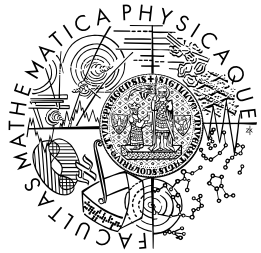

MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Marek Sádovský

Použitie strojového učenia pre deskovú hru Azul

Katedra softwaru a výuky informatiky (201. • 32-KSVI)

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.
Studijní program: Programování a vývoj software

Praha 2025

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Název práce: Použitie strojového učenia pre deskovú hru Azul

Autor: Marek Sádovský

Katedra: Katedra softwaru a výuky informatiky (201. • 32-KSVI)

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D., katedra

Abstrakt: Abstrakt.

Klíčová slova: klíčové slovo, složitější fráze

Title: Usage of the machine learning for the board game Azul

Author: Marek Sádovský

Department: Name of the department

Supervisor: RNDr. Tomáš Holan, Ph.D., department

Abstract: Abstract.

Keywords: key, words

Obsah

Úvod

Následuje několik ukázkových kapitol, které doporučují, jak by se měla závěrečná práce sázet. Primárně popisují použití TEXové šablony, ale obecné rady poslouží dobře i uživatelům jiných systémů.

1 Predstavenie hry

V hre Azul je cieľom čo najlepšie a čo najefektívnejšie uložiť dlaždice na stenu. Pri čom musia dlaždice odpovedať nejakému predom určenému vzoru.

1.1 Príprava hry

Každý hráč má na začiatku hernú dosku (ďalej len doska). Na pravej strane dosky môže každý hráč vidieť stenu pozostávajúcu z 5 krát 5 políčok kam sa budú časom dostávať dlaždice (**odkaz na kapitolu**). v ľavej časti sú zas zásobníky, do ktorých jednotlivý hráči vkladajú dlaždice počas hry. Vľavo hore na doske sa nachádza skóre daného hráča. Na záver na spodku dosky sa nachádza podlaha, miesto kam padajú dlaždice čo sa nám nezmestili do zásobníku (**odkaz**).

Okrem dosiek jednotlivých hráčov sú na hracej ploche aj výlohy s ponukami dlaždíc (**odkaz na postup pri hre**). Počet výloh je závislý na počte hráčov (**odkaz**). Okrem základných výloh existuje aj špeciálna centrálna výloha (ďalej len stred).

1.2 Priebeh hry

Samotná hra sa delí na dve fázy, ktoré sa periodicky striedajú kým nenastane koniec hry.

1.2.1 Ponuka výloh

Na začiatku tejto fázy sa naplnia všetky výlohy okrem stredu z banku (**odkaz**). Do stredu sa presunie dlaždica prvého hráča (**odkaz**). Ďalej sa hráči striedajú vo výbere výlohy a druhu dlaždice v nej. Keď si vyberú výlohu a v nej dlaždicu, zoberú z výlohy všetky dlaždice daného druhu (kliknutím na jednu z dlaždíc vo výlohe) a uložia ju do jedného z možných zásobníkov (kliknutím na zásobník). Zásobník musí vždy obsahovať maximálne jeden druh dlaždíc, ak pridá viac dlaždíc, ako je kapacita, tak nadbytočné dlaždice skončia na podlahe (**odkaz**).

Po úspešnom presunutí dlaždíc do zásobníku sa ostatné dlaždice z danej výlohy presunú do stredu. Nasleduje ťah ďalšieho hráča.

V momente keď sa prvý hráč rozhodne brať zo stredu tak si okrem vybraného druhu dlaždíc berie aj dlaždicu prvého hráča, ktorú si uloží na podlahu (**odkaz**).

V momente keď príde hráč na ťah a nemá si odial brať, všetky výlohy vrátane stredu sú prázdne, hra prechádza do druhej fázy.

1.2.2 Obkladanie stien

V tejto časti hráči presúvajú dlaždicu z plných zásobníkov do odpovedajúcich riadkov, pripisujú si okamžité body, následne vyprázdňujú podlahu a odčítavajú si za ňu bodu (**odkaz**).

Po tom, ako všetci hráči urobili vyššie popísaný krok, skontroluje sa či nenastal koniec hry, ten nastane ak niektorý z hráčov vyplnil celý riadok dlaždicami.

V prípade konca hry si hráči spočítajú bonusové body na základe nasledujúceho diagramu. - za každý celý riadok +2 body - za každý celý stĺpec +7 bodov - za 5 dlaždíc rovnakej farby na ploche +10 bodov

V prípade, že nenastal koniec hry opakuje sa prvá fáza hry

1.3 Parametre hry

Počet výloh je závislý od počtu hráčov: - pre 2 hráčov 5 výloh - pre 3 hráčov 7 výloh - pre 4 hráčov 9 výloh

V hre sa nachádza 100 unikátnych dlaždíc rovnomerne rozdelených na 5 druhov takže 20 dlaždíc každého druhu.

Doska obsahuje stenu o veľkosti 5 krát 5, 5 zásobníkov vo veľkostiach 1 až 5. Podlaha na spodku dosky má celkovú veľkosť 7. Ak by mal počet dlaždíc na podlahe prekročiť daný počet 7, je každá nadbytočná dlaždica odstránená.

Hodnotenie pri doplňaní na stenu je nasledovné. Ak doplnená dlaždica nesusedí s žiadnou inou dlaždicou tak dostaneme 1 bod, ak susedí aspoň s jednou ďalšou dlaždicou v rámci riadku alebo stĺpcu tak je počet bodov vypočítaný, ako súčet susediacich v riadku plus počet susediacich v stĺpci. (**odkaz**).

1.4 Ako hrať

Po spustení hry sa hráčovi/hráčom načíta menu kde uvidia tlačítko "Play", ktorým spustia hru, pod ním sa nachádza nastavenie na počet hráčov a ich druh, počet sa upravuje kliknutím na plus či mínus, a druh jednotlivých hráčov kliknutím na ikonu konkrétneho hráča, kde sú k dispozícii dve možnosti, AI (bot) alebo H (človek). Okrem toho kliknutím na meno hráča je možné ho zmeniť.

Po nastavení hráčov sa stlačením *Play* presunie hra do herného okna. Tu sa striedajú hráči v hraní, ako je popísané v priebehu hry, a to tak, že hráč na ťahu v rámci fáze jedna, Ponuka výloh, kliknutím na dlaždicu v rámci výloh, vyberie dlaždice z danej výlohy, daného druhu. Hráč ich môže ďalej položiť kliknutím pravého tlačidla myši, alebo ich môže uložiť do zásobníku kliknutím na zásobník, v prípade, že nemá žiadny zásobník môže ich uložiť na podlahu taktiež kliknutím na ňu.

V rámci druhej fázy je vidno preddefinovanú stenu hráča, takže hráč len klikne a automaticky sa presunie dlaždica z prvého plného zásobníku na stenu na správne miesto, toto iteruje kým má nejaký plný zásobník. V prípade, že nastane koniec hry po doplnení všetkých hráčov sa spočítajú bonusy a ukáže sa tabuľka s výsledkami.

2 Súvisiace práce

2.1 Aplikácia Reinforcement Learningu v hre Azul

Priame akademické publikácie o použití strojového učenia v deskovej hre Azul som nenašiel. Jediné zmienky sú vo forme open-source projektov a študentských prác. Napríklad repozitár "Azul-AI"uvádza použitie $TD(\lambda)$ učiaceho sa agenta [1], ktorý ale neukazuje finálnu funkčnú verziu. Projekt "AI-agent-Azul-Game-Competition"popisuje implementáciu MCTS (Monte-Carlo tree search), DQN (Deep Q-network) aj Minimax stratégií (finálny agent použil Minimax) [2]. Žiadny z výsledkov ale nebol nikde odborne publikovaný.

2.2 Reinforcement learning (RL) v iných deskových hrách

Medzi najstaršie zmienky o použití RL pri stolových hrách siaha do roku 1988 kde je štúdia na algoritmus TD-Gammon. Sieť sa sama učí len zo sebahraní pomocou TD( ). Výsledkom bolo „prekvapivo silné" hranie: agent dosiahol veľmi pokročilú úroveň (prekonal všetky doterajšie programy a aj siete trénované na ľudských hrách) [3]. Ďalej stojí za zmienku algoritmus AlphaZero, ktorý od nuly bez ľudských dát dosiahol superľudskú úroveň v šachu a šogi v priebehu 24 hodín trénovania. Ukazuje, že stratégia samo-hraním (self-play) s hlbokými sieťami dokáže zvládnuť náročné deskové hry (AlphaZero porazil v každej hre aj špičkové doménovo-špecifické programy)[4]. Pre o niečo komplexnejšie stolové hry dnes už tiež existujú isté algoritmy. V Deep Reinforcement Learning in Strategic Board Game Environments – Konstantia Xenou et al. (2019, EUMAS 2018, LNCS 11450). Autori navrhujú DRL architektúru (LSTM konvolučné siete s lokálnymi Q-hodnotami) pre viachernú hru Colonos de Catan (Settlers of Catan). Ich agent sa učí priamo počas hry (bez replay buffer a target sietí) a dosiahol lepšie výsledky ako existujúce heuristické riešenie jSettler [5].

2.3 Proximal Policy Optimization (PPO)

PPO je súčasťou rodiny metód založených na optimalizácii politiky. PPO optimalizuje „náhradnú" (surrogate) cieľovú funkciu s clipovacím členom, ktorý zabraňuje príliš veľkým aktualizáciám politík. Výhodou je jednoduchšia implementácia a dobrá sample-efficiency v porovnaní s predchádzajúcimi metódami (napr. TRPO). Experimentálne PPO preukázalo lepší výkon na bežných benchmarkoch vrátane robotického ovládania a Atari hier (ukazuje konzistentne lepšie výsledky než staršie online policy-gradient metódy).[6].

2.4 Deep Q-learning (DQN)

DQN prišiel na verejnosť v roku 2015 v článku "Human-level control through deep reinforcement learning", kde sa konvolučná sieť trénovaná metódou Q-learningu učí hrať 49 klasických Atari hier len zo surových pixelových vstupov. DQN agent dosiahol úroveň výkonu porovnateľnú alebo lepšiu ako profesionálny ľudský hráč naprieč rôznymi hrami. Článok dokazuje, že DQN dokáže zvládnuť vysokodimenzionálne vstupy a generalizovať medzi rôznymi hrami. [7]

3 Teoretické pozadie

V predchádzajúcej kapitole sme popísali pravidlá hry, skor ako sa pustíme priamo k práci treba si ešte predstaviť ako vyzerá daná knižnica aby bolo jenoznačné ako s ňou operovať.

- Podporované platformy: Linux, Windows - Jazyk: C s .net 8 - Knižnice: Unity knižnice (Unity, UnityEngine atď.)

3.1 Štruktúra knižnice

Knižnica má čisto backend-ové rozhranie, čo znamená, že neobsahuje žiadnu možnosť ovládania aplikácie, na to však máme vedlajšie riešenie. V princípe na hranie nie je potreba žiadna iná trieda takže konkrétne informácie o jednotlivých triedach sú dohladatelné v programátorskej dokumntácií. **TODO odkaz**

Board nám dáva možnosť zachytávať čo sa deje v hre pomocou eventov ktoré sa volajú keď hra očakáva nejakú interakciu z vonku. Z vonku sa volajú len dve metódy a to Move(), pre špecifikovanie výlohy dlaždice a zásobníka, a Calculate() na uloženie na stenu. Pre trénovanie umelej inteligencie može byť ešte zaujímavá metóda EncodeBoardState() ktorá vráti stav ako číselný vektor.

3.2 Reinforcement learning

Reinforcement learning (učenie formou odmeňovania). V rámci tejto metódy strojového učenia je agent (**odkaz**), nasadený do prostredia a učí sa na základe interakcií s prostredím.

Agentovi sú vopred určené pravidlá ako sa može správať a taktiež odmeňovacia funkcia, ktorá agentovi určuje či je jeho rozhodnutie pre neho prospešné alebo nie. Pomocou iteratívneho postupu sa agent metódou pokus omyl naučí ako sa správať v prostredí.

Prostredie nad ktorým budeme pracovať je hra Azul, spomínaná vyššie. Knižnica nám ponúka metódu EncodeBoardState, ktorá nám vracia vektor kde sú postupne zakódované stavy výloh, doska hráča čo je na ťahu a dosky ostatných hráčov. Pre zjednodušenie momentálny agent si vyberá len v rámci prvej fázy hry, kde existuje potenciálne 300 roznych akcií. Druhá fáza sa vie líšiť od štýlu hry, v rámci základnej hry, je deterministicky dané správanie.

Predpokladajme, že hrajú 4 hráči, to znamená, že máme 9 výloh plus stred, nech sa ťah skladá z troch čísel nejaké id výlohy, id dlaždice a id zásobníka, v tom prípade máme 10 výloh, 5 druhov dlaždíc, 5 zásobníkov a podlaha. 10 x 5 x (5 + 1) = 300.

3.3 Umelá inteligencia bez strojového učenia

Okrem strojového učenia existujú aj iné možnosti, umelej inteligencie, vo forme heruistik alebo pomocou stavových stromov (minmax). Tieto algoritmi **TODO bla bla**

4 Implemetácia proximal policy optimalizácie PPO

Táto kapitola detailne popisuje implementáciu algoritmu Proximal Policy Optimization (PPO) v jazyku C# pre potreby trénovania umelej inteligencie v hre Azul. Algoritmus bol implementovaný od základov bez použitia externých ML knižníc, čo umožnilo presnú kontrolu nad všetkými komponentmi systému.

4.1 Prostredie a agent

Prostredie predstavuje samotnú hru Azul, pričom každý stav hry je reprezentovaný, ako množina čísel opisujúcich aktuálne rozloženie dlaždíc **TODO ref** v rámci výloh, stav dosky hráča a stavy ostatných dosiek hráčov. Pre dosiahnutie menšieho vektora reprezentujúceho konkrétny stav, ho pred predaním kompresujeme na menšiu velkosť. Agent získava tieto informácie vo forme vstupného vektora, ktorý je ďalej spracovaný jeho modelom.

Priestor akcií majme definovaný, ako množinu všetkých možných pohybov existujúcich v hre. To preto, že v každom stave existuje inak veľká množina legálnych ťahov co by viedlo k zbytočným komplikáciam. Z tohoto dovodu používa agent v rámci ťahu maskovanie ilegálnych možností aby nebral do úvahy ťahy ktoré sú neplatné.

Agent, ako už spomínané v predchádzajúcej kapitole, využíva rozhranie IBot na komunikáciu s herným prostredím. Pri každom ťahu agent vykoná nasledovné kroky:

1. Zakóduje aktuálny stav hry do numerického vektora.
2. Pomocou policy siete získa pravdepodobnosti možných akcií.
3. Vynuluje pravdepodobnosti neplatných akcií a opätovne normalizuje zvyšné.
4. Na základe pravdepodobností náhodne vzorkuje akciu.
5. Zaznamená stav, akciu, pravdepodobnosti a priebežnú odmenu pre neskorší tréning.

4.2 Architektúra modelu

Politika (policy) a aj hodnotová funkcia (value function), používajú implementáciu jednoduchej doprednej neurónovej siete (feedforward network) so zdielanými vrstvami.

Architektúra našej neurénovej siete pozostáva z týchto vrstiev:

- Vstupná vrstva: počet neurónov zodpovedá veľkosti vektoru stavu.
- Skrytá vrstva 1: 256 neurónov, aktivačná funkcia ReLU

- Skrytá vrstva 2: 256 neurónov, aktivačná funkcia ReLU **TODO: vysvetlit preco tato velkost**
- Výstupná vetva politiky: softmax nad počtom možných akcií
- Výstupná vetva hodnoty: jeden neurón (skalárna hodnota stavu)

Implementácia siete prebieha v C# prostredníctvom vlastnej triedy NeuralNetwork, ktorá podporuje základné operácie ako dopredný prechod, spätnú propagáciu a optimalizáciu parametrov.

4.3 Výhodnosť a optimalizačný krok PPO

Agent počas hrania zbiera trajektóriu stavov, akcií a odmien v každej epizóde. Odmeny r_t sú nasledovne diskontované pomocou diskontného faktoru $\gamma \in [0,1)$ následovne:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Kde R_t je tzv. discounted return, diskontovaná kumulatívna odmena od času t.

Na rozdiel od klasického policy gradientu, PPO zavádza tzv. clipped surrogate objective, ktorý bráni príliš veľkým zmenám v politike medzi iteráciami. Základný tvar straty (loss) pre policy sieť je:

$$L^{CLIP}(\theta) = \mathbb{E}[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

Kde:

- θ sú parametre aktuálnej politiky
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{OLD}}(a_t|s_t)}$ je pomer pravdepodobností akcie pred a po aktualizacií politiky.
- ϵ je hiperparameter, ktorý určuje o koľko sa môže politika zmeniť
- funkcia clip() zaručuje že aktualizácia politiky nepresiahne povolený rozsah

Táto funkcionalita zabezpečuje, že aktualizácia sa bude riadiť výhodnosťou, ale zároveň nedovolí politike „skočiť“ príliš ďaleko, čím sa zvyšuje stabilita tréningu.

Na podporu explorácie počas tréningu sa do stratovej funkcie pridáva aj entropia výstupu politiky:

$$L^{total} = L^{CLIP}(\theta) + c \cdot H[\pi\theta(\cdot|s_t)]$$

Kde:

- $$H[\pi_\theta] = -\sum^{i} p_i \cdot log(p_i)$$

  - je entropia politiky

- c je váhový koeficient určujúci vplyv entropie na celkovú stratu

Po výpočte straty paramentre politiky aktualizujeme pomocou gradientného zostupu (gradient descent) v rámci spätnej propagácie (backpropagation). Ide o štandardný prístup pri optimalizácii parametrov siete, ktorý spočíva v iteratívnej aktualizácii váh a biasov tak, aby sa minimalizovala hodnota straty (loss funkcie).

4.4 Technické výzvy

Implementácia PPO priniesla viacero technických výziev. Medzi najväčšie patrila absencia pokročilých ML knižníc, keďže pracujeme v jazyku C#, čo si vyžadovalo ručné implementovanie všetkých komponentov: neurónových sietí, stratových funkcií, optimalizácie a numerických operácií. Z počiatku sa ukazoval problém so stabilitou, hodnoty v rámci výpočtov v neurónovej sieti často viedli k veľkým číslam umožnujúcim pretečenie alebo naopak v log funkciach k nule kde hrozil nedefinovaný stav. Ako riešenie tohoto problému sme použili clip hodnoty na obmedzenie hodnot a epsilon na zabránenie $log(0)$.

5 Konfigurácie PPO

Vzhľadom na komplexitu hry Azul nie je jednoduché určiť aká konfigurácia parametrov je najideálnejšia, či sa už jedná o velkosť neurónovej siete alebo o rýchlosť učenia.

5.1 Hodnotiace funkcie

Hra Azul je náhodná a má velké spektrum stratégií, preto si vytvoríme viaceré možnosti na porovnanie.

5.1.1 Maximalizácia okamžitého zisku a ignorovanie steny

Algoritmus 1 V hodnotení podľa maximalizácie okamžitého zisku sa ignorujú odložené hodnotenia a tým sa zanedbáva dlhodobejšia stratégia.

```
1: function VyhodnotStav(tah, stav)
2:     reward = 0 ← Nastavíme základnú hodnotu
3:     if tah == TahNaPodlahu then
4:         Vrátíme -10                          ▷ ukladat na podlahu je strata
5:     end if
6:     novyPocetNaDoske = tah.Pocet + stav.nasaDoska.pocetVRiadku
7:     if novyPocetNaDoske >= stav.nasaDoska.kapacitaRiadku then   ▷
   Znamená, že dotaneme body v tomto kole
8:         reward += 5                          ▷ 5 je maximálna velkosť riadku
9:         reward -= novyPocetNaDoske − stav.nasaDoska.kapacitaRiadku ▷
   Nechceme pridávať na podlahu
10:    else
11:        reward += tah.Pocet
12:    end if
13:    Vrátime reward
14: end function
```

Toto riešenie je jednoduché a javý sa účine ale len voči náhodnemu oponentovi. Pokiaľ oponent používa nejakú stratégiu nemá problém poraziť agenta s touto hodnotiacou funkciou. **TODO viac graf**

5.1.2 Maximalizácia okamžitého zisku s rešpektom ku stene

Na rozdiel od predchádzajúceho riešenia teraz, budeme brať do úvahy aj to ako to čo berieme interaguje už s uloženými vecami na stene.

V algoritme 2 sa nachádzajú konštanty c, c1, c2, c3 ktoré slúzia na jednotlivé upravenie výšky za jednotlivé časti hodnoty.

Algoritmus 2 V hodnotení podľa maximalizácie okamžitého zisku sa ignorujú odložené hodnotenia a tým sa zanedbáva dlhodobejšia stratégia.

```
1: function VyhodnotStav(tah, stav)
2:     reward = 0 ← Nastavíme základnú hodnotu
3:     nasaDoska = stav.dosky[naseId]   ▷ od stavu dosky zalezi pocet bodov
4:     if tah == TahNaPodlahu then
5:         Vrátime -10                          ▷ ukladat na podlahu je vždy zlé
6:     end if
7:     novyPocetNaDoske = tah.Pocet + nasaDoska.pocetVRiadku
8:     pocetBodovZaTah = nasaDoska.PocetZaPridanieDlazdice(tah.Dlazdica)
9:     vyplnene = nasaDoska.velkostSteny − nasaDoska.prazdneNaStene
10:    reward -= vyplnene
                ▷ kompenzacia rastu rewardu na zaklade bonusových bodov
11:    if novyPocetNaDoske >= nasaDoska.kapacitaRiadku then   ▷
   Znamená, že dotaneme body v tomto kole
12:        reward += c                                   ▷ menší základ
13:        reward += c1 ∗ pocetBodovZaTah
       reward− = novyPocetNaDoske − stav.nasaDoska.kapacitaRiadku   ▷
   Nechceme pridávať na podlahu
14:    else
15:        reward += c2
16:        reward += c3 * pocetBodovZaTah
17:    end if
18:    Vrátime reward
19: end function
```

5.1.3 Maximalizácia okamžitého zisku s rešpektom k stene a plným zásobníkom

K predchádzajúcemu riešeniu pridáme extra funkciu ktorá nám pripravý stenu tak ako bude vyzerat ked budeme pridávať konkrétny riadok.

Algoritmus 3 V tejto funkcií vylepšíme stenu o už doplnené zásobníky

```
1: function PridatNaStenu(staraStena, zasobniky, aktualnyZasobnik)
2:     novaStena = staraStena ← Pripravíme stenu
3:     for zasobnikinzsobnky do
4:         if zsobnk == aktualnyZsobnk then
5:             Vrátime novaStena
6:         end if
7:         if zsobnk.jePln then
8:             novaStena.TODO
9:         end if
10:    end for
11:    Vrátime novaStena
12: end function
```

v zápätí pridáme volanie na túto funkciu do algoritmu 2 a to tak že najprv aktoalizujeme našu dosku a až následne vypočítame bonus. TODO

Závěr

Literatura

1. Irene XiangYi Chen CCZ, Kevin Miao. *Azul-AI*. Dostupné také z: `https://github.com/irenexychen/Azul-AI`.
2. Hongsang Yoo YuqingXiao97, sheejack97. *AZUL game AI-agent competition*. Dostupné také z: `https://github.com/kaiyoo/AI-agent-Azul-Game-Competition`.
3. Gerald, Tesauro. TD-Gammon: A Self-Teaching Backgammon Program. In: *Applications of Neural Networks*. Ed. Murray, Alan F. Boston, MA: Springer US, 1995, s. 267–285. isbn 978-1-4757-2379-3. Dostupné z doi: `10.1007/978-1-4757-2379-3_11`.
4. Silver, David; Hubert, Thomas; Schrittwieser, Julian; Antonoglou, Ioannis; Lai, Matthew; Guez, Arthur; Lanctot, Marc; Sifre, Laurent; Kumaran, Dharshan; Graepel, Thore; Lillicrap, Timothy; Simonyan, Karen; Hassabis, Demis. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. Dostupné z arXiv: `1712.01815`.
5. Xenou, Konstantia; Chalkiadakis, Georgios; Afantenos, Stergos. Deep Reinforcement Learning in Strategic Board Game Environments. In: Slavkovik, Marija (ed.). *Multi-Agent Systems*. Cham: Springer International Publishing, 2019, s. 233–248. isbn 978-3-030-14174-5.
6. Schulman, John; Wolski, Filip; Dhariwal, Prafulla; Radford, Alec; Klimov, Oleg. *Proximal Policy Optimization Algorithms*. 2017. Dostupné z arXiv: `1707.06347`.
7. Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Rusu, Andrei A; Veness, Joel; Bellemare, Marc G; Graves, Alex; Riedmiller, Martin; Fidjeland, Andreas K; Ostrovski, Georg et al. Human-level control through deep reinforcement learning. *Nature*. 2015, roč. 518, č. 7540, s. 529–533.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A Přílohy

A.1 První příloha