Vysoké učení technické v Brně
Fakulta informačních technologií

Umělá inteligence a strojové učení SUI

Dicewars 2019/2020

Kohout Petr (xkohou14)
Kohout Pavel (xkohou15)
Tesařová Alena (xtesar36)

5. ledna 2020

1 Úvod

Naším úkolem bylo vytvořit umělou inteligenci (AI) pro hru Dicewars. Postupovali jsem nejprve tak, že jsme se pokoušeli modifikovat dostupné AI a zkoumali jejich výsledky. Prvním pokusem byla modifikace dt.wpm_c (sekce 2.1), další pokusem bylo implementování algoritmu ExpectiMiniMax (sekce 2.2) a jako poslední řešení bylo zapojení neuronové sítě na základě analýzy vstupních dat v nástroji RapidMiner.

2 Řešení

2.1 První AI

První navržená AI vychází z implementace dt.wpm_c, kde jsme zkoušeli připočítávat hodnotu možného navýšení skóre. Pro případ, kdy by se k největšímu území připojilo území o velikosti pět, vedl by tah k navýšení skóre ne o jedna, ale o pět. Experimentálně bylo zjištěno, že AI příliš riskuje, jelikož získání území najednou získalo větší váhu než udržení území. Po opravení hodnoty získaného území na logaritmus této hodnoty se výsledky zlepšily jen minimálně a AI se stále umisťovala na posledních místech v turnajích. Řešení proto nebylo ideální.

2.2 ExpectiMiniMax

Tato inteligence byla implementována pomocí abstraktní datové struktury strom. Strom obsahuje tři druhy uzlů – MAX, MIN a CHANCE. Oproti běžnému minimaxu se liší právě v přidání CHANCE uzlů, které zachycují náhodnost, která ve hře nastává. Uzel MAX reprezentuje tah hráče a MIN reprezentuje tah protihráče. Typy uzlů se systematicky střídají v posloupnosti MAX, CHANCE, MIN, CHANCE a MAX...

Velice problematické pro tento algoritmus je fakt, že jednotliví hráči mohou zahrát více než jeden tah za kolo. Tento fakt nebyl ve výpočtu zohledněn a bylo tedy vždy uvažováno o pravidelném střídání tahů hráčů.

Proces výpočtu nejlepšího tahu začíná v uzlu MAX, který tvoří kořen stromu. Současně mu jsou předány informace o bitevním poli a hodnota odpovídající zanoření. Uzel zpracuje data a začne provádět svoje vyhodnocení. Vypočtou se všechny možné útoky, které jsou iterovány a předány další vrstvě vytvořením instance zanořeného uzlu. Poté je zaznamenána hodnota tohoto uzlu. Po ukončení iterace je vybráno maximum, které je nastaveno jako hodnota uzlu. Uzel MIN provede výpočet možných útoků, jejich iteraci a následný výběr minima. Uzel CHANCE provádí vyhodnocování vzhledem k pravděpodobnosti proveditelnosti útoku. Uzel vypočte pravděpodobnost úspěšného a neúspěšného provedení útoku. Poté je simulována nová situace na mapě ziskem pole hráčem a přiřazením počtem kostek cílovému poli o jedna menší než je hodnota pole, ze kterého se provádí útok. Pro tuto situaci je provedeno zanoření do další vrstvy, čímž se vypočte i hodnota uzlu. Poté je vrácena situace na mapě a je provedena simulace pro neúspěšný útok nastavením pole, ze kterého se prováděl útok, na hodnotu jedna. Poté je situace opět vrácena do původní situace. Naměřené hodnoty jsou vynásobeny pravděpodobností daného útoku a je z nich vybráno maximu případně minimum. V případě dosáhnutí maximální hloubky je proveden výpočet pravděpodobnosti útoků a hodnoty útoků v nejbližší hloubce a je z nich vybráno maximum potažmo minimum. Jako další tah je vybrán uzel s maximální hodnotou následníka kořenového uzlu.

Byla provedena analýza výsledků našeho řešení a pokusy o zlepšení. Metoda dosáhla skvělých výsledků proti slabým inteligencím především `xlogin00`, ale při porovnání s ostatními soupeři nedosáhla úspěšných výsledků, a proto se přistoupilo k implementaci zpětnovazebného učení.

2.3 Zpětnovazební učení (Reinforcement learning)

Jako další řešení jsme zvolili zpětnovazebné učení. Takový systém je nejdříve potřeba natrénovat z dat poskytnutých expertem (v našem případě z dat spuštění hry dicewars). Natrénovaná neuronová síť nám

pak umožní soupeřit proti dalším AI. Prvním problémem je, jakým způsobem neuronovou síť sestavit – kolik bude mít vrstev nebo jakou bude používat aktivační funkci. Druhý problém spočívá v tom, jaké bude mít neuronová síť vstupy a výstupy tak, aby byla nejmenší chybovost. Proto byla potřeba použít další nástroje, které by nám zjednodušily výběr atributů trénovacích dat.

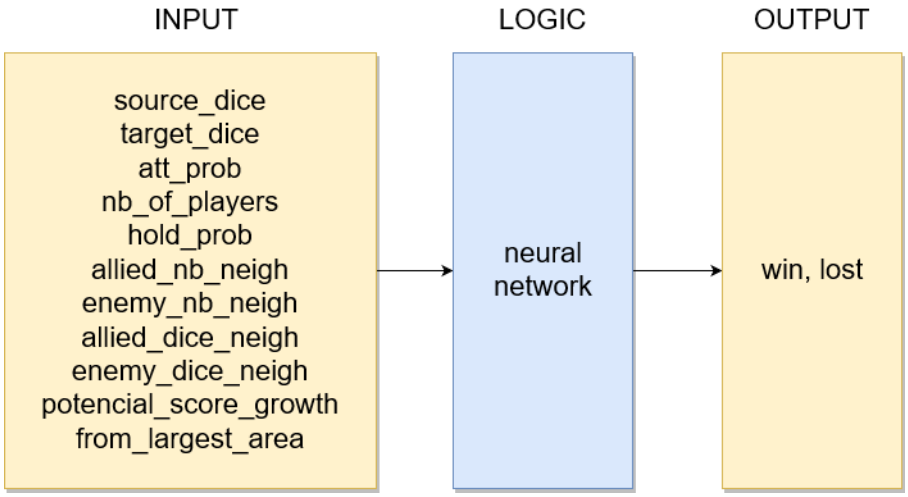
2.3.1 Příprava dat

Na začátku procesu realizace metody reinforcement learning bylo nutné získat dostatečné množství dat pro učení neuronové sítě. Data jsme získali pomocí hraní her a získáváním informací o tazích vedoucích k úspěšnému zakončení hry. Jako agenta k hraní her jsme použili existující implementaci `RAND`[1]. Byla rozšířena o sběr tahů. Vytváření ucelených záznamů o tazích bylo prováděno pomocí námi vytvořeného skriptu *turns_collector.py*, který obsahuje třídy `Turn`, která reprezentuje tah na mapě a třídy `TurnCollector` a `TurnLoader` provádějící ukládání a případné načítání dat do souboru. Třída `Turn` umožňuje ukládat informace o tazích provedených během hry a dodatečně je označuje značkami, podle výsledku celé hry či následných tazích. Poskytuje podpůrné metody pro jejich uložení v potřebném formátu do souboru a jejich následné načtení ve skriptu *trainer.py*.

Data získaná z her byla podrobena analýze evolučním algoritmem pro výběr vhodných atributů v nástroji **RapidMiner**. Poté byly vhodné atributy vybrány a bylo na nich provedeno učení neuronové sítě, jejichž výsledky jsme následně vyhodnotili. Finální schéma zapojení neuronové sítě lze vidět na obrázku 2.3.1. Samotná síť je tvořena vstupní vrstvou, jednou skrytou vrstvou s deseti neurony a výstupní vrstvou obsahující 2 neurony s aktivační funkcí *softmax*. Trénování probíhalo v dvaceti epochách. Iterativně jsme zvolili následující výběr atributů:

- source_dice – počet kostek pole, ze kterého je veden útok,
- target_dice – počet kostek pole, na který je veden útok,
- att_prob – pravděpodobnost úspěšného útoku,
- nb_of_players – počet hráčů ve hře,
- hold_prob – pravděpodobnost udržení pole do dalšího tahu,
- allied_nb_neigh – počet sousedních polí pod vlastnictvím hráče,
- enemy_nb_neigh – počet sousedních polí pod vlastnictvím protihráčů,
- allied_dice_neigh – součet kostek sousedních polí pod vlastnictvím hráče,
- enemy_dice_neigh – součet kostek sousedních polí pod vlastnictvím protihráčů,
- from_largest_area – informace zda se jedná o útok z největšího vlastněného území,
- nb_of_player_dices – počet kostek vlastněných hráčem.

[1] https://github.com/ibenes/dicewars/tree/master/dicewars/ai/dt/rand.py



Obrázek 1: Schéma zapojení neuronové sítě.

Trénování dat a přiložené testovací scripty

Trénování modelu probíhá tak, že se spustí souboj umělé inteligence (xkohou15.rand) s ostatními předem implementovanými AI po jednom souboji (skript *play-with-all-ais.sh xkohou15.rand*). Výsledky souboje se zapíší do dočasné složky temporary_turns. Po každé hře je spuštěn skript heap_swapper.py, který v případě, že byl souboj výherní, přepíše a přiřadí záznamům z temporary_turns příznak WIN (v případě prohry se vloží příznak FAIL) a uloží je do souboru *heap_file.training*, který bude sloužit jako výsledný trénovací soubor pro neuronovou síť. Celkový obsah složky *supplementary*:

- *supplementary/heap_swapper.py* – záznamům dává příznak výhra/prohra a zapisuje je z dočasného souboru do souboru pro sběr *heap_file.training*.
- *supplementary/NeuralNetworkBuilder.py* – třída neuronové sítě (NS), která obsahuje metody pro vytvoření NS (sestavení v podkapitole 2.3), trénování NS, vytvoření a uložení získaného modelu
- *supplementary/play-with-all-ais.py* – spouští bitvy po jedné s každou AI a po každé bitvě k nasbíraným datům přiřadí příznak, zda byly ve vítězné hře; pokud ano tak win, jinak fail (pomocí skriptu *heap_swapper.py*)
- *trainer.py* – vezme data z heap_file.training, která filtruje pouze pro ta s příznakem výhry a dá je neuronové síti k naučení
- *supplementary/turns_collector.py* – třída pro zpracování vstupů neuronové sítě, které se získávají pro každý tah
- *test_ai.sh* – skript, který spouští souboje (duely) umělých inteligencí tak, že
- *supplementary/collect_train_test_ai.sh* – skript, který spouští námi vytvořené skripty (pouze pro ulehčení práce testování)
- *default_model_copy.h5* – kopie odevzdávaného modelu

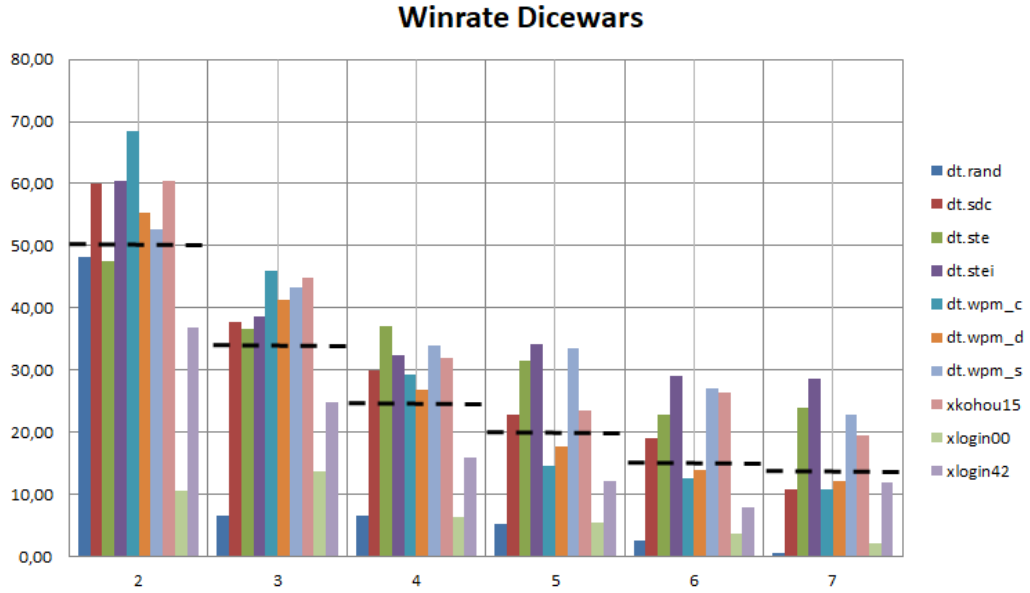
Postup vytvoření modelu

1. *./play-with-all-ais.sh xkohou15.rand* – spouštění ve složce *supplementary*, která se nachází v rootu projektu, vytváří heap_file.training ve složce naší AI
2. python3 *trainer.py* – vytvoření souboru default_model.h5, který je potřeba následně přesunout do složky, kde se nachází naše AI

Pro správné spuštění skriptů je potřeba mít nainstalované knihovny *tensorflow* a *keras*.

2.3.2 Naměřené výsledky

Měřili jsme výhru v soubojích se dvěma až sedmi hráči (vždy na 100 her). Naše AI **xkohou15** se umístila většinou na třetím nebo čtvrtém místě, jak jde vidět v grafu 2.3.2.



Obrázek 2: Dosažené výsledky v soubojích se třemi až sedmi hráči.

3 Závěr

Během práce byla provedena analýza hry a výhod jednotlivých umělých inteligencí. Byla provedena modifikace pod snahou zlepšení dosavadních výsledků již existujících řešení. Zlepšení nebylo dosaženo, a proto se provedla implementace algoritmu ExpectiMiniMax. Tato metoda opět nevykázala dobré výsledky v porovnání s ostatními soupeři, a proto bylo od jejího zlepšování upuštěno. Nakonec jsme zvolili metodu reinforcement learning, kde bylo nejprve potřebovat systém natrénovat a vytvořit model pro naši umělou inteligenci. K výběru vstupů nám sloužil dolovací nástroj RapidMiner, díky kterému se nám podařilo dosáhnout přesnosti modelu až na 80 %. Výsledky jsou nyní uspokojivé.

Reference

[1] *Artificial Intelligence for a Board Game. Brno, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing.Karel Beneš*