Zadanie 3.

1. Implementacja

1.1. State - klasa reprezentująca stan gry

1.1.1. Atrybuty klasy

- state - obecny stan planszy (lista list [3x3])
- xnumber, onumber - liczba x-ów i y-ów na planszy
- isTerminal - informacja, czy dany stan jest terminalny

1.1.2. Metody klasy

- countChosenPlaces - zwraca ilość x i y na planszy
- generateEmptyState - metoda generująca pustą planszę
- generateChildren - tworzy kolejne możliwe stany planszy
- isTerminal - sprawdza, czy dany stan jest terminalny
- checkWinner - sprawdza, kto wygrał rozgrywkę (ew. remis)

1.2. Player - klasa reprezentująca gracza

1.2.1. Atrybuty klasy

- isMaximizing - czy dany gracz jest Max, czy Min (do jakiej funkcji wypłaty dąży)

1.3. Game - klasa reprezentująca grę

1.3.1. Atrybuty klasy

- currentstate - obecny stan gry (obiekt klasy State)
- max_player - gracz dążący do maksymalizacji funkcji wypłaty (obiekt klasy Player)
- min_player - gracz dążący do minimalizacji funkcji wypłaty (obiekt klasy Player)
- depth - atrybut funkcji minimax, głębokość przeszukiwania stanów

1.3.2. Metody klasy

- play - reprezentuje rozgrywkę

1.4. minimax

1.4.1. Sposób działania

1. Sprawdzenie, czy dany stan jest terminalny
2. Jeśli jest, zwracamy jego funkcję wypłaty (funkcja heuristic)
3. Jeśli nie jest, generujemy stany potomne do obecnego
4. Wywołujemy funkcję minimax dla stanów potomnych
5. Jeśli dany gracz maksymalizuje funkcję wypłaty, zwracamy największą wartość spośród dzieci, w przeciwnym razie wartość najmniejszą

1.5. heuristic

1.5.1. Sposób działania

1. Wywołujemy funkcję calculate_heuristic dla danego stanu rozgrywki
2. Sprawdzamy, czy istnieje zwycięzca rozgrywki. Jeśli jest, zwracamy: dla X - 10, dla O - -10, dla remisu - 0
3. Jeśli rozgrywka nie jest zwycięska, zwracamy wartość wygenerowaną przez funkcję calculate_heuristic

1.6. calculate_heuristic

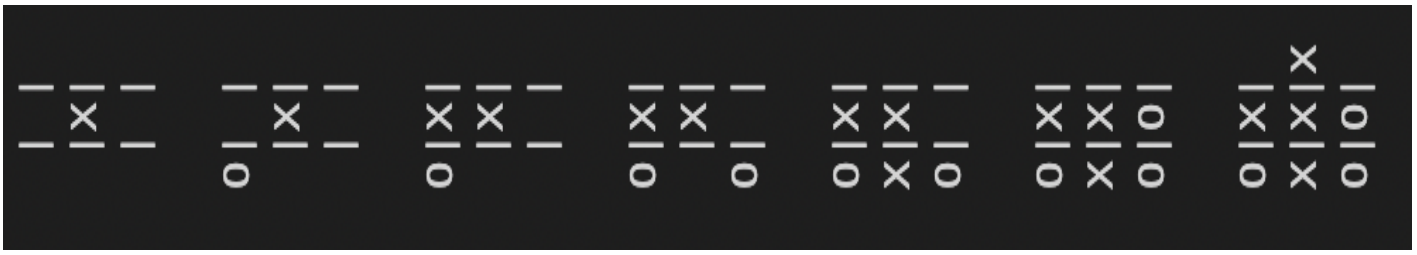
1.6.1 Sposób działania

1. Zgodnie z planszą pokazaną podczas wykłady, obliczamy wartość heurystyki dla danego stanu gry. Pola na rogach mają wartość 3, środkowe 4, pozostałe 2
2. Przechodzimy przez wszystkie pola na planszy. Jeśli na danym polu jest X, dodajemy wartość z planszy, jeśli O, odejmujemy
3. Zwracamy końcową wartość

2. Generowane rozgrywki

1.6. Głębokość mniejsza od 5

Przy takiej głębokości algorytm nie znajdzie stanu terminalnego przy pustej planszy, dlatego pierwszy ruch zostanie na pewno postawiony na środku planszy (największa wartość calculate_heuristic)



1.7. Głębokość większa od 5

Przy takiej głębokości algorytm znajdzie stan terminalny i wybór pola będzie dokonywany na tej podstawie