

Projektowanie Algorytmów i Metody Sztucznej Inteligencji			
Autor: Kacper Brodziak Nr indeksu: 258978	Grupa: Y03-51f	Rok ak: 2	Semestr: 4
	Termin: 13:15 – 15:00	Oddano: 14.06.2022	Ocena:
Nr projektu: 3	Temat: Gra kółko i krzyżyk		

1. Wstęp

Zadaniem projektu było stworzenie gry wykorzystującej algorytm sztucznej inteligencji. Zadanie, które wybrałem polegało na stworzeniu gry kółko i krzyżyk z wykorzystaniem algorytmu w oparciu o strategię Min-Max. W grze gracz może definiować rozmiar kwadratowej planszy, ilość znaków potrzebnych by wygrać oraz głębokość przeszukiwania algorytmu.

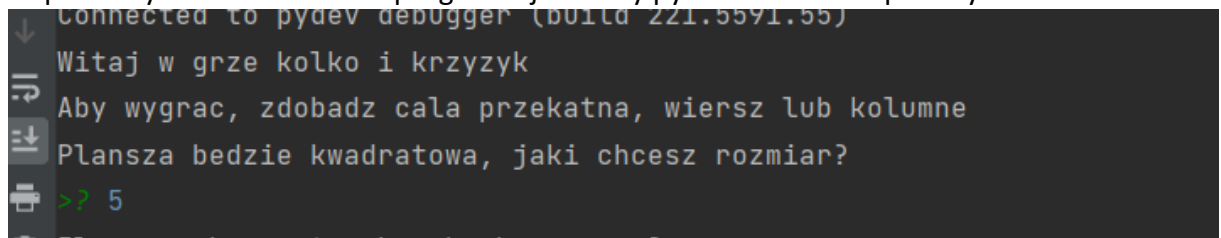
2. Algorytm Alfa-Beta

Jest to algorytm przeszukujący, który redukuje liczbę węzłów, potrzebnych do rozwiązania w drzewach przeszukujących przez algorytm min-max. Zastosowany algorytm jest to przeszukiwaniem wykorzystywanych najczęściej w grach dwuosobowych, takich jak właśnie kółko i krzyżyk, warcaby, czy też szachy. Zatrzymanie następuje pod warunkiem znalezienia przynajmniej jednego rozwiązania czyniącego obecnie badaną opcję ruchu gorszą od poprzednich. Wybranie takiej opcji ruchu nie przyniosłoby korzyści graczowi ruszającemu się, dlatego też nie ma potrzeby przeszukiwać dalej gałęzi drzewa tej opcji, ponieważ obniża to sprawność algorytmu. Możemy w ten sposób zaoszczędzić sporo czasu na niepotrzebne przeszukiwanie algorytmu.

3. Stworzona gra (przykład dla planszy 5x5)

Plansza gry oraz wszystkie komentarze do gry wyświetlane są w terminalu. Ze względu na ograniczenie czasowe nie zdążyłem stworzyć do gry interface'u graficznego.

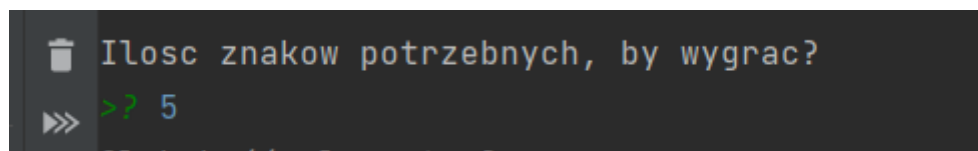
W pierwszym kroku działania programu jesteśmy pytani o wielkość planszy:



```
Connected to pydev debugger (build 221.5591.55)
Witaj w grze kolko i krzyzyk
Aby wygrac, zdobadz cala przekatna, wiersz lub kolumne
Plansza bedzie kwadratowa, jaki chcesz rozmiar?
> 5
```

Rysunek 1. Wielkość planszy

Następnie wprowadzamy ilość znaków potrzebnych do zwycięstwa :



```
Ilosc znakow potrzebnych, by wygrac?
> 5
```

Rysunek 2. Ilość znaków do zwycięstwa

Na sam koniec wpisujemy głębokość rekurencji algorytmu:

```
>? 5
Głębokość algorytmu?
>? 3
```

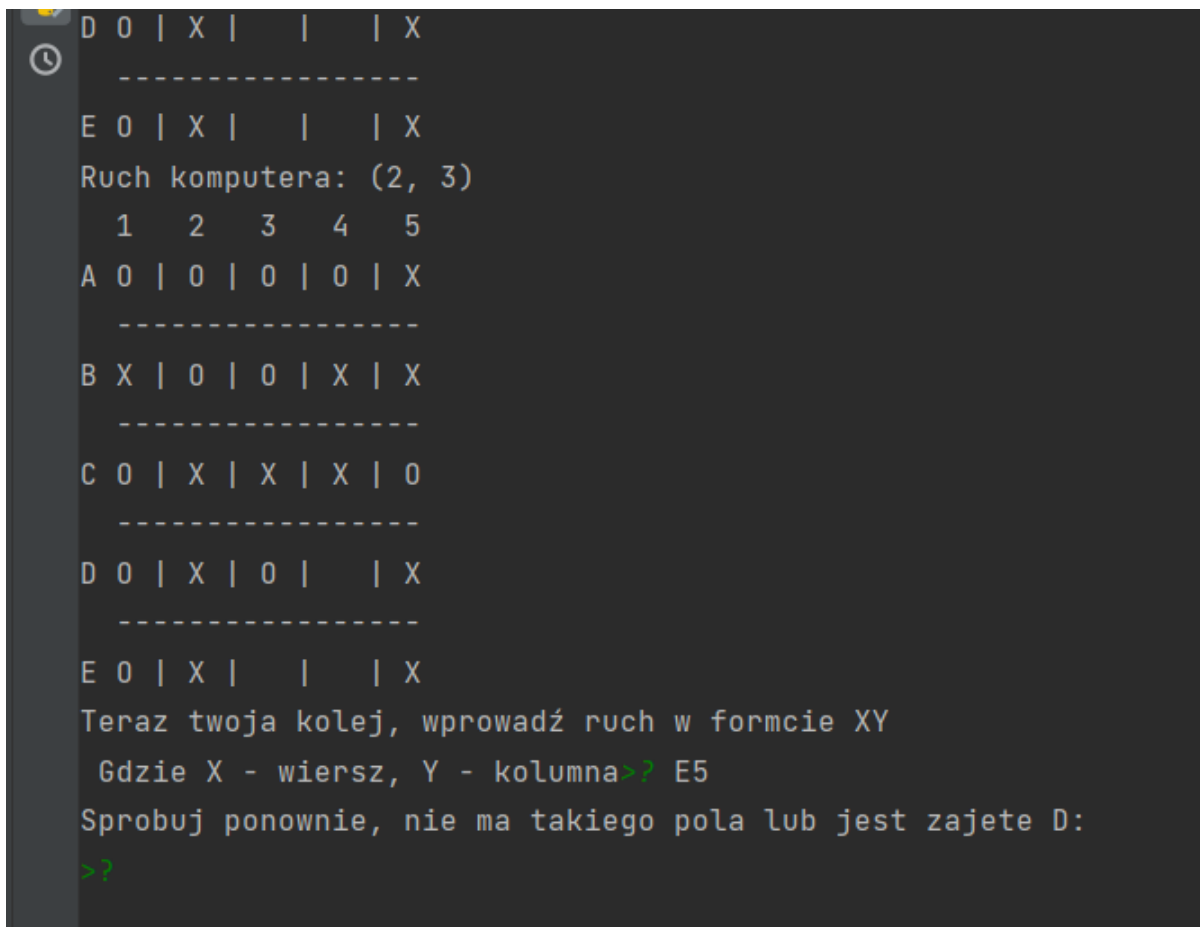
Rysunek 3. Głębokość algorytmu

Po czym w terminalu wyświetla się plansza gry, ruchy będziemy wprowadzali wpisując wielką literę (sygnalizując wiersz do postawienia X) oraz cyfrę (sygnalizując kolumnę):

```
>? 3
1 2 3 4 5
A | | | |
-----
B | | | |
-----
C | | | |
-----
D | | | |
-----
E | | | |
Teraz twoja kolej, wprowadź ruch w formcie XY
Gdzie X - wiersz, Y - kolumna
>?
```

Rysunek 4. Start gry

W przypadku błędnego wprowadzenia danych jesteśmy informowani o błędzie i możemy wpisać punkt ponownie:



```
D 0 | X |   |   | X
-----
E 0 | X |   |   | X
Ruch komputera: (2, 3)
  1  2  3  4  5
A 0 | 0 | 0 | 0 | X
-----
B X | 0 | 0 | X | X
-----
C 0 | X | X | X | 0
-----
D 0 | X | 0 |   | X
-----
E 0 | X |   |   | X
Teraz twoja kolej, wprowadź ruch w formcie XY
Gdzie X - wiersz, Y - kolumna>? E5
Spróbuj ponownie, nie ma takiego pola lub jest zajęte D:
>?
```

Rysunek 5. W trakcie gry

Na sam koniec wyświetla nam się wynik gry, w tym przypadku jest to remis:

```
E 0 | X | X |   | X
Teraz twoja kolej, wprowadź ruch w formcie XY
Gdzie X - wiersz, Y - kolumna>? E4
  1  2  3  4  5
A 0 | 0 | 0 | 0 | X
-----
B X | 0 | 0 | X | X
-----
C 0 | X | X | X | 0
-----
D 0 | X | 0 | 0 | X
-----
E 0 | X | X | X | X
Remis!
import sys; print('Python %s on %s' % (sys.version, sys.
Python Console
```

4. Wnioski

- Początkowym problemem w działaniu programu był długi czas oczekiwania na ruch komputera. Było to w głównej mierze przez to, że w celu oceny ruchu kopiowano planszę za każdym wywołaniem MinMaxa. Rozwiązałem problem, używając jednej planszy, ale cofając ruch do momentu otrzymania optymalnego rozwiązania.
- Podstawowa wersja algorytmu bardzo wolno działała już dla planszy 4x4, algorytm Alfa-Beta pozwolił przyspieszyć działanie programu, w szczególności w początkowych ruchach. Powolność była spowodowana początkową, dużą dostępnością ruchów, ze względu na pustą/prawie pustą planszę.
- Niestety nie udało mi się ustalić optymalizacji głębokości na podstawie rozmiaru planszy oraz ilości znaków potrzebnych do zwycięstwa
- Przy większych planszach, np. 6x6 i większe algorytm już ma problemy z wydajnością i oczekiwanie na ruch komputera jest już dość długie. Dobrym rozwiązaniem dla większych plansz byłoby przygotowanie baz danych z możliwymi kombinacjami ruchów

5. Literatura

- Książka „Python dla programistów. Big Data i AI. Studia przypadków” Autorzy: Paul J. Deitel, Harvey Deitel
- Wikipedia: Algorytm MinMax
- Wikipedia: Algorytm Alfa-Beta

Odnosnik do githuba: https://github.com/brodzi4k/kolko_krzyzyk