Pembuatan Bot Permainan Tic Tac Toe dengan Pendekatan Brute Force dan Minimax

Ahmad Nadil - 13521024
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13521024@std.stei.itb.ac.id

***Abstrak*—Permainan Tic Tac Toe merupakan sebuah permainan strategi yang melibatkan dua pemain yang cukup sederhana namun tetap menantang. Dalam penelitian ini, penulis akan merepresentasikan pendekatan *brute force* dan *minimax* untuk mengembangkan sebuah *bot* yang dapat bermain secara otomatis melawan pemain manusia. Pendekatan *brute force* dan *minimax* melibatkan explorasi semua kemungkinan langkah yang mungkin dalam permainan ini. *Bot* akan secara sistematis memeriksa setiap langkah yang dapat diambil, mempertimbangkan setiap kemungkinan akibatnya, dan memilih langkah terbaik berdasarkan evaluasi papan permainan saat ini, dan papan permainan saat *bot* mengambil langkah.**

Kata Kunci—Brute force; Minimax; Tic Tac Toe

I. PENDAHULUAN

Permainan Tic Tac Toe merupakan salah satu permainan strategi sederhana yang telah lama dikenal dan dimainkan oleh banyak orang di seluruh dunia. Dalam permainan ini, dua pemain bergantian menempatkan tanda mereka (biasanya direpresentasikan dengan tanda "X" dan "O") pada kotak dalam papan permainan 3x3. Tujuan permainan ini adalah untuk menciptakan garis horizontal, vertical, ataupun diagonal yang terdiri dari tiga tanda mereka sendiri.

Meskipun sederhana, Tic Tac Toe tetap menantang karena melibatkan pengambilan keputusan yang cerdas dalam rangka mencapai tujuan permainan. Keputusan yang diambil harus dapat mengevaluasi apakah kedepannya dapat membahayakan kemenangan atau tidak. Oleh karena itu, penulis memilih untuk melakukan pengembangan bot permainan Tic Tac Toe. Penulis menjadikan permainan ini sebagai subjek penelitian yang menarik dalam bidang kecerdasan buatan dan teori permainan yang cukup sederhana.

Dalam penelitian ini, penulis berfokus pada pengembangan bot permainan Tic Tac Toe yang mampu bermain secara otomatis melawan pemain manusia. Penulis akan menggabungkan dua buah pendekatan, yaitu pendekatan secara *brute force* dan pendekatan secara *minimax*.

Pendekatan *brute force* melibatkan eksplorasi kemungkinan Langkah yang dapat diambil dalam permainan Tic Tac Toe. Bot akan secara sistematis memeriksa setiap Langkah yang mungkin, mengevaluasi konsekuensi dari setiap langkah tersebut, dan memilih Langkah terbaik berdasarkan evaluasi papan permainan.

Pendekatan Minimax merupakan algoritma pengambilan keputusan yang berbasis teori permainan Pendekatan ini mengasumsikan bahwa kedua pemain dalam permainan bermain secara optimal. Bot akan menggunakan pendekatan Minimax dengan evaluasi heuristic untuk menentukan Langkah terbaik yang mengoptimalkan utilitas minimum untuk pemain yang membuat langkah tersebut. Dalam algoritma ini, akan dilakukan evaluasi secara rekursif untuk mengevaluasi kemungkinan dari setiap langkah dengan bergantian antara pemain yang memaksimumkan dan meminimumkan utilitas tersebut.

Makalah ini akan dijelaskan secara lebih rinci kedua pendekatan tersebut dan akan digambarkan dalam sebuah program sebagai implementasi bot permainan Tic Tac Toe serta menganalisis hasil eksperimen yang dilakukan.

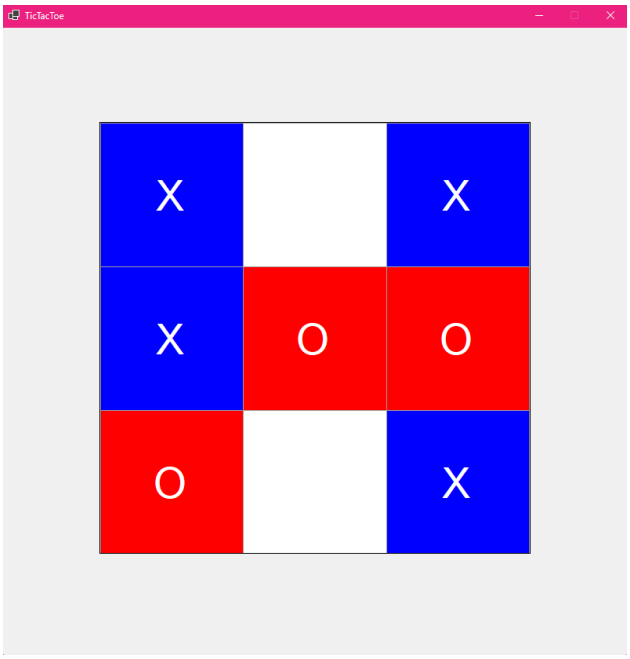
II. DASAR TEORI

A. *Permainan Tic Tac Toe*

Tic Tac Toe merupakan sebuah permainan strategi antara dua pemain yang dimainkan pada sebuah papan berukuran 3x3. Tujuan utama dari permainan ini adalah untuk menempatkan tiga tanda (biasanya direpresentasikan dengan tanda "X" dan "O") secara horizontal, vertikal, ataupun diagonal. Terdapat beberapa konsep dasar dari permainan ini, yaitu:

1. Papan Permainan

   Papan permainan dalam permainan Tic Tac Toe terdiri dari kotak berukuran 3x3. Setiap kotak dapat berisi tanda "X", "O", ataupun kosong. Pemain secara bergantian menempatkan tanda mereka pada kotak-kotak yang masih kosong.



Gambar 1. Permainan Tic Tac Toe
Sumber: Program TicTacToe Bot, Penulis

2. Giliran dan Pemain

   Tic Tac Toe dimainkan secara giliran antara dua pemain. Biasanya, satu pemain menggunakan tanda "X" dan pemain lainnya menggunakan tanda "O". Pemain X biasanya bermain pertama kali.

3. Strategi dan Langkah

   Setiap pemain memilih langkah dengan menempatkan tanda mereka di salah satu kotak kosong. Langkah yang dipilih harus mempertimbangkan tujuan jangka panjang untuk mencapat garis tiga tanda secara horizontal, vertikal, ataupun diagonal. Pemain harus berusaha untuk mencegah lawan mereka mencapai garis tiga tanda sebelum mereka.

4. Pemenang dan Seri

   Permainan berakhir ketika salah satu pemain berhasil membentuk garis tiga tanda atau jika seluruh kotak telah diisi tanpa ada garis tiga tanda. Jika seorang pemain berhasil membentuk garis tiga tanda, maka pemain tersebut adalah pemenangnya. Jika tidak ada pemain yang berhasil membentuk garis tiga tanda, maka permainan tersebut akan dianggap seri.

5. Keputusan Optimal

   Dalam Tic Tac Toe, permainan dengan keputusan yang sempurna akan selalu berakhir dengan seri jika kedua pemain bermain secara optimal. Namun jika ada kesalahan strategi dari salah satu pemain, maka pemain lainnya memiliki peluang untuk memenangkan permainan.

6. Keterbatasan Strategi

   Tic Tac Toe adalah permainan dengan kompleksitas yang cukup terbatas. Dalam permainan ini, terdapat jumlah kotak dan langkah yang terbatas. Oleh karena itu, mungkin untuk melakukan analisis terhadap semua kemungkinan langkah dan strategi.

B. Definisi Algoritma Brute Force

Algoritma *Brute Force* merupakan algoritma yang memiliki karakteristik yang lempang (*straightforward)* dalam memecahkan suatu persoalan. Algoritma ini memiliki pendekatan yang cukup sederhana tetapi kuat dalam menyelesaikan masalah. Hal ini dikarenakan, pada algoritma ini menguji setiap kemungkinan yang dapat muncul terhadap suatu permasalahan secara sistematis. Prinsip dasar dari algoritma ini adalah mencoba semua kemungkinan solusi untuk mencapai solusi yang diinginkan. Oleh karena itu, dalam pencariannya, akan melibatkan eksplorasi secara lengkap dari seluruh ruang pencarian solusi.

Pada umumnya, algoritma brute force dapat didasarkan pada pernyataan pada persoalan / *problem statement* serta definisi/konsep yang dilibatkannya. Algoritma *brute force* dapat memecahkan persoalan dengan cara yang sangat sederhana, secara langsung, dan dengan cara yang jelas.

C. Konsep Dasar Algoritma Brute Force

Terdapat beberapa konsep dasar yang diterapkan dalam algoritma *brute force*, antara lain adalah :

1. Eksplorasi Sistematis

   Algoritma *Brute Force* melakukan eksplorasi yang sistematis melalui semua kemungkinan solusi dengan menguji satu per satu. Hal ini akan menghasilkan dan memeriksa semua kemungkinan solusi yang ada, terlepas dari kualitas dari solusi tersebut.

2. Hasil yang Optimal

   Algoritma *Brute Force* dijamin akan menghasilkan solusi yang optimal, dikarenakan dapat mempertimbangkan semua kemungkinan yang dapat muncul.

3. Kompleksitas yang Buruk

   Salah satu kelemahan utama dari algoritma *Brute Force* adalah kompleksitas yang sangat besar. Jumlah total kemungkinan solusi yang harus diuji dan dievaluasi bisa sangat besar. Oleh karena itu, algoritma *brute force* mungkin tidak efisien dalam masalah dengan ruang pencarian yang sangat besar.

4. Optimalisasi

   Pada beberapa contoh kasus, kinerja algoritma *brute force* dapat ditingkatkan dengan mengimplementasikan teknik optimasi atau *pruning*, yang dapat mengurangi jumlah solusi yang harus dievaluasi. Contohnya adalah dengan menggunakan metode heuristic atau

memanfaatkan sifat khusus dari masalah yang sedang diselesaikan.

Dalam pemilihan algoritma, sangat penting untuk mempertimbangkan kembali jika menggunakan algoritma *brute force*. Meskipun algoritma ini mampu memberikan solusi yang benar dan optimal di kebanyakan kasus, kompleksitasnya yang sangat besar dapat membuatnya tidak efisien untuk masalah yang cukup besar. Oleh karena itu, pendekatan algoritma lain yang dapat memangkas kompleksitas pada ruang uji yang cukup besar, dapat dijadikan alternatif lain dari algoritma ini.

D. Definisi Algoritma Minimax

Algoritma *Minimax* merupakan sebuah algoritma pengambilan keputusan yang banyak digunakan dalam permainan dua pemain, terutama dalam konteks permainan nol-sum / zero-sum game, yaitu jenis permainan di mana keuntungan yang diperoleh oleh satu pemain secara langsung berbanding terbalik dengan kerugian yang dialami oleh pemain lainnya. Tujuan algoritma Minimax adalah untuk menentukan langkah terbaik yang harus diambil oleh pemain saat berada dalam situasi permainan tertentu.

E. Konsep Dasar Algoritma Minimax

Terdapat beberapa konsep dasar yang diterapkan dalam algoritma *Minimax*, antara lain adalah :

1. Permainan dengan nol-sum

   Algoritma *Minimax* biasanya diterapkan dalam permainan dengan dua pemain yang saling bersaing, dimana setiap keuntungan yang diperoleh oleh salah satu pemain diimbangi dengan kerugian yang sama oleh pemain lainnya. Dalam konteks ini, jika suatu pemain memenangkan poin, pemain lain akan kehilangan jumlah poin yang sama, sehingga total jumlah poin selalu tetap konstan.

2. Evaluasi Heuristik

   Pada setiap langkah dalam permainan, algoritma ini akan melakukan evaluasi heuristic terhadap setiap kemungkinan untuk mencapai keadaan terminal (kemenanga, kekalahan atau seri). Evaluasi ini memberikan skor atau utilitas yang menggambarkan seberapa baik posisi permainan untuk pemain yang mempertimbangkan langkah tersebut.

3. Maksimisasi dan Minimisasi

   Algoritma *Minimax* beroperasi secara bergantian antara pemain yang memaksimumkan utulitas (biasanya dianggap sebagai pemain “Max”) dan pemain yang meminimumkan utilitas (biasanya dianggap sebagai pemain “Min”). Pemain Max berusaha memaksimalkan skor, sedangkan pemain Min berusaha untuk meminimalkannya.

4. Rekursi dan Pengulangan

   Algoritma *Minimax* diterapkan secara rekursif. Selama rekursi, algoritma ini akan menghitung nilai *minimax* dengan suatu cara tertentu. Hal ini melibatkan pemanggilan rekursif untuk menghitung utilitas dan mempertimbangkan apakah pemain yang bergerak melakukan maksimisasi atau minimisasi.

5. Pemilihan Langkah Terbaik

   Setelah algoritma *minimax* menyelesaikan evaluasi heuristic pada semua kemungkinan, langkah terbaik akan dipilih berdasarkan nilai *minimax* yang dihasilkan. Pemain Max akan memilih langkah dengan skor utilitas tertinggi, sementara pemain Min akan memilih langkah dengan skor utilitas terendah.

Algoritma Minimax memberikan pendekatan yang sistematis serta optimal dalam pengambilan keputusan dalam permainan dua pemain. Meskipun algoritma ini dapat memberikan solusi yang optimal, kompleksitasnya meningkat seiring dengan pertumbuhan ruang lingkup permainan. Oleh karena itu, Teknik optimasi seperti alpha-beta *pruning* dapat digunakan untuk mengurangi jumlah evaluasi yang harus dilakukan dan meningkatkan efisiensi algoritma *minimax*.

III. HASIL PENELITIAN

Pada penelitian ini, penulis membuat sebuah program sederhana permainan tic tac toe berbasis GUI menggunakan bahasa pemrograman C#. Pada program ini, bot yang dibuat direpresentasikan dengan simbol “X” dan pemain dengan symbol “Y”.

A. Representasi Papan Permainan dalam Kelas Board

```
class Board{
    /* Attributes */
    public int[,] matrix;

    /* Constructor */
    public Board(){
        this.matrix = new int[3, 3];
        for(int i = 0; i < 3; i++){
            for(int j = 0; j < 3; j++){
                this.matrix[i, j] = -1;
            }
        }
    }

    /* Methods */
    public void move(int row, int col, int turn){
        /*
            -1 is empty
            O is 0
            X is 1
        */
        if (!isEmpty(row, col)){
            throw new Exception("Invalid move");
        }
        int value = turn % 2 == 1 ? 0 : 1;
        this.matrix[row, col] = value;
    }

    public Boolean isEmpty(int row, int col){
        return this.matrix[row, col] == -1;
    }

    public int checkWinner(){
        // Horizontal
```

```
        if (this.matrix[0, 0] == this.matrix[0, 1] && this.matrix[0, 1] == this.matrix[0, 2] && !isEmpty(0, 0)){
            return this.matrix[0, 0];
        } else if (this.matrix[1, 0] == this.matrix[1, 1] && this.matrix[1, 1] == this.matrix[1, 2] && !isEmpty(1, 0)){
            return this.matrix[1, 0];
        } else if (this.matrix[2, 0] == this.matrix[2, 1] && this.matrix[2, 1] == this.matrix[2, 2] && !isEmpty(2, 0)){
            return this.matrix[2, 0];
        }

        // Vertical
        else if (this.matrix[0, 0] == this.matrix[1, 0] && this.matrix[1, 0] == this.matrix[2, 0] && !isEmpty(0, 0)){
            return this.matrix[0, 0];
        } else if (this.matrix[0, 1] == this.matrix[1, 1] && this.matrix[1, 1] == this.matrix[2, 1] && !isEmpty(0, 1)){
            return this.matrix[0, 1];
        } else if (this.matrix[0, 2] == this.matrix[1, 2] && this.matrix[1, 2] == this.matrix[2, 2] && !isEmpty(0, 2)){
            return this.matrix[0, 2];
        }

        // Diagonal
        else if (this.matrix[0, 0] == this.matrix[1, 1] && this.matrix[1, 1] == this.matrix[2, 2] && !isEmpty(0, 0)){
            return this.matrix[0, 0];
        } else if (this.matrix[0, 2] == this.matrix[1, 1] && this.matrix[1, 1] == this.matrix[2, 0] && !isEmpty(0, 2)){
            return this.matrix[0, 2];
        }

        // Draw
        else {
            return  -1;
        }
    }

    public void resetGame(){
        this.matrix = new int[3, 3];
        for(int i = 0; i < 3; i++){
            for(int j = 0; j < 3; j++){
                this.matrix[i, j] = -1;
            }
        }
    }
}
```

Pada program yang dibuat oleh penulis, papan permainan direpresentasikan dengan sebuah matriks integer berukuran 3x3. Simbol X dan O disimpan dalam bentuk integer, yaitu 0 untuk O dan 1 untuk X. Pada kotak papan yang kosong, akan diisi oleh integer bernilai -1.

B. Algoritma Minimax dalam Penentuan Langkah Terbaik

```
public (int, int) getBestMove(){
        int bestScore = -1000;
        int moveRow = -1;
        int moveCol = -1;

        // Explore all cells in the board
        for (int row = 0; row < 3; row++){
            for (int col = 0; col < 3; col++){
                // Check if the cell is empty
                if (isEmpty(row, col)){
                    // Make the move
                    this.matrix[row, col] = 0;

                    // Compute evaluation function for this move.
                    int moveScore = minimax(0, false, -1000, 1000);

                    // Undo the move
                    this.matrix[row, col] = -1;

                    // If the result of the move is better than the best score, update bestScore, moveRow, and moveCol
                    if (moveScore >= bestScore){
                        bestScore = moveScore;
                        moveRow = row;
                        moveCol = col;
                    }
                }
            }
        }
        return (moveRow, moveCol);
    }

    private int minimax(int depth, bool isMaximizingPlayer, int alpha, int beta){
        int winner = checkWinner();
        if (winner != -1){
            return winner == 0 ? 10 : -10; // 10 for bot win, -10 for player win
        }

        // Tie
        bool isFull = true;
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++){
                if (isEmpty(i, j)){
                    isFull = false;
                    break;
                }
            }
            if (!isFull) break;
        }
        if (isFull) return 0; // Tie game returns score of 0

        if (isMaximizingPlayer){
            int bestScore = -1000;
            for (int i = 0; i < 3; i++){
                for (int j = 0; j < 3; j++){
                    if (isEmpty(i, j)){
                        this.matrix[i, j] = 0; // Bot's move
                        int score;
                        if (checkImmediateWin() == 1) { // If immediate win for bot, prioritize that
                            score = 1000;
                        } else {
                            score = minimax(depth + 1, false, alpha, beta);
                        }
                        this.matrix[i, j] = -1; // Undo move
                        bestScore = Math.Max(score, bestScore);
                        alpha = Math.Max(alpha, bestScore);
                        if (beta <= alpha) break; // Alpha Beta Pruning
                    }
```

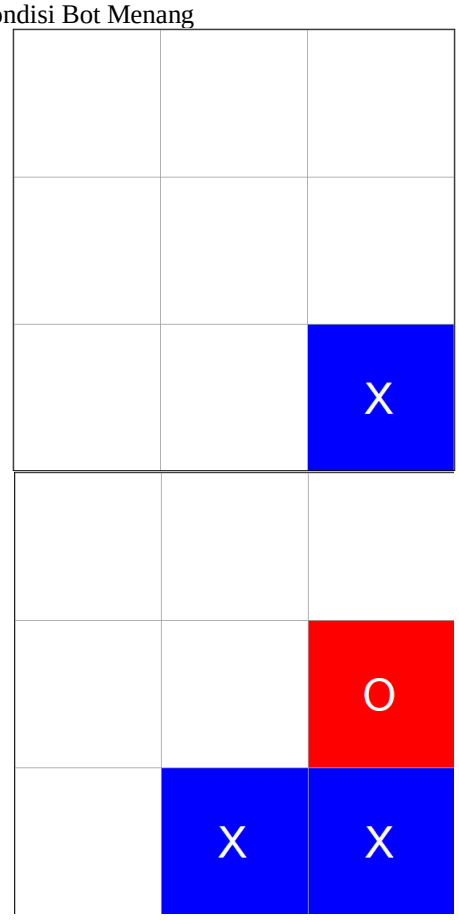
```
                }
            }
            return bestScore - depth; // Subtract
depth to prioritize faster wins
        } else {
            int bestScore = 1000;
            for (int i = 0; i < 3; i++){
                for (int j = 0; j < 3; j++){
                    if (isEmpty(i, j)){
                        this.matrix[i, j] = 1; //
Player's move
                        int score;
                        if (checkImmediateThreat()
== 1) { // If immediate win for player, prioritize
that
                            score = -1000;
                        } else {
                            score = minimax(depth +
1, true, alpha, beta);
                        }
                        this.matrix[i, j] = -1; //
Undo move
                        bestScore = Math.Min(score,
bestScore);
                        beta    =    Math.Min(beta,
bestScore);
                        if (beta <= alpha) break; //
Alpha Beta Pruning
                    }
                }
            }
            return bestScore + depth; // Add depth
to prioritize blocking the player faster
        }
    }
    private int checkImmediateThreat() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (this.matrix[i, j] == -1) { //
Check empty spots
                    this.matrix[i, j] = 1; // Assume
the player places a piece here
                    int winner = checkWinner(); //
Check if this would make the player win
                    this.matrix[i, j] = -1; // Reset
the cell
                    if (winner == 1) return 1; // If
the player could win, it's an immediate threat
                }
            }
        }
        return -1; // If no immediate threat found
    }

    private int checkImmediateWin(){
        for (int i = 0 ; i < 3 ; i++) {
            for (int j = 0 ; j < 3 ; j++) {
                if (this.matrix[i, j] == -1) { //
Check empty spots
                    this.matrix[i, j] = 0; // Assume
the bot places a piece here
                    int winner = checkWinner(); //
Check if this would make the bot win
                    this.matrix[i, j] = -1; // Reset
the cell
                    if (winner == 0) return 1; // If
the bot could win, it's an immediate win
                }
            }
        }
        return -1; // If no immediate win found
    }
```

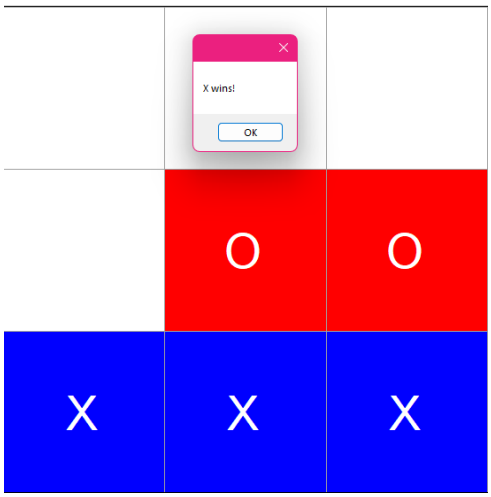
Pada potongan program diatas, diimplementasikan algoritma minmax dalam permainan Tic Tac Toe, berikut adalah alur kerja dari algoritma tersebut:

1. Fungsi *getBestMove* merupakan fungsi uytama yang digunakan untuk mendapatkan langkah terbaik yang harus diambil oleh bot.
2. Dalam fungsi *getBestMove*, variable *bestScore* diinisialisasi dengan nilai yang sangat rendah, yaitu -1000 untuk memulai pencarian nilai maksimum
3. Terdapat sebuah loop untuk melakukan eksplorasi semua kotak pada papan permainan secara *brute force*, dalam loop tersebut, setiap kotak akan diperiksa terlebih dahulu apakah kosong, jika kosong akan disimulasikan bahwa bot akan menempatkan pada posisi tersebut.
4. Setelah itu, fungsi *minimax* akan dipanggil untuk melakukan evaluasi langkah tersebut.
5. Nilai *moveScore* akan diperoleh dari pemanggilan, langkah tersebut akan mengubah nilai matriks kembali menjadi -1 untuk melakukan reset.
6. Akan dilakukan perbandingan antara *moveScore* dengan *bestScore*, lalu akan diambil nilai terbaik.
7. Setelah selesai mengiterasi seluruh papan permainan, akan dikembalikan posisi baris dan kolom terbaik.

C. Contoh Pengujian Program

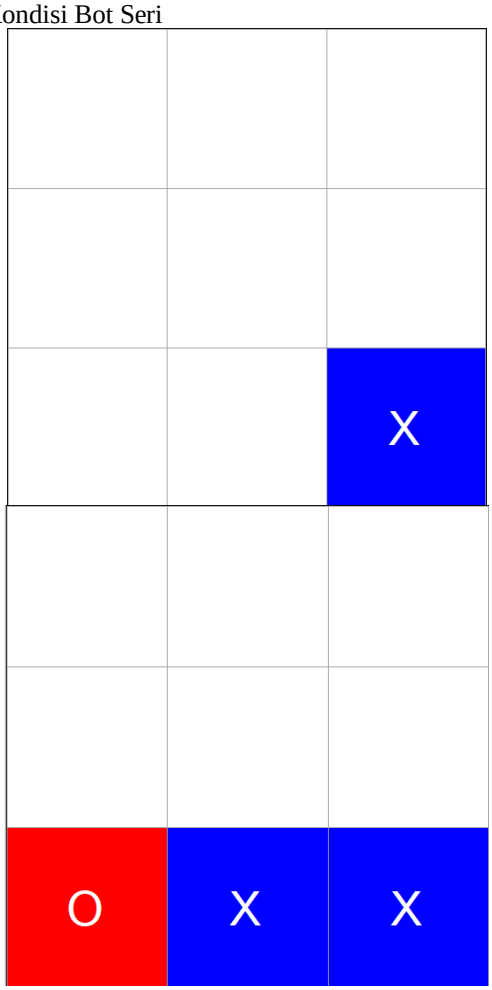
1. Kondisi Bot Menang

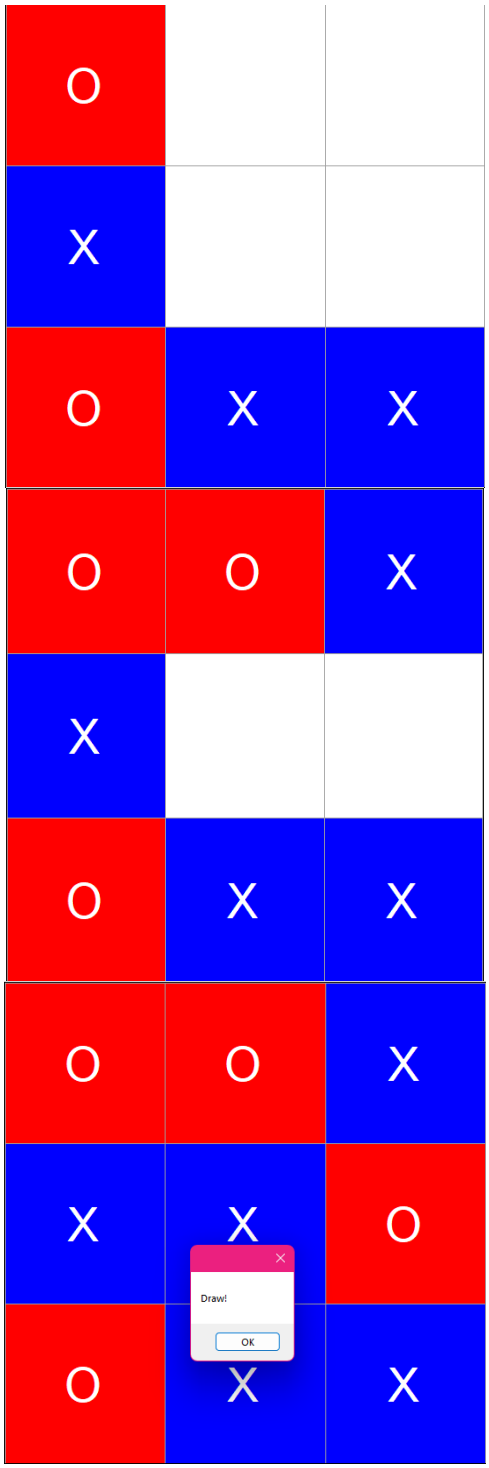




Gambar 2. Permainan Tic Tac Toe dimenangkan oleh Bot
Sumber: Program TicTacToe Bot, Penulis

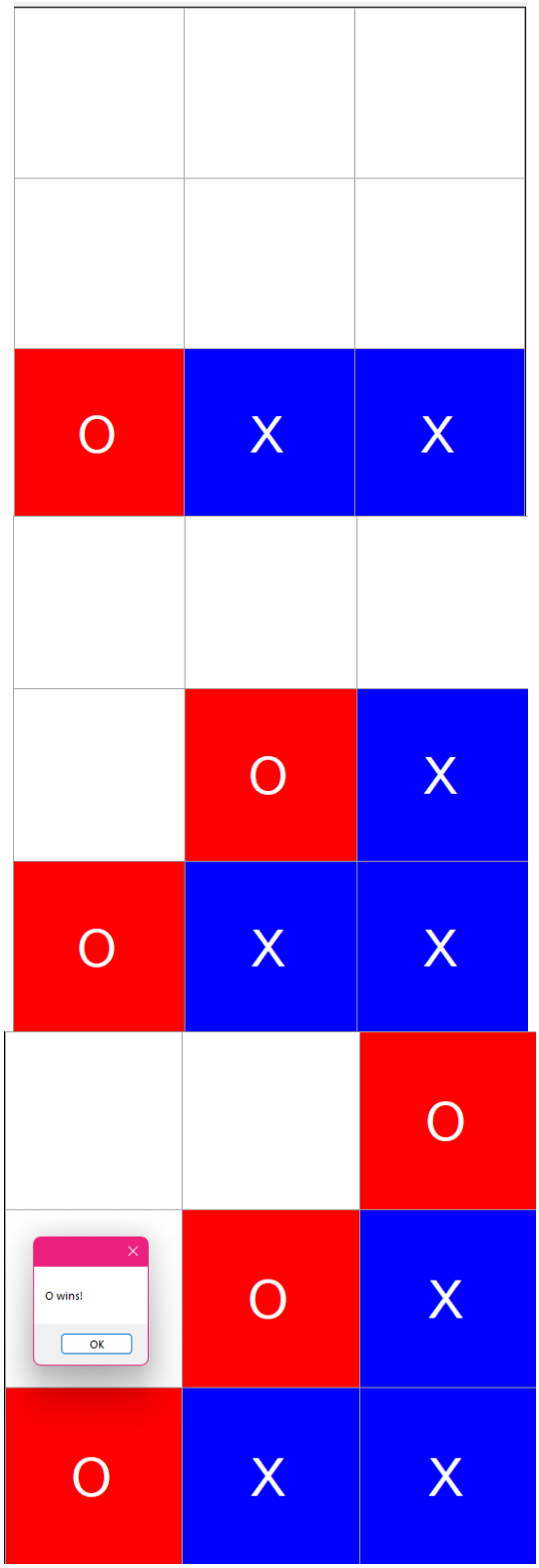
2. Kondisi Bot Seri





Gambar 3. Permainan Tic Tac Toe seri dengan Bot
Sumber: Program TicTacToe Bot, Penulis

3. Kondisi Bot Kalah



Gambar 4. Permainan Tic Tac Toe Bot mengalami kekalahan
Sumber: Program TicTacToe Bot, Penulis

KESIMPULAN

Dalam penelitian ini, penulis dapat mengimplementasikan algoritma *minimax* dan algoritma *brute force* dalam program permainan Tic Tac Toe sederhana.

Melalui hasil eksperimen yang dilakukan, yaitu tahap percobaan pada program, kami menemukan bahwa algoritma *minimax* dan *brute force* dapat memberikan hasil yang cukup optimal di kebanyakan kasus. Hasil optimal yang didapat adalah menang atau seri. Akan tetapi, pada beberapa kasus, bot masih dapat mengalami kekalahan jika pemain menggunakan strategi tertentu dalam bermain.

Pada pengembangan lebih lanjut, teknik optimasi serta heuristik lainnya dapat diterapkan untuk meningkatkan efisiensi serta kinerja pada bot. Selain itu, eksplorasi pendekatan kombinasi dengan metode lainnya dapat digunakan.

TAUTAN GITHUB PROGRAM

Program lengkap dapat dilihat pada tautan berikut: https://github.com/IceTeaXXD/TicTacToe-Bot

TAUTAN VIDEO YOUTUBE

Video berisi penjelasan program dapat dilihat pada tautan berikut:
https://youtu.be/knup5o6C9_I

UCAPAN TERIMA KASIH

Puji dan syukur diucapkan kepada Tuhan yang Maha Esa sehingga makalah ini dapat diselesaikan. Tidak lupa juga kepada Orang Tua penulis yang kerap memberi dukungan jasmani dan rohani sehingga makalah ini dapat disusun dengan baik. Lalu, kepada dosen pengampu mata kuliah Strategi Algoritma K3, Ir. Rila Mandala, M. Eng., Ph.D.yang telah menurunkan ilmunya kepada penulis. Terakhir, rekan-rekan penulis yang telah kerap menyemangati dan menginspirasi penulis dalam menyelesaikan makalah ini

REFERENCES

[1] Munir, Rinaldi (2023), Algoritma Brute Force Bagian 1, *Slide Kuliah IF 2211 Strategi Algoritma*, Bandung, Indonesia. Diakses 21 Mei 2023.

[2] Munir, Rinaldi (2023), Algoritma Brute Force Bagian 2, *Slide Kuliah IF 2211 Strategi Algoritma*, Bandung, Indonesia. Diakses 21 Mei 2023.

[3] "Minimax Algorithm in Game Theory | Set 1 (Introduction) - GeeksforGeeks," *GeeksforGeeks*, Oct. 05, 2018. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/ (accessed May 18AD).

[4] "Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning)," *GeeksforGeeks*, Jul. 24, 2016. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/ (accessed May 18AD).

[5] "Finding optimal move in Tic-Tac-Toe using Minimax Algorithm in Game Theory," *GeeksforGeeks*, Jun. 30, 2016. https://www.geeksforgeeks.org/finding-optimal-move-in-tic-tac-toe-using-minimax-algorithm-in-game-theory/ (accessed May 18AD).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Ahmad Nadil
13521024