Politechnika Wrocławska

# Projektowanie i analiza algorytmów

# Projekt 4

ArtsyKimiko

13 maja 2024

# 1 Wstęp

„Kółko i krzyżyk” to nieskomplikowana gra dla dwojga, w której gracze na przemian umieszczają na planszy wybrane wcześniej symbole: kółka lub krzyżyki. Gra kończy się, gdy plansza jest zapełniona, albo gdy jeden z graczy ułoży swoje symbole w linii (liczba symboli zależy od wcześniej ustalonych zasad). W pierwszym przypadku mamy remis, a w drugim zwycięża ten, kto ułoży linię ze swoich symboli.

# 2 Opis algorytmów

Poniżej przedstawiono najważniejsze funkcje gry wraz z ich omówieniem.

## 2.1 Algorytm MinMax

Algorytm MinMax, stosowany w grach zero-jedynkowych, służy do podejmowania decyzji przez sztuczną inteligencję. Jego celem jest znalezienie optymalnego ruchu dla gracza maksymalizującego (Max) przy założeniu, że przeciwnik (Min) gra optymalnie. Algorytm rekurencyjnie tworzy drzewo możliwych stanów gry, gdzie węzły reprezentują konfiguracje planszy, a krawędzie ruchy gracza. Max symuluje ruch Min i odwrotnie. Wygrana Max zwraca +1, wygrana Min -1, a remis 0. Algorytm MinMax jest deterministyczny, co oznacza, że przy tych samych warunkach początkowych zawsze zwróci ten sam ruch bota.

Dla zwiększenia efektywności używa się cięcia alfa-beta, które zmniejsza liczbę przeszukiwanych węzłów. Algorytm ocenia węzły rekurencyjnie, aktualizując wartości alfa i beta. Węzły i ich poddrzewa, które nie mogą poprawić tych wartości, są odcinane, ponieważ nie wpływają na ostateczną decyzję.

```
Move Game::minMax(int depth, Player player, int alpha, int beta, int maxDepth)
{
    if (checkWin(BOT)) return { -1, -1, 10 - depth };
    if (checkWin(HUMAN)) return { -1, -1, depth - 10 };
    if (isDraw() || depth == maxDepth) return { -1, -1, 0 };

    vector<Move> moves;
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < size; ++j)
            if (board[i][j] == NONE)
                moves.push_back({ i, j, 0 });

    Move bestMove;
    if (player == BOT)
    {
        int bestVal = NEG_INF;
        for (auto& move : moves)
        {
            board[move.row][move.col] = BOT;
            move.score = minMax(depth + 1, HUMAN, alpha, beta, maxDepth).score;
            board[move.row][move.col] = NONE;
            if (move.score > bestVal)
            {
                bestVal = move.score;
                bestMove = move;
            }
            alpha = max(alpha, bestVal);
            if (beta <= alpha)
                break;
        }
    }
    else
    {
        int bestVal = INF;
```

```
        for (auto& move : moves)
        {
            board[move.row][move.col] = HUMAN;
            move.score = minMax(depth + 1, BOT, alpha, beta, maxDepth).score;
            board[move.row][move.col] = NONE;
            if (move.score < bestVal)
            {
                bestVal = move.score;
                bestMove = move;
            }
            beta = min(beta, bestVal);
            if (beta <= alpha)
                break;
        }
    }
    return bestMove;
}
```

## 2.2 Funkcja sprawdzająca wygraną

Funkcja `checkWin` przeszukuje planszę w poszukiwaniu zwycięskiego ciągu symboli gracza. Najpierw sprawdza wiersze i kolumny, a następnie przekątne. Dla wierszy i kolumn używa funkcji `checkLine`, która sprawdza ciągi symboli gracza. Jeśli którakolwiek z linii zawiera zwycięski ciąg o długości `winLength`, funkcja zwraca `true`. Podobnie, przekątne są sprawdzane i zwracają `true`, jeśli znajdą zwycięski ciąg. Jeśli żadna linia nie zawiera zwycięskiego ciągu, funkcja zwraca `false`.

```
bool Game::checkWin(Player player)
{
    for (int i = 0; i < size; ++i)
        if (checkLine(i, 0, 0, 1, player))
            return true;
    for (int i = 0; i < size; ++i)
        if (checkLine(0, i, 1, 0, player))
            return true;
    for (int i = 0; i <= size - winLength; ++i)
    {
        if (checkLine(i, 0, 1, 1, player))
            return true;
        if (checkLine(0, i, 1, 1, player))
            return true;
        if (checkLine(i, size - 1, 1, -1, player))
            return true;
        if (checkLine(0, size - 1 - i, 1, -1, player))
            return true;
    }
    return false;
}
```

Funkcja `checkLine` dokładnie sprawdza pojedyncze linie (wiersze, kolumny, przekątne), zaczynając od określonych punktów i kierunków, zliczając kolejne symbole gracza. Zwraca `true`, gdy znajdzie ciąg o długości `winLength`, w przeciwnym razie `false`. Dzięki tej strukturze można skutecznie i efektywnie sprawdzić wynik gry.

```
bool Game::checkLine(int startRow, int startCol, int stepRow, int stepCol, Player
    player)
{
    int count = 0;
    for (int i = 0; i < size; ++i)
    {
        int row = startRow + i * stepRow;
        int col = startCol + i * stepCol;
        if (row >= size || col >= size || col < 0)
            break;
        if (board[row][col] == player)
        {
            count++;
            if (count == winLength)
                return true;
        }
        else
            count = 0;
    }
    return false;
}
```

# 3 Wnioski

W przypadku implementacji gry „Kółko i krzyżyk" wykorzystanie algorytmu MinMax z alfa-beta cięciami jest bardzo skuteczne, znacząco przyspieszając podejmowanie decyzji bez utraty dokładności. Cięcia alfa-beta zmniejszają liczbę przeszukiwanych węzłów, co poprawia wydajność algorytmu. Dzięki temu, nawet w pozornie prostych grach jak „Kółko i krzyżyk", algorytm MinMax z cięciami alfa-beta pozwala na szybkie i efektywne znajdowanie optymalnych ruchów.

# 4 Bibliografia


- eduinf.waw.pl
- geeksforgeeks.org
- wikipedia.com
- wikizmsi.zut.edu.pl