Sztuczna inteligencja
Pracownia 4
(terminy zostaną podane po weekendzie majowym)

TLDR: masz zaimplementować agentów grających w gry Reversi, Dżungla (https://en.wikipedia.org/wiki/Jungle_(b oraz (z gwiazdką) Szachy.

Zadanie o szachach (które będzie opisane w osobnym dokumencie), jest tegoroczną nowością. 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 oraz jedną z dwóch pozostałych gier można oddawać również na pierwszych zajęciach P5.

Dodatkowo, w przypadku zainteresowania studentów, przeprowadzony zostanie turniej dla programów grających w te gry (również z nagrodami punktowymi).

Do listy dodatkowo pojawią się (również w osobnym dokumencie) informacje o dodatkowych testach dla Sokobana.

Zadanie 1. (4-7+ p) W zadaniu tym powinieneś napisać agenta, grającego w Reversi (mówiliśmy o tej grze na wykładzie[1], więcej informacji znajduje się na Wikipedii. 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).
4. Poziom **pro1** (i ewentualne dalsze): specyfikacja zostanie podana w późniejszym terminie

Na poziomie *nowicjusz* zadanie warte jest 3 punkty, każdy kolejny poziom dodaje 1 punkt.
Ponadto można otrzymać następujące punkty:

a) 1 punkt za zaimplementowanie algorytmu $\alpha - \beta$ Search i sprawdzenie, jaka jest różnica czasów pomiędzy losową kolejnością ruchów, a kolejnością wyliczoną zgodnie z funkcją oceniającą planszę.

b) 1 punkt za wykorzystanie funkcji oceny, która ma (co najmniej) 1 parametr, wyznaczany eksperymentalnie[2])

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

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] Wykład 8, początkowe slajdy

[2] Na wszelki wypadek napiszę, że działanie funkcji powinno zależeć od parametrów.

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. 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.
- Celem gry jest wejście bierką do jamy przeciwnika. Po takim ruchu gra się kończy i wygrywa gracz wchodzący do jamy.

Te tradycyjne zasady wymagają drobnego skomplikowania, żeby utrudnić trywialną grę na remis przez obudowywanie swojej jamy zasiekami „nie do przejścia"[3]. Nowa reguła brzmi następująco:

> Jeżeli przez kolejne 30 ruchów nie nastąpi bicie (albo wejście do jamy), wówczas gra się kończy i zwycięstwo ustala się:
>
> a) porównując starszeństwo bierek: wygrywa gracz, który ma najstarszą bierkę, nieposiadaną przez drugiego gracza,
>
> b) a jeżeli gracze mają dokładnie te same bierki, wówczas wygrywa gracz, który poruszał się jako drugi.

Zadanie 2. (4p) Napisz agenta, który gra w Dżunglę. Powinien on wybierać ruch w następujący sposób:

[3] Porównaj: http://www.chessvariants.com/other.dir/shoudouqi2.html

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 3. (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 4. (2p) $\star$ Masz zaimlementować grę w Dżunglę samodzielnie, nie korzystając ze sprawdzaczki[5].

Zadanie 5. (5p) Wybierz jedną grę (Dżunglę, Szachy albo Reversi[6]). 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.

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