Sztuczna inteligencja
Pracownia 4
Terminy: zajęcia 10 i 12[1]

TLDR: masz zaimplementować agentów grających w gry Reversi, Dżungla (https://en.wikipedia.org/wiki/Jungle_(board_game)) oraz Szachy (przy czym dokładnie jest opisane, jak można sobie tę implementację ułatwiać). Lista jest dość obszerna, można o tym myśleć, że dwie z trzech gier są „obowiązkowe" (ale oczywiście możesz zrobić dowolną liczbę zadań).

Do wszystkich 3 gier będą zdefiniowane „bossy" (czyli programy, za których pokonanie będą dodatkowe punkty). Uwaga do szachów: można stosować bibliotekę python-chess (lub inną, analogiczną), która daje generator ruchów, estetyczną wizualizację, sprawdzanie, czy jest mat, itd. Szachy można oddawać również na pierwszej pracowni w czerwcu.

Dodatkowo przeprowadzony zostanie turniej dla programów grających w te gry (również z nagrodami punktowo/egzaminacyjnymi).

Zadanie 1. (4-6 p) W zadaniu tym powinieneś napisać agenta, grającego w Reversi (`https://pl.wikipedia.org/wik`
Agent, z którym ma pojedynkować się Twój program, będzie agentem grającym losowo (tzn. w każdej sytuacji z dostępnych ruchów ma wybrać 1, przypisując każdemu z nich to samo prawdopodobieństwo). W tym zadaniu powinieneś również tego agenta zaimplementować samodzielnie.

Poziom agenta, wymagany do zaliczenia tego zadania jest następujący:

1. Poziom **novice**: na 1000 gier co najwyżej 200 porażek (czas działania całego eksperymentu mniej niż 1 minuta)
2. Poziom **basic**: na 1000 gier co najwyżej 70 porażek (czas działania całego eksperymentu mniej niż 1 minuta)
3. Poziom **standard**: na 1000 gier co najwyżej 20 porażek (czas działania poniżej **7 minut** – czas został złagodzony, choć wersja poprzednia, czyli 3 minuty, powinna być osiągalna).

Na poziomie *nowicjusz* zadanie warte jest 4 punkty, każdy kolejny poziom dodaje 1 punkt. Dodatkowo 1 punkt można otrzymać za zaimplementowanie algorytmu $\alpha - \beta$ Search i sprawdzenie, jaka jest różnica czasów pomiędzy losową kolejnością ruchów, a kolejnością wyliczoną zgodnie z funkcją oceniającą planszę[2].

Jeżeli chcesz, możesz korzystać z programu `reversi_show.py`, który przeprowadza losowe rozgrywki.

Zadanie 2. (1+ p) Dostosuj Reversi do dueler.py, przeprowadź pojedynek z 'bossem' reversi_random.py i wgnieć go w ziemię[3]. Za każdego dodatkowego pokonanego bossa otrzymasz 1 punkt (przy czym tu zwycięstwo definiujemy jako 'statystycznie lepszy z progiem p=0.95', tabelka zależności koniecznej liczby zwycięstw w zależności od przeprowadzonych partii pojawi się na stronie wykładu). Wkrótce pojawią się bossy za więcej punktów. Dodatkowo zakładamy, że remis to 'pół zwycięstwa'.

W zadaniu do maksimum wlicza się 1 punkt.

Dżungla. Opis gry

Dżungla (inne nazwy: Animal Chess albo Dou Shou Qi) jest prostą grą dla dwóch graczy, rozgrywaną na planszy 7×9, zawierającej 4 rodzaje pól: łąki (`.`), pułapki (`#`), jamy (`*`) oraz stawy (`~`). Plansza wygląda następująco:

```
..#*#..
...#...
.......
```

[1] Na zajęciach 11 będą ćwiczenia

[2] Uwaga: sortowanie ruchów ma szansę byc opłacalne raczej blisko korzenia, niż liści

[3] Osiągnij co najmniej 80% zwycięstw. To nie powinno być trudne, bo masz około 1s na ruch, inaczej niż w zadaniu poprzednim

```
.~~.~~.
.~~.~~.
.~~.~~.
.......
...#...
..#*#..
```

Każdy z graczy początkowo dysponuje zestawem 8 następujących bierek: szczur (R), kot (C), pies (D), wilk (W), pantera (J), tygrys (T), lew (L), słoń (E). Kolejność w poprzednim zdaniu definiuje również starszeństwo bierek.

Początkowo bierki ustawione są w poniższy sposób:

```
L.....T
.D...C.
R.J.W.E
.......
.......
.......
e.w.j.r
.c...d.
t.....l
```

Obowiązują następujące zasady ruchów:

- Gracz nie może wchodzić do własnej jamy.
- Jedynie szczur może wchodzić do wody.
- Normalnym ruchem jest przesunięcie bierki na sąsiednie wolne pole, w kierunku góra, dół, lewo, lub prawo.
- Tygrys i lew mogą skakać przez stawy (aby wykonać skok bierka musi „tak jakby" wejść na staw i następnie poruszać się w tym samym kierunku aż do osiągnięcia pola niebędącego stawem). Nie wolno skakać nad wrogim szczurem.
- Wejście na pole zajęte przez inną bierkę jest równoważne z jej zbiciem. Można bić bierkę o równej sile, albo słabszą. Nie wolno wchodzić na pola z własną bierką lub z bierką silniejszą. Szczur (wbrew starszeństwu) jest silniejszy od słonia. Poza tym starszeństwo bierek odpowiada ich sile.
- Szczur nie może bić wykonując ruch z jeziora do lądu.
- Bierka znajdująca się w pułapce (jednym z pól otaczających jamę), traci całkowicie swoją siłę i może być zbita przez dowolną bierkę. Dotyczy to zarówno pułapek broniących dostępu do jamy gracza, jak i do jamy oponenta.
- Celem gry jest wejście bierką do jamy przeciwnika. Po takim ruchu gra się kończy i wygrywa gracz wchodzący do jamy.

Zadanie 3. (4p) Napisz agenta, który gra w Dżunglę. Powinien on wybierać ruch w następujący sposób:

1. Generuje w danym momencie wszystkie możliwe ruchy, każdy z nich wykonuje i uruchamia procedurę oceny sytuacji na planszy. Zbiór tak powstałych sytuacji nazwiemy S.
2. Oczywiście wybiera ruch, który daje najbardziej korzystną sytuację.
3. Ocenę sytuacji $s \in S$ agent przeprowadza wykonując w pełni losowe gry rozpoczynające się w s (przyjmijmy, że i-ta gra ma K_i ruchów)
4. Podczas całej analizy agent ma prawo zasymulować $N = \sum_i K_i$ ruchów. Wartość N rzędu 20000 powinna umożliwić w miarę komfortowe przeprowadzenie testów w zadaniu kolejnym. Dla takiego N na każdą sytuację ze zbioru S powinno przypaść po kilka partii.

5. Należy w miarę równomiernie rozdzielać partie testowe pomiędzy sytuacjami ze zbioru S.

Zadanie 4. (4p) Napisz agenta, który jednoznacznie[4] pokonuje agenta z poprzedniego zadania. Musi on realizować inny algorytm, w którym wykorzystywana jest jakaś heurystyczna funkcja oceniająca ruchy lub sytuację na planszy. Może działać do 4 razy wolniej.

Zadanie 5. (2p) $\star$ Masz zaimlementować grę w Dżunglę samodzielnie, nie korzystając ze sprawdzaczki[5].

Zadanie 6. (1+ p) Dostosuj Dżunglę do do dueler.py. Punktacja taka sama jak w analogicznym zadaniu o Reveersi. Uwaga: być może do Dżungli pojawi się gra na Coding Game. Wówczas zostaną ogłoszone dodatkowe zasady umożliwiające zdobycie dodatkowych punktów (zapisywanych w tym zadaniu)

W zadaniu do maksimum wlicza się 1 punkt.

Zadanie 7. (4+4p) $\star$ Napisz agenta, grającego w szachy. Do generacji ruchów wykorzystaj jakąś bibliotekę szachową (np. python-chess). Do zaliczenia zadania za 4p wymagane są jedynie dwie rzeczy:

a) Musisz napisać funkcję heurystyczną, oceniającą ruchy. Funkcja ta powinna oceniać **nie tylko** przewagę materialną. Funkcja powinna być wykorzystana (albo w MiniMax-ie, albo w MCTS – do symulacji, lub jak w AplhaZero[6])

b) Dołącz do agenta jakąś formę przeszukiwania drzewa gry (np. MCTS lub MiniMax)

Dodatkowe punkty są za:

- Wykorzystanie księgi otwarć (1p)
- Wykorzystanie bazy końcówek szachowych (1p)
- Dodatkowe elementy funkcji heurystycznej (0.5p za każdy element, max 2p). Przykładowo, student, w którego funkcji oceniającej będzie przewaga materialna, ruchliwość, ocena struktury pionów i szacowanie zagrożenia króla dostanie 1p (bo przewaga materialna i 1 dodatkowy element są wymagane, pozostałe dwa – dodatkowe)

Oczywiście nie wolno wywoływać zewnętrznych silników szachowych.

Zadanie 8. 1+ p($\star$) Dostosuj Szachy do do dueler.py. Uwaga: bossy w tym zadaniu (oprócz random_chess.py) nie są łatwe, bo 'oszukują' – korzystają z osłabionego, ale jednak, Stockfisha. W związku z tym:

a) Remis liczymy jako zwycięstwo

b) a prawdziwe zwycięstwo jako 3 zwycięstwa

a i tak za pokonanie (p=0.95, itd) bossów używających Stockfisha (borgov, beth_harmon oraz benny_watts) otrzymasz 2 punkty. Nawet, jeżeli nie pokonasz któregoś z nich (w sensie statystycznym), to zwycięstwo z nim w przynajmniej jednej partii daje 1p.

Zadanie 9. (5p) Wybierz jedną grę (Dżunglę, Szachy albo Reversi[7]). Zaimplementuj dla niej algorytm Monte Carlo Tree Search i następnie:

a) Dobierz parametry obu programów tak, żeby ruch trwał mniej więcej 0.5 sekundy.

b) Przeprowadź mecz 10 partii (każdy program zaczyna 5 razy), notując wyniki meczów.

Do eksperymentów możesz wykorzystać sprawdzaczkę. Jeżeli dla wybranej gry wykorzystałeś MCTS w którymś z poprzednich zadań, to w tym zadaniu powinieneś zaimplementować Alpha-Beta-Search.

[4] 10 parti wygrane w stosunku 8 do 2

[5] Oznacza to, że możesz z niej skorzystać w zadaniach poprzednich rezygnując z tego zadania

[6] Będzie o tym mowa w okolicy wykładu 12

[7] Możesz, po konsultacji z prowadzącym pracownię, wybrać inną grę. Prowadzący może uwzględnić, że jest to trudniejsza gra i zwiększyć liczbę punktów za to zadanie