Projektowanie Algorytmów i Metody Sztucznej Inteligencji				
Projekt 3	Prowadzący	Mgr inż. Marta Emirsajłow	Termin	Pt, 7:30 – 9:00
	Wykonał	Amadeusz Janiszyn 249013	Data	26.05.2020

1. Wprowadzenie

Założeniem tego projektu było stworzenie gry opartej o strategię Min-Max. Algorytm był badany na podstawie gry *Kółko – krzyżyk,* zmodyfikowanej w taki sposób, by można było grać na planszy kwadratowej o dowolnym rozmiarze większym lub równym 3 oraz różnej ilości wygrywających znaków w rzędzie.

2. Algorytm minimax z obcięciem alfa-beta

Algorytm przeszukujący, redukujący liczbę węzłów, które muszą być rozwiązywane w drzewach przeszukujących przez algorytm min-max. Jest to przeszukiwanie wykorzystywane w grach dwuosobowych, takich jak właśnie kółko i krzyżyk lub szachy.
Warunkiem stopu jest znalezienie przynajmniej jednego rozwiązania czyniącego obecnie badaną opcję ruchu gorszą od poprzednich opcji. Wybranie takiej opcji ruchu nie przyniosłoby korzyści graczowi ruszającemu się, dlatego też nie ma potrzeby przeszukiwać dalej gałęzi drzewa tej opcji. Ta technika pozwala zaoszczędzić czas poszukiwania bez zmiany wyniku działania algorytmu.
Obcięcia alfa – beta zostały wprowadzone, ponieważ program potrzebował bardzo dużo czasu na obliczenie wszystkich możliwych rozwiązań dla rozmiaru planszy większych od 3. Osobna funkcja zwracająca głębokość przeszukiwania w drzewie algorytmu też miała duży wpływ na optymalizację działania algorytmu oraz skrócenie czasu obliczeniowego.

3. Budowa programu

Najważniejsze moduły aplikacji:

- *Board.h* – Zawiera definicję klasy *Board,* oraz najpotrzebniejsze metody wykorzystywane do badania przebiegu gry takie jak: *CheckWinner(), IsMoveLeft().*
- *AI.h* – Zawiera funkcję algorytmu - *minimax(…),* funkcję znajdującą najlepszy możliwy ruch – *BestMove*(…), funkcję do dynamicznego ustawiania głębokości przeszukiwania - *_Depth(…),* oraz funkcję zwracającą indeksy ruchu komputera do wstawienia na planszę gry – *CPut(…).*
- *Player.h* – Zawiera funkcje potrzebne do wstawienia ruchu pobranego od gracza. *GetMove(…)* jest funkcją rekurencyjną która wywołuje sama siebie w momencie napotkania błędu ze strony użytkownika w przypadku gdy indeks wyjdzie poza rozmiar planszy lub gdy gracz chce wstawić znak w zajęte miejsce. Funkcja *Put(…)* wstawia na planszę znak gracza. Umożliwia grę we 2 osoby.
- *Move.h* – Zawiera strukturę przechowującą numer kolumny oraz wiersza potrzebne do wstawienia ruchu. Zawiera przeciążone operatory przesunięć bitowych.
- *Interface_handling.h* – zawiera pomniejsze funkcje, które ułatwiają użytkownikowi komunikację z programem.
- *main.cpp* – Zastosowano funkcję *Driver(…)* w celu zbadania czasu potrzebnego na wykonanie pierwszego ruchu komputera w 2 przypadkach – kiedy zaczyna gracz oraz kiedy zaczyna komputer. W tym pliku znajduje się również pętla odpowiedzialna za faktyczną rozgrywkę.

4. Wyniki

Rozmiar	Komputer wykonuje ruch jako pierwszy *[ms]*	Komputer wykonuje ruch jako drugi *[ms]*
3x3	198,01	22,11
4x4	28505,31	15321,14
5x5	56452,85	46364,82
6x6	20622,16	17592,10
7x7	2427,01	2211,66
8x8	8139,21	7455,41
9x9	22881,08	22616,89
10x10	645,08	598,14
15x15	11728,45	11687,54

Głębokość przeszukiwania była analizowana na podstawie rozmiaru planszy. Warunki:

- **Rozmiar 3** – głębokość poszukiwania *9*
- **Rozmiar [4;7)** - głębokość poszukiwania definiowana wzorem *9 – rozmiar + 1*
- **Rozmiar [7; 10)** – głębokość poszukiwania *3*
- Dla rozmiarów **10 i większych** - głębokość poszukiwania *2*

Takie rozwiązanie pozwoliło na zmniejszenie czasu oczekiwania na ruch komputera. W każdym wypadku komputer stara się zablokować ruch wygrywający gracza. Jednak wraz ze wzrostem rozmiaru spada ocena najlepszego ruchu dla komputera, to samo tyczy się w przypadku zmiany rozmiaru znaków w rzędzie potrzebnych do wygrania. Niestety ograniczenia sprzętowe uwarunkowały pewne rozwiązania.

Jednak dla rozmiarów *3x3*, *4x4*, nawet *5x5* nie da się pokonać komputera.

5. Wnioski

Trudności napotkane w trakcie pisania programu:

- Bez zastosowania obcięć alfa – beta, dla rozmiarów większych od 3 komputer potrzebował ogromną ilość czasu na przeszukanie wszystkich możliwości.
- Po zastosowaniu obcięć alfa – beta problem nie zniknął dla rozmiarów większych od 4 zatem trzeba było wprowadzić dynamiczną ocenę głębokości przeszukiwania która została omówiona w poprzednim podpunkcie.
- Ostatnią trudnością którą nie udało się rozwiązać jest pełna walidacja wpisywania danych o ruchu przez gracza. Program sprawdza czy indeksy nie wyszły poza rozmiar oraz sprawdza czy w danym miejscu nie stoi już jakiś znak. Jednak w momencie wpisania czegoś innego niż liczba, program zatrzymuje się.

Zastosowany algorytm świetnie się sprawdza dla mniejszych rozmiarów. Wszystko jest jednak podyktowane przez zastosowanie różnej głębokości w celu zmniejszenia czasu obliczeniowego. Dla mniejszych rozmiarów komputer jest nie do pokonania. Obcięcia alfa – beta pomogły zredukować czas obliczeniowy, jednak to nie wystarcza dla większych rozmiarów.

6. Literatura

- GeeksForGeeks, *Minimax Algorithm in ame Theory | Set 4 (Alfa-Beta Prunning).* https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/
- GeeksForGeeks, *Minimax Algorithm in Game Theory | Set 3 (Finding optimal move)*. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/