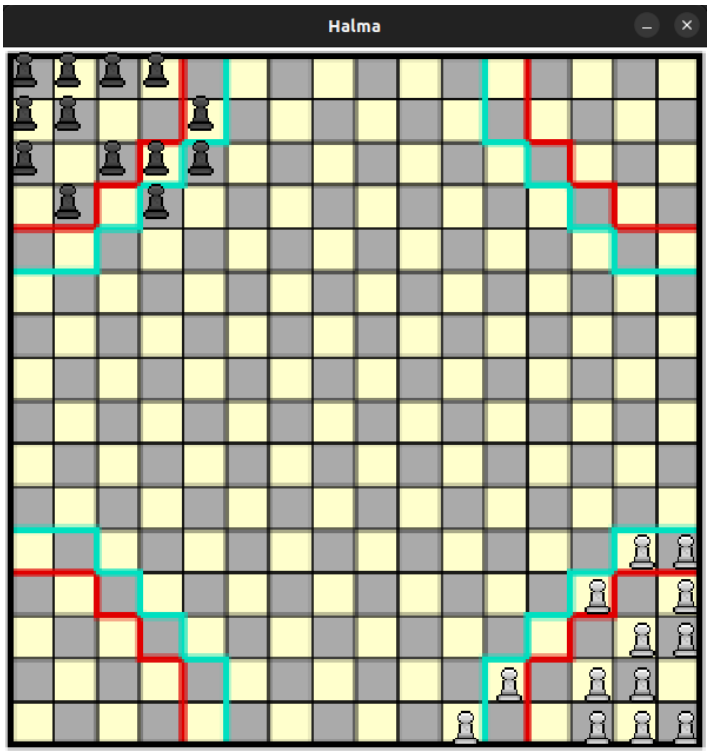
Ďalší dôležitý komponent prostredia je takzvané pozorovanie. Pozorovanie je agentovo vnímanie prostredia v určitom čase. Sú to dáta ktoré prostredie ponúkne agentovi aby si podľa toho vedel vybrať akú akciu zvoliť. Priestor pozorovaní je množina všetkých možných pozorovaní, ktoré môže agent v prostredí stretnúť [2]. V našom projekte sme na začiatok ako pozorovanie určili dve matice. Každá z nich reprezentuje hraciu plochu jedného z hráčov, kde je jednotka ak je na políčku figúrka daného hráča, inak tam je nula. Toto pozorovanie znamená, že náš priestor pozorovaní sú všetky možné rozloženia figúrok na hracej ploche.

5.2.4 Odmena

Odmena je produkt funkcie odmeny. Táto funkcia má za úlohu prevádzať agentove akcie a výsledky v prostredí na číselnú spätnú väzbu. Poskytnutá spätná väzba, pozitívna (odmeny) alebo negatívna (tresty), ovplyvňuje proces učenia agenta tým, že mu naznačuje, nakoľko sú jeho akcie žiaduce alebo nežiaduce na dosiahnutie konečného cieľa. V našej práci sme to mohli spraviť aj tak, že agent dostane odmenu až keď úplne vyhrá, ale zohľadňujúc komplexitu hry by trvalo strašne dlho kým by sa spočiatku náhodný agent dostal so svojimi figúrkami do cieľa. Ak však implementujeme správnu funkciu odmeny, môže mu to pomôcť vydať sa správnym smerom. Počas trénovania sme vyskúšali viaceré funkcie odmeny spojené s rôznymi cieľmi agentov, na začiatok sa ako vhodná ukázala funkcia, ktorá spočíta manhattanskú vzdialenosť všetkých figúrok hráča a vráti jej súčet. Táto funkcia je dobrá v tom, že agenta odmeňuje za to že dostane všetky figúrky smerom k cieľu.

5.3 Učenie

Dôležitý pojem pri učení posilňovaním, ktorý ale nesúvisí s prostredím, ale s agentom je stratégia (po anglicky policy). Stratégia je svojím spôsobom mozog agenta. Je to súbor pravidiel, ktoré agentovi hovoria akú akciu v danom ťahu zvoliť. V nasledujúcej časti



Obr. 5.2: Rozohraná partia halmy medzi dvoma náhodnými agentmi

sa pozrieme, aké vhodné boli rôzne stratégie pre náš problém. Pred začatím samotného učenia však bolo treba skontrolovať funkčnosť prostredia tým že proti sebe postavíme dvoch náhodných súperov. Pre takéto testy sme sa rozhodli do nášho prostredia pridať aj funkciu render, ktorá hranie hry spomalí a hlavne bude v každom kroku vykresľovať plochu. Takto sme mohli vidieť, že náhodný agenti správne hýbu figúrkami (obr. 5.2). Títo hráči však sami od seba ku cieľu neprídu a teda je potrebné zapojiť neurónovú sieť. Zároveň máme možnosť vykresliť graf získaných odmien za jednotlivé hry odohrané agentom, čo nám pomôže lepšie vizualizovať proces učenia. Na kreslenie grafu používame knižnicu Plotly.

5.3.1 Hlboké Q-učenie

Ako sme spomínali vyššie, na obyčajné Q-učenie je tento próblém príliš komplexný. Je to tak preto, lebo Q-učenie sa spolieha na takzvanú Q-tabuľku v ktorej má uloženú kombináciu všetkých párov stav-akcia a k nim je priradená hodnota podľa toho, aké sú výhodné. Avšak ak je akčný priestor veľký, aj táto tabuľka naberie enormné veľkosti, spomalí sa jej prehľadávanie a tým pádom aj celý proces učenia. Narozdiel od tohto, hlboké Q-učenie nepoužíva žiadnu tabuľku a namiesto nej používa neurónovú sieť, ktorá aproximuje tieto hodnoty pre každú akciu v danom stave. [2]. Na tento algoritmus sme využili triedu DQNPolicy z knižnice Tianshou.

Literatúra

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. https://github.com/openai/gym.

[2] Issam El Naqa, Ruijiang Li, and Martin Murphy. *Machine Learning in Radiation Oncology: Theory and Applications*. 01 2015.

[3] Charlie Giattino, Edouard Mathieu, Veronika Samborska, and Max Roser. Artificial intelligence. *Our World in Data*, 2023. https://ourworldindata.org/artificial-intelligence.

[4] David Heath and Derek Allum. The historical development of computer chess and its impact on artificial intelligence. In *Proceedings of the 4th AAAI Conference on Deep Blue Versus Kasparov: The Significance for Artificial Intelligence*, AAAIWS'97-04, page 63–68. AAAI Press, 1997.

[5] Qiong Liu and Ying Wu. Supervised learning. 01 2012.

[6] Martin Müller. Computer go. *Artificial Intelligence*, 134(1):145–179, 2002.

[7] Vladimir Nasteski. An overview of the supervised machine learning methods. *Horizons. b*, 4:51–62, 2017.

[8] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton,

et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[11] David J. Staley. Digital historiography: Kasparov vs. deep blue. *Michigan Publishing*, 3(2), 2000.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, second edition, 2018.

[13] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

[14] Jordan Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall Williams, and Yashas Lokesh. PettingZoo: Gym for multi-agent reinforcement learning.

[15] Wu Yisi, Mohd Nor Akmal Khalid, and Hiroyuki Iida. Analyzing the sophistication of chinese checkers. *Entertainment Computing*, 34:100363, 2020.

[16] Jie Zhu, Fengge Wu, and Junsuo Zhao. An overview of the action space for deep reinforcement learning. ACAI '21, New York, NY, USA, 2022. Association for Computing Machinery.