Implementasi Algoritma A* untuk Menemukan Gerakan Terbaik pada Permainan Tic-Tac-Toe

Diero Arga Purnama - 13522056
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13522056@std.stei.itb.ac.id

***Abstract*—Permainan Tic-Tac-Toe merupakan permainan strategi yang dimainkan oleh dua pemain di papam berukuran 3x3. Para pemain akan berganti giliran meletakkan tanda mereka pada kotak yang kosong hingga terdapat pemain yang telah memenuhi sebuah barisan dengan tanda mereka atau tidak terdapat lagi kotak kosong. Makalah ini akan membahas implementasi algoritma A* dalam permainan ini untuk menentukan gerakan terbaik yang dapat dilakukan oleh pemain. Pemilihan algoritma A* dilakukan karena kemampuan algoritma A* dalam menemukan jalur optimal menggunakan kombinasi dari jumlah biaya yang dibutuhkan oleh sebuah rute dan estimasi biaya untuk mencapai ke tujuan rute tersebut.**

Keywords—tic-tac-toe; A*

I. PENDAHULUAN

Permainan Tic-Tac-Toe merupakan permainan merupakan sebuah permainan papan yang sangat populer dan dikenal oleh banyak orang di seluruh dunia. Permainan ini dimainkan oleh dua pemain di papan berukuran 3x3, kedua pemain bergantian giliran meletakkan tanda mereka, X atau O, pada kotak yang kosong hingga terdapat pemain yang memenuhi sebuah barisan dengan tanda mereka, atau tidak terdapat lagi kotak kosong.

Algoritma A* adalah sebuah algoritma pencarian dalam graf berbobot yang menggabungkan pendekatan heuristik dari algoritma seperti *Greedy Best First Search* (Greedy BFS) dengan pendekatan pencarian jalur terpendek seperti dari algoritma *Uniform Cost Search* (UCS). Algoritma ini akan mengutamakan rute yang memiliki jumlah nilai heuristik dengan biaya yang paling rendah untuk menemukan jalur terpendek dengan biaya yang minimum. Dengan kombinasi ini, algoritma A* dapat menemukan solusi yang optimal dengan efisiensi yang tinggi.

Makalah ini bertujuan untuk mengimplementasikan algoritma A* untuk membuat sebuah program yang dapat menentukan gerakan terbaik bagi pemain dalam sebuah permainan Tic-Tac-Toe. Implementasi ini akan melibatkan penentuan fungsi heuristik yang tepat agar program dapat memberikan gerakan yang tidak hanya membawa pemain lebih dekat menuju kemenangan, tetapi juga mencegah terjadinya kekalahan.

Dengan demikan, diharapkan bahwa makalah ini dapat memberikan kontribusi dalam pengembangan dan pemahaman terhadap penggunaan algoritma dalam permainan seperti Tic-Tac-Toe.

II. DASAR TEORI

A. *Permainan Tic-Tac-Toe*

Permainan Tic-Tac-Toe merupakan permainan papan klasik yang seringkali dimainkan oleh kalangan anak-anak karena peraturannya yang sederhana dan mudah untuk dimengerti. Walaupun permainan ini memiliki peraturan yang sederhana, dibutuhkan strategi dan pengambilan keputusan yang baik untuk meraih kemenangan secara konsisten.

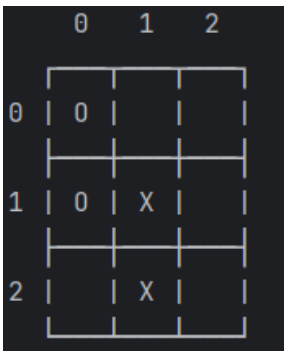

Fig. 1. Permainan Tic-Tac-Toe

Sumber: Dokumen Penulis

Jejak pertama permainan Tic-Tac-Toe dapat dijejak kembali ke Mesir Kuno, dimana terdapat sisa-sisa dari permainan papan 3x3 pada ubin atap dari sekitar tahun 1300 SM. Permainan ini dikenal dengan banyak nama, seperti *noughts and crosses*, *Xs and Os*, *Three-in-a-*Row, dan banyak lagi, nama "Tic-Tac-Toe" pertama kali digunakan pada abad-20. Permainan ini juga merupakan salah satu *video game* yang pertama. [2]

Permainan Tic-Tac-Toe dimainkan di sebuah papan permainan berukuran 3x3. Kotak-kotak pada papan permainan dapat berisi 'X', 'O', ataupun kosong. Pada awal permainan, semua kotak pada papan permainan adalah kosong.

Permainan ini dimainkan oleh dua orang, satu bermain sebagai 'X' dan yang lain bermain sebagai 'O'. Kedua pemain akan bergantian giliran untuk meletakkan tandanya di papan

permainan. Proses ini akan diulang secara terus menerus hingga permainan berakhir.

Permainan berakhir ketika terdapat suatu pemain yang telah memenuhi barisan dengan tandannya, baris ini dapat berupa baris horizontal, vertikal, maupun diagonal, jika hal ini terjadi, maka permainan dimenangkan oleh pemain tersebut. Jika semua kotak kosong telah diisi oleh pemain tanpa adanya barisan penuh yang dipenuhi oleh satu tanda, maka permainan akan berakhir tanpa adanya pemenang (seri).

Strategi yang umum digunakan dalam permainan ini adalah untuk mencoba memenuhi baris dengan tanda sendiri dan juga untuk mencegah lawan memenuhi baris dengan tandanya. Gerakan terbaik dalam konteks permainan Tic-Tac-Toe adalah gerakan yang mempertimbangkan tidak hanya keuntungan jangka pendek, tetapi juga potensi reaksi lawan dan kemungkinan hasil akhir dari permainan.

B. *Graf*

Sebuah graf adalah sebuah struktur matematika yang terdiri atas simpul dan noda yang dihubungkan oleh sisi atau *edge*. Simpul dalam graf mewakili sebuah objek atau entitas, sedangkan sisi merepresentasikan hubungan antara objek-objek tersebut.

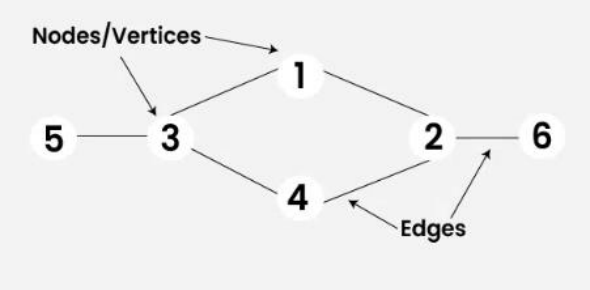

Fig. 2. Ilustrasi Graf

Sumber: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

C. *Graf berbobot*

Graf berbobot merupakan salah satu jenis graf dimana setiap sisi akan dikaitkan sebuah nilai yang merepresentasikan bobot dari sisi tersebut. Bobot ini menggambarkan biaya atau jarak yang diperlukan untuk mencapai simpul tujuan dari simpul awal.

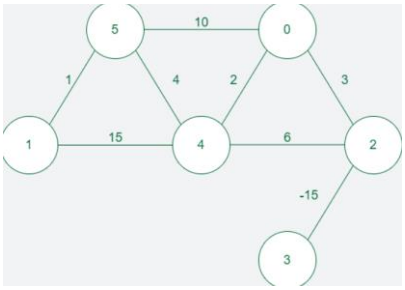

Fig. 3. Ilustrasi Graf Berbobot

Sumber: https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-weighted-graph/

D. *Algoritma A**

Algoritma A* merupakan sebuah algoritma pencarian jalur yang digunakan untuk menemukan jalur optimal dalam graf berbobot. Algoritma ini menggabungkan pendekatan heuristik dari algoritma seperti *Greedy Best First Search* dan pendekatan pencarian jalur terpendek seperti dari algoritma *Uniform Cost Search*. Algoritma ini akan mengutamakan simpul dalam graf yang memiliki jumlah dari nilai heurisitik dengan biaya terendah.

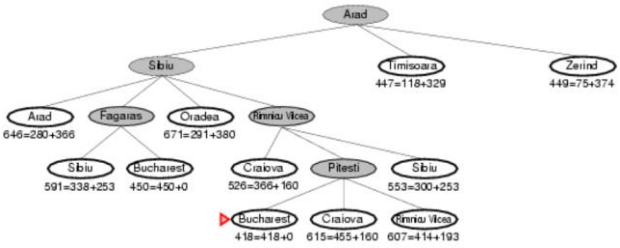

Fig. 4. Ilustrasi Algoritma A*

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

Algoritma ini dimulai dengan membuat dua himpunan untuk menyimpan simpul hidup yaitu simpul yang belum diekspan dan simpul mati untuk menyimpan simpul yang teleah diekspan. Simpul pertama dimasukkan ke dalam himpunan simpul hidup untuk memulai pencarian. Selama masih terdapat simpul hidup, simpul dengan nilai $f(n)$ terkecil akan dipilih untuk diekspansi, nilai $f(n)$ didapatkan dengan rumus berikut,

$$f(n) = g(n) + h(n) \quad (1)$$

dengan $g(n)$ merupakan biaya yang dibutuhkan untuk mencapai simpul n dan $h(n)$ merupakan estimasi biaya yang dibutuhkan untuk mencapai tujuan dari simpul n. Jika simpul yang diekspan tersebut merupakan simpul tujuan, jalur optimal ditemukan dan algoritma berhenti, jika tidak, semua tetangga dari simpul tersebut yang belum diekspan akan dimasukkan ke dalam himpunan simpul hidup untuk diekspan. Algoritma berakhir ketika jalur optimal telah ditemukan atau simpul hidup habis dan tidak ada jalur dari simpul awal ke simpul akhir.

Dalam konteks permainan Tic-Tac-Toe, karena penggunaan pendekatan heuristik biasa, yaitu estimasi jumlah langkah yang diperlukan menuju posisi tujuan tidak memungkinkan untuk digunakan karena pendekatan tersebut tidak mempertimbangkan gerakan yang dapat dimainkan oleh lawan sehingga dapat memberikan solusi yang sangat tidak optimal, pendekatan heuristik atau nilai $h(n)$ yang digunakan adalah untuk mengevaluasi potensi kemenangan dan potensi kekalahan dari sebuah posisi. Potensi kemenangan dan kekalahan dihitung dari panjang rantai ‘X’ dan ‘O’ yang ada dalam papan permainan, semakin panjang rantai yang dimiliki pemain, maka nilai heuristik akan semakin tinggi, demikian pula sebaliknya. Sedangkan nilai $g(n)$ yang didgunakan merupakan jumlah gerakan yang dibutuhkan untuk mencapai posisi tersebut. Dengan mengevaluasi nilai $f(n)$ dengan metode tersebut, diharapkan program dapat memberikan gerakan yang terbaik untuk posisi tersebut.

III. HASIL PENELITIAN

Untuk melakukan pengujian, penulis membuat sebuah program sederhana permainan Tic-Tac-Toe berbasis *Command Line Interface* (CLI) menggunakan bahasa pemrograman Java. Program ini akan menampilkan papan permainan beserta gerakan terbaik yang ditentukan oleh algoritma A* pada setiap giliran.

A. *Kelas Game*

Kelas ini dibuat untuk merepresentasikan permainan Tic-Tac-Toe. Atribut-atribut dalam kelas ini adalah data, yaitu sebuah matriks karakter yang digunakan sebagai papan permainan, turn untuk menyimpan giliran pemain, ongoing untuk menyimpan apakah game masih berjalan atau sudah berhenti, empty slots untuk menyimpan jumlah kotak yang masih kosong serta xTurns dan oTurns yaitu jumlah giliran yang telah diambil tiap-tiap pemain yang akan digunakan untuk menghitung $g(n)$ dari algoritma A*.

```
public class Game {  34 usages
    /*
     *  ATTRIBUTES
     */
    public final Matrix<Character> data;
    public char turn;  24 usages
    public boolean ongoing;  10 usages
    public int emptySlots;  3 usages
    public int xTurns;  5 usages
    public int oTurns;  5 usages

    public static final Game INSTANCE = new Game();  1 usage
```

Fig. 5. Atribut dalam Kelas Game

Sumber: Dokumen penulis

Dibuat juga sebuah fungsi displayBoard yang digunakan untuk menampilkan papan permainan. Guna fungsi ini adalah untuk memberitahu pemain mengenai keadaan papan permainan pada saat ini. Fungsi ini akan dipanggil pada setiap pergantian giliran pemain.

```
public void displayBoard() {  2 usages
    System.out.println("    0   1   2");
    System.out.println("  ┌───┬───┬───┐");
    for (int row = 0; row < 3; row++) {
        System.out.print(row + " | ");
        for (int col = 0; col < 3; col++) {
            char currentChar = this.data.getEl(row, col);

            System.out.print(currentChar + " | ");
        }
        if (row != 2)
            System.out.println("\n  ├───┼───┼───┤");
    }
    System.out.println("\n  └───┴───┴───┘");
}
```

Fig. 6. Fungsi displayBoard

Sumber: Dokumen penulis

Terdapat juga fungsi place yang akan menerima masukan baris dan kolom kemudian meletakkan tanda pemain pada papan permainan di indeks yang diberikan jika valid. Fungsi ini juga akan mengecek apakah permainan sudah dimenangkan oleh salah satu pemain dengan memanggil fungsi checkWinCondition. Jika jumlah kotak kosong pada papan permainan sudah habis, maka atribut *ongoing* akan di-*set* ke *false* untuk menandakan bahwa permainan telah berakhir.

```
public void place(int row, int col) throws IndexOutOfBoundsException {  2 usages
    // Validasi input
    if (row < 0 || row >= 3 || col < 0 || col >= 3)
        throw new IndexOutOfBoundsException("Index yang dimasukkan tidak valid!");

    if (this.data.getEl(row, col) != ' ')
        throw new IndexOutOfBoundsException("Hanya bisa meletakkan di kotak kosong!");

    this.data.setEl(row, col, this.turn);
    this.emptySlots--;

    if (this.turn == 'X') xTurns++;
    else oTurns++;

    checkWinCondition();

    if (this.ongoing)
        nextTurn();

    if (this.emptySlots == 0)
        this.ongoing = false;
}
```

Fig. 7. Fungsi place

Sumber: Dokumen penulis

Agar algoritma A* dapat diaplikasikan, terlebih dahulu diperlukan sebuah fungsi untuk menentukan kondisi akhir sebuah permainan, yaitu ketika permainan telah dimenangkan oleh salah satu pemain.

```
public boolean checkWinCondition() {  3 usages
    // Cek baris dan kolom
    for (int idx = 0; idx < 3; idx++)
        if ((this.data.getEl(idx,  col: 0) == this.turn &&
             this.data.getEl(idx,  col: 1) == this.turn &&
             this.data.getEl(idx,  col: 2) == this.turn) ||
            (this.data.getEl( row: 0, idx) == this.turn &&
             this.data.getEl( row: 1, idx) == this.turn &&
             this.data.getEl( row: 2, idx) == this.turn)) {
            this.ongoing = false;
            return true;
        }

    // Cek diagonal
    if ((this.data.getEl( row: 0,  col: 0) == this.turn &&
         this.data.getEl( row: 1,  col: 1) == this.turn &&
         this.data.getEl( row: 2,  col: 2) == this.turn) ||
        (this.data.getEl( row: 0,  col: 2) == this.turn &&
         this.data.getEl( row: 1,  col: 1) == this.turn &&
         this.data.getEl( row: 2,  col: 0) == this.turn)) {
        this.ongoing = false;
        return true;
    }
    return false;
}
```

Fig. 8. Fungsi checkWinCondition

Sumber: Dokumen penulis

Fungsi checkWinCondition akan mengembalikan *true* jika permainan telah dimenangkan oleh sebuah pemain, yaitu ketika terdapat sebuah barisan, kolom, atau diagonal yang telah

dipenuhi oleh tanda pemain tersebut, dan mengembalikan *false* jika permainan belum dimenangkan oleh siapa-siapa. Atribut *ongoing* juga akan di-*set* ke *false* untuk menandakan bahwa permainan telah berakhir.

B. Kelas AlgoritmaA

Kelas ini dibuat untuk menyimpan data-data yang diperlukan untuk melakukan pencarian gerakan terbaik menggunakan algoritma A*. Kelas ini memiliki 3 atribut, pq, yaitu sebuah *priority queue* yang menyimpan noda, noda akan diurutkan berdasarkan elemen *score*-nya dari kecil ke besar, atribut kedua yaitu visited yang merupakan sebuah *hash set* yang digunakan untuk menyimpan noda-noda yang telah dikunjungi, dan atribut terakhir yaitu currentNode yaitu noda yang sedang diekspansi.

```
public class AlgoritmaA {  3 usages
    /*
     *  ATTRIBUTES
     */
    private final PriorityQueue<Node> pq;  5 usages
    private final HashSet<Node> visited;  4 usages
    private Node currentNode;  10 usages
```

Fig. 9. Atribut dalam Kelas AlgoritmaA

Sumber: Dokumen penulis

Untuk memudahkan implementasi algoritma ini, dibuat sebuah kelas tambahan Noda, yang berisi sebuah game, skor yaitu nilai $f(n)$ dari game tersebut, dan prev yang menyimpan jalur yang diambil untuk mencapai noda tersebut.

```
public class Node {  18 usages
    /*
     *  ATTRIBUTES
     */
    public Game game;  9 usages
    public int score;  4 usages
    public List<Game> prev;  6 usages
```

Fig. 10. Atribut dalam Kelas Node

Sumber: Dokumen penulis

Program akan memanggil fungsi start yang akan memulai pencarian gerakan terbaik pada posisi saat ini menggunakan algoritma A*. Pertama-tama, akan dibuat sebuah noda dari game dan dimasukkan ke dalam *priority queue*. Setelah itu, program akan melakukan *loop* hingga *priority queue* kosong atau ditemukan kondisi game dimana pemain memenangkan permainan.

Setelah ditemukan noda dengan kondisi game dengan ketentuan tersebut, fungsi akan mengecek apakah noda pernah melewati game dengan skor lebih dari 100, jika iya, maka program tidak akan mengembalikan noda tersebut dan melanjutkan pencarian.

```
public Node start() {  1 usage
    currentNode = new Node(Game.getInstance(), new ArrayList<>());
    addToPq(currentNode);

    while (!pq.isEmpty()) {
        if (currentNode.game.checkWinCondition() && currentNode.game.turn == Game.getInstance().getTurn()) {
            boolean flag = true;
            for (int i = 0; i < currentNode.prev.size(); i++)
                if (i % 2 == 1)
                    if (Node.calcScore(currentNode.prev.get(i)) > 100) flag = false;

            if (flag)
                return currentNode;
        }

        try { next(); }
        catch (Exception ignored) { }
    }
    return null;
}
```

Fig. 11. Fungsi start

Sumber: Dokumen penulis

Dalam fungsi start diatas, dipanggil fungsi next pada setiap iterasinya, fungsi tersebut berfungsi untuk mengambil noda terdepan pada *priority queue*, melakukan ekspansi terhadap noda tersebut, kemudian menambahkan semua tetangga dari noda tersebut yang belum pernah diekspansi ke dalam *priority queue*.

Dalam konteks permainan Tic-Tac-Toe, tetangga dari tiap-tiap noda merupakan semua gerakan yang mungkin dilakukan oleh pemain pada giliran tersebut. Gerakan-gerakan tersebut didapatkan melalui fungsi findEveryMove yang akan mengembalikan sebuah larik yang berisi keadaan game setelah gerakan tersebut dimainkan.

```
public void next() throws NullPointerException {  1 usage
    if (pq.isEmpty()) {
        throw (new NullPointerException());
    }

    Node head = getHead();
    this.currentNode = head;

    if (this.visited.contains(head))
        return;
    this.visited.add(head);

    List<Game> nextGames = BestMoveFinder.findEveryMove(this.currentNode.game);
    for (Game g : nextGames) {
        ArrayList<Game> prevList = new ArrayList<>(head.prev);
        prevList.add(this.currentNode.game);
        Node temp = new Node(g, prevList);
        if (this.visited.contains(temp))
            continue;

        addToPq(temp);
    }
}
```

Fig. 12. Fungsi next

Sumber: Dokumen penulis

Setelah noda dimasukkan ke dalam *priority queue* oleh fungsi next, tiap-tiap noda akan diurutkan berdasarkan nilai score yang terdapat pada noda tersebut. Nilai score merupakan nilai $f(n)$ dari algoritma A*. Untuk menghitung nilai score digunakan calcScore yang akan mengembalikan hasil penjumlahan $g(n)$ dengan $h(n)$. Sebagai nilai $g(n)$ atau *cost so far to reach n* digunakan atribut xTurns atau oTurns dari kelas Game yaitu jumlah giliran yang telah diambil oleh pemain untuk mencapai posisi tersebut dikali dengan sepuluh. Sedangkan untuk nilai $h(n)$ digunakan fungsi evaluatePosition yang akan

mengembalikan “nilai” dari posisi, semakin bagus posisi, maka nilai yang akan dikembalikan akan semakin rendah.

Untuk posisi pemain sudah kalah akan diberikan nilai 100, sedangkan untuk posisi pemain sudah menang akan diberikan nilai 0. Jika permainan masih berlangsung dan belum ada pemenang, maka nilai akan berkurang jika terdapat barisan dengan tanda pemain yang belum ditempati oleh lawan dan nilai akan bertambah jika sebaliknya.

```java
public static int evaluatePosition(Game gameState) {  4 usages  Diero *
    char currentTurn = gameState.getTurn();

    int MAX_SCORE = 100;
    int score = 0;

    HashMap<String, Pair> summary = summarizeState(gameState);
    for (Pair pair : summary.values()) {
        int val = currentTurn == 'X' ? pair.second : pair.first;
        int valEnemy = currentTurn == 'X' ? pair.first : pair.second

        if (val == 3 || valEnemy == 3) {
            if (val == 3) score = MAX_SCORE;            // KALAH
            else score = 0;                             // MENANG
            break;
        }

        if (valEnemy == 2 && val == 0) {
            score = MAX_SCORE;                          // KALAH
            break;
        }

        if (valEnemy == 0) {
            if (val == 1) score -= 10;
            else if (val == 2) score -= 50;
        }

        if (val == 0 && valEnemy == 1)
            score += 10;
    }
    return score;
}
```

Fig. 13. Fungsi evaluatePosition

Sumber: Dokumen penulis

Fungsi evaluatePosition diatas akan menerima gambaran umum keadaan pada papan permainan melalui sebuah *hash map* yang didapatkan melalui fungsi summarizeState. Fungsi summarizeState akan menelusuri setiap kotak dalam papan permainan dan mencatat posisi semua tanda dalam papan ke dalam suatu *hash map*. Sebagai contoh, jika terdapat dua ‘X’ dan satu ‘O’ dalam barisan pertama, maka dalam hashmap akan dicatat nilai (2, 1) untuk barisan pertama. Setelah semua kotak dalam papan permainan telah ditelusuri, maka fungsi akan mengembalikan *hash map* yang berisi nilai-nilai tersebut.

```
    0   1   2
  ┌───┬───┬───┐
0 | X | O | X |
  ├───┼───┼───┤
1 |   | O |   |
  ├───┼───┼───┤
2 |   |   |   |
  └───┴───┴───┘
row1: (0, 1).
row0: (2, 1).
diag1: (1, 1).
diag2: (1, 1).
col2: (1, 0).
row2: (0, 0).
col0: (1, 0).
col1: (0, 2).
```

Fig. 14. Tampilan Hash Map yang Diberikan Fungsi summarizeState

Sumber: Dokumen penulis

C. *Kelas Main*

Kelas main digunakan untuk menjalankan seluruh kelas-kelas yang telah dibuat sebagai program yang utuh. Main akan mengambil instansiasi dari game, menampilkan papan, menerima input, serta menampilkan gerakan terbaik yang didapatkan dari algoritma A*.

```java
public class Main {
    public static void main(String[] args) {
        Game game = Game.getInstance();
        game.displayBoard();
        Scanner sc = new Scanner(System.in);
        while (game.ongoing) {
            int[] inputs = new int[2];
            game.takeInput(inputs, sc);
            try {
                game.place(inputs[0], inputs[1]);
                game.displayBoard();
                if (game.ongoing) {
                    AlgoritmaA algoritmaA = new AlgoritmaA();
                    Node solution = algoritmaA.start();
                    if (solution != null) {
                        Game firstMove;
                        if (solution.prev.size() == 1) firstMove = solution.game;
                        else firstMove = solution.prev.get(1);
                        for (int row = 0; row < 3; row++)
                            for (int col = 0; col < 3; col++)
                                if (firstMove.getData().getEl(row, col) != game.getData().getEl(row, col)) {
                                    System.out.println("Best move: (" + row + ", " + col + ")");
                                    break;
                                }
                    } else System.out.println("Tidak ada gerakan yang berakhir dengan kemenangan pemain.");
                }
            } catch (IndexOutOfBoundsException e) { System.out.println(e.getMessage()); }
        }
        if (game.checkWinCondition()) System.out.println("Permainan dimenangkan oleh " + game.turn + "!");
        else System.out.println("Permainan berakhir tanpa pemenang.");
        sc.close();
    }
}
```

Fig. 15. Kelas Main

Sumber: Dokumen penulis

D. *Analisis Algoritma*

Algoritma A* akan mengambil posisi pada saat ini dan mencari gerakan yang akan berakhir dengan kemenangan pemain. Program akan memberikan gerakan yang mencegah

lawan untuk menang dan memberikan peluang kemenangan terbesar.

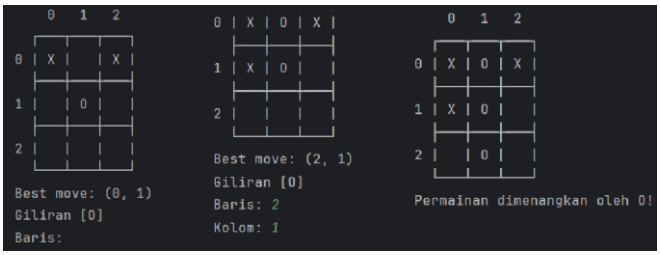

Fig. 16. Contoh Program Berjalan

Sumber: Dokumen penulis

Karena algoritma A* akan melakukan simulasi permainan untuk mencari *outcome* terbaik dari permainan untuk pemain, algoritma ini memiliki kekurangan dimana terkadang gerakan yang dipilih untuk lawan dalam simulasi tersebut merupakan gerakan buruk yang tidak mungkin akan dimainkan dalam permainan sebenarnya sehingga yang diberikan oleh algoritma belum tentu optimal.

Kekurangan lain algoritma A* untuk mencari gerakan terbaik dalam permainan Tic-Tac-Toe dalam implementasi yang dibuat penulis adalah karena algoritma hanya dapat memberikan saran gerakan jika kemenagan mungkin terjadi dalam posisi tersebut. Jika posisi akan berakhir dengan seri atau dengan kekalahan pemain, algoritma tidak dapat memberikan saran untuk gerakan terbaik dan akan mengembalikan *null*.

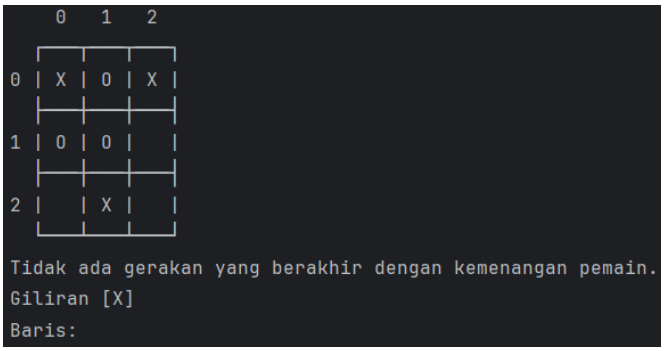

Fig. 17. Contoh Program Berjalan

Sumber: Dokumen penulis

IV. Kesimpulan

Algoritma A* merupakan algoritma yang efektif dan efisien dalam menemukan jalan optimal dalam suatu graf berbobot. Dalam permainan Tic-Tac-Toe, sulit untuk memetakan elemen-elemen dalam algoritma A* kedalam elemen-elemen permainan. Hal ini dikarenakan algoritma A* memperlukan tujuan akhir yang jelas untuk memberikan solusi yang optimal, sedangkan dalam permainan Tic-Tac-Toe, keadaan akhir dari permainan tidak pasti karena terdapat banyak variasi permainan tergantung gerakan yang dipilih oleh lawan, sehingga algoritma A* belum tentu memberikan solusi yang optimal. Kekurangan lain dari penggunaan algoritma A* dalam menemukan gerakan terbaik, setidaknya dalam implementasi yang dibuat oleh penulis, adalah karena program akan mencari keadaan akhir dimana pemain memenangkan permainan, jika permainan akan berakhir dengan seri, program tidak dapat memberikan solusi.

Secara keseluruhan, algoritma A* dapat digunakan untuk menemukan gerakan terbaik dalam permainan Tic-Tac-Toe, tetapi solusi yang diberikan oleh algoritma ini memiliki banyak kekurangan karena pemetaan elemen permainan Tic-Tac-Toe kepada elemen algoritma A* tidak dapat dilakukan dengan sempurna. Sehingga, akan lebih baik jika algoritma atau metode lain digunakan untuk membuat sebuah program yang dapat memberikan gerakan terbaik pada permainan Tic-Tac-Toe.

Pranala Youtube

https://youtu.be/TgTSKIOEJZ0

Pranala Github

https://github.com/DieroA/Makalah_IF2211

Ucapan Terima Kasih

Puji dan syukur saya panjatkan kepada Tuhan Yang Maha Esa yang telah memberikan rahmat dan hidayah-Nya, sehingga makalah dapat diselesaikan dengan baik. Ucapan terimakasih juga saya berikan kepada ibu Dr. Nur Ulfa Mauladevi, S.T, M.Sc. sebagai dosen Strategi Algoritma kelas 2.

References

[1] Munir, Rinaldi (2021). "Penentuan Rute (*Route*/*Path Planning*) Bagian 2: Algoritma A* Bahan Kuliah IF2211 Strategi Algoritma". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf. Diakses pada 1 Juni 2024.

[2] Choi, Alissa (2021). "Tic-Tac-Toe". https://momath.org/wp-content/uploads/2021/08/Alyssa-Choi-Tic-Tac-Toe.pdf. Diakses pada 1 Juni 2024.

[3] *GeeksForGeeks* (2024), "Graph Data Structure And Algorithms". https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/. Diakses pada 1 Juni 2024.

[4] *GeeksForGeeks* (2023), "What is Weighted Graph with Applications, Advantages, and Disadvantages". https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-weighted-graph/. Diakses pada 1 Juni 2024.

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, 12 Juni 2024

Diero Arga Purnama
13522056