Umělá inteligence a strojové učení
AI pro hru Dicewars

Ladislav Ondris (xondri07)
Alexander Polok (xpolok03)
Michal Rozsíval (xrozsi06)

2. ledna 2022

1 Úvod

Práce se zabývá tvorbou umělé inteligence pro hru Dicewars, což je strategická hra s otevřenou informací a prvkem náhody. Hráči na *tahu* se postupně střídají, přičemž každý může provést *sekvenci akcí*, která může být také prázdná. V tomto se tato hra liší od her typu piškvorek, kde každý hráč provádí vždy jeden tah.

Nejprve jsme se pokusili o posilované učení, kde jsme se snažili naučit neuronovou síť predikovat následující akci, což je popsáno v následující sekci společně s možnými důvody, proč tento přístup selhal.

Kvůli neúspěchu posilovaného učení jsme se uchýlili k prohledávání stavového prostoru. Dále jsme natrénovali neuronovou síť, která vrací ohodnocení hracího pole z pohledu zvoleného hráče, tudíž funguje jako heuristická funkce.

2 Pokus o posilované učení

Cílem bylo natrénovat neuronovou síť, která by dle stavu celé hry na vstupu říkala jaké akce provést. Pro tento účel bylo hrací pole reprezentováno sadou 2D matic. Tedy tyto matice reprezentovaly vztah každého políčka s každým. Jako příznaky byly zvoleny:

- hodnota 1 v řádcích odpovídajících aktuálnímu hráči,
- pravděpodobnost získání pole,
- pravěpodobnost obrany pole,
- velikost spojitého pole po dané akci,
- vzdálenost polí,
- rozdíl počtu kostek.

Jako model byla zvolená dopředná síť s několika lineárními vsrtvami. Vstup sítě byl (Batch size, $F \cdot A \cdot A$), kde F značí počet příznaků a A značí počet polí ve hře. Výstupem sítě pak byl (Batch size, $O \cdot A \cdot A$), kde O značí počet akcí mezi danými políčky (tedy hodnota 2 - útok a přesun).

Snažili jsme se model trénovat algoritmem QActorCritic, nicméně se model nenaučil nic rozumného. Problému se vyskytlo více, nicméně výstup hodnot pro všechny akce konvergovaly buď k hodnotě 0 nebo k hodnotě 1, což bylo nejspíše zapříčiněno způsobem, jakým algoritmus funguje.

Selhání mohlo být způsobeno také dalšími příčinami. Přesná přičina je však nejasná. Mezi nejpravděpodobnojší teorie patří následující:

- Vstupní reprezentace mapy je nedostatečná, nepřesná, nebo příliš komplexní.
- Hra je pro posilované učení příliš komplexní a agent není schopný prostředí porozumnět.
- Posilované učení vyžaduje mnohem větší množství hyperparametrů, které je velice obtížné zvolit. Při nefungujícím trénování se jen těžko ladí.
- Samotný algoritmus trénování je špatně.
- Příliš velké množství akcí na výstupu modelu.
- Trénování po příliš krátkou dobu.

3 Prohledávání stavového prostoru

K úspěšnému hraní lze přistupovat způsobem prozkoumování různých tahů a jejich vliv na hru. Pro to je potřeba zjišťovat možné tahy a taktéž neuvažovat tahy, které není třeba uvažovat, což je popsáno v sekci 3.1. Dále je potřeba tahy vyhodnocovat, k čemuž slouží heuristická funkce, viz sekce 3.2. Při hraní je taktéž důležité uvažovat tahy oponentů. K tomu slouží algoritmus Best Reply Search, viz sekce 3.3.

3.1 Prořezávání tahů a sekvencí

Stavový prostor je příliš velký na to, aby bylo možné ho celý prozkoumat. Kvůli čemuž je prováděno prořezávání jak do hloubky, tak do šířky.

Prohledávání do hloubky je provedeno voláním algoritmu Best Reply Search do určité hloubky. Při zadané hloubce 1 se algoritmus nedívá na tahy oponentů, ale pouze vybírá sekvence, které maximalizují heuristikou funkci pro naši AI. Od hloubky 2 již také probíhá prohledávání tahů oponentů. Viz sekci 3.3 pro přesnější popis fungování algoritmu Best Reply Search.

Prohledávání do šířky znamená neuvažování akcí, o kterých je předem známé, že nevedou k prospěchu hráče, nebo takové, které se nevlezly do vybrané množiny nejlepších akcí.

3.1.1 Výběr útoků

Nejprve je zjištěná celá množina proveditelných útoků, přičemž nejsou uvažovány útoky z políček, které mají stejně nebo méně kostek než cílové políčko. Výjimkou je útok z pole s 8 kostkami. Takto vybrané útoky jsou seřazeny dle rozdílu kostek zdrojového a cílového pole. Následně je vybráno pouze prvních A nejlepších útoků. A je parametr, který lze dynamicky měnit v průběhu hry.

3.1.2 Výběr přesunů

Opět jsou nejprve zjištěny všechny proveditelné přesuny. K vybrání těch nejlepších přesunů byl vytvořen systém jejich váhování dle důležitosti. Důležitost je tady definována jako vzdálenost cílového pole přesunu od nejbližší hranice oponenta. Maximální uvažovaná vzdálenost od nejbližší hranice oponenta označme M. Pak váha přesunu je definována jako

$$\text{Weight} = \frac{\text{Dice} \cdot \texttt{Dist}}{M},$$

kde Dice značí počet kostek k přesunu, a Dist nejbližší vzdálenost k hranici oponenta. Takto ováhované přesuny jsou seřany a vybráno pouze T nejlepších. T je parametr, který lze měnit dynamicky v průběhu hry.

3.1.3 Sestavování sekvencí

Tah je v této hře sekvence akcí, je proto prohledávání prostoru provedeno na celých sekvencích akcí. Sekvence je zahájena výběrem přesunů či výběrem útoků, viz předešlé podsekce. Každá z vybraných akcí tvoří základ nové sekvence. Tudíž počet sekvencí je omezen počtem vybraných akcí v prvním kroku. Poté ke každé sekvenci jsou vybíraný další akce a to takovým způsobem, že se vždy provede výběr útoků a přesunů, a z této množiny se vybere pouze ta akce, která vede k nejvyššímu ohodnocení heuristickou funkcí. Takto jsou přidávány tahy do sekvence, dokud se hodnota heuristické funkce zvyšuje. Pokud již není akce, která by heuristickou funkci pro danou sekvenci zvýšila, pak je sekvence zakončena akcí ukončení tahu.

Speciálním případem sekvencí je sekvence s jedinou akcí, kterou je ukončení tahu, která je vytvořena, když žádné vhodné tahy neexistují.

3.2 Heuristická funkce

Heuristická funkce je používaná k ohodnocení proveditelných akcí a jejich přidělení do sekvencí a také k ohodnocení celých sekvencí v rámci algoritmu Best Reply Search.

Tato funkce sestává v případě bez použití strojového učení ze dvou příznaku, kterými je počet polí hráče a váhovaného ohodnocení kostek hráče. Váhování je opět provedeno vzdáleností od nejbližší hranice, viz podsekce 3.1.2.

Konečně, funkce má tvar

$$f(p, m) = 4096 \cdot \text{areas}(p, m) + \text{weighted\_dice}(p, m),$$

kde p je hráč a m je stav hracího pole. Počet polí je násobeno libovolně velkou konstantou takovou (v tomto případě 4096), která zajistí nekonfliktní vyhodnocení obou částí heuristiky. Cílem je, aby zabrání pole převažovalo nad součtem váhovaných kostek od hranice, neboť při zabrání pole nastane, že se hranice posune a počet kostek u hranice se sníží, což by také snížilo heuristickou funkci.

3.3 Algoritmus Best Reply Search

Algoritmus Best Reply Search [2], dále jen BRS, je alternativou algoritmu Max^n [1] či Paranoid [3]. Stejně jako u těchto dvou algoritmů se i v BRS maximalzuje skóre pro našeho hráče a minimalizuje skóre pro oponenty. Oproti Max^n neuvažuje posloupnost všech oponentů, nýbrž odsimuluje všechny oponenty a vybere pouze toho s nejvyšším ohodnocením. Je to jakýsi kompromis mezi Max^n, který simuluje posloupnost tahů všech oponentů, a Paranoid, který slučuje

všechny oponenty do jednoho, čímž se prakticky chová paranoidně, neboť si myslí, že všichni oponenti jsou spolčeni proti němu.

Výhoda BRS je vyšší množství přohledání MAX uzlů, neboť i v případě více oponentů se uvažuje pouze jediný MIN uzel pro všechny oponenty.

3.3.1 Vliv hloubky prohledávání

V praxi je třeba uvažovat omezený čas dostupný k prohledávání, což vede na omezení hloubky a počtu prozkoumaných tahů v rámci hloubky. Prakticky je třeba tyto dva parametry vyvažovat. Vzhledem k tomu, že při hraní Dicewars se na budoucí tahy protihráčů nedívá tolik jako třeba v hře šachy, tak je vhodnější prozkoumat širší množinu tahů, ačkoliv do nižší hloubky.

Z výše uvedeného důvodu má AI dobrou úspěšnost i v případě prohledávání do hloubky 1, kdy nejsou uvažovány tahy protihráče.

Prohledávání do hloubky 4 je prakticky nemožné kvůli časovým omezením.

3.3.2 Aplikace zjištěných sekvencí

Agent si po zavolání BRS uloží celou sekvenci jako vnitřní stav. Při každém zavolání metody `ai_turn` agenta se vyjme první akce v sekvenci a zároveň se zkontroluje, jestli je akce proveditelná, neboť nějaký předešlý, jemu závislý tah se nemusel vykonat kvůli randomizovanému provádění útoků. Validní tah je pak vrácen. Jakmile je zbývající počet tahů v sekvence nulový, pak je opět zavolán algoritmus BRS ke zjištění nové sekvence.

4 Strojové učení

Model je konstruován tak, aby se učil vhodné váhy jednotlivých částí heuristické funkce.

4.1 Architektura

Byla zvolena jednoduchá dopředná neuronová síť s třemi lineárními vrstvami mezi nimiž je aktivační funkce ReLU. Celkový počet parametrů je 737

Layer	Type	Size	Input	Output	Params
0	Linear	32	Batch size × 5	Batch size × 32	192
1	ReLU	–	Batch size × 32	Batch size × 32	–
2	Linear	16	Batch size × 32	Batch size × 16	528
3	ReLU	–	Batch size × 16	Batch size × 16	–
4	Linear	1	Batch size × 16	Batch size × 1	17

Tabulka 1: Neuronová síť sloužící pro zjištění ohodnocení stavu hry, což je použito v heuristické funkci. Celkový počet trénovatelných parametrů je 737.

4.2 Příznaky na vstupu

Na vstupu modelu je stav hry z pohledu určitého hráče, a tento stav je reprezentován jako soubor 5 příznaků:

- Počet území AI - jako poměr ze všech území, tedy normalizovaně na rozsah [0, 1]
- Počet území oponentů - jako poměr ze všech území, tedy normalizovaně na rozsah [0, 1]
- Počet hracích kostek AI - normalizovaně na rozsah [0, 1], kde 1 značí maximální počet kostek ve hře
- Počet hracích kostek oponentů - normalizovaně na rozsah [0, 1]
- Relativní síla mezi naší hranici a všech oponentů vyjádřená jako rozdíl počtu kostek na hranicích - normalizovaně na rozsah [-1, 1].

Model se pak během trénování naučí váhovat vstupní příznaky a zkombinuje jejich hodnoty do jediného skóre, které je použito v heuristické funkci.

4.3 Heuristická funkce

Heuristická funkce se potom skládá ze dvou částí: výstup modelu a váhované ohodnocení kostek hráče na hrací ploše. Podobně jako v případě bez strojového učení je první část vynásobená dostatečně velkou konstantou. Heuristická funkce má tedy tvar:

$$f(p, m) = 32768 \cdot \text{model}(\text{features}) + \text{weighted\_dice}(p, m),$$

kde p je hráč a m je stav hracího pole.

4.4 Sběr trénovacích dat

K trénování modelu byly použity hry především zaznamenáváním výsledků turnajů z pohledů každé hráče. Toto zaznamenání bylo provedeno na straně klienta v souboru `ai_driver.py`. V konstruktoru se vytvoří list pro vkládání instancí stavu hry a rovnou se do něj vloží iniciální stav. V metodě `run` se pak při každém tahu klienta zaznamená stav hry. Na konci hry se v metodě `handle_server_message` ve stavu `game_end` zaznamená finální podoba hrací plochy. V tomto místě se taktéž serializuje celý zaznamenaný list stavů hry do souboru společně s informací, o jakého hráče (klienta) se jedná a kdo vyhrál. Všechny serializované hry se ukládají do složky `dataset` na nejvyšší úrovni projektu.

4.5 Trénování modelu

Pro účely trénování modelu se uložené hry načtou a vytvoří se příznaky popsané v sekci 4.2, které slouží jako vstup modelu. Očekávaný výstup modelu je hodnota v rozsahu [-1, 1]. Pokud daný hráč vyhrál, pak hodnota pro každý vstup ve hře se postupně zvyšuje od 0 do 1. V případě prohry je každému stavu přidělena hodnota 0 až -1. Tedy 0 značí začátek hry, 1 značí výhru a -1 prohru. Hodnoty mezi těmito extrémy značí vzdálenost od výhry nebo prohry.

K trénování je použit optimalizátor Adam a účelová funkce byla zvolena MSE, neboť se jedná o regresní úlohu. Parametr učení je nastaven na 0.001 a *batch size* na 16. Trénování je provedeno přibližně na 40 epochách asi na 350 instancích her. Další trénování po více epoch způsobuje přetrénování.

5 Práce s časem

Byly zjištěny přibližné časy zpracování, abychom měli lepší představu o tom, jak dlouho trvá zpracovat jednu akci. Následující statistiky byly zjištěny na hrách s maximálním počtem výběru útoků na 15, což omezuje počet jak prohledávaných sekvencí, tak i prohledávaných tahů. V hloubce 1 trvá najít jeden tah (sekvenci akcí) přibližně 0,15 sekundy a 0,0124 sekundy na jednu akci. V hloubce 2 je to 0,81 sekundy na sekvenci a 0,0561 sekundy na akci. V hloubce 3 je to přibližně 1,5 až 2 sekundy na sekvenci a 0,2 sekundy na jednu akci.

Průchod modelem, jehož výstup je používán heuristikou, trvá asi 0,06 ms. Tento rychlý průchod umožňuje vyhodnocovat heuristiku nad velkým množstvím tahů.

Během hry dynamicky měníme hloubku a měníme maximální počet zkoumaných útoků v každém jejich výběru. Pokud je zbývající čas více jak 20 sekund, pak nastavíme hloubku na 2, jinak ponecháváme na 1. Dále, pokud je zbývající čas vyšší než ten minulý, pak zvýšíme maximální možný počet prohledávaných útoků. Jinak tento parametr snížíme. Tento způsob umožňuje prohledává více do šířky než do hloubky. To je především proto, že u této hry člověk nepřemýšlí několik tahů dopředu jako to je u hry šachy, ale spíše se rozmýšlí, které z akcí v aktuálním tahu udělat.

6 Vyhodnocení

Nejprve jsme implementovali prohledávání stavového prostoru bez použití strojového učení. Provedli jsme turnajové vyhodnocení, jehož výsledky jsou vidět v tabulce 2. Je vidět, že naše umělá inteligence téměř dosahuje úspěšnost těch nejlepších agentů a to i přes to, že je použita velice jednoduchá heuristika.

kb.stei_at	37.29 % winrate	[88 / 236]
kb.stei_adt	37.08 % winrate	[89 / 240]
xpolok03	32.50 % winrate	[130 / 400]
kb.sdc_pre_at	19.17 % winrate	[46 / 240]
kb.stei_dt	10.17 % winrate	[24 / 236]
dt.stei	7.26 % winrate	[18 / 248]

Tabulka 2: Vyhodnocení umělé inteligence na 400 hrách bez použití strojového učení.

Poté jsme implementovali dopřednou neuronovou síť, jehož výstup je hlavní částí heuristiky. V tabulce 3 je vidět, že použití modelu zlepšilo výslednou úspěšnost a náš agent se úspěšností vyrovná ostatním agentům nebo je poráží.

kb.stei_adt	36.33 % winrate	[218 / 600]
kb.stei_at	36.33 % winrate	[218 / 600]
xpolok03	36.00 % winrate	[360 / 1000]
kb.sdc_pre_at	20.86 % winrate	[126 / 604]
kb.stei_dt	7.72 % winrate	[46 / 596]
dt.stei	4.00 % winrate	[24 / 600]

Tabulka 3: Vyhodnocení agenta na X hrách s použitím dopředné neuronové sítě k vyhodnocení stavu hrací plochy.

7 Závěr

V rámci projektu jsme vytvořili umělou inteligenci, která kombinuje prohledávání prostoru, přičemž používá natrénovaný model pomocí strojového učení jako heuristiku. Náš agent v turnaji dosahuje 36% úspěšnosti.

Reference

[1] C. A. Luckhardt and K. B. Irani. An algorithmic solution of n-person games. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, page 158–162. AAAI Press, 1986.

[2] M. Schadd and M. Winands. Best reply search for multiplayer games. *IEEE Transactions on Computational Intelligence and AI in Games*, 3:57 – 66, 04 2011. doi: 10.1109/TCIAIG.2011.2107323.

[3] N. Sturtevant and R. Korf. On pruning techniques for multi-player games. pages 201–207, 01 2000.