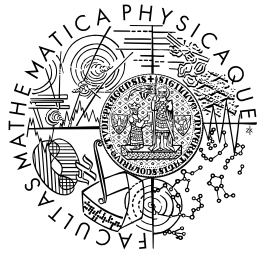


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

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 **7**

1 Pravidlá hry Azul **8**
1.1 Výber dlaždíc . . . 8
1.2 Umiestňovanie dlaždíc . . . 8
1.3 Príprava na ďalšie kolo . . . 8
1.4 Koniec hry a bodovanie . . . 8

2 Vlastné spracovanie hry **9**
2.1 Začiatok hry . . . 9
2.2 Priebeh hry . . . 9
2.2.1 Výber dlaždíc . . . 9
2.2.2 Umiestňovanie dlaždíc . . . 10
2.3 Parametre hry . . . 10

3 Súvisiace práce **11**
3.1 Aplikácia Reinforcement Learningu v hre Azul . . . 11
3.2 Reinforcement learning (RL) v iných deskových hrách . . . 11
3.3 Proximal Policy Optimization (PPO) . . . 11
3.4 Deep Q-learning (DQN) . . . 12

4 Programátorská časť **13**
4.1 Návrh riešenia . . . 13
4.2 Implementácia . . . 14
4.2.1 Trieda `Board` . . . 14
4.2.2 Trieda `Plate` . . . 15
4.2.3 Trieda `CenterPlate` . . . 16
4.2.4 Trieda `Player` . . . 16
4.2.5 Trieda `Tiles` . . . 16
4.2.6 Štruktúra `Buffer` . . . 17
4.2.7 Štruktúra `Tile` . . . 17
4.2.8 Záznam `Move` . . . 17
4.2.9 Statická trieda `Globals` . . . 18
4.2.10 Trieda `IllegalOptionException` . . . 18
4.2.11 Statická trieda `Logger` . . . 18
4.3 Jednoduchá umelá inteligencia . . . 18

5 Implementácia Deep Q-learning (DQN) **19**
5.1 Prostredie a agent . . . 19
5.2 Architektúra modelu . . . 20
5.3 Trénovanie DQN . . . 20
5.4 Technické výzvy . . . 20

6 Implemetácia proximal policy optimalizácie PPO **21**
6.1 Prostredie a agent 21
6.2 Architektúra modelu 21
6.3 Výhodnosť a optimalizačný krok PPO 22
6.4 Technické výzvy 23

7 Konfigurácie PPO **24**
7.1 Hodnotiace funkcie 24
7.1.1 Maximalizácia okamžitého zisku a ignorovanie steny . . . 24
7.1.2 Maximalizácia okamžitého zisku s rešpektom ku stene . . 24
7.1.3 Maximalizácia okamžitého zisku s rešpektom k stene a plným zásobníkom 26

8 Experimenty a výsledky **27**
8.1 Výsledky PPO 27

Závěr **31**

Literatura **32**

Seznam obrázků **33**

Seznam tabulek **34**

Seznam použitých zkratek **35**

A Přílohy **36**
A.1 První příloha 36

Úvod

Stolová hra Azul, ktorú navrhol nemecký herný dizajnér Michael Kiesling a prvýkrát vydal v roku 2017, predstavuje abstraktnú strategickú hru inšpirovanú tradičným portugalským umením zdobenia keramických dlaždíc známym ako azulejos. [1] Hráči sa vžívajú do úlohy umelcov, ktorí zdobia steny kráľovského paláca v portugalskom meste Évora farebnými dlaždicami, čím sa snažia vytvoriť čo najkrajšiu mozaiku. Hra získala viaceré ocenenia, vrátane prestížneho titulu Spiel des Jahres (Hra roka) v roku 2018, a je oceňovaná pre svoju estetiku, jednoduché pravidlá a strategickú hĺbku.

1 Pravidlá hry Azul

Kompletné pravidlá sa dajú nájsť na stránke výrobcu *Mindok* [2]. A v tejto kapitole sa budeme venovať podstatným častiam z nich.

Každý hráč má svoju vlastnú hraciu dosku, ktorá je rozdelená na tri časti: prípravné rady (ďalej aj zásobníky), mozaikovú stenu (ďalej len stena) a podlahu pre prebytočné dlaždice. Hra prebieha v niekoľkých kolách, z ktorých každé pozostáva z troch fáz: výber dlaždíc, umiestňovanie dlaždíc a príprava na ďalšie kolo.

1.1 Výber dlaždíc

Na začiatku každého kola sú na kruhové výlohy (ich počet závisí od počtu hráčov) náhodne umiestnené štyri dlaždice z vrecka. Hráči sa striedajú v ťahoch, počas ktorých si vyberajú všetky dlaždice jednej farby z jednej výlohy a zvyšné dlaždice z tejto výlohy presúvajú do stredu stola. Alternatívne môžu hráči vybrať všetky dlaždice jednej farby zo stredu stola. Prvý hráč, ktorý tak urobí, si vezme aj žetón začínajúceho hráča, ktorý umiestni na svoju podlahu, čo mu prinesie záporné body. Následne hráč uloží vopred vybrané dlaždice do jedného zo svojich zásobníkov, pričom musí dodržať, že v jednom zásobníku môžu byť len dlaždice jedného typu. Dlaždice, čo sa do vybraného zásobníku nezmestia, idú na podlahu.

1.2 Umiestňovanie dlaždíc

Po vyčerpaní všetkých dlaždíc z výloh a stredu stola hráči presúvajú dlaždice z plne zaplnených zásobníkov na príslušné miesta na stene. Pri umiestnení dlaždice získava hráč body: jedna samostatná dlaždica prináša jeden bod, avšak ak nadväzuje na ďalšie dlaždice vo vodorovnom alebo zvislom smere, hráč získa body za každú nadväzujúcu dlaždicu v danom smere. Prebytočné dlaždice, ktoré nebolo možné umiestniť a teraz sa nachádzajú na podlahe, prinášajú záporné body.

1.3 Príprava na ďalšie kolo

Hráči odstránia nepoužité dlaždice z podlahy a zo zásobníkov, ktoré boli v predchádzajúcej fáze plné a bola z nich dlaždica uložená na stenu. Následne sa doplnia výlohy novými dlaždicami z vrecka a začína sa nové kolo.

1.4 Koniec hry a bodovanie

Hra končí po kole, v ktorom aspoň jeden hráč dokončí vodorovný rad piatich dlaždíc na stene. Nasleduje záverečné bodovanie, počas ktorého hráči získavajú bonusové body za každú kompletnú vodorovnú radu (2 body), zvislý stĺpec (7 bodov) a za každú sadu piatich dlaždíc rovnakej farby umiestnených na stene (10 bodov). Hru vyhráva hráč, ktorý mal najviac bodov.

2 Vlastné spracovanie hry

Hra je implementovaná pomocou jazyka C verzia .net 8, s podporou na Windows aj Linux. Pri implementácii hry bol použitý herný engine **Unity** [3]. Použijeme projekt s 2D scénami, keďže nie je potreba pre 3D scény. V rámci tejto kapitoly sa venujeme výhradne užívateľskej dokumentácii našej implementácie hry Azul. Umelej inteligencii sa venujeme až v kapitole **TODO** a programátorskú dokumentáciu môžete nájsť **TODO**.

2.1 Začiatok hry

Po tom, ako zapneme hru, sa nám zobrazí menu **todo**. V rámci menu vieme pomocou tlačidiel plus a mínus pridávať či odoberať hráčov. Kliknutím na tlačidlo hráča (tlačidlo s nápisom H alebo AI) prepíname medzi tým, či je daný hráč človek alebo umelá inteligencia. Pod každým hráčom sa nachádza text, kde si vie hráč zmeniť meno priamo kliknutím na text a jeho následnou úpravou. Pre spustenie hry je následne potrebné, aby jeden z hráčov stlačil tlačidlo Play **TODO**.

2.2 Priebeh hry

Počas hry, keď je hráč na ťahu, vidí pred sebou svoju hraciu dosku, výlohy a na ľavej strane dosky ostatných hráčov. Na základe **Pravidiel** z predchádzajúcej kapitoly 1.

2.2.1 Výber dlaždíc

Počas tejto fázy má hráč možnosť si vybrať jednu farbu z ľubovoľnej výlohy alebo zo stredu. Následne kliknutím ľavého tlačidla myši označí vybranú farbu z vybranej výlohy. Pokiaľ má hráč vybrané dlaždice, hýbu sa spolu s jeho kurzorom. Hráč má dve možnosti, vybrať si zásobník, do ktorého chce uložiť zobrané dlaždice, pri dodržaní pravidiel 1.1, a uložiť ich pomocou kliknutia ľavým tlačidlom myši na daný zásobník. Alebo môže hráč kliknúť pravým tlačidlom a tým vrátiť vybrané dlaždice a môže si vybrať iné.

Ak sa hráč pokúsi vybrané dlaždice uložiť do zásobníku, kam ich podľa pravidiel nesmie uložiť, vybrané dlaždice sa vrátia a hráč môže znova vyberať.

Ak sa hráč rozhodne vybrať si dlaždice zo stredu a je prvý, ktorý sa takto rozhodol, automaticky sa mu pri uložení dlaždíc do zásobníku presunie na podlahu žetón prvého hráča 1.1.

Po úspešnom presunutí dlaždíc do zásobníku sa ostatné dlaždice z danej výlohy presunú do stredu. Dlaždice, ktoré sa do zásobníku nezmestia, sú presunuté na podlahu. Následne kliknutím ľubovoľného klávesu, či tlačidla myši sa ťah ukončí. Nasleduje ťah ďalšieho hráča.

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

2.2.2 Umiestňovanie dlaždíc

Hráč, ktorý príde na ťah v tejto fáze, vidí rovnaký pohľad ako v predchádzajúcej fáze, jediný rozdiel je, že sú všetky výlohy prázdne, vrátane stredu. V tejto fáze hráč kliknutím ľavého tlačidla myši na ľubovoľnom mieste okna hry automaticky presunie dlaždicu z plného zásobníka na korešpondujúce miesto na stene. Následne sa automaticky pripočítajú body podľa pravidiel. Každým kliknutím sa presunie len z jedného zásobníka na stenu a presúva sa postupne odhora nadol. Spolu s posledným zásobníkom sa vyprázdni aj podlaha, spolu s ňou teda dostaneme aj záporné body, ktoré nám zredukujú celkový počet bodov.

Po tom, ako všetci hráči urobili vyššie popísaný krok, skontroluje sa, či nenastal koniec hry. Ak áno, prebehne počítanie bonusových bodov 1.4 a následne sa zobrazí tabuľka s výsledkami. Ak nenastal koniec hry, hra sa automaticky pripraví na ďalšiu fázu výberu dlaždíc, kde bude začínať hráč, ktorý mal naposledy na podlahe žetón prvého hráča.

2.3 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 hráča obsahuje mozaikovú stenu o veľkosti 5 krát 5. Ďalej 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.

3 Súvisiace práce

3.1 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 [4], 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) [5]. Žiadny z výsledkov ale nebol nikde odborne publikovaný.

3.2 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) [6]. 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)[7]. 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 [8].

3.3 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).[9].

3.4 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. [10]

4 Programátorská časť

V predchádzajúcej kapitole sme popísali pravidlá hry, skôr ako sa pustíme priamo k riešeniu umelej inteligencie, predstavme si ako vyzerá naša knižnica, pre jednoduchšie operovanie s ňou.

- Podporované platformy: Linux, Windows
- Jazyk: C# with .net 8
- Knižnice: Unity libraries (Unity, UnityEngine, etc.)

4.1 Návrh riešenia

- `Board`: Trieda reprezentujúca hernú dosku a riadiaca celkový priebeh hry. Sprostredkúva interakciu medzi hernou logikou a používateľským rozhraním.
- `Plate`: Abstraktná trieda reprezentujúca zdroj dlaždíc (výlohy).
- `CenterPlate`: Trieda dediaca od `Plate`, špecifická pre stred stola.
- `Player`: Trieda reprezentujúca hráča, jeho hraciu dosku. V nej nájdeme stenu (`Wall`), zásobníky (`Buffers`), podlahu (`Floor`) a počet bodov. Obsahuje logiku pre ukladanie dlaždíc, výpočet bodov a kontrolu či daný hráč neukončil hru.
- `Tiles`: Trieda reprezentujúca kolekciu dlaždíc určenú typami a počtami. Poskytuje metódy pre náhodný výber, zistenie počtu dlaždíc daného typu, odobratie a pridanie dlaždíc, a zjednotenie s inou triedov `Tiles`.
- `Buffer`: Štruktúra reprezentujúca riadok na hráčovej doske pre dočasné ukladanie dlaždíc. Udržiava informácie o maximálnej kapacite, aktuálne uloženom type dlaždíc a počte.
- `Tile`: Štruktúra reprezentujúca jednu sadu dlaždíc určitého typu a počtu.
- `Move`: Záznam (record) reprezentujúci herný ťah (ID dlaždice, ID výlohy-/stredu, ID zásobníka).
- `Globals`: Statická trieda obsahujúca konštanty a globálne nastavenia hry.
- `IllegalOptionException`: Vlastná trieda výnimky pre indikáciu neplatných herných operácií.
- `Logger`: Statická trieda pre jednoduché logovanie priebehu hry do súboru.

Komunikácia s frontendom je zabezpečená pomocou udalostí (`EventHandler`):

- `NextTakingMove`: Udalosť signalizujúca, že je na rade ďalší hráč vo fáze ťahania dlaždíc.
- `NextPlacingMove`: Udalosť signalizujúca, že začína fáza ukladania dlaždíc.

4.2 Implementácia

4.2.1 Trieda `Board`

- **`public Player[] Players { get; private set; }`**: Pole objektov `Player` reprezentujúcich hráčov.
- **`public Plate[] Plates { get; private set; }`**: Pole objektov `Plate` reprezentujúcich výlohy.
- **`public CenterPlate Center { get; private set; }`**: Objekt `CenterPlate` reprezentujúci stred stola.
- **`public Tiles Storage { get; private set; }`**: Objekt `Tiles` reprezentujúci vrecko z dlaždicami.
- **`public int CurrentPlayer { get; private set; }`**: Index aktuálneho hráča.
- **`public Phase Phase { get; private set; }`**: Enum `Phase` určujúci aktuálnu fázu hry (`Taking`, `Placing`, `GameOver`).
- **`public static int[,] PredefinedWall { get; private set; }`**: Statická dvojrozmerná matica definujúca rozloženie dlaždíc na stene v základnej verzii hry.
- **`public bool IsAdvanced { get; private set; }`**: Booleovská hodnota určujúca, či sa hrá pokročilá verzia hry.
- **`public bool FisrtTaken { get; set; }`**: Booleovská hodnota indikujúca, či bola už z centra stola zobratá prvá dlaždica v danom kole.
- **`public event EventHandler<MyEventArgs> NextTakingMove;`**: Udalosť signalizujúca ďalší ťah vo fáze ťahania.
- **`public event EventHandler<MyEventArgs> NextPlacingMove;`**: Udalosť signalizujúca začiatok fázy ukladania.
- **`public Board(...)`**: Verejný konštruktor pre vytvorenie novej hry s daným počtom hráčov, ich menami a nastavením pokročilej verzie a logovacieho súboru.
- **`public void StartGame()`**: Verejná metóda na inicializáciu a spustenie novej hry.
- **`public bool CanMove(Move move)`**: Verejné metódy na overenie, či je daný ťah platný.
- **`public bool Move(Move move)`**: Verejné metódy na vykonanie ťahu hráča, vracia rovnakú hodnotu a `CanMove`.
- **`public bool Calculate(int col = Globals.EmptyCell)`**: Verejná metóda na spracovanie ukladania dlaždíc z hráčovho zásobníka na stenu a výpočet bodov. V základnej verzií sa neočakáva parameter col z dôvodu preddefinovanej steny.

- **`public Move[] GetValidMoves()`**: Verejná metóda vracajúca pole platných ťahov pre aktuálneho hráča.
- **`public double[] EncodeBoardState(int id)`** : Verejná metóda na zakódovanie aktuálneho stavu hry do poľa čísel pre AI.
- **`public static double[] GetMyPlayerData(double[] state)`**: Statická verejná metóda na extrahovanie dát jedného hráča zo zakódovaného stavu.
- **`public static Player DecodePlayer(double[] playerData)`**: Statická verejná metóda na dekódovanie dát hráča z poľa čísel.
- **`public static double[] GetNextState(double[] state, Move move)`**: Statická verejná metóda vracajúca zakódovaný stav hry po vykonaní daného ťahu.
- **`public static int FindColInRow(int row, int typeId)`**: Statická verejná metóda na určenie stĺpca pre uloženie dlaždice na stene (základná verzia).
- **`public int[] GetPlatesData()`**: Verejná metóda vracajúca pole zakódovaných dát pre všetky výlohy a stred stola.
- **`public int EncodePlateData(Plate plate) / static int EncodePlateData(int plateData)`**: Verejné metódy na zakódovanie dát výlohy.
- **`public static int EncodeBufferData(Tile tile) / static int EncodeBufferData(int[] arrData)`**: Statické verejné metódy na zakódovanie dát zásobníka hráča.
- **`public static int[] DecodeBufferData(int encoded)`**: Statická verejná metóda na dekódovanie dát zásobníka hráča.
- **`public static int[] DecodePlateData(int encoded)`**: Statická verejná metóda na dekódovanie dát továrne.
- **`protected virtual void OnNextTakingMove(MyEventArgs e)`**: Virtuálna chránená metóda na vyvolanie udalosti `NextTakingMove`.
- **`protected virtual void OnNextPlacingMove(MyEventArgs e)`**: Virtuálna chránená metóda na vyvolanie udalosti `NextPlacingMove`.

4.2.2 Trieda `Plate`

- **`public bool isEmpty { get; protected set; }`**: Verejná vlastnosť indikujúca, či je výloha prázdna.
- **`public Tile[] GetCounts()`**: Verejná metóda vracajúca pole štruktúr `Tile`, kde každá štruktúra obsahuje ID typu dlaždice a jej počet vo výlohe.
- **`public int TileCountOfType(int typeId)`**: Verejná metóda vracajúca počet dlaždíc daného typu (`typeId`) vo výlohe.

4.2.3 Trieda `CenterPlate`

- **`public bool isFirst { get; protected set; }`**: Verejná vlastnosť indikujúca, či bola z centra už zobratá prvá dlaždica v aktuálnom kole (pravidlá 1.1).

4.2.4 Trieda `Player`

- **`public string name { get; private set; }`**: Verejná vlastnosť pre meno hráča.
- **`public int pointCount { get; private set; }`**: Verejná vlastnosť pre aktuálny počet bodov hráča.
- **`public int[,] wall { get; private set; }`**: Verejná vlastnosť reprezentujúca hráčovu stenu, dvojrozmerné pole s ID uložených dlaždíc.
- **`public List<int> floor { get; private set; }`**: Verejná vlastnosť reprezentujúca hráčovu podlahu, zoznam ID dlaždíc, ktoré sa nezmestili do zásobníkov.
- **`public bool isFirst { get; private set; }`**: Verejná vlastnosť indikujúca, či bude hráč začínať nasledujúce kolo.
- **`public event EventHandler? OnWin;`**: Verejná udalosť, ktorá sa vyvolá, keď hráč dokončí riadok na svojej stene.1.4
- **`public bool CanPlace(int row, int tileId)`**: Verejná metóda overujúca, či môže hráč uložiť dlaždicu daného typu (`tileId`) do daného zásobmíka (`row`). Riadok `Globals.WallDimension` reprezentuje podlahu.
- **`public Tile GetBufferData(int row)`**: Verejná metóda vracajúca informácie o obsahu daného zásobníka (`row`).
- **`public int[] GetFullBuffersIds()`**: Verejná metóda vracajúca pole ID riadkov zásobníkov, ktoré sú plné.
- **`public bool HasFullBuffer()`**: Verejná metóda zistí, či má hráč aspoň jeden plný zásobník.
- **`public void CalculateBonusPoints()`**: Verejná metóda na výpočet bonusových bodov na konci hry za dokončené riadky, stĺpce a sady rovnakej farby na stene.1.4
- **`public int AddedPointsAfterFilled(int row, int col)`**: Verejná metóda na výpočet bodov získaných za práve pridanú dlaždicu na stenu.

4.2.5 Trieda `Tiles`

- **`public int typesCount { get; }`**: Verejná vlastnosť pre počet typov dlaždíc, ktoré táto kolekcia obsahuje.

- **`public Tiles(int typesCount, int totalCount)`**: Verejný konštruktor pre vytvorenie kolekcie dlaždíc s daným počtom typov a celkovým počtom dlaždíc (rovnomerne rozdelených).
- **`public Tiles GetRandom(int count)`**: Verejná metóda na náhodné vybratie daného počtu (`count`) dlaždíc z kolekcie a vrátenie novej kolekcie s vybranými dlaždicami.
- **`public int TileCountOfType(int id)`**: Verejná metóda vracajúca počet dlaždíc daného typu (`id`) v kolekcii.
- **`public int TotalTiles()`**: Verejná metóda vracajúca celkový počet dlaždíc v kolekcii.

4.2.6 Štruktúra `Buffer`

- **`public int Size { get; }`**: Verejná vlastnosť pre maximálnu kapacitu zásobníka.
- **`public int TypeId { get; private set; }`**: Verejná vlastnosť pre ID typu dlaždice, ktorý je aktuálne v zásobníku (alebo `Globals.EmptyCell`, ak je prázdny).
- **`public int CurrentlyFilled { get; private set; }`**: Verejná vlastnosť pre aktuálny počet dlaždíc v zásobníku (alebo `Globals.EmptyCell`, ak je prázdny).
- **`public bool CanAssign(int id)`**: Verejná metóda overujúca, či je možné pridať dlaždice daného typu (`id`) do zásobníka.
- **`public int FreeToFill()`**: Verejná metóda vracajúca počet voľných miest v zásobníku.
- **`public bool IsFull()`**: Verejná metóda zistí, či je zásobník plný.

4.2.7 Štruktúra `Tile`

- **`public int Id { get; }`**: Verejná vlastnosť pre ID typu dlaždice.
- **`public int Count { get; }`**: Verejná vlastnosť pre počet dlaždíc daného typu.
- **`public Tile(int id, int count)`**: Verejný konštruktor pre vytvorenie inštancie `Tile`.

4.2.8 Záznam `Move`

- **`public readonly int TileId;`**: Verejná vlastnosť pre ID typu dlaždice v ťahu.
- **`public readonly int PlateId;`**: Verejná vlastnosť pre ID továrne (alebo `Plates.Length` pre stred stola) v ťahu.

- `public readonly int BufferId;`: Verejná vlastnosť pre ID riadku zásobníka (alebo `Globals.WallDimension` pre podlahu) v ťahu.
- `public Move(int tileId, int plateId, int bufferId)`: Verejný konštruktor pre vytvorenie inštancie `Move`.
- `public override string ToString()`: Verejná metóda pre textovú reprezentáciu ťahu.

4.2.9 Statická trieda `Globals`

Obsahuje verejné konštanty definujúce rôzne aspekty hry (počet typov dlaždíc, rozmery steny, celkový počet dlaždíc, kapacita tovární, špeciálne hodnoty pre prázdnu bunku a prvú dlaždicu, veľkosť podlahy, počet zásobníkov a predvolený ťah).

4.2.10 Trieda `IllegalOptionException`

Verejná trieda výnimky, ktorá sa používa na signalizáciu neplatných herných operácií.

4.2.11 Statická trieda `Logger`

Poskytuje verejné statické metódy `WriteLine` a `Write` na zapisovanie textových správ do logovacieho súboru. Metóda `SetName` umožňuje nastaviť názov logovacieho súboru.

4.3 Jednoduchá umelá inteligencia

Treba si ešte predstaviť jednoduchých agentov na neskoršie porovnanie voči cielenej umelej inteligencii. V rámci tohto riešenia máme dvoch takýchto agentov.

- **RandomBot** reprezentuje agenta ktorý si vyberá akciu úplne náhodne z možných ťahov danom momente.
- **HeuristicBot** jedná sa o agenta, ktorý si vyberá možnosť ktorá prináša najväčší okamžitý zisk a preferuje brať si viac dlaždíc. Ako menej, pričom sa snaží neukladať na podlahu. V rámci ľudského testovania sa ukázal tento bot celkom porovnateľný z bežným hráčom.

5 Implementácia Deep Q-learning (DQN)

Základná idea viacerých reinforcement learning algoritmov je zisťovanie akcia-hodnota funkcie pomocou Bellmanovej rovnice ako iteratívna aktualzácia. $Q_{i+1}(s,a) = r+\gamma max_{a'} Q_i(s',a')$ Takýto iteračný algoritmus nám konverguje k optimálnej akcia-hodnota funkcí $Q_{i+1} \to Q^*$ pre $i \to \infty$. Je bežné použiť aproximátor ako napríklad neurónovú sieť, pre možnú prácu z vačšími dátami.[10]

Q-hodnota môže byť ľubovolne reálne číslo čo robý z nášho problému regresnú úlohu, ktorá sa dá optimalizovať pomocou použitia jednoduchecj štvorcovej chybovej straty:

$$L_i(\theta_i) = E_{(s,a,r,s')\sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

Držime parametre z predchádzajúcej iterácie θ_i^- ficné pri optimalizácií i-tej stratovej funkcie.

Následne môžeme napísať gradient update rule (aktualizačné pravidlo) na základe ktorého sa nám budu aktualizovať hodnoty v našej sieti:

$$\nabla_{\theta_i} L_i(\theta_i) = E_{(s,a,r,s')} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s,a; \theta_i) \right]$$

Z ktorého už priamo vieme určiť nové váhy: $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$ kde:

- α: rýchlosť učenia
- $\nabla_{\theta_i} L_i(\theta_i)$ gradienty straty

[10]

5.1 Prostredie a agent

V kontexte implementácie algoritmu Deep Q-Learning (DQN) pre hru Azul, prostredie predstavuje samotnú hru so svojimi pravidlami a stavmi 1. Stav hry Azul je možné reprezentovať ako numerický vektor, ktorý zahŕňa informácie o aktuálnom rozložení dlaždíc vo výlohách a na stole, stave dosky hráča (vrátane steny, zásobníkov a podlahy)4.2.1.

Agent DQN interaguje s týmto prostredím s cieľom naučiť sa optimálnu stratégiu hrania hry Azul. Priestor akcií agenta zodpovedá všetkým možným ťahom, ktoré v hre existujú. V našej implementácií to predstavuje všetky validné hodnoty `Move` 4.2.8. Priestor akcií tvorí 300 možných akcií, ale len malá časť z nich je v danom stave platná. Z tohto dôvodu naša knižnica používa maskovanie neplatných akcií, aby sa agent zameral len na legálne ťahy.

Agent v každom kroku pozoruje aktuálny stav hry, vyberie akciu na základe svojej politiky (-chamtivej), vykoná akciu v prostredí a získa odmenu a nový stav. Cieľom agenta je maximalizovať kumulatívnu odmenu získanú počas hry.

5.2 Architektúra modelu

Algoritmus DQN využíva hlbokú neurónovú sieť (DNN) na aproximáciu Q-funkcie, ktorá odhaduje očakávanú budúcu odmenu pre každú dvojicu stav-akcia. Táto sieť preberá stav hry ako vstup a produkuje Q-hodnoty pre všetky možné akcie ako výstup.

Výstupná vrstva neurónovej siete má zvyčajne toľko neurónov, koľko je možných akcií v hre. Každý neurón vo výstupnej vrstve predstavuje Q-hodnotu pre danú akciu v aktuálnom stave.

5.3 Trénovanie DQN

DQN sieť je viac-vrstvová neurónová sieť, ktorá pre vstupný stav s vráti vektor akčných hodnôt $Q(s,.;\theta)$ kde θ sú parametre siete. Pre n-dimenzionálny stav obsahujúci m akcií, je neurónová sieť prechodová funkcia z R^n do R^m[11].

- **Experience Replay (Prehrávanie skúseností):** Agent ukladá svoje skúsenosti (stav, akcia, odmena, nasledujúci stav) do vyrovnávacej pamäte. Počas trénovania sa náhodne vzorkujú mini-dávky z tejto pamäte, čo pomáha rozbiť koreláciu medzi po sebe idúcimi skúsenosťami a zlepšuje generalizáciu a spolu s ňou aj stabilitu [10].
- **Target Network (Cieľová sieť):** Používa sa samostatná cieľová neurónová sieť s rovnakou architektúrou ako hlavná Q-sieť, ale jej váhy sa aktualizujú len periodicky z hlavnej siete. To stabilizuje trénovanie tým, že poskytuje konzistentnejší cieľ pre aktualizácie Q-hodnôt [10].

5.4 Technické výzvy

Implementácia algoritmu DQN pre komplexnú hru ako je Azul nám predstavila niekoľko technických výziev.

- **Rozsiahly priestor akcií:** Hra Azul má potenciálne veľmi veľký priestor akcií, z ktorých je v každom stave platná len malá časť. Efektívne preskúmanie tohto priestoru môže byť náročné. My sme si ako riešenie zvolily, že budeme zo všetkých ťahov brať v úvahu len tie platné.
- **Oneskorené odmeny:** V hre Azul sa významné odmeny získavajú až na konci kola alebo hry. To môže spôsobiť, že náš agent si vyberie ťah ktorý z dlhodobého hladiska nie je až tak dobrý.

6 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. Všetky časti tejto kapitoli sa viažu k [9]

6.1 Prostredie a agent

Prostredie predstavuje samotnú hru Azul, pričom každý stav hry je reprezentovaný, ako množina čísel opisujúcich aktuálne rozloženie dlaždíc v rámci výloh, stav dosky hráča a stavy ostatných dosiek hráčov 1. 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.

6.2 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.

6.3 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).

6.4 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 $log(0)$.

7 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.

7.1 Hodnotiace funkcie

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

7.1.1 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**

7.1.2 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
```

7.1.3 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
```

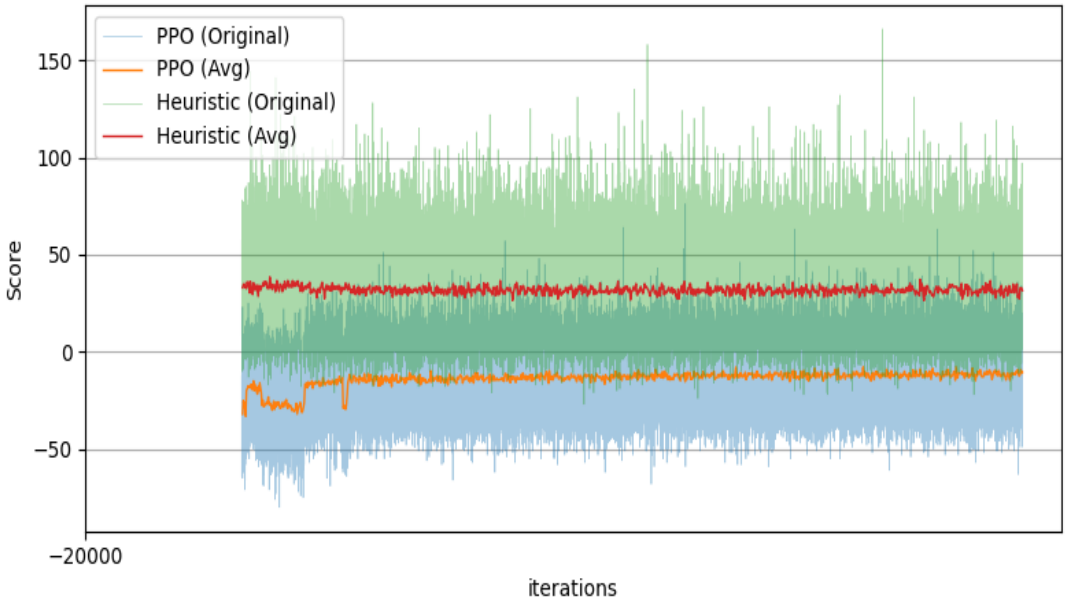
v zápätí pridáme volanie na túto funkciu do algoritmu 2 a to tak že najprv aktoalizujeme našu dosku a až následne vypočítame bonus. TODO

8 Experimenty a výsledky

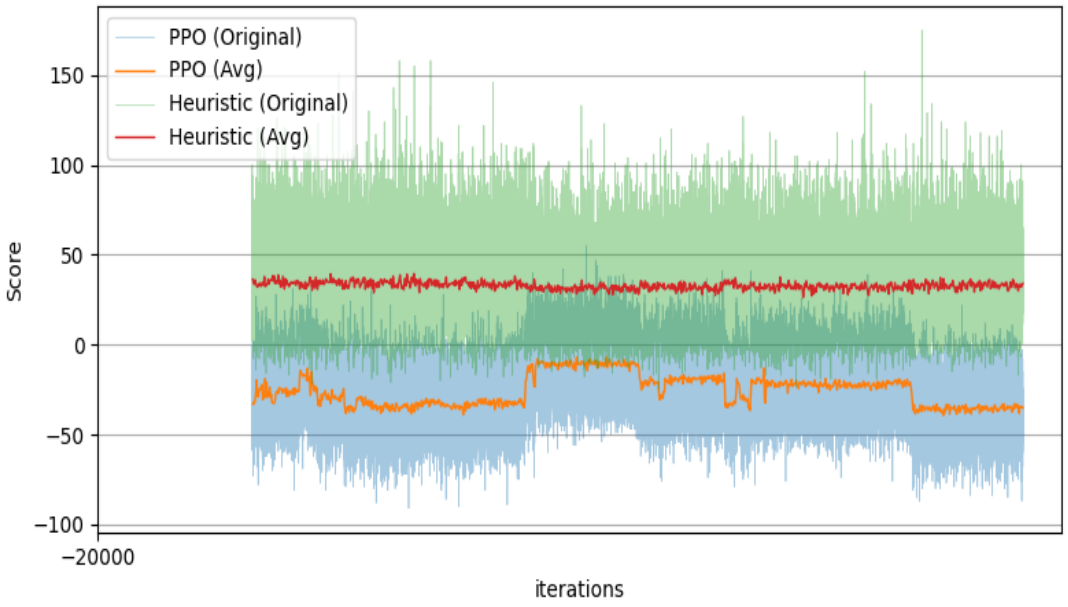
Na základe predchádzajúcich hodnotiacych algoritmov sme skúsili trénovať umelú inteligenciu pomocou DQN a PPO. Trénovali sme priamo hraním umelej inteligencie proti základnej stratégií.

8.1 Výsledky PPO

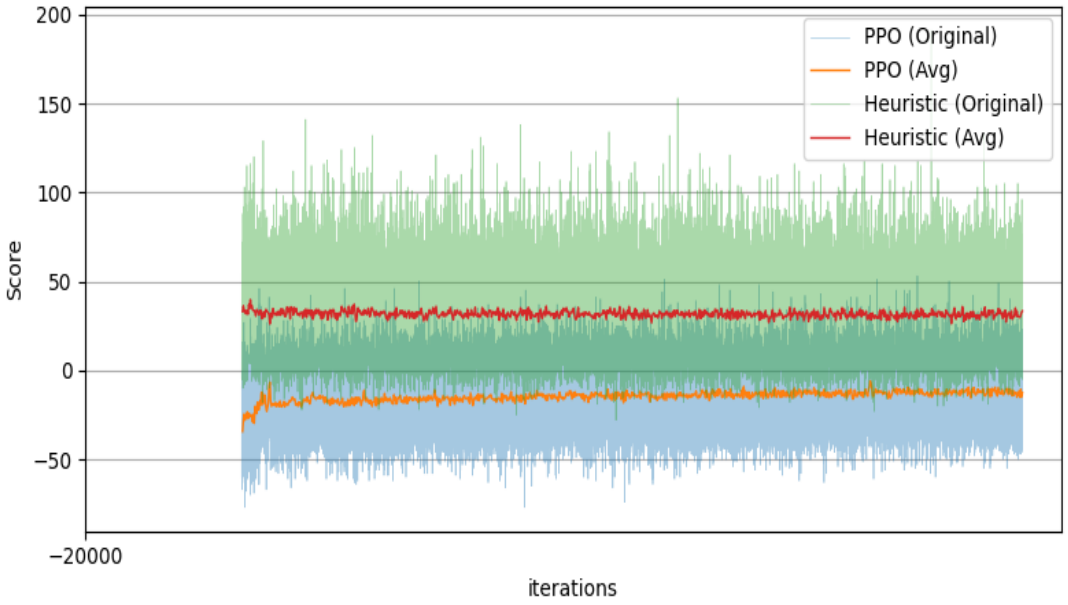
Nechali sme hrať PPO algoritmus so všetkými verziami hodnotiacej funkcie, a na základe toho sme sa dostali k výsledku ktorý sa dá pozorovať na nasledujúcich grafoch:



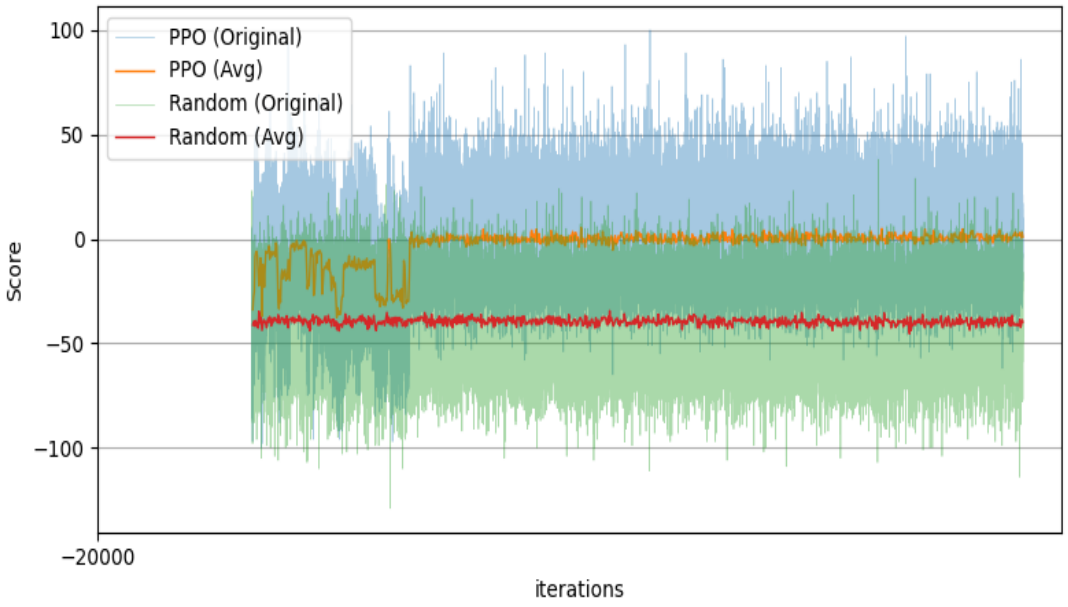
Obrázek 8.1 Výsledky hier PPO proti heuristike pre hodnotiacej funkcií 1



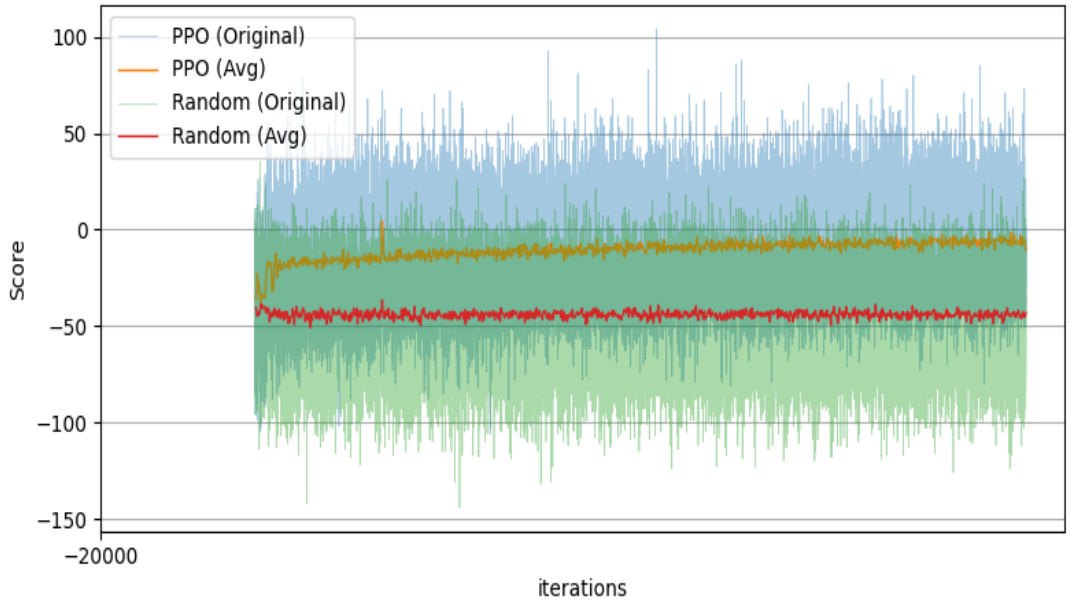
Obrázek 8.2 Výsledky hier PPO proti heuristike pre hodnotiacej funkcií 2



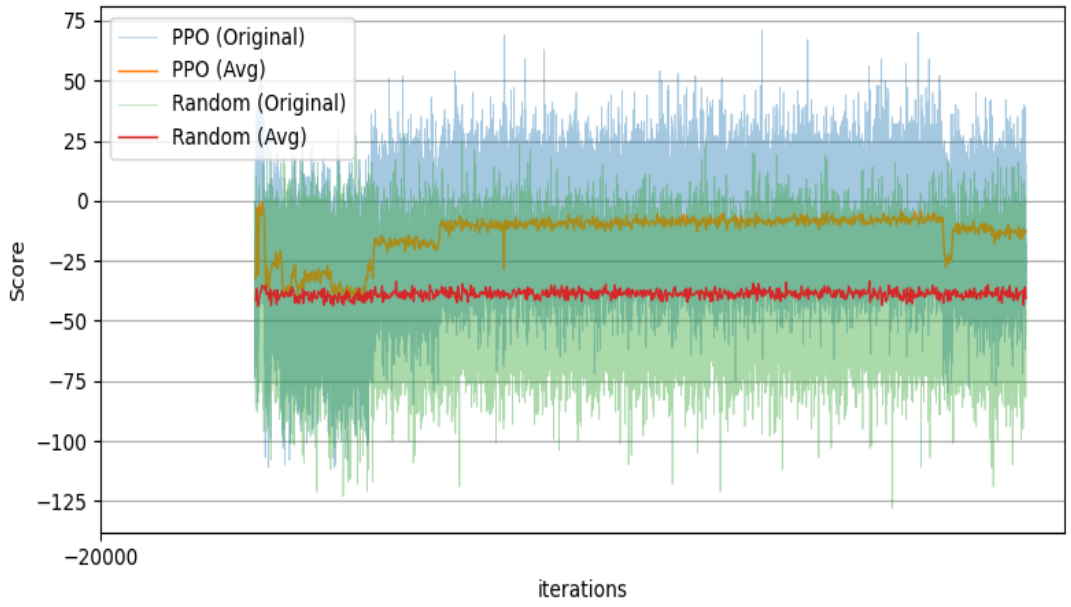
Obrázek 8.3 Výsledky hier PPO proti heuristike pre hodnotiacej funkcií 3



Obrázek 8.4 Výsledky hier PPO proti náhodnému agentovi pre hodnotiacej funkcií 1



Obrázek 8.5 Výsledky hier PPO proti náhodnému agentovi pre hodnotiacej funkcií 2



Obrázek 8.6 Výsledky hier PPO proti náhodnému agentovi pre hodnotiacej funkcií 3

Závěr

Literatura

1. *Ako hrať hru Azul [Online]*. Dostupné také z: https://www.originalnehracky.sk/navody/ako-hrat-hru-azul.
2. KIESLING, Michael. *Pravidlá [Online]*. Dostupné také z: https://www.mindok.cz/userfiles/files/pravidla/azul_pravidla_web.pdf.
3. TECHNOLOGIES, Unity. *Unity [Online]*. Dostupné také z: https://unity.com/.
4. IRENE XIANGYI CHEN CCZ, Kevin Miao. *Azul-AI*. Dostupné také z: https://github.com/irenexychen/Azul-AI.
5. HONGSANG YOO YuqingXiao97, sheejack97. *AZUL game AI-agent competition*. Dostupné také z: https://github.com/kaiyoo/AI-agent-Azul-Game-Competition.
6. 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.
7. 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.
8. 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.
9. SCHULMAN, John; WOLSKI, Filip; DHARIWAL, Prafulla; RADFORD, Alec; KLIMOV, Oleg. *Proximal Policy Optimization Algorithms*. 2017. Dostupné z arXiv: 1707.06347.
10. 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.
11. HASSELT, Hado van; GUEZ, Arthur; SILVER, David. *Deep Reinforcement Learning with Double Q-learning*. 2015. Dostupné z eprint: arXiv:1509.06461.

Seznam obrázků

8.1 Výsledky hier PPO proti heuristike pre hodnotiacej funkcií 1 . . . 28
8.2 Výsledky hier PPO proti heuristike pre hodnotiacej funkcií 2 . . . 28
8.3 Výsledky hier PPO proti heuristike pre hodnotiacej funkcií 3 . . . 29
8.4 Výsledky hier PPO proti náhodnému agentovi pre hodnotiacej funkcií 1 . . . 29
8.5 Výsledky hier PPO proti náhodnému agentovi pre hodnotiacej funkcií 2 . . . 30
8.6 Výsledky hier PPO proti náhodnému agentovi pre hodnotiacej funkcií 3 . . . 30

Seznam tabulek

Seznam použitých zkratek

A Přílohy

A.1 První příloha