Penerapan Algoritma DFS dan *Backtracking* pada Permainan *Tic-Tac-Toe*

Alex Sander - 13521061
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13521061@std.stei.itb.ac.id

***Abstract*—Permainan tic-tac-toe adalah permainan yang dimainkan oleh dua pemain di atas papan 3x3. Setiap pemain bergantian menempatkan tanda X atau O (berdasarkan pilihan awal pemain) pada kotak yang masih belum terisi. Seorang pemain dikatakan menang jika berhasil membentuk garis horizontal, vertikal atau diagonal berukuran 3 satuan dengan tandanya. Pada makalah ini, dirancang algoritma DFS dan Backtracking untuk mencari solusi untuk memencari pilihan langkah terbaik untuk mendekatkan pemain ke kemenangan. Kedekatan seseorang dengan kemenangan akan diukur berdasarkan jumlah langkah minimum yang diperlukan untuk mencapai kemenangan. Dengan algoritma ini, pemain dapat secara sistematis mengeksplorasi setiap kemungkinan langkah yang tersedia berdasarkan kondisi papan tertentu.**

Keywords—*tic tac toe, algoritma, dfs, backtracking, minimax*

I. PENDAHULUAN

Permainan Tic-Tac-Toe merupakan salah satu permainan papan yang populer dan sering dimainkan di seluruh dunia. Meskipun terlihat sederhana, permainan ini menawarkan tantangan strategis yang menarik, di mana dua pemain bergantian menempatkan tanda mereka (biasanya "X" dan "O") pada kotak-kotak yang tersedia dalam sebuah grid 3x3. Tujuan dari permainan ini adalah untuk menciptakan urutan tiga tanda yang sama secara horizontal, vertikal, atau diagonal.

Dalam beberapa dekade terakhir, penerapan algoritma kecerdasan buatan dalam permainan telah menjadi topik penelitian yang menarik. Salah satu algoritma yang sering digunakan adalah Depth-First Search (DFS) dan Backtracking. Algoritma DFS digunakan untuk menjelajahi semua kemungkinan langkah dalam permainan secara sistematis, dengan mengeksplorasi setiap cabang dalam pohon permainan. Sementara itu, algoritma Backtracking digunakan untuk mencari solusi dengan menguji semua kemungkinan secara rekursif dan mengembalikan langkah sebelumnya jika tidak ada solusi yang memuaskan.

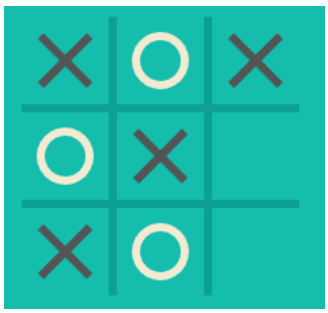
Dalam paper ini, kami akan mengeksplorasi penerapan algoritma DFS dan Backtracking pada permainan Tic-Tac-Toe. Kami akan menjelaskan secara rinci bagaimana kedua algoritma ini dapat digunakan untuk mencari solusi optimal dalam permainan Tic-Tac-Toe.

Penelitian ini bertujuan untuk memberikan pemahaman yang mendalam tentang penerapan algoritma DFS dan Backtracking dalam permainan Tic-Tac-Toe. Kami berharap bahwa penelitian ini dapat memberikan kontribusi yang berharga dalam pengembangan dan pemahaman lebih lanjut tentang algoritma kecerdasan buatan pada permainan papan klasik seperti Tic-Tac-Toe.

II. LANDASAN TEORI

A. *Permainan Tic Tac Toe*

Tic Tac Toe adalah salah satu permainan papan klasik yang sederhana namun mengasyikkan. Juga dikenal dengan nama "Xs and Os" atau "Three-in-a-Row", permainan ini biasanya dimainkan di atas grid 3x3. Walaupun terlihat mudah, Tic Tac Toe membutuhkan strategi yang cerdas dan pengambilan keputusan yang baik.



Gambar 1 Ilustrasi Permainan Tic-Tac-Toe

Asal mula Tic Tac Toe tidak dapat ditelusuri dengan pasti, namun permainan ini telah ada sejak berabad-abad yang lalu. Versi awal Tic Tac Toe diduga dimainkan di Mesir Kuno dan Romawi dengan menggunakan batu kecil sebagai gantinya. Permainan ini terus berkembang dan menjadi populer di berbagai budaya di seluruh dunia.

Tic Tac Toe adalah permainan berdua, di mana dua pemain bergantian menempatkan tanda mereka pada grid 3x3. Pemain pertama biasanya menggunakan tanda "X", sedangkan pemain kedua menggunakan tanda "O". Tujuan dari permainan ini adalah untuk menempatkan tiga tanda secara berurutan dalam baris, kolom, atau diagonal.

Karena batasan grid yang kecil, Tic Tac Toe sering kali berakhir dengan hasil seri atau seri jika kedua pemain bermain dengan benar. Namun, permainan ini tetap menarik karena adanya kemungkinan variasi strategi dan tantangan untuk mengantisipasi langkah lawan.

Tic Tac Toe telah menjadi permainan yang populer di berbagai platform, mulai dari permainan papan fisik hingga implementasi digital yang dapat dimainkan di komputer, ponsel pintar, atau perangkat lainnya. Selain itu, permainan ini sering digunakan sebagai contoh atau latihan dalam mempelajari konsep dasar pemrograman.

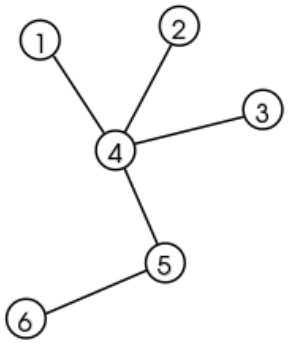
Saat Tic Tac Toc dimainkan dengan optimal oleh kedua belah pihak, akan pasti tiap permainan akan berakhir dengan seri.

B. Algoritma DFS

Algoritma Depth-First Search (DFS) adalah algoritma pencarian graf yang digunakan untuk menjelajahi atau memeriksa setiap simpul atau node dalam graf secara sistematis. DFS berfungsi dengan memulai dari simpul awal (atau simpul akar dalam kasus pohon) dan secara rekursif menjelajahi semua simpul tetangga sebelum kembali ke simpul sebelumnya dan menjelajahi simpul tetangga yang belum dijelajahi.

Berikut ini adalah beberapa konsep dasar yang menjadi landasan teori dari algoritma DFS:

1. Graf: DFS diterapkan pada struktur data graf, yang terdiri dari simpul atau node yang terhubung oleh sisi atau tepi. Graf dapat digunakan untuk merepresentasikan berbagai masalah, seperti jaringan sosial, struktur hierarki, peta jalan, dan lain sebagainya.



Gambar 2 Ilustrasi Graf

2. Simpul (Node): Setiap simpul dalam graf memiliki nilai atau informasi yang terkait dengannya. Setiap simpul dapat memiliki simpul-simpul tetangga yang terhubung dengan tepi atau sisi. Pada Gambar 2, *node* dilambangkan dengan lingkarang dengan label yang unik.
3. Tepi (Edge): Tepi adalah koneksi yang menghubungkan dua simpul dalam graf. Tepi dapat memiliki arah (digunakan dalam graf berarah) atau tidak memiliki arah (digunakan dalam graf tak berarah). Pada Gambar 2, *edge* dilambangkan dengan garis tak berarah yang menghubungkan antara 2 *node*.
4. Penandaan (Visitation): Saat menjalankan DFS, setiap simpul dikunjungi dan ditandai agar tidak dikunjungi lagi. Penandaan dapat dilakukan dengan menggunakan tanda flag atau atribut khusus pada simpul.
5. Stack: DFS menggunakan struktur data stack untuk melacak simpul yang harus dikunjungi berikutnya. Setiap kali simpul baru dikunjungi, simpul tersebut dimasukkan ke dalam stack atau memanggil fungsi rekursif.

Pencarian Mendalam (Depth-First) menjelajahi secara mendalam pada setiap jalur dari simpul awal yang dipilih. Ini berarti bahwa DFS akan mengunjungi simpul tetangga dalam satu jalur sejauh mungkin sebelum kembali ke simpul sebelumnya dan menjelajahi jalur alternatif.

Selama proses menjalankan DFS pada graf tak berarah, DFS membentuk pohon yang terdiri dari simpul-simpul yang dikunjungi dan tepi-tepi yang membentuk jalur dari simpul awal ke simpul lainnya.

Algoritma DFS dapat diterapkan pada berbagai macam masalah, seperti pencarian jalur terpendek (shortest path), penyelesaian puzzle, dan identifikasi struktur data seperti komponen terhubung dan siklus.

Kelebihan algoritma DFS adalah kecepatan pencarian yang relatif cepat, terutama jika pohon atau graf yang dicari berukuran besar dan berbentuk simpul berantai (linear). Namun, kekurangan algoritma DFS adalah kemungkinan terjebak dalam siklus yang tidak diinginkan dan tidak menjamin menemukan jalur terpendek.

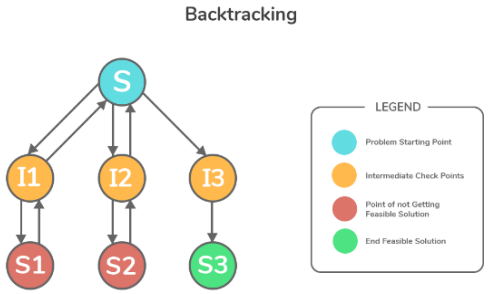
C. Backtracking

Backtracking adalah sebuah teknik algoritma yang digunakan untuk mencari solusi melalui percobaan berulang dan pemilihan mundur (undo) jika suatu langkah tidak mengarah ke solusi yang diinginkan. Algoritma backtracking sangat berguna untuk menyelesaikan masalah yang memerlukan eksplorasi semua kemungkinan solusi yang ada.
Prinsip dasar dari algoritma backtracking adalah sebagai berikut:

1. Pilih langkah awal: Pilih satu langkah awal dari kondisi masalah yang diberikan.
2. Coba setiap kemungkinan: Coba setiap kemungkinan langkah berikutnya dari langkah saat ini, secara rekursif atau iteratif.
3. Evaluasi solusi: Evaluasi apakah langkah saat ini mengarah ke solusi yang valid atau belum. Jika langkah saat ini mengarah ke solusi yang valid, solusi diterima. Jika langkah saat ini tidak mengarah ke solusi yang valid, kembali ke langkah sebelumnya (backtrack).
4. Pemilihan mundur (Undo): Jika langkah saat ini tidak mengarah ke solusi yang valid, lakukan pemilihan mundur (undo) untuk membatalkan langkah terakhir yang diambil. Kembali ke langkah sebelumnya dan coba langkah alternatif lainnya.

5. Ulangi langkah-langkah 2-4: Ulangi langkah-langkah 2 hingga 4 sampai solusi ditemukan atau semua kemungkinan solusi telah dijelajahi.

Algoritma backtracking biasanya digunakan dalam pemecahan masalah yang melibatkan pemilihan, penempatan, atau kombinasi objek atau elemen dalam satu atau beberapa set dengan aturan atau batasan tertentu. Beberapa contoh penerapan algoritma backtracking termasuk Sudoku, N-Queen, kriptaritmatika, penyelesaian maze, dan banyak lagi.



Gambar 3 Ilustrasi Proses Backtracking

Kelebihan algoritma backtracking adalah kemampuannya untuk menemukan solusi yang optimal dalam beberapa kasus dan fleksibilitasnya dalam menangani berbagai masalah. Namun, kekurangan algoritma backtracking adalah kompleksitas waktu yang tinggi karena mencoba semua kemungkinan solusi, terutama pada masalah dengan ruang pencarian yang besar. Oleh karena itu, teknik pruning (pemangkasan) sering digunakan untuk membatasi jumlah cabang yang dieksplorasi dan meningkatkan efisiensi algoritma backtracking.

D. Algoritma Minimax

Algoritma Minimax adalah salah satu algoritma yang digunakan dalam pengambilan keputusan pada permainan dengan dua pemain yang bergantian. Algoritma ini berbasis pada pemikiran bahwa setiap pemain berusaha untuk memaksimalkan keuntungannya sendiri sementara pemain lawan berusaha untuk meminimalkan keuntungan tersebut.

Landasan teori algoritma Minimax yang relevan dijelaskan sebagai berikut:

1. Representasi pohon permainan: Permainan dapat direpresentasikan sebagai pohon di mana setiap simpul mewakili keadaan permainan pada suatu titik tertentu. Pemain pertama berada di tingkat maksimal dalam pohon, sedangkan pemain kedua berada di tingkat minimal.
2. Evaluasi pohon permainan: Setiap simpul di pohon permainan dievaluasi berdasarkan keuntungan yang mungkin didapat oleh pemain saat itu. Pada tingkat maksimal, pemain mencari nilai maksimum (keuntungan maksimal), sedangkan pada tingkat minimal, pemain mencari nilai minimum (keuntungan minimal).
3. Strategi pencarian dengan DFS: Algoritma DFS digunakan untuk menjelajahi pohon permainan secara rekursif. Dalam setiap langkah rekursif, algoritma mencoba semua langkah yang mungkin dari keadaan permainan saat itu. Langkah-langkah ini diteruskan secara berulang hingga mencapai kondisi terminal, yaitu keadaan di mana permainan selesai.
4. Backtracking pada pemilihan langkah: Setiap langkah yang mungkin diuji secara berurutan. Jika langkah tersebut mengarah ke keadaan yang menguntungkan, nilai evaluasi akan dikembalikan. Namun, jika langkah tersebut mengarah ke keadaan yang tidak menguntungkan, maka backtracking dilakukan dan langkah selanjutnya yang mungkin diuji.
5. Nilai evaluasi Minimax: Setelah menjelajahi semua kemungkinan langkah pada suatu tingkat, nilai evaluasi Minimax dihitung. Jika pemain berada di tingkat maksimal, maka nilai evaluasi akan menjadi nilai maksimum dari semua nilai evaluasi pada tingkat minimal di bawahnya. Sebaliknya, jika pemain berada di tingkat minimal, nilai evaluasi akan menjadi nilai minimum dari semua nilai evaluasi pada tingkat maksimal di bawahnya.
6. Pemilihan langkah terbaik: Setelah nilai evaluasi Minimax dihitung untuk setiap langkah pada tingkat maksimal, langkah dengan nilai evaluasi terbaik dipilih sebagai langkah yang diambil.

Dengan kombinasi algoritma Minimax, backtracking, dan DFS, kita dapat menemukan langkah terbaik dalam permainan dengan mempertimbangkan keuntungan dan strategi lawan. Algoritma ini sangat penting dalam permainan seperti catur, go, tic-tac-toe, dan sejumlah permainan lainnya.

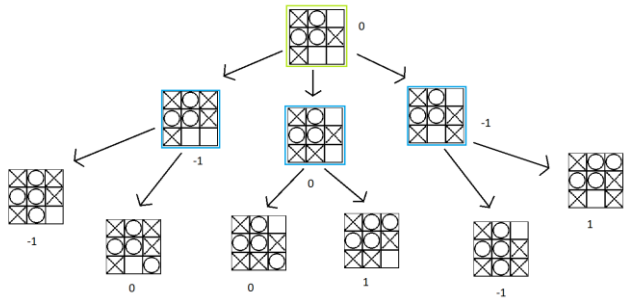
Pada permainan tic-tac-toe, algoritma minimax yang digunakan akan berdasarkan hasil akhir papan. Hasil akhir yang memungkinkan adalah salah satu pemain menang dan seri. Pada algoritma yang diterapkan, pemain dengan simbol ‘X’ akan menjadi maximizer, sedangkan permain dengan symbol ‘O’ akan menjadi minimizer.

Algoritma Minimax pada permainan tic-tac-toe akan memperluas representasi pohon permainan dan evaluasi simpul dalam konteks permainan ini. Dalam tic-tac-toe, pohon permainan dapat direpresentasikan sebagai semua kemungkinan langkah yang mungkin dilakukan pada papan permainan. Setiap simpul dalam pohon mewakili keadaan papan permainan pada suatu titik tertentu.

Proses evaluasi simpul juga berbeda dalam permainan tic-tac-toe. Pada setiap simpul, evaluasi dilakukan berdasarkan hasil akhir papan permainan. Jika pemain 'X' menang, simpul tersebut akan diberi nilai 1 sebagai nilai maksimal yang bisa dicapai oleh maximizer. Jika pemain 'O' menang, simpul tersebut akan diberi nilai -1 sebagai nilai minimal yang bisa dicapai oleh minimizer. Jika permainan berakhir dengan hasil seri, simpul tersebut akan diberi nilai 0.

Setelah nilai evaluasi Minimax dihitung untuk setiap langkah pada tingkat maksimal, langkah dengan nilai evaluasi terbaik akan dipilih sebagai langkah yang diambil. Dalam konteks permainan tic-tac-toe, langkah dengan nilai evaluasi maksimum akan dipilih oleh pemain 'X', sementara langkah dengan nilai evaluasi minimum akan dipilih oleh pemain 'O'.

Dengan menerapkan algoritma Minimax yang disesuaikan dengan permainan tic-tac-toe, kita dapat menemukan langkah terbaik yang mungkin dalam permainan ini, dengan mempertimbangkan keuntungan dan strategi lawan.



Gambar 4 Ilustrasi Algoritma Minimax pada Permainan Tic-Tac-Toe

III. Pembahasan dan Analisis

A. Penentuan Kondisi Papan

Sebelum mengaplikasikan algoritma pencarian langkah terbaik pada suatu kondisi papan, diperlukan fungsi untuk menentukan pemenang (jika ada).

```
def check_winner(self):
    for row in self.board:
        if row.count(row[0]) == 3 and row[0] != ' ':
            return row[0]

    for col in range(3):
        if self.board[0][col] == self.board[1][col] == self.board[2][col] != ' ':
            return self.board[0][col]

    if self.board[0][0] == self.board[1][1] == self.board[2][2] != ' ':
        return self.board[0][0]
    if self.board[0][2] == self.board[1][1] == self.board[2][0] != ' ':
        return self.board[0][2]

    return None
```

```
def evaluate_board(self):
    winner = self.check_winner()
    if winner == 'X':
        return 1
    elif winner == 'O':
        return -1
    else:
        return 0
```

Gambar 5 dan 6 Algoritma Penentuan Pemenang

Fungsi penentuan pemenang dibagi menjadi 2 bagian, fungsi *check_winner* dan fungsi *evaluate_board*. Fungsi *check_winner* akan memeriksa apabila terdapat pemenang pada kondisi papan pada suatu saat, secara horizontal, vertical, maupun diagonal. Kemudian fungsi *evaluate_board* akan mengembalikan suatu nilai, tergantung terhadap kembalian fungsi *check_winner*.

B. Algoritma Minimax

Setelah didapatkan kondisi papan pada suatu saat tertentu, dapat ditentukan suatu *value* sebuah papan melalui algoritma minimax

```
def minimax(self, depth, maximizing_player):
    score = self.evaluate_board()

    if score == 1 or score == -1:
        return score

    if self.is_board_full():
        return 0

    if maximizing_player:
        max_score = float('-inf')
        for cell in self.get_empty_cells():
            row, col = cell
            self.board[row][col] = 'X'
            score = self.minimax(depth + 1, False)
            self.board[row][col] = ' '
            max_score = max(max_score, score)
        return max_score
    else:
        min_score = float('inf')
        for cell in self.get_empty_cells():
            row, col = cell
            self.board[row][col] = 'O'
            score = self.minimax(depth + 1, True)
            self.board[row][col] = ' '
            min_score = min(min_score, score)
        return min_score
```

Gambar 7 Algoritma Minimax

Algoritma minimax yang tersusun akan menerima hasil fungsi *evaluate_board*. Hasil tersebut pertama akan diperiksa apabila sudah ada pemenang. Jika ada, akan dikembalikan nilai hasil tersebut. Jika papan sudah penuh namun tidak ada pemenang, akan dikembalikan nilai 0 untuk menandakan kondisi papan adalah seri. Jika papan belum penuh dan belum ada pemenang, akan dilakukan pemeriksaan DFS secara rekursif pada kondisi papan sekarang dengan fungsi *minimax*.

Jika pemain sekarang adalah ‘X’ atau maximizer, akan dilakukan pemeriksaan secara rekursif setiap langkah yang mungkin dapat dilakukan dan mengembalikan nilai paling besar. Jika pemain sekarang adalah ‘O’ atau minimizer, akan dilakukan pemeriksaan secara rekursif setiap langkah yang mungkin dapat dilakukan dan mengembalikan nilai yang paling kecil.

C. Algoritma Langkah Terbaik

Setelah disediakan algoritma minimax, akhirnya dapat dicari langkah terbaik yang dapat dilakukan seorang permain pada suatu saat tertentu.

```
def find_best_move(self):
    best_score = float('-inf') if self.current_player == 'X' else float('inf')
    best_move = None

    for cell in self.get_empty_cells():
        row, col = cell
        self.board[row][col] = self.current_player
        score = self.minimax(0, self.current_player == 'O')
        self.board[row][col] = ' '

        if (self.current_player == 'X' and score > best_score) or (self.current_player == 'O' and score < best_score):
            best_score = score
            best_move = cell

    r1, c1 = best_move
    best_move2 = r1 + 1, c1 + 1

    return best_move2
```

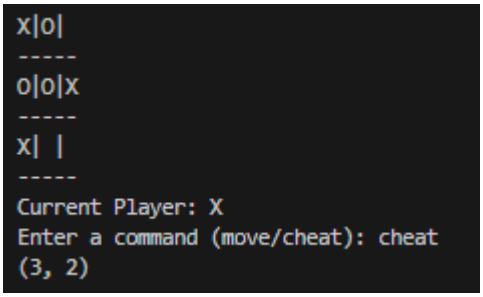
Gambar 8 Algoritma Langkah Terbaik

Fungsi *find_best_move* ini akan mencari langkah terbaik yang memungkinkan pada kondisi papan tertentu oleh pemain tertentu. Fungsi ini akan mencari semua langkah yang mungkin dilakukan dan memeriksa nilai minimax papan apabila

dilakukan langkah tersebut. Semua nilai minimax akan dibandingkan dan dicari nilai paling optimal (tergantung pada giliran pemain).

D. Analisis Algoritma

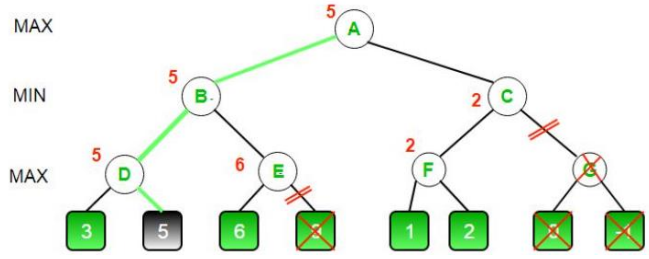
Setiap algoritma yang dibutuhkan sudah diterapkan dan dapat langsung diaplikasikan. Jika diujikan sesuai dengan gambar 6, maka akan didapatkan hasil yang sesuai dengan ilustrasi tersebut.



Gambar 9 Pengujian 1

Algoritma yang diterapkan sudah cukup optimal dengan algoritma minimax, namun algoritma masih cukup memakan memori komputer. Algoritma dapat dikembangkan lebih lanjut menggunakan konsep *alpha-beta pruning* atau *pruning* singkatnya.

Alpha-Beta Pruning adalah sebuah teknik atau fungsi yang digunakan untuk meningkatkan kinerja algoritma backtracking (minimax). Dengan menggunakan metode ini, program dapat membatasi dan menghilangkan node-node yang tidak mungkin mengarah ke solusi, sehingga hanya node-node yang berpotensi menuju solusi yang tetap dipertahankan. Alpha-Beta Pruning menjamin bahwa hasil yang diberikan akan sama dengan algoritma minimax biasa. Dengan menerapkan metode ini, program dapat melakukan pencarian solusi dengan lebih efisien daripada tanpa menggunakan Alpha-Beta Pruning. Sebagai hasilnya, program dapat melakukan pencarian ke kedalaman yang lebih dalam.



Gambar 10 Alpha-Beta Pruning

Dalam setiap tahap, untuk setiap anak dari suatu node, jika ditemukan node dengan nilai yang lebih baik, node tersebut secara langsung diabaikan, sehingga hanya menyisakan anak node terbaik. "Lebih baik" di sini berarti nilai yang lebih tinggi dalam fase Maximizer atau nilai yang lebih rendah dalam fase Minimizer.

Selain dari optimasi algoritma, terdapat kelemahan kecil dalam pencarian solusi yang mungkin. Saat suatu papan memiliki nilai yang sama dan setiap langkah akan menuju pada kekalahan pemain, algoritma tidak akan memberikan informasi bahwa kekalahan telah mutlak.

Kemudian solusi yang diberikan tidak memeriksa kedalaman atau depth. Algoritma hanya akan mengembalikan langkah pertama ditemukan yang memiliki nilai minimax yang optimal. Oleh karena itu, dapat terjadi kasus dimana langkah terbaik yang diberikan dapat menjamin kemenangan dalam n langkah. Namun, langkah yang sebenarnya terbaik menjamin kemenangan kurang dari n langkah. Hal ini disebabkan algoritma minimax yang hanya memberikan 3 status yang memungkinkan pada suatu papan, yaitu -1, 0 dan 1.

Gambar 11 Pengujian 2

Pada gambar 11, dapat diamati bahwa pemain ‘X’ dapat langsung memenangkan permainan apabila menempati lokasi (2,1), namun algoritma memberikan kembalian (1,2) karena kedua solusi memiliki nilai yang sama namun karena cara tersusunnya algoritma, hanya solusi pertama yang ditemukan yang memiliki nilai minimax teroptimal yang dikembalikan. Hal ini dapat dikomplekskan dengan mengimplementasi nilai depth pada perhitungan kembalian nilai pada fungsi minimax.

IV. KESIMPULAN DAN SARAN

A. Kesimpulan

Penerapan algoritma DFS dan *backtracking* dapat memberikan langkah atau solusi terbaik dalam situasi tertentu pada permainan Tic-Tac-Toe. Solusi yang dihasilkan oleh algoritma tidak selalu mengembalikan solusi yang optimum global karena kurangnya kompleksitas algoritma dan keterbatasan komputer. Penggunaan algoritma *backtracking*, *minimax*, dan DFS sudah cukup optimal dalam penyelesaian masalah ini, namun dapat dikembangkan lagi dengan mengaplikasikan konsep *alpha-beta pruning* atau *pruning*.

Algoritma yang digunakan dalam penyelesaian masalah ini banyak digunakan untuk mencari solusi pada permainan 2 pemain yang bergiliran. Hal ini dikarenakan algoritma *minimax* yang memiliki 2 state, minimizing dan maximizing. Solusi ini jauh lebih optimum daripada solusi lain yang bersifat exhaustive total seperti *Brute Force* karena memiliki waktu pencarian yang lebih singkat dan tidak perlu memeriksa setiap state papan yang mungkin.

B. Saran

Dalam penyusunan makalah, penulis menyadari bahwa masih banyak terdapat kekurangan, baik dalam penulisan makalah maupun penerapan algoritma. Oleh karena itu,

terdapat beberapa saran yang dapat membuat makalah dan algoritma menjadi lebih bagus. Pertama, program dapat dibuat menjadi lebih optimal, dengan mengaplikasikan algoritma-algoritma kompleks yang dapat memberikan solusi yang paling optimal. Kedua, memperluas pengetahuan mengenai algoritma yang serupa agar dapat mengoptimalkan algoritma. Ketiga, memperbanyak visualisasi kerja algoritma agar algoritma yang dibentuk dapat dipahami lebih lanjut.

C. Ucapan Terimakasih

Terima kasih dan puji syukur penulis sampaikan kepada Tuhan Yang Maha Esa, atas rahmat-Nya yang telah memungkinkan penulis menyelesaikan makalah ini. Makalah ini disusun sebagai bagian dari tugas mata kuliah IF2211 Strategi Algoritma pada semester 4 tahun 2022/2023.

Penulis ingin mengucapkan terima kasih kepada beberapa pihak yang telah memberikan dukungan dan bantuan selama penulisan makalah ini, antara lain:

1. Dosen-dosen pengajar mata kuliah IF2211 Strategi Algoritma, yang telah memberikan pembelajaran dan bimbingan yang berharga.
2. Orang tua penulis, atas dukungan, doa, dan motivasi yang tidak henti-hentinya diberikan.
3. Teman dan sahabat penulis, yang telah memberikan dorongan, masukan, dan diskusi yang berarti.

Makalah IF2211 Strategi Algoritma, Semester II Tahun 2021/2022 berhasil diselesaikan berkat bantuan dari pihak-pihak tersebut. Penulis menyadari bahwa dalam penulisan makalah ini masih terdapat kekurangan-kekurangan tertentu. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang dapat diberikan guna menyempurnakan karya tulis ilmiah ini.

Penulis ingin menyampaikan terima kasih yang sebesar-besarnya kepada semua yang telah membantu dalam penulisan makalah ini, dan berharap makalah ini dapat memberikan manfaat bagi siapa saja yang membutuhkannya.

REFERENSI

Munir, Rinaldi dan Maulidevi, Nur (2021). *Breadth/Depth First Search (BFS/DFS) Bagian 1*. Diakses 18 Mei 2023, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

Munir, Rinaldi dan Maulidevi, Nur (2021). *Breadth/Depth First Search (BFS/DFS) Bagian 2*. Diakses 18 Mei 2023, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

GeeksforGeeks (2023). *Depth First Search or DFS for a Graph*. Diakses 18 Mei 2023, dari https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

GeeksforGeeks (2023). *Backtracking Algorithms*. Diakses 18 Mei 2023, dari https://www.geeksforgeeks.org/backtracking-algorithms/?ref=gcse

GeeksforGeeks. *Minimax Algorithm in Game Theory | Set 1 (Introduction)*. Diakses 18 Mei 2023, dari https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

GeeksforGeeks. *Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning)*. Diakses 18 Mei 2023, dari https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

Javatpoint. *Mini-Max Algorithm in Artificial Intelligence*. Diakses 18 Mei 2023, dari https://www.javatpoint.com/mini-max-algorithm-in-ai

Spanning Tree. *Minimax: How Computers Play Games*. Diakses 19 Mei 2023, dari https://www.youtube.com/watch?v=SLgZhpDsrfc

PRANALA PENTING

Source code: https://github.com/maximatey/TicTacToe

Pranala Video: https://bit.ly/VidTugasMakalah061

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

Ttd

Alex Sander 13521061