ALGORITMA MINIMAX DALAM PERMAINAN *CHECKERS*

Nadhira Ayuningtyas (13506048)

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha No. 10 Bandung
e-mail: if16048@students.if.itb.ac.id

ABSTRAK

***Checkers* merupakan jenis permainan *game board*, yang mengandalkan strategi sebagai elemen utamanya. Permainan ini dimainkan oleh dua orang pemain dengan tujuan menghabiskan kepingan lawan. Permainan *checkers* yang dibuat dengan AI (*Artificial Intelligence*) tertentu menerapkan algoritma Minimax. Algoritma ini digunakan untuk menentukan pilihan agar memperkecil kemungkinan kehilangan nilai maksimal, yang akan mendeskripsikan dimana jika terdapat pemain yang mengalami pendapatan akan maka pemain lain akan mengalami kehilangan sebesar pendapatan tersebut. Untuk memperkecil lingkup pencarian pada algoritma Minimax, dikembangkan suatu algoritma yang dinamakan *alpha-beta* untuk mengurangi jumlah node pada pohon pencarian. Pada permainan *checkers* algoritma ini akan menentukan langkah yang diambil oleh AI agar menghasilkan pendapatan maksimum dengan mempertimbangkan kemungkinan langkah yang dapat dilakukan lawan selanjutnya.**

Kata kunci: *checkers, Artificial Intellingence, (AI), Minimax, zero-sum, alpha-beta, min, max, king, heuristic.*

1. PENDAHULUAN

Permainan merupakan suatu aktivitas yang biasa dilakukan untuk tujuan memberi hiburan atau rekreasi. Permainan dapat juga dilakukan untuk tujuan pendidikan. Salah satu kategori permainan adalah *board game*, yaitu permainan yang dilakukan pada suatu papan tertentu dimana pemain dilambangkan dengan *token*, beberapa *board game* juga menggunakan dadu atau kartu. *Board game* banyak mensimulasikan permainan peperangan, dengan papan sebagai peta dimana *token* pemain bergerak. Beberapa permainan lain, seperti catur, go, reversi, checkers, parchessi merupakan permainan yang mengandalkan strategi sebagai daya tariknya, biasa disebut permainan dengan strategi abstrak.

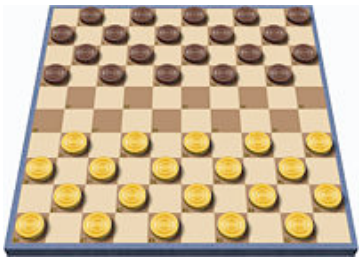
Permainan dengan strategi abstrak memberikan informasi langkah selanjutnya terhadap semua pemain, serta dalam permainan ini tidak terdapat kesempatan dan biasanya dimainkan dua pemain atau tim. Kemampuan pemain dalam mengambil keputusan merupakan faktor utama permainan. Setiap langkah yang diambil oleh pemain akan menghasilkan hasil yang berbeda-beda.

Perkembangan permainan saat ini telah sampai kepada pembuatan *artificial intelligent* (AI) sebagai teknik yang digunakan pada komputer atau *video game* untuk memproduksi ilusi atau kepintaran karakter yang bukan pemain. Dalam *boad game*, AI akan berperan sebagai musuh yang akan bergerak sesuai algoritma yang digunakan.

2. *CHECKERS*

Permainan *checkers* (dalam bahasa Inggris Amerika) atau disebut *draughts* (dalam bahasa Inggris British) merupakan permainan yang menggunakan strategi abstrak dimainkan oleh dua pemain dengan menggunakan langkah diagonal *token* dan menangkap dengan melompati *token* musuh.

Permainan ini telah dimainkan di Eropa sejak abad ke-16, dikembangkan dari permainan *alquerque*. Bentuk yang paling populer dari pemainan ini adalah *international draughts,* yang dimainkan pada papan 10x10. Bentuk yang juga populer adalah *English draughts*, yang disebut *American checkers*, dimainkan pada papan 8x8.



Gambar 1 *International Checkers*

2.1 Peraturan *Checkers*

Dimainkan oleh dua orang, dengan pemain berada pada sisi yang berlawanan dari papan. Salah satu pemain memiliki kepingan berwarna gelap dan pemain lain berwarna terang. Pemain dengan kepingan berwarna gelap melakukan langkah pertama, kecuali telah ditentukan sebelumnya. Kepingan akan bergerak diagonal dan

kepingan lawan ditangkap dengan meloncatinya. Kepingan yang ditangkap akan dihilangkan dari papan. Gerak kepingan pada papan hanya dapat dilakukan pada kotak yang tidak ditempati. Permukaan yang dapat menjadi papan permainan hanya kotak dengan warna gelap. Pemain yang kalah adalah pemain yang tidak memiliki kepingan yang tersisa atau tidak dapat melakukan langkah lagi.

Kepingan tanpa mahkota disebut orang, akan bergerak satu langkah maju diagonal dan menangkap kepingan dengan melakukan dua langkah pada arah yang sama, melompati kepingan lawan pada kotak tengah. Sejumlah kepingan lawan dapat ditangkap dengan satu loncatan, tidak harus pada arah yang sama tapi bisa *zigzag*. Pada *English draughts* kepingan hanya dapat ditangkap maju, tetapi pada *international draughts* kepingan dapat ditangkap mundur.

Ketika mencapai baris terjauh, kepingan berubah menjadi raja, ditandai dengan memberikan mahkota. Kepingan raja ini memiliki kekuatan tambahan untuk berjalan dan menangkap mundur (pada jenis yang tidak dapat melakukaknnya). Pada *international draughts*, raja dapat begerak sejauh yang ia inginkan secara diagonal.

3. ALGORITMA MINIMAX

Algoritma Minimax merupakan algoritma yang digunakan untuk menentukan pilihan agar memperkecil kemungkinan kehilangan nilai maksimal. Algoitma ini diterpkan dalam permainan yang melibatkan dua pemain seperti tic tac toe, checkers, go dan permainan yang menggunakan strategi atau logika lainnya. Hal ini berarti permainan-permainan tersbut dapa dijelaskan sebagai suatu rangkaian aturan dan premis.

Algoritma ini mulai dikembangkan dari teori game *zero-sum*. Teori ini mendeskripsikan situasi dimana jika terdapat pemain yang mengalami pendapatan, pemain lain akan mengalami kehilangan dengan nilai yang sama dari pendapatan tersebut, dan sebaliknya. Jumlah pendapatan dari pemain yang dikurangi dengan jumlah kehilangan akan berjumlah nol. Teori minimax menyatakan :

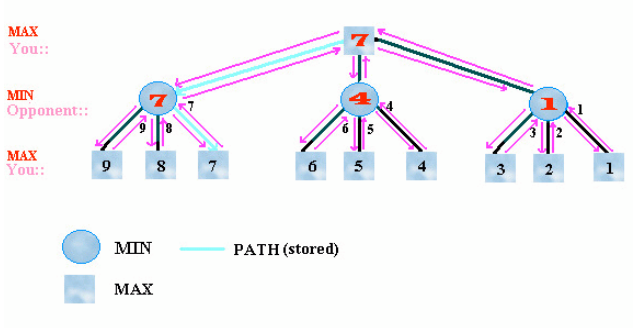
Untuk setiap dua orang pemain dalam *zero-sum game*, terdapat nilai V dari strategi yang dimiliki pemain seperti :

1. Stratregi yang ditentukan pemain kedua akan menghasilkan konsekuensi kemungkinan untuk pemain pertama, V
2. Strategi yang dutentukan pemain pertama akan menghasilkan konsekuensi kemungkinan untuk pemain pertama, -V

Secara setara, strategi pemain pertama akan memastikan suatu nilai V tanpa memperdulikan strategi pemain kedua, dan bersamaan dengan itu pemain kedua akan memastikan dirinya kehilangan nilai sebesar –V.

Algoritma Minimax merupakan algoritma dasar pencarian DFS (*Depth-First Search*) untuk melakukan traversal dalam pohon. DFS akan mengekspansi simpul paling dalam terlebih daulu. Setelah simpul akar dibangkitkan, algoritma ini akan membangkitkan simpul pada tingkat kedua, yang akan dilanjutkan pada tingkat ketiga, dst. Dalam melakukan treversal, misalkan dimulai dari suatu simpul *i*, maka simpul selanjutnya yang akan dikunjungi adalah simpul tetangga *j*, yang bertetangga dengan simpul *k*, selanjutnya pencarian dimulai lagi secara rekursif dari simpul *j*. Ketika telah mencapai simpul *m*, dimana semua simpul yang bertetangga dengannya telah dikunjungi, pencarian akan dirunutbalik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul *j* yang belum dikunjungi. Selanjutnya pencarian dimulai kembali dari *j*. Ketika tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi maka pencarian selesai.

Dalam repersentasi pohon dalam algoritma Minimax, terdapat dua jenis node, yaitu node *min* dan node *max*. *Max* node akan memilih langkah dengan nilai tertinggi dan *min* node akan memilih langkah dengan nilai terendah. Berikut merupakan gambar pohon untuk algoritma Minimax.



Gambar 2 Pohon Pencarian Algoritma Minimax

Dalam algoritma ini, langkah yang dapat dilakukan pemain ditentukan oleh langkah pemain lawan sebelumnya. Sebagai contoh pada tabel berikut di berikan tabel nilai yang memberitahukan hasil dari pilihan.

Tabel 1 Tabel Contoh Nilai Pilihan

	B memilih B1	B memilih B2	B memilih B3
A memilih A1	+3	-2	+2
A memilih A2	-1	0	+4
A memilih A3	-4	-3	+1

Pada tabel ini diperlihatkan setiap pemain memiliki tiga pilihan yang harus dipertimbangkan. Dengan mengasumsikan nilai pilihan yang dipilih untuk suatu pemain akan bernilai kebalikannya bagi pemain lawan. Maka pilihan minimal untuk A adalah A2 karena nilai terburuk adalah kehilangan -1, dengan pilihan minimax untuk B adalah B3 karena kemungkinan teburuk adalah mendapatkan nilai 1. Bagaimanapun, solusi ini tidak stabil, jika B mengira A akan memilih A2 maka B akan

memilih B1 untuk mendapatkan nilai 1. Jika A mengira B akan memilih B1 maka A akan memilih A1 untuk mendapatkan 3, maka B akan memilih B2 yang dimana kedua pemain akan menyadari kesulitan menentukan pilihan. Disinilah dibutuhkan strategi. Pada beberapa pilihan, terlihat dominasi salah satu pemain dan dapat dieliminasi, seperti : A tidak akan memilih A3 karena A1 dan A2 memiliki hasil yang lebih baik, apapun yang B pilih. B tidak akan memilih B3 karena B2 akan memberikan hasil yang lebih baik, apapun yang A pilih. A dapat menghindari kehilangan lebih dari 1/3 dengan memilih A1 dengan kemungkina 1/8 dan A2 dengan kemungkinan 5/6 apapun yang B pilih. B dapat memastikan pendapatan setidaknya 1/3 dengan menggunakan strategi acak untuk memilih B1 dengan kemungkinan 1/3 atau B2 dengan kemungkinan 2/3 apapun yang A pilih.

Berdasarkan contoh tersebut diketahui bahwa dalam algoritma ini terdapat dua peran, yaitu *max* dan *min*. Pembuatan pohon dimulai dari posisi awal hingga posisi akhir permainan. Sekanjutnya, posisi akhir dievaluasi dari sudut pandang *max*, seperti terdapat pada gambar 1. Setelah itu, node bagian dalam diisi dengan nilai yang telah dievaluasi. Node yang dimiliki *max* akan menerima nilai maksimum dari anak-anaknya. Node untuk *min* akan memilih nilai minumum dari anak-anaknya. Algoritma Minimax untuk hal ini, adalah :

```
function MinMax (game : GamePosition)
  return MaxMove (game)
```

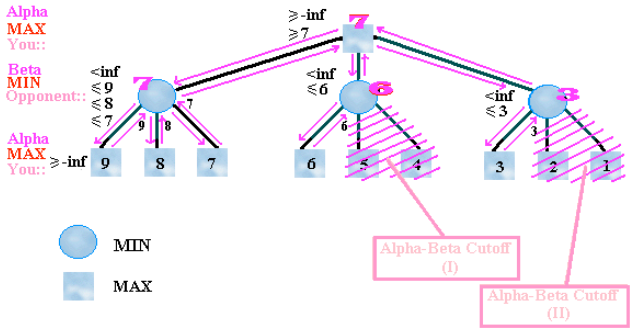
```
function MaxMove (game : GamePosition)
  if (GameEnded(game)) then
    return EvalGameState(game)
  else
    best_move ← ()
    moves ← GenerateMoves(game)
    ForEach moves do
       move ← MinMove(ApplyMove(game))
       if (Value(move) > Value(best_move))
then
          best_move ← move
       endif
    endfor
    return best_move
   endif
```

```
function MinMove (game : GamePosition)
if (GameEnded(game)) then
    return EvalGameState(game)
  else
  best_move ← ()
  moves ← GenerateMoves(game)
  ForEach moves do
     move ← MaxMove(ApplyMove(game))
     if (Value(move) > Value(best_move)) then
        best_move ← move
     endif
  endfor
  return best_move
endif
```

Pada algoritma tersebut `Value` akan merepresentasikan bagaimana baiknya langkah permainan, sehingga *max* akan mencoba memilih langkah dengan nilai terbesar. Tetapi *min*, yang juga memiliki peran terhadap langkah tersebut, akan mencoba memilih langkah yang lebih baik untuk dirinya, yang akan memperkecil hasil yang diperoleh *max*.

3.1 *Alpha-Beta Pruning*

Dalam algoritma Minimax, pencarian dilakukan pada seluruh bagian pohon, sementara sebagian pohon tidak seharusnya diperiksa. *Alpha-beta pruning* merupakan modifikasi dari algoritma Minimax, yang akan mengurangi jumlah node yang dievaluasi oleh pohon pencarian. Pencarian untuk node berikutnya akan dipikirkan terlebih dahulu. Algoritma ini akan berhenti mengevaluasi langkah ketika terdapat paling tidak satu kemungkinan yang ditemukan dan membuktikan bahwa langkah tersebut lebih buruk jika dibandingkan dengan langkah yang diperiksa sebelumnya. Sehingga, langkah berikutnya tidak perlu dievaluasi lebih jauh. Dengan algoritma ini hasil optimasi dari suatu algoritma tidak akan berubah. Berikut merupakan pohon dengan algoritma *alpha-beta pruning*.



Gambar 3 Pohon Pencarian Algoritma Minimax dengan *Alpha-Beta Pruning*

Diperlihatkan, pada pohon tersbut, bahwa terdapat pemotongan pencarian dengan menggunakan algoritma ini.

Pada algoritma ini, terdapat dua nilai yang diatur, yaitu alpha dan beta, yang merepresentasikan nilai minum dari *max* yang diyakini dan nilai maksimum dari *min* yang diyakini. Nilai awal alpha adalah tak hingga negatif dan nilai awal beta adalah tak hingga positif. Sebagai hasil dari proses rekursif, area pencarian akan semakin kecil. Ketika beta menjadi lebih kecil dari alpha, akan berarti posisi saat itu tidak dapat menjadi hasil terbaik permainan untuk kedua pemain dan pencarian tidak perlu dilakukan lebih jauh. Pseudocode untuk algoritma Minimax yang telah mengimplementasikan alpha beta pruning, yaitu :

```
function MinMax (game : GamePosition)
  return MaxMove (game)
```

```
function MaxMove (game : GamePosition, alpha
: integer, beta: integer)
  if (GameEnded(game)) then
    return EvalGameState(game)
  else
    best_move ← ()
    moves ← GenerateMoves(game)
    ForEach moves do
       move ← MinMove(ApplyMove(game), alpha,
beta)
       if (Value(move) > Value(best_move))
then
          best_move ← move
           alpha ← Value(move)
        endif
        if (beta>alpha)
           return best_move
        endif
    endfor
    return best_move
   endif
```

```
function MinMove (game : GamePosition, alpha
: integer, beta : integer)
if (GameEnded(game)) then
    return EvalGameState(game)
  else
  best_move ← ()
  moves ← GenerateMoves(game)
  ForEach moves do
     move ← MaxMove(ApplyMove(game), alpha,
beta)
     if (Value(move) > Value(best_move)) then
        best_move ← move
        beta ← Value(move)
     endif
     if (beta<alpha)
        return move
     endif
  endfor
  return best_move
endif
```

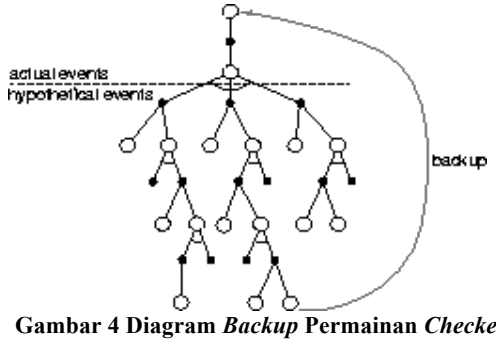
Pembentukan pohon DFS biasa membutuhkan big-O sebesar $O(b^m)$ dan dengan *alpha-beta pruning* pohon big-O akan menjadi sebesar $O(b^{m/2})$.

3. PENERAPAN ALGORITMA MINIMAX DALAM *CHECKERS*

Penerapan algoritma Minimax dalam *checkers* pertama kali dilakukan oleh Arthur Samuel dalam melakukan pembuatan program untuk mempelajari permainan *checkers*. Prah program yang dibuat berdasarkan prosedur Minimax untuk mendapatkan langkah terbaik dari posisi yang ada. Setiap posisi memiliki nilai yang dapat dihasilkan dari langkah terbaik, dengan berasumsi bahwa AI akan selalu mencoba memaksimalkan nilai, ketika lawan akan mencoba untuk meminimalkannya. Ketika prosedur minimax mencapai akar pada pohon pencarian (posisi saat tersebut), akan menghasilkan langkah terbaik dengan asumsi lawan akan menggunakan kriteria evaluasi yang sama. Beberapa versi program yang dibuat Samuel juga telah menerapkan algoritma pemotongan *alpha-beta*.

Samuel menggunakan dua macam metode, yang disebut *rote learning*. Metode tersebut memiliki penyimpan untuk setiap posisi yang ditemui selama permainan dengan tidak menghilangkan nilai yang ditentukan oleh prosedur Minimax. Hasilnya adalah jika terdapat posisi yang pernah ditenukan sebelumnya, akan dimunculkan sebagai posisi terminal pada pohon pencarian. Sehingga, pencarian akan semakin mudah karena nilai posisi diambil dari hasil pencarian yang telah dilakukan sebelumnya. Satu masalah awal yang ditemukan adalah program tidak mendukung untuk melangkah langsung menuju kemenangan. Samuel memberikan pengarahan dengan mengurangi sedikit nilai posisi setiap tahap (disebut *ply*) pada analisis Minimax. Jika program berhadapan dengan pilihan posisi dengan nilai yang hanya dibedakan oleh *ply*, maka program akan secara otomatis melangkah pada pilihan yang paling menguntungkan.

Samuel mencoba permainan ini berkali-kali dengan melawan berbagi versi sebelumnya dan melakukan *backup* pada setiap langkah. Ide ini digambarkan pada gambar berikut ini.



Gambar 4 Diagram *Backup* Permainan *Checkers* yang Dikembangkan Samuel

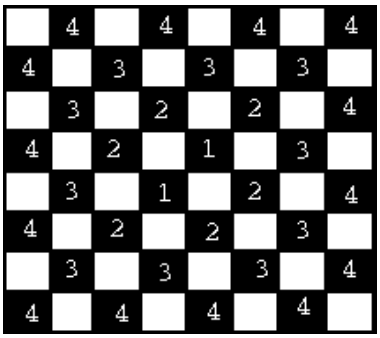
Setiap lingkaran putih merepresentasikan posisi langkah program berikutnya dan setiap lingkaran hitam merepresentasikan posisi langkah lawan selanjutnya. *Backup* dibuat untuk setiap nilai pada posisi setelah perpindahan sisi, yang akan menghasilkan langkah berikutnya. Hal ini dibuat berdasarkan nilai yang dihasilkan dari algoritma Minimax.

Perkembangan *checkers* menggunakan algoritma Minimax banyak dipengaruhi oleh pembuatan yang dilakukan Samuel tersebut. Untuk menerapkan algoritma Minimax pada permainan *checkers* diperlukan suatu fungsi optimasi tertentu yang ditambahkan. Salah satu fungsi optimasi yang paling dasar adalah membatasi kedalaman dari pohon pencarian. Jika permainan memiliki pohon 3-*ary*, maka pohon tersebut akan memiliki nilai yang diperlihatkan pada tabel berikut.

Tabel 2 Kedalaman Pohon Pencarian

Depth	*Node Count*
0	1
1	3
2	9
3	27
...	...
n	3^n

Berdasarkan tabel tersebut dapat dilihat bahwa untuk pohon pencarian dengan kedalaman 5 akan membutuhkan 1+3+9+27+81+243 = 364 * 1s = 364s = 6m. Waktu ini merupakan waktu yang sangat lama untuk ukuran permainan. Fungsi optimasi selanjutnya yang perlu ditambahkan adalah fungsi yang dibutuhkan untuk melakukan evaluasi posisi permainan dari pemain tertentu. Hal ini dapat dilakukan dengan memberikan nilai pada langkah tertentu pada permainan, seperti menghitung jumlah kepingan di papan atau jumlah langkah yang tersisa di akhir permainan. Sebagai pengganti sebaiknya diperlukan suatu fungsi estimasi yang dapat melakukan penghitungan kemungkinan posisi agar pemain dapat memenangkan permainan. Fungsi ini harus memiliki fungsi heuristik dari permainan tersebut. Pada *checkers*, kepingan pada pojok dan pinggir posisi tidak buat suatu fungsi dapat dimakan. Sehingga, dapat dibuat suatu fungsi yang memberikan nilai yang lebih tinggi pada posisi tersebut. Sebagai ilustrasi dapat dilihat pada gambar 5. Untuk memperkecil kemungkinan, fungsi heuristik nilai kepingan juga dapat ditambahkan, misalnya raja yang memiliki nilai lebih dibanding kepingan biasa.



Gambar 5 Pemberian Nilai pada *Checkers*

Sehingga algoritma Minimax dengan menerapkan *alpha-beta pruning* yang telah dimodifikasi untuk *checkers* akan menjadi :

```
function MinMax (game : GamePosition)
  return MaxMove (game)
```

```
function MaxMove (game : GamePosition, alpha
: integer, beta: integer)
  if (GameEnded(game) or DepthLimitReached)
then
    return EvalGameState(game,MAX)
  else
    best_move ← ()
    moves ← GenerateMoves(game)
    ForEach moves do
       move ← MinMove(ApplyMove(game), alpha,
beta)
       if (Value(move) > Value(best_move))
then
          best_move ← move
          alpha ← Value(move)
       endif
       if (beta>alpha)
          return best_move
       endif
    endfor
    return best_move
  endif
```

```
function MinMove (game : GamePosition, alpha
: integer, beta : integer)
if (GameEnded(game) or DepthLimitReached)
then
    return EvalGameState(game,MIN)
  else
  best_move ← ()
  moves ← GenerateMoves(game)
  ForEach moves do
     move ← MaxMove(ApplyMove(game), alpha,
beta)
     if (Value(move) > Value(best_move)) then
        best_move ← move
        beta ← Value(move)
     endif
     if (beta<alpha)
        return move
     endif
  endfor
  return best_move
endif
```

AI pada permainan *checkers* dapat dikembangkan untuk memiliki dua kemungkinan metode pencarian *alpha-beta*. Pertama, pohon pencarian akan mencari hingga kedalaman tertentu, misalkan, pada sebuah permainan ditetapkan kedalaman pohon tingkat 4-5 untuk AI pada level *beginner*, kedalaman 6-8 untuk level *intermediate* dan kedalaman 9-10 untuk level *advanced*. Metode kedua yaitu memungkinkan AI untuk mencari dalam waktu tertentu. Metode ini dianggap lebih baik, karena jika bergantung pada keadaan permainan sejumlah langkah yang mungkin untuk setiap posisi, pencarian berdasarkan kedalaman akan menghasilkan variasi pohon yang sangat berbeda-beda. Pada metode tersebut, pengurangan pohon pada kedalaman rendah tidak diperlukan. Dengan mencari pada waktu tertentu, AI memulai pencarian pada

kedalaman 4 dan melakukan pencarian lebih dalam secara iteratif dengan menambahkan kedalaman sebanyak 1 pada setiap pencarian. Jika waktu yang ditentukan habis pada tengah pencarian, pencarian akan dihentikan pada tingkatan tersebut dan langkah akan dihasilkan dari pencarian sebelumnya.

5. KESIMPULAN

Permainan *checkers* merupakan permainan yang dimainkan oleh dua orang denga tujuan untuk menghabiskan kepingan yang dimiliki lawan. Dalam pembuatannya dengan AI, permainan ini menerapkan algoritma Minimax. Algoritma Minimax memiliki dasar berupa *zero-sum game*, dimana jika pemain mendapatkan nilai tertentu maka pemain lain akan kehilangan nilai yang sama dengan pemain tersebut. Dengan menambahkan nilai *alpha-beta* algoritma Minimax akan memiliki pohon pencarian yang lebih singkat sehingga akan membutuhkan waktu singkat untuk melakukan aksinya. Nilai ini akan berhenti mengevaluasi langkah ketika terdapat paling tidak satu kemungkinan yang ditemukan dan membuktikan bahwa langkah tersebut lebih buruk jika dibandingkan dengan langkah yang diperiksa sebelumnya. Algoritma Minimax menggunakan DFS sebagai dsar pembuatan pohon pencarian.

Dengan menambahkan fungsi heuristik yang tepat pengambilan langkah yang dilakukan oleh AI akan memberikan hasil pendapatan yang lebih baik. Sehingga, penerapan algoritma akan mempersulit pemain untuk melawan AI. Karena AI akan mengevaluasi setiap langkah agar pendapatan yang diambil olehnya maksimum dengan memperhitungkan seluruh kemungkinan langkah pemain sehingga langkah pemain akan menghasilkan nilai minimum.

REFERENSI

[1] http://www.cs.ualberta.ca/~sutton/book/11/node3.html
Tanggal akses : 19 Mei 2008 pukul 16.00 WIB

[2] http://migo.sixbit.org/papers/AI_and_Perl/presentation.html
Tanggal akses : 19 Mei 2008 pukul 16.00 WIB

[3] http://www1.cs.columbia.edu/~devans/TIC/AB.html
Tanggal akses : 19 Mei 2008 pukul 16.00 WIB

[4] http://en.wikipedia.org/wiki/Draughts
Tanggal akses : 19 Mei 2008 pukul 16.00 WIB

[5] http://en.wikipedia.org/wiki/Minimax
Tanggal akses : 19 Mei 2008 pukul 16.00 WIB

[6] http://en.wikipedia.org/wiki/Alpha-beta_pruning
Tanggal akses : 19 Mei 2008 pukul 16.00 WIB

[7] http://ai-depot.com/articles/minimax-explained/
Tanggal akses : 19 Mei 2008 pukul 16.00 WIB