SUI - Umělá inteligence a strojové učení

Technická zpráva k projektu Dicewars

FIT VUT v Brně, 2021

Název týmu
Tým xklism00

Autoři
Kliš Michal, Bc. (xklism00)
Gregorová Jana, Bc. (xgrego20)
Lorenc Jan, Bc. (xloren15)
Sova Michal, Bc. (xsovam00)

Agent

Za herní styl agenta byla zvolena střední cesta mezi agresivní a defenzivní strategií. Agent začíná kolo posílením svých hranic. Následuje fáze, v níž provádí útoky. Na konci kola poté stáhne kostky z ohrožených území, aby o ně nepřišel.

Agent pracuje ve dvou módech. První z nich je mód útoku, kdy se snaží posílit hranice, pokud je to možné, pokud ne a existuje vhodný útok, tak útok provede. Jestliže tedy během útočení má volné transfery, může útoky a transfery kombinovat. Transfer má vždy přednost. Jakmile nastane situace, kdy se neprovedl transfer k posílení hranic ani útok, přesouvá se agent do defenzivního módu. Již neútočí ani neposiluje hranice a jen stahuje kostky z ohrožených území, dokud taková území existují a má volné transfery.

Z celkem 6 možných transferů má pro posilování hranic k dispozici 5 a jeden si schovává pro evakuaci kostek. Nicméně, oněch 5 posilovacích transferů nemusí nutně využít a tedy může mít k evakuaci i více než jeden zaručený transfer. K této strategii se dospělo po dlouhém testování, které je více popsáno v kapitole Vyhodnocování a výsledky.

Výběr posilovacích transferů implementovaný metodou `transfer_to_border()` z jmenného prostoru `Utils` probíhá následovně. Hraniční území největšího regionu s méně než osmi kostkami se seřadí vzestupně podle skóre hodnocení konkrétního území. Toto skóre je dáno kombinací pravděpodobnosti udržení pole do dalšího kola a natrénovanou heuristickou funkcí vysvětlenou dále. Nejslabšímu území se agent pokusí provést transfer kostek z jeho vnitřního souseda (tedy souseda, který není zároveň hraničním), jenž má nejvíce kostek. Přirozeně se může stát, že takový soused neexistuje nebo nemá dost kostek. V takovém případě se s výběrem hraničního pole k posílení postupuje druhým nejslabším až k nejsilnějšímu. Nedojde-li ani tak ke všem dostupným transferům, stále může dojít k již zmíněnému zpětnému posílení i mezi útoky. I poté přebytečné transfery lze stále použít k evakuaci.

Výběr útoku poté provádí algoritmus expectiminimax, jehož implementace je popsána v následující kapitole. Zjednodušeně však vybere možné útoky s dostatečně velkou pravděpodobností úspěchu, ohodnotí pro ně stav hry, seřadí je dle skóre a k útoku vybere nejlepší tah. Jestliže nebyl vrácen žádný tah, agent se přesouvá do závěrečné defenzivní fáze kola.

K možné evakuace agent zvolí všechna hraniční území největšího regionu s více než jednou kostkou a pravděpodobností na udržení nižší než 50%. Tato území ohodnotí stejně jako při posilovacích transferech. Pokud opačný jev skóre (1 - skóre) vynásobíme počtem kostek, získáme jistou míru ztráty, pokud o území přijdeme. Tato ztráta se vypočítá před a po simulaci transferu a zvolí se transfer s největším rozdílem ztrát, tedy kde nejvíce získáme, provedeme-li daný transfer. Výběr transferu implementují metody `transfer_from_border()` a `transfer_from_border_loss()` ve jmenném prostoru `Utils`.

Prohledávání stavového prostoru

Jak již bylo zmíněno, stavový prostor je prohledáván algoritmem expectiminimax, jenž implementuje metoda `run_expectiminimax()` třídy `AI`. Za stavy (možné tahy hráče) jsou považovány možné útoky, které mají pravděpodobnost úspěchu i následného udržení alespoň 55% (cíl je být trochu silnější než jen 50%) nebo jsou vedeny z území s plným počtem kostek, tedy 8. Tyto jsou získány metodou `get_possible_attacks()` třídy `Utils`, která kromě útoků vrací i jejich pravděpodobnostní skóre dané vynásobením pravděpodobnosti úspěchu útoku a pravděpodobnosti následného udržení území. Samotné prohledávání spočívá v simulaci provedení tahu a reakce protihráčů. Hloubka zanoření simulace není omezena, nicméně pro hloubku 4 a více již výpočet trvá poměrně dlouho, a proto bylo zvoleno zanoření hloubky 3, pokud agentovi zbývá dost času, nebo hloubky 2 či 1, pokud je času méně.

Simulace probíhá následovně. Na samotném počátku získáme možné útoky splňující výše zmíněné metodou `get_possible_attacks()`. Pro každý útok (stav) se zkopíruje hrací deska, na níž se provede daný útok. Jedná-li se o nejhlubší zanoření, vyhodnotí se heuristikou skóre stavu hry, které se dále vynásobí pravděpodobnostím skóre útoku, čímž získáme ohodnocení tahu. Heuristika se násobí pravděpodobnostím skóre útoku proto, že je lepší získat méně, ale jistě, než hodně, ale s mnohem menší pravděpodobností. Vynásobení těchto dvou významných faktorů se získá nejlepší kompromis. Použití samotné heuristiky taktéž dosahuje pěkných výsledků, nicméně v kombinaci s pravděpodobnostím skóre bývá winrate v průměru o 2% vyšší. Pokud se nejedná o dno zanoření, spustí se metoda rekurzivně pro všechny ostatní stále žijící hráče. Na zkopírované hrací desce se provedou i jejich tahy a až poté se vypočítá a uloží skóre. Metoda poté vrací nejlépe hodnocený útok, tedy akci, nebo žádný útok značící, že již není výhodné žádnou akci provádět.

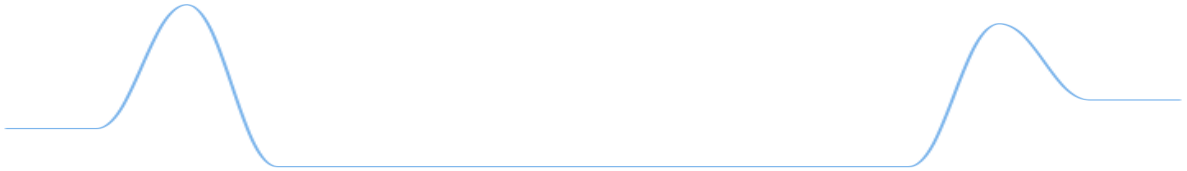
Heuristika

Heuristická funkce zastřešená třídou Heuristic vyhodnotí stav hry pro daného hráče na základě modelu natrénované neuronové sítě v souboru model.zip python modulu torch. Prakticky se jedná o pravděpodobnost (jistotu modelu) výhry hráče v daném stavu, což je jeden z důvodů proč lze snadno kombinovat s pravděpodobností úspěchu a udržení útoku, neboť se jedná o dvě pravděpodobnosti.

Zatímco pro hodnocení hry v expectiminimaxu vrací metodou evaluate_board_state() pouze svou pravděpodobnost pro daného hráče, která je dále násobená pravděpodobnostím skóre útoku, pro transfery je použita funkce evaluate_tranfer_score(), která kromě modelu vyhodnotí i pravděpodobnost udržení území. Tyto 2 pravděpodobnosti sečte a normalizuje do 0-1 dělením dvěma. Zde by již násobení mělo negativní vliv na výsledek následného násobení počtem kostek území.

Sběr dat pro model

V první řadě bylo potřeba reprezentovat stav hry vektorem, tedy formátem potřebným pro strojové učení. To provádí metoda vectorize_game_state() třídy Utils dle následující ideje. Je potřeba zaznamenat, který hráč vlastní která území. Území je celkem 34, takže jména=čísla hráčů vlastnících území je jedněch 34 hodnot pro vektor. Dále je třeba znát počet kostek v územích (dalších 34 hodnot). Důležitou roli hraje i distribuce území. Poněvadž nepotřebujeme duplikáty ani informace o tom, že území sousedí se sebou samým, máme celkem (34*34-34)/2=595 dvojic území, pro něž volíme hodnotu 1, pokud spolu sousedí a 0, pokud ne. Významnými faktory jsou dále počty vlastněných území pro každého hráče (další 4 hodnoty) a kolik z nich patří do největšího regionu (4 hodnoty). Celkem tedy získáme vektor o délce 34+34+595+4+4=637 reprezentující stav hry. Tyto části vektoru je pak dobré poskládat tak, aby co nejvíce vynikly rozdíly hodnot (např. sousedství nabývají hodnot jen 0 nebo 1, území může být až 34, hráči jsou v rozmezí 1-4, kostky pak 1-8). Při poskládání za sebe v pořadí vlastníci území, počty vlastněných území, sousedství, počty území v největších regionech a počty kostek vznikne následující silně přibližná distribuce hodnot.



Samotný sběr pak provádí třída DataCollector v souboru data_collector.py, která je použita v souboru game.py řídící hru. Na konci každého kola uloží stav hry (vektor). Na konci hry pak všem stavům přiřadí vítěze a jako pandas DataFrame se sloupci BoardState (vektor stavu hry) a Winner (jméno=číslo hráče) uloží modulem pickle do složky supp-xklism00/raw_data. Modifikovaný soubor game.py i data_collector.py lze nalézt v doprovodné složce supp-xklism00 a pro sbírání dat je stačí překopírovat do dicewars/server (game.py nahradit). Pak se data sbírají při každém spuštění libovolné hry.

Pro trénování modelu jsme si rozkopírovali agenta kb.stei_adt, neboť se jevil z dostupných existujících agentů nejsilnější a spustili turnaj 4 hráčů o 500 hrách (tedy 2000 celkem) pro

tyto agenty. Bylo zapotřebí hry spouštět se stejnými agenty pro rovnoměrné šance na výhry v daných stavech. Celkem bylo nasbíráno 706 266 stavů, z toho 288 378 mělo vítěze prvního hráče, 167 591 druhého, 144 059 třetího a 106 238 čtvrtého. Jak lze vidět, záleží na pořadí.

Dále data pro trénování modelu připraví skript `data_preparation.py`. Ten načte všechny výsledky a sjednotí je do jednoho datasetu, který zamíchá. Dále sjednotí množství dat pro jednotlivé třídy (hráče) dle nejmenšího (pro trénování je dobré mít stejně dat pro všechny třídy), tedy dost zahodí. Data opět zamíchá a rozdělí v poměru 80:20 (o velikostech 363 920 a 90 980 stavů hry) na trénovací a validační dataset, který ve dvou souborech uloží do složky `supp-xklism00/data` opět v podobě `pickle` formátu.

Trénování modelu

Za model byla použita lineární neuronová síť následující architektury:

```
NeuralNet(
  (seq): Sequential(
    (0): Linear(in_features=637, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=64, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.25, inplace=False)
    (9): Linear(in_features=32, out_features=4, bias=True)
    (10): Softmax(dim=1)
  )
)
```

Architektura byla zvolena experimentálně, přičemž se vycházelo z těchto principů. Vstupní vrstva má 637 neuronů, neboť tak velký je vektor stavu hry a výstupní pak 4, neboť tolik máme tříd (hráčů). Zakončení musí být aktivační funkce softmax, neboť ta nám dá pravděpodobnost, že se jedná o danou třídu, což chceme. Za aktivace vnitřních vrstev se vzaly ReLU jako standard, který fungoval dobře, a tak se i ponechal. Pro zabránění přetrénování byly použity dropout vrstvy, u kterých je v praxi běžně užívaná hodnota 0.5 a i nám se síť učila lépe s touto hodnotou, než s nižšími.

Síť je trénovaná ve 128 epochách s velikostí batch 32, loss funkcí CrossEntropy a SGD optimalizátorem. Zkoušen byl ještě Adam, ten však na SGD ztrácel v řádech vyšších jednotek procent přesnosti. Při trénování bylo experimentováno s několika hodnotami learning rate, přičemž nejpřesnějších výsledků dosahoval 0,001. Nicméně, trénování bylo příliš pomalé. Z toho důvodu je prvních 24 epoch trénováno s lr=0.01, což síť velmi rychle katapultuje k 90% přesnosti, poté se sníží na hodnotu 0.005 až do 64. epochy, kde se dostáváme k přesnosti přesahující 95% a dále až do konce je využíván lr=0.001, se kterým se síť podařilo natrénovat na přesnost 98.07% pro trénovací data a 97.02% pro validační data, s čímž jsme se již spokojili.

Trénování je spuštěno skriptem model_training.py a síť reprezentuje třída NeuralNet, která se nachází v dicewars/ai/xklism00/neural_net.py, neboť k ní potřebuje mít přístup i agent při načítání modelu. K trénování je využíván modul torch1.9.1 a model je uložen tímto modulem ve formátu .zip do složky supp-xklism00. Odtud si ho lze překopírovat do modulu agenta.

Vyhodnocování a výsledky

K výše zmíněné heuristice ani strategii hry jsme se nedostali okamžitě. Po dokončení implementace prohledávání stavového prostoru jsme čistě pro zajímavost, jak to funguje, zkusili za heuristiku použít jednoduše jen počet vlastněných polí. Pro 100 her v turnaji nám to získalo následující winrate:

```
.                 .     % winrate [ .  / .   ] dt.stei kb.sdc_pre_at kb.stei_adt kb.stei_at kb.stei_dt xklism00.xklism00
kb.stei_adt       50.00 % winrate [ 30 / 60  ] 60.7/28 37.5/32       50.0/60     55.0/20    50.0/40    50.0/60
kb.stei_at        37.50 % winrate [ 21 / 56  ] 50.0/36 39.3/28       15.0/20     37.5/56    35.7/28    37.5/56
kb.sdc_pre_at     30.00 % winrate [ 18 / 60  ] 17.9/28 30.0/60       37.5/32     21.4/28    40.6/32    30.0/60
xklism00.xklism00 24.00 % winrate [ 24 / 100 ] 21.7/60 26.7/60       21.7/60     26.8/56    23.4/64    24.0/100
dt.stei           8.33  % winrate [ 5  / 60  ] 8.3/60  7.1/28       3.6/28      8.3/36     14.3/28    8.3/60
kb.stei_dt        3.12  % winrate [ 2  / 64  ] 7.1/28  3.1/32       2.5/40      0.0/28     3.1/64     3.1/64
```

I při jiných spuštěních se winrate držel mezi 20-25% a umísťovali jsme se na 4. pozici. Než jsme dospěli k použití neuronové sítě, chvíli jsme se trápili u hledání vhodných parametrů ohodnocujících hru, které bychom potenciálně nějak mohli natrénovat. Úspěch však nepřišel, a proto jsme zvolili cestu popsanou výše, která fungovala působivě, a proto jsme se jí již drželi.

Dále jsme zkoušeli různé strategie. Zejména jsme zkoumali 2 cesty. První byla na počátku posílit hranice, poté útočit a na konci opět posílit hranice. Druhou byla zvolená cesty evakuace namísto posílení v závěrečné fázi. U obou se zkoušely možnosti ponechat si na konec 0-3 transfery.

U prvního typu strategie jsme se však pravidelně, a to s libovolným ponecháním transferů na závěr kola, umísťovali na 3. pozici s pravděpodobností výhry pohybující se okolo 33%. To sice není úplně špatné, nicméně evakuační strategie byla značně úspěšnější. Výjimkou u první strategie bylo ponechání 0 transferů na konec, což nás posouvalo na 2. příčku a moc se nelišilo od 0 transferů na konec u evakuační strategie, kde rozdíl byl jen v posílení vs. evakuaci nevyužitými transfery.

Evakuační strategie je již mnohem úspěšnější. Hned na začátku jsme se zbavili konfigurace 3 transfery na začátku a 3 na konci, neboť toto znamenalo opět 3. příčku a jen cca 28% šanci na výhru. Zbyly tedy 3 možnosti (4-2 jako kb.stei_adt, 5-1, 6-0 jako kb.stei_at). Všechny se pravidelně v turnajích o stovkách her dělily o 1.-2. místo s winrate kolem 38% právě s nepřátelskými agenty kb.stei_adt a kb.stei_at. Konfigurace 6-0 však občas propadla i na 3. příčku. Rozhodli jsme se pro každou možnost maximalizovat přesnost winrate spuštěním turnajů o 250 hracích plochách (1000 her).

<u>Strategie 6-0:</u>

```
.                 .     % winrate [ .   / .    ] dt.stei  kb.sdc_pre_at kb.stei_adt kb.stei_at kb.stei_dt xklism00.xklism00
kb.stei_adt       38.17 % winrate [ 229 / 600  ] 37.2/312 41.0/288      38.2/600    33.9/304   40.9/296   38.2/600
xklism00.xklism00 37.10 % winrate [ 371 / 1000 ] 40.1/604 38.5/600      34.2/600    33.0/600   39.8/596   37.1/1000
kb.stei_at        34.67 % winrate [ 208 / 600  ] 40.1/292 33.8/296      28.3/304    34.7/600   36.7/308   34.7/600
kb.sdc_pre_at     17.17 % winrate [ 103 / 600  ] 19.6/316 17.2/600      15.6/288    16.6/296   16.7/300   17.2/600
kb.stei_dt        9.23  % winrate [ 55  / 596  ] 11.8/288 9.7/300       7.8/296     7.8/308    9.2/596    9.2/596
dt.stei           4.80  % winrate [ 29  / 604  ] 4.8/604  5.7/316       2.9/312     4.5/292    6.2/288    4.8/604
```

<u>Strategie 5-1:</u>

```
.                 .     % winrate [ .   / .    ] dt.stei  kb.sdc_pre_at kb.stei_adt kb.stei_at kb.stei_dt xklism00.xklism00
xklism00.xklism00 39.20 % winrate [ 392 / 1000 ] 42.4/604 38.8/600      34.1/596    36.1/604   44.6/596   39.2/1000
kb.stei_adt       37.75 % winrate [ 225 / 596  ] 44.5/292 37.0/308      37.8/596    33.2/304   36.5/288   37.8/596
kb.stei_at        32.95 % winrate [ 199 / 604  ] 34.0/312 32.1/296      32.6/304    32.9/604   33.1/296   32.9/604
kb.sdc_pre_at     20.50 % winrate [ 123 / 600  ] 19.9/296 20.5/600      16.9/308    20.6/296   24.7/300   20.5/600
kb.stei_dt        6.88  % winrate [ 41  / 596  ] 10.7/308 7.7/300       3.5/288     5.4/296    6.9/596    6.9/596
dt.stei           2.98  % winrate [ 18  / 604  ] 3.0/604  3.7/296       1.7/292     2.9/312    3.6/308    3.0/604
```

Strategie 4-2:

.	.	% winrate	[. / .]	dt.stei	kb.sdc_pre_at	kb.stei_adt	kb.stei_at	kb.stei_dt	xklism00.xklism00
kb.stei_at	38.26	% winrate	[228 / 596]	41.9/320	39.6/308	30.4/276	38.3/596	40.3/288	38.3/596
xklism00.xklism00	37.80	% winrate	[378 / 1000]	40.7/600	37.2/608	35.7/600	34.2/596	41.3/596	37.8/1000
kb.stei_adt	35.17	% winrate	[211 / 600]	38.9/296	32.0/328	35.2/600	31.9/276	38.0/300	35.2/600
kb.sdc_pre_at	18.59	% winrate	[113 / 608]	21.1/280	18.6/608	18.6/328	15.3/308	19.7/300	18.6/608
kb.stei_dt	7.55	% winrate	[45 / 596]	7.6/304	8.3/300	7.0/300	7.3/288	7.6/596	7.6/596
dt.stei	3.83	% winrate	[23 / 600]	3.8/600	6.1/280	2.4/296	2.5/320	4.6/304	3.8/600

Jak lze vidět, strategie 6-0, kdy si nenecháváme žádný transfer na evakuace skončila na druhé pozici. Vyloučení této strategie dále podporuje skutečnost, že u jiných menších turnajů se umisťovala na 1.-3. pozici, zatímco strategie 5-1 ani 4-2 třetí nebyly nikdy. Strategie 4-2 podpořila své pravidelné dělení o 1.-2. místo tím, že skončila velmi těsně druhá. Strategie 5-1 nejen že v předběžných testech (turnajích o méně hrách) dosahovala podobných výsledků jako 4-2, tedy dělba o 1.-2. místo, tak i ve velkém turnaji, kde se pravděpodobnost projevuje přesněji, jako jediná dosáhla prvního místa, a to dokonce s náskokem téměř 1.5% na druhého. Byl to i náš předběžný favorit, neboť v debugu jsme pozorovali, že se nejčastěji provádí jen 1 evakuační transfer, tedy mít našetřené 2 v mnoha případech jen znemožňuje využití plného počtu transferů. Pro podpoření zvolené strategie a závěrečné vyhodnocení našeho agenta byl spuštěn ještě větší turnaj o 2000 hrách s následujícím výsledkem.

.	.	% winrate	[. / .]	dt.stei	kb.sdc_pre_at	kb.stei_adt	kb.stei_at	kb.stei_dt	xklism00.xklism00
xklism00.xklism00	38.50	% winrate	[770 / 2000]	42.0/1204	39.2/1196	35.2/1204	35.4/1200	40.6/1196	38.5/2000
kb.stei_adt	36.88	% winrate	[444 / 1204]	39.7/652	36.0/580	36.9/1204	30.3/584	41.0/592	36.9/1204
kb.stei_at	35.75	% winrate	[429 / 1200]	37.8/564	36.2/644	31.0/584	35.8/1200	38.0/608	35.8/1200
kb.sdc_pre_at	18.39	% winrate	[220 / 1196]	20.9/584	18.4/1196	15.3/580	17.2/644	20.2/584	18.4/1196
kb.stei_dt	6.52	% winrate	[78 / 1196]	7.7/608	6.3/584	6.1/592	5.9/608	6.5/1196	6.5/1196
dt.stei	4.40	% winrate	[53 / 1204]	4.4/1204	4.5/584	3.8/652	3.2/564	6.1/608	4.4/1204

Agent opět vyhrál a znovu s náskokem kolem 1.5% na druhého jako v předchozím turnaji. Lze z toho tedy vyvodit, že i když občas při menších turnajích skončí až druhý, v obecnosti má větší pravděpodobnost skončit první a jedná se proto o nejsilnějšího z agentů.

Struktura adresáře

- */xklism00* - Modul agenta, patří do adresáře dicewars/ai.
  - *__init__.py* - Umožňuje import AI agenta pro hru a Utils pro sběr dat.
  - *heuristic.py* - Obsahuje třídu pro výpočet heuristiky na základě natrénovaného modelu.
  - *model.zip* - Natrénovaný model neuronové sítě.
  - *neural_net.py* - Obsahuje třídu reprezentující neuronovou síť pro modul torch.
  - *utils.py* - Obsahuje jmenný prostor Utils s pomocnými statickými metodami.
  - *xklism00* - Třída AI agenta.

- */supp-xklism00* - Adresář obsahující skripty pro sběr dat a trénování modelu.
  - *data_collector.py* - Sbírá a ukládá data o stavech a výsledku hry. Patří do adresáře dicewars/server.
  - *data_prepatation.py* - Skript transformující nasbíraná data na trénovací a validační datasety.
  - *game.py* - Modifikovaný původní soubor dicewars/server/game.py o přidané 4 řádky kódu sbírající data třídou DataCollector.
  - *model_training.py* - Skript trénující neuronovou síť z neural_net.py s využitím dat připravených od data_preparation.py.

- *xklism00.pdf* - Tento dokumentační soubor.

Zvláštní požadavky

1) Pro běh agenta je mimo základní moduly z requirements.txt zapotřebí torch1.9.1.
2) Pro sběr dat (nikoliv však už pro běh agenta) je třeba přidat modul pandas.