Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji
Kierunek: Informatyczne Systemy Automatyki

Projektowanie i analiza algorytmów

Michał Wróblewski
272488

Termin zajęć:
wtorek 13:15 - 15:00

18 maja 2024

Spis rysunków

Wprowadzenie

Niniejsze sprawozdanie przedstawia rozwój i implementację gry Kółko i Krzyżyk w języku C++. Gra oferuje możliwość gry z przeciwnikiem AI, który wykorzystuje algorytm Minimax z przycinaniem alfa-beta do podejmowania optymalnych ruchów.

Celem tego projektu jest stworzenie wymagającego przeciwnika dla gracza poprzez zastosowanie efektywnych algorytmów wyszukiwania.

Opis gry i technik SI

Gra Kółko i Krzyżyk jest prostą grą dla dwóch graczy, rozgrywaną na planszy o dowolnym rozmiarze. Gracze na przemian zaznaczają puste pola na planszy swoimi symbolami (X lub O).

Grę wygrywa ten, kto pierwszy umieści w rzędzie (poziomo, pionowo lub ukośnie) określoną liczbę swoich symboli. Jeśli cała plansza zostanie wypełniona, a żaden z graczy nie spełni tego warunku, gra kończy się remisem.

Implementacja gry

Gra została zaimplementowana w języku C++ z wykorzystaniem biblioteki SFML do wyświetlania grafiki i interakcji z użytkownikiem. Główne komponenty gry to:

- **Pętla gry**: Zarządza stanami gry i obsługą wejścia użytkownika.
- **Reprezentacja planszy**: Dwuwymiarowy wektor reprezentujący planszę do gry.
- **Generowanie ruchów AI**: Przeciwnik AI wykorzystuje algorytm Minimax do określenia najlepszego ruchu.

Algorytm Minimax z przycinaniem alfa-beta

Algorytm Minimax jest rekurencyjną metodą stosowaną do podejmowania decyzji w grach dla dwóch graczy. Oceni on wszystkie możliwe ruchy obu graczy i wybiera ten ruch, który maksymalizuje minimalny zysk AI (czyli minimalizuje potencjalną stratę).

Przycinanie alfa-beta jest techniką optymalizacyjną dla algorytmu Minimax, która zmniejsza liczbę ocenianych węzłów. Działa ona poprzez eliminację gałęzi w drzewie wyszukiwania, które nie muszą być przeszukiwane, ponieważ nie mogą one wpłynąć na ostateczną decyzję.

Kod algorytmu

```
int Game::minimax(char player, int depth, int maxDepth, bool isMaximizing,
↪  int alpha, int beta) {
    totalDepth += depth;
    numSearches++;
    minimaxCalls++;
    auto start = std::chrono::high_resolution_clock::now();
    char opponent = (player == 'O') ? 'X' : 'O';
```

```
// Sprawdzenie stanów terminalnych (wygrana/przegrana/remis)
if (checkWin(player)) {
    auto end = std::chrono::high_resolution_clock::now();
    minimaxTime += end - start;
    return (player == 'O') ? 100 - depth : depth - 100;
}
if (checkWin(opponent)) {
    auto end = std::chrono::high_resolution_clock::now();
    minimaxTime += end - start;
    return (opponent == 'O') ? 100 - depth : depth - 100;
}
if (isBoardFull() || depth == maxDepth) {
    auto end = std::chrono::high_resolution_clock::now();
    minimaxTime += end - start;
    return 0; // Remis lub osiągnięto maksymalną głębokość
}

if (isMaximizing) {
    int bestScore = INT_MIN;
    for (int i = 0; i < boardSize; i++) {
        for (int j = 0; j < boardSize; j++) {
            if (board[i][j] == ' ') {
                board[i][j] = 'O';
                int score = minimax('O', depth + 1, maxDepth, false,
                ↪  alpha, beta);
                board[i][j] = ' ';
                bestScore = std::max(score, bestScore);
                alpha = std::max(alpha, score);
                if (beta <= alpha) {
                    return bestScore;
                }
            }
        }
    }
    auto end = std::chrono::high_resolution_clock::now();
    minimaxTime += end - start;
    return bestScore;
} else {
    int bestScore = INT_MAX;
    for (int i = 0; i < boardSize; i++) {
        for (int j = 0; j < boardSize; j++) {
            if (board[i][j] == ' ') {
                board[i][j] = 'X';
                int score = minimax('X', depth + 1, maxDepth, true,
                ↪  alpha, beta);
                board[i][j] = ' ';
                bestScore = std::min(score, bestScore);
```

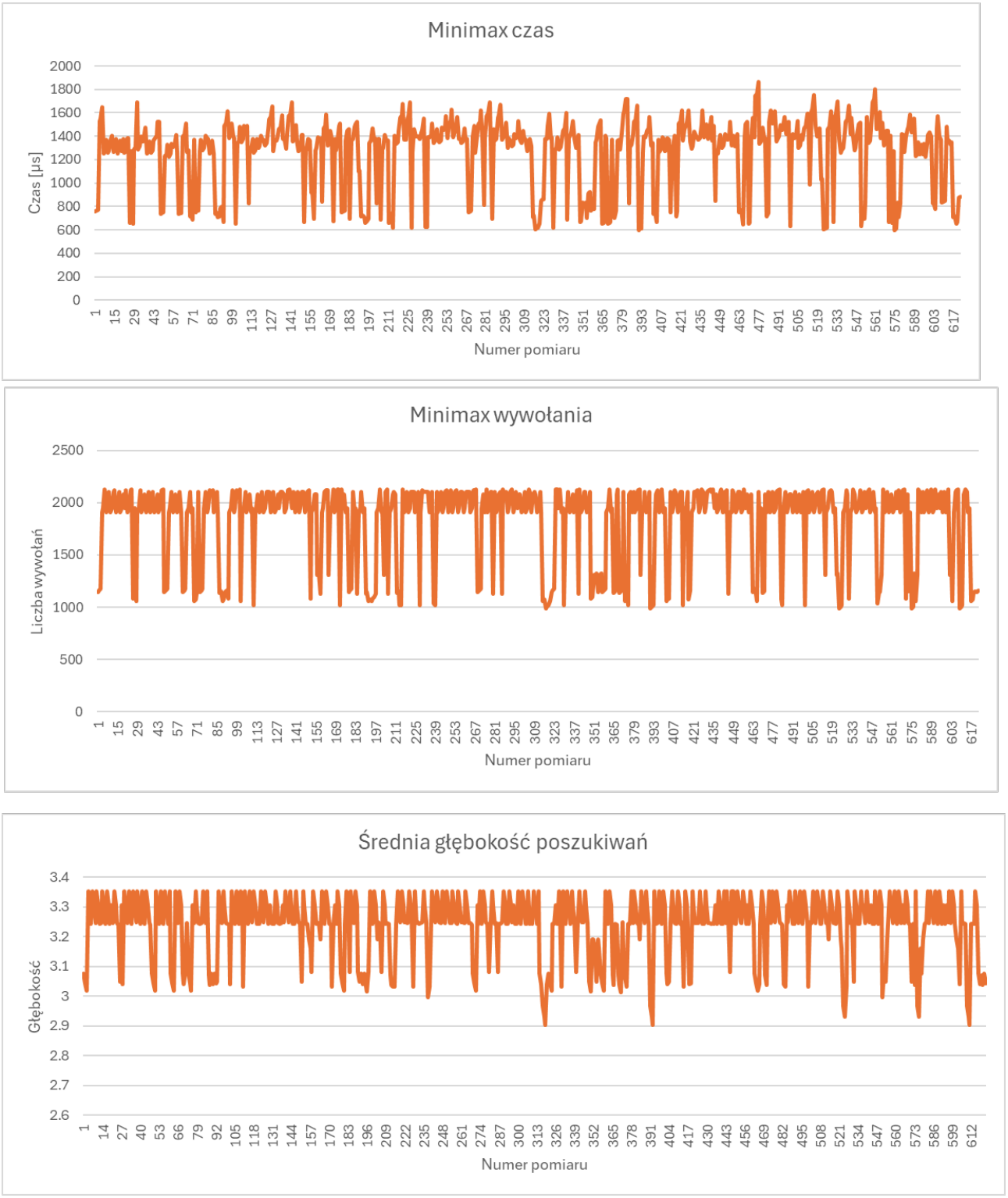
```
                    beta = std::min(beta, score);
                    if (beta <= alpha) {
                        return bestScore;
                    }
                }
            }
        }
        auto end = std::chrono::high_resolution_clock::now();
        minimaxTime += end - start;
        return bestScore;
    }
}
```

Analiza wydajności

Wydajność algorytmu Minimax z przycinaniem alfa-beta można analizować za pomocą kilku metryk:

- **Czas wykonania**: Całkowity czas potrzebny algorytmowi Minimax na obliczenie optymalnego ruchu.
- **Liczba wywołań Minimax**: Całkowita liczba wywołań funkcji Minimax podczas gry.
- **Średnia głębokość**: Średnia głębokość drzewa gry przeszukiwanego podczas podejmowania decyzji przez AI.

Wykres 1: Poziom łatwy algorytmu Minimax



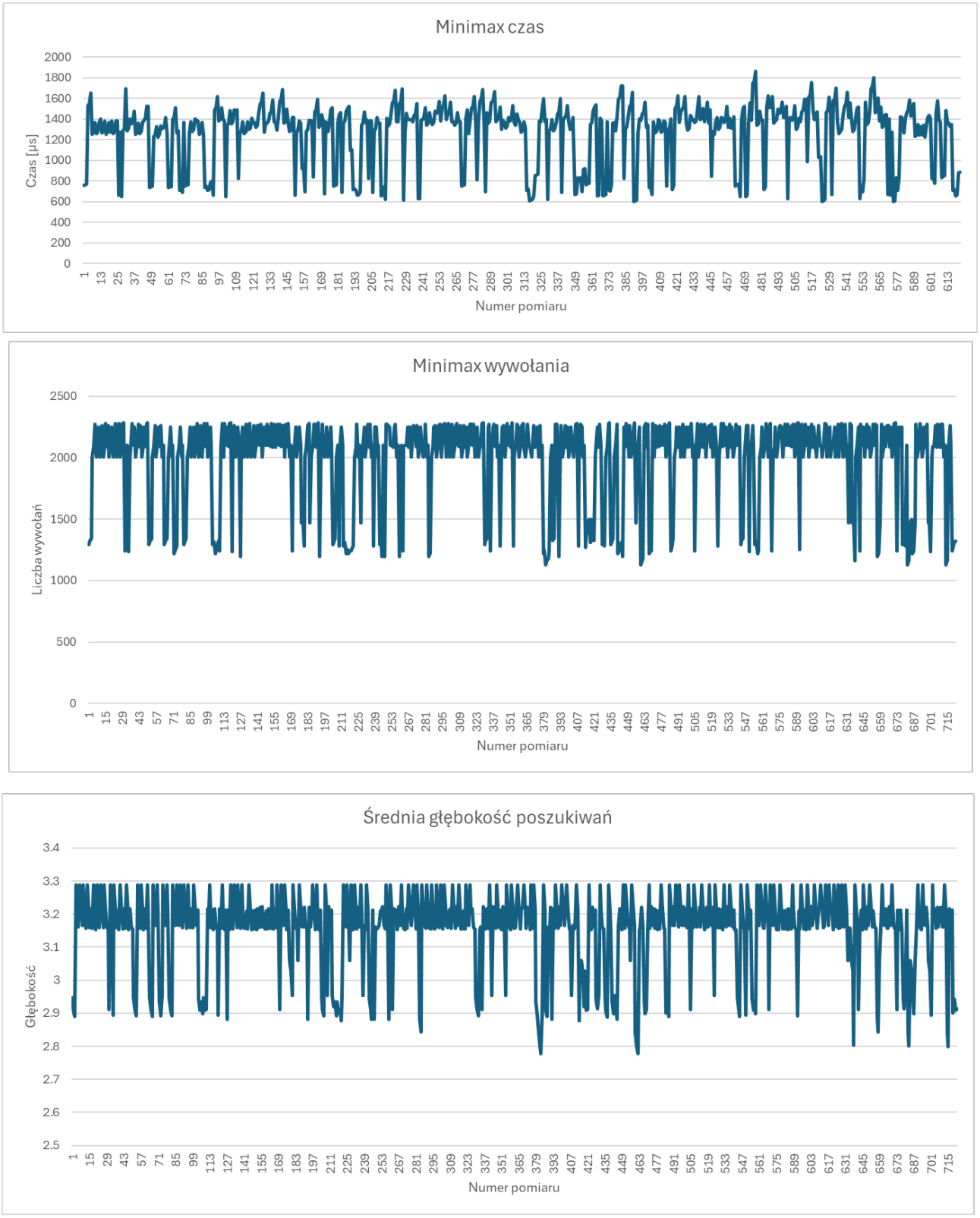
Rysunek 1: Poziom łatwy algorytmu Minimax

Opis: Pomiary wydajności dla poziomu łatwego algorytmu Minimax.

Wnioski:

- Czas wykonania: Algorytm potrzebuje stosunkowo mało czasu na obliczenia. Jest to spodziewane, ponieważ *poziom łatwy oznacza mniejszą głębokość przeszukiwania drzewa gry.*
- Liczba wywołań Minimax: *Liczba wywołań jest niska*, co potwierdza, że algorytm nie przeszukuje wielu stanów gry.
- Średnia głębokość: Średnia głębokość drzewa gry jest niska, co oznacza, że *decyzje są podejmowane na podstawie płytkich analiz.*

Wykres 2: Poziom średni algorytmu Minimax



Rysunek 2: Poziom średni algorytmu Minimax

Opis: Pomiary wydajności dla poziomu średniego algorytmu Minimax.

Wnioski:

- Czas wykonania: Czas potrzebny na obliczenia wzrasta w porównaniu do poziomu łatwego. Algorytm analizuje głębsze stany gry.
- Liczba wywołań Minimax: Liczba wywołań wzrasta, co wskazuje na bardziej skomplikowane przeszukiwanie drzewa gry.
- Średnia głębokość: Średnia głębokość drzewa gry jest większa niż w poziomie łatwym, co oznacza *bardziej dogłębne analizy*.

Wykres 3: Poziom trudny algorytmu Minimax

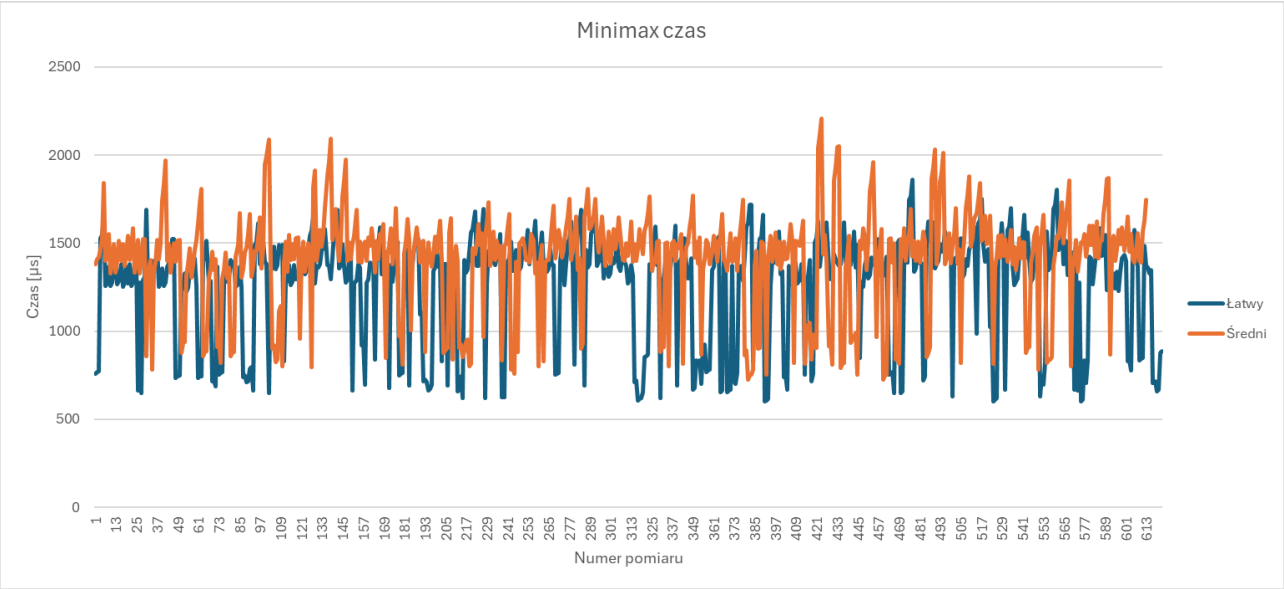
Rysunek 3: Poziom trudny algorytmu Minimax

Opis: Pomiary wydajności dla poziomu trudnego algorytmu Minimax.

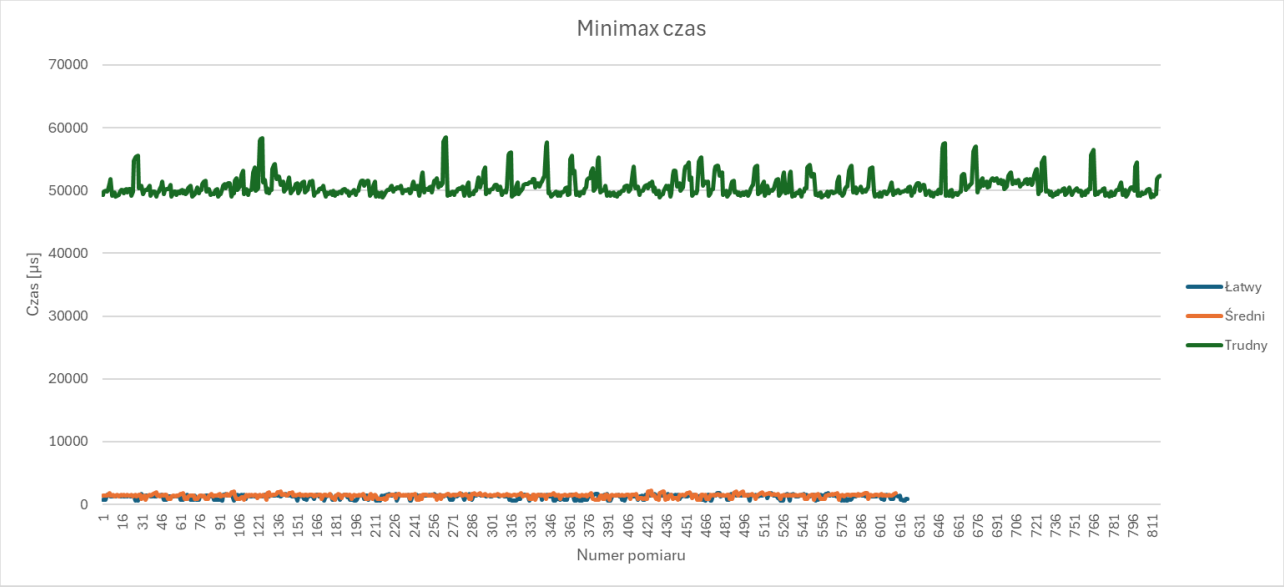
Wnioski:

- Czas wykonania: Czas wykonania jest najwyższy ze wszystkich poziomów, co jest wynikiem *najgłębszych analiz.*
- Liczba wywołań Minimax: Liczba wywołań funkcji Minimax jest najwyższa, co wskazuje na *największe przeszukiwanie drzewa gry.*
- Średnia głębokość: Średnia głębokość jest największa, co oznacza *najbardziej kompleksowe podejmowanie decyzji przez AI.*

Wykres 4 i 5: Porównanie poziomów algorytmu Minimax



Rysunek 4: Porównanie poziomów algorytmu Minimax



Rysunek 5: Porównanie poziomów algorytmu Minimax

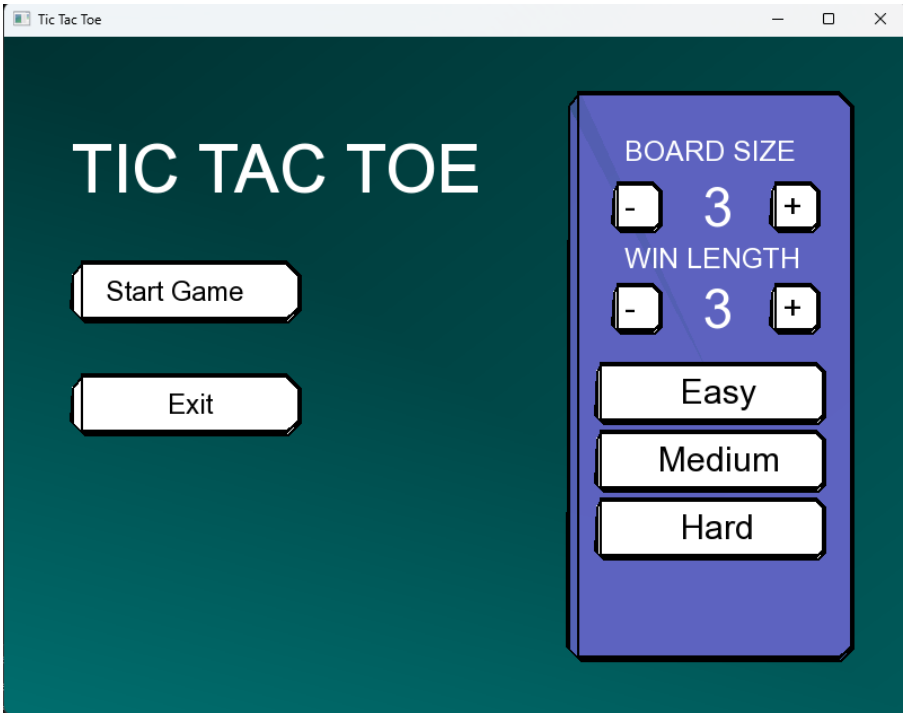
Opis: Porównanie pomiarów wydajności dla wszystkich trzech poziomów trudności algorytmu Minimax.

Wnioski:

- Czas wykonania: Wyraźnie wzrasta z poziomem trudności, co jest spowodowane *rosnącą złożonością przeszukiwania.*
- Liczba wywołań Minimax: Wzrasta z poziomem trudności, co potwierdza *bardziej skomplikowane obliczenia na wyższych poziomach.*

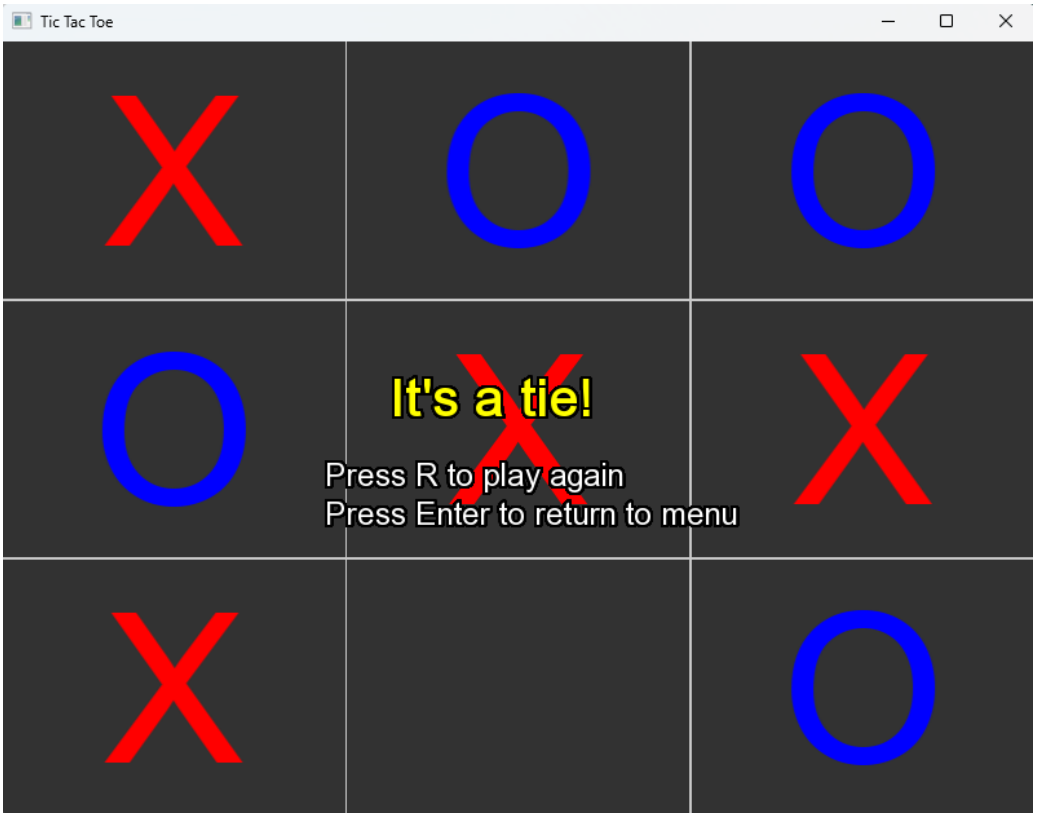
- Średnia głębokość: Rosnąca średnia głębokość drzewa gry wraz ze wzrostem poziomu trudności wskazuje na *bardziej złożone analizy AI na wyższych poziomach*.

Wyniki

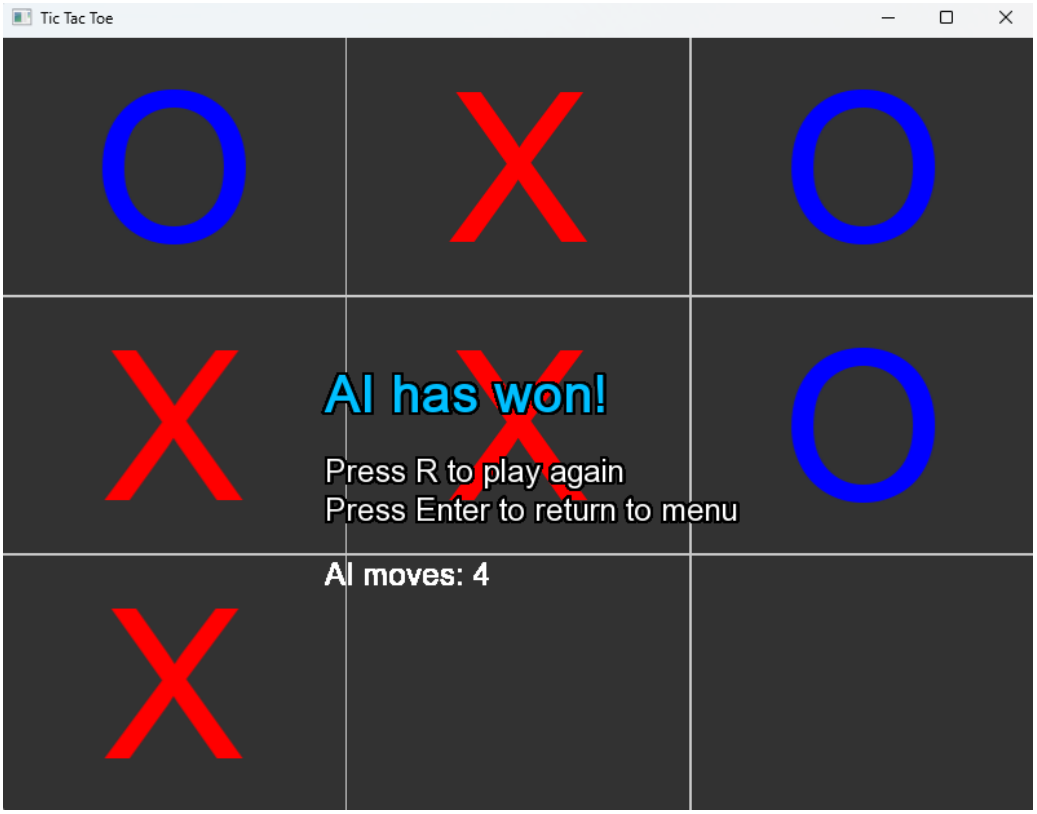


Rysunek 6: Strona początkowa

Rysunek 7: Wygrana gracza



Rysunek 8: Remis



Rysunek 9: Wygrana AI

Wnioski

- Na wyższych poziomach trudności algorytm staje się bardziej wymagający i angażujący dla gracza, co jest wynikiem głębszych analiz i większej liczby przeszukiwanych stanów gry.
- Przycinanie alfa-beta skutecznie optymalizuje proces przeszukiwania, zmniejszając liczbę koniecznych ocen stanów gry.

Podsumowanie

Gra Kółko i Krzyżyk zaimplementowana z algorytmem Minimax i przycinaniem alfa-beta demonstruje skuteczność tych technik w tworzeniu konkurencyjnego przeciwnika AI.

Implementacja algorytmu Minimax z przycinaniem alfa-beta w grze Kółko i Krzyżyk pokazuje, że jest on skuteczny i efektywny, *szczególnie na wyższych poziomach trudności*, gdzie znacząco zwiększa się liczba przeszukiwanych stanów gry oraz czas potrzebny na wykonanie obliczeń, co czyni grę bardziej wymagającą dla gracza.

Bibliografia

- Russell, S. J., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.

- Bratko, I. (2001). *Prolog Programming for Artificial Intelligence* (3rd ed.). Addison-Wesley.
- Millington, I., & Funge, J. (2009). *Artificial Intelligence for Games* (2nd ed.). CRC Press.