Wstęp do sztucznej inteligencji

Laboratorium trzecie - dwuosobowe gry deterministyczne

Krystian Kamiński nr 304013

Polecenie:

Tematem trzecich ćwiczeń są dwuosobowe gry deterministyczne. Państwa zadaniem będzie napisanie programu / skryptu, który buduje drzewo zadanej gry a następnie gra sam ze sobą w sposób losowy (tzn. jeden z graczy używa naszego algorytmu a drugi nie) i odrzuca ścieżki, które prowadzą do przegranej. Proszę też sprawdzić przypadek, gdy obaj gracze grają w sposób optymalny. Aktualny stan planszy powinien być wyświetlany w konsoli, ale mogą Państwo użyć dodatkowych bibliotek do pisania i wyświetlania gier w Pythonie takich jak pygame.

W raporcie należałoby przedstawić przykład wykonania programu wraz z odpowiadającymi stanami drzewa gry i wyborami algorytmu min-max. Zastosowanie algorytmu alpha-beta nie jest wymagane, ale może okazać się potrzebne. Proszę wykazać w raporcie wygraną każdej ze stron, czyli że gra nie jest ustawiona.

Osoby z nazwiskami od K do Ż

Program buduje drzewo gry dla gry w Reversi (zasady: https://en.wikipedia.org/wiki/Reversi).
Wejściem są wymiary planszy NxN i maksymalna głębokość drzewa.

Założenia:

W grze zamiast pionów czarnych i białych występują „x” i „o”.

Ze względu na dosyć dużą planszę 8x8, przyjąłem głębokość drzewa nie większą od 4.

(W przypadku zwiększenia nawet o 1 do 5 czas oczekiwania na rezultat trwa nawet do kilku minut, więc jest to nieopłacalne)

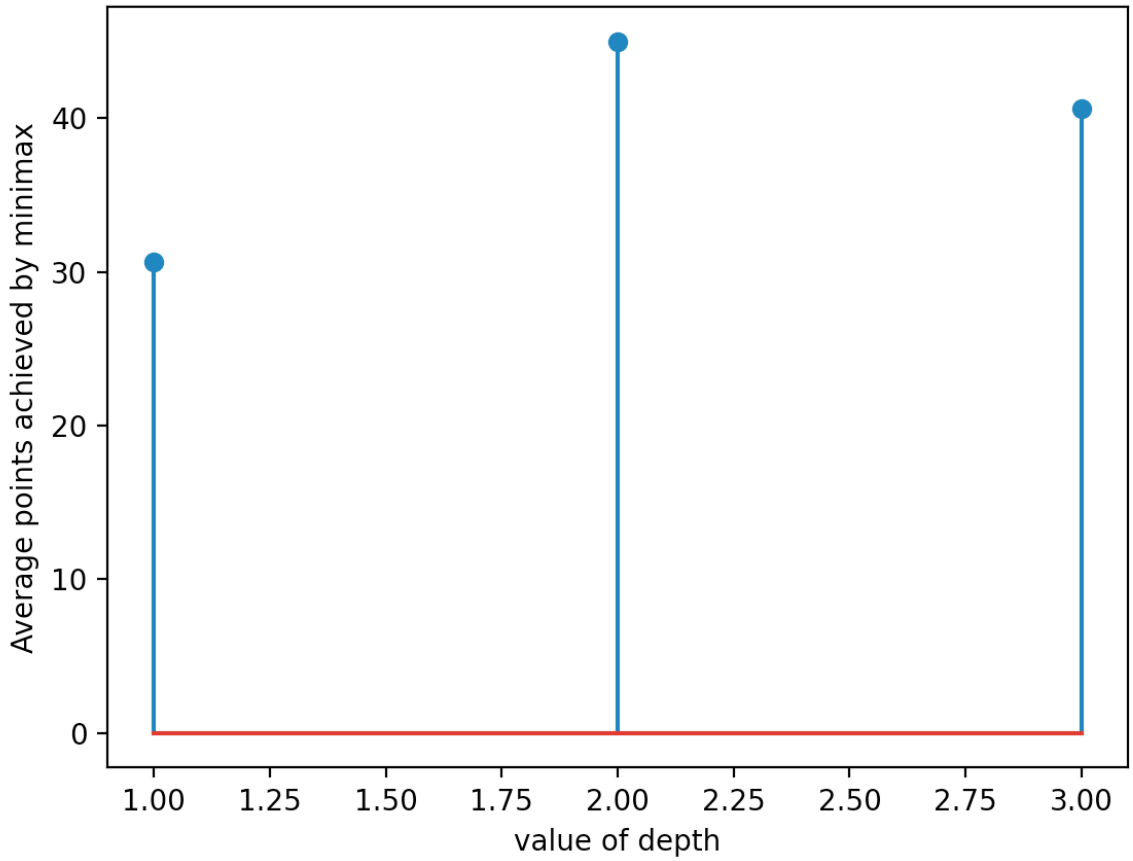
W przypadku gry Othello przewidywanie rezultatu do 4 ruchów do przodu powoduje czasami, że przeciwnik nawet wykonując losowe ruchy, ma szanse wygrać rozgrywkę z „minimax’em”.

Implementacja:

Cały program jest podzielony na następujące funkcje:

- generate_board - Tworzy podstawową tablicę do gry o wymiarach 8x8
- get_board - Tworzy reprezentację tablicy do wyświetlenia
- find_moves - Zwraca wszystkie dozwolone ruchy dla aktualnej planszy
- selections - Umożliwia dodawanie kolejnych „pionów” wraz z aktualizacją planszy
- result - Zwraca aktualny stan rywalizacji
- is_game_end - sprawdza czy gra jest zakończona
- othello - funkcja scalająca pozostałe funkcje w grę othello
- minimax - realizuje algorytm minimax

Wyniki:



Wykres przedstawia średnie wartości końcowe gracza z algorytmem minimax.

Statystycznie im większa wartość głębokości drzewa w algorytmie, tym „lepsze” wyniki zostaną uzyskane. Niestety w tym przypadku zmuszony byłem do używania jedynie bardzo małych wartości głębokości ze względu na długość czasu wykonywania programu.

Poniżej znajduje się fragment rozgrywki, gdzie „o” odpowiada za algorytm minimax, „x” za sposób losowy, przy głębokości równej 3.

```
Move: o
dict_keys([(5, 0), (5, 1), (0, 2), (5, 2), (6, 4), (5, 5), (6, 5)])
Chosen point (5, 1)

o x . o . . . .
o o x o . . . .
o o o o o . . .
o o o o o . . .
o o o o o . . .
. o . o x . . .
. . . . . . . .
. . . . . . . .

Move: x
dict_keys([(6, 1), (5, 2), (1, 4), (4, 5)])
Chosen point (1, 4)

o x . o . . . .
o o x x x . . .
o o o o x . . .
o o o o x . . .
o o o o x . . .
. o . o x . . .
. . . . . . . .
. . . . . . . .
```

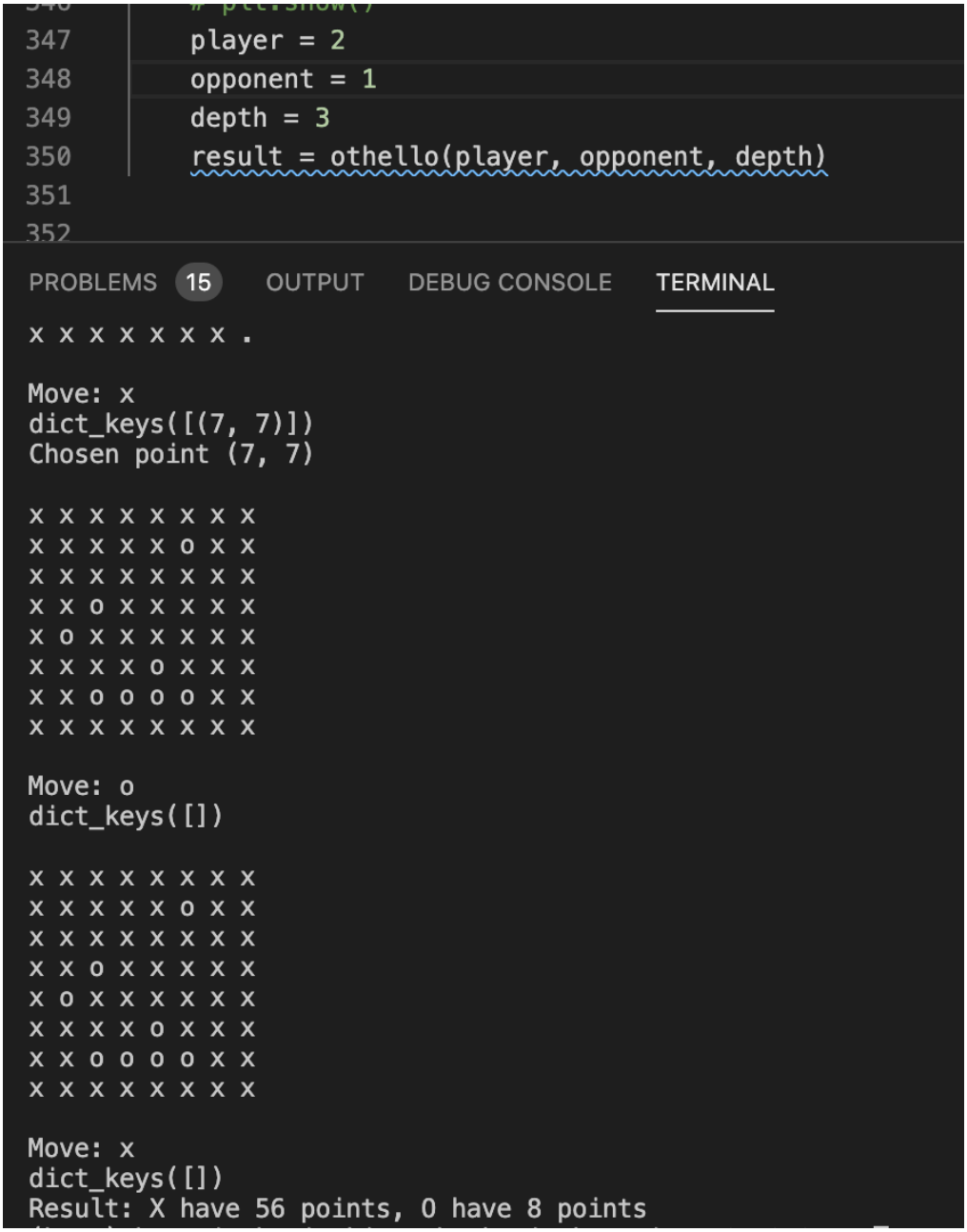
Gra kończy się gdy plansza zostanie zapełniona, lub 2 razy pod rząd żaden z graczy nie będzie miał możliwości ustawienia kolejnego piona. Grę wygrał w tym przypadku algorytm minimax wynikiem 49 do 15

```
Move: o
dict_keys([])

o o o o o o o o
o o o o o o o x
o o o o o o x x
o o o o o x o x
o o o o x o o x
o o o x o x o x
o o x o o o x x
o o o o o o x x

Move: x
dict_keys([])
Result: X have 15 points, O have 49 points
```

W przypadku zmiany wartości player'a na 2 (Wartość 1 w przypadku gracza i oponenta oznacza stosowanie algorytmu minimax, a wartość 2 oznacza sposób losowy) i wartości oponenta na 1, otrzymałem następujący rezultat: 56 do 8 wygrał algorytm minimax ze sposobem losowym.



Poniżej kilka początkowych ruchów pojedynku minimax z przeciwnikiem losowym wraz w wynikiem, zwycięstwo algorytmu.

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . o x . . .
. . . x o . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .

Move: o
dict_keys([(4, 2), (5, 3), (2, 4), (3, 5)])
Chosen point (5, 3)

. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . o x . . .
. . . o o . . .
. . . o . . . .
. . . . . . . .
. . . . . . . .

Move: x
dict_keys([(3, 2), (5, 2), (5, 4)])
Chosen point (5, 4)

. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . o x . . .
. . . o x . . .
. . . o x . . .
. . . . . . . .
. . . . . . . .

Move: o
dict_keys([(2, 5), (3, 5), (4, 5), (5, 5), (6, 5)])
Chosen point (2, 5)
```

```
Move: o
dict_keys([(2, 5), (3, 5), (4, 5), (5, 5), (6, 5)])
Chosen point (2, 5)

. . . . . . . .
. . . . . . . .
. . . . . o . .
. . . o o . . .
. . . o x . . .
. . . o x . . .
. . . . . . . .
. . . . . . . .

Move: x
dict_keys([(2, 2), (3, 2), (4, 2), (5, 2), (6, 2), (2, 4)])
Chosen point (2, 2)

. . . . . . . .
. . . . . . . .
. . x . . o . .
. . . x o . . .
. . . o x . . .
. . . o x . . .
. . . . . . . .
. . . . . . . .

Move: o
dict_keys([(3, 2), (2, 3), (6, 4), (3, 5), (4, 5), (5, 5), (6, 5)])
Chosen point (3, 2)

. . . . . . . .
. . . . . . . .
. . x . . o . .
. . o o o . . .
. . . o x . . .
. . . o x . . .
. . . . . . . .
. . . . . . . .

Move: x
dict_keys([(2, 1), (4, 2), (5, 2), (6, 2), (2, 4)])
Chosen point (5, 2)
```

```
O O O O O O O X
O X O O O O O X
O X X X X X X X
O O O O O O X X
O X O X X X X X
O O X O O X O X
O O O O O O X X
O O O O O O O X

Move: x
dict_keys([])
Result: X have 24 points, O have 40 points
```

Przykładowe zakończenia gry, dla obu graczy stosujących algorytm minimax

```
349     player = 1
350     opponent = 1
351     depth = 3
352     result = othello(player, opponent, depth)
353
354
```

PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL

```
x x x o o o o o
x x x x x o o o
x o o o o o o o
x x o o x x o o
o x x o o x o o
o x x x o o o o
o x x o x x x o
x x x x x x x o

Move: x
dict_keys([])
Result: X have 31 points, O have 33 points
```

```
349     player = 1
350     opponent = 1
351     depth = 2
352     result = othello(player, opponent, depth)
353
```

PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL

```
o o o o o o o o
x o o o x x o o
x x x x o x o o
x o x o o x x x
x o o x x x x x
x o o x x x x x
x x x x o x x x
x x x x x x x x

Move: x
dict_keys([])
Result: X have 40 points, O have 24 points
```

Wnioski:

Po wielu symulacjach nie zaobserwowałem wpływu pozycji początkowych na wynik końcowy.

W ogólności przy braku ograniczeń związanych z doborem głębokości drzewa, pewnie byłoby to zauważalne, lecz dla głębokości mniejszych od 5 istnieje zbyt duża losowość pojedynków.