Wstęp do sztucznej inteligencji
Sprawozdanie 3

Jan Kwiatkowski 299374

Kwiecień 2023

1 Treść zadania

Tematem trzeciego zadania jest napisanie programu, który buduje drzewo zadanej gry ('Connect four') a następnie gra sam ze sobą wykorzystując do tego algorytm **minimax**. Należało sprawdzić dwa przypadki:

1. Jeden z graczy gra w sposób losowy (tzn. nie używa tego algorytmu) a drugi stara się optymalizować swoje ruchy (random vs minimax).
2. Obaj gracze podejmują optymalne decyzje (minimax vs minimax).

Aktualny stan planszy powinien być wyświetlany w konsoli, ale można użyć dodatkowych bibliotek do pisania i wyświetlania gier w Pythonie takich jak 'pygame'. W raporcie należy przedstawić przykład wykonania programu wraz z odpowiadającymi stanami drzewa gry i wyborami algorytmu minimax. Zastosowanie przycinania alfa-beta nie jest wymagane, ale może okazać się potrzebne. W raporcie należy pokazać wygraną każdej ze stron, czyli że gra nie jest ustawiona (przeprowadzić 100 testów i zliczyć sumy wyników każdej ze stron).

1.1 Problem

Program buduje drzewo gry w **Czwórki** (**Connectfour**). Wejściem są wymiary planszy NxM (N$\geqslant$ 5,M $\geqslant$ 4) oraz maksymalna głębokość drzewa.
Zasady:

- gracze co turę wrzucają 1 kolorowy token (kolor odpowiada kolorowi danego gracza) do jednej z kolumn na planszy (token opada na spód kolumny – to nie jest kółko i krzyżyk!),
- jeżeli w pionie, poziomie lub na ukos znajdą się 4 tokeny gracza to ten gracz wygrywa,
- jeżeli plansza się zapełni to dochodzi do remisu.

2 Założenia gry

Gra w czwórki jest przykładem dwuosobowej gry deterministycznej w postaci ekstensywnej (rozwiniętej) o sumie zerowej. Oznacza to, że gracze podejmują decyzje sekwencyjnie, mając określone informacje o decyzjach a zysk

jednego gracza oznacza stratę oponenta (interesy graczy są dokładnie przeciwstawne).
W związku z tym wyróżniamy następujące założenia:

- dwaj przeciwnicy,
- wykonują posunięcia naprzemiennie,
- na przebieg gry nie wpływa element losowy (założenie to nie jest spełnione dla przypadku gdy jeden z graczy gra w sposób losowy.),
- stan gry jest znany obu przeciwnikom,
- koniec gry ma przypisany wynik - jest to liczba rzeczywista nazywana wypłatą.

3 Algorytm MinMax

Algorytm Minimax to popularna technika w teorii gier i podejmowaniu decyzji, która umożliwia graczom znalezienie najlepszego ruchu przy założeniu, że przeciwnik również gra optymalnie.

Algorytm ten nie wyczerpuje pełnego drzewa gry, kolejne stany potomne są analizowane niezależnie. Przy czym nie zachodzi sprawdzanie, czy do nowo rozwijanego stanu prowadzi jakaś inna, wcześniej rozwijana droga. Analizowane są sekwencje ruchów (gałęzie drzewa) o ograniczonej długości, oznacza to że nie muszą się one kończyć stanem terminalnym, tylko na pewnym stanie, który należy ocenić. To podejście wymaga zdefiniowania odpowiedniej funkcji oceny stanu. Gracz wybiera ruch zapewniający największą wypłatę, patrząc na kilka ruchów do przodu. Jest to metoda powracania, która ocenia kilka możliwych ruchów naprzód i określa najlepsze rozwiązanie dla tej analizy.

3.1 Cel gry

Algorytm obejmuje dwóch graczy, celem każdego z nich jest:

- dla maksymalizatora (gracz Max), uzyskanie jak najwyższego wyniku,
- dla minimalizatora (gracz Min), dążenie do uzyskania jak najniższego wyniku (minimalizacja wypłaty).

3.2 Model gry

Algorytm do gry w czwórki działa następująco:
1. Utwórz planszę gry i ustaw ją na początkową pozycję, gdzie każde pole planszy jest puste.
2. Gracz 1 wykonuje pierwszy ruch(może być to ruch wybrany przez algorytm minMax), umieszczając pionek w dowolnym wolnym polu planszy.
3. Sprawdź, czy gracz 1 wygrał, tzn. czy udało mu się ułożyć cztery pionki w linii poziomej, pionowej lub skośnej. Jeśli tak, zakończ grę i ogłoś zwycięzcę.
4. Jeśli nie, gracz 2 wykonuje ruch(może być to ruch wybrany przez algorytm minMax), umieszczając pionek w dowolnym wolnym polu planszy.
5. Sprawdź, czy gracz 2 wygrał. Jeśli tak, zakończ grę i ogłoś zwycięzcę.

6. Jeśli żaden z graczy nie wygrał, kontynuuj grę, powracając do kroku 2.

Poniżej znajduje się pseudokod głównego algorytmu rozwijającego stany gry i zwracającego najlepszy wynik obliczonej funkcji heurystycznej. W węźle (parametr node) przekazywany jest chwilowy stan, a całość jest wykonywana rekurencyjnie (analizujemy tyle stanów w głąb tzn.'rozwijamy grę' tak bardzo jaka jest zadana głębokość).

Algorithm 1 Algorytm MinMax

```
Input: depth
function MINMAX(node, depth, isMaximizingPlayer)
    for n : od 1 do I: do
        if depth = 0 or node is a terminal node then
            Return the heuristic value of node
            if isMaximizingPlayer then
                value := inf
                for each child of node do
                    value := max(value, minmax(child, depth  1, FALSE))
                    Return value
                end for
            else * minimizing player *
                value := inf
                for each child of node do
                    value := min(value, minmax(child, depth  1, TRUE))
                    Return value
                end for
            end if
        end if
    end for
end function
```

3.3 Metody klasy MinMax

3.3.1 get_best_move(is_maximizing_player)

Służy do wyznaczenia najlepszego ruchu dla danego gracza w algorytmie min_max. Metoda iteruje przez wszystkie możliwe ruchy w bieżącym stanie gry, generuje ich następniki (czyli stany gry po wykonaniu tych ruchów), a następnie wywołuje metodę min_max() rekurencyjnie na tych następnikach, zmieniając kolejność graczy (jeśli is_maximizing_player jest True, to zmienia na False, itd), oraz zmniejszając głębokość przeszukiwania o 1. Następnie na podstawie ocen zwróconych przez min_max() wybierany jest najlepszy ruch (najwyższa ocena dla gracza maksymalizującego, najniższa ocena dla gracza minimalizującego), który jest zwracany przez metodę jako wynik. Ten najlepszy ruch jest następnie wykonany przez grę, aby zaktualizować jej stan.

3.3.2 min_max(game_state, depth, is_maximizing_player)

Ta metoda ocenia stany gry za pomocą wywoływania rekurencyjnego i zwracania wartości oceny stanów gry na podstawie funkcji heurystycznej. Na

końcu, metoda get_best_move() zwraca najlepszy ruch do wykonania w danym stanie gry na podstawie ocen stanów gry uzyskanych z metody min_max(). W ten sposób metoda get_best_move() służy do wyznaczania optymalnego ruchu.

4 Funkcja oceny

Funkcja ta, która jest również nazywana jest funkcją heurystyczną ma za zadanie obliczyć wartość planszy w zależności od położenia pionków. Jest stosowana w grach turowych, do oszacowania wartości pozycji (w liściu) w drzewie gry.

Funkcja oceny jest unikalna dla każdego rodzaju gry. Podstawową ideą stojącą za funkcją oceny jest przypisanie wysokiej wartości planszy, gdy tura należy do maksymalizatora, lub niskiej wartości planszy, gdy tura należy do minimalizatora.

Skład funkcji oceny jest określany empirycznie poprzez umieszczenie potencjalnej funkcji w automacie i ocenę jej działania w późniejszym czasie. Patrzy ona tylko na bieżącą pozycję (tj. w jakich przestrzeniach znajdują się elementy i ich wzajemne relacje) i nie bierze pod uwagę historii pozycji ani nie bada możliwych ruchów do przodu od danego stanu.

4.1 Strategia

W tym algorytmie, funkcja heurystyczna ocenia stan planszy na podstawie:

- Jeśli gracz 1 wygra na planszy, przypiszemy mu wartość +1.
- Jeśli 2 wygra na planszy, przypiszemy mu ujemną wartość -1.
- Jeśli nikt nie wygrał a plansza jest maksymalnie zapełniona (remis), przypiszemy wartość +0.
- W przeciwnym wypadku wyznaczamy ocene dla danego stanu. Wyznaczona wartość mieści się w przedziale od -1 do 1.

, gdzie gracz 1 to maksymalizator, a gracz 2 to minimalizator.

4.1.1 Ocena stanu

W przypadku braku dojścia do wezła terminalnego w liściu metoda w następujący sposób oblicza wartość oceny korzystając z niżej wymienionych kryteriów oceny:

- Liczba ciagłych połączeń dla gracza i przeciwnika w poziomych, pionowych i ukośnych liniach planszy.
- Wagi dla różnych długości ciągłych linii:
  1. Ciąg o długości 1: waga 0.1
  2. Ciąg o długości 2: waga 0.3
  3. Ciąg o długości 3: waga 0.9

- Wagi dla kolumny środkowej planszy, aby zachęcić algorytm do utrzymania swoich pionków w centralnej kolumnie.
- Środkowe wagi dla różnych rozmiarów planszy:
    1. Plansza o szerokości 4: waga 0.5
    2. Plansza o szerokości 5: waga 0.4
    3. Plansza o szerokości 6: waga 0.3
    4. Plansza o szerokości 7: waga 0.2
    5. Plansza o szerokości 8: waga 0.1

Oznacza to, że krótkie ciągi są mniej ważne, a dłuższe - bardziej wartościowe. Algorytm uwzględnia również wagi dla środkowej kolumny planszy. Im większa plansza, tym mniejsza waga środkowej kolumny.

Wartość końcowa zwraca różnicę między liczbą kontynuujących się linii gracza 1 a gracza 2.

Implementacja oceny stanu znajduje się w klasie 'EvaluateFunctions', która przy inicjalizacji kopiuje dany stan gry (obiekt klasy 'Connect4Game') jako parametr na podstawie którego wyznaczana jest ocena. Obiekt funkcji oceny jest inicjalizowany w momencie wywołania metody 'get_score'.

5 Eksperymenty

5.1 AI vs Random

Eksperymenty przeprowadzono dla 100 uruchomień dla każdego rodzaju parametrów wejściowych. Dla każdej 100 elementowej próby wyznaczono procent zwycięstw algorytmu minMax, procent remisów oraz procent zwycięstw drugiego gracza.

Tabela 1: Wyniki eksperymentów dla gry AI vs Random.
W kolejności [AI win %, Random win %, Raw %]

Głębokość	Tablica 5x5	Tablica 7x7
2	[74,11,15]	[99,0,1]
3	[55,15,30]	[81,18,1]
4	[89,3,8]	[95,1,4]

5.1.1 Wnioski

Algorytm minMax działa na zasadzie naprzemiennego przeszukiwania przestrzeni i dla danego stanu końcowego wyznacza wartość funkcji heurystycznej. W zależności od tego czy stan ten będzie osiągnięty dla ruchu gracza minimalizującego czy też maksymalizującego, wartość heurystyki będzie różna. Dla przypadku w którym głebokość (depth) jest równa 3, ostatni rozpatrywany stan jest wyznaczany właśnie dla gracza minimalizującego co w początkowych ruchach zmniejsza szanse gracza maksymalizującego na wyznaczenie odpowiedniej wartości (tzn. gdy wartość heurystyki jest liczona dla danego gracza to gracz ten zazwyczaj lepiej wypada, algorytm gorzej wnioskuje przyszłość).

5.2 AI vs AI

Tabela 2: Wyniki eksperymentów dla gry AI vs AI.
W kolejności [AI_1 win %, AI_2 win %, Raw %]

Głębokość	Tablica 5x5	Tablica 7x7
2	[0,0,100]	[100,0,0]
3	[100,0,0]	[0,100,0]
4	[0,0,100]	[100,0,0]

5.2.1 Wnioski

Dla małej wielkości tablicy gry algorytm zazwyczaj dochodzi do remisu albo do zwycięstwa zawodnika rozpoczynającego gre. Wyniki pokazują, że jeśli obaj grają maksymalnie optymalnie to wygrywa albo ten, który zaczynał grę i posiada jeden ruch więcej albo ten dla którego ostatecznie była liczona funkcja heurystyczna (czyli przypadek gry depth = 3).