Andrzej Pultyn

325213

WSI – zadanie 3

- Wstęp

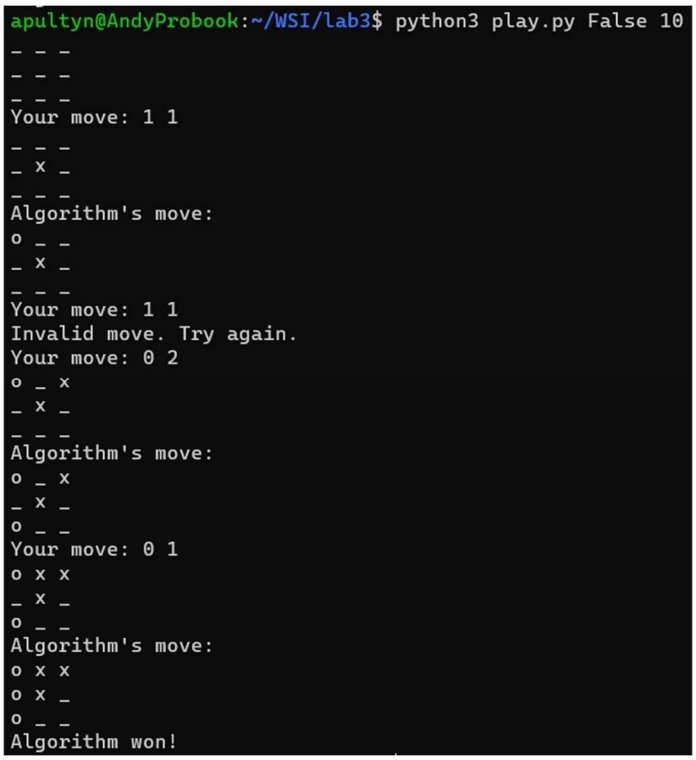
Zadaniem było stworzenie algorytmu do gry w kółko i krzyżyk, który będzie budował drzewo gry. Następnie należało go przetestować na większej ilości gier, oraz zmierzeniem go z algorytmem losowym (losowo wybierającym ruchy).

Algorytm najpierw tworzy sobie listę możliwych ruchów, następnie sprawdza każdy z nich i ocenia ich wartość. Ich wartość liczona jest poprzez znalezienie najlepszej możliwej dla niego kontynuacji partii, przy założeniu, że przeciwnik też gra najlepiej. Jeżeli algorytm gra jako krzyżyk, to dąży do sytuacji, w której będzie miał najwięcej punktów. Dostaje 10 punktów za wygraną, a następnie dodaje do tego obecną wartość depth, czyli ile jeszcze ruchów do przodu ma wymyślić. Im większa wartość depth, tym szybsze zwycięstwo. Analogicznie dla kółka dostaje się -10 punktów za zwycięstwo, wartość depth się odejmuje, a algorytm dąży do jak najmniejszej wartości. Remis traktowany jest jako 0 punktów, nie ważne, jak późno jest osiągany.

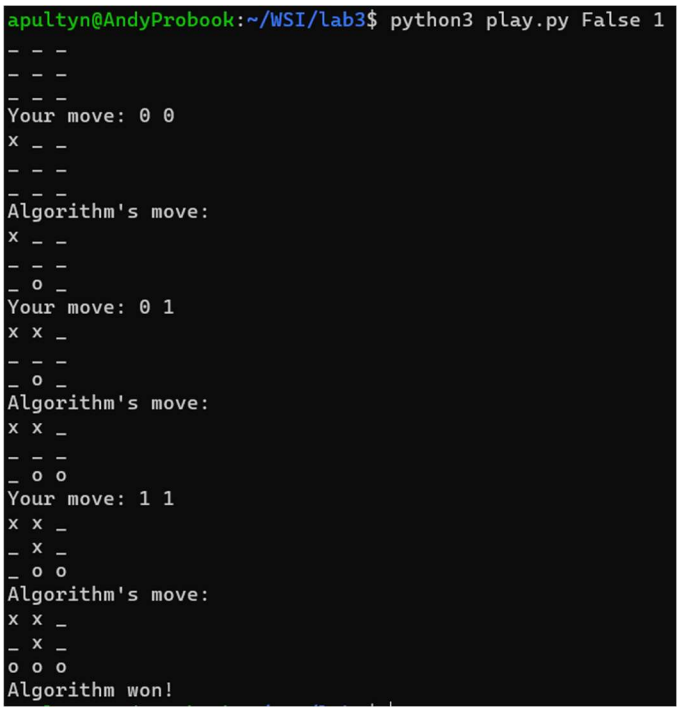
- Przykładowe gry:

```
apultyn@AndyProbook:~/WSI/lab3$ python3 play.py False 10
_ _ _
_ _ _
_ _ _
Your move: 1 1
_ _ _
_ x _
_ _ _
Algorithm's move:
_ _ o
_ x _
_ _ _
Your move: 2 2
_ _ o
_ x _
_ _ x
Algorithm's move:
o _ o
_ x _
_ _ x
Your move: 0 1
o x o
_ x _
_ _ x
Algorithm's move:
o x o
_ x _
_ o x
Your move: 1 2
o x o
_ x x
_ o x
Algorithm's move:
o x o
o x x
_ o x
Your move: 2 0
o x o
o x x
x o x
Draw!
```

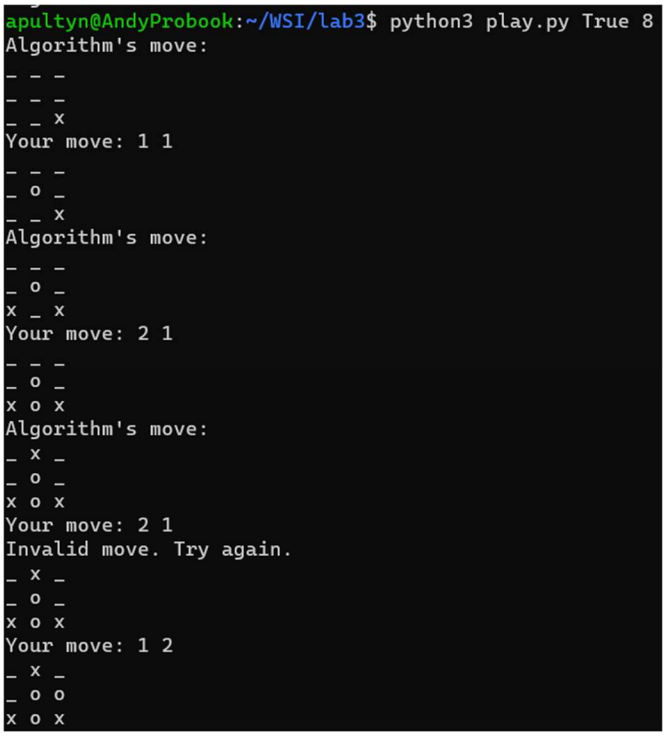
W tej grze grałem jako krzyżyk, z maksymalną głębią (każda wartość depth powyżej 8 jest traktowana jak 8, bo tylko tyle ruchów maksymalnie może być wykonane w grze). Jak widać, algorytm blokował każdy mój atak.

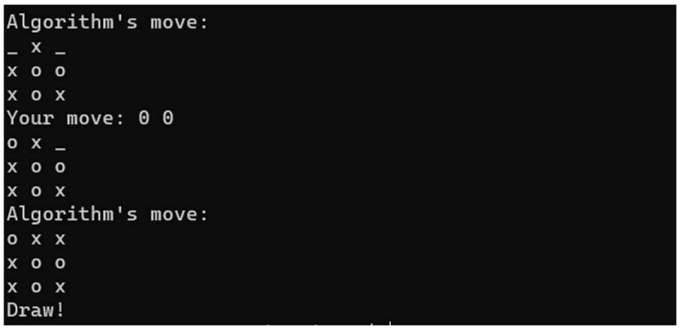


W tej grze po pierwsze zademonstrowałem, że gra nie pozwoli na wykonanie nielegalnego ruchu, a po drugie, jak się podłożyłem, to algorytm od razu z tego skorzystał.



Tutaj grałem na głębokości 1, więc algorytm szukał tylko, czy jego następny ruch to jest zwycięstwo czy nie. Nie był w stanie przewidzieć żadnego mojego ruchu, dlatego w jego drugim ruchu każda pozycja była dla niego równoważna. Za to jak mu się podłożyłem, to zauważył zwycięstwo w 1 ruchu i z niego skorzystał.





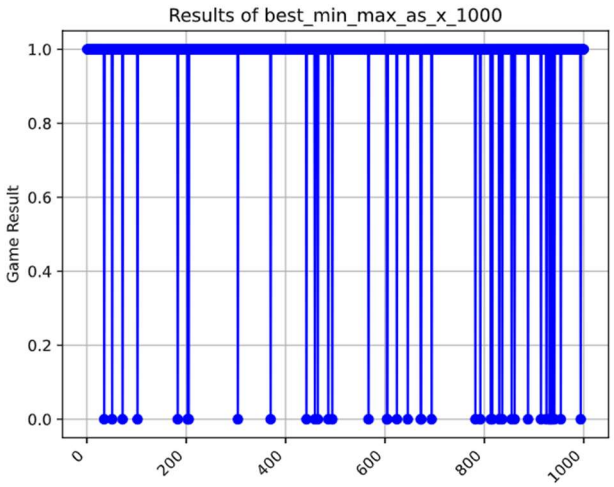
W tej grze grałem jako kółko, na maksymalnej głębokości. Znowu algorytm blokował moje ataki i gra skończyła się remisem.

```
apultyn@AndyProbook:~/WSI/lab3$ python3 play.py True 8
Algorithm's move:
x _ _
_ _ _
_ _ _
Your move: 2 0
x _ _
_ _ _
o _ _
Algorithm's move:
x _ _
_ _ _
o _ x
Your move: 1 1
x _ _
_ o _
o _ x
Algorithm's move:
x _ x
_ o _
o _ x
Your move: 0 1
x o x
_ o _
o _ x
Algorithm's move:
x o x
_ o x
o _ x
Algorithm won!
```

Ten przykład chyba najlepiej demonstruje siłę algorytmu. Algorytm po moim 1 ruchu zastawił na mnie pułapkę bez wyjścia. Mogłem albo przegrać od razu, albo zagrać na środek. Później algorytm zaatakował mnie w 2 miejscach na raz, czym wygrał grę.

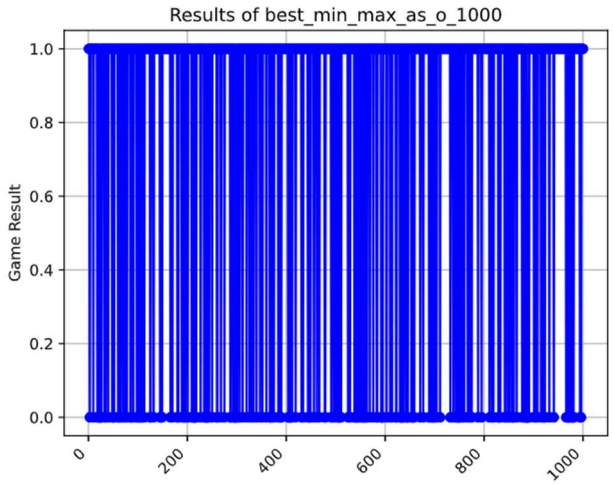
- Testy

Testy polegały na grach pomiędzy algorytmem min-max, a algorytmem losowym. Wynik 1 oznacza wygraną algorytmu min-max, 0 oznacza remis, a -1 wygraną algorytmu losowego. Na początku komendą [python3 testing.py 1000 True 10 False --test_name=best_min_max_as_x_1000] uruchomiłem test, w którym min-max grał jako krzyżyk, na maksymalnej głębokości:



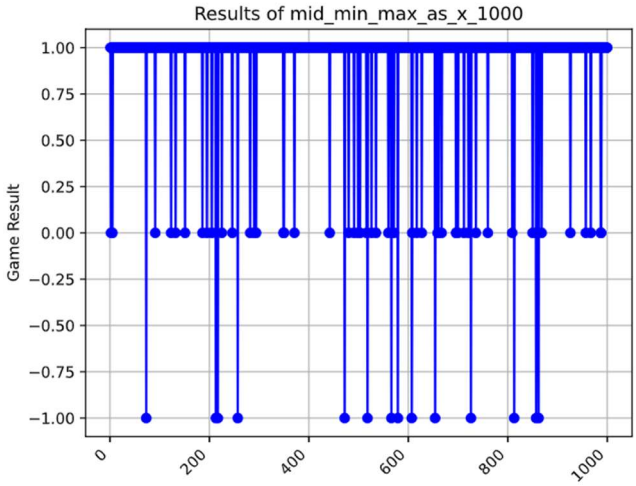
```
{
    "params": {
        "iter": 1000,
        "min_max_as_x": true,
        "depth": 10,
        "print_results": false,
        "test_name": "best_min_max_as_x_1000"
    },
    "results": {
        "wins": 961,
        "win_per": 0.96,
        "lost": 0,
        "lost_per": 0.0,
        "draws": 39,
        "draws_per": 0.04
    }
}
```


Jak widać, po 1000 rozegranych partii, min-max wygrał 961 razy, a resztę partii zremisował. Był to oczekiwany wynik, ponieważ przy najlepszej grze jednej ze stron w kółko i krzyżyk, nie jest możliwe, aby ta strona przegrała.



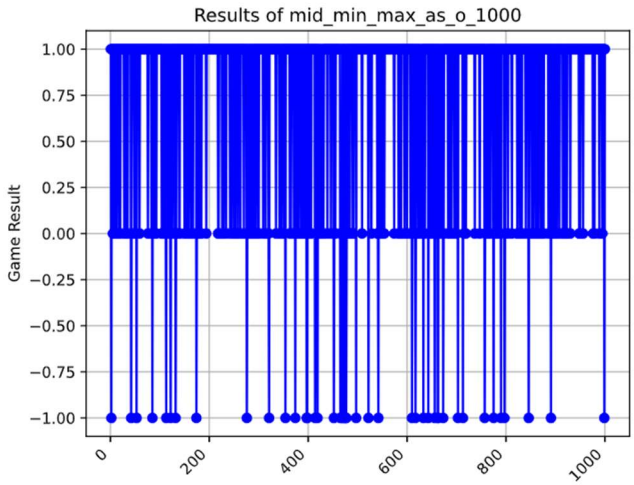

```
{
    "params": {
        "iter": 1000,
        "min_max_as_x": false,
        "depth": 10,
        "print_results": false,
        "test_name": "best_min_max_as_o_1000"
    },
    "results": {
        "wins": 788,
        "win_per": 0.79,
        "lost": 0,
        "lost_per": 0.0,
        "draws": 212,
        "draws_per": 0.21
    }
}
```


Gdy min-max grał jako kółko, czyli algorytm losowy zaczynał grę, to zdecydowanie więcej partii kończyło się remisem (21% w porównaniu do 4%), ale nadal żadnej partii nie przegrał. Warte odnotowania jest też, że testowanie poszło zdecydowanie szybciej, gdy algorytm losowy zaczynał, ponieważ min-max miał wtedy mniej pozycji do przeliczenia.



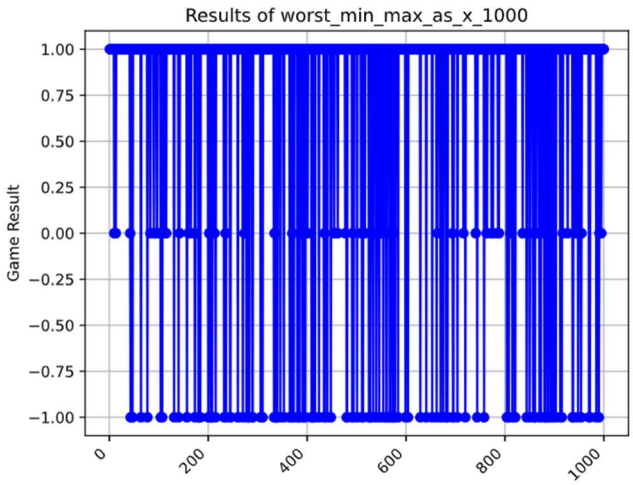

```
{
    "params": {
        "iter": 1000,
        "min_max_as_x": true,
        "depth": 4,
        "print_results": false,
        "test_name": "mid_min_max_as_x_1000"
    },
    "results": {
        "wins": 937,
        "win_per": 0.94,
        "lost": 14,
        "lost_per": 0.01,
        "draws": 49,
        "draws_per": 0.05
    }
}
```


Przy zmniejszeniu głębokości do 4, na 1000 partii algorytm losowy był w stanie pokonać min-maxa, ale tylko pojedyncze razy.



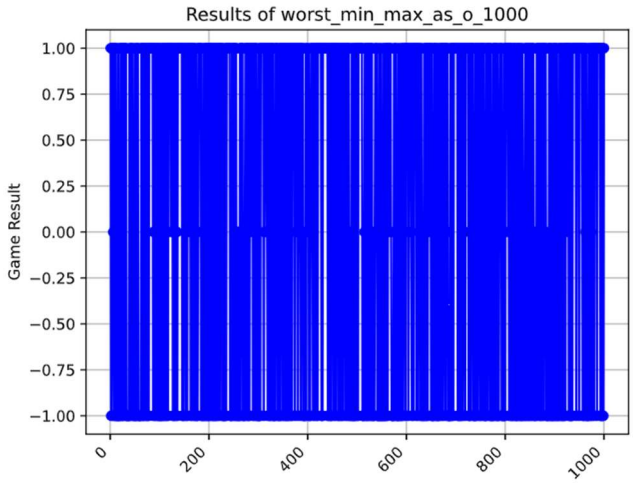
```
{
    "params": {
        "iter": 1000,
        "min_max_as_x": false,
        "depth": 4,
        "print_results": false,
        "test_name": "mid_min_max_as_o_1000"
    },
    "results": {
        "wins": 759,
        "win_per": 0.76,
        "lost": 43,
        "lost_per": 0.04,
        "draws": 198,
        "draws_per": 0.2
    }
}
```


Gdy min-max grał jako kółko, to powtórzył się scenariusz, w którym zdecydowanie więcej partii kończyło się remisem niż przy pełnej głębokości liczenia, ale i tym razem przez mniejszą głębokość liczenia kilka partii przegrał.




```
{
    "params": {
        "iter": 1000,
        "min_max_as_x": true,
        "depth": 1,
        "print_results": false,
        "test_name": "worst_min_max_as_x_1000"
    },
    "results": {
        "wins": 809,
        "win_per": 0.81,
        "lost": 126,
        "lost_per": 0.13,
        "draws": 65,
        "draws_per": 0.07
    }
}
```

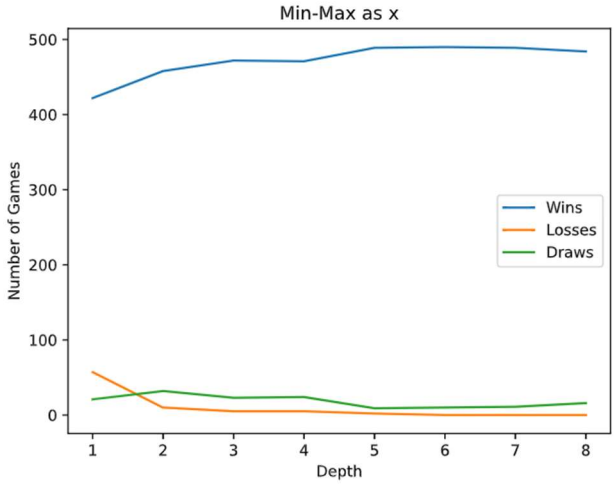

Przy zredukowaniu głębokości do najmniejszej możliwej (1), to min-max osiągnął swój zdecydowanie najgorszy wynik jako krzyżyk, ale nadal wygrał w 81% przypadków.

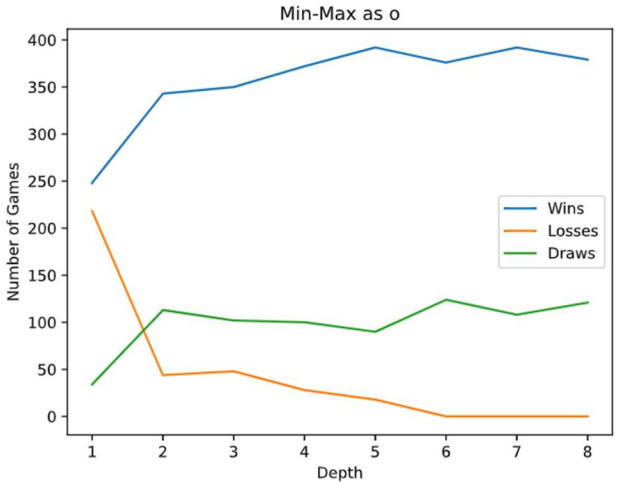



```
{
    "params": {
        "iter": 1000,
        "min_max_as_x": false,
        "depth": 1,
        "print_results": false,
        "test_name": "worst_min_max_as_o_1000"
    },
    "results": {
        "wins": 521,
        "win_per": 0.52,
        "lost": 400,
        "lost_per": 0.4,
        "draws": 79,
        "draws_per": 0.08
    }
}
```


Przy najgorszych warunkach dla min-maxa, czyli głębokości 1, i graniu jako kółko, wyniki gry były rzeczywiście losowe (52% zwycięstw i 40% porażek). Co warte odnotowania, w takiej sytuacji stosunkowo mało partii kończyło się remisem (4%).

Ostatnim testem było rozegranie przez min-maxa po 500 partii (odpowiednio jako krzyżyk i kółko) na każdą możliwą głębokość (od 1 do 8) i pokazanie, jak zmieniała się ilość zwycięstw, porażek i remisów:




Min-Max as o
Number of Games
Depth
Wins
Losses
Draws
400
350
300
250
200
150
100
50
0
1
2
3
4
5
6
7
8