VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Umělá inteligence a strojové učení

Umělá inteligence pro hru Dicewars

Martin Kostelník (xkoste12)
Ivo Meixner (xmeixn00)
Michal Dostál (xdosta51)

2. ledna 2022

1 Úvod

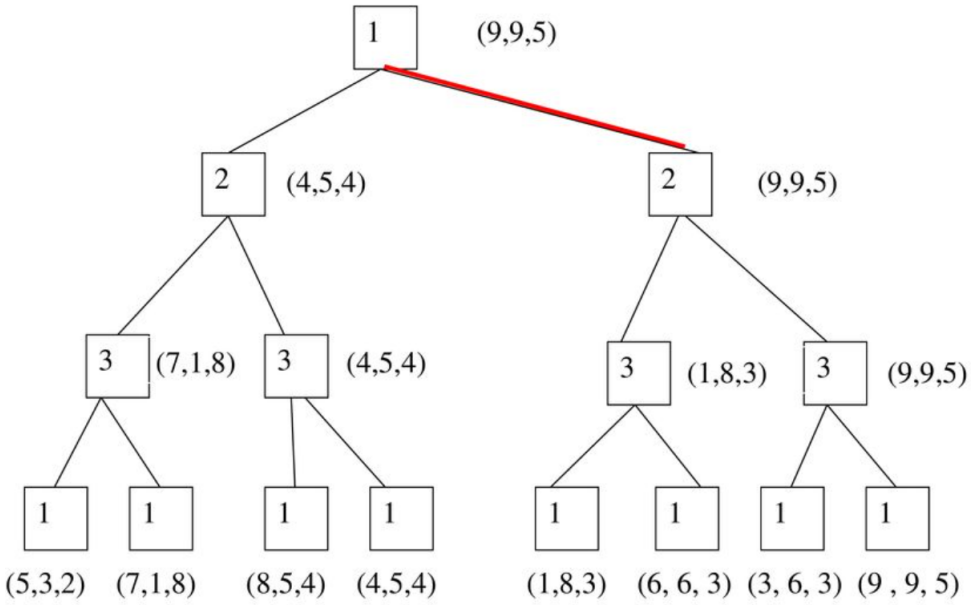
Cílem tohoto projektu je vytvořit umělou inteligenci pro hru Dicewars[1]. Dicewars je tahová hra pro dva a více hráčů s úplnou informací o prostředí (nebereme-li v potaz rezervy) a s nedeterministickým výsledkem útoků. V rámci projektu uvažujeme pouze hry čtyř hráčů. Dokumentace je rozdělena do dvou hlavních částí. První, kapitola 2, pojednává o prohledávání stavového prostoru. Kapitola 3 popisuje heuristickou funkci využitou k ohodnocování stavů hry.

2 Prohledávání stavového prostoru

Jedním z velice známých algoritmů pro prohledávání stavových prostorů ve hrách je `MiniMax`. Ten ale předpokládá deterministické akce, což zde nemáme. Rozšířením pro nedeterminismus je algoritmus `ExpectiMiniMax`. Oba algoritmy jsou pouze pro hry dvou hráčů. Naše řešení vždy očekává hru, ve které budou hráči čtyři. K prohledávání stavového prostoru her ve hrách více hráčů se dá využít algoritmus Max^n. Algoritmus Max^n je tedy zobecněním algoritmu `MiniMax` pro více než dva hráče. V této kapitole se podíváme, jak funguje a jak byl v projektu implementován.

2.1 Algoritmus Max^n

V `MiniMax` je ohodnocením stavů jedno číslo. Jeden hráč se snaží číslo maximalizovat, druhý naopak minimalizovat. Pokud přidáme hráčů více, jedno číslo pro ohodnocení stavů nám už nestačí. Použije se proto n-tice, kde n značí počet hráčů. Každý hráč se poté snaží maximalizovat své vlastní číslo. To je ilustrováno na obrázku 1. V praxi to funguje tak, že prohledáváme pouze do omezené hloubky. Všechny stavy v maximální hloubce ohodnotíme heuristickou funkcí a následně je každému rodičovskému stavu přiřazeno odpovídající ohodnocení vzhledem k tomu, který hráč je na řadě.



Obrázek 1: Ukázka algoritmu Max^n pro tři hráče. Převzato z https://slideplayer.com/slide/15266490/

[1] https://github.com/ibenes/dicewars

2.2 Implementace

Algoritmus je implementován jako rekurzivní funkce `maxn()`, která má na vstupu aktuální stav herní plochy, počet již provedených přesunů a aktuální hloubku. Funkce vrací dvojici, kde první prvek je ohodnocení aktuálního stavu a druhý prvek nejlepší nalezený tah. Funkce nejprve nalezne všechny možné tahy (útoky i přesuny, pokud přesunů nebylo provedeno maximální množství). Funkce následně prochází všechny možné tahy na aktuální úrovni kromě útoků s pravděpodobností úspěchu menší než 50 %. Tyto tahy jsou simulovány na kopii herní plochy, nad kterou je následně rekurzivně volána funkce `maxn()`. Útoky se vždy simulují jako úspěšné. Pokud je ohodnocení stavu po provedení simulace příznivější než aktuální, tah si zapamatujeme.

Stavový prostor je velice rozsáhlý, zejména pokud bereme v potaz i přesuny. Maximální hloubku prohledávání jsme omezili následovně:

- 3 – pokud zbývá více než 10 sekund
- 2 – pokud zbývá více než 2 sekundy
- 1 – jinak

Nechtěli jsme přidávat přílišnou logiku pro filtrování lidsky nepříliš logicky vypadajících tahů, abychom neomezovali model strojového učení.

Před vytvořením modelu jsme pro ohodnocení stavu využili naivní funkci, která spočítá pravděpodobnost výhry každého hráče podle podílu jeho kostek vůči celkovému počtu kostek na herní ploše. Ta je implementována jako `evaluate_state()`. Ve finálním řešení není vůbec použita.

3 Neuronová síť pro aproximaci heuristické funkce

Pro prohledávání stavů pomocí Max^n algoritmu je nutné vytvořit heuristickou funkci pro ohodnocení jednotlivých stavů. Tuto funkci jsme se rozhodli aproximovat s využitím neuronové sítě. K dosažení tohoto cíle jsou zapotřebí několik kroků, které jsou dále popsány v této sekci.

Vytvořená neuronová síť se skládá celkem z tří lineární vrstev a dvou `dropout` vrstev. První dvě lineární vrstvy používají aktivační funkci `ReLU`. Pravděpodobnost v `dropout` vrstvách byla nastavena na hodnotu 0.5. Popis architektury získaný pomocí pomocné funkce knihovny `PyTorch` je ilustrován na 2.

Při implementaci a experimentaci s modelem jsme narazili na problém ohledně aktivační funkce poslední lineární vrstvy. Nejprve jsme zvolili aktivační funkci `softmax`, aby výsledná n-tice reprezentovala pravděpodobnost výhry každého hráče. Stávalo se ale, že funkce velice často začala vracet n-tice s jedním elementem s hodnotou 1 a ostatní v nule. Pokud toto nastalo, nebyli jsme schopni vybrat lepší tah. Stejný problém nastal u aktivační funkce `log_softmax`. Nakonec jsme se rozhodli nechat poslední vrstvu čistě lineární, bez jakékoliv aktivační funkce. Výstupem je tak n-tice, kde každý prvek představuje skóre každého hráče. Tato hodnota není nijak omezena shora ani zdola a nikdy se tak nestane, že bychom nebyli schopni najít lepší tah.

3.1 Serializace stavu hry

Pro optimalizaci velikosti souborů reprezentujících průběh hry je informace o jejím vítězi a sousednosti herních políček uložena na disku pouze jednou, společně pro všechny stavy herního pole, které během hry nastaly. Serializace do tohoto formátu se provádí při generování trénovacích dat.

Na disku je každá hra reprezentována objektem typu `tuple` složeným ze tří hodnot:

- Vítěz hry zakódovaný v one-hot formátu jako pole čtyř hodnot typu `int`,
- Jednorozměrné pole popisující sousednost herních políček typu `list` s prvky typu `int`, kde 0 označuje dvojici políček, které spolu nesousedí, a 1 značí dvě sousední políčka.

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Linear-1           [-1, 3, 637, 64]          40,832
         Dropout-2           [-1, 3, 637, 64]               0
          Linear-3           [-1, 3, 637, 32]           2,080
         Dropout-4           [-1, 3, 637, 32]               0
          Linear-5            [-1, 3, 637, 4]             132
================================================================
Total params: 43,044
Trainable params: 43,044
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 4.64
Forward/backward pass size (MB): 2.86
Params size (MB): 0.16
Estimated Total Size (MB): 7.67
----------------------------------------------------------------
```

Obrázek 2: Architektura modelu neuronové sítě.

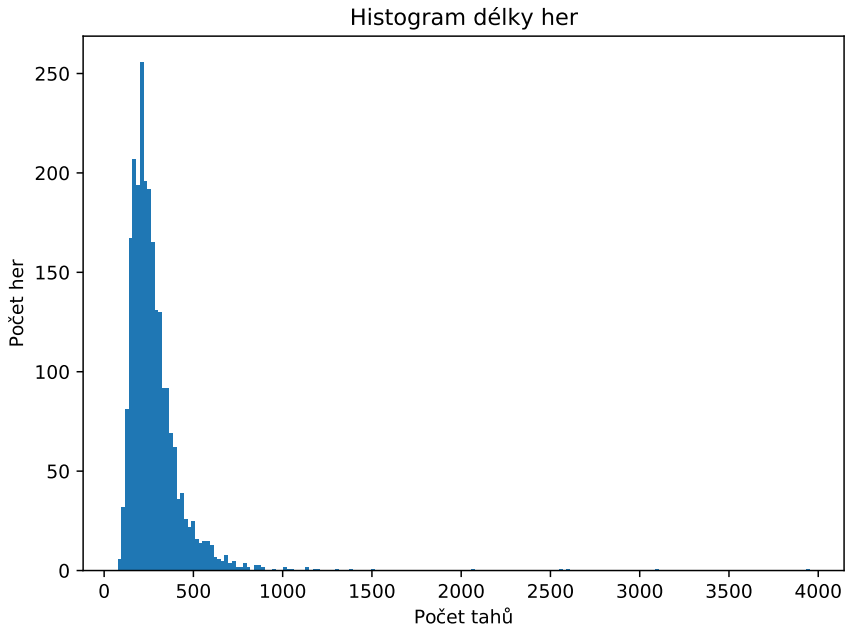
- Pole typu `list` obsahující jednotlivé stavy hry v pořadí, ve kterém nastaly, přičemž jednotlivé prvky tohoto pole mají následující formát:
  - One-hot zakódovaná informace o hráči, který je aktuálně na tahu,
  - Jednorozměrné pole popisující vlastníky jednotlivých herních políček, kde je pro každé políčko jedna hodnota typu `int` obsahující index hráče od 1 do 4,
  - Jednorozměrné pole obsahující počet kostek na jednotlivých herních políčcích uložených v typu `int` od 1 do 8,
  - Jednorozměrné pole s velikostmi největších oblastí jednotlivých hráčů v typu `int` od 0 do 34.

Jako vstup neuronové sítě se používá jednorozměrné pole složené z hodnot popisující stav herní plochy rozšířené o pole popisující sousednost herních políček. Vstupní pole má tedy celkem 637 prvků, čímž je kompletně popsán aktuální stav hry. Vítěz je v one-hot kódování použit jako výstup neuronové sítě při trénování. Funkce pro serializaci při běhu hry jsou uloženy v souboru `utils.py`.

3.2 Generování trénovacích dat

Data pro trénování byla vygenerována pomocí upravené verze hry DiceWars, která se nachází v adresáři `dicewars/supp_xkoste12/generator/`. Většina důležitých změn byla provedena v souboru `dicewars/server/game.py`, kde bylo přidáno zaznamenávání stavů hry na začáku hry a následně po každém tahu každého hráče. Tyto stavy jsou na konci hry doplněny o pole popisující sousednost herních políček a informaci o výherci dané hry zakódované v one-hot formátu. Výsledná data jsou uložena na disk do adresáře `dicewars/supp_xkoste12/data/` v populárním Pythonovém serializačním formátu *pickle*. Přesnější popis formátu serializovaných dat se nachází v sekci 3.1. Histogram délky vygenerovaných her použitých při trénování lze vidět na obrázku 3.

Pro spuštění generátoru je nutné být v jeho kořenovém adresáři (`dicewars/supp_xkoste12/generator/`), spustit skript `install.sh` a aktivovat virtuální prostředí příkazem `source path.sh`. Samotný generátor se spouští v podobě skriptu `gen.sh`. Pro zastavení tohoto skriptu stačí vytvořit v kořenovém adresáři generátoru soubor s názvem `stop`, například příkazem `touch stop`.

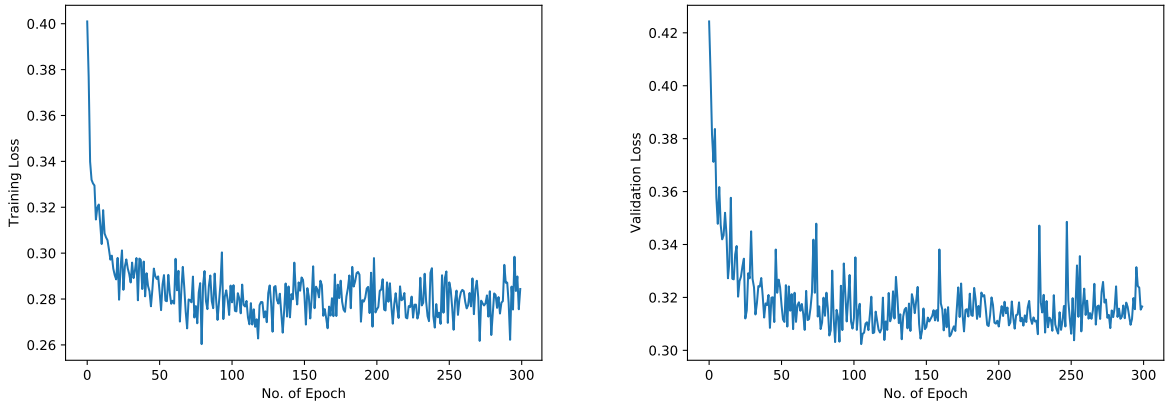


Obrázek 3: Histogram délky her.

3.3 Trénování modelu

Pro natrénování modelu bylo vygenerováno celkem 2250 her, ze kterých jsme získali 638255 různých stavů herní plochy. Tato data byla následně rozdělena na trénovací a validační datovou sadu. Trénovací sada obsahuje 80 % z původních dat. Validační sata zbylých 20 %. Bylo nutné vytvořit třídu `GameDataSet`, která implementuje funkce `__len__()` a `__getitem__()`. Tyto funkce poté používá třída `torch.DataLoader`, která data zamíchá a rozdělí do skupin o velikosti 256.

Jako optimalizační algoritmus jsme zvolili algoritmus `Adam` s `learning rate` s hodnotou 0.01. Pro `loss` funkci jsme zvolili `CrossEntropy`. Počet epoch trénování jsme nastavili na 1000 s tím, že nový model se uloží pouze tehdy, pokud validační loss dosáhne nového minima. Nejlepší model jsme získali v epoše číslo 330. Průběh loss funkce pro trénovací i validační data lze vidět na obrázku 4. Veškeré trénování probíhalo na stolním počítači s grafickou kartou nVidia RTX 3070.



Obrázek 4: Průběh trénovací a validační loss funkce během trénování.

3.4 Použití modelu

V hlavním souboru s implementací umělé inteligence je vytvořena nová heuristická funkce `evaluate_state_nn()`, která dostane aktuální stav herní plochy serializuje jej jak je popsáno v 3.1 a vrátí její ohodnocení. V inicializační funkci třídy je třeba model nejprve načíst a vyhodnotit. Načtení vyžaduje přístup k třídě popisující neuronovou síť. Zkopírovali jsme potřebný kód do souboru `utils.py` abychom předešli problémům s importem ze složky `supp-xkoste12`. Další problémy se projevili, když jsme se snažili model spouštět na grafické kartě. Nejjednodušším způsobem bylo toto zakázat a používat pouze procesor.

4 Obsah archivu

Odevzdaný archiv obsahuje následující:

- `xkoste12.pdf` - Dokumentace projektu.
- `supp-xkoste12/data` – Adresář obsahující vzorek dat použitých ke trénování.
- `supp-xkoste12/generator` – Adresář obsahující změněný kód použitý při generování trénovacích dat.
- `supp-xkoste12/model.py` – Skript použitý pro natrénování modelu neuronové sítě.
- `xkoste12/model.pth` – Natrénovaný model neuronové sítě pro aproximaci heuristické funkce.
- `xkoste12/xkoste12.py` – Skript obsahující definici třídy `AI`.
- `xkoste12/utils.py` – Skript obsahující architekturu neuronové sítě a funkce pro serializaci stavu hry.

5 Závěr

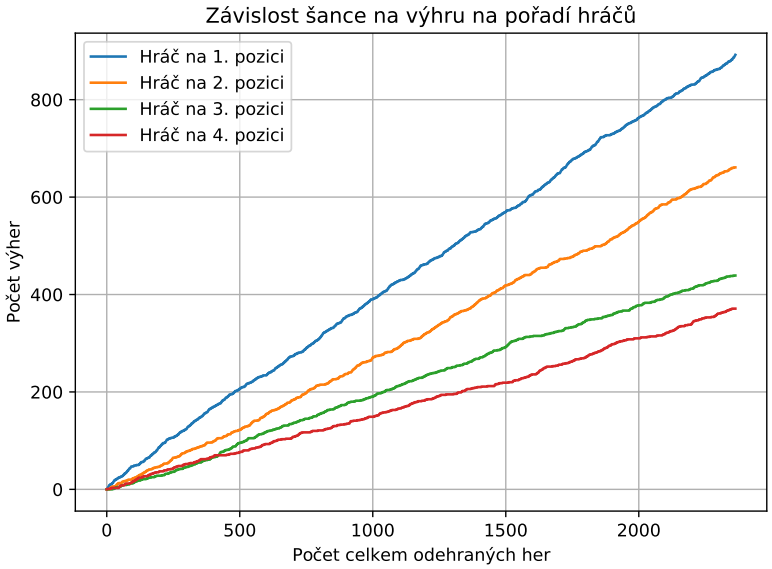
V projektu byla implementována umělá inteligence pro hru Dicewars využívající algoritmus Max^n s měnící se maximální hloubkou dle zbývajícího času. Dále byla vytvořena a natrénována neuronová síť pro ohodnocení jednotlivých stavů. Vytvoření modelu zahrnovalo úpravu původní verze hry pro vygenerování trénovacích dat, trénování modelu a jeho integraci do prohledávání stavového prostoru.

Výsledná umělá inteligence dosahuje výsledků podobných jako implementace v modulu `dt`, avšak na implementace v modulu `kb` nestačí. Výsledek turnaje o sto hrách lze vidět na obrázku 5.

```
kb.stei_adt       63.89 % winrate [ 23 / 36  ]
kb.stei_at        63.89 % winrate [ 23 / 36  ]
kb.sdc_pre_at     37.50 % winrate [ 15 / 40  ]
kb.stei_dt        31.25 % winrate [ 10 / 32  ]
dt.stei           22.50 % winrate [ 9  / 40  ]
kb.sdc_post_dt    11.11 % winrate [ 4  / 36  ]
xkoste12.xkoste12 11.00 % winrate [ 11 / 100 ]
kb.sdc_post_at    5.56  % winrate [ 2  / 36  ]
dt.wpm_c          4.55  % winrate [ 2  / 44  ]
```

Obrázek 5: Výsledek turnaje o sto hrách.

Při testování jsme si všimli zajímavé věci. Pořadí hráčů značně ovlivňuje jejich šance na výhru. Hráč, který začíná, je na tom vždy lépe než druhý, ten lépe než třetí a tak dále. To lze vidět na obrázku 6. Spuštění velkého množství her by mělo tuto skutečnost vyvážit, ale člověk musí být opatrný při spouštění malého množství při testování funkčnosti.



Obrázek 6: Graf zobrazující závislost šance na výhru na pořadí hráčů.