Projekt 4

Gra ze sztuczną inteligencją

PAMSI

Projektowanie algorytmów i metody sztucznej inteligencji

Prowadzący: **mgr. inż. Magdalena Skoczeń**
Termin zajęć: **Środa $18^{\overline{55}}$**

Piotr Niedziółka, 249023

08.04.2020

1 Wprowadzenie

Celem projektu było utworzenie prostej gry, a także stworzenie do niej sztucznej inteligencji, która miała opierać się na algorytmie Min-Max. Spośród możliwych gier do stworzenia, wybrałem na początku Warcaby, jednak miałem problem z zaimplementowaniem algorytmu i punktowaniem, postanowiłem więc zrobić Kółko i Krzyżyk. W grze gracz ma możliwość definiowania rozmiaru pola, wraz z ilością znaków w rzędzie potrzebnych do wygrania.

2 Opis tworzonej gry

a) **Warcaby**

Gra polega na zbiciu wszystkich pionków, lub doprowadzenia do zablokowania pionków przeciwnika. W mojej implementacji pionek ma możliwość poruszania się w 4 kierunkach, lecz może bić tylko jeden pionek przeciwnika na raz. Bicie nie jest obowiązkowe, a także nie występują damki w grze.

Miałem problem z zaimplementowaniem algorytmu, a także ze zliczaniem punktów dla poszczególnych pionków - możliwe było branie jednego pionka i poruszanie się nim do przodu i tyłu, w nieskończoność, dlatego zrezygnowałem z tej gry.

Całość kodu znajduje się w repozytorium, pod tym linkiem:
https://github.com/Piotr601/PAMSI/tree/master/Projekt4/Warcaby

Całość kodu znajduje się w pliku **Warcaby.cpp**

b) **Kółko i Krzyżyk**

Gra rozgrywana przez dwóch graczy, która polega na objęciu trzech pól w jednej linii. Najczęściej gra się na polach o wymiarach 3x3, gdzie jedno pole może być zajęte tylko przez jednego gracza. W mojej implementacji użytkownik jest w stanie wybrać na jakiej tablicy ma się odbywać ta gra, a także ile znaków w rzędzie potrzeba do wygranej. W menu dostępna jest opcja z drugim graczem, a także z komputerem - gdzie można wybrać kto zaczyna.

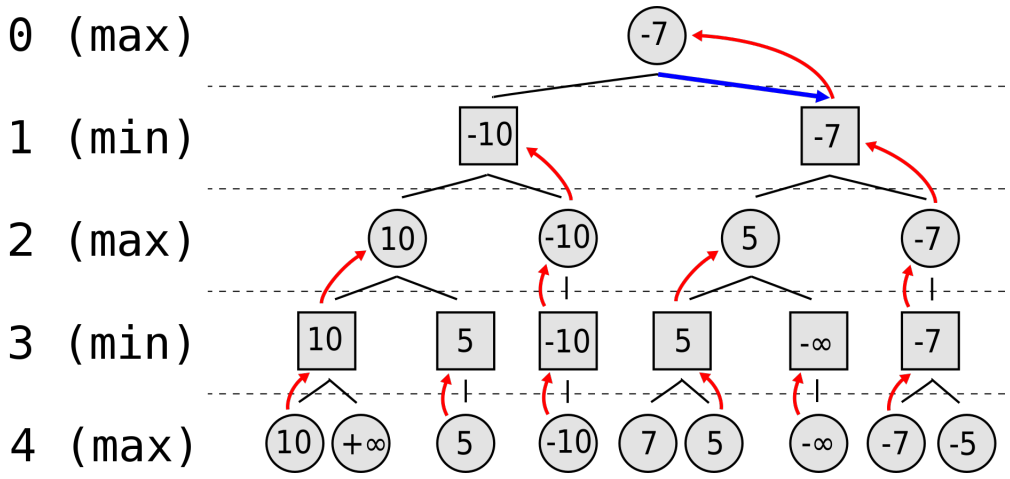
Całość kodu znajduje się w repozytorium, pod tym linkiem:
https://github.com/Piotr601/PAMSI/tree/master/Projekt4/KolkoKrzyzyk

Całość kodu znajduje się w pliku **main.cpp**

3 Omówienie algorytmu

W zaimplementowanej przeze mnie grze użyto algorytmu **Min-Max**, który jest jednym z najbardziej podstawowych do tworzenia sztucznej inteligencji w grach. Obecnie stosuje się już sieci neuronowe, a także uczenie maszynowe.

Metoda Min-Max polega na minimalizowaniu maksymalnych możliwych strat, alternatywnie maksymalizację minimalnego zysku. Algorytm ten głównie używa się w grach dwuosobowych, gdzie gracze wykonują ruchy naprzemiennie, lub jednocześnie - szachy, warcaby, kółko i krzyżyk. Algorytm przeszukuje wszystkie możliwe rozwiązania, a następnie wybiera to najbardziej optymalne w danym momencie.



Rysunek 1: Schemat minimax

Pseudokod:

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

4 Podsumowanie

W zaimplementowanym przeze mnie algorytmie, gdy zaczyna komputer, gracz jest w stanie zremisować lub przegrać. Gdy zaczyna gracz możliwe jest wygranie z komputerem, zremisowanie lub przegranie. Dzieje się tak, ponieważ algorytm nie bierze na początku środkowego pola - gdy jest wolne - co daje najwięcej kombinacji i możliwości. Lecz algorytm blokuje ruchy gracza, tak by nie miał możliwości szybkiego wygrania.

Po przeprowadzonych testach udało mi się osiągnąć powyższe rezultaty:
Gdy zaczyna gracz: **wygrana**, **remis**, **przegrana**
Gdy zaczyna komputer: **remis**, **przegrana**

Przy rozmiarach tablicy większej niż 3x3, gra z komputerem była niemożliwa - za dużo czasu zajmowało zliczanie wszystkich możliwych ruchów. Aby temu zapobiec należałoby użyć algorytmu Alfa-Beta. Gra z dwoma graczami jest możliwa na dowolnym rozmiarze tablicy, przy nieujemnej ilości symboli w rzędzie prowadzących do wygrania.

Napotkane przeze mnie trudności polegały głównie na sprawdzaniu planszy w poszukiwaniu wygranej. Pierwszą z nich, było niepoprawne sprawdzanie, gdy nie było 3 znaków z rzędu w jednej linii. Zliczało ono wszystkie przekątne, pion i poziom, aż do znalezienia wolnego znaku, a więc w poniższych przypadkach wskazywało wygraną.

	X	X
O		X
O		

	X	X
X		
	O	O

	X	O
X		O
X		

Zostało to naprawione i po testach wszystko działa jak należy.

Drugą było złe działanie komputera - traktował on ruch gracza, jako swój. Po przeanalizowaniu kodu, okazało się, że algorytm sprawdzał ruch gracza, zostało to szybko naprawione zamieniając w algorytmie zmienną *gracz1* na *gracz2*.

5 Wnioski

Algorytm Min-Max dla dużych tablic potrzebuje bardzo dużo czasu, na znalezienie wszystkich możliwych ruchów, a także na wybranie tego najbardziej optymalnego. Dla większych tablic (większych niż 3x3) zalecane jest użycie algorytmu Alfa-Beta, który jest rozbudowanym Min-Maxem - niektóre gałęzie drzewa są odcinane, przy zachowaniu najbardziej optymalnego wynik i drogi.

6 Bibliografia

- www.stackoverflow.com
- https://en.wikipedia.org/wiki/Minimax
- https://pl.wikipedia.org/wiki/Algorytm_min-max
- http://lukasz.jelen.staff.iiar.pwr.edu.pl
- https://bit.ly/2M4BDzu
- https://athena.ecs.csus.edu/~gordonvs/Beijing/Minimax.pdf
- https://www.javatpoint.com/mini-max-algorithm-in-ai