WSI LAB 3
Dwuosobowe gry deterministyczne - Isolation

Będkowski Patryk, 310603

I. WSTĘP

Niniejszy raport przedstawia ćwiczenie polegające na implementacji algorytmu *minimax* do implementacji agenta grającego w grę Isolation. Zasady gry **Isolation**:

- 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

II. URUCHOMIENIE

Do uruchomienia skryptu potrzebne są następujące narzędzia:

- Python w wersji 3.8.8
- Moduły:
    - tabulate 0.8.9
    - pandas 1.2.4
    - numpy 1.20.1

Aby uruchomić skrypt należy użyć polecenia:

```
python main.py --mode
```

gdzie `--mode` może przyjmować jedną z następujących opcji:

Opcja	Rywalizujący gracze
mixed	RandomPlayer vs MinimaxPlayer
minimax	MinimaxPlayer vs MinimaxPlayer

Aby uruchomić skrypt z domyślnym pojedynkiem graczy *MinimaxPlayer* oraz *RandomPlayer*:

```
python main.py
```

III. PRZYGOTOWANE PLIKI

W załączniku raportu przesyłam plik .zip wraz z implementacją gry Isolation oraz wymaganego algorytmu *minimax* dla agenta optymalizującego grę. Plik `main.py` zawiera implementację gry oraz agentów w formie klas. Owe klasy zostały skonstruowane tak, aby mogły przyjmować odpowiednie parametry opisujące rozgrywkę.
Uruchomienie skryptu powoduje utworzenie planszy gry Isolation, utworzeniu dwóch graczy oraz wygenerowaniu ich losowych punktów startowych. Domyślnie jeden z graczy będzie wykonywał ruchy w sposób losowy, drugi natomiast będzie starał się podejmować decyzje przy pomocy algorytmu *minimax.* Skrypt umożliwia wizualizację rozgrywki w konsoli.

IV. ROZWIĄZANIE

Zadanie polega na odtworzeniu gry Isolation w języku python oraz stworzenie agentów grających ze sobą nawzajem. W plikach źródłowych istnieją dwie klasy agentów grających w grę Isolation:

Nazwa Agenta	Sposób gry
RandomPlayer (gracz losowy)	Losowy wybór kolejnych ruchów
MinimaxPlayer (gracz minimax)	Kolejne ruchy są optymalizowane przy pomocy algorytmu minimax

(tabela 1.1) Typy agentów

Gracz losowy – gracz podejmujący decyzję o kolejnym kroku w sposób losowy
Gracz minimax – gracz podejmujący decyzję o kolejnym kroku przy pomocy algorytmu *minimax*

1. Wizualizacja przebiegu rozgrywki MinimaxPlayer oraz RandomPlayer

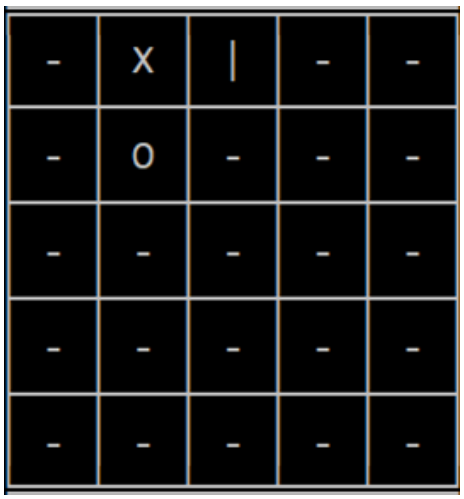
Uruchomienie skryptu opisanego w *sekcji II* powoduje wyświetlenie informacji dotyczących wybranego trybu oraz oznaczeń graczy.

```
Game initialized with a MinMax player and a Random player
O - MinMax player
X - Random player
```

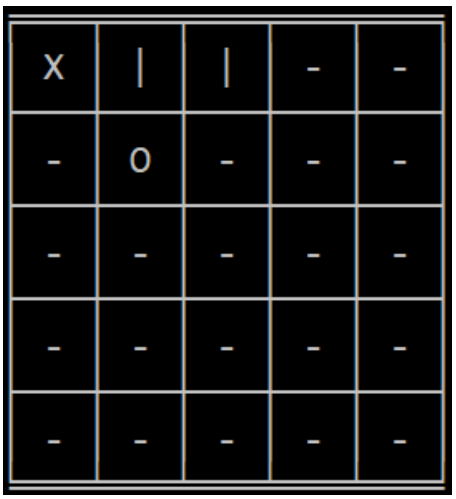
(grafika 1.1) Informacje dotyczące rozgrywki

Po upływie 3 sekund następuje właściwa wizualizacja planszy gry oraz kolejne ruchy graczy opóźnione o 1 sekundę (dla przejrzystości rozgrywki). Przykładowa rozgrywka między graczem typu *MinimaxPlayer* oraz *RandomPlayer* została przedstawiona poniżej:

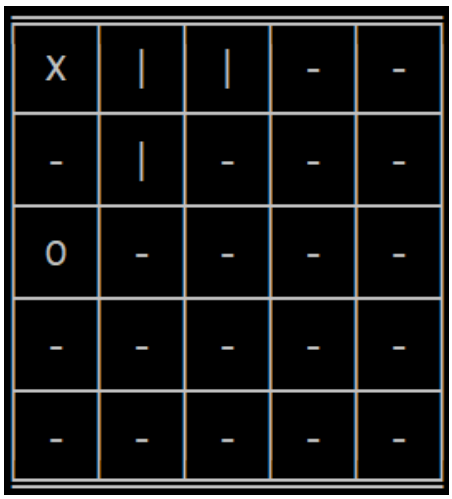
Pozycja startowa



Ruch nr.1

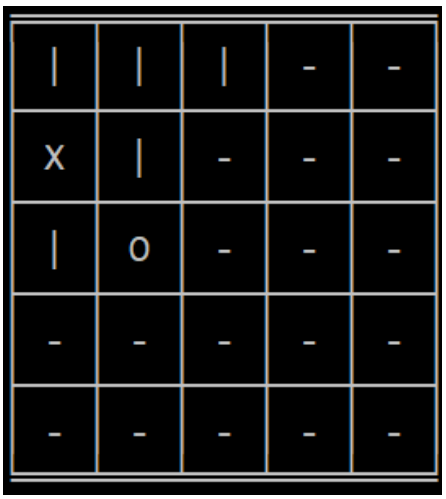


Ruch nr.2



Ruch nr.3

Ruch nr.4



Ruch nr.5

(grafika 1.2) Przykładowa rozgrywka gracza typu MinimaxPlayer (oznaczenie „O") oraz gracza RandomPlayer (oznaczenie „X")

Koniec rozgrywki jest sygnalizowany odpowiednim komunikatem. Prezentowane są również informacje o samej rozgrywce.

```
Player1 (MinmaxPlayer) has won the game

Total number of moves:  5
Execution time with subtracted delay: 0.26 [s]
```

(grafika 1.3) Podsumowanie rozgrywki z grafiki 1.2

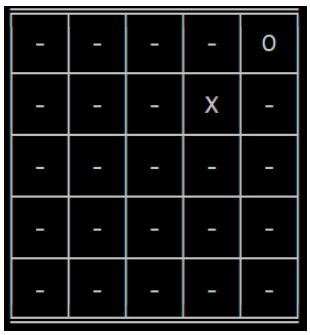
2. Wizualizacja przebiegu rozgrywki dwóch graczy typu MinimaxPlayer

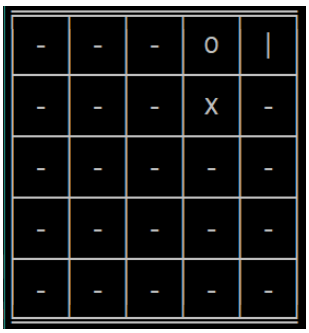
Uruchomienie skryptu opisanego w *sekcji II* powoduje wyświetlenie informacji dotyczących wybranego trybu oraz oznaczeń graczy.

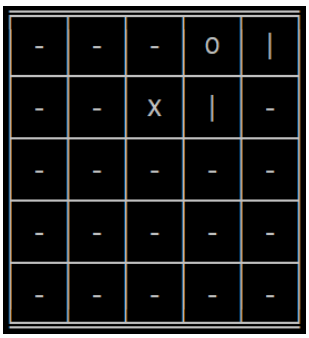
```
Game initialized with two MinMax players
O - MinMax player - Maximizer
X - MinMax player - Minimizer
```

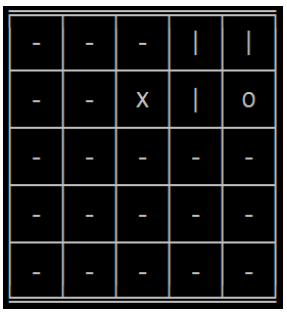
(grafika 2.1) Informacje dotyczące rozgrywki

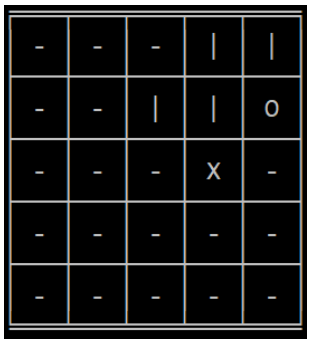
Przykładowa rozgrywka między dwoma graczami typu *MinimaxPlayer* została przedstawiona poniżej:

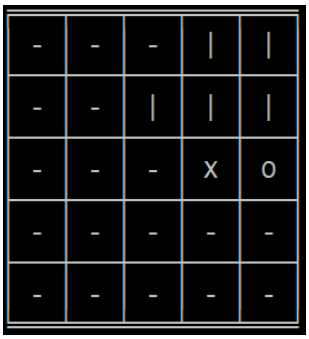

Pozycja startowa

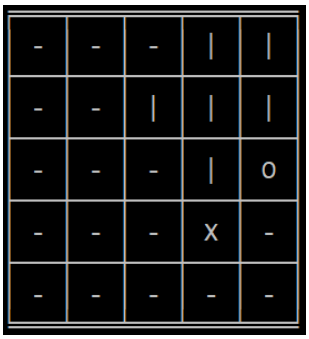

Ruch nr.1

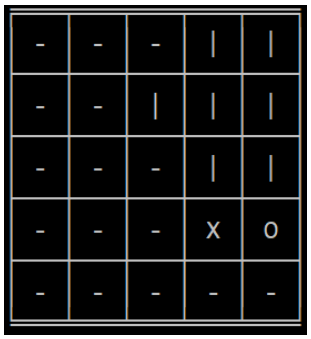

Ruch nr.2

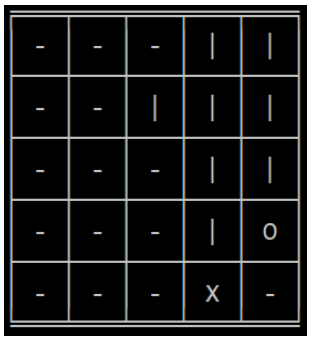

Ruch nr.3


Ruch nr.4

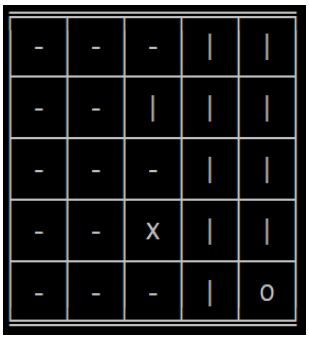

Ruch nr.5


Ruch nr.6


Ruch nr.7


Ruch nr.8


Ruch nr.9


Ruch nr.10

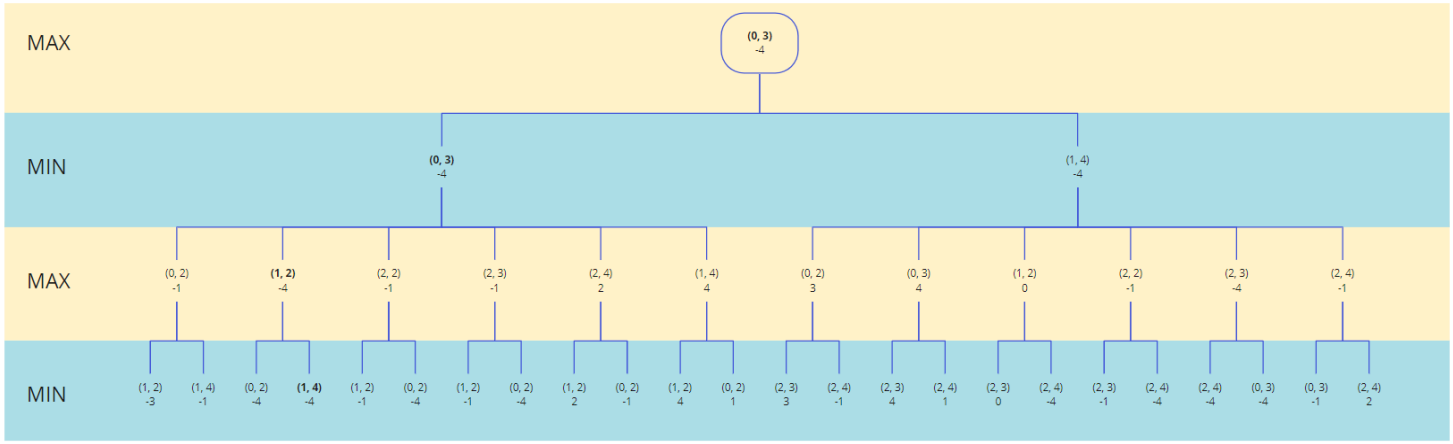
(grafika 2.2) Przykładowa rozgrywka dwóch graczy typu MinimaxPlayer

Koniec rozgrywki jest sygnalizowany odpowiednim komunikatem. Prezentowane są również informacje o samej rozgrywce.

```
Player2 (MinmaxPlayer) has won the game

Total number of moves:  10
Execution time with subtracted delay: 0.49 [s]
```

(grafika 2.3) Podsumowanie rozgrywki z grafiki 2.2



(grafika 2.4) Drzewo decyzyjne ruchu 1 z grafiki 2.3

3. Gracz minimax przeciwko graczowi losowemu

Zbadano gry utworzone z gracza losowego oraz z gracza posługującego się algorytmem minimax. Plansza gry miała rozmiary 5 na 5 pól. Uruchomiono 200 gier o różnych parametrach głębokości *drzewa minimax* i zbadano ilości wygranych każdego z graczy:

a. Głębokość *drzewa minimax* (depth) równa 2:

	Wygrywający gracz	
Rozpoczynający gracz	**MinimaxPlayer**	**RandomPlayer**
MinimaxPlayer	143	57
RandomPlayer	148	52

(Tabela 3.1) Ilość wygranych pojedynków każdego z graczy dla *depth* = 2

Z powyższej tabeli możemy wnioskować, że gracz posługujący się algorytmem *minimax* skutecznie wygrywa w większości pojedynkach, nawet jeśli nie inicjuje on rozgrywki. Równocześnie możemy stwierdzić, że gra nie faworyzuje wyłącznie gracza *Minimax,* gracz losowy również potrafi z nim wygrać w części pojedynków.

b. Głębokość *drzewa minimax* (depth) równa 3:

	Wygrywający gracz	
Rozpoczynający gracz	**MinimaxPlayer**	**RandomPlayer**
MinimaxPlayer	186	14
RandomPlayer	180	20

(Tabela 3.2) Ilość wygranych pojedynków każdego z graczy dla *depth* = 3

Widzimy, że parametr głębokości drzewa w algorytmie *minimax* ma znaczący wpływ na ilość wygranych pojedynków z graczem losowym.

c. Głębokość *drzewa minimax* (depth) równa 4:

	Wygrywający gracz	
Rozpoczynający gracz	**MinimaxPlayer**	**RandomPlayer**
MinimaxPlayer	193	7
RandomPlayer	192	8

(Tabela 3.3) Ilość wygranych pojedynków każdego z graczy dla *depth* = 4

Ponownie, zwiększenie wartości parametru *depth* spowodowało, że gracz posługujący się algorytmem *minimax* wygrał więcej rozgrywek.

Skoro wiemy, że zwiększenie parametru depth wpływa na wynik rozgrywki, sprawdźmy wpływ wartości tego parametru na średni czas wykonywania się 400 rozgrywek:

Głębokość drzewa minimax	Średni czas rozgrywki
2	0,0246 [s]
3	0,0790 [s]
4	0,2923 [s]

(Tabela 3.4) Średni czas trwania rozgrywki

Widzimy, że zwiększenie parametru głębokości drzewa wpływa znacząco na średni czas trwania rozgrywki.

4. Gracz minimax przeciwko graczowi minimax

Zbadano gry utworzone z dwóch graczy typu *minimax*. Plansza gry miała rozmiary 5 na 5 pól. Uruchomiono 200 gier o różnych parametrach głębokości *drzewa minimax* i zbadano ilości wygranych każdego z graczy:

a. Głębokość *drzewa minimax* (depth) równa 2:

	Wygrywający gracz	
Rozpoczynający gracz	**MinimaxPlayer1**	**MinimaxPlayer2**
MinimaxPlayer1	117	83
MinimaxPlayer2	88	112

(Tabela 4.1) Ilość wygranych pojedynków każdego z graczy dla *depth* = 2

Z powyższej tabeli możemy wnioskować, że ilość wygranych każdych z graczy jest zbliżona do siebie nawzajem. Dodatkowo gracz, który rozpoczyna rozgrywkę jako pierwszy ma większe prawdopodobieństwo wygranej, niż gracz wykonujący ruch jako drugi.

b. Głębokość *drzewa minimax* (depth) równa 3:

	Wygrywający gracz	
Rozpoczynający gracz	**MinimaxPlayer1**	**MinimaxPlayer2**
MinimaxPlayer1	105	95
MinimaxPlayer2	90	110

(Tabela 4.2) Ilość wygranych pojedynków każdego z graczy dla *depth* = 3

Widzimy, że parametr głębokości drzewa w algorytmie *minimax* ma wpływ na ilość wygranych pojedynków. Zwiększenie tego parametru spowodowało, że widoczna jest mniejsza faworyzacja gracza rozpoczynającego rozgrywkę.

c. Głębokość *drzewa minimax* (depth) równa 4:

	Wygrywający gracz	
Rozpoczynający gracz	**MinimaxPlayer1**	**MinimaxPlayer2**
MinimaxPlayer1	101	99
MinimaxPlayer2	98	102

(Tabela 4.3) Ilość wygranych pojedynków każdego z graczy dla *depth* = 4

Ponownie, zwiększenie wartości parametru *depth* spowodowało, że to, który gracz rozpocznie rozgrywkę nie ma prawie wpływu na to, który z nich wygra.

Sprawdźmy dodatkowo wpływ wartości tego parametru na średni czas wykonywania się 400 rozgrywek:

Głębokość drzewa minimax	Średni czas rozgrywki
2	0,0586 [s]
3	0,1714 [s]
4	0,7121 [s]

(Tabela 4.4) Średni czas trwania rozgrywki

Widzimy, że zwiększenie parametru głębokości drzewa wpływa znacząco na średni czas trwania rozgrywki. Dodatkowo rozgrywka, w której oboje z graczy korzystają z algorytmu minimax trwa dłużej, niż rozgrywka z graczem losowym.

5. Heurystyka gracza minimax

Gracz typu MinimaxPlayer podejmuje decyzje według heurystyki bazującej na ilości pól aktualnego gracza oraz przeciwnika. Decyzje odnośnie odpowiednich ruchów są podejmowane na podstawie funkcji zysku, która jest dana wzorem:

$$f_{score}(x) = ilość\ dostępnych\ ruchów\ gracza\ \mathbf{max} - ilość\ dostępnych\ ruchów\ gracza\ \boldsymbol{min}$$

6. Podsumowanie

Zaimplementowana przeze mnie gra Isolation pozwala przetestować grę pomiędzy agentami losowymi i tymi wykonującymi bardziej optymalne ruchy. Przebadane przeze mnie sytuacje dla danych parametrów gry sugerują, iż gracz wykorzystujący algorytm *minimax* radzi sobie lepiej w porównaniu do gracza wykonującego losowe ruchy.
Z powyższej analizy wynika, że parametry algorytmu *minimax*, takie jak głębokość drzewa decyzyjnego, wpływają na jakoś uzyskiwanych wyników. Jednakże, udowodniłem równocześnie, że gra nie faworyzuje żadnego z graczy. Każdy z nich ma szansę wygrać.