

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

SUI — Umělá inteligence a strojové učení
Umělá inteligence pro Válku kostek

Josef Kolář (xkolar71)
Dominik Harmim (xharmi00)
Petr Kapoun (xkapou04)
Jindřich Šesták (xsesta05)

19. prosince 2020

Obsah

| | | |
|----------|--|----------|
| 1 | Úvod | 1 |
| 2 | Implementace umělé inteligence prohledáváním stavového prostoru | 1 |
| 2.1 | Algoritmus Max^n | 1 |
| 2.2 | Konkrétní implementace a použití algoritmu Max^n | 2 |
| 3 | Strojově učený model pro heuristickou funkci | 3 |
| 3.1 | Serializace hry | 3 |
| 3.2 | Trénování modelu neuronové sítě | 4 |
| 3.3 | Integrace modelu | 5 |
| 4 | Vyhodnocení implementace umělé inteligence | 5 |
| 5 | Obsah odevzdaného archivu | 7 |
| 6 | Závěr | 7 |

1 Úvod

Cílem projektu bylo vytvořit *umělou inteligenci (AI)* pro hru **Válka kostek**. Umělá inteligence musí *prohledávat stavový prostor* hry a musí využívat *strojové učení*. Projekt je postaven na bakalářské práci [3]¹.

Dokumentace je dále strukturována následovně. Kapitola 2 pojednává o základní implementaci umělé inteligence založené na prohledávání stavového prostoru hry. Kapitola 3 popisuje rozšíření implementace o prvky strojového učení. Vyhodnocení výsledné umělé inteligence a srovnání použitých přístupů je popsáno v kapitole 4. Kapitola 5 vysvětluje obsah odevzdávaného archivu. Konečně kapitola 6 shrnuje výsledky tohoto projektu.

2 Implementace umělé inteligence prohledáváním stavového prostoru

Po analýze prostředí a implementace hry bylo zjištěno, že prostředí je *plně pozorovatelné* a *stochastické*. Na základě těchto vlastností bylo nejdříve uvažováno o implementaci umělé inteligence jako *agenta* (agent reprezentuje jednoho hráče), který bude implementovat *prohledávání stavového prostoru* algoritmem *ExpectiMiniMax* popsaným v původní bakalářské práci [3]. Tento algoritmus vychází z algoritmu *MiniMax*. Je však rozšířen o *nedeterministické* akce, proto je vhodný pro toto stochastické prostředí.

Algoritmus *MiniMax* i *ExpectiMiniMax* je však ve své standardní variantě definován pouze pro hru dvou hráčů. V tomto projektu je nutné uvažovat hru čtyřech (a obecně n) hráčů. Existuje zobecnění těchto algoritmů pro hru n hráčů. Jedná se o tzv. Max^n algoritmus, který lze opět rozšířit o nedeterminismus. Dále bude hovořeno pouze o Max^n algoritmu, ale ve výsledné implementaci tohoto projektu byl patřičně rozšířen i o nedeterministické akce. Algoritmus Max^n byl poprvé představen v článku *An Algorithmic Solution of N-Person Games* [1], kde je podrobně popsán spolu s problematikou her pro více hráčů. Článek *Comparison of Algorithms for Multi-player Games* [2] pak srovnává algoritmus Max^n s alternativními algoritmy pro hry více hráčů.

2.1 Algoritmus Max^n

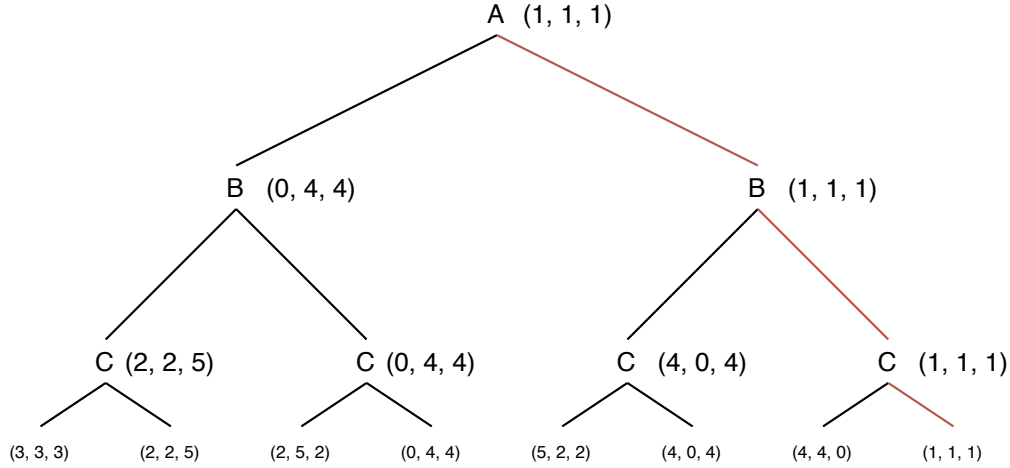
Algoritmus Max^n je přesně definován ve výše uvedených člancích [1, 2], proto bude v této kapitole už jen neformálně popsán a znázorněn.

V klasickém algoritmu *MiniMax* se střídají tahy dvou hráčů. Hráč na tahu — *MAX* — se snaží vybírat tahy vedoucí do stavů s maximálním ohodnocením, tj. snaží se maximalizovat svoji *objektivní funkci*. Protihráč — *MIN* — se naopak snaží minimalizovat objektivní funkci hráče *MAX*. Tyto dvě hodnoty lze reprezentovat jednou hodnotou vytvořením jejich sumy. Je také možné dívat se na tento problém tak, že každý z hráčů se snažím maximalizovat svoji vlastní objektivní funkci. Potom by se dalo při prohledávání stavového prostoru tohle reprezentovat jako dvojice maximálních hodnot objektivní funkce pro každého z hráčů.

Algoritmus Max^n potom funguje stejně jako *MiniMax*, kde se ale při prohledávání uvažuje n -tice, kde n je počet hráčů. Každý prvek této n -tice reprezentuje maximalizovanou hodnotu objektivní funkce daného hráče a každý z hráčů se snaží svoji objektivní funkci při svém tahu maximalizovat. Algoritmus je ilustrován na obrázku 1. Uvažuje se hra pro 3 hráče — A, B, C. Tito hráči se v uvedeném pořadí střídají při hraní svých tahů. Každý z hráčů může provést dva možné tahy. Po třech tazích hra skončí (obecně by po tahu hráče C hrál opět hráč A, ale zde jsou tahy hráče C uvažovány jako koncové, po nichž hra končí). Stav hry je tady popsán trojicí — $(obj(A), obj(B), obj(C))$ — kde $obj(X)$ je hodnota objektivní funkce hráče X. Na obrázku je vidět expandovaný graf hry pro dané hráče. Listové uzly jsou

¹Projekt je konkrétně založen na upravené verzi z původní bakalářské práce: <https://github.com/ibenes/dicewars>.

ohodnoceny objektivní funkcí a nelistové uzly jsou ohodnoceny podle maximalizace objektivní funkce aktuálního hráče na tahu. Je zřejmé, že hráč A učiní tah „vpravo“, protože stav, do kterého tento tah vede, je ohodnocen pro něho nejlepší objektivní funkcí — (1, 1, 1). Cesta do koncového stavu s tímto ohodnocením je vyznačena červenými hranami.



Obrázek 1: Příklad *prohledávání stavového prostoru* hry pro 3 hráče algoritmem Max^n

2.2 Konkrétní implementace a použití algoritmu Max^n

Tato kapitola popisuje výslednou implementaci umělé inteligence prohledáváním stavového prostoru algoritmem Max^n pro tento projekt. Dále popisuje rozšíření tohoto algoritmu o nedeterministické akce a konkrétní použití v projektu.

Implementace prohledávání stavového prostoru se nachází ve třídě `dicewars.ai.xkolar71.AI` (kromě implementace heuristické funkce pro ohodnocování stavů hry, která je implementována ve třídě `dicewars.ai.xkolar71_orig.AI`, vzhledem k tomu, že heuristická funkce byla nakonec nahrazena strojově učeným modelem, viz kapitola 3).

Definice 2.1 ($maxN$). Byla definována *rekursivní* funkce $maxN$, která implementuje algoritmus Max^n pro účely tohoto projektu. Funkce v každém tahu daného hráče zkoumá následující tahy všech ostatních hráčů v patřičném pořadí a vrací nejlepší spočítaný tah. Funkce konkrétně uvažuje n iterací prohledávání tahů všech následujících hráčů. Parametr n byl experimentálně zvolen na hodnotu 1, tj. uvažuje se nanejvýše jedenkrát sekvence tahů každého z následujících hráčů. Při vyšších hodnotách tohoto parametru už byly naměřeny horší výsledky, což je způsobené tím, že prostředí hry je *silně nedeterministické* a jen obtížně lze přesněji predikovat výsledky akcí dále do budoucnosti. Při zkoumání tahů hráče následujícího v pořadí se generuje m útoků. Následně z každého útoku se generuje $m - 1$ útoků, z každého z nich $m - 2$ útoků atd. až do hloubky l . Parametry m a l byly experimentálně nastaveny na hodnotu 5, opět s ohledem na stochastické prostředí. Útoky, které se generují v daném tahu, jsou vybrány jako ty možné útoky s *největší pravděpodobností* na výhru funkcí *possibleTurns*, viz definice 2.2. Tímto byl algoritmus Max^n rozšířen o nedeterminismus. Uvažované útoky jsou vždy simulovány na kopii aktuální konfigurace hry. Funkce $maxN$ potom v listových uzlech (definovaných parametry n a l) ohodnotí daný stav *heuristickou funkcí* h , viz definice 2.3. Konečně $maxN$ maximalizuje hodnotu funkce h pro každého z hráčů a vrací tah do uzlu s nejlepším ohodnocením.

Definice 2.2 (*possibleTurns*). Tato funkce vrací všechny tahy, které je možné provést seřazené podle *pravděpodobnosti* p . Tahy, které je možné provést, jsou všechny možné útoky (tj. útoky z území, na nichž jsou alespoň 2 kostky a sousedí s územím protihráče), jejichž pravděpodobnost p je větší než 0,4 (bylo zvoleno experimentálně) a nebo je počet kostek na útočícím území roven 8 (největší možný

počet kostek na území). Pravděpodobnost p je spočtena následovně $p = P_{A \rightarrow D} \cdot P_D \cdot P_L$, kde $P_{A \rightarrow D}$ je pravděpodobnost úspěšného útoku z území A na území D , P_D je pravděpodobnost udržení území D po úspěšném útoku při tazích následujících hráčů a P_L je 2 (zvoleno experimentálně) v případě, že území A leží v největším regionu daného hráče, tj. je proveden útok z největšího území hráče, v opačném případě je $P_L = 1$. Výpočty těchto pravděpodobností jsou blíže diskutovány v původní bakalářské práci [3].

Definice 2.3 (h). Hodnota této *heuristické funkce* udává ohodnocení stavu hry pro každého hráče ve smyslu maximalizace jeho *objektivní funkce*. Hodnota h pro hráče i je spočtena následovně $h(i) = D(i) + \sum_r^{R(i)} 5r + 50 \max(R(i))$, kde $D(i)$ je počet kostek hráče i , $R(i)$ jsou velikosti regionů hráče i a číselné konstanty byly zvoleny experimentálně.

Na nejvyšší úrovni je tah hráče určen tak, že pokud je zbývajících časové omezení tahu větší než x sekund a zároveň je počet tahů v aktuální sérii tahů daného hráče menší než y , tak se nalezne nejlepší tah funkcí $\max N$. V opačném případě se vybere nejlepší možný útok s ohledem na pravděpodobnost výhry funkcí possibleTurns . Pokud žádný takový útok není možný, tah se ukončí. Parametry x a y byly experimentálně nastaveny na hodnoty 1,0 a 5.

3 Strojově učený model pro heuristickou funkci

Model substituuje *heuristickou funkci* a pro každého hráče vyhodnocuje v rámci aktuální konfigurace hry jeho šanci na výhru. Vstupem je tedy serializace aktuální konfigurace hry (stavu desky a stavu hráčů) do XX celých čísel a výstupem je vektor 4 hodnot v intervalu $< 0; 1 >$, který udává odhad modelu na výhru jednotlivých hráčů; 0 značí, že model si je jistý s nevýhrou hráče s konkrétním indexem (resp. jménem).

3.1 Serializace hry

Pro trénování modelu jsme navrhli serializaci hry do vektoru XX celých čísel, který odpovídá konkrétnímu stavu hry, tedy desky a hráčů na ní hrajících. Výsledný vektor má následující strukturu:

- matice sousednosti M_s definovaná jako

$$M_s = \begin{bmatrix} s(1,1) & s(1,2) & s(1,3) & \dots & s(1,29) \\ \cdot & s(2,2) & s(2,3) & \dots & s(2,29) \\ \cdot & \cdot & s(3,3) & \dots & s(3,29) \\ \cdot & \cdot & \cdot & \dots & s(4,29) \\ \cdot & \cdot & \cdot & \cdot & \dots \\ \cdot & \cdot & \cdot & \cdot & s(29,29) \end{bmatrix}$$

kdy funkce $s(x, y)$ je definovaná následovně

$$s(x, y) = \begin{cases} 1 & \text{jestliže území } x \text{ sousedí s územím } y \\ 0 & \text{jinak} \end{cases}$$

a výsledný vektor sousednosti je poté získán jako konkatenace řádků M_s z trojúhelníku nad diagonálou. Jeho celková délka je $\frac{30 \cdot (30-1)}{2} = 435$ prvků.

- vlastníci polí — tedy vektor o délce 30 s hodnotami, resp. názvy, jednotlivých vlastníků polí, opět 30 prvků
- počty kostek — vektor udávající množství kostek pro každé pole na desce
- velikosti největších regionů — vektor o délce 4 s $\max(R(i))$ pro každého hráče i

Serializovaná hra je tedy vektor celočíselných nezáporných čísel složený ze čtyř sektorů s celkovou délkou $435 + 30 + 30 + 4 = 499$ prvků.

Po každém ukončeném tahu hráče je uschována serializace hry. Po konci hry je ke všem serializacím přiložen index vítěze konkrétní posloupnosti tahů a tento datový balíček je binárně uložen na disk. Pro tento účel byla upravena metoda `run` třídy `Game`, která se nachází v modulu `dicewars.server.game`. Pro tuto serializaci se dále používají funkce z vytvořeného modulu `dicewars.ml`.

3.2 Trénování modelu neuronové sítě

Všechny skripty zmíněné v této sekci se nachází v adresáři *supplementary/ml-scripts*.

Čtyřikrát původní AI s Max^n algoritmem, souborový systém se soubory s jednotlivými konfiguracemi. Pro finální natrénování modelu bylo použito 201 291 různých konfigurací her s odpovídajícími indexy vítězů. Pro generování konfigurací byla hra spuštěna vytvořeným skriptem `dump-data-to-learn.py`.

Pomocí skriptu `transform-to-numpy.py` je tato datová sada ze souborů transformována na pole typu `numpy` o dimenzích $(N, 1+499)$ — N značí celkový počet konfigurací a $1+499$ je index vítěze a kompletní serializace hry. Takto zkonstruované pole je uloženo jako jeden samostatný soubor pro další zpracování.

Implementace dalšího zpracování datové sady je pak v souboru `shuffle-datasets.py`, který ji načte, náhodně zamíchá podle první dimenze (tedy pořadí různých konfigurací) a následně rozdělí na *trénovací*, *validační* a *testovací* data. To dělá v poměru 70 %, 20 % a zbylých 10 % pro testovací data. Takto rozdělená sada je následně uložena do třech samostatných souborů.

Trénovací a validační část datové sady je následně použita pro natrénování modelu (skript `train-model.py`) — tím je v tomto případě *lineární neuronová síť*. Její architektura byla zvolena experimentálně a celkově se skládá z pěti vrstev, z čehož první vrstva je čistě vstupní se vstupním vektorem o délce 499 (délka vychází ze způsobu serializace). Následuje plně propojená vrstva s 64 parametry a ReLU aktivací, jako třetí figuruje v síti `Dropout` vrstva pro prevenci přeučení s pravděpodobností vynulování vstupů $p = 0,25$ a model je zakončen dvěma plně propojenými vrstvami s 32, resp. 4 parametry. Čtvrtá vrstva používá opět aktivaci ReLU, výstupní poté `softmax`. Níže je přiložen programový výstup z `keras` popisu modelu.

| Layer (type) | Output Shape | Param # |
|--------------------------|---------------|---------|
| input_1 (InputLayer) | [(None, 499)] | 0 |
| dense (Dense) | (None, 64) | 32000 |
| dropout (Dropout) | (None, 64) | 0 |
| dense_1 (Dense) | (None, 32) | 2080 |
| dense_2 (Dense) | (None, 4) | 132 |
| Total params: 34,212 | | |
| Trainable params: 34,212 | | |
| Non-trainable params: 0 | | |

Nutno zmínit, že v případě předpokládaných výsledků pro každou konfiguraci dochází ke *kategorizaci* — transformaci celočíselného indexu vítěze na vektor o počtu hráčů s hodnotou 1 na místě indexu vítěze

a hodnotami 0 na ostatních pozicích — tedy platí, že pro vítěze s indexem i je modelu předložen předpokládaný vektor $[x_1 \ x_2 \ x_3 \ x_4]$ u kterého platí, že $x_i = 1$, a 0 jinak.

Pro samotné natrénování je použit algoritmus Adam pro optimalizátor s *learning rate* na hodnotě 0,01. Jako *loss* pak trénování používá instanci `CategoricalCrossentropy` dodanou přímo knihovnou `keras`. Jako velikost trénovací i validační dávky jsme experimentálně stanovili hodnotu 32 a počet epoch jsme opět experimentálně nastavili na 15.

Pro ověření správnosti modelu jsme připravili skript `check-test-accuracy.py`, který určí přesnost odhadů nad částí datové sady určenou pro testování (tedy 10 % z původní sady). Natrénovaný model z testovací datové sady o velikosti 20 130 správně určil celkem 19 804 vzorků, což značí úspěšnost cca 98,3 %.

Pro ověření rychlosti vyhodnocení modelu pro vstupní data jsme připravili skript `profile-predict.py`, s pomocí kterého jsme za asistence `CProfile` profileru měřili rychlost vyhodnocení. Pro výše definovanou síť a testovací část datové sady bylo experimentálně změřeno první vyhodnocení cca 1,5 násobně delší, než ty následující (tedy 550 ms oproti 350 ms) — což je známý a popsáný problém modelů založených na TensorFlow².

3.3 Integrace modelu

Pro použití natrénovaného modelu v AI je upravena heuristická funkce, která nyní na základě předané konfigurace hry vyhodnocuje jednotlivé hráče, resp. jejich šanci na vítězství. Konkrétně se jedná o funkci `_heuristic(players: List[int], board: Board)`, která na základě předaného seznamu aktivních hráčů a instance desky provede nejprve serializaci (dle postupu popsaného výše), následně vyhodnotí konfiguraci v natrénovaném modelu a nazpět vrátí ohodnocení stavu hry pro jednotlivé hráče — konkrétní hodnota pro hráče odpovídá maximalizaci jeho *objektivní funkce*.

4 Vyhodnocení implementace umělé inteligence

Po prvotní implementaci algoritmu Max^n pro účely umělé inteligence pro tuto hru jsme začali experimentovat s hraním soutěžních turnajů, kterých se účastnila i tato inteligence. Do turnajů jsme zapojovali umělé inteligence dle zadání výsledného soutěžního turnaje, tedy `dt.ste`, `dt.wpm_c`, `dt.sdc`, `xlogin00` a nakonec i `xkolar71`. Po prvotním odladění parametrů (popsaných v kapitole 2.2) na základě experimentů, se turnajová úspěšnost pohybovala mezi 35 % a 60 % — viz výsledky turnajů zobrazených v 2.

S prvními výsledky po implementaci algoritmu Max^n jsme vygenerovali serializace konfigurací her určené pro natrénování modelu, jak je popsáno v 3.2, a natrénovaný model následně integrovali do umělé inteligence, konkrétně do její heuristické funkce (jak popisuje 3.3). Nad takto upravenou umělou inteligencí jsme spustili další turnaje, z nichž naše umělá inteligence vycházela s úspěšností na úrovni umělé inteligence bez integrace modelu — na základě tohoto pozorování jsme iterativně vylepšovali trénování modelu (experimentální změny architektury sítě, konkrétní struktury serializovaných her či velikostí datových sad).

Tyto iterace změn vedly ve výsledku k úspěšnosti umělé inteligence s integrovaným modelem na úrovni stabilních 39 % až 43 % na turnajích s násobně vyššími počty her (nad 1000). Výsledky z některých turnajů jsou zobrazeny v 3 a nutné je poznamenat, že AI `xkolar71` s integrovaným modelem vycházela z těchto turnajů úspěšněji než verze AI `xkolar71_orig`, která je původní implementací bez integrovaného modelu.

²<https://github.com/tensorflow/tensorflow/issues/39458>


```
dt.sdc 34.38 % winrate [ 11 / 32 ] 37.5/24 34.4/32 33.3/24 37.5/24 29.2/24
xkolar71 34.38 % winrate [ 11 / 32 ] 41.7/24 25.0/24 33.3/24 37.5/24 34.4/32
dt.ste 25.00 % winrate [ 8 / 32 ] 29.2/24 25.0/24 25.0/32 20.8/24 25.0/24
dt.wpm_c 25.00 % winrate [ 8 / 32 ] 16.7/24 29.2/24 25.0/24 25.0/32 29.2/24
dt.rand 6.25 % winrate [ 2 / 32 ] 6.2/32 8.3/24 8.3/24 4.2/24 4.2/24
```

```
xkolar71 60.42 % winrate [ 29 / 48 ] 59.4/32 62.5/24 67.9/28 50.0/28 62.5/32 60.4/48
dt.ste 38.46 % winrate [ 20 / 52 ] 44.4/36 43.8/32 38.5/52 42.9/28 34.4/32 25.0/28
dt.wpm_c 30.36 % winrate [ 17 / 56 ] 29.5/44 30.6/36 25.0/28 30.4/56 40.6/32 25.0/28
dt.sdc 17.31 % winrate [ 9 / 52 ] 19.4/36 17.3/52 9.4/32 25.0/36 17.9/28 12.5/24
dt.rand 5.00 % winrate [ 3 / 60 ] 5.0/60 5.6/36 5.6/36 6.8/44 3.1/32 3.1/32
xlogin00 3.85 % winrate [ 2 / 52 ] 6.2/32 3.6/28 3.1/32 3.1/32 3.8/52 3.1/32
```

```
xkolar71 45.00 % winrate [ 27 / 60 ] 45.5/44 39.6/48 47.7/44 47.7/44 45.0/60
dt.ste 33.82 % winrate [ 23 / 68 ] 34.6/52 33.8/68 34.6/52 36.5/52 29.2/48
dt.wpm_c 18.75 % winrate [ 12 / 64 ] 20.8/48 17.3/52 18.8/64 18.8/48 18.2/44
dt.sdc 17.19 % winrate [ 11 / 64 ] 17.2/64 19.2/52 16.7/48 16.7/48 15.9/44
xlogin00 10.94 % winrate [ 7 / 64 ] 10.4/48 13.5/52 10.4/48 10.9/64 9.1/44
```

Obrázek 2: Výsledky soutěžních turnajů po prvotní implementaci umělé inteligence — tedy bez strojového učení

```
xkolar71 39.43 % winrate [ 358 / 908 ] 40.7/452 41.1/440 35.9/468 40.1/444 42.8/444 39.4/908 36.3/476
dt.ste 37.66 % winrate [ 351 / 932 ] 42.1/468 40.3/476 37.7/932 35.6/464 40.2/448 33.1/468 34.7/472
xkolar71_orig 35.04 % winrate [ 321 / 916 ] 35.5/456 33.3/456 34.7/472 36.7/444 36.9/444 33.2/476 35.0/916
dt.wpm_c 24.67 % winrate [ 224 / 908 ] 28.5/452 23.7/452 19.0/464 24.7/908 31.6/468 22.3/444 22.7/444
dt.sdc 24.34 % winrate [ 223 / 916 ] 30.7/456 24.3/916 22.3/476 24.1/452 26.9/468 19.3/440 22.6/456
xlogin00 7.93 % winrate [ 72 / 908 ] 10.8/452 5.8/468 7.1/448 7.5/468 7.9/908 9.0/444 7.4/444
dt.rand 5.59 % winrate [ 51 / 912 ] 5.6/912 7.5/456 4.9/468 7.5/452 6.2/452 2.9/452 4.6/456
```

```
xkolar71 39.00 % winrate [ 443 / 1136 ] 44.6/572 37.1/568 34.1/580 41.4/568 40.7/560 39.0/1136 36.1/560
xkolar71_orig 35.87 % winrate [ 406 / 1132 ] 37.1/564 37.5/560 35.6/568 36.0/572 38.3/572 30.7/560 35.9/1132
dt.ste 34.67 % winrate [ 405 / 1168 ] 37.8/576 36.0/620 34.7/1168 35.2/576 38.4/584 29.5/580 31.0/568
dt.sdc 27.09 % winrate [ 311 / 1148 ] 32.6/564 27.1/1148 25.5/620 24.1/568 28.4/564 26.8/568 25.4/560
dt.wpm_c 25.00 % winrate [ 285 / 1140 ] 28.5/576 21.0/568 22.9/576 25.0/1140 31.6/560 22.0/568 24.1/572
xlogin00 7.48 % winrate [ 85 / 1136 ] 7.9/568 6.0/564 8.0/584 7.9/560 7.5/1136 8.2/560 6.8/572
dt.rand 5.70 % winrate [ 65 / 1140 ] 5.7/1140 7.1/564 4.5/576 5.2/576 7.6/568 4.7/572 5.1/564
```

```
xkolar71 43.56 % winrate [ 115 / 264 ] 45.0/160 42.5/160 37.2/156 44.9/156 48.1/160 43.6/264
dt.ste 41.29 % winrate [ 109 / 264 ] 45.6/160 37.8/156 41.3/264 45.0/160 39.4/160 38.5/156
dt.sdc 28.41 % winrate [ 75 / 264 ] 30.5/164 28.4/264 25.0/156 28.2/156 32.7/156 25.6/160
dt.wpm_c 23.51 % winrate [ 63 / 268 ] 26.2/168 25.0/156 21.9/160 23.5/268 26.8/164 17.3/156
xlogin00 8.58 % winrate [ 23 / 268 ] 11.0/164 9.0/156 10.6/160 4.9/164 8.6/268 7.5/160
dt.rand 5.51 % winrate [ 15 / 272 ] 5.5/272 5.5/164 3.8/160 6.5/168 6.1/164 5.6/160
```

Obrázek 3: Výsledky soutěžních turnajů po integraci naučeného modelu — tedy se strojovým učním. AI xkolar71_orig značí umělou inteligenci bez integrovaného modelu

5 Obsah odevzdaného archivu

Odevzdaný archiv `xkolar71.zip` obsahuje následující soubory:

- `xkolar71/__init__.py` a `xkolar71/ai.py`: Balíček, který obsahuje implementaci umělé inteligence, která *používá strojové učení*. Viz kapitola 3.
- `xkolar71/model.h5`: Natrénovaný model založený na *neuronové síti*, který se používá pro modelování *heuristické funkce* pro ohodnocování stavů hry.
- `xkolar71/game.py`: Modul s implementací serializační funkce popsané v kapitole 3.1.
- `supplementary/dicewars/ai/xkolar71_orig.py`: Původní implementace umělé inteligence *bez strojového učení* popsaná v kapitole 2.
- `supplementary/dicewars/ai/xkolar71_2.py`, `supplementary/dicewars/ai/xkolar71_3.py`, `supplementary/dicewars/ai/xkolar71_4.py`: Další instance původní implementace umělé inteligence `supplementary/dicewars/ai/xkolar71_orig` vytvořené pro účely trénování modelu.
- `supplementary/dicewars/server/game.py`: Modifikovaný soubor s třídou, která řídí hru. Byla provedena úprava pro generování konfigurací hry za účelem trénování modelu. Viz kapitola 3.1.
- `supplementary/dicewars/ml/__init__.py` a `supplementary/dicewars/ml/game.py`: Balíček obsahující funkce sloužící k serializaci konfigurací hry. Viz kapitola 3.1.
- `supplementary/ml-scripts/dump-data-to-learn.py`: Skript pro generování konfigurací hry. Viz kapitola 3.2.
- `supplementary/ml-scripts/transform-to-numpy.py`: Skript pro převod vygenerovaných konfigurací hry na pole typu `numpy`. Viz kapitola 3.2.
- `supplementary/ml-scripts/shuffle-datasets.py`: Skript pro zamíchání konfigurací hry a jejich následné rozdělení na *trénovací*, *validační* a *testovací* datové sady. Viz kapitola 3.2.
- `supplementary/ml-scripts/train-model.py`: Skript pro vytvoření a trénování modelu založeném na neuronové síti. Viz kapitola 3.2.
- `supplementary/ml-scripts/check-test-accuracy.py`: Skript pro ověření správnosti modelu na testovacích datech. Viz kapitola 3.2.
- `supplementary/ml-scripts/profile-predict.py`: Skript pro ověření rychlosti predikce na testovacích datech. Viz kapitola 3.2.
- `requirements.txt`: Aktualizovaný seznam knihoven, na kterých je projekt závislý.
- `xkolar71.pdf`: Tato dokumentace k projektu.

6 Závěr

V rámci řešení tohoto projektu jsme po úvodním rozdělení práce nastudovali funkci algoritmu Max^n , abychom jej mohli následně implementovat a experimentálně nastavit jednotlivé parametry pro jeho běh — samotná funkce algoritmu je stručně popsána v kapitole 2 a to včetně popisu konkrétní implementace a jednotlivých parametrů. Po prvotní implementaci dosahovala umělá inteligence míry výher pohybujících nad úrovní 30 %, ve většině případů na prvním místě spouštěných turnajů.

Takto chovající se umělá inteligence byla následně použita (resp. čtyři identické její instance) pro vygenerování trénovací a validační datové sady, podle kterých byl poté natrénován model ze třívrstvé neuronové sítě — tento proces včetně architektury sítě a trénovacího *workflow* je popsán v kapitole 3.

Po integraci do původního algoritmu pomocí substituce heuristické funkce se míra výher nové umělé inteligence pohybovala nad 35 % a i nadále na prvních pozicích desítek soutěžních turnajů.

Závěrem této dokumentace je výčet souborů v odevzdaném archivu včetně popisu funkce jednotlivých skriptů pro možnou replikovatelnost.

Reference

- [1] LUCKHARDT, C. A. a IRANI, K. B. An Algorithmic Solution of N-Person Games. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence, Lecture Notes in Computer Science*. Philadelphia, Pennsylvania: AAAI Press, srpen 1986. S. 158—162. AAAI’86, sv. 2883. Dostupné na: <<https://www.aaai.org/Papers/AAAI/1986/AAAI86-025.pdf>>.
- [2] STURTEVANT, N. A Comparison of Algorithms for Multi-player Games. In SCHAEFFER, J., MÜLLER, M. a BJÖRNSSON, Y. (ed.). *Computers and Games*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. S. 108–122. ISBN 978-3-540-20545-6.
- [3] TUREČEK, D. *Umělá inteligence pro deskovou hru*. Brno: Vysoké učení technické v Brně, Fakulta informačních technologií, 2018. Bakalářská práce. Ústav počítačové grafiky a multimédií. Vedoucí práce Ing. Karel Beneš. Dostupné na: <<https://www.fit.vut.cz/study/thesis/20290>>.