GAME PLAYING UNTUK OTHELLO DENGAN MENGGUNAKAN ALGORITMA NEGASCOUT DAN MTDF

Gunawan, Yosi Kristian, Hermawan Andika
Jurusan Teknik Informatika, Sekolah Tinggi Teknik Surabaya
Jl.Ngagel Jaya Tengah 73-77, Surabaya 60284
Telp. (031) 5027920, Faks. (031) 895007
E-mail: admin@hansmichael.com, Yosi@stts.edu, shoya_motor_sport@yahoo.co.id

ABSTRAK

Game playing adalah salah satu sistem kecerdasan buatan (AI). Untuk permasalahan pada permainan yang berbasis pada giliran pemain, salah satu algoritma yang cukup membudaya adalah Negascout dan MTDF (Memory-enhanced Test Driver value f) yang diimplementasikan pada game tree algoritma minimax. Dengan konsep utama yang terletak pada semua kemungkinan yang bisa ditelusuri pada permainan. Dalam hal ini, permainan othello dapat menggunakan algoritma ini karena permainan dapat diimplementasikan dalam sebuah tree. Tree memiliki cabang-cabang yang terdiri dari node yang akan menyatakan nilai yang selanjutnya akan digunakan dalam menentukan langkah terbaik dari permainan. Nilai tersebut didapat dari proses evaluasi terhadap segala kemungkinan yang terjadi pada tiap perubahan keping dari papan permainan othello. Nilai evaluasi ini berkisar antara minus tak hingga sampai tak hingga. Beberapa cara yang dapat digunakan dalam mengevaluasi nilai pada permainan othello ini, diantaranya adalah dengan mengevaluasi jumlah langkah yang dapat dilakukan oleh pemain pada tiap kali kesempatan, memperkecil kemungkinan dari keping pemain yang berbatasan dengan petak kosong dan penguasaan pada posisi-posisi pojok dari papan permainan.

Kata Kunci: Game playing, minimax, Negascout, MTDF, othello

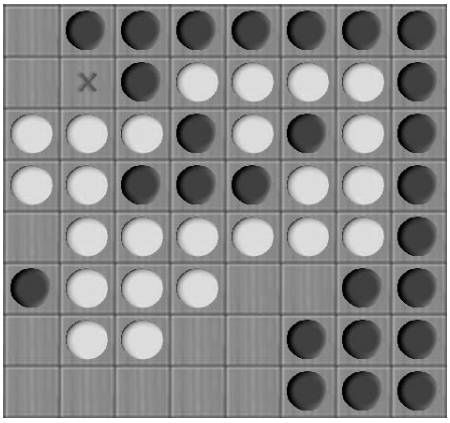
1. PENDAHULUAN

Pada permainan othello beberapa faktor utama yang diperlukan dalam memperbesar kemenangan yang dapat diperoleh, yaitu dengan mengevaluasi nilai yang didapat dari tiap perubahan keping atau disc pada papan permainan dan pencarian nilai terbaik dari hasil evaluasi yang telah dilakukan (Chuong Do, 2002). Untuk selanjutnya akan dipakai disc untuk menunjukkan keping othello. Adapun beberapa cara yang dapat digunakan dalam proses evaluasi, antara lain mobility, potential mobility atau frontier dan penguasaan corner. Proses evaluasi ini disebut juga dengan fungsi evaluasi. Sedangkan untuk pencarian nilai terbaik, beberapa metode yang dapat digunakan adalah Negascout dan MTDF. Pencarian nilai terbaik ini disebut juga dengan pencarian minimax.

2. OTHELLO

Sejarah othello berawal tahun 1945, setelah bom atom dijatuhkan di Hiroshima dan Nagasaki. September 1945, Hasegawa Goro yang tengah duduk di kelas satu SMP menerima pelajaran sembari duduk di tanah di bawah langit biru. Mito juga menjadi sasaran pengeboman hingga kastil dan bangunan bersejarah lain ikut habis dalam kobaran api. Dalam suasana seperti itulah permainan ini dilahirkan. Aturan awalnya, bila batu milik pemain pertama diapit oleh batu milik pemain kedua, maka batu pemain pertama menjadi milik pemain kedua. Permainan ini cukup merepotkan karena harus memakai batu dengan 2 warna, hingga akhirnya dipakai kertas yang diwarnai hitam pada salah satu sisi dan putih pada sisi yang lain.

Sebenarnya asal kata othello itu sendiri, bukan dari bahasa Jepang. Ayah Hasegawa Goro, memberikan nama ini dari salah satu karya Shakespeare dengan judul yang sama. Othello dalam karya Shakespeare tersebut mengisahkan seorang kulit hitam yang mempunyai istri kulit putih yang cantik bernama Desdemona.



Gambar 1. Papan permainan

Pada gambar 1 menunjukkan salah satu contoh permainan pada othello. Terlihat terdapat tanda silang pada gambar tersebut yang menandakan bahwa disc hanya dapat diletakkan pada daerah yang bertanda silang. Pada gambar tersebut giliran pemain disc putih yang melakukan peletakkan disc. Disc hitam pada gambar tersebut telah menguasai

dua daerah pojok, yang merupakan daerah yang menguntungkan dalam permainan othello.

3. GAME PLAYING

Pada teori game playing ini berisi mengenai metode-metode pencarian minimax. Pencarian minimax merupakan pencarian nilai terbaik dari nilai-nilai evaluasi yang didapat dari berbagai macam cara untuk menghitung nilai evaluasi tersebut. Pencarian ini bekerja dengan cara menelusuri segala kemungkinan yang terjadi pada papan dengan melakukan pencarian untuk beberapa langkah kedepan (Aske Plaat, 1994). Berbagai macam metode yang dapat digunakan dalam pencarian minimax, beberapa di antaranya adalah Negascout dan MTDF.

3.1 Negascout

Negascout adalah salah satu metode pencarian minimax dengan berasumsi bahwa langkah pertama yang diambil merupakan langkah terbaik, sedangkan sisanya merupakan langkah terburuk (Aske Plaat, 1994). Namun jika ternyata ada langkah yang lebih baik dari langkah pertama, maka akan terjadi proses *research* atau proses pencarian ulang. Berikut penjelasan algoritma Negascout yang dijabarkan pada gambar 2.

```
Function Negamax(u,Alpha,Beta)
1. [Cek apakah node u adalah leaf]
   IF node u = leaf THEN
   1.1 [Panggil fungsi evaluasi]
       RETURN eval(u)
   ELSE
   1.2 B ← Beta
   1.3 REPEAT FOR I = 1,2,3,...,jumlah anak
       1.3.1 t ← -Negascout(u[i],-B,-Alpha)
       1.3.2 IF (t>B) and (t<Beta) and
             (i>1) and (depth>1) THEN
             1.3.2.1 t ← -Negascout(u[i],-Beta,-t)
       1.3.3 Alpha ← max(Alpha, t)
       1.3.4 IF Alpha >= Beta THEN
             1.3.4.1 RETURN Alpha
       1.3.5 B = Alpha + 1
   1.4 RETURN Alpha
```

Gambar 2. Algoritma Negascout

Pada algoritma dari gambar 2, syarat untuk melakukan research adalah jika nilai dari *t* lebih besar dari *B*, lebih kecil dari beta, bukan anak pertama dari node yang diproses (i>1), dan pada node tersebut harus mempunyai anak sampai kedalaman lebih besar dari satu (depth>1). Syarat *t* lebih kecil dari beta diperlukan karena jika *t* lebih besar dari beta, maka beta pruning akan diproses sehingga proses research tidak akan terjadi.

3.2 Memory-enhanced Test Driver value f

Pencarian pada MTDF menggunakan bound sebagai tempat penyimpanan nilai minimax, dimana bound tersebut terbagi menjadi 2 macam, yaitu upperbound dan lowerbound yang menunjukkan rentang dimana nilai minimax berada (Aske Plaat, 1994). Cara kerja dari algoritma mtdf adalah dengan cara melakukan serangkaian pemanggilan algoritma alpha beta secara berulang-ulang. Berikut penjelasan algoritma MTDF yang dijabarkan pada gambar 3.

```
Function MTD(u,f)
1. [Inisialisasi variabel]
   1.1 g ← f
   1.2 Upperbound ← ∞
   1.3 Lowerbound ← -∞
2. [Melakukan iterasi MTD(f)]
   2.1 REPEAT WHILE Upperbound > Lowerbound
       2.1.1 IF g = Lowerbound THEN
             2.1.1.1 Beta ← g +1
             ELSE
             2.1.1.2 Beta ← g
       2.1.2 g ← AlphabetaWithMemory(u,Beta-1,Beta)
       2.1.3 IF g < Beta THEN
             2.1.3.1 Upperbound ← g
             ELSE
             2.1.3.2 Lowerbound ← g
```

Gambar 3. Algoritma MTDF

Pada algoritma gambar 3, parameter *f* merupakan nilai perkiraan. Jika nilai dari parameter tersebut mendekati nilai minimax maka proses pemanggilan terhadap algoritma alpha beta akan semakin sedikit. Minimal dilakukan dua kali proses pemanggilan, dimana untuk menentukan nilai dari upperbound dan lowerbound . Proses pemanggilan algoritma alpha beta akan berhenti jika nilai dari lowerbound lebih besar dari upperbound.

Algoritma alpha beta yang digunakan sedikit berbeda dengan algoritma alpha beta konvensional. Dimana pada algoritma *alpha beta with memory* menyimpan nilai upperbound dan lowerbound tiap-tiap node, yang nantinya digunakan untuk perbandingan dengan nilai alpha dan beta yang didapat. Berikut penjelasan algoritma *alpha beta with memory* yang dijabarkan pada gambar 4.

Pada algoritma gambar 4, dapat dilihat pada baris 1, merupakan proses pengambilan nilai upperbound dan lowerbound yang kemudian dicek dengan nilai alpha beta saat itu, dari situ akan banyak pruning yang akan dihasilkan. Pruning terjadi untuk lowerbound jika nilai lowerbound lebih besar sama dengan nilai beta, kemudian nilai yang dikembalikan adalah nilai lowerbound saat itu. Pruning terjadi untuk upperbound jika nilai upperbound kurang dari sama dengan nilai alpha, kemudian nilai yang dikembalikan adalah nilai upperbound saat itu.

```
Function Alphabeta(u,Alpha,Beta)
1. IF retrieve(u) = OK THEN
   1.1 IF u.Lowerbound >= Beta THEN
       1.1.1 RETURN u.Lowerbound
   1.2 IF u.Upperbound <= Alpha THEN
       1.2.1 RETURN u.Upperbound
   1.3 Alpha ← max(Alpha, u.Lowerbound)
   1.4 Beta ← min(Beta, u.Upperbound)
2. [Cek apakah node u adalah leaf]
   IF node u = leaf THEN
   2.1 [Panggil fungsi evaluasi]
       RETURN eval(u)
   ELSE
   2.2 [Cek apakah node u adalah node max]
       IF u = max THEN
       2.2.1 g ← -∞
       2.2.2 a ← Alpha
       2.2.3 REPEAT FOR I = 1,2,3,...,jumlah anak
             2.2.3.1 g ← max(g, Alphabeta(u[i], a, Beta))
             2.2.3.2 a ← max(a, g)
             2.2.3.3 IF g >= Beta THEN
                     2.2.3.3.1 Break
       ELSE
       2.2.4 g ← ∞
       2.2.5 b ← Beta
       2.2.6 REPEAT FOR I = 1,2,3,...,jumlah anak
             2.2.6.1 g ← min(g, Alphabeta(u[i], Alpha, b))
             2.2.6.2 b ← min(b, g)
             2.2.6.3 IF g <= Alpha THEN
                     2.2.6.3.1 Break

3. [Cek apakah Fail Low]
   IF g <= Alpha THEN
   3.1 u.Upperbound ← g
   3.2 store u.Upperbound
4. [Cek apakah Fail High]
   IF g >= Beta THEN
   4.1 u.Loverbound ← g
   4.2 store u. Lowerbound
5. RETURN g
```

Gambar 4. Algoritma alpha beta with memory

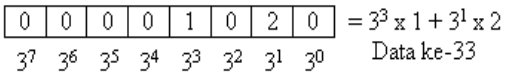
Pada algoritma gambar 4, untuk baris ke 3 dan 4 merupakan proses penyimpanan nilai upperbound dan lowerbound kedalam memory node saat itu. Untuk baris 2 terlihat sama dengan algoritma pada alpha beta konvensional. Jadi pada algoritma alpha beta with memory, terdapat tambahan algoritma untuk menyimpan nilai upperbound dan lowerbound, yang mana nilai minimax berada diantara nilai upperbound dan lowerbound.

4. FUNGSI EVALUASI

Banyak orang beranggapan bahwa yang terpenting dalam memenangkan permainan othello adalah jumlah disc yang dapat dibalik atau dirubah pada tiap kali kesempatan. Anggapan tersebut tidaklah benar, yang terpenting dalam permainan tersebut sebenarnya ada pada jumlah langkah yang dapat diambil disebut juga dengan mobility. Semakin banyak mobility yang didapat maka akan semakin banyak pula langkah-langkah bagus yang dapat dimainkan. Berikut beberapa fungsi evaluasi yang digunakan beserta komponen pendukungnya, antara lain edge table, mobility, liberties, frontier dan penguasaan corner.

4.1 Edge Table

Edge table merupakan wadah untuk menyimpan nilai-nilai mobility dari papan permainan. Cara kerja edge table adalah dengan hanya mengevaluasi satu sisi papan, yang mana selanjutnya hanya perlu mencerminkannya kesemua sisi-sisi pada papan permainan. Berikut penjelasan berupa gambar contoh cara penyimpanan nilai pada edge table.



Gambar 5. Penyimpanan edge table

Maksud dari penjelasan pada gambar 5 adalah jika misalkan terdapat pola pada salah satu sisi papan 00001020 (0 menunjukkan petak kosong, 1 disc putih, 2 disc hitam), maka nilai dari hasil pola tersebut akan disimpan pada edge table pada index ke-33. Selanjutnya data yang telah tersimpan dalam edge table akan diolah oleh mobility.

4.2 Mobility

Seperti yang telah dijelaskan sebelumnya bahwa mobility merupakan jumlah langkah yang dapat dimainkan oleh pemain pada tiap kali kesempatan. Jumlah langkah ini didapat dengan mengevaluasi pola pada satu sisi papan dan selanjutnya disimpan kedalam edge table. Berikut penjelasan berupa gambar tabel, yang berisi mengenai beberapa contoh cara mengevaluasi nilai mobility.

Tabel 1. Contoh evaluasi mobility

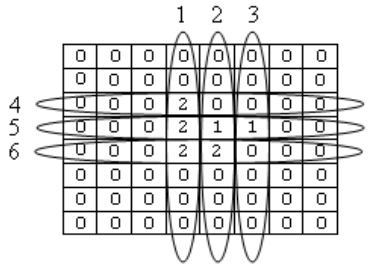
No	*Index Edge*	*Pola*	*Mobility*	
			Disc Putih	*Disc Hitam*
1	$3^0 \times 1 + 3^1 \times 2 = 7$	0 0 0 0 0 0 2 1 ↑	1	0
2	$3^2 \times 2 + 3^3 \times 1 = 45$	0 0 0 0 1 2 0 0 ↑ ↑	1	1
3	$3^0 \times 1 + 3^1 \times 2 + 3^3 \times 2 + 3^4 \times 1 = 142$	0 0 0 1 2 0 2 1 ↑ ↑	1	1
4	$3^2 \times 1 + 3^3 \times 2 + 3^5 \times 2 + 3^6 \times 1 = 1278$	0 1 2 0 2 1 0 0 ↑ ↑ ↑	1	2

Pada tabel 1, pada kolom pola terdapat tanda panah ke atas, yang menunjukkan posisi-posisi dimana pemain dapat meletakkan disc. Pemberian nilai pada kolom mobility ini berdasarkan jumlah posisi disc yang dapat diletakkan dan harus sesuai dengan jenis disc yang diletakkan.Berikut langkah-langkah yang diperlukan mulai dari awal hingga didapat nilai mobility total untuk keseluruhan sisi papan.

a. Siapkan 5 jenis edge table dengan ukuran penampung berbeda-beda. Edge untuk 8 petak dengan ukuran 6561, 7 petak ukuran 2187, 6 petak 729, 5 petak 243 dan 4 petak 81. Diperlukan edge untuk menampung 7 sampai 4 petak, karena untuk mengevaluasi petak dengan sisi diagonal.

b. Proses semua kombinasi pola pada satu sisi papan mulai dari 00000000 sampai 22222222 dan hitung jumlah mobility yang ada seperti pada contoh tabel evaluasi mobility, kemudian simpan kedalam edge table.
c. Evaluasi seluruh sisi papan, baik horisontal, vertikal dan diagonal. Keseluruhan ada 34 sisi yang diproses, 8 sisi horisontal, 8 sisi vertikal dan 18 sisi diagonal.

Setelah mengetahui bagaimana langkah-langkah yang diperlukan dalam mengevaluasi nilai mobility, selanjutnya akan dijelaskan mengenai contoh permasalahan yang mungkin terjadi papan permainan. Berikut contoh gambar permasalahan yang mungkin terjadi, dapat dilihat pada gambar 6.



Gambar 6. Contoh permasalahan papan

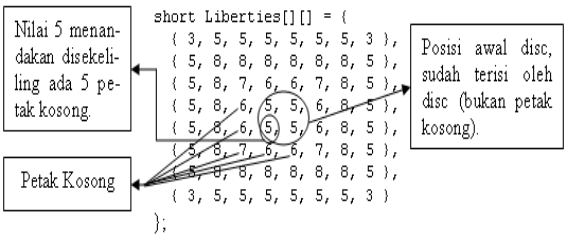
Berikut penjelasan mengenai cara perhitungan berdasarkan pada gambar 6. Sebelumnya perlu diketahui untuk perhitungan hanya untuk disc putih dan sisi-sisi yang dilingkari saja yang akan dihitung (pada penerapan, seluruh sisi horisontal, vertikal dan diagonal perlu dihitung).

Mobility 1 = 0 ← Edge_Tabel[3^5 x 2 + 3^4 x 2+ 3^3 x 2 = 702]
Mobility 2 = 1 ← Edge_ Tabel[3^4 x 1 + 3^3 x 2 = 135]
Mobility 3 = 0 ← Edge_ Tabel[3^4 x 1 = 81]
Mobility 4 = 0 ← Edge_ Tabel[3^4 x 2 = 162]
Mobility 5 = 1 ← Edge_ Tabel[3^4 x 2 + 3^3 x 1 + 3^2 x 1 = 198]
Mobility 6 = 0 ← Edge_ Tabel[3^4 x 2 + 3^3 x 2 = 216]
Nilai evaluasi = Mobility 1 + … + Mobility 6
= 0 + 1 + 0 + 0 + 1 + 0 = 2

Berdasarkan pada contoh permasalahan yang terjadi pada papan pada gambar 6 dan hasil perhitungan nilai mobility yang telah dilakukan dapat disimpulkan bahwa nilai evaluasi total yang didapat untuk disc putih adalah 2. Nilai evaluasi total berdasarkan perhitungan yang telah dilakukan didapat dengan cara menjumlahkan seluruh mobility yang didapat dari hasil evaluasi yang telah diproses untuk sisi horisontal dan vertikal dari contoh permasalahan pada gambar 6.

4.3 Liberties

Liberties digunakan untuk menampung jumlah petak kosong yang berada disekeliling tiap-tiap petak pada papan. Berikut nilai-nilai yang disimpan kedalam liberties dapat dilihat pada gambar inisialisasi nilai-nilai liberties.



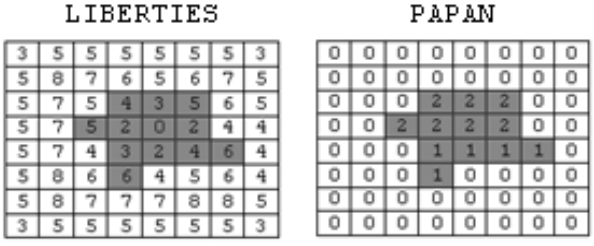
Gambar 7. Inisialisasi nilai-nilai liberties

Setelah proses penyimpanan nilai-nilai liberties dilakukan, selanjutnya nilai-nilai yang disimpan tersebut akan diproses oleh frontier untuk mengetahui jumlah disc yang berbatasan dengan petak kosong.

4.4 Potential Mobility atau Frontier

Frontier merupakan jumlah disc yang berbatasan dengan petak kosong. Jika terjadi kondisi dimana semakin banyak frontier yang didapat, maka akan semakin jelek pula posisinya, karena semakin banyak frontier memungkinkan lawan untuk mendapatkan semakin banyak mobility tambahan pada beberapa langkah ke depan dan juga mengurangi mobility-nya sendiri. Berikut penjelasan berupa gambar contoh nilai-nilai liberties dan papan, untuk selanjutnya dihitung frontiernya.

Berdasarkan gambar 8 cara menghitung frontier adalah jumlahkan banyaknya disc pada papan dengan melihat nilai-nilai pada liberties yang lebih besar dari nol. Dari gambar contoh nilai-nilai liberties dan papan, untuk nilai frontier disc putih sebanyak 5, sedangkan untuk disc hitam sebanyak 6 (jumlah disc 7, namun nilai liberties pada petak (5,4) adalah nol, jadi tidak dihitung).



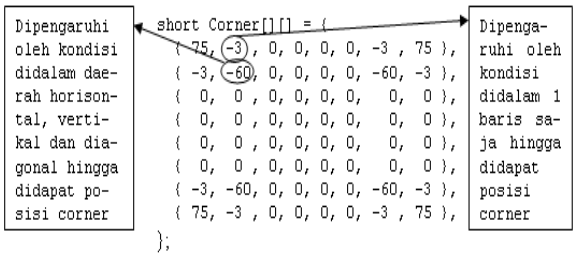
Gambar 8. Contoh liberties dan papan

Gambar contoh nilai-nilai liberties pada gambar 8 dengan gambar inisialisasi liberties pada gambar 7 berbeda, karena perubahan yang terjadi pada papan

mengakibatkan nilai-nilai liberties juga ikut berubah. Cara merubah nilai-nilai pada liberties adalah dengan mengurangi sebanyak satu kali dari nilai liberties tiap-tiap petak yang berbatasan dengan disc yang baru saja diletakkan.

4.5 Penguasaan Corner

Penguasaan corner merupakan penguasaan terhadap posisi-posisi pojok dari papan, karena disc yang diletakkan pada posisi tersebut tidak dapat dirubah atau dibalik. Jadi kemungkinan besar kemenangan dapat diraih dengan menguasai posisi-posisi pojok tersebut. Berikut penjelasan berupa gambar mengenai pemberian nilai pada posisi-posisi pojok tersebut dapat dilihat pada gambar inisialisasi nilai-nilai corner.



Gambar 9. Inisialisasi nilai-nilai corner

Pemberian nilai-nilai pada gambar inisialisasi nilai-nilai corner hanya berupa nilai perkiraan saja. Pemberian nilai -3 karena pada posisi tersebut tidak terlalu rawan, sedangkan nilai -60 merupakan posisi paling rawan untuk mendapatkan posisi pojok.

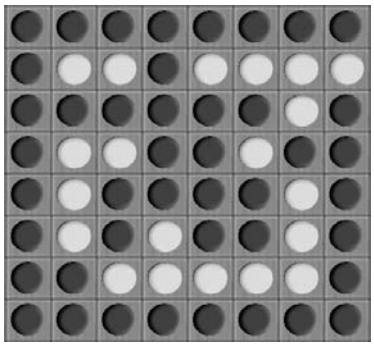
5. HASIL EKSPERIMEN

Hasil eksperimen berisi mengenai percobaan pertandingan antara algoritma Negascout dan MTDF yang telah dibuat melawan game yang telah didownload. Hasil eksperimen tersebut berupa jumlah kemenangan yang diraih, efisiensi waktu, jumlah langkah dalam memenangkan permainan. Berikut penjelasan mengenai pertandingan antara Negascout melawan othello cyclog, yang dijabarkan pada tabel 2.

Tabel 2. Hasil pertandingan Negascout melawan othello cyclog

No	*Negascout*					*Othello Cyclog*	
	Node	*Waktu*	*Langkah*	*Score*	*Pemenang*	*Score*	*Pemenang*
1	141041	1357	33	44	√	20	-
2	199324	1912	34	59	√	5	-
3	191772	1872	32	30	-	34	√
4	112294	1100	34	52	√	11	-
5	154614	1593	31	47	√	17	-
6	143160	1432	32	30	-	34	√
7	176664	1880	32	58	√	6	-
8	167107	1612	33	54	√	10	-
9	171885	1662	32	42	√	22	-
10	166869	1500	35	55	√	9	-
11	152052	1408	33	44	√	20	-
12	165559	1711	30	44	√	20	-

Dari hasil pertandingan pada tabel 2, dapat disimpulkan bahwa dari 12 kali pertandingan, othello cyclog hanya 2 kali memenangkan pertandingan, sedangkan sisanya dimenangkan oleh Negascout. Pada kolom node menandakan banyaknya node yang ditelusuri pada proses pencarian, mulai dari awal hingga akhir permainan. Pada kolom waktu menandakan rata-rata berpikir Negascout dengan kedalaman 6 level pencarian dalam tiap-tiap permainan (dalam satuan millisecond). Pada kolom langkah menandakan banyaknya langkah yang dilakukan dalam tiap-tiap pertandingan. Berikut hasil akhir pertandingan antara Negascout melawan othello cyclog pada pertandingan ke 12, penjelasan dijabarkan pada gambar 10.



Gambar 10. Negascout melawan othello cyclog

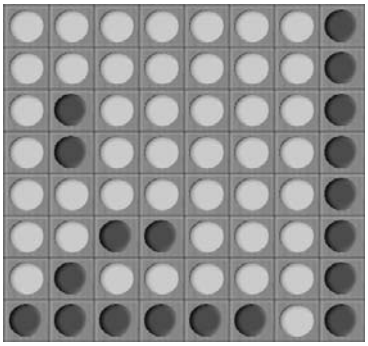
Gambar 10, merupakan hasil akhir pertandingan antara Negascout melawan othello cyclog dimenangkan oleh Negascout, dengan score 44 untuk Negascout dan 20 untuk othello cyclog. Negascout menggunakan disc hitam, dengan menguasai keempat daerah pojok dari papan yang merupakan daerah yang menguntungkan.

Setelah mengetahui hasil eksperimen untuk metode Negascout, selanjutnya akan dijelaskan mengenai hasil eksperimen untuk metode MTDF. Berikut penjelasan mengenai pertandingan antara MTDF melawan othello cyclog, yang dijabarkan pada tabel 3.

Tabel 3. Hasil pertandingan MTDF melawan othello cyclog

No	*MTD(f)*					*Othello Cyclog*	
	Node	*Waktu*	*Langkah*	*Score*	*Pemenang*	*Score*	*Pemenang*
1	124241	1313	35	55	√	9	-
2	125679	1264	34	34	√	30	-
3	144368	647	32	29	-	35	√
4	156899	1135	33	49	√	15	-
5	145679	1356	32	45	√	19	-
6	156569	856	32	34	√	30	-
7	157383	1743	31	43	√	21	-
8	146458	569	35	59	√	5	-
9	134347	678	33	41	√	23	-
10	147674	1125	36	49	√	15	-
11	145348	1168	31	36	√	28	-
12	155756	1757	33	35	√	29	-

Dari hasil pertandingan pada tabel 3 dapat disimpulkan bahwa dari 12 kali pertandingan yang dilakukkan, ternyata othello cyclog hanya 1 kali memenangkan pertandingan, sedangkan sisanya dimenangkan oleh MTDF. Berikut hasil akhir pertandingan antara MTDF melawan othello cyclog pada pertandingan ke 5, penjelasan dijabarkan pada gambar 11.



Gambar 11. MTDF melawan othello cyclog

Gambar 11, merupakan hasil akhir pertandingan antara MTDF melawan othello cyclog yang dimenangkan oleh MTDF, dengan score 45 untuk MTDF dan 19 untuk othello cyclog. MTDF menggunakan disc putih, dengan menguasai satu daerah pojok sedangkan othello cyclog menggunakan disc hitam dengan menguasai tiga daerah pojok. Dapat disimpulkan bahwa dengan menguasai banyak daerah pojok belum tentu kemenangan dapat diraih.

6. PENUTUP

Dari penjelasan sebelumnya yang berisi mengenai metode pencarian minimax dan proses perhitungan untuk fungsi evaluasi, dapat disimpulkan bahwa, Negascout berasumsi langkah pertama yang ditelusuri adalah langkah terbaik dan menganggap bahwa langkah-langkah selanjutnya sebagai langkah terburuk. Jika terdapat langkah yang lebih baik dari langkah pertama maka akan dilakukan proses research.

MTDF yang menggunakan upperbound dan lowerbound bekerja dengan melakukan pemanggilan *alpha beta with memory* secara berulang-ulang. Proses pruning akan banyak dihasilkan dari penggunaan bound ini.

Edge table digunakan untuk menyimpan nilai mobility dengan mengevaluasi satu sisi papan, selanjutnya untuk evaluasi total didapatkan dengan cara mencerminkannya keseluruh sisi-sisi papan. Semakin banyak mobility atau jumlah langkah yang dapat dimainkan oleh pemain pada satu kali kesempatan maka akan semakin bagus, karena semakin banyak pula langkah bagus yang dapat dimainkan.

Potential mobility atau frontier menggunakan liberties untuk mengevaluasi jumlah disc yang berbatasan dengan petak kosong. Semakin banyak frontier maka akan semakin buruk, karena lawan akan mendapat semakin banyak mobility tambahan dan juga akan mengurangi mobility yang didapat.

Penguasaan corner pada permainan othello juga penting, karena kemungkinan besar kemenangan akan diraih dengan menguasai daerah-daerah pojok. Pemberian nilai yang tepat akan mempengaruhi hasil dari perhitungan fungsi evaluasi, jadi perlu diperhatikan.

PUSTAKA

Alexander Reinefeld, T.A. Marsland. (1989). *A Quantitative Analysis of Minimal Window Search*.

Aske Plaat, Jonathan Schaeffer, Wim Pijls and Arie de Bruin. (1994). *A New Paradigm for Minimax Search*.

Aske Plaat, Jonathan Schaeffer, Wim Pijls and Arie de Bruin. (2000). *An Algorithm Faster than Negascout and SSS in Practice*.

Chuong Do, Sanders Chong, Mark Tong, Anthony Hui. (2002). *Othello Report Demosthenes*.

Donald E. Knuth and Ronald W. Moore. (1975). *An analysis of alpha-beta pruning. Artificial Intelligence*.

George C. Stockman. (1979). *A minimax algorithm better than alpha-beta? Artijcial Intelligence*.

Hermann Kaindl, Reza Shams, and Helmut Horacek. (1991). *Minimax search algorithms with and without aspiration windows*.

Johannes Willem Romein. (2001). *Multigame – An Environment for Distributed Game-Tree Search*.

Lim Yew Jin. (2007). *On Forward Pruning in Game-Tree Search*.

Mark Henricus Maria Winands. (2004). *Informed Search in Complex Games*.

Prof.Dr. P.W.C. Akkermans M.A. (1996). *Research RE: Search & Re-Search (Aske Plaat)*.

Qian Liang. (1994). *The Evolution of Mulan: Some Studies in Game-Tree Pruning and Evaluation Functions in the Game of Amazons*.

Richard E. Korf and David W. Chickering. (1994). *Best-first minimax search: Othello results*.

Rion Snow, Keyur Patel, Jared Jacobs. (2002). *Othello: The Moors of Denice*.

T. Anthony Marsland, Alexander Reinefeld. (1992). *Heuristic Search in One and Two Players Games*.

Valavan Manohararajah. (2001). *Parallel Alpha-Beta Search on Shared Memory Multiprocessors*.