Vysoké učení technické v Brně
Fakulta informačních technologií

SUI – Dicewars

Filip Bali (xbalif00)
Natália Holková (xholko02)
Roland Žitný (xzitny01)
Vít Barták (xbarta47)

1. januára 2022

Obsah

1 Úvod **2**

2 Správanie AI a prehľadávanie stavového priestoru **2**

3 Presuny kociek medzi poliami **2**

4 Strojové učenie **3**
4.1 Serializácia stavu hry 3
4.2 Dataset hier 3
4.3 Architektúra neurónovej siete 4
4.4 Učenie 4
4.5 Predikovanie a integrácia 4

5 Testovanie **5**

6 Záver **6**

1 Úvod

Hra Dicewars[1] je strategická hra s volne dostupnými informáciami a prvkom náhody, kedy víťazným stavom hry je plné ovládnutie celej hernej plochy, ktorá pozostáva z jednotlivých polí, na ktorých je jedna až osem kociek. Prevažne štyria hráči sa v presne danom poradí striedajú v ťahoch, ktoré pozostávajú z neobmedzeného počtu ťahov, ktoré môžu byť presun, útok alebo ukončenie ťahu.

Cieľom tejto práci bolo vytvoriť dostatočne silnú umelú inteligenciu (AI) pre napodobnenie správania skúseného hráča tejto hry, ktorá využíva prehľadávanie stavového priestoru s pomocou strojového učenia.

2 Správanie AI a prehľadávanie stavového priestoru

Chovanie a teda ťah AI začína volaním metódy `ai_turn(...)`, kedy pracujeme s informáciami o aktuálnom stave hernej plochy, počte vykonaných presunov kociek v danom kole a o aktuálnom type pohybu, teda útok alebo presun kociek. Ak je aktuálnym typom pohybu útok, tak sa získa zoznam všetkých možných útokov v danej situácií pomocou metódy `possible_attacks(board, player_name)`. Z pomedzí týchto útokov je následne nutné vybrať jeden najlepší ktorý sa vykoná. Tento výber prebieha v metódy `choose_best_attack(attacks)`, ktorá vytvára vlastný zoznam položiek pozostávajúcich z dvojice útok a jeho ohodnotenie, kedy vyberá útok s naväčším ohodnotením. Takáto položka vzniká vo volanej metóde `evaluate_attack(attack)`, kedy sa prehľadáva stavový priestor a vyhodnocované sú len tie útoky, ktorých počet kociek útočiaceho poľa prevyšuje, alebo je rovný s počtom kociek cieleného poľa. Týmto spôsobom zabránime nízko pravdepodobným útokom, kedy AI radšej počká na nárast kociek v danom poli a tým zvýši pravdepodobnosť úspechu tohto útoku a teda ukončí ťah. Ohodnocovanie útokov, ktoré spĺňajú takúto podmienku prebieha pomocou simulácií. To znamená že v počiatku metódy `evaluate_attack(attack)` sa vytvorí hlboká kópia celej hracej plochy, na ktorej sa odsimuluje aktuálne ohodnocovaný útok. Keďže vieme že je presne stanovené poradie jednotlivých hráčov tak sa na tejto kópii hracej plochy vykonajú simulácie útokov jednotlivých protivníkov, kedy sa vyberá jeden najsilnejší útok na základe rozdielu počtu kociek medzi protivníkovym útočiacim poľom a jeho cieľovým. Týmito úkonmi nám vznikne stav celej hracej plochy po aplikovaní týchto útokov. Tento stav hracej plochy sa následne musí vyhodnotiť zavolaním metódy `evaluate_board_NN(...)`, kedy na získanie finálnych hodnôt využívame strojové učenie popísané v sekcií 4. Vďaka tejto metóde získame hodnotu, ktorá ohodnocuje potenciál vedúci k výhre s využitím daného útoku a vzniká nám dvojica `[útok, ohodnotenie]`, čo je položka v zozname vytváranom v metóde `choose_best_attack(attacks)`.

V prípade kedy AI nemôže vykonať žiaden vhodný útok a má sa vykonávať pohyb typu útok, tak AI vyhodnotí takúto situáciu ako konečnú a ukončí kolo.

3 Presuny kociek medzi poliami

Samotná schopnosť prehľadávania stavového priestoru a teda výber najvhodnejších útokov nieje dostačujúca a je nutné taktiež využiť ďalší aspekt hry Dicewars, čo sú presuny kociek medzi vlastnenými hracími poliami.

AI sleduje dva prípady presúvania kociek a to na začiatku každého kola alebo presuny v prípade náročnej situácie, kedy je vhodnejšie sa stiahnuť a teda presunúť kocky smerom do bezpečia.

Vždy na začiatku volanej metódy `ai_turn(...)` a pred vykonávaním akcií popísaných v sekcii 2, sa sleduje celkový počet vykonaných presunov kociek v danom kole, ktorý je obmedzený. V prípade, kedy je tento počet menší ako 3/4 povoleného počtu presunov, vykonávame toľko presunov smerom do predku útočnej línie pokiaľ tento počet nedosiahne daný limit. Tento počet možných presunov vedúcich k agresívnejšiemu správaniu AI je obmedzený na 3/4 preto, aby bol uchovaný dostatočný priestor pre náročné situácie, kedy nebude možné vykonať žiadne útoky a teda nastane etapa evakuácie. Táto etapa správania AI nastáva v tých prípadoch popísaných v sekcií 2, kedy zoznam položiek vytváraný v metóde `choose_best_attack(attacks)` je prázdny. V tomto momente nastáva presun kociek z polí ktoré sú v nebezpečí späť a teda na polia, ktoré sú hlbšie vo vlastnenom regióne až kým nevyčerpáme celkový počet možných presunov za kolo.

[1] `https://github.com/ibenes/dicewars`

4 Strojové učenie

V projekte využívame strojové učenie na **predikciu ohodnotenia stavu hracej plochy** v danom momente. Toto ohodnotenie sa využíva ako heuristická funkcia pri prehľadávaní stavového priestoru. Konkrétne implementujeme metódu *učenia s učiteľom*, ktorá sa učí trénovacích dát.

4.1 Serializácia stavu hry

Prvým krokom pri tvorbe trénovacieho datasetu, na ktorom sa neskôr bude učiť naša neurónová sieť, je vymedzenie vlastností hracej plochy, ktoré sú dôležité pri určovaní jej ohodnotenia. Neurónová sieť by nebola schopná naučiť sa zmysluplne predikovať na niečom, čo je úplne bezvýznamné vzhľadom k úlohe, ktorú má riešiť. Dospeli sme k záveru, že podstatné vlastnosti hracej plochy sú:

- informácie o tom, ako spolu políčka na hracej ploche susedia
- kto je vlastníkom každého políčka a koľko sa na ňom nachádza kociek
- aká je veľkosť najväčšieho regiónu každého hráča

Implementácia nástroja na serializáciu hry, ktorú sme použili pri tvorbe datasetu, bola veľmi podobná triede `Serializer` v súbore `xholko02/ml.py`. Rozdiel spočíval v tom, že preberala definície `Board` z modulu `server.board` namiesto `client.game.board`. Táto implementácia bola dodaná v priečinku `supp-xholko02\dataset_creation`. Pre replikáciu tvorby datasetu treba nahradiť súbory v pôvodnej implementácii za tie priložené v priečinkoch `dicewars` a `scripts`. Pre spustenie zberu dát je nutné spustiť príkaz:

```
python3 ./scripts/dicewars-ai-only.py --ai [zoznam AI] -n 10 --nby [uloženie datasetu]
```

Stavy každej hry sa ukladajú do samostatného súboru a pre zlúčenie do jediného súboru bol pripravený skript `supp-xholko02\dataset_creation\dataset_compiler.py`.

Informácia o susednosti je prvotne reprezentovaná ako matica $N \times N$, kde N je celkový počet políčok (default 34). Z hľadiska optimalizácie veľkosti serializácie stavu hry nie je potrebné udržovať celú maticu, ale postačí iba horný trojuholník nad diagonálou. Navyše, políčka nemenia svojich susedov počas hry, takže je nutné spraviť ich výpočet iba raz na začiatku hry. Toto prebieha počas inicializácie triedy vo funkcii `get_neighbours_matrix()`, ktorá vráti celú maticu a funkcia `get_matrix_upper_triangle()` vyrobí horný trojuholník reprezentovaný ako 1D pole.

Informácie o jednotlivých políčkach sú reprezentované ako N dvojíc v tvare (v, k), kde v je číslo vlastníka a k počet kociek. Získanie týchto informácií rieši funkcia `get_array_of_area_info()`.

Veľkosti najväčších regiónov jednotlivých hráčov sú pole P čísel, kde P je počet hráčov. Rieši to funkcia `get_array_of_biggest_regions()`. Nakoniec je všetko spojené do jedného poľa.

4.2 Dataset hier

Nakoľko sme používali metódu učenia s učiteľom, každý prvok datasetu musí obsahovať 2 veci: *vstupné dáta* a *predpokladanú výstupnú hodnotu*. Vstupnými dátami je seriazácia stavu hry ako je bližšie popísané v sekcii 4.1. Výstupnou hodnotou je víťaz danej hry, teda poradové číslo hráča.

Dataset sme vytvárali iba z hier 4 hráčov, z čoho vyplýva, že strojové učenie je možné použiť iba pri hre 4 hráčov. Je možno ľahko obdobne implementovať strojové učenie aj pri inom počte hráčov, no vyžadovalo by to samostatné modely trénované na datasetoch o danom počte.

Dôležitým rozhodnutím bolo, ktorých predpripravených botov využiť pri tvorbe datasetu. Riadili sme sa myšlienkou, že príliš jednoduchý boti by nám nepomohli proti skúseným súperom. Prvotný skúšobný dataset bol vytvorený na botov `dt.stei` a finálny model bol vytvorený na `kb.sdc_pre_at`. V hrách sme proti sebe postavili vždy štyroch rovnakých botoch z dôvodu, že pri reálnom hraní nemáme istotu, v akom poradí bude začínať naše AI, preto musia byť boti rovnocenní.

Celková veľkosť finálneho datasetu bola **71 310** vstupných dát (rôznych stavov hier). Pri testovaní sme zistili, že toto bola dostatočne postačujúca veľkosť. Dataset bol spojený do jedného súboru `dataset.npy`, ktorý bol následne komprimovaný. V odovzdanom archíve prikladáme iba časť tohto datasetu (10 000 vstupných dát) v archíve `dataset_sample.zip`.

4.3 Architektúra neurónovej siete

Problém predikcie ohodnotenia stavu hry sme poňali ako *klasifikačný problém*. Model sa bude snažiť predikovať do akej triedy (kto bude víťazom) bude patriť daný vstup (serializovaný stav hry).

Pri implementácii neurónovej siete sme využili knižnicu **Pytorch**. Definícia siete a metódy spojené s jej učením sa nachádzajú v súbore `supp-xholko02/nn.py`. Architektúra je neskôr znovu definovaná v súbore `xholko02/ml.py`, aby nedochádzalo k importovaniu z priečinku `supp-xholko02`.

Neurónovú sieť predstavuje trieda `Network` v súbore `supp-xholko02/nn.py`. Zvolili sme použiť iba jednoduchú architektúru. Za vstupnou vrstvou nasledovali dva *plne prepojené vrstvy* (`Linear`), na ktoré je aplikovaná *ReLU* aktivačná funkcia. Za každou lineárnou vrstvou nasledovala *normalizačná vrstva* (`BatchNorm1d`), ktorou úlohou bolo pomôcť predísť pretrénovaniu. Nakoniec nasleduje posledná výstupná vrstva. Zaujímavou vlastnosťou knižnice Pytorch je, že aktivačnú funkciu *softmax* na výstupnú vrstvu je nutné aplikovať až pri predikovaní a nie počas učenia.

Nižšie je možné podrobnejšie vidieť architektúru našeho modelu:

```
Network(
  (fc1): Linear(in_features=633, out_features=256, bias=True)
  (b1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Linear(in_features=256, out_features=64, bias=True)
  (b2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc4): Linear(in_features=64, out_features=4, bias=True)
)
```

4.4 Učenie

Všetky triedy a metódy spojené s učením neurónovej siete sa nachádzajú v súbore `supp-xholko02/nn.py`. Bolo nutné vytvoriť vlastnú implementáciu triedy `Dataset` a to `CustomDataset`, ktorá pri inicializácii rozdelí každý prvok datasetu na serializovaný stav hry a víťaza. K tomuto bola ďalej definovaná funkcia `get_train_val_loader()`, ktorá rodelý dataset na trénovaciu a validačnú časť vo zvolenom pomere. Pri našich experimentoch sme využívali hodnotu $val\_split = 0.8$, teda trénovacia časť tvorí 80%.

V metóde `run_gradient_descent()` prebieha samotné učenie. Nakoľko riešime klasifikačnú úlohu s viacerými triedami, používame ako objektívnu (*loss*) funkciu `CrossEntropyLoss()`. Ako algoritmus učenia využívame klasický *stochastic gradient descent* s $learningrate = 0.01$. Model trénujeme po dobu niekoľko epóch, v prípade finálneho modelu $epochs = 3$.

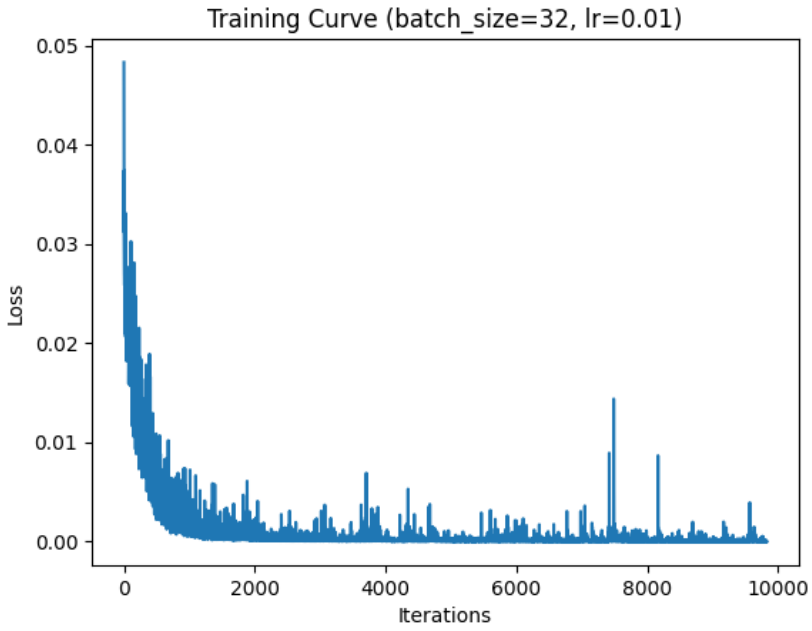
Po skončení učenia sa ukladá model na zvolené miesto.

Na obrázku 1 je možné vidieť priebeh *loss* funkcie počas trénovania a na obrázku 4 je možné vidieť vývoj presnosti nášho modelu.

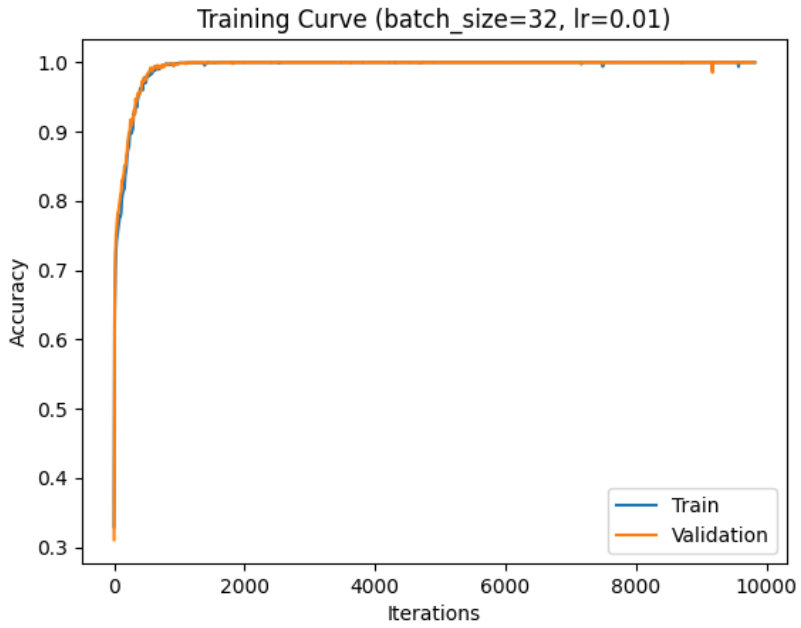
4.5 Predikovanie a integrácia

Natrénovaný model je uložený v `xholko02/model.pt`. Počet inicializácie AI si vytvorí vlastnú kópiu modelu so štruktúrou definovanou v `Network`, do ktorej načíta naučené váhy pomocou `load_state_dict()`. Je nutné prepnúť model do režimu predikovania cez `eval()`. Zároveň je vytvorený objekt zodpovedný za serializáciu hry.

AI využíva strojové učenie iba ak sa jedná o hru 4 hráčov. V tom prípade sa používa funkcia `evaluate_board_NN()` namiesto `evaluate_board()`. Táto funkcia pošle modelu stavy hry a ako výstup dostane pravdepodobnosti víťaztva jednotlivých hráčov. Až v tomto momente sa aplikuje *softmax* aktivačná funkcia, aby sme získali pravdepodobnosti z intervalu $< 0, 1 >$. Na základe toho, koľká v poradí je naša AI sa vráti príslušná pravdepodobnosť jej víťazstva.



Obr. 1: Loss funkcia v priebehu trénovania modelu



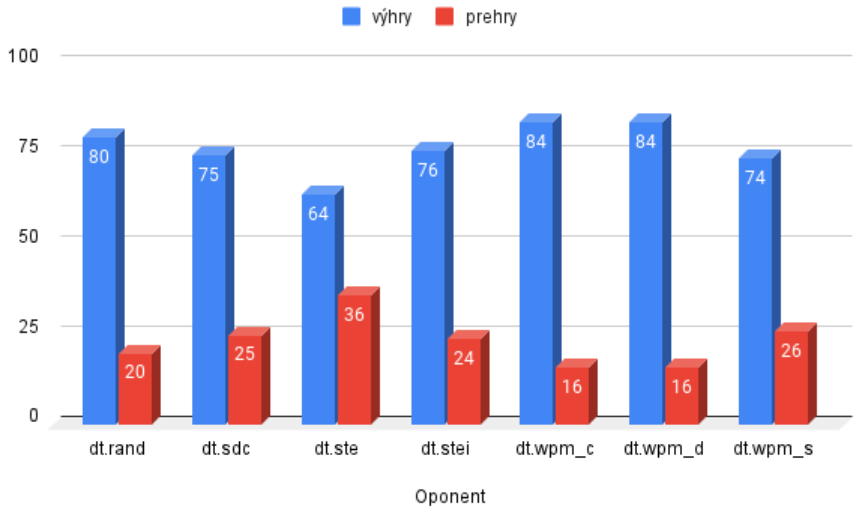
Obr. 2: Presnosť modelu v priebehu trénovania

5 Testovanie

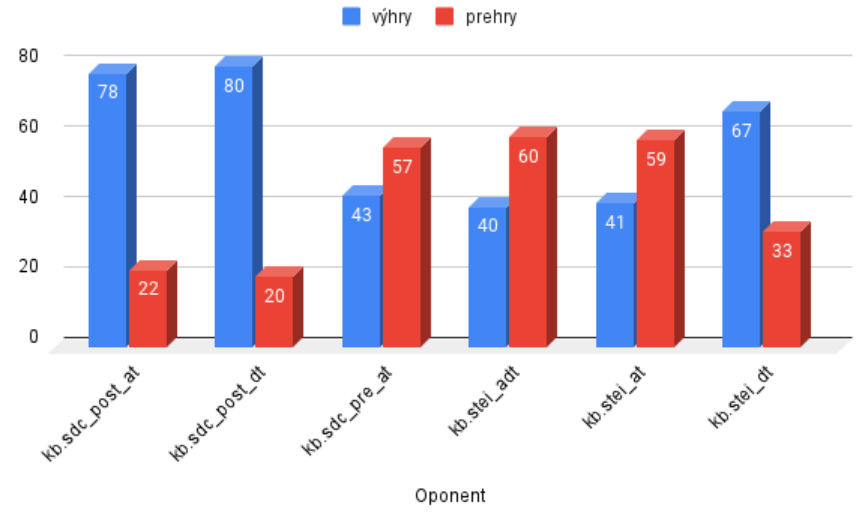
Pre správne ohodnotenie efektivity vytvoreného AI bolo vykonaných niekoľko testov, kde sa sledoval počet vyhratých a prehratých hier. Tieto testy boli spúšťané ako 100 hier štyroch hráčov, kedy jedného hráča predstavuje naše vytvorené AI a ostatný traja predstavujú oponenta rovnakého druhu, teda jednu z 13 predom vytvorených umelých inteligencií rôznych kvalít. Z takto vykonaných testov je vidno že predstavovaná umelá inteligencia je schopná porážať väčšinu oponentov s miernym oslabením voči `kb.sdc_pre_at`, `kb.stei_adt` a `kb.stei_at`.

Priemerná úspešnosť AI je teda 68%.

Pri hodnotách kedy počet výhier našeho AI oproti oponentom nedosahuje 50 a viac výhier, je nutné brať v ohľad to, že počet prehier je súčtom výhier všetkých troch oponentov, to znamená že stále dosahujeme lepšie výsledky a teda porážame daného oponenta pri pohľade na súboj dvoch hráčov hry. Teda napríklad v hre proti oponentovi `kb.stei_at` bol počet výhier našeho AI rovný 43 ale počet výhier prvého oponenta bol 16, druhého 22 a posledného 19. Tento fakt teda naznačuje že sme stále schopný súperiť s týmto typom umelého oponenta.



Obr. 3: Pomer výhier a prehier proti prvej polovici oponentov



Obr. 4: Pomer výhier a prehier proti druhej polovici oponentov

6 Záver

Pri vytváraní predstaveného prístupu bolo experimentované s viacerými možnosťami prehľadávania stavového priestoru, kedy sa sledoval vyšší počet vykonávaných útokov zo strany protivníkov z čoho vzišli približne rovnaké alebo slabšie výsledky v rámci pomeru výhier a prehier, a preto sme pristúpili na voľbu so sledovaním jedného útoku od každého protivníka. Takáto voľba predstavuje efektívny prístup z pohľadu sily vytváranej umelej inteligencie a jej časovej náročnosti.

Zapojenie vykonávania presunov kociek medzi vlastnenými hracími poliami malo marginálny efekt na silu AI, kedy sme zaznamenali nárast výhier približne o 15% pri presune kociek do popredia útočnej línie a zabezpečenie voľných posunov pre evakuáciu taktiež pomohol v prípade agresívnejších protivníkoch.

V prípade ohodnocovania potenciálneho zisku cieleného k víťazstvu pri jednotlivých útokoch sme využívali prístup priamej heuristiky z čoho bola vo finále zhotovená neurónová sieť. Vďaka dobre natrénovanej neurónovej sieti sme dosiahli efektívny spôsob ohodnocovania možných útokov a teda aj výber najvhodnejšieho, vďaka čomu sme zabezpečili konzistentné výsledky proti viacerým druhom protivníkov teda agresívnejším a pasívnejším.

Vytvorená umelá inteligencia vo finále predstavuje schopné AI, ktoré v pomeroch na výhry a prehry zvláda porážať valnú väčšinu testovaných umelých protivníkov. Najnižší percentuálny počet výhier bol zaznamenaný proti protivníkovi `kb.stei_adt` a to 40% a najvyšší percentuálny počet výher bol zaznamenaný 84% pri `dt.wpm_c` a `dt.wpm.d`. Tieto výsledky sú zhotovené na hrách o štyroch hráčoch a teda moment dosiahnutia menších hodnôt ako 50% stále znamená že daného oponenta porážame v pohľade na hru o dvoch hráčoch.

Možné rozšírenie

Možným rozšírením je pridanie adaptívneho spôsobu správania AI, kedy by sa prechádzalo na agresívnejší alebo pasívnejší štýl hry na základe jej odvíjania. Takáto zmena správania by sa mohla vykonávať na základe počtu vykonaných kôl v hre alebo veľkosti regiónov hráča, kedy by sa umožnilo vykonávať viac riskantné respektíve istejšie útoky. Na rovnakom princípe je možné upravovať spôsob vykonávania presunov kociek, alebo meniť model neurónovej siete pre vyhodnocovanie útokov.