Politechnika Wrocławska

Raport
Mikroprojekt 3
Projektowanie i Analiza Algorytmów

31 maja 2023

Filip Mirski nr indeksu 264267

1 Wprowadznie

Celem ćwiczenia było stworzenie gry "Kółko krzyżyk" z wybieralnym przez gracza rozmiarem planszy i warunkiem wygranej. Do tego celu został zastosowany algorytm MiniMax z cięciami alfa, beta oraz funkcja ewaluacyjna, które zostaną opisane poniżej.

```cpp
void Game::AITurn(std::vector<std::vector<char>>& board, char sign, char opSign)
{
    int bestScore = INT_MIN;
    int bestRow = -1;
    int bestCol = -1;

    for (size_t i = 0; i < size; i++) {
        for (size_t j = 0; j < size; j++) {
            if (isValidMove(i, j)) {
                board[i][j] = sign;
                int score = minimax(board, 0, false, sign, opSign, INT_MIN, INT_MAX);
                board[i][j] = ' ';
                if (score > bestScore) {
                    bestScore = score;
                    bestRow = i;
                    bestCol = j;
                }
            }
        }
    }
    if (bestRow != -1 && bestCol != -1) {
        board[bestRow][bestCol] = sign;
    }
}
```

Rysunek 1: Fragment kodu z funkcji odpowiadającej za ruch komputera

2 Algorytm minimax z cięciami alfa beta

Algorytm minimax służy do znajdywania najlepszego ruchu dla komputera. Działa ze złożonością obliczeniową $O((n^2-1)!)$, gdzie N to ilość wierszy lub kolumn planszy; jego złożoność pamięciowa to $O(n^2)$. Natomiast, aby usprawnić działanie algorytmu stosuje się cięcia alfa beta, które w najlepszym wypadku mogą zmniejszyć złożoność obliczeniową do poziomu $O(k^(D/2))$, gdzie K to ilość pól na planszy, ergo ilość gałęzi, które algorytm musi rozpatrzeć. Najgorszy przpadek dalej będzie miał taką samą złożoność obliczeniową co sam minimax.

```cpp
int Game::minimax(std::vector<std::vector<char>>& board, int depth, bool isMaximizingPlayer, char sign, char opSign, int alpha, int beta)
{
    if (depth == maxDepth || checkWin(sign) || checkWin(opSign) || checkDraw()) {
        return evaluate(board, sign, opSign);
    }

        int bestRow = -1;
        int bestColumn = -1;

        if (isMaximizingPlayer) {
            int bestScore = INT_MIN;
            for (size_t i = 0; i < size; i++)
            {
                for (size_t j = 0; j < size; j++) {
                    if (isValidMove(i, j)) {
                        board[i][j] = sign;
                        int score = minimax(board, depth + 1, false, sign, opSign, alpha, beta);
                        board[i][j] = ' ';
                        bestScore = std::max(bestScore, score);
                        bestRow = i;
                        bestColumn = j;
                        alpha = std::max(alpha, bestScore);
                        if (alpha >= beta) {
                            break;
                        }
                    }
                }
                if (alpha >= beta) {
                    break;
                }
            }
            return bestScore;
```

Rysunek 2: Fragment kodu z algorytmem minimax i cięciami alfa beta

3 Funkcja ewaluacyjna

W celu zapewnienia poprawnego funckjonowania i płynności rozgrywki dodaje się heurystyczne funckje ewaluacyjne. Polegają na tym, żeby zapobiec "przebieganiu" drzewa możliwości przez komputer i ocenieniu najlepszej możliwości na danym poziomie przeszukiwania. Poniżasza funkcja właśnie coś takiego prezentuje, jej działanie polega na tym, że bierze maksymalną dopuszczalną głębokość przeszukiwania, która jest wyliczana ze wzoru $\frac{100}{n^2}$ i jeżeli funkcja minimax dojdzie do tej głębokości to przechodzi do funkcji ewaluacyjnej, która zlicza ile jest znaków komputera i przeciwnika oraz, którą ścieżką najlepiej podążać. W innych grach, takich jak na przykład szachy pozycje można obliczać na podstawie wartości pola i jego pionka. Złożoność takiego algorytmu to: O(1). Natomiast skraca algorytm minimax do $O(D \cdot (N^2 - 1))$, gdzie D to jest głębokość do której algorytm może przeszukiwać.

```
int Game::evaluate(std::vector<std::vector<char>>& board, char sign, char opSign)
{
    // Check rows
    for (size_t i = 0; i < size; i++) {
        int aiCount = 0;
        int opponentCount = 0;
        for (size_t j = 0; j < size; j++) {
            if (board[i][j] == sign) {
                aiCount++;
            }
            else if (board[i][j] == opSign) {
                opponentCount++;
            }
        }
        if (aiCount == winCondition) {
            return 100;  // AI wins
        }
        else if (opponentCount == winCondition) {
            return -100;  // Opponent wins
        }
    }

    // Check columns
    for (size_t j = 0; j < size; j++) {
        int aiCount = 0;
        int opponentCount = 0;
        for (size_t i = 0; i < size; i++) {
            if (board[i][j] == sign) {
                aiCount++;
            }
            else if (board[i][j] == opSign) {
                opponentCount++;
            }
        }
        if (aiCount == winCondition) {
```

Rysunek 3: Fragment kodu z funkcją ewaluacyjną

4 Wnioski

ALgorytm minimax jest szeroko stosowanym algorytmem w świecie gier. Niestety jego zapotrzebowanie na zasoby komputera jest bardzo duże, dlatego dodatki takie jak cięcia alfa beta oraz funkcje ewaluacyjne oceniające wartość pozycji (ruchu) są niesamowicie istotne i potrzebne do zaistnienia takiej gry (rozgrywki). Najlepszym przykładem może być gra w szachy, gdzie ilość możliwości ruchu komputera jest ogromna, przez co silnik gry wymaga nałożenia ograniczenia, a wtedy do gry wchodzą te dwa algorytmy, natomiast one same też mogą być modyfikowane w zależności od zapotrzebowania i gry.

5 Bibliografia

- Stuart Russell "Artificial Intelligence: A Modern Approach (4th ed.)." Prentice Hall, 2020. ISBN 0-13-461099-7