Laboratorium 3 - sprawozdanie

Wstęp do sztucznej inteligencji

Marcin Grabysz

28.11.2021

1 Treść zadania

Tematem trzecich ćwiczeń są dwuosobowe gry deterministyczne. Państwa zadaniem będzie napisanie programu / skryptu, który buduje drzewo zadanej gry a następnie gra sam ze sobą wykorzystując do tego algorytm minimax.

Program buduje drzewo gry dla gry typu Isolation na planszy 4x4 (NxN, gdzie $N \geqslant 3$). Wejściem programu jest wielkość planszy, położenie na niej obu graczy oraz maksymalna głębokość drzewa.

Zasady:

- każdy z graczy zaczyna z pionkiem, który może ruszyć się o 1 pole na turę (pionowo, poziomo i na ukos)
- gracz nie może postawić pionka na już wcześniej odwiedzonym polu lub na polu przeciwnika
- gra kończy się, jeżeli przeciwnik nie może wykonać już ruchu

W przyjętej implementacji nie jest dopuszczony wariant z remisem.

2 Implementacja

2.1 Wykorzystane narzędzia

Algorytmy i funkcje służace badaniu funkcji zaimplementowane są w języku Python w wersji 3.8.10.

2.2 Zdefiniowane klasy

Podstawowymi komórkami implementacji są obiekty klasy `State` reprezentujące stan gry. `State` przechowuje informacje takie jak stan planszy, położenie każdego z graczy, gracz przy ruchu, lista następników (stanów, które mogą zaistnieć po wykonaniu jednego ruchu w bieżącym stanie).

Stan planszy jest reprezentowany przez obiek klasy `Board`. Klasa przechowuje macierz (listę list) pól. Liczba 0 reprezentuje pole aktywne (nieodwiedzone) a liczba 1 pole nieaktywne. Współrzędna X rośnie w miarę przesuwania się w prawo, zaś współrzędna Y - w miarę przesuwania się ku dołowi planszy, zatem pole (0, 0) znajduje się w lewym górnym rogu planszy. Domyślnym punktem startowym dla gracza Max jest punkt (0, 0), zaś gracza Min - punkt $(N-1, N-1)$.

Lista następników (atrybut `successors`) jest domyślnie pusta i może zostać zainicjalizowana przez użycie metody `initialize_successors`. Inicjalizacja następuje, gdy zasygnalizowana jest taka potrzeba - z bieżącego stanu chcemy wykonać ruch lub zbadać dostępne stany algorytmem minimax. Gdyby lista inicjalizowała się automatycznie, utworzenie instancji stanu początkowego powodowałoby rekursywne utworzenie się instancji wszystkich możliwych stanów, co nie jest potrzebne.

Obiek klasy `State` może sam się ocenić, korzystając z prostej heurystyki (odpowiada za to metoda `find_payoff`). Dla gracza Max przyznana zostaje wypłata równa liczbie dostępnych ruchów (1 - 8). W przypadku, gdy liczba ruchów jest równa 0, co jest równoznaczne z tym, że gracz Max przegrał, funkcja wypłaty zwraca wartość -10. Dla gracza Min wypłata liczona jest w sposób analogiczny; różnica polega na przemnożeniu wartości przez -1.

Metoda `minimax` przeprowadza ewaluację stanu zgodnie z algorytmem minimax. Przyjmuje parametr `depth` (głębokość - maksymalna liczba rekursywnych wywołań).

Za przeprowadzenie jednej rozgrywki odpowiada klasa `Game`. Obiekt tej klasy przechowuje bieżący stan gry oraz informacje o strategii każdego z graczy (losowa, zgodna z algorytmem minimax albo określana przy każdym ruchu przez użytkownika). W momencie utworzenia gry można określić także

rozmiar planszy, początkowe położenie każdego z graczy, gracza rozpoczynającego, głębokość przeszukiwania algorytmu minimax oraz zadeklarować, czy każdy ruch powinien być wyświetlany w konsoli.

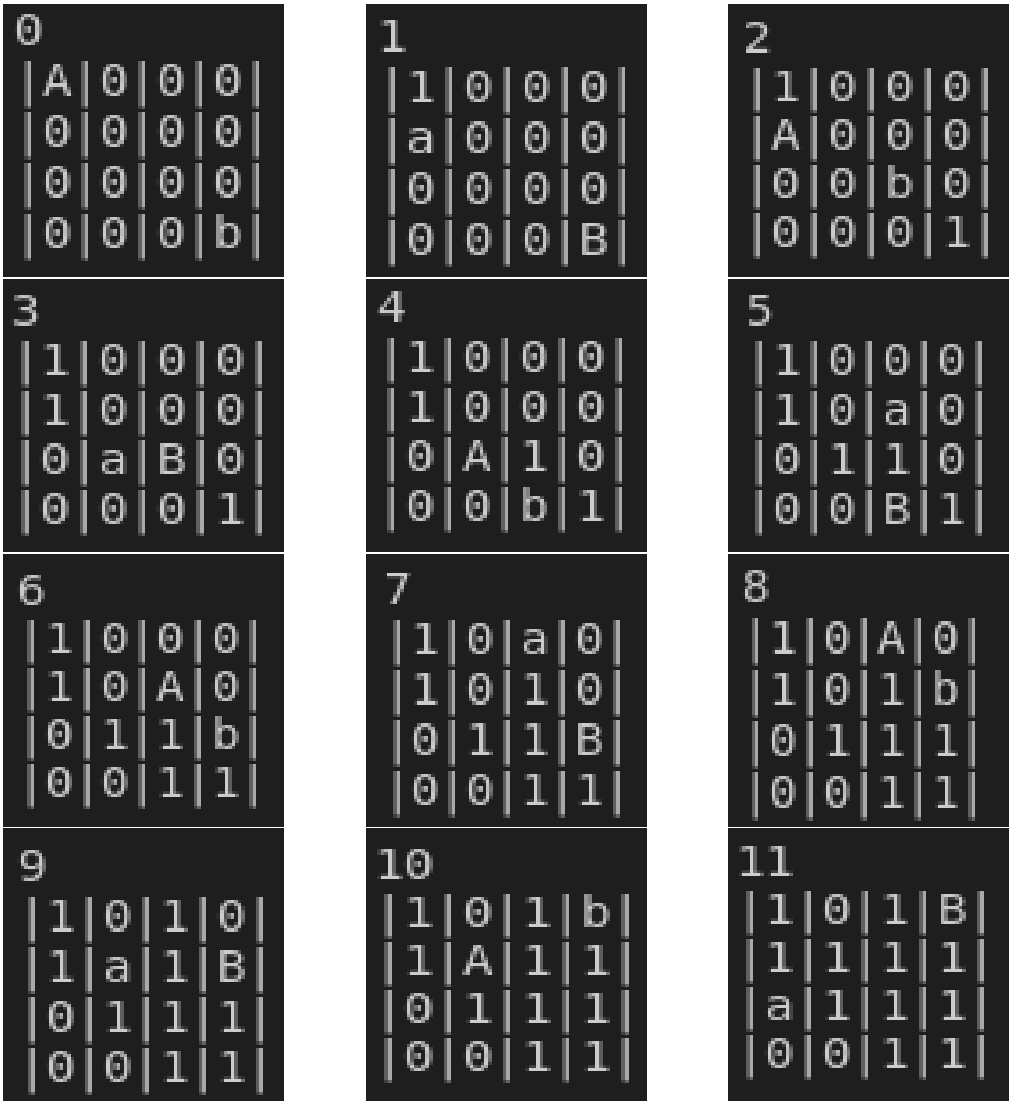
Klasy zaimplementowane są w modułach `classes.py` oraz `game.py`. Moduł `main.py` przeprowadza 1000 gier zgodnie z podanymi parametrami i zapisuje wynik do pliku `info.json`. Aby zmierzyć się ze sztuczną inteligencją posługującą się algorytmem minimax w pojedynczej rozgrywce, należy uruchomić moduł `single_game.py`. Program będzie oczekiwał od użytkownika wprowadzenia kierunku ruchu postaci N (do góry), S (na dół) i analogicznie E, W, NW, NE, SW, SE. Testy jednostkowe powstałe podczas tworzenia projektu znajduję się w module `test_game.py`.

3 Przykładowa rozgrywka

Poniżej przedstawione są kolejne stany gry rozegranej między graczem Max posługującym się algorytmem minimax a graczem Min, który wykonywał losowe ruchy.

- A, a - gracz Max
- B, b - gracz Min
- wielka litera (A, B) - dany gracz jest przy ruchu
- 0 - pole aktywne
- 1 - pole nieaktywne

Zwyciężył gracz Max.



Rysunek 1: Przykładowa rozgrywka

4 Badanie implementacji

4.1 Pomiary

W poniższej tabeli przedstawiony jest stosunek zwycięstw gracza Max do zwycięstw gracza Min po przeprawadzeniu 1000 rozgrywek przy podanych parametrach. We wszystkich rozgrywkach kwadratowa plansza miała szesnaście pól, punktem startowym gracza Max był punkt (0, 0) a gracza Min - punkt (3, 3).

L.p.	Tryb Max	Tryb Min	Rozpoczyna	Głębokość	Max:Min
1	minimax	losowy	Max	2	855:145
2	minimax	losowy	Min	2	933:67
3	losowy	minimax	Max	2	76:924
4	losowy	minimax	Min	2	160:840
5	minimax	minimax	Max	2	269:731
6	minimax	minimax	Min	2	753:247
7	losowy	losowy	Max	2	467:533
8	losowy	losowy	Min	2	537:463
9	minimax	losowy	Max	3	836:164
10	minimax	losowy	Max	4	887:113
11	minimax	losowy	Max	5	923:77

4.2 Wnioski

Wnioski z pomiarów 1. - 4. nasuwają się od razu - po pierwsze, algorytm minimax działa poprawnie i używający go gracz osiąga przytłaczającą przewagę nad przeciwnikiem wykonującym ruchy losowe. Więcej niż 80% partii kończy się zwycięstwem gracza stosującego algorytm. Po drugie, gra typu Isolation bez remisów faworyzuje gracza, który nie rozpoczyna. Wynikająca z tego przewaga nie jest aż tak znacząca, jak przewaga minimaxa nad ruchami losowymi, jednak daje się zauważyć, że gracz postawiony w najkorzystniejszej sytuacji - stosuje algorytm minimax i nie zaczyna - wygrywa ponad 90% partii. Po trzecie, gra jest sprawiedliwa wobec graczy Max i Min, tj. ich skuteczność zależy od wymienionych wyżej parametrów, a nie od tego że jeden z nich dąży do maksymalizacji wypłaty (Max), a drugi do jej minimalizacji (Min).

Pomiary 5. - 8. potwierdzają tezę, że faworyzowany jest gracz, który wykonuje ruch jako drugi. W tych przypadkach mierzą się ze sobą przeciwnicy stosujący taką samą taktykę (obydwaj minimax lub obydwaj ruchy losowe) i każdy z pomiarów pokazuje przewagę gracza, który odpowiada na posunięcia przeciwnika. Przewaga ta jest w niewielkim stopniu znacząca przy partiach losowych, ale zdecydowanie istotna w partiach zagranych według prawie deterministycznego algorytmu.

Pomiary 9. - 11. służą zbadaniu, w jaki sposób głębokość przeszukiwania wpływa na działanie algorytmu. Ponieważ rozgrywki o dużej głębokości zajmują relatywnie dużo czasu, wszystkie pomiary prowadzone są przy założeniu, że gracz Max stosuje algorytm i zaczyna, zaś jego oponent wykonuje ruchy losowe (wszak wpływ tych parametrów został już zbadany w poprzednich pomiarach).

Pierwsza z obserwacji jest zaskakująca - minimax o głębokości 3 okazał się nieznacznie mniej skuteczny niż minimax o głębokości 2 (pomiary 1. i 4.). Dalsze zwiększanie głębokości daje jednakże rezultaty zgodne z oczekiwaniami, czyli powoduje zwiększenie przewagi gracza stosującego strategię minimax nad jego rywalem.