VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

SUI – projekt

AI pro válku kostek

Jakub Frejlach (xfrejl00)
Tomáš Sasák (xsasak01)
Martin Krajči (xkrajc21)
Tomáš Venkrbec (xvenkr01)

30. prosince 2020

1 Úvod

Cílem tohoto projektu bylo vytvořit umělou inteligenci pro hru Dicewars. Vytvořená umělá inteligence obsahuje prvky strojového učení a prohledává stavový prostor herního pole. V této dokumentaci jsou popsány jak dosažené výsledky, tak také veškeré použité postupy a významné změny, ke kterým došlo v průběhu vývoje. Rovněž je zde také věnována sekce trénování a vyhodnocování implementovaných umělých inteligencí.

2 Zvolený způsob řešení

Při výběru způsobu, kterým bude umělá inteligence realizována, hrálo velkou roli zejména to, že v posledních letech dochází díky rostoucímu výkonu počítačů k rozšíření posilovaného učení a objevují se různé zajímavé aplikace této metody například v některých známých počítačových hrách. Zde vznikl nápad použít na řešení tohoto projektu také posilované učení, a vznikl pracovní název pro tuto umělou inteligenci - **AlphaDice**.

2.1 Q-learning

Prvním pokusem bylo použití algoritmu **Q-learning**. Jeho výhodou je vcelku jednoduchá implementace, dále si lze prohlédnout hodnoty uložené v Q-table pro různé vybrané stavy, aby se dalo evaluovat, jestli se trénování vyvíjí zhruba očekávaným směrem. Velkou nevýhodou je ovšem to, že stavový prostor samotné hry Dicewars je obrovský. Pole mohou patřit různým hráčům, mít různé počty kostek a i celé samotné rozložení hracího pole je náhodné. Tento problém byl vyřešen zejména tím, že stavy charakterizují jednotlivé tahy, namísto stavu celé herní plochy. Postupně jsme během vyvíjení algoritmu Q-learning vymysleli 6 různých stavů, které charakterizují každý možný tah.

- Šance na získání pole
- Šance na udržení pole
- Zvětšení velikosti regionu při úspěšném útoku
- Snížení velikosti regionu protivníka při úspěšném útoku
- Počet sousedících políček cílového pole
- O kolik polí by se snížila velikost regionu při neúspěchu a následném obsazení zdrojového pole

Dalším problémem je to, že každý ze stavů výše má příliš mnoho hodnot, kterých může dosahovat. Pokud se však na některé stavy podíváme a zamyslíme se, jako například na dva tahy, které se liší například pouze v šanci na získání pole, kde první tah má šanci na úspěch 0.99 a druhý stav má šanci na úspěch 1.0, můžeme si uvědomit, že strategie umělé inteligence se zde příliš lišit asi nebude, přesto se jedná o dva různé stavy, které je třeba v Q-table uchovávat a natrénovat je, aby měly správnou Q-value. Abychom tedy stavový prostor ještě výrazněji zmenšili, diskretizujeme ho. Každý stav může tedy po diskretizaci dosahovat 4 různých hodnot, s výjimkou stavu pro počet sousedících polí cílového pole, který dosahuje 3 různých hodnot. Jelikož každý stav je uchováván v Q-table pro akci ”útok”a ”obrana”, je celkový počet možných stavů v Q-table roven 6144. V praxi během trénování výsledného modelu na více než 30000 hrách bylo ovšem umělou inteligencí objeveno necelých 3900 stavů.

Trénování bylo ovšem komplikované tím, jak těžké bylo získat hodnoty pro Bellmanovu rovnici, aby bylo vůbec možné počítat nové hodnoty stavů v Q-table. Problémem bylo zejména zjistit, jaký vliv měl tah na stav v příštím kole, abychom toto mohli v rovnici využít. Z tohoto důvodu jsme naší umělé inteligenci umožnili udělat si kopii herního pole a odsimulovat si tahy ostatních hráčů pomocí vlastní Q-table, abychom dostali stav desky, který může být podobný tomu, který by skutečně byl v příštím kole. Na základě stavu desky, zvoleného tahu a akce a stavu desky po hře ostatních hráčů počítáme odměny, které zvolenému tahu v Q-table přidáváme. Jak je zřejmé, toto řešení není ani zdaleka ideální a proto největší dosažený poměr vyhraných her touto umělou inteligencí byl zhruba 20% jako klouzavý průměr z 2000 her.

2.2 Q-learning s klasifikátorem šance na výhru

Jako největší nedostatek řešení pomocí Q-table jsme identifikovali právě aproximaci Q-value nového stavu. Proto jsme tedy implementovali neuronovou síť, která je trénována na různých statistikách aktuální hry, s labely značícími, zda hra vedla k výhře či ne. Tento klasifikátor tedy odhaduje, jaká je aktuální šance na výhru v libovolné fázi hry. Jedná se o jednoduchou vícevrstvou neuronovou síť s aktivační funkcí sigmoid ve výstupní vrstvě. Pro aktuální stav desky nám tedy klasifikátor vrací hodnotu z intervalu $(0; 1)$, značící šanci na výhru. Q-value nového stavu, do kterého jsme se dostali vykonáním tahu, počítáme jako:

$$\textbf{approximated\_next\_turn\_qvalue} = \textbf{Qvalue} + 0.5 * \textbf{Qvalue} * (\textbf{probability\_before} - \textbf{probability\_after}). \quad (1)$$

Kde **probability_before** je šance na výhru před vykonáním tahu a **probability_after** je šance na výhru po vykonání tahu. Nová Q-value je tedy v intervalu $< Qvalue * 0.5; Qvalue * 1.5 >$ na základě toho, jak moc byla pozitivně či negativně ovlivněna šance na výhru.

2.3 Deep Q-learning s klasifikátorem šance na výhru

V rámci řešení problému s velikostí Q-table jsme zkusili implementovat také Deep Q-learning, které Q-table nahrazuje neuronovou sítí, která hodnoty Q-value aproximuje. Tuto síť se nám ale nepodařilo příliš úspěšně natrénovat. Zkoušeli jsme, aby neuronová síť vybírala takový tah, který nejvíce maximalizuje získanou odměnu, či nejvíce zlepší šanci na výhru, kterou určuje dříve zmíněný klasifikátor, ovšem žádné řešení svými výkony nepředčilo umělou inteligenci využívající klasický Q-learning s klasifikátorem. Nejlepší výkon dosažený touto metodou byl zhruba 25% jako klouzavý průměr z 2000 her, při určování, který tah nejvíce zlepší šanci na výhru.

3 Trénování a vyhodnocování

K trénování jednotlivých iterací umělé inteligence jsme si vytvořili vlastní skript, který umělou inteligenci a neuronové sítě trénuje, přidává odměny tahům do Q-table na základě výsledku hry, ukládá natrénované umělé inteligence, ukládá výsledky her a tvoří grafy poměru vyhraných her.

Trénování probíhalo proti všem dostupným umělým inteligencím ve hře, a zároveň také proti některým starším úspěšným iteracím naší umělé inteligence. Protivníci jsou vybíráni náhodně a každá zvolená kombinace protivníků a naší AI hraje 4 hry, kde se postupně mění pořadí hráčů.

Klasifikátor byl trénován na 22 různých statistikách vypočítaných z aktuálního stavu hrací desky, kde část z nich se zabývá statistikami našeho AI, další z nich statistikami celé hry a další z nich statistikami jednotlivých oponentů. Celý seznam všech 22 pozorovaných statistik je k nalezení ve třídě `GameStats` v souboru `utils.py`. Neuronová síť pro Deep Q-learning je trénována na stejných stavech jako klasický Q-learning, ovšem používají se nediskretizovaná data. Odevzdaný model klasifikátoru byl trénován na datové sadě obsahující přes 900.000 vzorků z vítězných her a 800.000 vzorků z prohraných her a nejvyšší dosažená validační přesnost byla 92%.

K usnadnění vyhodnocování vytváříme několik typů grafů, zajímají nás hodnoty loss funkce a přesnost neuronových sítí, ale především poměr výher ku celkovému počtu her naší umělé inteligence. Aby naše vyhodnocování nebylo komplikováno výkyvy tohoto poměru, sledujeme klouzavé průměry posledních 200 a 2000 her. Dalším subjektem vyhodnocování byla samotná Q-table, kde šlo vypozorovat určité nedostatky, když jsme studovali velikosti Q-value různých vybraných stavů. Celý vývoj byl spíše řadou experimentů, kde byla stanovena baseline a byly vymyšleny a spuštěny experimenty, kde došlo ke změně například některého z parametrů, přidání stavů, změnou výpočtu odměn Bellmanovy rovnice, apod. Úspěšné experimenty byly zakomponovány do nové baseline a vývoj pokračoval dalšími experimenty.

4 Dosažené výsledky

Použitím základního algoritmu Q-learning, tedy bez neuronové sítě pro aproximaci ohodnocení jednotlivých stavů, společně s klasifikátorem šance na výhru, se nám podařilo dosáhnout během trénování až 60% poměru vyhraných her z 200 her, ovšem přesnější je spíše **klouzavý průměr z 2000 her**, ve kterém jsme dosáhli až **43% poměru výher**. Tento výsledek sloužil spíše k trénovacím účelům, než k validačním, jelikož byl ovlivněn tím, že naše AI figurovalo ve všech hrách a hrálo proti jiným umělým inteligencím, než které budou použity při hodnocení.

Z tohoto důvodu byl ještě spuštěný turnaj proti umělým inteligencím použitým při hodnocení, kde se naší umělé inteligenci podařilo vyhrát **279** z celkového počtu **536** her, dosáhla tedy poměru **52.05%** vyhraných her ze všech her. Všechny umělé inteligence v tomto turnaji hrály až na odchylku jediné série her stejný počet her. Naše umělá inteligence v počtu výher překonala všechny soupeřící umělé inteligence.

5 Odevzdané soubory

- `snapshots/final/winrate_all.csv` - údaje o výsledcích her z průběhu trénování
- `snapshots/final/winrate_all.png` - graf o průměrném poměru výher z posledních 200 her
- `snapshots/final/winrate_all_2000.png` - graf o průměrném poměru výher z posledních 2000 her
- `snapshots/final/snapshot.pickle` - natrénovaná Q-table
- `classifier.py` - Torch model klasifikátoru
- `dqn.py` - Torch model Deep Q-learning neuronové sítě
- `create_datasets.py` - Skript k rozdělení získaných dat na trénovací a validační dataset
- `qtable.py` - implementace třídy pro manipulaci s Q-table
- `train.py` - skript pro trénování umělé inteligence
- `utils.py` - soubor se pomocnými funkcemi pro AI
- `xfrejl00.py` - základní soubor pro AI