**Data:** 06.2022

**Justyna Krzemień, 259226**

|  |  |
| --- | --- |
| **Kurs:** | Projektowanie algorytmów i metody sztucznej inteligencji |
| **Dane prowadzącego:** | Mgr inż. Marta Emirsajłow |
| **Termin zajęć:** | poniedziałek 17:05 |

Projekt 3: Gry & AI

Wprowadzenie

W ramach projektu należało wykonać wybraną grę z podanych, najlepiej opartą o algorytm MinMax.

Wybrano Kółko i Krzyżyk. Gra miała posiadać możliwość wybrania rozmiaru planszy (np. 3x3, 4x4 itd.) oraz ilości znaków w jednym rzędzie potrzebnej do wygranej.

Opis badanego algorytmu

Minimax (czy też MinMax) – algorytm drzewowy, wywodzi się on z teorii gry o sumie zerowej. W grze dwuosobowej, w której ruchy graczy występują naprzemiennie, każdego ruchowi przypisuje się wartość (jest ona dodatnia, gdy dany ruch będzie korzystny dla komputera, i ujemna, gdy tak nie jest). Ruch jest wybierany tak, aby dla danego gracza zminimalizować straty lub zmaksymalizować zyski (możliwie minimalizując zysk drugiego gracza). Algorytm przewiduje *n* następnych ruchów i je porównuje ze sobą. W zależności od tego czy w danym momencie dąży do maksymalizacji czy minimalizacji wybierany jest ruch o wartości największej lub najmniejszej.

Dodatkowo usprawniono algorytm o **cięcia alfa-beta**.

Cięcia alfa-beta – metoda pomijania gałęzi drzewa ruchów, jeśli nie wpływają na wynik. Aby to osiągnąć dodane zostają zmienne alfa i beta. Alfa jest na początku ustawiona na najmniejszą wartość na danej głębokości drzewa, a beta – największą. Przykładowo, alfa jest nadpisywana, gdy komputer znajdzie zysk w danym ruchu lepszy od alfy. Znajdując natomiast ruch o mniejszym zysku niż nadpisana już alfa, rezygnuje ze sprawdzania ruchów drugiego gracza po takim ruchu dla siebie, gdyż jest on mniej opłacalny niż ten wartości równej alfie, odcinając w ten sposób konkretne poddrzewo. Podobnie jest z betą, ale w drugą stronę, gdy szukamy ruchu najgorszego dla przeciwnika, nadpisywane są jak najmniejsze wartości, a odrzucamy ruchy o większym zysku oponenta. Zastosowanie cięć pozwala na szybsze wykonywanie się Minimax, gdyż nie tracimy czasu na przeszukiwanie niektórych poddrzew.

Wnioski

Ilość pól planszy w jednym rzędzie została ograniczona do 6, ponieważ przy tym rozmiarze algorytm znacznie wolniej działa, podobnie byłoby z każdym coraz większym rozmiarem, gdzie czas algorytmu jeszcze bardziej by się wydłużał. Duży wpływ na szybkość działania algorytmu ma wybór głębokości (depth). Z obserwacji wynikło, że dla różnego rozmiaru tablicy głębokość nie powinna być taka sama. Najbardziej optymalną wartością jaką udało mi się dobrać, pozwalającą na w miarę płynną rozgrywkę na jak największej możliwej kombinacji rozmiaru tablicy i znaków potrzebnych do wygranej, jest depth=10-size (zmienna size określa rozmiar tablicy).

Algorytm został zrealizowany w formie, w której przede wszystkim zwiększa on swoje szanse na wygraną, a dopiero w drugiej kolejności ma na uwadze minimalizowanie szans gracza. Widać to na większych tablicach, gdy warunek wygranej jest równy wartości zmiennej size. Ustawiając znaki cały czas w jednym tym samym rzędzie, algorytm blokuje nam możliwość wygranej dopiero tuż przed ostatnim ruchem zapewniającym nam zwycięstwo. To rozwiązanie ma swoje wady, jednakże z braku czasu oraz bądź co bądź, spełniania zakładanych funkcji, został pozostawiony w takiej formie.

Dodanie cięć alfa/beta widocznie przyspieszyło działanie programu, co było zauważalne przy większych tablicach jak np. 5x5.

Literatura

* Algorytm min-max - Wikipedia: <https://pl.wikipedia.org/wiki/Algorytm_min-max>
* GeeksforGeeks Minimax algorithm in game theory: <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/>

<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/>

* Algorithms Explained – minimax and alpha-beta prunning (film na YouTube): <https://www.youtube.com/watch?v=l-hh51ncgDI>