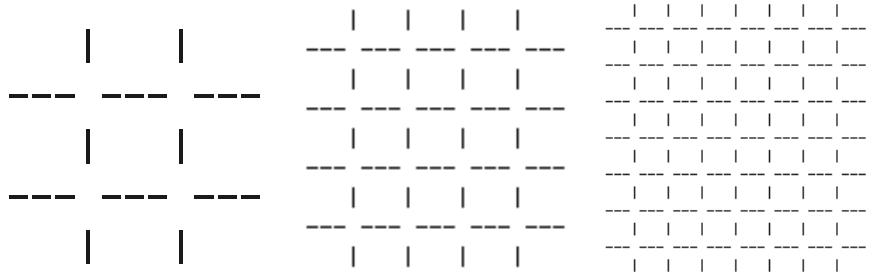
PAMSI - projekt 3
Kółko i krzyżyk

Aleksandra Rzeszowska (234780)
środa, 18:55-20:35

6 czerwca 2018

1 Sposób implementacji

Tablica gry jest reprezentowana jako jednowymiarowa tablica rozmiaru $n \cdot n + 1$, gdzie n jest rozmiarem tablicy, której elementy o indeksach od 1 do $n \cdot n$ reprezentują kolejne pola kratownicy. Kratownica wyświetla się w terminalu po uprzednim pobraniu od użytkownika danych niezbędnych do rozpoczęcia gry (rozmiar planszy i ilość elementów w rzędzie potrzebnych do wygrania). Przykładową planszę, wraz z rozmiarem przedstawia grafika poniżej.



(a) 3x3 (b) 5x5 (c) 8x8

Rysunek 1: Kratownica do gry

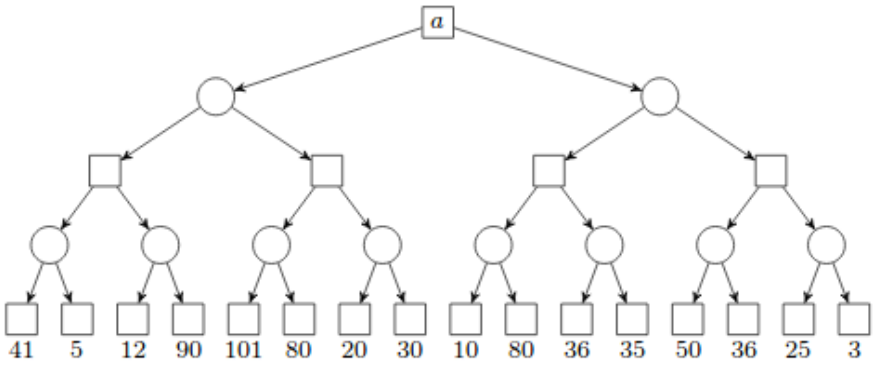
Gra została zaimplementowana jako klasa KolkoIKrzyzyk. Atrybutami tej klasy są dwie liczby całkowite przechowujące dane o rozmiarze kratownicy i liczbie elementów w rzędzie niezbędnych do wygrania rundy oraz wskaźnik na tablicę do gry przechowującą literały znakowe. Metody klasy KolkoIKrzyzyk opisano poniżej:

- **KolkoIKrzyzyk(int rozm, int ile)** - konstruktor klasy KolkoIKrzyzyk, który tworzy planszę do gry
- **void RysujPlansze()** - metoda rysująca planszę do gry w jej aktualnym stanie zapełnienia
- **CzyWygrana(char gracz, bool cisza)** - metoda sprawdzająca, czy dany gracz wygrał rozgrywkę. Jeśli logiczna zmienna *cisza* jest ustawiona na $false$, metoda wypisze na standardowy wyjście odpowiedni komunikat, a jeśli zmienna ta ustawiona jest na *true*, zwróci tylko odpowiedni wynik
- **CzyRemis(bool cisza)** - metoda sprawdzająca, czy nastąpił remis (brak pustego miejsca na planszy). Podobnie jak metoda sprawdzająca czy nastąpiła wygrana, generuje lub nie odpowiedni komunikat na standardowe wyjście w zależności od wartości przechowywanej w zmiennej *cisza*
- **MiniMax(char gracz)** - metoda realizująca działanie algorytmu MiniMax (opis algorytmu poniżej), celem wyboru najlepszej możliwej drogi prowadzącej do zwycięstwa sztucznej inteligencji
- **AlfaBeta(char gracz, int alfa, int beta)** - metoda będąca usprawnieniem algorytmu MiniMax, realizuje alfa-beta cięcia, by przyspieszyć działanie programu, redukując ilość rekurencyjnych wywołań
- **RuchKomputera()** - metoda zwracająca najlepszy możliwy ruch komputera - korzysta z metody MiniMax lub AlfaBeta

- **Ruch(char &gracz)** - metoda realizująca wykonanie ruchu. Gdy graczem aktualnie zagrywającym jest człowiek, pobiera informację o polu, w które gracz chce wstawić kółko, gdy graczem jest komputer, wywołuje metodę RuchKomputera
- **CzyscPlansze()** - metoda czyszcząca kratownicę ze wszystkich znaków, jakie zostały tam wpisane
- **ObslugaGry()** - jedyna, poza konstruktorem, publiczna metoda klasy KolkoIKrzyzyk, która realizuje obsługę gry użytkownika z komputerem

1.1 MiniMax

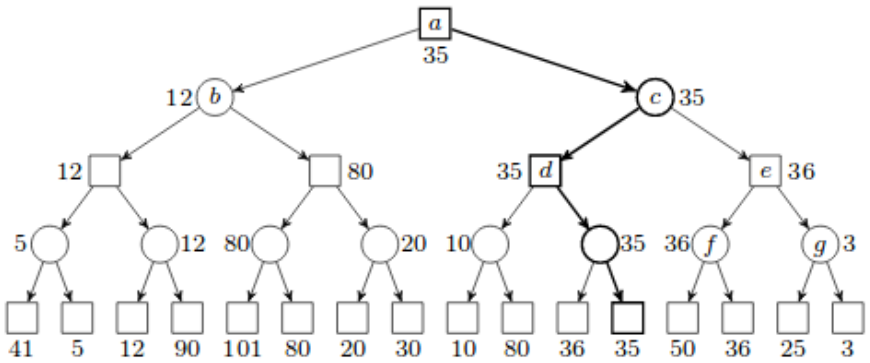
Wprowadzenie elementu sztucznej inteligencji do gry jest związane z koniecznością stworzenia *drzewa gry*. Przykład takiego drzewa przedstawia rysunek[1] poniżej. Przy każdym z liści zaznaczona jest wartość funkcji oceny.



Rysunek 2: Przykładowe drzewo gry

Z tak wyznaczonym drzewem staramy się znaleźć najlepszą możliwą drogę do wygranej (czyli do liścia o największej wartości funkcji oceny), nasz przeciwnik stara się nas powstrzymać, sprowadzając na drogę do najmniejszej wartości funkcji oceny. Wartość, do jakiej dojdziemy z danego wierzchołka możemy policzyć "od dołu" - □ maksymalizujemy wartość, a w ◯ minimalizujemy.

Wartości otrzymane dla przykładowego drzewa gry przedstawiono na rysunku poniżej, zaznaczając jednocześnie optymalną rozgrywkę obu graczy.



Rysunek 3: Przebieg optymalnej rozgrywki dla przykładowego drzewa gry

[1] Grafika zaczerpnięta z *https : //www.mimuw.edu.pl/ pan/gry.pdf*

1.2 Alfa-Beta cięcia

Przy obliczaniu wartości MiniMax nie potrzebujemy przeglądać niektórych wierzchołków drzewa. Istnieje ogólna metoda, która pozwoli sprawdzić, których wierzchołków nie musimy już przeglądać. W tym celu wystarczy lekko zmodyfikować algorytm MiniMax tak, by nie zawsze zwracał dokładne wartości. Metoda ta wymaga podania jeszcze dwóch parametrów - alfa i beta. Przedział $(alfa, beta)$ będziemy nazywać *oknem wywołania funkcji*.

$$w \leqslant \alpha \Rightarrow MiniMax \leqslant w$$

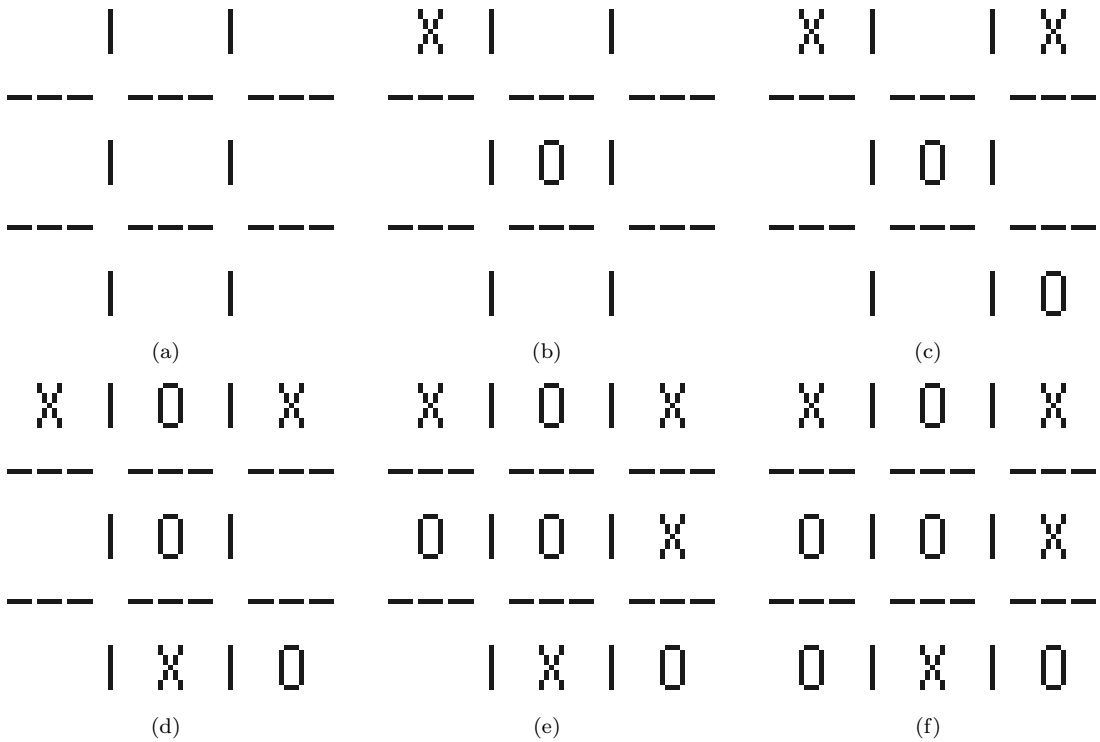
$$\alpha < w < \beta \Rightarrow MiniMax = w$$

$$\beta \leqslant w \Rightarrow MiniMax \geqslant w$$

Chcąc uzyskać dokładną wartość dla wierzchołka, z którego rozpoczynamy przeszukiwanie, przyjmujemy $\alpha = -\infty$ oraz $\beta = \infty$. Dla synów danego wierzchołka możemy modyfikować wartości tych parametrów. Jeśli jesteśmy w wierzchołku *max* i znaleźliśmy ruch, którego wartość jest większa niż β, możemy w tym momencie zakończyć przeszukiwanie i zwrócić znalezioną wartość. W ten sposób wykonujemy β-cięcie. Analogicznie wprowadzamy α-cięcie w wierzchołku *min*.

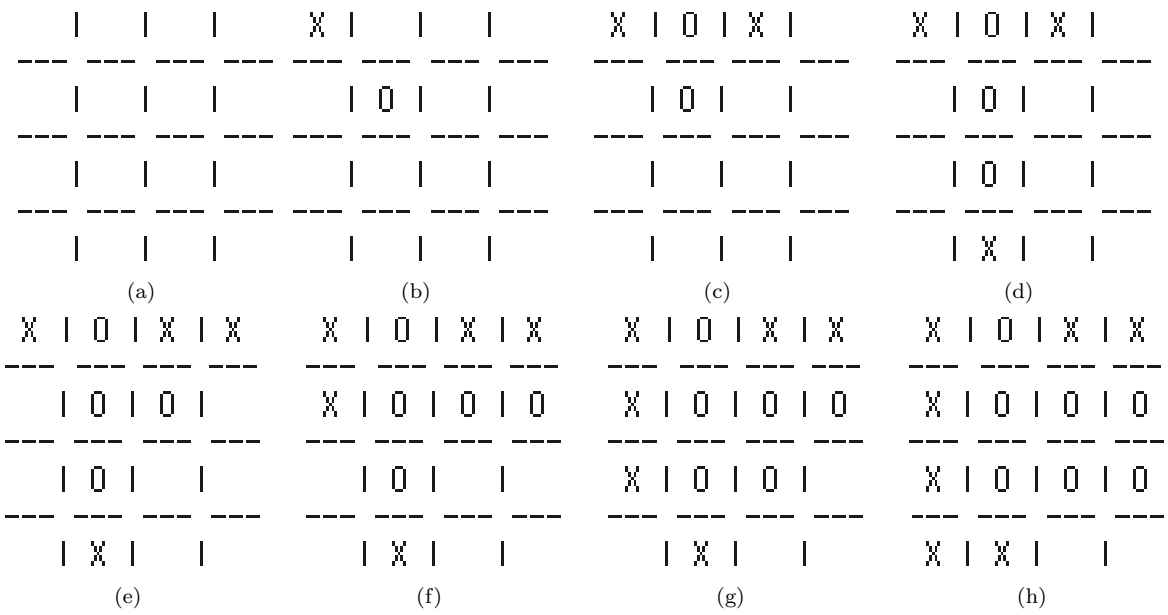
2 Przebieg gry

Dla planszy $3x3$ uruchomiono grę z komputerem. Przebieg tej gry przedstawiono poniżej.



Rysunek 4: Przebieg gry

Grę przeprowadzono też na większej kratownicy $4x4$. Przebieg zawarto w grafice poniżej.



Rysunek 5: Przebieg gry

3 Uwagi i wnioski

Sztuczna inteligencja realizowana przez algorytmy MiniMax oraz AlfaBeta rozgrywa z użytkownikiem partię, której człowiek nie jest w stanie wygrać, co najwyżej może zremisować.

Stosowanie algorytmu MiniMax do wyznaczania najlepszego ruchu dla komputera działa w szybkim tempie dla planszy $3x3$, jednak dla planszy o większym rozmiarze - $4x4$ nie radzi sobie. Dla usprawnienia działania zostało wprowadzenie usprawnienie alfa-beta cięć. W tym przypadku program działa w zadowalającym tempie dla kratownicy $4x4$, ale większe plansze wymagają znacznie dłuższego czasu na podjęcie decyzji o ruchu komputera.

Algorytm alfa-beta cięć mógłby być znacznie usprawniony, na przykład poprzez wprowadzenie kontroli głębokości wywołań rekurencyjnych, haszowanie Zobrista czy zmniejszanie okna wywołania. Wprowadzenie któregokolwiek z usprawnień niewątpliwie pozwoliłoby na przyspieszenie działania programu.

Przy tworzeniu drzewa korzystano z zasobów dostępnych w Internecie:

$$https://www.mimuw.edu.pl/\ pan/gry.pdf$$
$$http://lukasz.jelen.staff.iiar.pwr.edu.pl/downloads/files/wyklad11www.pdf$$
$$https://pl.wikipedia.org/wiki/Algorytm\_min-max$$
$$https://pl.wikipedia.org/wiki/Algorytm\_alfa-beta$$
$$http://eduinf.waw.pl/inf/utils/002\_roz/p012.php$$
$$http://www.algorytm.org/praktyka/kolko-krzyzyk.html$$

Kod programu udostępniony pod adresem:

$$https://github.com/234780/PAMSI3$$