

W12n, AIR Projektowanie Algorytmów i Metody Sztucznej Inteligencji	Sprawozdanie nr 3 Temat: Kółko i Krzyżyk
Jakub Cebula	Wt 15:15, 5 czerwca 2023 Dr inż. Marek Bazan

Spis treści

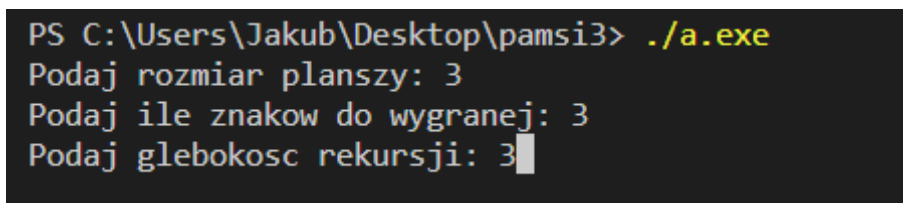
1 Treść zadania	2
2 Opis zadania	2
3 Algorytm minimax()	2
3.1 Działanie algorytmu	2
4 Optymalizacja (alpha-beta cięcia)	3
5 Testy	4
6 Wnioski	4
7 Bibliografia	5

1 Treść zadania

Należy zaimplementować grę w kółko i krzyżyk z wykorzystaniem algorytmu MinMax z alfa-beta cięciami. Gracz powinien posiadać możliwość definiowania rozmiaru pola (kwadratowego) wraz z ilością znaków w rzędzie.

2 Opis zadania

Spośród wymienionych w instrukcji zadania gier wybrano kółko i krzyżyk. Rozgrywane pomiędzy dwoma graczami, polega na naprzemiennym obejmowaniu przez nich pól kwadratowej planszy - wstawianiu znaku 'X' lub 'O' - w taki sposób, aby ustalona ilość tych samych znaków znajdowała się obok siebie w poziomej, pionowej bądź ukośnej linii. Na początku gry trzeba ustawić wymagania tj. wielkość planszy, ilość potrzebnych znaków do zwycięstwa oraz głębokość rekurencji algorytmu minimax.



```
PS C:\Users\Jakub\Desktop\pamsi3> ./a.exe
Podaj rozmiar planszy: 3
Podaj ile znakow do wygranej: 3
Podaj glebokosc rekursji: 3
```

Rysunek 1: Konfiguracja

3 Algorytm minimax()

Algorytm AI służy do wykonywania optymalnych ruchów i przyjmuje takie argumenty:

- depth - głębokość rekurencji
- alpha, beta - zmienne wykorzystane do alfa-beta cięcia
- isMaximizingPlayer - wskazuje, na to który gracz wykonuje ruch, czy jest on maksymalizujący czy minimalizowany.

3.1 Działanie algorytmu

1. Sprawdza, czy osiągnięto maksymalną głębokość przeszukiwania (depth) lub zakończono grę (gameOver()). Jeśli tak, zwraca wartość oceny (evaluate()).
2. Jeśli aktualny gracz jest maksymalizujący (isMaximizingPlayer == true), to dla każdego możliwego ruchu komputera (pustego pola na planszy) wykonuje następujące kroki:
 - Wykonuje ruch komputera na danym polu.
 - Wywołuje rekurencyjnie funkcję minimax dla gracza minimalizującego, zmniejszając depth o 1 i aktualizując wartość alfa (alpha) i beta.
 - Aktualizuje najlepszą wartość (bestValue) i wartość alfa (alpha) na podstawie wartości zwróconej z rekurencyjnego wywołania.
 - Jeżeli $\beta \leq \alpha$, wykonuje cięcie alfa-beta (przerywa pętlę).
 - Zwraca najlepszą wartość (bestValue).
3. Jeśli aktualny gracz jest minimalizujący (isMaximizingPlayer == false), to dla każdego możliwego ruchu gracza (pustego pola na planszy) wykonuje podobne kroki jak w przypadku maksymalizującego gracza, ale aktualizuje najlepszą wartość (bestValue) i wartość beta (beta).
4. Na końcu zwraca najlepszą wartość (bestValue), która reprezentuje ocenę ruchu komputera.

```

130 int minimax(int depth, int alpha, int beta, bool isMaximizingPlayer) {
131     if (depth == 0 || gameOver()) return evaluate();
132
133     if (isMaximizingPlayer) {
134         int bestValue = numeric_limits<int>::min();
135         for (int i = 0; i < boardSize; i++) {
136             for (int j = 0; j < boardSize; j++) {
137                 if (board[i][j] != EMPTY) continue;
138
139                 board[i][j] = COMPUTER;
140
141                 int value =
142                     minimax(depth - 1, alpha, beta, !isMaximizingPlayer);
143                 bestValue = max(bestValue, value);
144                 alpha = max(alpha, value);
145
146                 board[i][j] = EMPTY;
147
148                 if (beta <= alpha) break;
149             }
150             if (beta <= alpha) break;
151         }
152         return bestValue;
153     }
154     else { // minimalizacja szans gracza
155         int bestValue = numeric_limits<int>::max();
156         for (int i = 0; i < boardSize; i++) {
157             for (int j = 0; j < boardSize; j++) {
158                 if (board[i][j] != EMPTY) continue;
159
160                 board[i][j] = HUMAN;
161
162                 int value =
163                     minimax(depth - 1, alpha, beta, isMaximizingPlayer);
164                 bestValue = min(bestValue, value);
165                 beta = min(beta, value);
166
167                 board[i][j] = EMPTY;
168
169                 if (beta <= alpha) break;
170             }
171             if (beta <= alpha) break;
172         }
173         return bestValue;
174     }
175 }

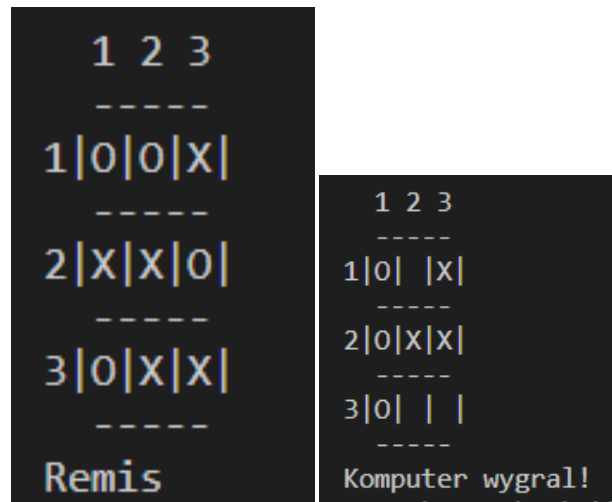
```

Rysunek 2

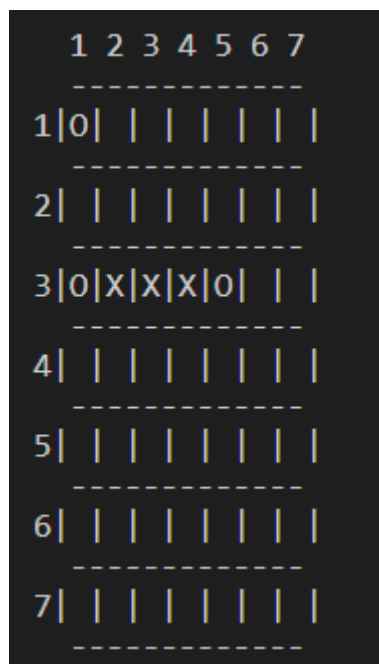
4 Optymalizacja (alpha-beta cięcia)

Algorytm minimax z alfa-beta cięciami pozwala na znaczne zmniejszenie liczby przeszukiwanych węzłów w drzewie gry poprzez eliminację niepotrzebnych gałęzi. W każdej iteracji, gdy osiągnięte zostaną odpowiednie warunki cięcia ($\beta \leq \alpha$), algorytm przerywa przeszukiwanie i nie rozważa dalszych gałęzi, które nie wpłyną na wynik. Dzięki temu algorytm jest bardziej wydajny i może obsłużyć większe drzewa gry.

5 Testy



Rysunek 3



Rysunek 4

6 Wnioski

- Dla planszy 3x3 i 3 punktów do wygranej nie da się wygrać z komputerem.
- Czas wykonywania algorytmu minimax silnie zależy od rozmiaru planszy, ale także ustawionej głębokości rekursji i ilości znaków w rzędzie- w przeciwieństwie do 3x3, dla większych planszy komputer nie wykonywał ruchów niezwłocznie i na ruch trzeba było czekać nawet kilkadziesiąt sekund,
- Optymalizacja alfa-beta cięciami znacznie poprawiła czas wykonywania ruchu przez komputer.

7 Bibliografia

1. <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>
2. https://pl.wikipedia.org/wiki/Algorytm_min-max
3. https://pl.wikipedia.org/wiki/Algorytm_alfa-beta
4. Michale T. Goodrich, Roberto Tamassia, David M. Mount. Data Structures and Algorithms in C++. Second Edition. John Wiley & Sons, Inc.