Grygoruk Piotr 260299

**Sztuczna Inteligencja i Inżynieria Wiedzy**

Prowadzący: Mgr inż. Michał Karol

1. **Zdefiniowanie stanu gry i funkcji generującej możliwe ruchy**

Stan gry w Reversi reprezentowany jest przez planszę 8x8, gdzie każde pole może być puste, mieć monetę gracza 1 lub monetę gracza 2. Gracz 1 rozpoczyna grę, ustawiając dwie monety na planszy - jedną na polu E4 i drugą na polu D5. Gracz 2 ustawia swoją monetę na polu E5 oraz na polu D4. Funkcja generująca możliwe ruchy w grze Reversi zwraca listę koordynat, gdzie gracz może umieścić swoją monetę.

1. **Zbudowanie zbioru heurystyk oceny stanu gry**  
     
   Heurystyki to funkcje, które mają za zadanie oszacować wartość stanu gry. W grze Reversi mogą być użyte następujące heurystyki:

* Liczba monet na planszy - im więcej monet, tym lepiej dla gracza.
* Różnica liczby monet między graczami - im większa różnica, tym lepiej dla jednego z graczy.
* Miejsce na planszy - pewne pola na planszy są bardziej wartościowe niż inne, np. pola w rogach i krawędziach planszy.

1. **Implementacja metody Minimax.**

Minimax to algorytm, który przeszukuje drzewo gry, aby znaleźć najlepszy ruch dla gracza. Metoda ta działa na zasadzie rekurencyjnego przeszukiwania drzewa, w którym węzłami są stany gry, a krawędziami możliwe ruchy. W implementacji algorytmu Minimax dla gry Reversi należy ustalić głębokość przeszukiwania drzewa oraz wykorzystać heurystyki oceny stanu gry. Algorytm działa w ten sposób, że dla każdego węzła drzewa, który reprezentuje stan gry, obliczane są wartości heurystyk. Następnie, dla gracza, który wykonuje ruch, wybierany jest ruch o najwyższej wartości heurystyk, a dla przeciwnika ruch o najniższej wartości. W ten sposób, algorytm przeszukuje drzewo gry, aż do osiągnięcia zadanej głębokości lub znalezienia terminalnego stanu gry

1. **Implementacja alfa-beta cięcia**Implementacja algorytmu Minimax dla gry Reversi w oparciu o strategię alfa-beta cięcia pozwala na przyspieszenie działania algorytmu. W celu wprowadzania tego usprawniania należy do należy wprowadzić dwa dodatkowe argumenty do funkcji minimax: **alpha** oraz **beta**.

Zmienne **alpha** oraz **beta** reprezentują granice wartości najlepszego wyniku, który został już przetworzony. Wartość **alpha** reprezentuje najlepszy wynik, który gracz maksymalizujący może osiągnąć na pewnym poziomie drzewa gry. Wartość **beta** natomiast reprezentuje najlepszy wynik, który gracz minimalizujący może osiągnąć na tym samym poziomie drzewa gry.

W algorytmie alfa-beta cięcia, w momencie, gdy wartość beta dla węzła rodzica jest mniejsza lub równa wartości alfa dla któregokolwiek z węzłów potomnych, nie ma potrzeby dalszego przeszukiwania tego węzła, ponieważ przeciwnik nie będzie wybierał tej ścieżki. Dzięki temu, algorytm może pominąć pewną część drzewa gry, co skutkuje przyspieszeniem działania.

W implementacji algorytmu wartość **alpha** jest początkowo inicjowana na wartość ujemną nieskończoność, natomiast wartość **beta** na dodatnią nieskończoność. Następnie, po każdym ruchu wykonywanym przez gracza, wartość **alpha** jest aktualizowana na podstawie wartości **max(alpha, val)**, gdzie **val** to wartość zwracana przez rekurencyjne wywołanie funkcji minimax dla kolejnego poziomu drzewa gry. Analogicznie, po ruchu przeciwnika, wartość **beta** jest aktualizowana na podstawie wartości **min(beta, val)**.

1. **Podsumowanie**

Do implementacji użyte zostały standardowe biblioteki języka Python takie jak random i copy oraz colorama.

Implementacja odbyła się bez większych problemów, gdyż miałem już wcześniej styczność z algorytmem minimax.

Cały kod został przetestowany i działa zgodnie z oczekiwaniami, umożliwiając rozgrywkę z komputerem lub komputer vs komputer.

1. **Źródła**

[**https://inventwithpython.com/chapter15.html**](https://inventwithpython.com/chapter15.html)

[**https://chat.openai.com/chat**](https://chat.openai.com/chat)

<https://en.wikipedia.org/wiki/Minimax>

<https://eduinf.waw.pl/inf/utils/001_2008/0415.php>