APLIKASI GAME TIC TAC TOE 6X6 BERBASIS ANDROID MENGGUNAKAN ALGORITMA MINIMAX DAN HEURISTIC EVALUATION

Ever Jayadi[1)], Muhammad Aziz Fatchur Rachman[2)], Muhammad Yuliansyah[3)]

[1), 2), 3)] Teknik Informatika STMIK AMIKOM Yogyakarta
Jl Ring road Utara, Condongcatur, Sleman, Yogyakarta 55281
Email : evernewtron@gmail.com[1)], muhammad.rachman@students.amikom.ac.id [2)], devil.yansah@gmail.com [3)]

Abstrak

Kemampuan otak manusia dapat dilatih dengan berbagai macam cara yang menyenangkan, salah satunya adalah dengan permainan yang mengasah otak seperti tic-tac-toe Namun belakangan ini tic-tac-toe makin banyak dilupakan oleh berbagai kalangan. Bukan karena permainan ini sulit, namun karena pemain biasanya kesulitan mencari lawan yang bisa diajak untuk bermain tic-tac-toe dimanapun dan kapanpun. Sebab permainan ini membutuhkan setidaknya 2 orang untuk bermain dan media tulis untuk menggambarkan papan permainan. Dikarenakan dua kendala ini, maka kami menemukan sebuah solusi yaitu dengan membuat sebuah aplikasi Android yang memiliki kemampuan berpikir (Kecerdasan Buatan) untuk diajak bermain tic-tac-toe pada perangkat Android.

Dalam membuat aplikasi ini, kami menggunakan algoritma MiniMax yang dikombinasikan dengan Heuristic Evaluation untuk memperoleh kemampuan berpikir yang memadai untuk dijadikan lawan main. Pemilihan algoritma ini karena dengan algoritma ini, perangkat dengan spesifikasi rendah pun dapat menjalakan aplikasi ini sebab perulangan dalam algroitma ini hanya mengeksekusi kemungkinan terbaik saja.

Hasil dari aplikasi ini adalah sebuah perangkat lunak yang interaktif dengan pengguna dan dapat dijadikan lawan bermain tic-tac-toe dimanapun dan kapanpun tanpa perlu risih membawa kertas atau media lain untuk menjadi papan permainan dan juga tidak perlu bingung dalam mencari lawan main.

Kata kunci: *permainan, puzzle, algoritma, kecerdasan buatan, android.*

1. Pendahuluan

Aplikasi permainan yang berada pada sebuah komputer atau *smartphone* merupakan aplikasi yang dikenal oleh banyak orang dari yang muda sampai yang tua. Aplikasi ini cukup mudah dan menyenangkan sehingga sering digunakan untuk menghilangkan kepenatan. Penggunaan aplikasi permainan sering kali dilakukan secara seorang diri saja karena tidak ada lawan yang dapat diajak bermain bersama. Beberapa aplikasi permainan harus dilakukan setidaknya oleh dua orang pemain terutama permainan *two player*, *board games,* misalnya: permainan catur, *tic-tac-toe*, dan *battleship*[1].

Artificial Intelegence (AI) atau kecerdasan buatan biasanya digunakan sebagai teknik untuk menggerakan komputer sebagai lawan bermain dalam aplikasi permainan. Tujuan *AI* digunakan untuk mencari pola permainan, merancang strategi, mengkolaborasikan entity dalam computer, dan belajar dari pengalaman sebelumnya untuk mengambil keputusan, dalam permainan *tic-tac-toe* dapat dilihat bahwa kesuksesan dari permainan merupakan kemenangan dari satu pemain atau pemain lainnya, dimana mencari hasil yang paling masimal dan meminimalisir hasil dari lawan. Maka algoritma *minimax* digunakan pada *boardgames tic-tac-toe*[1].

Pada makalah ini akan membahas salah satu jenis *two player board games,* yaitu permainan *tic-tac-toe*. Apliaksi yang digunakan disini merupakan permainan *tic-tac-toe* yang terdiri dari grid 6x6 dengan 5 kemenangan. Pembuatan permain ini sendiri menggunakan *tools* Android Studio untuk membangun program, beserta algoritma yang di pakai ialah *minimax* dan *heuristic evaluation*.

2. Pembahasan

Pencarian pada AI di kelompokkan kedalam *blind* atau *un-informed search* (pencarian buta) dan *heuristic* atau *informed search* (pencarian terbimbing).

Setiap metode mempunyai katakteristik yang berbeda-beda dengan kelebihan dan kekurangannya masing-masing[2]

Untuk mengukur performansi metode pencarian, terdapat empat kriteria yang dapat digunakan, yaitu :

a. Completeness

Apakah metode tersebut menjamin penemuan solusi jika solusi memang ada?

b. Time Complexity

Berapa lama waktu yang diperlukan?

c. Space Complexity

Berapa banyak memory yang diperlukan?

d. Optimality

Apakah metode tersebut menjamin menemukan solusi yang terbaik jika terdapat beberapa solusi.

2.1 Algoritma Minimax

Penerapan AI dalam permainan *tic-tac-toe* ini menggunakan algoritma minimax dan heuristic evaluation. Minimax merupakan salah satu teknik pencarian menggunakan *depth-first search* dengan kedalaman terbatas.

Minimax adalah sebuah algoritma yang di desain untuk memaksimalkan kemenangan dan meminimalkan kekalahan dalam skenario terburuk di dalam *game*. Idenya adalah untuk memilih langkah berikutnya yang mempunyai nilai minimax tertinggi (mencapai langkah terbaik ketika melawan musuh yang mempunyai langkah terbaik.

Pada permainan *tic-tac-toe* ini mempunyai lebih sedikit kemungkinan solusi, sehingga kita akan mempunyai cukup komputasi untuk memainkan setiap kombinasi langkah dari setiap posisi dan kondisi. Namun hal ini dapat dihindari dengan membatasi sejauh mana komputer akan menganalisis hasil dari langkah-langkah yang mungkin (menentukan kedalaman pohon). Tetapi dengan hal ini, kita harus menambah kedalaman pohon pada *state* tersebut dama dengan *state* sebelumnya.

Algoritma minimax ini bekerja secara rekursif dengan mencari langkah yang akan membuat lawan mengalami kerugian minimum. Semua strategi lawan akan dihitung dengan algoritma yang sama dan seterusnya. Ini berarti pada langkah pertama komputer akan menganalisis seluruh pohon permainan. Dan untuk setiap langkahnya, komputer akan memilih langkah yang paling membuat lawan mendapatkan keuntungan minimum, dan yang paling membuat komputer itu sendiri mendapatkan keuntungan maksimum.

Dalam penentuan keputusan tersebut dibutuhkan suatu nilai yang merepresentasikan kerugian atau keuntungan yang akan diperoleh jika langkah tersebut dipilih. Untuk itulah disini digunakan sebuah fungsi *heuristic evaluation* yang berfungsi untuk mengevaluasi nilai sebagai nilai yang merepresentasikan hasil permainan yang akan terjadi jika langkah tersebut dipilih. Biasanya pada permainan *tic-tac-toe* ini digunakan nilai 1,0,-1 untuk mewakili hasil akhir permainan berupa menang, seri, dan kalah. Dari nilai-nilai *heuristic* inilah komputer akan menentukan simpul mana dari pohon permainan yang akan dipilih, tentunya simpul yang akan dipilih tersebut adalah simpul dengan nilai *heuristic* yang akan menuntun permainan ke hasil akhir yang menguntungkan bagi komputer.

Berikut garis besar algoritma minimax secara umum :

> Cari langkah yang dengan nilai maksimum
> IF langkah tersebut merupakan langkah kemenangan THEN pilih langkah tersebut.
> ELSE
> FOR EACH kemungkinan langkah yang ada
> Cari langkah lawan yang minimum.
> RETURN nilai dari langkah tersebut.
> Pilih langkah yang bernilai maksimum dari langkah-langkah tersebut.[3]

Pemakaian algoritma umum diatas untuk permainan *tic-tac-toe* adalah sebagai berikut :

> IF ada langkah kemenangan THEN pilih langkah tersebut.
> ELSE lawan mempunyai 2 spot terisi dalam satu garis dengan spot ketiga masih kosong THEN tutup langkah tersebut (isi spot kosong ketinga tersebut).
> ELSE melangkah ke *state* yang mempunyai kemungkinan menang tertinggi (berdasarkan nilai *heuristic* yang dibangkitkan.[3]

Dilihat dari algoritma diatas, nilai heuristic yang dibangkitkan akan sangat mempengaruhi analisis dari AI tersebut.

2.2 Aturan Permainan

Berikut beberapa aturan dalam permainan *tic-tac-toe* 6x6 :

a. Permainan memilih simbol "X" atau "O" untuk digunakan dalam permainan.
b. Yang melakukan permainan pertama bebas meletakkan simbol pada papan permainan berukuran 6x6.
c. Komputer (AI) akan jalan sesuai dengan strategi yang dia miliki.
d. Pemain ataupun komputer harus membentuk 1 garis lurus baik vertikal, horizontal maupun diagonal.

Untuk menyelesaikan permainan ini terdapat 3 kondisi yang mungkin terjadi yaitu :

1. Kondisi Menang

   Kondisi menang terjadi apabila pemain berhasil membentuk garis lurus yang terdiri dari 5 simbol yang sama secara vertikal, horizontal, maupun diagonal.

2. Kondisi Kalah

   Kondisi kalah terjadi apabila komputer (AI) mampu membentuk garis lurus yang terdiri dari 5 simbol yang sama secara vertikal, horizontal, maupun diagonal terlebih dahulu.

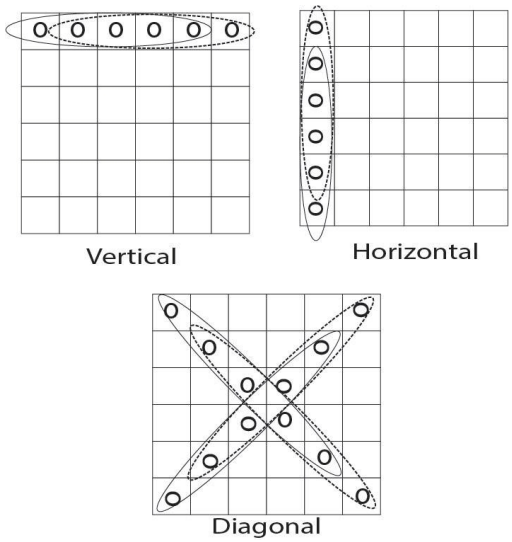
3. Kondisi Seri

   Kondisi seri terjadi apabila pemain maupun komputer (AI) tidak mampu membentuk garis lurus yang terdiri dari 5 simbol yang sama secara vertikal, horizontal, maupun diagonal.

Melakukan cek kemenangan

Pada permainan *tic-tac-toe* ini terdapat 32 kemungkinan untuk memenangkan permainan. Kemungkinan kemenangan tersebut di dapat dari 12 kombinasi simbol

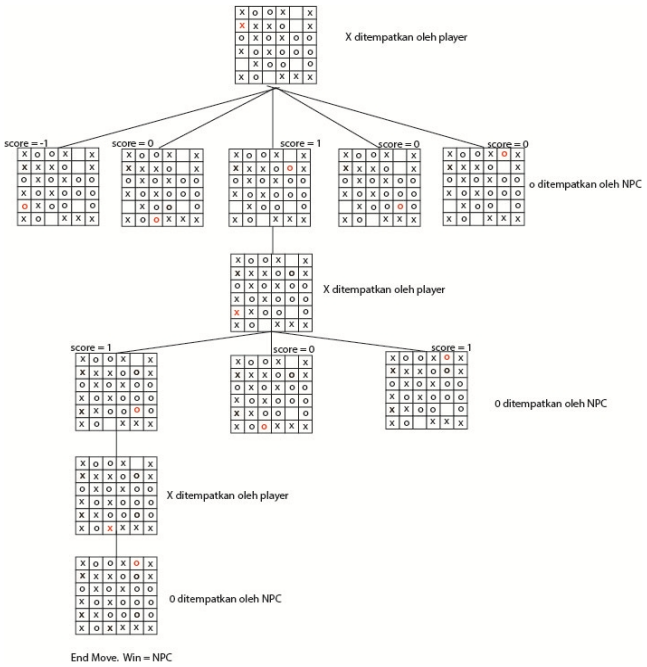
yang sama secara horizontal, 12 kombinasi simbol yang sama secara vertikal dan 8 kombinasi simbol yang sama secara diagonal seperti gambar dibawah ini.



Gambar 1. *Kemungkinan kemenangan*

Gambar pada bagian vertikal dan horizontal diatas mewakili tiap baris dan kolom dimana terdapat 2 kemungkinan kemengangan tiap baris dan kolomnya.

2.3 Pohon Keputusan



Gambar 2. *Pohon keputusan*

Pada simulasi pohon algoritma diatas, kita asumsikan bahwa posisi permainan seperti kondisi puncak pohon yang terlihat. Dimana pada puncak tersebut hanya tersisa empat kotak kosong yang bisa diberi tanda.

Pada langkah paling atas, pemain memberi tanda di kolom (1,0). Program akan langsung menganalisa semua kemungkinan yang dapat terjadi dan mencari kemungkinan pergerakan yang paling menguntungkan berdasarkan poin yang diberikan pada tiap kemungkinan. Dalam kasus diatas, maka langkah yang memiliki poin paling tinggi adalah memberi tanda di kolom (1,4). Hal ini berulang secara terus menerus hingga ditemukanya pemenang atau hingga tidak ada lagi kotak yang dapat diberi tanda.

Program memilih kemungkinan yang paling menguntungkan berdasarkan kecocokan antara syarat kemenangan dengan kondisi kemungkinan yang diuji coba oleh program itu sendiri.

Pada kasus diatas, dari 5 kemungkinan, terdapat satu kemungkinan yang bisa menjadi kesempatan dari pengguna untuk memenangkan permainan dan ini disebut ancaman oleh program. Program akan secara otomatis memprioritaskan ancaman tersebut dan mengambil tindakan yang relevan dengan cara memberi tanda di kolom (4,1) sehingga meniadakan kemungkinan pengguna untuk menang dari kondisi tersebut.

2.4 Source Code Minimax dan Heuristic Evaluation

```
private int evaluate() {
    int score = 0;
    // diagonals)
    score += evaluateLine(0, 0, 0, 1, 0, 2, 0, 3, 0, 4, 0, 5); // row 0
    score += evaluateLine(1, 0, 1, 1, 1, 2, 1, 3, 1, 4, 1, 5); // row 1
    score += evaluateLine(2, 0, 2, 1, 2, 2, 2, 3, 2, 4, 2, 5); // row 2
    score += evaluateLine(3, 0, 3, 1, 3, 2, 3, 3, 3, 4, 3, 5); // row 3
    score += evaluateLine(4, 0, 4, 1, 4, 2, 4, 3, 4, 4, 4, 5); // row 4
    score += evaluateLine(5, 0, 5, 1, 5, 2, 5, 3, 5, 4, 5, 5); // row 5
    score += evaluateLine(0, 0, 1, 0, 2, 0, 3, 0, 4, 0, 5, 0); // col 0
    score += evaluateLine(0, 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1); // col 1
    score += evaluateLine(0, 2, 1, 2, 2, 2, 3, 2, 4, 2, 5, 2); // col 2
    score += evaluateLine(0, 3, 1, 3, 2, 3, 3, 3, 4, 3, 5, 3); // col 3
    score += evaluateLine(0, 4, 1, 4, 2, 4, 3, 4, 4, 4, 5, 4); // col 4
    score += evaluateLine(0, 5, 1, 5, 2, 5, 3, 5, 4, 5, 5, 5); // col 5
    score += evaluateLine(0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5);
    // diagonal
    score += evaluateLine(5, 0, 4, 1, 3, 2, 2, 3, 1, 4, 0, 5);
    // alternate diagonal
    return score;
}
```

Gambar 3.*Evaluasi heuristic pada setiap baris, kolom, dan diagonal*

Evaluasi heuristik disini berfungsi sebagai penyekoran terhadap setiap baris, kolom, dan diagonal. Pada tahap ini *NPC (Non Player Character)* akan menghitung setiap baris, kolom, dan diagonal untuk menentukan jalan selanjunya. (0,0 , 0,1 , 0,2 , 0,3 , 0,4 , 0,5) disini menunjukkan *NPC* akan mengecek pada baris 0 kolom 0, selanjutnya baris 0 kolom 1, dan seterusnya.

```
private int[] minmax(int depth, String player, int alpha, int beta) {
    List<int[]> nextMoves = generateMoves();

    int score;
    int bestRow = -1;
    int bestCol = -1;

    if (nextMoves.isEmpty() || depth == 0) {
        score = evaluate();
        return new int[] { score, bestRow, bestCol };
    } else {
        for (int[] move : nextMoves) {
            board[move[0]][move[1]] = player;
            if (player.equals(myMark)) {
                score = minmax(depth - 1, oppMark, alpha, beta)[0];
                if (score > alpha) {
                    alpha = score;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            } else {
                score = minmax(depth - 1, myMark, alpha, beta)[0];
                if (score < beta) {
                    beta = score;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            }
            board[move[0]][move[1]] = "";
            if (alpha >= beta)
                break;
        }
    }
    return new int[] { (player == myMark) ? alpha : beta, bestRow, bestCol };
```

Gambar 4.*program minmax yang diimplementasikan pada NPC*

Pada program *minimax* akan menghitung score nilai heuristik pada kedua pemain, *player* dan *NPC (Non Player Character)*. Jika *score* itu lebih besar dari *alpha* (*alpha disini untuk nilai maximal)* maka *player atau NPC* tersebut mendapatkan dan mencari nilai maksimalkan, sedangkan jika *score* kurang dari *beta* (beta disini untuk nilai minimal) maka *player atau NPC* tersebut akan mencari nilai minimal.

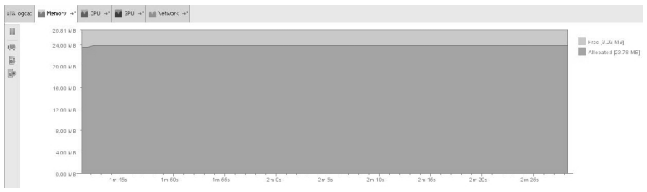
```
public boolean isWinner() {
    //Diagonal ( \ )
    //00-44
    if (theBoard[0][0].equals(theBoard[1][1])

            && theBoard[0][0].equals(theBoard[2][2])
            && theBoard[0][0].equals(theBoard[3][3])
            && theBoard[0][0].equals(theBoard[4][4])

            && theBoard[0][0] != "")
        return true;

    //55-11
    if (theBoard[5][5].equals(theBoard[4][4])
            && theBoard[5][5].equals(theBoard[3][3])
            && theBoard[5][5].equals(theBoard[2][2])
            && theBoard[5][5].equals(theBoard[1][1])
            && theBoard[5][5] != "")
        return true;

    //10-54
    if (theBoard[1][0].equals(theBoard[2][1])
            && theBoard[1][0].equals(theBoard[3][2])
            && theBoard[1][0].equals(theBoard[4][3])
            && theBoard[1][0].equals(theBoard[5][4])
            && theBoard[1][0] != "")
        return true;
```

Gambar 5.*knowledge based*

Pada potongan program diatas ialah *knowledge based* yang dipasang pada papan permainan. Dimana akan ada pengecekan kemenangan pada diagonal di baris nol kolom 0 ke baris empat ke kolom 4. Jika salah satu pemain mengisi semua pada baris tersebut maka permainan akan selesai.

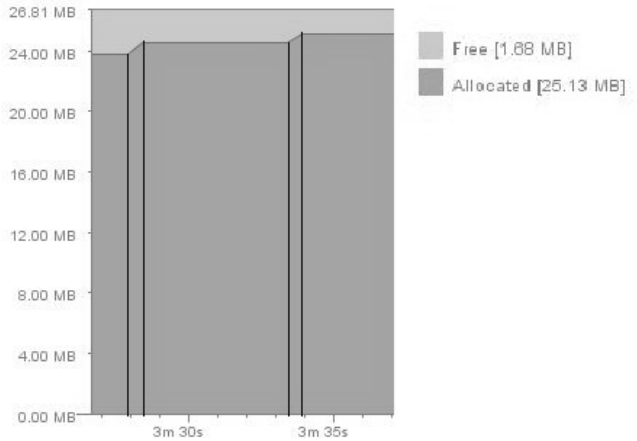
2.5 Performansi Metode Pencarian

Berikut adalah pengujian berapa lama waktu dan memori yang dibutuhkan oleh *AI* untuk menyelesaikan 1 iterasi. Percobaan ini dilakukan *one by one*, maksudnya yaitu ketika player sudah mengisi satu kotak maka akan dilanjutkan oleh si *NPC* atau *AI*. Ketika *player* sudah mengisi kotak dan dilanjutkan *NPC* disana terdapat *delay*, *delay* tersebut ialah waktu yang dibutuhkan *NPC/AI* untuk menyelesaikan iterasinya.



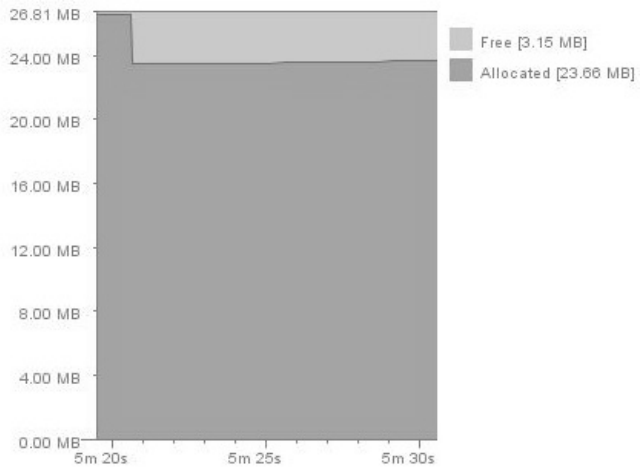
Gambar 6.*penggunaan memory ketika tidak adanya aktifitas dalam aplikasi*

Keadaan aplikasi ketika tidak adanya aktifitas saat itu, *memory* yang digunakan ialah *23.78 MB*.



Gambar 7.*keadaan ketika player telah mengisi 3 kotak kosong*

Pada grafik diatas *player* telah mengisi 3 kotak yang kosong, waktu yang dibutuhkan oleh *NPC* untuk mencari jawaban atau melakukan 1 iterasi ialah *0.5 second*, dan *memory* yang digunakan ialah sekitar *0.27 MB*.



Gambar 8.*grafik ketika permainan telah selesai*

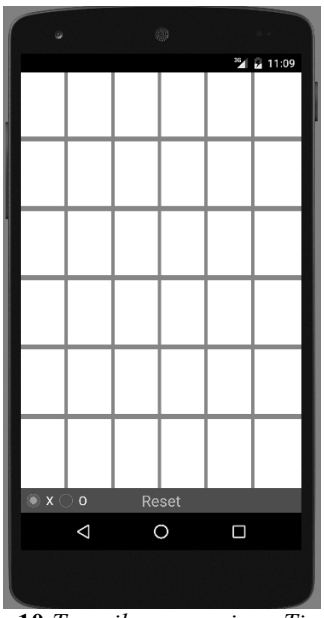
Pada saat permainan telah berakhir dan sebelum direset, memory yang digunakkan ialah *23.66 MB*.

Metode pencarian dengan algoritma minimax ini dilakukan pada semua simpul dalam setiap level secara mendalam mulai dari simpul yang paling kiri. Jika sampai pada level terdalam tidak ditemukan solusi, maka pencarian dilanjutkan pada simpul sebelah kanan, demikian seterusnya sampai ditemukan solusi. Dengan cara seperti ini, metode pencarian algoritma minimax menjamin ditemukannya solusi (jika solusinya memang ada). Dan dengan mengkombinasikan dengan nilai-nilai *heuristic evaluiation*, solusi yang ditemukan pasti yang paling terbaik, dengan kata lain metode ini adalah *complete* dan *optimal*.

2.6 Hasil

Gambar 9.*Tampilan menu permainan Tic Tac Toe*

Tampilan menu awal ketika aplikasi dijalankan. Pada permainan ini player dapat memilih jenis permainannya, bermain dengan player lainnya atau bermain bersama melawan *AI/NPC*



Gambar 10.*Tampilan permainan Tic Tac Toe*

Tampilan permainan *tic-tac-toe* menggunakan *emulator android*, disana terdapat dimana player dapat memilih menggunakan tanda "X' atau "O", serta pemain dapat mereset permainan jika permainan telah berakhir pada tombol Reset.

3. Kesimpulan

Tic-tac-toe merupakan permainan yang melatih kemampuan otak sehingga cocok untuk semua kalangan usia. *Game* ini mengutamakan kemampuan logika untuk memprediksi langkah lawan dari segala kemungkinan yang dapat terjadi.

Game ini kami buat dengan menggunakan algoritma *minimax* yang digabungkan dengan metode *heuristic evaluation*. Kombinasi ini memberikan game sebuah kemampuan untuk berpikir (kecerdasan buatan). Kami memilih penerapan algoritma *minimax* dan metode *heuristic evaluation* ini karena kombinasi ini dapat memberikan hasil yang akurat namun dengan konsumsi sumber daya yang sedikit. Dengan kata lain *game* kami ini dapat berjalan di perangkat dengan spesifikasi teknis rendah sekalipun.

Kemampuan berpikir dari program kecerdasan buatan di dalam *game* ini pun dapat diatur tingkat kecerdasannya sesuai dengan keinginan pengguna sehingga dapat menyesuaikan dengan kemampuan berpikir pengguna.

Daftar Pustaka

[1] http://zikky.lecturer.pens.ac.id/Project203/Paper20Minmax.pdf.
[2] Suyanto, *Artificial Intelligent*, Bandung:Informatika, 2007.
[3] Kevin McGee, "Advance Game Programming : AI", Desember 9, 2005.

ISSN : 2302-3805

Biodata Penulis

Ever Jayadi, Mahasiswa S1 Jurusan Teknik Informatika STMIK AMIKOM Yogyakarta,

Muhammad Aziz Fatchur Rachman, Mahasiswa S1 Jurusan Teknik Informatika STMIK AMIKOM Yogyakarta,

Muhammad Yuliansyah, Mahasiswa S1 Jurusan Teknik Informatika STMIK AMIKOM Yogyakarta,