Penggunaan Algoritma *DFS* dalam Pencarian Strategi Permainan Catur

Muhammad Anwari Leksono - 13508037
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
e-mail : *if18037@students.itb.ac.id*

***Abstract*—Makalah ini membahas strategi *DFS* dalam permainan catur. Catur dikenal sebagai permainan strategi yang melibatkan dua pemain dengan masing-masing permain dibekali dengan 16 bidak catur. Pemain harus membuat bidak Raja lawan tidak berkutik (*checkmate*) untuk menang dalam permainan. Dalam permainan pemain dapat menggerakan bidak-bidak caturnya untuk menjebak Raja lawan.**

Algoritma *DFS* dapat digunakan untuk mencari simpul tertentu yang berada dalam pohon status yang lengkap. Status atau simpul yang dicari adalah simpul yang mengandung informasi bahwa lawan berada dalam keadaan *checkmate* atau *check* atau kedua pemain *remis*.

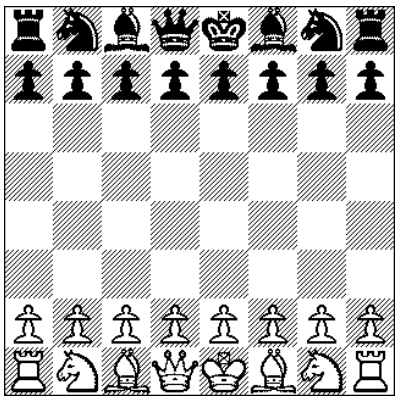
***Kata Kunci*—*DFS*, catur, permainan, strategi.**

I. PENDAHULUAN

Permainan catur telah dimainkan oleh jutaan orang di seluruh pelosok dunia. Catur bahkan telah menjadi cabang olahraga dan ajang kompetisi bergengsi di dunia. Catur dipertimbangkan sebagai permainan dengan usia yang sangat tua karena pada abad 16 kompetisi catur yang terorganisasi dengan baik telah muncul.

Catur merupakan permainan berbasis strategi yang kompleks sehingga dalam kurun waktu yang lama strategi dalam memenangkan permainan pun semakin meluas.

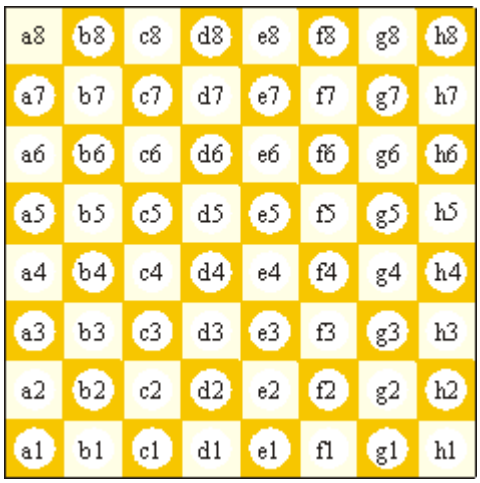
II. CATUR



Gambar 1 Permainan Catur

Permainan catur pertama kali ditemukan pada abad pertengahan yaitu abad kerajaan. Ketika itu dunia masih terdiri dari banyak kerajaan. Abad itu juga dikenal dengan *Medieval Age*. Hal ini tercermin dari bentuk dari bidak catur itu sendiri.

Permainan catur dikenal sebagai permainan strategi. Permainan catur dimainkan oleh dua orang pemain. Bidak catur memiliki dua warna yaitu hitam dan putih. Masing-masing pemain dapat memilih warna bidak yang ingin dia kendalikan. Permainan catur dilakukan pada papan catur yang memiliki luas 8x8 satuan luas. Jumlah petak adalah 64 buah dengan warna yang silih-berganti antara hitam dan putih. Warna putih akan memulai permainan. Dengan kata lain langkah pertama dilakukan oleh warna putih. Sedangkan warna hitam akan selalu memiliki giliran setelah putih. Tiap petak memiliki koordinat yang jelas. Hal ini dapat terlihat pada gambar berikut :



Gambar 2 Koordinat Papan Catur

Bidak catur berjumlah 32 buah dengan dua warna, hitam dan putih. Masing-masing warna terdiri dari 1 raja, 1 ratu, 2 gajah, 2 kuda, 2 benteng, dan 8 pion/prajurit. Masing-masing bidak memiliki cara gerak masing-masing. Raja hanya dapat bergerak satu petak ke segala arah yaitu ke depan, ke belakang, ke kiri, ke kanan, dan gerakan diagonal. Ratu dapat bergerak bebas ke segala arah, sama seperti Raja namun jumlah petak tidak dibatasi. Gajah hanya dapat bergerak diagonal dengan jumlah petak yang tidak dibatasi. Benteng dapat bergerak maju, mundur, kiri, dan kanan dengan jumlah petak yang tidak dibatasi. Kuda hanya dapat begerak dengan jalur berbentuk huruf '**L**'.

Permainan catur dianggap selesai jika salah seorang pemain dapat menjebak bidak raja milik lawan sehingga tidak bisa lagi menghindar dari bahaya. Dengan kata lain

posisi raja saat itu berbahaya dan bergerak kemana pun raja tetap dalam keadaan terancam atau sering disebut dengan *checkmate*. Selain *checkmate* ada pula kondisi *remis* atau seri. Kondisi ini terjadi ketika kedua pemain telah tidak memiliki bidak catur selain raja. *Remis* juga dapat terjadi ketika raja pihak lawan tidak memiliki ruang gerak yang aman. Dengan kata lain raja berada dalam posisi aman namun pemain tidak dapat menggerakkan rajanya ke tempat lain karena semua pilihan tempat tujuan hanya mengantarkan raja pada kematian oleh bidak catur lawan. Selain itu ada pula kondisi dimana Raja berada dalam posisi yang berbahaya. Kondisi ini disebut dengan *check.* Dalam permainan pemain harus mengingatkan lawan mainnya jika raja miliknya berada dalam kondisi *check* atau *checkmate*.

III. ALGORITMA *DFS*

DFS adalah singkatan dari *Depth First Search* atau penacrian runut ke bawah terlebih dahulu. Cara pencariannnya adalah dengan traversal yaitu tiap simpul pada graf akan dikunjungi satu per satu ke arah mendalam. Algoritma *DFS* secara umum dapat dijabarkan sebagai berikut :

1. Kunjungi simpul *v*.
2. Kunjungi simpul *w* yang bertetangga dengan simpul *w*.
3. Ulangi *DFS* dari simpul *w*.
4. Ketika sampai pada simpul *u* sedemikian sehingga semua tentangganya telah dikunjungi, pencarian runut balik dilakukan ke simpul terakhir yang dikunjungi sedemikian sehingga sempul tersebut masih memiliki tetangga yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Algoritma ini dapat ditulis dengan *pseudocode* sebagai berikut :

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan
algoritma pencarian DFS
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis ke
layar
}
Deklarasi
w : integer
Algoritma:
write(v)
dikunjungi[v]←true
for tiap simpul w yang bertetangga dengan
simpul v do
if not dikunjungi[w] then
DFS(w)
endif
  endfor
```

.

IV. APLIKASI *DFS* PADA PERMAINAN

Penggunaan *DFS* didasari oleh banyaknya kemungkinan susunan bidak catur yang mungkin terjadi pada permainan ketika salah satu pemain menggerakan bidaknya. Misalnya ketika pemain putih yang melangkah lebih dahulu, memilih bidak pion di ujung kiri pemain putih untuk melangkah sejauh dua langkah maka kemungkinan bidak yang akan dijalankan oleh lawannya ada sebanyak (8x2)+2 cara. Hal ini juga berlaku ketika pemain putih akan mengawali permainan. 8 adalah jumlah pion catur, 2 adalah kemungkinan pergerakan pion catur yaitu satu atau dua langkah ke depan. 2 yang lain adalah 2 bidak kuda yang dapat berjalan.

Pohon susunan bidak catur untuk pencarian strategi permainan catur memiliki simpul berupa informasi mengenai kondisi papan catur. Informasi ini berisi posisi semua bidak yang ada di atas papan serta status pemain (*check*, *chechkmate*, atau *remis*). Solusi dari pohon keputusan ini adalah ketika informasi menunjukkan bawah status pemain adalah *checkmate* atau *remis* dilihat dari susunan bidak yang ada di atas papan catur.

Jika permainan dibuat dalam bentuk program maka *header* program (dalam bahasa C/C++) kurang-lebih akan seperti di bawah ini.

```
  typedef struct {
        String ID_Bidak;
        int posisiBidak;
        String namaBidak;
        String warnaBidak;
        Boolean apakahBidakMati;
  } Bidak;

  typedef struct{
        Bidak bidakCatur;
        Boolean apakahAdaBidak;
  } PapanCatur;

  PapanCatur ChessBoard[8][8];

  typedef struct{
        String koordinatPetak;
        String isiPetak;
  } TabelPosisi;

  Stack<TabelPosisi> chessBoard;

  bool apakahCheck(int pemain);
  /*
        Memeriksa apakah pemain dalam keadaan
skak atau tidak. Jika ya nilai TRUE
dikembalikan. Jika tidak nilai FALSE
dikembalikan.
  */
  bool apakahCheckMate(int pemain);
  /*
        Memeriksa apakah pemain dalam keadaan
skakmat atau tidak. Jika ya nilai TRUE
dikembalikan. Jika tidak nilai FALSE
dikembalikan.
  */
  bool apakahRemis();
  /*
        Memeriksa apakah kondisi permainan dalam
keadaan remis atau tidak. Jika ya nilai TRUE
dikembalikan. Jika tidak nilai FALSE
dikembalikan.
  */
  static void set();
  /*
        set papan untuk permainan catur secara
lengkap.
```

```
        papan catur terisi oleh bidak-bidak catur
secara lengkap
        TabelPosisi terisi sesuai dengan keadaan
awal papan catur
        variabel apakahCheck diisi dengan nilai
false
        variabel apakahCheckMate diisi dengan
nilai false
        variabel apakahRemis diisi dengan nilai
false
   */
   static void update();
   /*
        memperbarui isi tabel posisi bidak pada
papan
        memperbarui nilai variabel apakahCheck,
apakahCheck, dan apakahCheckMate
   */

   static void pindahBidak(String namaBidakCatur,
int koordinatTujuan);
   /*
        pindahkan bidak yang memiliki nama
namaBidakCatur dan pindahkan posisi bidak
        sesuai dengan cara geraknya yang
bersesuaian dengan namaBidakCatur
        menuju koordinatTujuan. jika berhasil
maka posisi bidak akan diisi dengan
        koordinatTujuan, jika gagal, nilai tidak
berubah
   */

   static      void      eliminasiBidakMusuh(String
ID_BidakPenyerang, String ID_BidakTarget);
   /*
        mengeliminasi bidak target yang memiliki
ID ID_BidakTarget jika posisi bidak
        dengan ID ID_BidakPenyerang dan posisi
bidak dengan ID ID_BidakTarget cocok
        hasilnya posisi bidak penyerang akan
diisi dengan posisi bidak target
        nilai variabel apakahBidakMati untuk
bidak target, jika berhasil dieliminasi
        akan menjadi TRUE, jika gagal eliminasi,
FALSE
   */
```

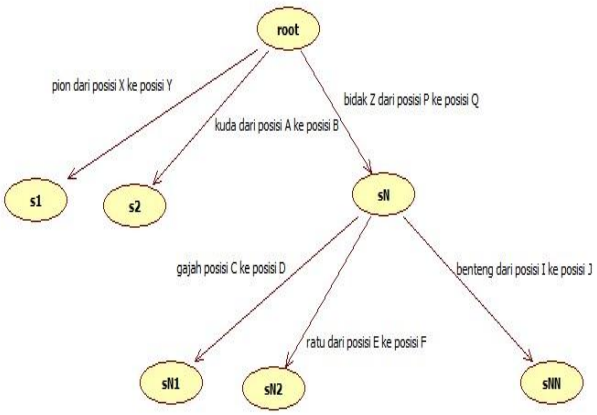
Pada header di atas, elemen *stack* dianggap telah terdefinisi di sini. Telah disebutkan di atas bahwa pada tiap giliran terdapat banyak kemungkinan pergerakan bidak catur dan hal ini akan membuat kombinasi susunan bidak catur akan semakin beragam namun di antara banyaknya regam tersebut hanya susunan tertentu yang dapat mematikan lawan dan untuk menuju susunan mematikan tersebut pemain harus melakukan beberapa langkah.

Pada awal permainan program akan menyimpan informasi letak semua bidak dengan tabel lalu tabel tersebut disimpan pada suatu tumpukan(*stack*), lalu dilanjutkan dengan pemeriksaan apakah ada keadaan skak atau *check*, keadaan *checkmate*, atau *remis*. Jika salah satu dari ketiga kondisi ini muncul maka permainan selesai dan pemenang akan muncul, jika tidak maka permainan akan berlanjut.

Program kemudian akan mencoba semua kemungkinan pergerakan bidak yang mungkin, diikuti dengan pengecekan status pemain dan papan, dan diteruskan pada penyimpanan informasi ini pada *stack*. Proses ini terus berulang sampai solusi ditemukan. Jika solusi tidak ditemukan pada suatu cabang pohon susunan bidak catur maka *stack* akan di-*pop* satu persatu elemennya dengan kata lain algoritma runut-balik bekerja di sini. Secara sederhana algortima ini dapat ditulis sebagai berikut :

1. Simpul awal (**So**) menyatakan situasi awal papan catur dan susunan bidaknya. Informasi disimpan pada *stack*.
2. Jika informasi mengandung status *checkmate* maka ini adalah solusi dan program berakhir. Jika tidak maka lanjutkan pada tahap 3.
3. Simpul membangkitkan anak simpul (**Si**) dengan suatu kondisi pembangkitan tertentu seperti pergerakan pion pada posisi **A** menuju posisi **B**. jika tidak memungkinkan pembuatan simpul anak maka lanjutkan pada tahap 5.
4. Pergantian giliran. Informasi **Si** terakhir disimpan pada *stack*. Kembali pada tahap 2.
5. Lakukan runut balik ke simpul di atas sampai pencarian tiba pada simpul yang masih mungkin memiliki simpul anak. Selama pencarian *stack* akan di-*pop*. Jika tidak ditemukan solusi maka lanjutkan pada tahap 6. Jika ditemukan maka lanjutkan pada tahap 2.
6. Permainan berakhir dengan hasil *remis*.

Dengan bentuk pohon penerapan algortima ini dapat terlihar seperti berikut :



Gambar 3 Pohon Keputusan Strategi Catur

Pencarian dimulai pada simpul ***root*** dan kemudian memeriksa anaknya yang pertama dan kemudian status diperiksa untuk susunan bidak catur seperti pada ***s1***. Jika ternyata ***s1*** adalah solusi maka pencarian selesai. Jika tidak maka runut balik dilakukan sampai ***root*** karena ia masih memiliki anak simpul yang belum dikunjungi(***s2***). Pengecekan status dilakukan lagi dan hal ini diulangi sampai sebuah solusi ditemukan atau tidak ada solusi sama sekali yang membuat permainan menjadi *remis*.

Jenis strultur data yang didapat dari algoritma di atas adalah sebuah *stack* dengan elemen yang berisi informasi mengenai tahap-tahap permainan dan hal ini dapat dijadikan solusi untuk permainan catur. Tentu saja hal ini

akan memakan banyak waktu karena sangat banyak kemungkinan yang akan terjadi dan mungkin saja suatu saat *stack* akan tidak mampu lagi menerima input lagi atau tidak dapat di-*push* lebih lanjut.

Bentuk informasi mengenai sususan bidak pada papan catur, yang disimpan pada *stack*, dapat secara sederhana dibuat seperti berikut :

ID Bidak	Posisi
Raja_1	E5
…	…

Tabel 1 Contoh Tabel Posisi Bidak Catur

Bentuk ID yang digunakan pada tabel di atas adalah nama bidak yang diikuti dengan penomoran bidak. Nomor **A** untuk warna putih dan **B** untuk warna hitam. Huruf **1** dan **2** menandakan jumlah bidak tersebut.

Pada saat permainan baru dimulai status berada pada simpul ***root*** dengan tabel yang berisi lokasi awal semua bidak yaitu :

ID Bidak	Posisi
Raja_A	D8
Ratu_A	E8
Gajah_A_1	C8
Gajah_A_2	F8
Kuda_A_1	G8
Kuda_A_2	B8
Benteng_A_1	A8
Benteng_A_2	H8
Pion_A_1	A7
Pion_A_2	B7
Pion_A_3	C7
Pion_A_4	D7
Pion_A_5	E7
Pion_A_6	F7
Pion_A_7	G7
Pion_A_8	H7

ID Bidak	Posisi
Raja_B	E1
Ratu_B	D1
Gajah_B_1	C1
Gajah_B_2	F1
Kuda_B_1	G1
Kuda_B_2	B1
Benteng_B_1	A1
Benteng_B_2	H1
Pion_B_1	A2
Pion_B_2	B2
Pion_B_3	C2
Pion_B_4	D2
Pion_B_5	E2
Pion_B_6	F2
Pion_B_7	G2
Pion_B_8	H2

Tabel 2 Tabel Posisi Awal Permainan

Setelah tabel selesai diatur, tabel ini dimasukkan ke dalam *stack*. Ketika ada pemain yang mulai menggerakkan bidaknya maka tabel akan berubah. Misalkan pemain A menggerakkan pion yang berada pada posisi C7 ke C6 dan pengecekan status selesai dan pergerakan ini mungkin dilakukan maka pion akan berpindah posisinya sehingga tabel akan berubah menjadi seperti dibawah ini :

ID Bidak	Posisi
…	…
Pion_A_3	C6
Pion_A_4	D7
…	…

Tabel 3 Tabel Posisi Setelah Pergerakan Pion

Tabel akan disimpan kembali pada *stack* dengan kata lain *stack* akan di-*push*.

Ketika program dimulai maka computer akan membuat pohon graf lengkap sehingga semua kemungkinan susunan bidak catur telah diketahui dengan baik oleh computer.

Ketika permainan dimulai *stack* kosong dibuat. Pemain dapat memilih apakah dia akan menjadi pemain putih atau hitam. Ketika pemain menggerakkan bidaknya maka akan terjadi perubaha susunan bidak catur pada papan permainan. Informasi susunan ini akan disimpan pada tabel seperti pada Tabel 1. Setelah itu tabel akan di-*push* ke *stack*.

Komputer akan merespon hal ini dan segera memulai pencarian pada pohon graf lengkap yang telah dibuat sebelumnya dengan algoritma *DFS*. Setelah computer menemukan simpul yang bersangkutan maka komputer akan memulai pencarian kembali untuk solusi permainan. Computer akan mencari simpul-simpul mana saja yang harus ia lewati untuk memenangkan permainan.Tiap simpul yang ia lewati ia akan menyimpan informasi itu dalam *stack* untuk keperluan runut balik jika diperlukan. Setelah ia mendapatkan solusi ia akan melakukan runut balik dengan *pop stack* tersebut sampai ia mendapatkan elemen sebelum elemen terakhir. Setelah ia mendapatkan elemen tersebut maka computer akan menggerakkan bidak yang sesuai agar susunan bidak pada papan permainan sesuai dengan elemen *stack* yang ia dapatkan dan setelah bidak dijalankan, *stack* dikosongkan kembali. Kemudian pemain akan mendapat giliran untuk bermain dengan menggerakkan bidaknya. Setelah ia menggerakkan bidaknya program akan memeriksa apakah dengan menggerakan bidak tersebut pemain akan skak atau tidak, skakmat atau tidak, remis atau tidak. Jika skak atau skakmat maka pemain tidak dapat menyelesaikan gilirannya. Bidak yang telah digerakkan akan dikembalikan ke posisi semula dan pemain diminta untuk menjalankan bidak lain.

Ketika pemain menjalankan bidak lain, pengecekan akan kembali terulang dan jika pemain lulus dalam pemeriksaan maka giliran pemain selesai dan informasi susunan bidak dalam ke dalam *stack*. Kemudian computer akan memulai pencarian solusi seperti pada paragraph sebelumnya namun pencarian tidak dimulai dari simpul awal ketika susunan bidak seperti awal permainan. Simpul yang manjadi awal pencarian adalah simpul yang informasi didalamnya bersesuaian dengan elemen yang telah di-*push* pada *stack* tersebut.

Tiap giliran akan diperiksa pergerakan bidaknya sehingga tidak ada pergerakan yang membuat pemain berada dalam keadaan *checkmate* atau *check* tanpa peringatan. Hal ini sama dengan pergerakan yang dilakukan computer.

V. ANALISA HASIL PENERAPAN

Algoritma *DFS* dapat bekerja dengan baik pada permainan catur. Pohon yang dibentuk pada awal permainan mencakup semua kemungkinan yang ada dan kemudian pohon itu disimpan agar pada saat permainan

diulang pohon tidak perlu dibuat dari awal. Simpul yang merupakan solusi tidak selalu berada di daerah kiri atau kanan karena dari tiap susunan bidak catur akan terdapat lebih dari satu solusi. Hal ini yang membaut pencarian memakan waktu cukup lama.

Selain itu proses pembuatan pohon juga memakan waktu yang tidak kalah lama karena jumlah kemungkinan pergerakan bidak sangat variatif. Pada saat permainan baru akan dimulai jumlah gerakan yang mungkin dilakukan pemain bisa mencapai (8x2)+2. Angka ini didapat dengan perincian berikut :

a. Jumlah pion ada 8 buah dan masing-masing pion dapat maju 2 langkah ke depan atau satu langkah saja. Dari sini didapat (8x2) macam gerakan.
b. Bidak kuda dapat bergerak melangkahi bidak lain karena jalur gerakan bidak ini membentuk huruf 'L'. jumlah kuda 2 buah dan masing-masing memiliki satu kemungkinan gerakan.

Jumlah gerakan itu masih sangat sedikit jika dibandingkan dengan permainan yang telah berlangsung. Misalkan bidak ratu. Bidak ini dapat berjalan dengan leluasa ke segala arah tanpa melompati bidak lain seperti bidak kuda. Misalkan posisi ratu ada di petak **D4**. Kita dapat melihat bahwa ratu akan memiliki jumlah gerakan yang mungkin dia lakukan sebanyak hamper 20 gerakan. Hal ini belum dihitungan dengan gerakan dia dalam rangka mengeliminasi bidak musuh dan dalam permainan kita tidak hanya memiliki satu bidak saja tapi 16 buah bidak. Dari titik ini kita dapat menduga bahwa pohon yang akan dibentuk akan sangat besar dan kompleks serta jumlah solusi yang banyak.

Selain pembuatan pohon yang memakan waktu lama, *stack* yang dibutuhkan dalam permainan ini juga memerlukan alokasi memori besar atau *stack* harus dapar digunakan untuk menyimpan tabel posisi bidak dalam jumlah banyak. Hal lain lain yang cukup rumit adalah mekanisme pengecekan status *checkmate* atau *check* atau *remis*. Tiap kondisi dapat terdiri dari banyak macam susunan bidak dan untuk memeriksa apakah raja berada dalam status *checkmate* atau *check* atau *remis* diperlukan cara yang cukup sulit. Misalkan raja berada pada titik **D4** dan untuk memeriksa raja tersebut kita harus menelusuri tiap petak baik secara diagonal, vertical, maupun horizontal. Tiap petak akan diperiksa apakah ada bidak atau tidak, jika tidak maka pencarian menjadi lebih mudah, jika ada maka bidak tersebut harus dilihat semua kemungkinan gerakannya.

Proses pembaruan tabel posisi tidak memakan waktu dan memori besar karena proses itu hanya pengisian tabel dengan jumlah elemen 36 buah saja.

Kelemahan pada penggunaan metode dalam permainan catur adalah computer akan bekerja cukup berat untuk membangkitkan graf yang berisi semua kemungkinan susunan bidak catur. Pencarian solusi pun akan menjadi sangat panjang dan kompleks sehingga selain computer harus bekerja keras untuk membuat graf, ia juga harus mencari solusi yang ada.

VI. KESIMPULAN

Komputasi pada pencarian solusi akan sangat rumit dan banyak. Solusi yang dapat dihasilkan pun akan berjumlah sangat banyak sehingga komputasi akan memakan waktu sangat lama.

Algortima *DFS* dapat digunakan untuk mencari semua kemungkinan solusi yang ada. Pada permainan catur dengan AI yang menggunakan metode ini maka tiap pola permainan permain akan dicatat dan AI akan menyesuaikannya dengan mencari simpul yang mengandung informasi yang sesuai dengan gerakan kita. Hal ini mengakibatkan pencarian dengan *DFS* akan dilakukan berulang dan menghabiskan waktu cukup lama.

REFERENSI

[1] History of Chess
http://www.essortment.com/all/chesshistory_rmct.htm
waktu akses : 4 Desember 2010 pukul 4.00-4.30 WIB.

[2] FIDE Laws of Chess
http://www.fide.com/fide/handbook.html?id=125&view=article
waktu akses : 4 Desember 2010 pukul 4.00-4.30 WIB

[3] Wikipedia
http://en.wikipedia.org/wiki/Chess
waktu akses : 4 Desember 2010 pukul 4.00-4.30 WIB
http://en.wikipedia.org/wiki/Template_talk:Chess_position
waktu akses : 7 Desember 2010 pukul 18.24-18.40 WIB

[4] Munir, Rinaldi. 2005. *Diktat Kuliah Strategi Algoritmik IF3051Strategi Algoritmik.* Departemen Teknik Informatika ITB

[5] Chess Zone
http://www.thechesszone.com/images/articles/chess_rules_initial_board.gif
waktu akses : 7 Desember 2010 pukul 08.35-08.42 WIB

[6] Free Chess
http://www.free-chess.net/Chess_Rules_files/squares.gif
waktu akses : 7 Desember 2010 pukul 19.00-19.06 WIB

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, 8 Desember 2010

Muhammad Anwari Leksono - 13508037