Pembuatan Kecerdasan Buatan untuk Permainan Catur Jawa Dengan Menggunakan Algoritma MiniMax

Brian Al Bahr

Program Studi Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10,Bandung
e-mail: if16093@students.if.itb.ac.id

ABSTRAK

Banyak sekali algoritma yang digunakan untuk mengimplementasikan kecerdasan buatan. Salah satu dari algoritma itu adalah minimax. Algoritma minimax merupakan salah satu implementasi dari *depth first search*. Algoritma minimax terutama diaplikasikan pada permainan yang melibatkan dua orang, dan lebih khusus lagi, permainan dua orang yang saling berganti giliran bermain seperti *checkers, tic-tac-toe*, catur, catur jawa dan lain sebagainya. Permainan-permainan tersebut dapat dideskripsikan dengan sejumlah aturan dan premis. Dengan itu, kita dapat mengetahui, pada titik tertentu permainan, langkah-langkah yang mungkin berikutnya. Permainan tersebut berbagi karakteristik yang sama, yakni "permainan dengan penuh informasi". Dengan cara inilah penulis akan berusaha membuat sebuah kecerdasan buatan untuk permainan catur jawa, yang lagi, memiliki karakteristik yang sama dengan permainan yang telah disebutkan di atas. Pada suatu titik di makalah ini juga terdapat cara untuk mengoptimasi algoritma minimax agar memiliki performansi yang lebih baik, salah satunya dengan menggunakan alpha-beta.

Kata kunci: *minimax, alpha-beta cutoffs, catur jawa, depth first search, kecerdasan buatan*.

1. PENDAHULUAN

Istilah kecerdasan buatan atau AI adalah sesuatu tiruan atau buatan yang cerdas. Cerdas di sini kemungkinan maksudnya adalah kepandaian atau ketajaman dalam berpikir, seperti halnya otak manusia dalam menyelesaikan suatu masalah. Kecerdasan buatan dapat diterapkan atau dimplementasikan ke dalam berbagai bentuk aplikasi.

Walau pun menyadari bahwa kecerdasan buatan bisa jadi adalah suatu ancaman untuk manusia, tapi manusia masih saja mengembangkan apa yang disebut dengan kecerdasan buatan. Manusia masih saja mencoba mengembangkan / mendapatkan sesuatu (teknologi) yang baru, yang dapat berpikir seperti manusia. Hal ini terjadi karena adanya ketidakpuasan dalam diri manusia, manusia ingin mendapatkan sesuatu dengan cara yang lebih mudah. Lagipula memang ada keterbatasan-keterbatasan dalam diri manusia, seperti otak manusia yang hanya mampu berpikir dengan frekuensi kira-kira 100 Hz dan karena manusia mempunyai rasa capai. Bandingkan dengan komputer sekarang yang mampu mengolah data dengan frekuensi 4 GHz. Komputer juga tidak mempunyai rasa capai walau pun harus mengolah data yang sama berulang-ulang.

Bentuk implementasi yang paling mudah untuk diukur tingkat keberhasilan (kecerdasan buatan) dan cukup digemari oleh sebagian besar publik yaitu pada *games* atau permainan. Salah satu algoritma yang dapat diimplementsaikan sebagai kecerdasan buatan dalam sebuah permainan adalah algoritma minimax. Lebih khusus lagi, algoritma minimax sangat baik diterapkan pada permainan yang melibatkan 2 orang dan bermain bergantian, misalnya pada permainan catur, *othello*, *backgammon,* catur jawa, dan lain sebagainya.

Pada kesempatan kali ini, penulis akan mengimplementasikan algoritma minimax pada permainan catur jawa. Catur jawa adalah permainan yang sangat menyenangkan, permainan ini sudah dapat dimainkan bermodalkan pensil dan kertas. Satu pemain mendapat simbol 'X' dan yang lain mendapat simbol 'O'. Pemain secara bergantian bebas untuk menempatkan simbol-simbol tersebut pada tempat yang telah disediakan. Tujuan dari permainan ini adalah mendapatkan 4 buah simbol yang sama pada satu baris, satu kolom, atau pada diagonal yang sama.

Proses utama algoritma minimax yaitu pencarian nilai terbaik berdasarkan nilai-nilai yang telah diberikan pada tiap-tiap langkah. Nilai-nilai tersebut dibangkitkan berdasarkan basis pengetahuan yang dimiliki oleh algoritma tersebut. Dengan penerapan algoritma minimax sebagai pondasi suatu kecerdasan buatan pada permainan catur jawa, maka diharapkan akan dihasilkan suatu permainan yang interaktif. Alat bantu yang akan digunakan penulis dalam mengimplementasikan algoritma

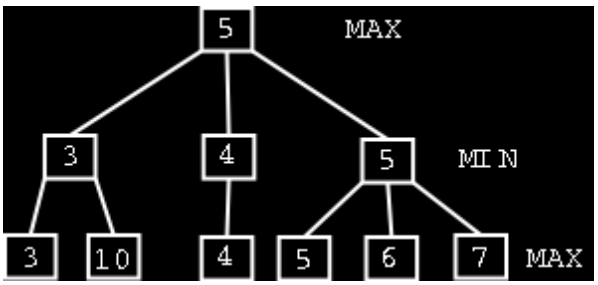
minimax pada permainan catur jawa ini adalah bahasa pemrograman *Java.*

2. METODE

2.1 Algoritma Minimax

Algoritma minimax diaplikasikan pada permainan yang melibatkan dua orang. Permainan-permainan tersebut dapat dideskripsikan dengan sejumlah aturan dan premis. Dengan itu, kita dapat mengetahui, pada titik tertentu permainan, langkah-langkah yang mungkin berikutnya. Permainan tersebut berbagi karakteristik yang sama, yakni "permainan dengan penuh informasi". Setiap pemain mengetahui semua langkah-langkah yang mungkin dari pemain lawannya.

Sebelum menjelaskan algoritma minimax, pengenalan mengenai pohon pencarian dibutuhkan. Pohon pencarian adalah cara untuk merepresentasikan pencarian. Kotak disebut sebagai simpul dan simpul-simpul tersebut merepresentasikan titik keputusan pada pencarian. Simpul dihubungkan dengan cabang. Pencarian dimulai pada simpul akar, ditunjukkan pada bagian atas pada gambar 1. Pada setiap simpul keputusan, simpul berikutnya yang mungkin untuk pencarian dibangkitkan, sampai tidak ada lagi keputusan yang mungkin. Simpul yang merepresentasikan akhir pencarian disebut sebagai simpul daun.



Gambar 1. Representasi pohon pencarian untuk permainan logika

Pada algoritma ini ada dua pemain yang terlibat, kita asumsikan MAX dan MIN. Pohon pencarian dibangkitkan, *depth-first,* dimulai dari posisi permainan saat ini sampai pada akhir posisi permainan. Lalu, kondisi permainan final dievaluasi sebagai sudut pandang MAX, seperti tergambar pada gambar 1. Setelah itu, simpul-simpul di atas simpul daun diisi secara *bottom up* dengan nilai pada simpul anak-anaknya. Simpul yang dimiliki pemain MAX menerima nilai maksimum dari simpul anak-anaknya dan pemain MIN memperoleh nilai minimum dari nilai-nilai yang dimiliki simpul anak-anaknya. Berikut gambaran algoritmanya:

```
MinMax (PosisiPermainan game) {
  return GerakMax (game);
}
```

```
GerakMax (PosisiPermainan game) {
  if (PermainanSelesai(game)) {
    return EvalGameState(game);
  }
  else {
    best_move < - {};
    moves <- BangkitkanGerakan(game);
    ForEach moves {
       move <- GerakMin(LakukanGerakan(game));
       if (Nilai(move) > Nilai(best_move)) {
          best_move < - move;
       }
    }
    return best_move;
  }
}

GerakMin (PosisiPermainan game) {
  best_move <- {};
  moves <- BangkitkanGerakan(game);
  ForEach moves {
     move <- GerakMax(LakukanGerakan(game));
     if (Nilai(move) > Nilai(best_move)) {
        best_move < - move;
     }
  }
  return best_move;
}
```

'Nilai' pada algoritma di atas merepresentasikan seberapa baik permainan bergerak. Pemain MAX akan mencoba memilih gerakan dengan nilai maksimum pada akhirnya sedangkan pemain MIN akan memilih gerakan yang lebih baik baginya, karena itu MIN akan meminimkan keluaran dari MAX (*minimizing MAX's outcome*).

2.2 Optimasi Algoritma

Pembangkitan seluruh pohon pencarian hanya dimungkinkan jika dan hanya jika permainan yang diimplementasikan sangat sederhana. Pada kebanyakan permainan, hal ini tidak mungkin, oleh karena itu, ada sedikit optimasi yang harus ditambahkan pada algoritma tersebut.

Kendati demikian, optimasi ini datang dengan sebuah efek samping. Dengan mengoptimasi, kita menukar informasi seluruhnya tentang permainan dengan kemungkinan dan jalan pintas. Kita tidak lagi memilih jalan yang menghasilkan kemenangan tetapi memilih jalan yang 'menuju' ke arah kemenangan. Jika pengoptimasian tidak dipilih dengan bijak, atau salah pengaplikasian, bisa-bisa kecerdasan yang dibuat menjadi bodoh dan malahan lebih baik memilih langkah sebarang daripada menggunakan kecerdasan buatan.

Optimasi dasar adalah dengan cara membatasi kedalaman pohon pencarian. Mengapa ini bisa menolong? Membangkitkan seluruh pohon pencarian bisa memakan waktu yang sangat lama. Jika permainan memiliki faktor pencabangan 3 (masing-masing simpul memiliki 3 buah simpul anak), maka pohon pada tingkat n akan memiliki

simpul sebanyak 3^n buah. Pohon pencarian ini juga akan menghasilkan simpul sebanyak jumlah setiap jumlah simpul pada setiap tingkat. Memori tidak akan cukup untuk menampung pohon pencarian pada kebanyakan permainan, seperti catur yang memiliki banyak faktor pencabangan. Walaupun memori mencukupi, tapi proses pembangkitan pohon pencarian akan membutuhkan waktu yang sangat lama. Jika setiap simpul membutuhkan 1 detik untuk dianalisis, untuk pohon dengan faktor pencabangan 3 dan tingkat=5 akan menghasilkan waktu pembangkitan sebanyak 1+3+9+27+81+243 = 364 * 1 = 364 detik atau sekitar 6 menit! Waktu 6 menit terlalu lama untuk sebuah permainan. Pemain akan meninggalkan permainan jika harus menunggu masing-masing 6 menit untuk setiap gerakan dari komputer.

Optimasi kedua adalah dengan menggunakan fungsi yang mengevaluasi posisi permainan saat ini dari sudut pandang suatu pemain. Cara ini dilakukan dengan cara memberikan nilai tertentu pada *state* tertentu pada permainan, seperti menghitung jumlah biji catur yang ada di papan, atau hal lain seperti memberi nilai tertentu pada posisi permainan.

Selain mengevaluasi posisi tertentu pada permainan, fungsi ini juga dapat mengkalkulasi bagaimana suatu kondisi dalam permainan dapat membantu mengakhiri permainan. Dengan kata lain, berapa peluang bagi kita untuk memenangkan permainan pada posisi tertentu dalam permainan. Dalam kasus ini, fungsi ini disebut sebagai fungsi estimasi (prekiraan).

Fungsi ini akan menggunakan metode heuristik. Heuristik adalah pengetahuan yang kita punya dari permainan yang akan membantu membangkitkan fungsi evaluasi yang lebih baik. Sebagai contoh, dalam *checkers,* biji pada posisi ujung papan tidak bisa dimakan, dan karena itu nilainya akan lebih tinggi bila biji itu berada pada posisi unjung papan. Salah satu alasan fungsi evaluasi harus bisa mengevaluasi posisi permainan untuk kedua pemain adalah karena kita tidak tahu pemain mana yang memiliki batas kedalaman.

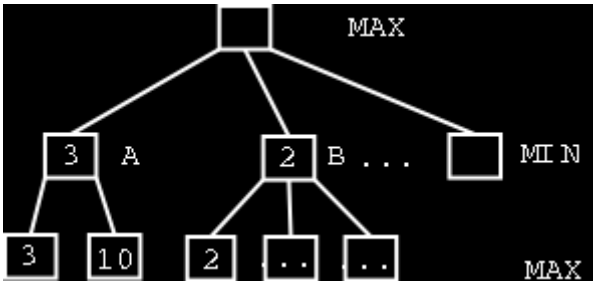
Kendati demikian, pembuatan dua fungsi (masing-masing satu untuk pemain) dapat dihindari jika permainan simetrik. Ini berarti bahwa 'kehilangan' dari suatu pemain sebanding dengan 'perolehan' dari pemain yang satunya. Contoh permainan ini adalah permainan *ZERO-SUM*. Pada permainan-permainan seperti ini, satu fungsi sudah mencukupi karena pemain yang lain hanya perlu menegasikan kembalian dari fungsi yang pertama. Berikut algoritma minimax setelah dioptimasi:

```
MinMax (PosisiPermainan game) {
  return GerakMax (game);
}

GerakMax (PosisiPermainan game) {
  if
(PermainanSelesai(game)||(batasKedalamanTercapai(
))) {
    return EvalGameState(game,MAX);
  }
  else {
    best_move < - {};
    moves <- BangkitkanGerakan(game);
    ForEach moves {
       move <- GerakMin(LakukanGerakan(game));
       if (Nilai(move) > Nilai(best_move)) {
          best_move < - move;
       }
    }
    return best_move;
  }
}

GerakMin (PosisiPermainan game) {
if
(PermainanSelesai(game)||(batasKedalamanTercapai(
))) {
     return EvalGameState(game,MIN);
    }
else{
best_move <- {};
  moves <- BangkitkanGerakan(game);
  ForEach moves {
     move <- GerakMax(LakukanGerakan(game));
     if (Nilai(move) > Nilai(best_move)) {
        best_move < - move;
     }
  }
  return best_move;
 }
}
```

Celah terbesar yang mungkin terjadi setelah mengaplikasikan optimasi algoritma minimax adalah masalah kedalaman yang terbatas. Posisi permainan yang kelihatan amat baik mungkin saja berubah menjadi sangat buruk. Hal ini dapat terjadi karena algoritma tidak mampu untuk melihat bahwa gerakan yang diambil setelah beberapa langkah ke depan dapat menghasilkan keuntungan yang amat besar bagi pemain lawan. Algoritma tidak melihat gerakan yang berakibat fatal tersebut karena algoritma ini dibutakan oleh batas kedalaman.

2.3 Mempercepat Algoritma

Masih ada beberapa cara yang masih dapat dilakukan untuk mengurangi waktu pencarian. Kita lihat gambar 2. Nilai untuk simpul A adalah 3, dan nilai yang ditemukan pertama kali untuk upapohon yang dimulai dari simpul B adalah 2. Maka, karena simpul B ada pada tingkat MIN, kita tahu bahwa dengan nilai yang dipilih untuk B harus lebih kecil atau sama dengan 2. Tapi kita juga tahu bahwa simpul A memiliki nilai 3, dan simpul A dan B keduanya memiliki simpul *parent* yang sama pada tingkat MAX. Hal ini dapat diartikan bahwa jalur yang dimulai dari simpul B tudak akan dipilih karena 3 lebih baik daripada 2 pada simpul MAX. Oleh karena itu, pencarian anak-anak simpul B tidak perlu dilakukan dan kita dapat dengan aman mengabaikan sisa anak-anak simpul B.



Gambar 2. Pencarian minimax yang menunjukkan bahwa cabang dapat di-potong

Optimasi ini dikenal dengan nama pemotongan Alpha-Beta (alpha beta cutoffs) dan berikut adalah cara mengaplikasikannya:

1. Dapatkan nilai alpha dan beta. Alpha adalah nilai yang berisi nilai maksimum yang ditemukan. Beta adalah nilai yang berisi nilai minimum yang ditemukan.
2. Pada tingkat MAX, sebelum mengevaluasi jalur anak, bandingkan dulu nilai yang dikembalikan dengan jalur sebelumnya dengan nilai beta. Jika nilainya lebih besar, batalkan pencarian untuk simpul ini.
3. Pada tingkat MIN, sebelum mengevaluasi jalur anak, bandingkan dulu nilai yang dikembalikan dengan jalur sebelumnya dengan nilai alpha. Jika nilainya lebih kecil, batalkan pencarian untuk simpul ini.

Berikut adalah *pseudocode* algoritma minimax dengan *alpha-beta cutoffs.*

```
MinMax (PosisiPermainan game) {
  return GerakMax (game);
}

GerakMax (PosisiPermainan game,int alp,int bet) {
  if
(PermainanSelesai(game)||(batasKedalamanTercapai(
))) {
    return EvalGameState(game,MAX);
  }
  else {
    best_move < - {};
    moves <- BangkitkanGerakan(game);
    ForEach moves {
        move <-
          GerakMin(LakukanGerakan(game),alp,bet);
        if (Nilai(move) > Nilai(best_move)) {
           best_move < - move;
           alp <- Nilai(move);
        }
        if (bet > alp)
          return best_move;
    }
    return best_move;
  }
}

GerakMin (PosisiPermainan game,int alp,int bet) {
```

```
if
(PermainanSelesai(game)||(batasKedalamanTercapai(
))) {
      return EvalGameState(game,MIN);
    }
else{
best_move <- {};
  moves <- BangkitkanGerakan(game);
  ForEach moves {
      move <-
       GerakMax(LakukanGerakan(game),alp,bet);
      if (Nilai(move) > Nilai(best_move)) {
          best_move < - move;
          bet <- Nilai(move)
      }
      if (bet < alp)
       return best_move;
  }
  return best_move;
 }
}
```

Seberapa baik minimax dengan *alpha-beta cutoffs* jika dibandingkan dengan minimax normal benar-benar bergantung pada urutan pencarian dilakukan. Jika cara pembangkitan posisi permainan tidak menciptakan situasi dimana algoritma bisa mengambil keuntungan dari *alpha-beta cutoffs*, maka peningkatan algoritma tidak akan kelihatan. Kendati demikian, jika evaluasi fungsi dan pembangkitan posisi permainan mengarah pada *alpha-beta cutoffs*, maka peningkatan algoritma akan sangat baik.

3. Implementasi Algoritma

Pada pengimplementasian algoritma minimax pada permainan catur jawa ini, penulis menggunakan bahasa pemrograman Java. Pada pengimplementasiannya, catur jawa yang dibuat penulis diberi batasan-batasan khusus. Batasan-batasan ini antara lain : lebar papan hanya dibatasi 7 kotak saja dan tinggi papan hanya 6 kotak saja. Permainan juga selalu dimulai dari bawah untuk menyederhanakan persoalan.

Pemain akan memulai permainan dengan cara memilih salah satu kolom dari tujuh buah kolom yang tersedia. Pemain akan mendapatkan simbol 'X'. Komputer pun akan segera melakukan gerakan untuk melawan pemain sesuai dengan algoritma minimax yang diimplementasikan. Demikian selanjutnya sampai salah satu pemain (kita atau komputer) memperoleh 4 simbol yang sama ('X' untuk kita dan 'O' untuk komputer) pada baris yang sama, pada kolom yang sama, atau pada diagonal yang sama.

Method-method yang dipakai dalam mengimpelementasikan algoritma ini pada intinya hanya tiga buah, yakni konstruktor caturJawa, fungsi miniMax sebagai algoritma inti pada kecerdasan buatan pada permainan catur jawa ini, dan *method* fourInARow yang berfungsi untuk menghitung apakah suatu pemain telah

memperoleh simbol empat buah yang berurutan. Untuk memperjelas, mari kita lihat algoritmanya :

Berikut adalah variabel global yang digunakan:

```
private static final int lebar=7, tinggi=6;
private static final int maxReccur=6;
private int[][] papan=new int[lebar][tinggi];
private int[] tinggiKolom=new int[lebar];
private static int recurDepth=0;
private static BufferedReader in;
```

Berikut adalah konstruktor caturJawa :

```
public caturJawa() {
   int kolom;
   int player=1;
   Vector<Integer> gerakan=new Vector<Integer>();
   while(true) {
      if(player==1) {
        printPapan();
        do {
          System.out.print("Masukkan  gerakan  (1-
            7): ");
          kolom=readInt()-1;
        }while(kolom<0||kolom>=lebar||
           tinggiKolom[kolom]>=tinggi);
      } else {
        gerakan.removeAllElements();
        kolom=0;
        int prevValue=-maxReccur-1;
        for(int x=0; x<lebar; x++) {
        if(tinggiKolom[x]>=tinggi) continue;
        int value=miniMax(2, x);
        if(value>prevValue) {
          gerakan.removeAllElements();
          prevValue=value;
        }
        if(value==prevValue)
          gerakan.add(new Integer(x));
        }
        if(gerakan.size()>0) {
          Collections.shuffle(gerakan);
          kolom=(gerakan.get(0)).intValue();
        }
        if(gerakan.size()==0) {
          System.out.println("Permainan seri");
          break;
        }
      }

      papan[kolom][tinggiKolom[kolom]]=player;
      tinggiKolom[kolom]++;
      int menang=0;
      menang=fourInARow();
      if(menang>0) {
        printPapan();
        System.out.println("Player"+player+"
          menang");
        break;
      }
      player=3-player;
   }
}
```

Algoritma miniMax-nya adalah sebagai berikut:

```
int miniMax(int player, int kolom) {
   int value=0;
   if(tinggiKolom[kolom]>=tinggi) return 0;
   recurDepth++;
   papan[kolom][tinggiKolom[kolom]]=player;
   tinggiKolom[kolom]++;
   if(fourInARow()>0) {
      if(player==2)
        value=maxReccur+1-recurDepth;
      else value=-maxReccur-1+recurDepth;
   }
   if(recurDepth<maxReccur && value==0) {
      value=maxReccur+1;
      for(int x=0; x<lebar; x++) {
        if(tinggiKolom[x]>=tinggi) continue;
        int v=miniMax(3-player, x);
        if(value==(maxReccur+1)) value=v;
        else if(player==2) {if(value>v)value=v;}
        else if(v>value) value=v;
      }
   }
   recurDepth--;
   tinggiKolom[kolom]--;
   papan[kolom][tinggiKolom[kolom]]=0;
   return value;
}
```

Dan yang terakhir, fungsi FourInARow :

```
int fourInARow() {
   int num, player;
   for(int y=0; y<tinggi; y++) {
      num=0; player=0;
      for(int x=0; x<lebar; x++) {
        if(papan[x][y]==player) num++;
        else { num=1; player=papan[x][y]; }
        if(num==4 && player>0) return player;
      }
   }
   for(int x=0; x<lebar; x++) {
      num=0; player=0;
      for(int y=0; y<tinggi; y++) {
        if(papan[x][y]==player) num++;
        else { num=1; player=papan[x][y]; }
        if(num==4 && player>0) return player;
      }
   }
   for(int xStart=0, yStart=2; xStart<4; ) {
      num=0; player=0;
      for(int  x=xStart,  y=yStart;  x<lebar  &&
         y<tinggi; x++, y++) {
        if(papan[x][y]==player) num++;
        else { num=1; player=papan[x][y]; }
        if(num==4 && player>0) return player;
      }
      if(yStart==0) xStart++;
      else yStart--;
   }
   for(int xStart=0, yStart=3; xStart<4; ) {
      num=0; player=0;
        for(int  x=xStart,  y=yStart;  x<lebar  &&
          y>=0; x++, y--) {
          if(papan[x][y]==player) num++;
          else { num=1; player=papan[x][y]; }
          if(num==4 && player>0) return player;
        }
        if(yStart==tinggi-1) xStart++;
        else yStart++;
   }
   return 0;
 }
```

Algoritma lengkap caturJawa.java ini dapat diunduh di http://students.if.itb.ac.id/~if16093/caturJawa.java

Pada konstruktor caturJawa, yang dilakukan adalah menunggu masukan user kemudian mengantisipasi masukan user itu dengan memanggil *method* miniMax dan kemudian membandingkan hasil keluaran *method* miniMax tersebut. Karena *method* ini digunakan untuk memaksimalkan langkah komputer, maka yang dicari adalah nilai yang maksimum dari semua keluaran miniMax ini.

Fungsi heuristik pada prosedur miniMax adalah dengan mencari langkah paling cepat menghasilkan akhir permainan. Akhir permainan ini ditentukan dengan menggunakan method FourInARow (jika sudah ada 4 simbol berurutan maka permainan berakhir). Jika sedang berada pada langkah komputer (MAX), algoritma akan mencari nilai minimum yang dihasilkan oleh pemain dan jika berada pada langkah pemain algoritma akan mencari nilai maksimum yang dihasilkan oleh komputer.Fungsi miniMax ini merupakan fungsi rekursif dengan maksimum kedalaman 6. Kedalaman ini dimaksudkan untuk mengoptimasi algoritma miniMax yang dibuat.

Gambar 3. Tampilan program catur jawa dengan menggunakan algoritma minimax

4. Analisis Kompleksitas

Kompleksitas waktu komputer untuk menentukan gerakan selanjutnya sangat ditentukan oleh algoritma minimax yang digunakan. Karena penulis tidak mengimplementasikan *alpha-beta cutoffs* pada algoritma minimax di atas, maka kasus yang terjadi selalu merupakan kasus terburuk di mana setiap simpul memiliki 7 buah kemungkinan gerakan dengan kedalaman 6. Selain itu, algoritma ini juga menggunakan method fourInARow yang memiliki kompleksitas O(mp) dengan m adalah lebar papan dan p adalah tinggi papan. Oleh sebab itulah algoritma miniMax ini memiliki kompleksitas $O(7^n) * O(mp) = O(mp7^n)$ dengan n adalah kedalaman maksimum dalam pohon pencarian.

Dengan menggunakan *alpha-beta cutoffs,* kita dapat mengurangi kebutuhan waktu jika dan hanya jika urutan pembangkitannya mengarah ke arah berakhirnya permainan. Jadi, *alpha-beta cutoffs* ini hanya berguna pada beberapa kasus saja, tidak selalu berguna pada setiap kasus yang umum. Dengan pengimplementasian *alpha-beta cutoffs*, kompleksitas tetap saja $O(mp7^n)$.

5. KESIMPULAN

Algoritma minimax mungkin bukan jawaban terbaik untuk semua jenis permainan yang membutuhkan kecerdasan buatan yang menyerupai manusia. Kendati demikian, algoritma ini sangat cocok untuk diimplementasikan pada permainan-permainan yang melibatkan dua orang, di mana setiap pemain mengetahui semua langkah-langkah yang mungkin dari pemain lawannya seperti catur, catur jawa, *checkers*, dan lain sebagainya.

Perhitungan algoritma yang membutuhkan waktu pencarian yang lama ini dapat dioptimasi dengan beberapa cara, misalnya dengan menggunakan *alpha-beta cutoffs* atau membatasi kedalaman pohon pencarian. Kompleksitas algoritma minimax bergantung pada kedalaman maksimum pohon pencarian. Kecerdasan buatan untuk permainan catur jawa dapat dibuat dengan menggunakan algoritma minimax.

REFERENSI

[1] http://ai-depot.com. Diakses pada tanggal 9 Mei 2008 pukul 10.00 WIB.

[2] http://www.aihorizon.com. Diakses pada tanggal 9 Mei 2008 pukul 10.00 WIB.

[3] http://www.codeproject.com. Diakses pada tanggal 9 Mei 2008 pukul 10.00 WIB.

[4] http://informatika.org/~rinaldi. Diakses pada tanggal 9 Mei 2008 pukul 13.00 WIB.

[5] http://lyree.wordpress.com. Diakses pada tanggal 9 Mei 2008 pukul 13.00WIB.

[5] Munir, Rinaldi, "Strategi Algoritmik", edisi 2007, Program Studi Teknik Informatika STEI ITB, 2007