**Tugas Besar Strategi Algoritma**

**Investigasi Efisiensi Algoritma Brute Force dan Backtracking Minimax dalam Menentukan Langkah Langkah Terbaik pada permainan Tic Tac Toe**

**A logo of a person's face

Description automatically generated**

Disusun Oleh :

|  |  |
| --- | --- |
| Dimas Ario Tri Kusumo | 103012380515 |
| Hifdzi Hisan | 103012380494 |
| Muhammad Imam Fernandi | 103012380494 |

**PROGRAM STUDI S1 INFORMATIKA**

**FAKULTAS INFORMATIKA**

**UNIVERSITAS TELKOM**

**2024**

## **ABSTRAK**

Permainan Tic-Tac-Toe telah lama menjadi subjek penelitian dalam bidang kecerdasan buatan dan pemrograman permainan. Dalam proyek ini, kami memperkenalkan sebuah studi perbandingan antara algoritma Brute Force dan Backtracking Minimax untuk menentukan langkah terbaik dalam permainan Tic-Tac-Toe.   
 Kami mengevaluasi kinerja kedua pendekatan tersebut menggunakan berbagai kasus uji dan menganalisis faktor-faktor seperti waktu komputasi dan penggunaan memori. Hasil eksperimen menunjukkan bahwa algoritma Backtracking Minimax menunjukkan keunggulan dalam efisiensi dibandingkan dengan pendekatan *Brute Force*. Temuan ini dapat menjadi landasan untuk pengembangan algoritma yang lebih canggih dalam menyelesaikan permainan Tic-Tac-Toe dan aplikasi lebih lanjut dalam domain kecerdasan buatan dan pemrograman permainan.

**Kata kunci:** Tic-Tac-Toe, algoritma *Brute Force*, algoritma Backtracking Minimax, efisiensi, kecerdasan buatan, pemrograman permainan.

**Daftar Isi**

[**ABSTRAK**2](#_Toc166170330)

[**PROPOSAL**4](#_Toc166170331)

[**BAB I**5](#_Toc166170332)

[**PENDAHULUAN**5](#_Toc166170333)

[**1.1.Latar Belakang**5](#_Toc166170334)

[**1.2.Perumusan Masalah**5](#_Toc166170335)

[**1.3.Batasan Masalah**5](#_Toc166170336)

[**1.4.Tujuan**6](#_Toc166170337)

[**1.5.Metode Penyelesaian Masalah**6](#_Toc166170338)

[**1.6.Pembagian Tugas Anggota**6](#_Toc166170339)

[**BAB II**7](#_Toc166170340)

[**Tinjauan Pustaka**7](#_Toc166170341)

[**2.1.Algoritma Brute Force dan Backtracking Minimax dalam Tic- Tac-Toe**7](#_Toc166170342)

[**2.2.Brute Force dalam Permainan Tic-Tac-Toe**7](#_Toc166170343)

[**2.3.Backtracking Minimax dalam Permainan Tic-Tac-Toe**7](#_Toc166170344)

[**2.4.Implementasi dan Evaluasi**7](#_Toc166170345)

**PROPOSAL**

**• Masalah yang akan diinvestigasi:** Menemukan strategi/langkah terbaik dalam permainan Tic-Tac-Toe menggunakan pendekatan algoritma Brute Force dan Backtracking Minimax. Permasalahan ini menarik karena permainan Tic-Tac-Toe merupakan permainan klasik yang cukup sederhana namun memiliki kompleksitas dalam mencari solusi optimal.

**• Bacaan yang akan dipelajari:** Materi tentang algoritma Brute Force, Backtracking Minimax, serta implementasinya dalam menyelesaikan permasalahan pada permainan. Referensi dapat diambil dari buku teks, jurnal, atau sumber online terkait topik tersebut.

**• Data yang akan digunakan:** Papan permainan Tic-Tac-Toe dengan berbagai konfigurasi awal yang dibangkitkan secara acak atau menggunakan kumpulan kasus uji yang telah ditentukan sebelumnya.

**• Metode/Algoritma yang diusulkan:** Mengimplementasikan algoritma Brute Force dan Backtracking Minimax menggunakan bahasa pemrograman yang sesuai untuk menentukan langkah terbaik pada setiap kondisi papan permainan. Modifikasi yang mungkin dilakukan adalah optimasi kinerja algoritma melalui teknik pruning atau memoization.

**• Evaluasi hasil:**

**- Kualitatif:** Membandingkan visualisasi langkah-langkah yang dihasilkan oleh kedua algoritma.

**- Kuantitatif:** Menganalisis metrik kinerja seperti waktu komputasi dan penggunaan memori untuk berbagai ukuran input, menggunakan uji statistik seperti uji t atau ANOVA untuk membandingkan kinerja kedua algoritma.

# **BAB I**

# **PENDAHULUAN**

## **Latar Belakang**

Permainan Tic-Tac-Toe telah menjadi bagian dari budaya populer dalam dunia permainan. Sederhana dalam aturannya namun menawarkan kesempatan untuk strategi yang kompleks, Tic-Tac-Toe menarik minat banyak orang dalam mencari solusi terbaik untuk memenangkan permainan ini. Dengan hanya menggunakan sebuah grid 3x3, pemain bertujuan untuk menempatkan simbol (biasanya X atau O) dalam upaya untuk membuat baris, kolom, atau diagonal yang terdiri dari tiga simbol yang sama.

Dalam konteks penelitian ini, kami tertarik untuk mengeksplorasi strategi terbaik dalam permainan Tic-Tac-Toe menggunakan pendekatan algoritma *Brute Force* dan *Backtracking Minimax*. Algoritma *Brute Force* akan mencoba semua kemungkinan langkah secara sistematis, sementara algoritma Backtracking Minimax akan menggunakan penyimpanan hasil sebelumnya untuk menghindari perhitungan ulang.

## **Perumusan Masalah**

Permasalahan yang dibahas dalam proyek ini adalah bagaimana mengimplementasikan algoritma *Brute Force* dan *Backtracking Minimax* untuk menentukan langkah terbaik dalam permainan Tic-Tac-Toe. Kami akan menganalisis kinerja kedua algoritma dalam hal waktu komputasi dan penggunaan memori, serta membandingkan keefektifan keduanya dalam mencari solusi optimal dalam permainan ini.

## **Batasan Masalah**

Batasan masalah yang kami tetapkan dalam proyek ini adalah sebagai berikut:

1. Kami akan membatasi permainan Tic-Tac-Toe pada grid standar 3x3.
2. Implementasi algoritma akan dilakukan dalam bahasa pemrograman Python.
3. Evaluasi kinerja algoritma akan dilakukan menggunakan metrik waktu komputasi dan penggunaan memori.

## **Tujuan**

Tujuan dari proyek ini adalah untuk mengevaluasi dan membandingkan kinerja algoritma *Brute Force* dan *Backtracking Minimax* dalam menentukan langkah terbaik dalam permainan Tic-Tac-Toe. Kami juga bertujuan untuk mengimplementasikan kedua algoritma dalam lingkungan pemrograman yang sesuai dan menganalisis hasilnya secara menyeluruh.

## **Metode Penyelesaian Masalah**

Metode yang akan digunakan dalam penyelesaian masalah ini meliputi:

1. Studi literatur tentang algoritma *Brute Force* dan *Backtracking Minimax*.
2. Implementasi algoritma dalam bahasa pemrograman Python.
3. Pengujian dan evaluasi kinerja algoritma menggunakan metrik waktu komputasi dan penggunaan memori.

## **Pembagian Tugas Anggota**

Berikut adalah pembagian tugas tim proyek akhir:

1. Dimas Ario Tri Kusumo

Peran: Implementasi algoritma *Brute Force*

Tanggung Jawab:

* Menerapkan algoritma *Brute Force* dalam pemrograman Python.
* elakukan pengujian terhadap algoritma *Brute Force*.
* Menganalisis kinerja algoritma *Brute Force.*

1. Hifdzi Hisan

Peran: Implementasi algoritma *Backtracking Minimax*

Tanggung Jawab:

* Mengimplementasikan algoritma *Backtracking Minimax* dalam bahasa pemrograman *Python*.
* Melakukan pengujian kinerja terhadap algoritma *Backtracking Minimax*.
* Menganalisis hasil eksperimen untuk algoritma *Backtracking Minimax*.

1. Muhammad Imam Fernandi

Peran: Analisis dan dokumentasi

Tanggung Jawab:

* Menganalisis hasil eksperimen dari kedua algoritma.
* Mendokumentasikan proses implementasi dan hasil eksperimen.

# **BAB II**

# **Tinjauan Pustaka**

## **Algoritma Brute Force dan Backtracking Minimax dalam Tic- Tac-Toe**

Permainan Tic-Tac-Toe adalah permainan papan yang dimainkan oleh dua pemain, biasanya disebut pemain X dan pemain O, yang bergantian menempatkan simbol X atau O pada grid 3x3. Tujuan dari permainan ini adalah untuk mendapatkan tiga simbol yang sama secara berurutan dalam satu baris, kolom, atau diagonal. Meskipun permainan ini memiliki kompleksitas yang relatif sederhana, namun mencari strategi atau langkah terbaik dalam permainan Tic-Tac-Toe dapat menjadi tantangan yang menarik.

## **Brute Force dalam Permainan Tic-Tac-Toe**

Brute Force adalah pendekatan algoritma yang sederhana namun kuat, di mana semua kemungkinan langkah yang mungkin dieksplorasi secara langsung untuk mencari solusi optimal. Dalam konteks permainan Tic-Tac-Toe, algoritma Brute Force akan mencoba semua kemungkinan langkah pada setiap kondisi papan permainan secara rekursif. Meskipun pendekatan ini dapat menemukan solusi optimal, namun kompleksitas waktu yang tinggi dapat menjadi masalah, terutama karena permainan Tic-Tac-Toe memiliki jumlah kemungkinan langkah yang terbatas.

## **Backtracking Minimax dalam Permainan Tic-Tac-Toe**

Implementasi algoritma Minimax dengan pendekatan backtracking pada permainan Tic-Tac-Toe memungkinkan evaluasi semua kemungkinan langkah yang dapat diambil oleh pemain dan lawan, sehingga dapat menentukan langkah terbaik untuk memenangkan permainan atau mencegah kekalahan dengan mengevaluasi setiap langkah secara mendalam dan memilih langkah yang memberikan hasil terbaik.

## **Implementasi dan Evaluasi**

Implementasi kedua pendekatan algoritma, yaitu Brute Force dan Backtracking Minimax, akan dilakukan dalam bahasa pemrograman yang sesuai, misalnya Python. Langkah-langkah terbaik yang dihasilkan oleh kedua algoritma akan dievaluasi dan dibandingkan untuk menentukan keefektifan dan keefisienan masing-masing pendekatan. Evaluasi akan dilakukan dengan membandingkan kinerja kedua algoritma dalam hal waktu komputasi dan penggunaan memori untuk berbagai ukuran input (konfigurasi papan permainan). Uji statistik seperti uji t atau ANOVA dapat digunakan untuk membandingkan kinerja kedua algoritma secara kuantitatif.

**b) Deskripsi Singkat Studi Kasus:**

Tic-Tac-Toe adalah permainan klasik yang dimainkan oleh dua pemain pada papan 3x3. Tujuan dari permainan ini adalah untuk membentuk garis lurus (baik horizontal, vertikal, atau diagonal) dari tiga buah simbol yang sama (X atau O). Dalam tugas besar ini, kami akan menerapkan dua algoritma, yaitu Brute Force dan Backtracking Minimax, untuk menentukan langkah terbaik dalam permainan Tic-Tac-Toe. Kedua algoritma ini akan diimplementasikan dalam bentuk program komputer dan dianalisis efisiensinya.

**c) Strategi-strategi Algoritma yang Dipilih beserta Penerapannya:**

1. Brute Force: Algoritma Brute Force adalah pendekatan sederhana yang mencoba semua kemungkinan solusi yang ada hingga menemukan solusi yang valid. Dalam kasus Tic-Tac-Toe, algoritma ini akan mencoba semua kemungkinan langkah yang tersedia dan mengevaluasi setiap hasilnya secara rekursif hingga ditemukan solusi terbaik.

Implementasi Brute Force pada Tic-Tac-Toe melibatkan fungsi rekursif yang mencoba semua kemungkinan langkah yang tersisa pada papan permainan. Fungsi ini akan memanggil dirinya sendiri dengan menempatkan simbol pemain pada setiap sel kosong dan mengevaluasi hasil permainan. Jika ditemukan solusi yang menguntungkan bagi pemain, maka langkah tersebut akan dipilih sebagai langkah terbaik.

1. BackTracking Minimax merupakan algoritma backtracking yang digunakan dalam pengambilan keputusan dan teori permainan untuk menentukan langkah terbaik bagi seorang pemain, dengan syarat lawan juga bermain maksimal.

Implementasi algoritma Minimax dengan pendekatan backtracking pada permainan Tic-Tac-Toe memungkinkan evaluasi semua kemungkinan langkah yang dapat diambil oleh pemain dan lawan, sehingga dapat menentukan langkah terbaik untuk memenangkan permainan atau mencegah kekalahan dengan mengevaluasi setiap langkah secara mendalam dan memilih langkah yang memberikan hasil terbaik.

**d) Analisis Perbandingan Efisiensi:**

Dalam bagian ini, kami akan membandingkan efisiensi kedua algoritma (Brute Force dan Backtracking Minimax) dalam menentukan langkah terbaik pada permainan Tic-Tac-Toe. Kami akan menguji kedua algoritma dengan 5 input yang terurut secara ascending (misalnya, dari konfigurasi papan permainan yang paling sederhana hingga yang paling kompleks).

Hasil analisis akan disajikan dalam bentuk grafik yang membandingkan waktu komputasi (running time) dari kedua algoritma pada setiap input yang diberikan. Sumbu X pada grafik akan menunjukkan urutan input, sedangkan sumbu Y akan menunjukkan waktu komputasi (dalam satuan waktu yang sesuai, misalnya milidetik atau detik).

Grafik akan dilengkapi dengan keterangan yang menjelaskan setiap input yang digunakan, serta interpretasi hasil analisis perbandingan efisiensi antara kedua algoritma.

**1.Tabel Strategi Brute force untuk permainan Tic Tac Toe**

Untuk memahami bagaimana algoritma brute force bekerja dalam permainan Tic Tac Toe, kita dapat membuat tabel yang menunjukkan semua kemungkinan langkah yang diambil oleh pemain 'X' dan 'O'. Tabel ini akan mencakup setiap langkah yang mungkin, hasil dari langkah tersebut, dan siapa yang menang atau jika permainan berakhir seri.

Langkah 1: Papan Awal

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Langkah 2: Pemain 'X' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Langkah 3: Pemain 'O' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | O | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Langkah 4: Pemain 'X' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | O | X |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Langkah 5: Pemain 'O' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | O | X |
| O | 5 | 6 |
| 7 | 8 | 9 |

Langkah 6: Pemain 'X' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | O | X |
| O | X | 6 |
| 7 | 8 | 9 |

Langkah 7: Pemain 'O' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | O | X |
| O | X | O |
| 7 | 8 | 9 |

Langkah 8: Pemain 'X' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | O | X |
| O | X | O |
| X | 8 | 9 |

Langkah 9: Pemain 'O' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | O | X |
| O | X | O |
| X | O | 9 |

Langkah 10: Pemain 'X' Memilih Posisi

|  |  |  |
| --- | --- | --- |
| X | O | X |
| O | X | O |
| X | O | X |

Tabel Perhitungan

| **Langkah** | **Papan Permainan** | **Pemain** | **Hasil** |
| --- | --- | --- | --- |
| 1 | 1 2 3 \n 4 5 6 \n 7 8 9 | X | - |
| 2 | X 2 3 \n 4 5 6 \n 7 8 9 | O | - |
| 3 | X O 3 \n 4 5 6 \n 7 8 9 | X | - |
| 4 | X O X \n 4 5 6 \n 7 8 9 | O | - |
| 5 | X O X \n O 5 6 \n 7 8 9 | X | - |
| 6 | X O X \n O X 6 \n 7 8 9 | O | - |
| 7 | X O X \n O X O \n 7 8 9 | X | - |
| 8 | X O X \n O X O \n X 8 9 | O | - |
| 9 | X O X \n O X O \n X O 9 | X | - |
| 10 | X O X \n O X O \n X O X | - | Seri |

**Penjelasan**

•Langkah 1-10: Setiap langkah menunjukkan perubahan pada papan permainan setelah pemain 'X' atau 'O' membuat langkah mereka.

•Hasil: Kolom hasil menunjukkan apakah permainan berakhir dengan kemenangan untuk salah satu pemain atau seri.

**Kesimpulan**

Algoritma brute force mencoba semua kemungkinan langkah hingga permainan selesai. Dalam contoh ini, permainan berakhir seri setelah semua langkah diambil. Algoritma ini memastikan bahwa setiap kemungkinan langkah dievaluasi untuk menemukan langkah terbaik, meskipun tidak efisien untuk permainan yang lebih kompleks.

**Kode Program untuk Langkah-langkah Tabel Perhitungan Brute Force untuk Permainan Tic Tac Toe**

Berikut adalah kode program Python yang mengimplementasikan langkah-langkah brute force untuk permainan Tic Tac Toe. Kode ini akan mencoba semua kemungkinan langkah dan mencatat hasil dari setiap langkah.

def print\_board(board):

for row in board:

print(" | ".join(row))

print("-" \* 5)

def is\_winner(board, player):

# Cek baris, kolom, dan diagonal

for i in range(3):

if all([cell == player for cell in board[i]]):

return True

if all([board[j][i] == player for j in range(3)]):

return True

if all([board[i][i] == player for i in range(3)]) or all([board[i][2 - i] == player for i in range(3)]):

return True

return False

def is\_full(board):

return all([cell != ' ' for row in board for cell in row])

def get\_empty\_cells(board):

return [(i, j) for i in range(3) for j in range(3) if board[i][j] == ' ']

def minimax(board, player):

if is\_winner(board, 'X'):

return 1 # X menang

if is\_winner(board, 'O'):

return -1 # O menang

if is\_full(board):

return 0 # Seri

scores = []

for cell in get\_empty\_cells(board):

i, j = cell

board[i][j] = player

score = minimax(board, 'O' if player == 'X' else 'X')

scores.append(score)

board[i][j] = ' '

return max(scores) if player == 'X' else min(scores)

def best\_move(board, player):

best\_score = -float('inf') if player == 'X' else float('inf')

move = None

for cell in get\_empty\_cells(board):

i, j = cell

board[i][j] = player

score = minimax(board, 'O' if player == 'X' else 'X')

board[i][j] = ' '

if player == 'X':

if score > best\_score:

best\_score = score

move = (i, j)

else:

if score < best\_score:

best\_score = score

move = (i, j)

return move

def play\_game():

board = [[' ' for \_ in range(3)] for \_ in range(3)]

current\_player = 'X'

steps = 0

while not is\_full(board) and not is\_winner(board, 'X') and not is\_winner(board, 'O'):

print(f"Langkah {steps + 1}: Pemain {current\_player}")

print\_board(board)

move = best\_move(board, current\_player)

if move:

board[move[0]][move[1]] = current\_player

current\_player = 'O' if current\_player == 'X' else 'X'

steps += 1

print("Papan Akhir:")

print\_board(board)

if is\_winner(board, 'X'):

print("Pemain X menang!")

elif is\_winner(board, 'O'):

print("Pemain O menang!")

else:

print("Permainan berakhir seri!")

# Main program

play\_game()

Penjelasan Kode

•print\_board: Fungsi untuk mencetak papan permainan.

•is\_winner: Fungsi untuk mengecek apakah pemain tertentu menang.

•is\_full: Fungsi untuk mengecek apakah papan sudah penuh.

•get\_empty\_cells: Fungsi untuk mendapatkan semua sel yang kosong di papan.

•minimax: Fungsi rekursif yang menggunakan algoritma minimax untuk mengevaluasi semua kemungkinan langkah dan mengembalikan skor terbaik untuk pemain saat ini.

•best\_move: Fungsi untuk mencari langkah terbaik bagi pemain berdasarkan evaluasi minimax.

•play\_game: Fungsi utama untuk menjalankan permainan Tic Tac Toe menggunakan algoritma brute force.

Cara Kerja

1.Inisialisasi Papan: Papan permainan diinisialisasi dengan sel kosong.

2.Langkah Pemain: Pemain 'X' dan 'O' bergantian membuat langkah terbaik mereka menggunakan fungsi best\_move.

3.Evaluasi Papan: Setelah setiap langkah, papan permainan dievaluasi untuk menentukan apakah ada pemenang atau jika permainan berakhir seri.

4.Cetak Hasil: Papan akhir dan hasil permainan dicetak.  
  
  
Kode ini memastikan bahwa setiap langkah yang diambil adalah optimal berdasarkan kondisi saat ini, sehingga pemain dapat membuat keputusan yang paling menguntungkan.

[Finding optimal move in Tic-Tac-Toe using Minimax Algorithm in Game Theory - GeeksforGeeks](https://www.geeksforgeeks.org/finding-optimal-move-in-tic-tac-toe-using-minimax-algorithm-in-game-theory/)

A diagram of a game

Description automatically generated

Langkah 1: Giliran pemain x

• Pindah (0, 0)

• Nilai Papan: 0

• Keadaan Papan:

|  |  |  |
| --- | --- | --- |
| X | - | - |
| - | - | - |
| - | - | - |

Langkah 2: Giliran pemain o

* **Pindah (0, 1)**
* Nilai Papan: 10
* Keadaan Papan:

|  |  |  |
| --- | --- | --- |
| X | O | - |
| - | - | - |
| - | - | - |

**Langkah 3: Giliran pemain x**

* **Pindah (1, 0)**
* Nilai Papan: 10
* Keadaan Papan:

|  |  |  |
| --- | --- | --- |
| X | O | - |
| X | - | - |
| - | - | - |

Langkah 4: Giliran pemain o

* **Pindah (0, 2)**
* Nilai Papan: 10
* Keadaan Papan:

|  |  |  |
| --- | --- | --- |
| X | O | O |
| X | - | - |
| - | - | - |

Langkah 5: Giliran pemain x

* **Pindah (1, 1)**
* Nilai Papan: 10
* Keadaan Papan:

|  |  |  |
| --- | --- | --- |
| X | O | O |
| X | X | - |
| - | - | - |

Langkah 6: Giliran pemain o

* **Pindah (1, 2)**
* Nilai Papan: 10
* Keadaan Papan:

|  |  |  |
| --- | --- | --- |
| X | O | O |
| X | X | O |
| - | - | - |

Langkah 7: Giliran pemain x

* **Pindah (2, 0)**
* Nilai Papan: 10
* Keadaan Papan:

|  |  |  |
| --- | --- | --- |
| X | O | O |
| X | X | O |
| X | - | - |

Hasil Akhir

* Pemain x menang!

|  |  |  |
| --- | --- | --- |
| X | O | X |
| O | X | O |
| X | O | X |

Tabel ringkasan langkah:

| **Langkah** | **Pemain** | **Pindah** | **Keadaan Papan** | **Nilai** |
| --- | --- | --- | --- | --- |
| 1 | x | (0, 0) | x \_ \_ | 0 |
|  |  |  | \_ \_ \_ |  |
|  |  |  | \_ \_ \_ |  |
| 2 | o | (0, 1) | x o \_ | 10 |
|  |  |  | \_ \_ \_ |  |
|  |  |  | \_ \_ \_ |  |
| 3 | x | (1, 0) | x o \_ | 10 |
|  |  |  | x \_ \_ |  |
|  |  |  | \_ \_ \_ |  |
| 4 | o | (0, 2) | x o o | 10 |
|  |  |  | x \_ \_ |  |
|  |  |  | \_ \_ \_ |  |
| 5 | x | (1, 1) | x o o | 10 |
|  |  |  | x x \_ |  |
|  |  |  | \_ \_ \_ |  |
| 6 | o | (1, 2) | x o o | 10 |
|  |  |  | x x o |  |
|  |  |  | \_ \_ \_ |  |
| 7 | x | (2, 0) | x o o | 10 |
|  |  |  | x x o |  |
|  |  |  | x \_ \_ |  |

Pemain x menang setelah langkah ke-7.  
  
  
**Kode Program untuk Langkah-langkah Tabel Perhitungan Backtracking Minimax untuk Permainan Tic Tac Toe**

Berikut adalah kode program Python yang mengimplementasikan langkah-langkah Backtracking Minimax untuk permainan Tic Tac Toe. Kode ini akan mencoba semua kemungkinan langkah dan mencatat hasil dari setiap langkah.

|  |
| --- |
| player, opponent = 'x', 'o'  def isMovesLeft(board):  for i in range(3):  for j in range(3):  if board[i][j] == '\_':  return True  return False  def evaluate(b):  for row in range(3):  if b[row][0] == b[row][1] and b[row][1] == b[row][2]:  if b[row][0] == player:  return 10  elif b[row][0] == opponent:  return -10  for col in range(3):  if b[0][col] == b[1][col] and b[1][col] == b[2][col]:  if b[0][col] == player:  return 10  elif b[0][col] == opponent:  return -10  if b[0][0] == b[1][1] and b[1][1] == b[2][2]:  if b[0][0] == player:  return 10  elif b[0][0] == opponent:  return -10  if b[0][2] == b[1][1] and b[1][1] == b[2][0]:  if b[0][2] == player:  return 10  elif b[0][2] == opponent:  return -10  return 0  def minimax(board, depth, isMax):  score = evaluate(board)  if score == 10:  return score  if score == -10:  return score  if not isMovesLeft(board):  return 0  if isMax:  best = -1000  for i in range(3):  for j in range(3):  if board[i][j] == '\_':  board[i][j] = player  best = max(best, minimax(board, depth + 1, not isMax))  board[i][j] = '\_'  return best  else:  best = 1000  for i in range(3):  for j in range(3):  if board[i][j] == '\_':  board[i][j] = opponent  best = min(best, minimax(board, depth + 1, not isMax))  board[i][j] = '\_'  return best  def findBestMove(board):  bestVal = -1000  bestMove = (-1, -1)  for i in range(3):  for j in range(3):  if board[i][j] == '\_':  board[i][j] = player  moveVal = minimax(board, 0, False)  board[i][j] = '\_'  if moveVal > bestVal:  bestMove = (i, j)  bestVal = moveVal  print(f"Move ({i}, {j}) => Board Value: {moveVal}")  for row in board:  print(" ".join(row))  print()  print("The value of the best Move is :", bestVal)  print()  return bestMove  def playGame():  board = [['\_', '\_', '\_'], ['\_', '\_', '\_'], ['\_', '\_', '\_']]  current\_player = player  step = 1  while isMovesLeft(board) and evaluate(board) == 0:  print(f"Step {step}: Player {current\_player}'s move")  bestMove = findBestMove(board)  board[bestMove[0]][bestMove[1]] = current\_player  for row in board:  print(" ".join(row))  print()  current\_player = opponent if current\_player == player else player  step += 1  final\_score = evaluate(board)  if final\_score == 10:  print("Player x wins!")  elif final\_score == -10:  print("Player o wins!")  else:  print("The game is a draw!")  # Driver code to start from an empty board  playGame() |

### Penjelasan Kode dan Cara Kerjanya

#### 1. **Fungsi** isMovesLeft(board)

* **Tujuan:** Mengecek apakah masih ada langkah yang bisa dimainkan.
* **Cara Kerja:** Mengecek setiap sel dalam papan untuk melihat apakah ada yang kosong ('\_').

#### 2. **Fungsi** evaluate(b)

* **Tujuan:** Mengevaluasi papan untuk melihat apakah ada pemenang.
* **Cara Kerja:** Mengecek setiap baris, kolom, dan diagonal untuk melihat apakah ada tiga simbol yang sama ('x' atau 'o'). Mengembalikan nilai 10 jika pemain 'x' menang, -10 jika pemain 'o' menang, dan 0 jika tidak ada yang menang.

#### 3. **Fungsi** minimax(board, depth, isMax)

* **Tujuan:** Menentukan nilai terbaik yang bisa dicapai pemain.
* **Cara Kerja:** Menggunakan rekursi untuk mensimulasikan setiap langkah yang mungkin dan memilih langkah yang memaksimalkan nilai untuk pemain 'x' (isMax=True) atau meminimalkan nilai untuk pemain 'o' (isMax=False).

#### 4. **Fungsi** findBestMove(board)

1. **Tujuan:** Menemukan langkah terbaik untuk pemain 'x'.
2. **Cara Kerja:** Mengevaluasi setiap langkah yang mungkin menggunakan fungsi minimax dan mengembalikan langkah dengan nilai tertinggi.

#### 5. **Fungsi** playGame()

* **Tujuan:** Menjalankan permainan dari papan kosong.
* **Cara Kerja:**
  1. Menginisialisasi papan kosong.
  2. Memilih langkah terbaik untuk pemain yang sedang bermain (dimulai dengan pemain 'x').
  3. Menampilkan langkah dan keadaan papan setelah setiap langkah.
  4. Mengganti giliran pemain.
  5. Mengulangi langkah 2-4 hingga permainan selesai (tidak ada langkah tersisa atau ada pemenang).
  6. Menampilkan hasil akhir permainan (pemenang atau seri).

### Cara Kerja Algoritma Minimax

1. **Inisialisasi Papan:** Papan permainan diinisialisasi dengan semua sel kosong ('\_').
2. **Pengecekan Langkah yang Tersedia:** Fungsi isMovesLeft mengecek apakah masih ada langkah yang bisa dimainkan.
3. **Evaluasi Papan:** Fungsi evaluate mengevaluasi keadaan papan untuk menentukan apakah ada pemenang.
4. **Rekursi Minimax:** Fungsi minimax menggunakan rekursi untuk mensimulasikan setiap langkah yang mungkin dan mengembalikan nilai terbaik untuk pemain yang sedang bermain. Jika giliran pemain 'x', maka algoritma memilih langkah yang memaksimalkan nilai. Jika giliran pemain 'o', maka algoritma memilih langkah yang meminimalkan nilai.
5. **Menentukan Langkah Terbaik:** Fungsi findBestMove mengevaluasi setiap langkah yang mungkin dan memilih langkah dengan nilai tertinggi untuk pemain 'x'.
6. **Permainan Berlangsung:** Fungsi playGame menjalankan permainan dengan langkah-langkah optimal yang dipilih oleh fungsi findBestMove, bergantian antara pemain 'x' dan 'o', hingga permainan selesai.

Dengan cara ini, algoritma Minimax memastikan bahwa setiap pemain selalu mengambil langkah terbaik yang tersedia, sehingga hasil permainan selalu optimal.

# **BAB III**

# **Hasil Analisis**

### Kelebihan dan Kekurangan dari Kedua Metode

**Brute Force:**

1. Kekurangan:

* Kekurangan Utama: Metode brute force mencoba semua kemungkinan langkah dalam permainan, yang membutuhkan waktu dan sumber daya komputasi yang besar, terutama untuk permainan yang lebih kompleks.
* Kurang Efisien: Metode ini tidak mengambil keuntungan dari strategi atau keadaan papan yang sudah dievaluasi sebelumnya, sehingga tidak efisien dalam penggunaan sumber daya.
* Kompleksitas Tinggi: Kompleksitas waktu eksponensial terhadap kedalaman pohon pemanggilan rekursif, dan kompleksitas yang tinggi dalam hal jumlah langkah yang harus dipertimbangkan, membuatnya tidak praktis untuk permainan dengan keadaan papan yang lebih besar.

1. Kelebihan:

* Sederhana: Metode ini mudah dipahami dan diimplementasikan.
* Solusi Optimal: Dengan mencoba semua kemungkinan, metode ini dapat menemukan solusi optimal jika ada.

**Backtracking Minimax:**

1. Kekurangan:\*\*

* Keterbatasan Heuristik: Kinerja algoritma bergantung pada kualitas fungsi evaluasi (heuristik). Heuristik yang buruk atau tidak sesuai dapat menghasilkan solusi yang tidak optimal.
* Keterbatasan Kompleksitas Ruang: Meskipun lebih efisien daripada brute force dalam hal kompleksitas waktu, metode ini membutuhkan ruang memori yang cukup besar untuk menyimpan seluruh pohon pemanggilan rekursif.

1. Kelebihan:

* Efisien: Dengan menggunakan teknik backtracking, algoritma ini mengurangi jumlah percobaan yang harus dilakukan untuk mengevaluasi semua kemungkinan langkah, sehingga meningkatkan efisiensi secara signifikan.
* Mengambil Keputusan yang Lebih Cerdas: Algoritma ini secara cerdas memilih langkah-langkah yang akan dieksplorasi lebih lanjut, sehingga mengurangi jumlah pemanggilan fungsi yang diperlukan.

### Perhitungan Kompleksitas A graph with orange lines Description automatically generated

**Brute Force:**

* Kompleksitas Waktu: Kompleksitas waktu brute force dalam permainan Tic Tac Toe adalah sekitar *O*(9!) atau *O*(362,880). Ini karena ada 9 sel yang dapat diisi dalam papan, dan setiap sel dapat diisi oleh salah satu dari dua pemain atau dibiarkan kosong.
* Kompleksitas Ruang: Kompleksitas ruang tergantung pada bagaimana informasi papan disimpan dalam memori. Dalam kasus ini, setiap perubahan papan disalin, sehingga membutuhkan ruang tambahan yang signifikan.

**Backtracking Minimax:**

* Kompleksitas Waktu: Kompleksitas waktu backtracking minimax bergantung pada kedalaman pohon pemanggilan rekursif. Dalam kasus Tic Tac Toe, kedalaman pohon tidak melebihi 9 (jumlah maksimum langkah dalam permainan). Setiap tingkat pemanggilan rekursif memiliki jumlah cabang maksimum sekitar 9 (jumlah maksimum langkah yang dapat diambil dalam satu langkah). Oleh karena itu, kompleksitas waktu secara kasar dapat diestimasi sebagai *O*(*bd*), di mana *b* adalah faktor cabang rata-rata per langkah dan *d* adalah kedalaman pohon.
* Kompleksitas Ruang: Kompleksitas ruang tergantung pada cara papan disimpan dalam memori dan jumlah panggilan rekursif yang disimpan dalam tumpukan pemanggilan. Dalam kasus ini, algoritma menyimpan salinan papan untuk setiap langkah, sehingga membutuhkan ruang tambahan yang signifikan.

# **BAB IV**

# **Penutup**

### Kesimpulan

Dengan mempertimbangkan kelebihan dan kekurangan masing-masing metode, backtracking minimax umumnya lebih diinginkan dalam permainan seperti Tic Tac Toe karena efisiensinya yang lebih tinggi dibandingkan brute force.

Metode Backtracking Minimax lebih baik digunakan untuk permainan Tic-Tac-Toe karena lebih efisien dan cerdas dalam menentukan langkah optimal. Meskipun implementasinya lebih kompleks, hasil yang diperoleh lebih optimal dan dapat dioptimalkan lebih lanjut dengan teknik tambahan seperti Alpha-Beta Pruning. Metode Brute Force, meskipun lebih sederhana, tidak efisien untuk papan permainan yang lebih besar dan tidak memberikan hasil yang optimal.

### Saran

* 1. **Pemilihan Metode:**
* Dari hasil analisis, dapat disimpulkan bahwa metode backtracking minimax lebih unggul dalam hal efisiensi pemrosesan dan waktu eksekusi dibandingkan dengan metode brute force. Oleh karena itu, lebih disarankan untuk menggunakan pendekatan backtracking minimax dalam implementasi permainan Tic Tac Toe.
  1. **Optimalisasi Implementasi:**
* Saat menggunakan backtracking minimax, pastikan bahwa fungsi evaluasi (heuristik) telah dioptimalkan untuk memastikan bahwa algoritma memilih langkah-langkah yang paling optimal dengan cepat.
* Selain itu, pastikan bahwa implementasi backtracking minimax telah dioptimalkan secara efisien untuk meminimalkan penggunaan sumber daya dan waktu eksekusi.

1. **Pemahaman Hasil Output:**

* Hasil output dari kedua metode memberikan gambaran yang jelas tentang bagaimana algoritma berperilaku dalam permainan Tic Tac Toe.
* Hasil output backtracking minimax menunjukkan kemampuannya untuk memilih langkah-langkah yang optimal dengan cepat, sementara hasil output brute force memperlihatkan bahwa metode tersebut membutuhkan lebih banyak waktu untuk mengevaluasi semua kemungkinan langkah.

1. **Pengembangan Lebih Lanjut:**

* Setelah mengimplementasikan backtracking minimax, Anda dapat mempertimbangkan untuk meningkatkan permainan dengan menambahkan fitur tambahan, seperti antarmuka pengguna grafis (GUI) atau tingkat kesulitan yang dapat disesuaikan.
* Selain itu, Anda juga dapat mempertimbangkan untuk menerapkan algoritma alpha-beta pruning untuk lebih mengoptimalkan algoritma backtracking minimax.