Implementasi Algoritma Negascout Untuk Permainan Checkers

Aditya Kurniawan Effendi[1] Rosa Delima[2] Antonius R. C. [3]
aditya.kurniawan.eff@gmail.com rosadelima@ukdw.ac.id anton@ti.ukdw.ac.id

Abstract

Checker is a zero sum game which means if one player is declared win then the other player is declared lose, if one player gets 1 point then the other player gets -1 point. In this type of game every player have full knowledge of other player condition, like every move the other player has, what kind of pawn and the position of other player's pawn.

This game will be implemented with Negascout and compared to Alpha-Beta to see wether it offer better or worse performance than Alpha-Beta. Both algorithm will be given 5 identical board condition to solve and the search depth will be limited to 4, 6 and 8 level.

The result showed Negascout outperformed Alpha-Beta on 86% of the test performed. It searched less node than Alpha-Beta especially with depth 8. The result also showed Negascout found one difference node solution compared to Alpha-Beta with identical heuristic score. Negascout returned identical heuristic score with Alpha-Beta on all test.

Kata Kunci : Negascout, Permainan, Checkers, Kecerdasan Buatan

1. PENDAHULUAN

Seiring dengan berkembangnya teknologi komputer, teknologi perangkat lunak juga berkembang. Salah satu teknologi perangkat lunak yang berkembang adalah aplikasi permainan. Dewasa ini sudah bermunculan berbagai macam jenis aplikasi permainan. Dari permainan klasik seperti kartu, *board games*, sampai permainan-permainan lainnya yang lebih kompleks.

Semua permainan itu diperlukan sebuah kecerdasan yang memungkinkan komputer untuk bermain melawan manusia. Cabang ilmu komputer yang berusaha untuk menirukan kecerdasan manusia adalah kecerdasan buatan atau *Artificial Intelligence* (A.I.). Salah satu permainan yang menggunakan A.I. adalah *checkers*, Pada permainan ini pemain dinyatakan menang jika pemain lain kalah, seandainya pemain membutuhkan 1 poin untuk menang, maka untuk kalah pemain membutuhkan poin -1. Permainan seperti ini disebut juga dengan *zero-sum game* yang berarti kemenangan pemain adalah kekalahan pemain lainnya. Dalam permainan ini setiap pemain dapat mengetahui semua kondisi permainan, seperti jumlah bidak yang dimiliki pemain dan lawan, posisi bidak, dan langkah apa saja yang dimiliki oleh pemain maupun lawan. Oleh karena itu maka

[1] *Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*
[2] *Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*
[3] *Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*

permainan *checkers* ini sangat cocok jika mengimplementasikan algoritma *negascout* untuk penentuan langkah komputer. *Negascout* ditemukan oleh Alexander Reinefeld tahun 1983. Algoritma ini berusaha memotong node-node dengan melihat dahulu node yang akan dicek, apabila node tersebut memiliki nilai yang lebih baik maka akan dilakukan pencarian ulang dengan menggunakan *search window* seperti algoritma *alpha-beta pruning* untuk mengetahui nilai asli dari node tersebut.

2. LANDASAN TEORI

2.1. *Negascout*

Negascout merupakan optimalisasi *minimax* dengan mempersempit ruang pencarian (*minimal search window*), dengan semakin sempitnya selisih nilai *alpha* dan *beta* maka semakin besar kemungkinan terjadinya pemotongan pencarian. *Negascout* memiliki dasar bahwa langkah-langkah setelah langkah pertama akan menghasilkan pemotongan, maka mengevaluasi semua langkah adalah sia-sia. *Negascout* akan mengecek dengan *null window* terlebih dahulu yang dinotasikan dengan m dan n dimana m adalah batas atas dan n adalah batas bawah, *null window* memiliki batas atas dan bawah yang bersilisih satu. Ketika sebuah node memiliki nilai yang lebih tinggi dari m maka akan dilakukan pencarian ulang dengan menggunakan *window* yang lebih besar untuk mengetahui nilai yang terbaik.

```
 1 FUNCTION negascout (p: POSITION; alpha, beta, depth: INTEGER) : INTEGER;
 2 VAR i,t,m,n: INTEGER;
 3 BEGIN
 4     IF depth = d THEN RETURN (evaluate(p))
 5     ELSE
 6     BEGIN m := -∞;
 7           n := beta;
 8           FOR i := 1 TO b DO
 9           BEGIN t := -negascout (p.i, -n, -max(alpha,m), depth+1);
10                 IF t > m THEN
11                    IF (n = beta) OR (depth >= d-2)
12                    THEN m := t
13                    ELSE m := -negascout (p.i, -beta, -t, depth+1);
14                 IF m >= beta THEN RETURN (m);
15                 n := max (alpha,m) +1;
16           END;
17           RETURN (m);
18     END;
19 END;
```

Gambar 1. Pseudocode*Negascout*
Dikutip dari: Reinefeld, A. (1983)

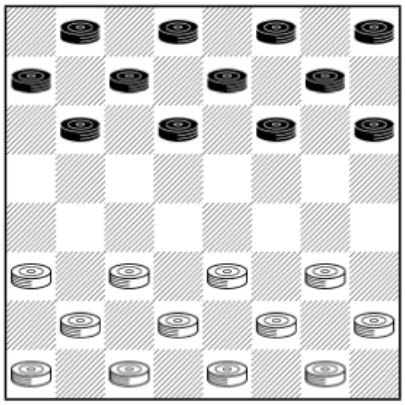
Gambar 1. merupakan *pseudocode* untuk algoritma Negascout, berdasarkan algoritma diatas dapat dijelaskan cara kerja algoritma *negascout* sebagai berikut, *statement* pertama sama seperti *alpha-beta*, yaitu jika posisi p adalah *node leaf* maka *negascout* akan mengembalikan nilai fungsi evaluasi (baris 4). Selain itu variable m dan n diinisialisasi dengan -∞ dan beta (baris 6 dan 7). Kemudian *negascout* akan melakukan "scout" pada node anak dari p dari kiri ke kanan. Node anak paling kiri akan dicari dengan menggunakan interval (-beta,-alpha) dan anak lainnya dicari dengan *zero-width window* (-m-1, -m) yang sudah diisi pada baris 15 sesudah melakukan pencarian anak paling kanan, karena *null window* ini tidak memiliki elemen, maka pencarian pasti akan gagal. Arah dari kegagalan ini menunjukkan apakah node tersebut dapat dipotong atau tidak.

Jika *null window* gagal karena t > m (baris 10), *negascout* harus mengecek kembali node tersebut dengan *search window* yang lebih lebar untuk mengetahui nilai aslinya. Hanya terdapat dua kasus dimana tidak diperlukan pencarian ulang yaitu, ketika n=beta (baris 11), dan *negascout's "fail-soft refinement"*selalu mengembalikan nilai minimax yang benar pada dua level terbawah. Pada kasus lainnya pencarian ulang harus dilakukan dengan menggunakan *search window* baru yaitu (-beta,-t) (baris 13).

Kondisi untuk pemotongan (baris 14) sama seperti *alpha-beta* : jika m ≥ beta maka semua node anak lain dapat diabaikan.

2.2. Checkers

Checkers atau *draught* adalah *board games* yang pertama kali dimainkan sekitar 3000 tahun sebelum masehi, pada saat itu papan yang dan jumlah bidak yang digunakan berbeda dengan *checkers* yang kita kenal sekarang.



Gambar 2. Papan Checker
Dikutip dari: Reinefeld, F. (2011)

Checkers yang akan penulis gunakan untuk tugas akhir ini merupakan *English checkers* dimana papan yang digunakan berukuran 8x8 (memiliki jumlah total 64 kotak) dengan 12 bidak pada setiap sisi (total 24 bidak).

Berdasarkan buku How to Win at Checkers, setiap pemain menyusun 12 bidaknya pada papan permainan pada kotak gelap di 3 baris paling dekat dengan pemain. Baris paling dekat dengan pemain disebut dengan *crownhead* atau *king's row*. Pemain dengan warna bidak gelap bergerak dahulu.

Ada 2 jenis gerakan dalam permainan ini yaitu :

1. *Simple move* yaitu gerakan diagonal satu langkah ke kotak kosong di depan.
2. *Jump move* yaitu gerakan diagonal melompati satu bidak lawan ke kotak kosong di depan. Bidak yang dilompati akan dihilangkan dari papan. Apabila terdapat *jump move* setelah gerakan *jump move*, pemain harus melakukan gerakan tersebut sampai tidak terdapat *jump move* lagi.

Ketika pemain memiliki kesempatan untuk melakukan *jump move*, pemain harus melakukan gerakan itu. Apabila pemain memiliki beberapa gerakan *jump move*, pemain bebas memilih *jump move* mana yang akan diambil.

Checkers memiliki dua jenis bidak, yaitu :

1. Bidak manusia. Bidak manusia hanya dapat bergerak ke depan. Ketika bidak manusia mencapai *king's row*, bidak tersebut akan berubah menjadi bidak raja.
2. Bidak raja. Bidak raja dapat bergerak ke depan dan ke belakang.

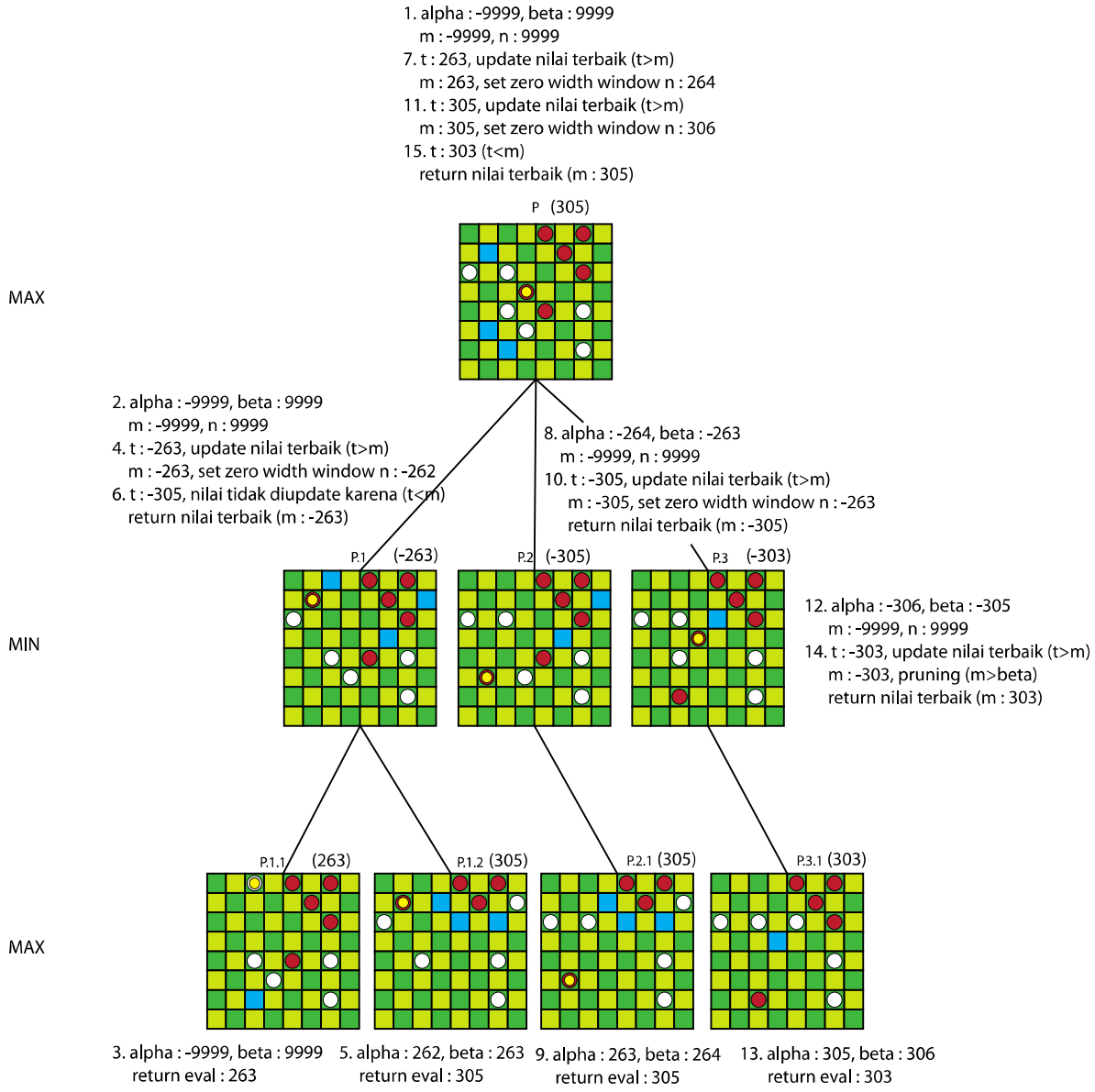
Permainan berakhir apabila salah satu pemain tidak dapat melakukan pergerakan atau kehilangan semua bidaknya.

2.3. Fungsi Evaluasi

Dalam mengambil keputusan untuk menentukan langkah yang diambil, komputer membutuhkan sebuah fungsi untuk menilai untung atau tidaknya suatu langkah. Setiap permainan memiliki fungsi evaluasi yang berbeda-beda. Dalam penelitian ini fungsi evaluasi akan dipengaruhi oleh beberapa faktor, yaitu : jumlah bidak, jumlah langkah yang tersedia, jumlah lompatan yang tersedia dan jumlah bidak di pinggir papan.

3. HASIL DAN PEMBAHASAN

Pada Gambar 3 , dapat dilihat algoritma *negascout* dalam permainan *checkers.* Penelitian ini akan dilakukan 2 pengujian. Pengujian pertama dilakukan untuk melihat performa *Negascout* dibandingkan dengan *Alpha-Beta Pruning*. Pengujian kedua dilakukan untuk melihat performa *Negascout* untuk permainan *Checkers.*

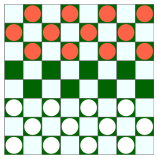


Gambar 3. *Negascout* dalam Permainan *Checkers*

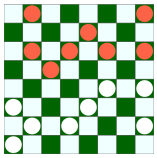
Pengujian pertama akan dilakukan dengan memberikan 5 kondisi papan yang berbeda dengan 3 kedalaman yang berbeda. Setiap kondisi papan akan dilakukan pencarian solusi dengan menggunakan kedua algoritma sebanyak 3 kali. Dari pengujian akan dicatat waktu (dalam ms), node solusi, jumlah node dan nilai heuristiknya.

Kondisi papan yang diberikan untuk pengujian ini adalah sebagai berikut :

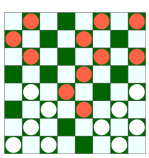
1. Kondisi papan pertama



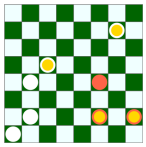
2. Kondisi papan kedua



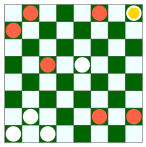
3. Kondisi papan ketiga



4. Kondisi papan keempat



5. Kondisi papan kelima



Pengujian kedua dilakukan untuk mengetahui performa *Negascout* dalam permainan *checkers*. Pada pengujian ini *user* memainkan bidak merah dan komputer (*Negascout*) memainkan bidak putih. Pengujian dilakukan dengan menggunakan *depth* 4 dan 6, masing-masing *depth* - dilakukan pengujian sebanyak 5 kali.

5. EVALUASI

5.1. Hasil Pengujian Perbandingan *Negascout* dengan *Alpha-Beta Pruning*

Pengujian dilakukan untuk kedalaman 4, 6, dan 8. Masing-masing hasil pengujian dapat dilihat pada tabel 1, tabel 2, dan tabel 3.

Tabel 1.
Perbandingan *Negascout* dan *Alpha-Beta* Dengan *Depth* 4

Kondisi	No.	Negascout				Alpha-Beta Pruning			
		Jumlah Node	Waktu (ms)	Node Solusi	Nilai	Jumlah Node	Waktu (ms)	Node Solusi	Nilai
1	1.	281	1.004	P.1.2.1.1	679	442	1.736	P.1.2.1.1	679
	2.	281	996	P.1.2.1.1	679	442	1.637	P.1.2.1.1	679
	3.	281	1.002	P.1.2.1.1	679	442	1.734	P.1.2.1.1	679
2	1.	275	881	P.4.2.1.1	436	262	872	P.4.2.1.1	436
	2.	275	873	P.4.2.1.1	436	262	827	P.4.2.1.1	436
	3.	275	854	P.4.2.1.1	436	262	844	P.4.2.1.1	436
3	1.	347	1.302	P.5.4.1.1	662	369	1.418	P.5.4.1.1	662
	2.	347	1.288	P.5.4.1.1	662	369	1.356	P.5.4.1.1	662
	3.	347	1.247	P.5.4.1.1	662	369	1.369	P.5.4.1.1	662
4	1.	396	791	P.2.1.2.1	232	431	932	P.2.1.2.1	232
	2.	396	811	P.2.1.2.1	232	431	889	P.2.1.2.1	232
	3.	396	818	P.2.1.2.1	232	431	893	P.2.1.2.1	232
5	1.	765	1.695	P.9.1.1.1	557	851	1.917	P.9.1.1.1	557
	2.	765	1.749	P.9.1.1.1	557	851	1.929	P.9.1.1.1	557
	3.	765	1.695	P.9.1.1.1	557	851	1.944	P.9.1.1.1	557

Pada tabel 1 dapat dilihat *Negascout* menghasilkan pohon pencarian lebih sedikit dibandingkan dengan *Alpha-Beta Pruning* kecuali pada kondisi papan nomor 2. Waktu yang dibutuhkan untuk menemukan solusi berbanding lurus dengan jumlah node yang dihasilkan.

Tabel 2.
Perbandingan *Negascout* dan *Alpha-Beta* Dengan *Depth* 6

Kondisi	No.	Negascout				Alpha-Beta Pruning			
		Jumlah Node	Waktu (ms)	Node Solusi	Nilai	Jumlah Node	Waktu (ms)	Node Solusi	Nilai
1	1.	2.154	8.082	P.1.2.1.1.2.1	679	3.192	12.724	P.1.2.1.1.2.1	679
	2.	2.154	8.159	P.1.2.1.1.2.1	679	3.192	12.681	P.1.2.1.1.2.1	679
	3.	2.154	7.871	P.1.2.1.1.2.1	679	3.192	12.714	P.1.2.1.1.2.1	679
2	1.	1.297	3.966	P.1.2.1.1.2.1	436	1.853	5.835	P.1.2.1.1.2.1	436
	2.	1.297	3.943	P.1.2.1.1.2.1	436	1.853	5.800	P.1.2.1.1.2.1	436
	3.	1.297	3.939	P.1.2.1.1.2.1	436	1.853	5.813	P.1.2.1.1.2.1	436
3	1.	1.767	6.446	P.4.5.1.2.5.1	562	2.064	7.548	P.4.5.1.2.5.1	562
	2.	1.767	6.356	P.4.5.1.2.5.1	562	2.064	7.573	P.4.5.1.2.5.1	562
	3.	1.767	6.451	P.4.5.1.2.5.1	562	2.064	7.601	P.4.5.1.2.5.1	562
4	1.	1.947	3.931	P.1.7.1.10.1.1	310	1.820	3.293	P.1.7.1.10.1.1	310
	2.	1.947	3.927	P.1.7.1.10.1.1	310	1.820	3.284	P.1.7.1.10.1.1	310
	3.	1.947	3.984	P.1.7.1.10.1.1	310	1.820	3.303	P.1.7.1.10.1.1	310
5	1.	5.142	11.913	P.8.2.4.2.1.1	515	6.132	13.765	P.8.2.7.2.1.1	515
	2.	5.142	11.419	P.8.2.4.2.1.1	515	6.132	13.758	P.8.2.7.2.1.1	515
	3.	5.142	11.522	P.8.2.4.2.1.1	515	6.132	13.724	P.8.2.7.2.1.1	515

Pengujian kedua dengan *depth* 6 juga menunjukkan pohon pencarian dan waktu yang dihasilkan *Negascout* lebih kecil dibandingkan dengan *Alpha-Beta Pruning*. Dari 5 kondisi yang diujikan *Negascout* menghasilkan node solusi yang sama dengan *Alpha-Beta Pruning* pada kondisi 1 sampai 4. *Negascout* menghasilkan solusi yang berbeda pada kondisi 5 dengan nilai heuristik yang sama. Pada kondisi nomor 4 *Negascout* menghasilkan jumlah node yang lebih banyak daripada *Alpha-Beta Pruning.*

Pengujian ketiga membatasi pohon pencarian pada kedalaman 8. Hasil pengujian memperlihatkan *Negascout* menghasilkan jumlah node yang lebih sedikit pada semua kondisi pengujian dengan solusi yang sama dengan *Alpha-Beta Pruning*

Tabel 3.
Perbandingan *Negascout* dan *Alpha-Beta* Dengan *Depth* 8

Kondisi	No.	Negascout				Alpha-Beta Pruning			
		Jumlah Node	Waktu (ms)	Node Solusi	Nilai	Jumlah Node	Waktu (ms)	Node Solusi	Nilai
1	1.	10.964	42.355	P.1.2.1.2.2.2.1.1	619	24.288	93.497	P.1.2.1.2.2.2.1.1	619
	2.	10.964	42.622	P.1.2.1.2.2.2.1.1	619	24.288	94.252	P.1.2.1.2.2.2.1.1	619
	3.	10.964	41.828	P.1.2.1.2.2.2.1.1	619	24.288	94.210	P.1.2.1.2.2.2.1.1	619
2	1.	8.309	25.537	P.1.2.1.1.2.1.3.1	436	13.064	40.150	P.1.2.1.1.2.1.3.1	436
	2.	8.309	24.123	P.1.2.1.1.2.1.3.1	436	13.064	39.783	P.1.2.1.1.2.1.3.1	436
	3.	8.309	25.541	P.1.2.1.1.2.1.3.1	436	13.064	41.078	P.1.2.1.1.2.1.3.1	436
3	1.	8.965	32.410	P.2.1.5.1.1.3.1.1	594	10.083	36.773	P.2.1.5.1.1.3.1.1	594
	2.	8.965	32.576	P.2.1.5.1.1.3.1.1	594	10.083	36.575	P.2.1.5.1.1.3.1.1	594
	3.	8.965	32.702	P.2.1.5.1.1.3.1.1	594	10.083	37.246	P.2.1.5.1.1.3.1.1	594
4	1.	55.831	118.111	P.2.1.2.1.1.1.1.1	334	64.181	145.078	P.2.1.2.1.1.1.1.1	334
	2.	55.831	117.614	P.2.1.2.1.1.1.1.1	334	64.181	146.440	P.2.1.2.1.1.1.1.1	334
	3.	55.831	117.881	P.2.1.2.1.1.1.1.1	334	64.181	147.011	P.2.1.2.1.1.1.1.1	334
5	1.	33.825	76.670	P.4.2.10.4.8.5.5.1	579	42.725	99.003	P.4.2.10.4.8.5.5.1	579
	2.	33.825	75.907	P.4.2.10.4.8.5.5.1	579	42.725	100.045	P.4.2.10.4.8.5.5.1	579
	3.	33.825	75.969	P.4.2.10.4.8.5.5.1	579	42.725	98.924	P.4.2.10.4.8.5.5.1	579

5.2. Hasil Pengujian *User* Melawan *Negascout* dan *Alpha-Beta Pruning*

Pengujian ini dilakukan oleh 2 *user* melawan komputer dengan menggunakan algoritma *Negascout* dan *Alpha-Beta Pruning* dengan *depth* 4, 6 dan 8 sebanyak masing-masing 10 kali permainan. Pengalaman *user* dalam pengujian ini satu *user* masih pemula dan satu *user* memiliki pengalaman bermain selama 10 tahun. Pemain dengan pengalaman bermain lebih lama akan memainkan pengujian nomor 1 sampai 5 dan pemain pemula memainkan pengujian nomor 6 sampai dengan 10.

Dari pengujian yang dilakukan komputer menggunakan *Negascout* memiliki persentase kemenangan 60% pada *depth* 4 dan 80% pada *depth* 6. Pada *depth* 8 pemain menghentikan permainan karena waktu tunggu yang terlalu lama (pencarian solusi komputer lebih dari 1 menit) sehingga hasil pengujian *depth* 8 tidak dimasukkan. Dengan menggunakan *Alpha-Beta Pruning* komputer miliki persentase kemenangan 50% dan 70% pada *depth* 4 dan 6.

Tabel 4.
Hasil Pengujian *User* Melawan *Negascout*

Depth	No.	Pemenang			
		Negascout	User	Alpha-Beta	User
4	1.	✔	-	-	✔
	2.	✔	-	✔	-
	3.	✔	-	✔	-
	4.	-	✔	-	✔
	5.	-	✔	-	✔
	6.	✔	-	✔	-
	7.	✔	-	✔	-
	8.	-	✔	-	✔
	9.	-	✔	-	✔
	10.	✔		✔	
6	1.	✔	-	✔	-
	2.	-	✔	-	✔
	3.	✔	-	-	✔
	4.	-	✔	-	✔
	5.	✔	-	✔	-
	6.	✔	-	✔	-
	7.	✔	-	✔	-
	8.	✔	-	✔	-
	9.	✔	-	✔	-
	10.	✔	-	✔	-

6. KESIMPULAN DAN SARAN

6.1. Kesimpulan

Berdasarkan implementasi serta setalah melakukan analisa algoritma *Negascout* pada permainan *Checkers* didapatkan kesimpulan sebagai berikut:

a) Dari semua pengujian yang dilakukan 86% hasil pengujian menunjukkan penerapan algoritma *Negascout* pada permainan *Checkers* menghasilkan jumlah node yang lebih sedikit dan waktu pencarian yang lebih singkat dibandingkan dengan algoritma *Alpha-Beta Pruning*terutama pada *depth* 8.

b) Melihat dari nilai heuristik yang dihasilkan algoritma *Negascout* pada pengujian yang dilakukan algoritma ini menghasilkan solusi yang sama dengan algoritma *Alpha-Beta Pruning*

c) Apabila algoritma *Negascout* melakukan pencarian ulang pada level-level awal, maka algoritma ini akan mengecek ulang semua anak node dari node yang dilakukan pencarian ulang. Apabila pencarian ulang terjadi pada level atas maka *Negascout* akan menghasilkan node yang lebih banyak dibandingkan dengan pencarian ulang pada level-level bawah.

d) Berdasarkan pengujian komputer lawan *user* yang dilakukan dapat dilihat penerapan algoritma *Negascout* pada permainan *Checkers* menghasilkan kecerdasan komputer yang hampir sama dengan algoritma *Alpha-Beta Pruning* dengan waktu respon yang lebih cepat.

e) Berdasarkan pengujian komputer lawan *user,* komputer memiliki persentase kemenangan lebih besar dengan *depth* 6 dibandingkan dengan *depth* 4, sedangkan untuk *depth* 8 komputer memerlukan waktu terlalu lama sehingga tidak memungkinkan untuk digunakan ketika melawan *user*.

6.2. Saran

Berdasarkan beberapa kelemahan yang dimiliki oleh sistem, penulis memberikan usulan sebagai berikut :

a) Fungsi evaluasi yang digunakan belum cukup bagus, hal ini dapat dilihat dari pengujian kedua dimana komputer masih dapat dikalahkan oleh *user*. Untuk itu diperlukan fungsi baru yang dapat merepresentasikan kondisi papan lebih akurat untuk menghasilkan A.I. yang lebih baik. Fungsi evaluasi yang digunakan menghitung faktor-faktor seperti : jumlah bidak, jenis bidak, lokasi bidak dan jumlah langkah yang tersedia. Mungkin dengan ditambahkan faktor lain seperti awal permainan atau akhir permainan dapat meningkatkan representasi kondisi papan.

b) Dari 13% hasil pengujian menunjukkan *Negascout* menghasilkan jumlah node yang lebih banyak dari *Alpha-Beta Pruning*. Hal tersebut dikarenakan *Negascout* melakukan terlalu banyak pencarian ulang yang disebabkan jeleknya *move ordering* yang ada. Untuk menghasilkan performa *Negascout* yang lebih baik diperlukan penambahan *move ordering* yang baik.

Daftar Pustaka

Ayuningtyas, N. (2008). *Algoritma Minimax Dalam Permainan Checkers.* Diakses 2 Januari 2012, dari http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2007-2008/Makalah2008/MakalahIF2251-2008-031.pdf

Ertel, W. (2011). *Introduction To Artificial Intelligence.* Germany: Springer.

Gunawan, Kristian, Y., & Andika, H. (2009). *Game Playing Untuk Othello Dengan Menggunakan Algoritma Negascout dan MTDF*. Diakses 2 Januari 2012, dari http://journal.uii.ac.id/index.php/Snati/article/viewFile/1278/1088

Millington, I. (2006). *Artificial Intelligence For Games*. San Francisco : Morgan Kaufmann.

Reinefeld, A. (1983).*An improvement to the scout tree algorithm.* Diakses 2 Januari 2012, dari http://www.top-5000.nl/ps/An%20improvement%20to%20the%20scout%20tree%20search%20algorithm.pdf

Reinfeld, F. (2011). *How To Win at Checkers.* Diakses 2 Januari 2012, dari http://www.bobnewell.net/checkers/howtowin.pdf

Schaeffer, J. (2009). *One Jump Ahead.* New York : Springer.

Schaeffer, J., Lake, R. (1996). *Solving the Game of Checkers.* Diakses 2 Januari 2012, dari http://library.msri.org/books/Book29/files/schaeffer.pdf

Schaeffer, J., Lake, R., Lu, P., & Bryant, M. (n.d). *Chinook : The World Man-Machine Checkers Champion.* Diakses 2 Januari 2012, dari http://webdocs.cs.ualberta.ca/~jonathan/publications/ai_publications/aimag96.pdf