IMPLEMENTASI ALGORITMA *UPPER CONFIDENCE BOUNDS FOR TREES* (UCT) UNTUK PENGAMBILAN KEPUTUSAN PADA PERMAINAN DAMDAMAN

Hilmi Abdul Aziz

[1]Teknik Informatika – Universitas Komputer Indonesia
Jl. Dipatiukur 112-114 Bandung
E-mail : hilmi.a.aziz@gmail.com[1]

ABSTRAK

Permainan tradisional merupakan jenis permainan yang mengandung nilai-nilai budaya pada hakikatnya merupakan warisan leluhur yang harus dilestarikan keberadaannya. Ada permainan yang sifatnya bertanding dan ada juga yang diutamakan utamakan untuk mengisi waktu luang sebagai bentuk rekreasi. Salah satu permainan tradisional dengan sifat kompetisi 1 lawan 1 yang banyak dimainkan adalah Damdaman atau kadang disebut Basbasan. Dengan berkembangnya teknologi, permainan tradisional mulai ditinggalkan karena kurangnya adaptasi dari permainan-permainan tersebut. Maka penelitian ini akan mencoba mengadaptasi permainan Damdaman yang juga dapat memberi contoh pengambilan langkah yang baik pada pemainnya.

Penggunaan metode-metode kecerdasan buatan telah banyak digunakan pada jenis permainan papan seperti Damdaman. Selain dapat menguji kemampuan komputer dalam menyelesaikan masalah, juga dapat digunakan sebagai sarana hiburan. Metode pencarian yang diterapkan pada jenis permainan ini akan mencari langkah-langkah yang memberi hasil yang berpotensi kemenangan. Metode pencarian langkah yang digunakan pada penelitian ini adalah *Upper Confidence Bounds For Trees* (UCT).

Dari hasil penelitian ini dapat diketahui bahwa algoritma UCT dapat diterapkan pada permainan Damdaman dengan persentase kemenangan dihasilkan nilai sebesar 53%, nilai ini dianggap cukup baik. Serta performansi waktu pencarian 8011 *milisecond*/langkah dan performansi penggunaan memori sebesar 11.338 *bytes*/langkah.

Kata kunci : Permainan Papan, Permainan Damdaman, *Upper Confidence Bounds For Trees, Artificial Intelligence,Tree Search.*

1. PENDAHULUAN

Permainan tradisional merupakan jenis permainan yang mengandung nilai-nilai budaya pada hakikatnya merupakan warisan leluhur yang harus dilestarikan keberadaannya. Ada permainan yang sifatnya bertanding dan ada juga yang diutamakan utamakan untuk mengisi waktu luang sebagai bentuk rekreasi. Salah satu permainan tradisional yang banyak dimainkan adalah Damdaman atau kadang disebut Basbasan. Damdaman merupakan salah satu bentuk permainan papan bersifat kompetitif yang dimainkan oleh dua pemain, dengan jumlah bidak masing-masing pemain berjumlah 16 buah. Tujuan akhir dari permainan ini adalah untuk menghabiskan bidak milik lawan [1].

Dengan berkembangnya teknologi, permainan tradisional mulai ditinggalkan karena kurangnya adaptasi dari permainan-permainan tersebut. Serta berkurangnya orang yang mengerti dan mampu memainkannya, maka penelitian ini akan mencoba mengadaptasi permainan Damdaman yang juga dapat memberi contoh pengambilan langkah yang baik pada pemainnya.

Beberapa penelitian sebelumnya telah menggunakan algoritma *Minimax* dan *Negamax* untuk mencari keputusan optimal pada permainan Damdaman [2]. Algoritma yang pernah digunakan ialah metode *Negascout*, yang merupakan varian dari metode *Minimax*, kedua metode ini belum menunjukan hasil yang memuaskan [3]. Hasil pencarian juga kurang memuaskan/dinilai rendah pada algoritma *Negamax* dengan tingkat kemenangan 40% [4]. Penelitian-penelitian tersebut masih menghasilkan nilai kemenangan yang belum memuaskan, maka penelitian-penelitian tersebut menyarankan penggunaan algoritma heuristik dan meningkatkan kedalaman pohon pencarian untuk dapat mengembangkan kecerdasan buatan dalam permainan Damdaman.

Pada permainan papan sejenis, salah satu algoritma yang sering digunakan dan menghasilkan nilai kemenangan yang baik ialah algoritma *Monte Carlo Tree Search* (MCTS). Algoritma ini digunakan karena kelebihannya yaitu anytime dengan arti dapat dioperasikan dan dihentikan kapan saja namun tetap dapat memberikan hasil yang sesuai, memerlukan sedikit pengetahuan tentang masalah yang dihadapi, dan telah berhasil pada masalah yang metode lain telah mengalami

kegagalan [5]. Serta penggunaan simulasi acak pada pencarian yang juga mempengaruhi pada kecepatan algoritma dalam mencari solusi tanpa harus mengevaluasi setiap kemungkinan yang ada [6].

Algoritma ini sudah diterapkan dan memberi hasil yang menjanjikan pada berbagai permainan papan 1 lawan 1, diantaranya *Go* dengan tingkat kemenangan 75% melawan pemain *expert*, dan 95% melawan pemain biasa, serta pada permainan *Checkers* [7] .Sedangkan algoritma UCT ialah varian algoritma MCTS yang paling populer digunakan. UCT menggabungkan *algoritma Upper Confidence Bound* ke dalam algoritma MCTS untuk mengoptimalkan proses pemilihan solusi pada pencarian MCTS [8].

Berdasarkan latar belakang tersebut, diharapkan penelitian ini dapat menunjukan hasil performansi algoritma UCT pada pengambilan langkah permainan Damdaman.

2. ISI PENELITIAN

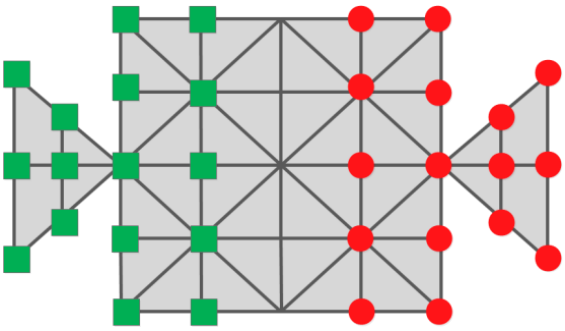
Pada bagian ini akan dijelaskan deskripsi tentang permainan Damdaman, Metode yang digunakan yait algoritma UCT dan implementasinya pada permainan Damdaman

2.1 Permainan Damdaman

Permainan damdaman termasuk permainan tradisional, permainan ini dimainkan oleh dua pemain, permainan ini dimainkan di papan atau karton yang telah digambar. Dalam permainan ini terdapat 16 bidak untuk setiap pemain.

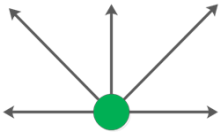
Untuk memulai permainan dam daman, langkah-langkahnya sebagai berikut:

a. Pemain harus berjumlah 2 orang
b. Pemain harus mencari batu/kerikil untuk bidaknya tetapi setiap pemain tidak boleh sama
c. Jumlah bidak untuk setiap pemain ada 16 buah
d. Gambar papan permainan dam daman, seperti pada gambar 1
e. Susun setiap batu/kerikil tadi pada papan permainan seperti pada gambar 1
f. Lakukan suit untuk menentukan pemain giliran pertama.
g. Permainan berakhir apabila salah satu pemain sudah kehabisan bidak



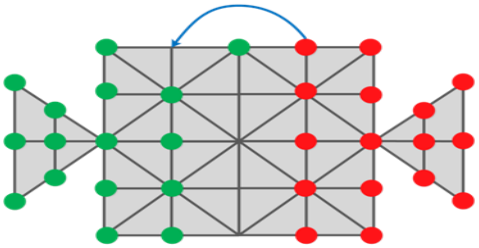
Gambar 1. Bentuk papan permainan

Langkah bidak pada permainan Damdaman terdiri dari dua langkah yang berbeda, yaitu langkah bidak biasa dan langkah bidak raja/satria. Bidak biasa hanya bisa melakukan langkah ke samping, depan dan diagonal. Sedangkan bidak raja bisa bergerak bebas. Untuk bisa menjadi bidak raja harus melewati ujung yang berbentuk segitiga di daerah lawan.



Gambar 2 Contoh langkah bidak normal

Sedangkan untuk memakan bidak lawan, ialah jika terdapat koordinat kosong dibelakang bidak lawan. Seperti terlihat pada gambar 3.



Gambar 3 Gerak memakan bidak lawan

Bidak melewati ujung dari daerah lawan, maka bidak tersebut akan berstatus sakti. Bidak yang sakti bisa bebas bergerak lebih dari satu langkah ke depan, ke belakang, ke samping ataupun bergerak diagonal.

2.2 *Monte Carlo Tree Search*

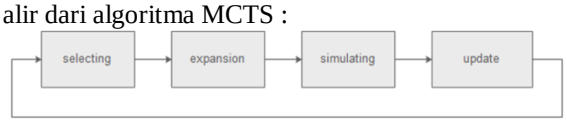
Algoritma *Monte Carlo Tree Search* merupakan algoritma pencarian *heuristik Best First Search* yang umum digunakan sebagai metode pemilihan langkah optimal. Metode ini menggunakan pencarian dalam bentuk pohon pencarian dan juga menggunakan simulasi kemungkinan yang dipilih secara acak dengan metode *Monte Carlo* [13]. MCTS sudah diterapkan dan memberi hasil yang menjanjikan pada berbagai permainan papan 1 lawan 1, diantaranya *Go*, *Othelo*, *connect4* dan *Checkers*.

Proses dalam pengambilan keputusan oleh algoritma MCTS yang memiliki empat tahap yaitu:

a. *Selection*, algoritma akan membuat pohon pencarian berdasarkan langkah yang valid dalam permainan. Dari node pohon pencarian yang ada sementara, Selanjutnya akan dipilih node mana yang merupakan langkah terbaik dengan metode *Best-First Search*.
b. *Expanding the Tree*, Langkah ini diambil ketika menemui keadaan/state yang belum diketahui pada pohon pencarian/jika simulasi belum menemukan nilai *node level* selanjutnya, Maka akan dibuat node baru dari *state* tersebut. *Node-node* baru tersebut adalah kemungkinan langkah bidak yang dapat digunakan.

c. *Simulation*, melakukan simulasi langkah permainan secara acak dan sesuai dengan aturan permainan untuk mencari langkah optimal. Setiap langkah akan diberi aturan bobot agar proses ini akan cenderung kepada langkah yang terbaik.
d. *Update*, setelah sampai pada akhir dari simulasi permainan, proses ini akan memperbaharui nilai dan jumlah kunjungan dari setiap langkah yang dilewati kepada *root*. Kembali ke langkah satu jika *state* akhir pemainan belum tercapai [14]

Akan dilakukan perulangan langkah hingga proses pemilihan langkah dengan algoritma UCT mencapai state akhir dari permainan. Berikut bagan alir dari algoritma MCTS :



Gambar 4 Alur metode MCTS

2.3 *Algoritma Upper Confidence Bound to Tree* (UCT)

Merupakan algoritma varian dari *Monte Carlo Tree Search*, pada algoritma ini MCTS digabungkan dengan algoritma *Upper Confidence Bound* (UCB) pada metode penyeleksian *node* pohon pencariannya. UCB merupakan salah satu algoritma yang digunakan untuk permasalahan yang memiliki banyak pilihan solusi,dan tiap solusi yang diambil memiliki reward yang berbeda. Maka algoritma ini tujuannya ialah mendapatkan *reward* semaksimal mungkin dari setiap langkah/pemilihan solusi yang diambil [15].

Pada Algoritma UCT, setiap node yang telah diketahui akan disimpan *cost*-nya, lalu algoritma akan membuat sampel acak namun berdasar pada aturan permainan, jika jumlah sampel sedikit, maka kemungkinan rata-rata pada sampel akan kurang akurat. Karena itu UCT akan melakukan pemilihan node secara optimistik, UCT akan memilih langkah yang memiliki nilai *confidence* tertinggi (*Upper Confidence Bound*) yaitu nilai kemenangan tertinggi dari sampel yang diketahui [8]. Pada proses pemilihan ini akan digunakan algoritma *Upper Confidence Bound to Tree* sebagai aturan pemilihan. Rumusnya ialah sebagai berikut:

$$\frac{Q(v')}{N(v')} + \sqrt{\frac{\ln\ (N\ (v))}{5 \times N(v')}} = UCT \qquad (1)$$

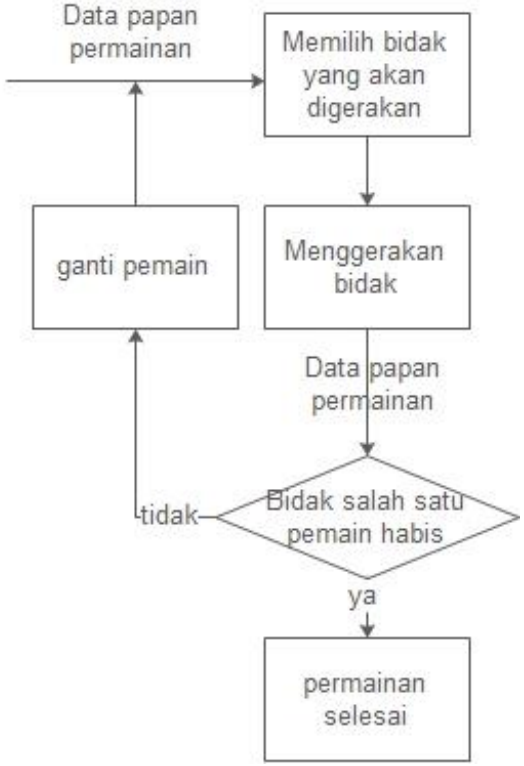
Dengan keterangan notasi sebagai berikut:
Q = nilai kemenangan
N = jumlah visit
v = node parent
v' = node anak

Q(v') didapatkan dari evaluasi sebuah *node*, jika *node* menguntungkan berdasar aturan permainan maka *node* akan diberi nilai 1, nilai akan bertambah 1 jika anak dari *node* tersebut juga merupakan langkah menguntungkan, sehingga semakin tinggi nilai ini, maka akan semakin sering *node* ini diprioritaskan dalalm pencarian. Nilai *N(v')* menunjukan berapa kali node tersebut pernah dilewati untuk dievaluasi oleh algoritma UCT, sedangkan *N(v)* adalah jumlah visit dari parent *node*-nya. Kedua nilai tersebut akan bertambah jika suatu *child node* dilewati/dievaluasi.

Algoritma UCT akan mencari nilai UCT tertinggi sebagai solusi pencariannya. Dengan mencari nilai UCT tertinggi, dari *node* anak, maka didapat *node* yang akan dijadikan langkah optimal untuk pemain komputer.

2.4 Analisis Permainan

Secara umum alur permainan pada aplikasi yang akan dibuat akan terlihat seperti gambar berikut. Pemain akan tersu berganti giliran hingga didapatkan keadaan yang menunjukan statu kemenangan salah satu pemain.

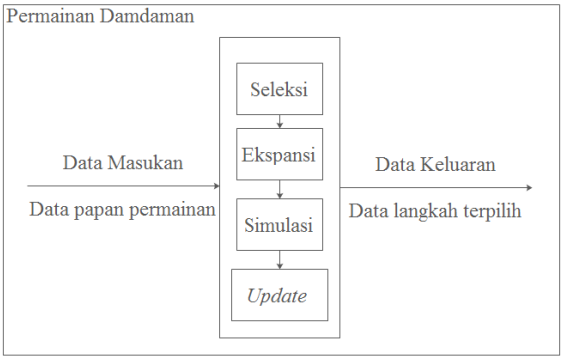


Gambar 5 Alur permainan secara umum

Pada pemain manusia pengambilan keputusan pada proses pemilihan bidak yang digerakan akan dilakukan secara natural atau spontan dengan mengandalkan logika berdasarkan pada aturan permainan. Untuk pemain komputer, pengambilan keputusan tersebut haruslah menggunakan proses pencarian terlebih dahulu. Layaknya pada pemain manusia proses pengambilan keputusan ini juga harus menggunakan aturan permainan sebagai acuan.

Maka pada proses inilah yang akan diimplementasikan metode pencarian langkah optimal, pada kasus ini ialah dengan mengimplementasikan algoritma UCT. Secara umum implementasi algoritma UCT pada permainan damdaman ini ialah seperti terlihat pada gambar 6.

.



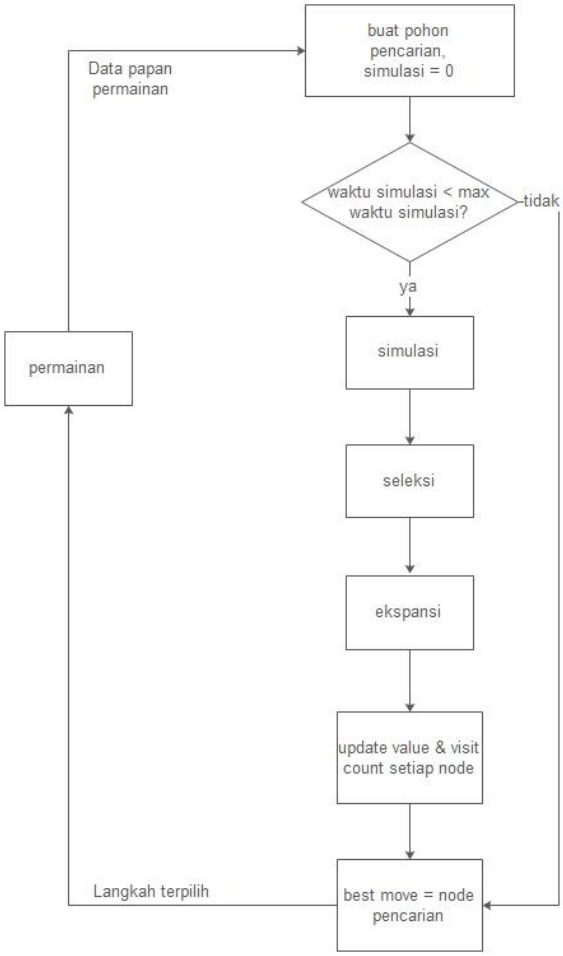
Gambar 6 Alur pencarian UCT pada permainan

Dengan mengambil data dari papan permainan berupa posisi bidak dan langkah yang mungkin diambil setiap bidak dari papan permainan, lalu algoritma UCT akan memproses data tersebut dengan membuat pohon pencarian yang setiap daun/*node*-nya ialah *array* yang berisi *state* dari papan permainan. Dan tiap *node* tersebut akan diberi bobot untuk menentukan *node*/langkah bidak mana saja yang menguntungkan bagi pemain komputer. *Node*/langkah bidak yang berisi langkah terpilih tersebut pula yang akan menjadi data keluaran untuk dijadikan acuan langkah pemain selanjutnya.

Data masukan yang akan digunakan pada proses pencarian diambil dari data permainan, data-data tersebut ialah:

a. Koordinat setiap bidak, Data masukan koordinat bidak ini akan memberi info tentang koordinat bidak pada papan dan sehingga algoritma dapat menentukan kemungkinan tiap langkah tiap bidak.
b. Langkah valid bidak, Algoritma akan membutuhkan langkah valid pada permainan agar dapat menentukan simulasi langkah yang dilakukan dan langkah yang akan diambil.
c. Giliran Pemain, Permainan akan memberi variabel giliran pemain pada algoritma UCT agar algoritma dapat mengetahui kapan waktu yang tepat untuk menggunakan langkah hasil pencarian pada permainan.
d. Status Kemenangan, Tujuan permainan untuk pemain komputer ialah menghabiskan bidak pemain manusia sehingga status menang menjadi *true.* Hal tersebut akan dijadikan *goal state* pada pohon pencarian, sehingga jika sudah tercapai maka algoritma akan mengetahui langkah apa saja yang dapat diambil untuk mencapai *goal state* tersebut.

Pada algoritma UCT terdapat 4 proses yaitu pemilihan node, ekspansi pohon pencarian, simulasi pohon pencarian dan *update value* setiap node. Pada implementasinya proses *update value* akan digabungkan pada proses lain dan proses ekspansi dan simulasi akan berjalan bersamaan.



Gambar 7 Alur pencarian algoritma UCT

Pada awal proses akan dilakukan inisialisasi *root* pohon pencarian. *Root* pohon pencarian akan diambil dari data masukan koordinat bidak papan permainan.

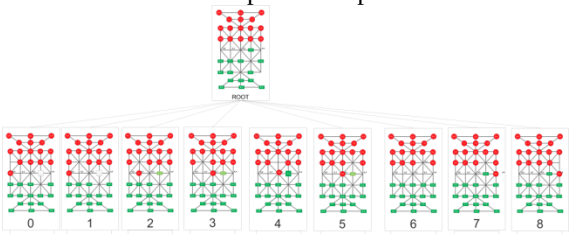
Selanjutnya akan dilakukan pengecekan apakah waktu simulasi telah melebihi batas maksimum waktu simulasi, Karena pencarian baru dimulai maka nilai waktu simulasi masih 0, maka akan dilakukan proses ekspansi dan simulasi.

Pada proses ekspansi akan dilakukan pengecekan pada pohon pencarian apakah *node* memiliki anak atau tidak. Pada algoritma UCT, nilai *visit* akan menjadi acuan apakah algoritma akan membuat simulasi atau mengevaluasi pohon pencarian yang sudah ada. Nilai *visit = 0* pada *node*, akan membuat algoritma menjalankan proses simulasi, sedangkan untuk nilai *visit* di atas 0 maka akan dilakukan perhitungan dengan rumus UCT.

Karena jika visit lebih dari 0, maka nilai UCT akan dapat dihitung, sehingga setelah proses ini, nilai-nilai UCT yang didapat akan dapat dibandingkan untuk pengambilan keputusan. Sedangkan nilai *win* akan ditambah 1 setiap kali suatu *node* disimulasikan. Namun, jika pada akhir pohon pencarian jalur tersebut merupakan potensi kekalahan, maka nilai *win node* menuju kekalahan akan diperbaharui menjadi = 0.

Karena hanya terdapat *node root* pada pohon pencarian, maka algoritma UCT akan membuat anak dari *root* berdasar pada data masukan langkah valid.

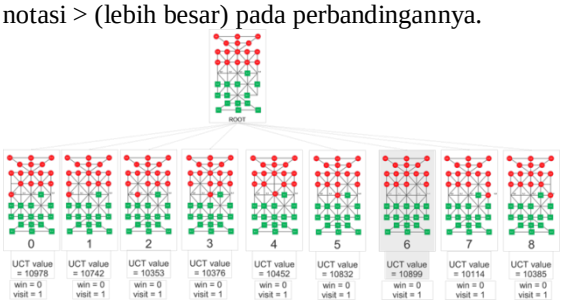
Berdasarkan pada data masukan langkah valid, maka akan dibuat *node* baru sebagai anak dari *root* yang merupakan pilihan langkah yang dapat diambil oleh pemain komputer pada gilirannya. Setiap kemungkinan adalah *array* yang akan dibentuk menjadi anak pohon dari *root* dengan menggerakkan bidak sesuai dengan *array* bersangkutan. Gambar berikut ialah hasil dari proses ekspansi dari *root*.



Gambar 8 Pohon pencarian algoritma UCT

Pohon pencarian tersebut belum memiliki nilai *visit/win* pada *node*-nya karena belum pernah dilewati dan dievaluasi, oleh karena itu akan dipilih secara acak *node* yang akan digunakan pada proses simulasi.

Node root memiliki 9 anak, cara pemilihan untuk *node* yang belum memiliki nilai ialah tiap *node* akan diberi UCT *value* dengan nilai acak, berikut contoh nilai acak rentan 10.000 sampai 11.000 [20] yang diberikan pada *node-node* tersebut. *Node* dengan nilai UCT tertinggi akan dipilih. Jika pada proses pemilihan / pembandingan nilai UCT yang didapat secara acak maupun dengan rumus 2.1 ternyata didapat 2 nilai yang sama, maka akan dipilih nilai yang lebih dulu ditemukan. Hal ini berdasar pada algoritma seleksi yang menggunakan notasi > (lebih besar) pada perbandingannya.



Gambar 9 Pohon pencarian dengan nilai UCT acak

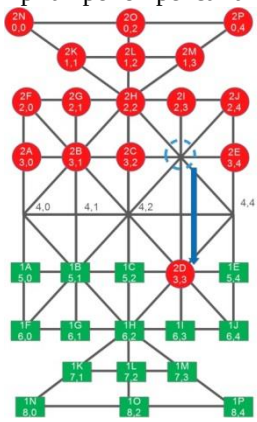
Pada proses simulasi akan dicari kemungkinan yang akan diambil oleh pemain manusia sebagai acuan langkah selanjutnya. Proses ini akan dilakukan berulang-ulang dengan setiap iterasi akan dilakukan pergantian pemain. Setiap *node* yang mengalami proses simulasi nilai *visit*-nya akan ditambah 1 untuk *node* tersebut dan juga *parent node*-nya. Hal tersebut terlihat pada gambar 3.15 dimana *parent node* (*node 6*) telah memiliki nilai visit 2 karena telah dilakukan 1 kali simulasi dan 1 kali ekspansi dari *node* tersebut. Sedangkan setiap *child node*-nya masih memiliki nilai *visit*=1 dan *win*=0.

Status *win* akan didapatkan dari perhitungan jumlah bidak pemain manusia, jika jumlahnya = 0, maka *win* bernilai *true* dan sebaliknya jika jumlahnya > 0 maka *win* bernilai *false*. Pada *node* 312 merupakan keadaan dimana pemain manusia kehabisan bidak. Maka *node-node* yang mengarah pada *node* 312, akan disimpan pada memori dan akan diperbaharui nilai *win*-nya.

Tabel 1 Data nilai UCT setiap node dilewati

Level pohon pencarian	***Node* Terpilih**	**UCT**
1	6	5.824311
2	16	5.730502
3	23	5,635468
4	30	5,560089
5	45	5,441503
6	57	5,342448
7	68	5,241922
8	79	5,139849
9	91	5,036147
10	98	4,930726
11	109	4,823486
12	117	4,71432
13	119	4,603107
14	137	4,489712
15	149	4,373986
16	160	4,255761
17	169	4,134847
18	176	4,011027
19	194	3,884054
20	203	3,753641
21	212	3,619456
22	221	3,481104
23	236	3,338118
24	241	3,189929
25	253	3,035842
26	261	2,874982
27	272	2,706229
28	276	2,528101
29	284	2,338566
30	286	2,134703
31	300	1,912018
32	302	1,662906
33	312	1,37233

didapat *node* yang berpotensi kemenangan pemain komputer pada setiap *level* pohon pencarian. pada level 1 pohon pencarian akan dipilih *node* 6 sebagai langkah pilihan untuk pemain komputer. Sesuai dengan *node* tersebut, Algoritma akan menghasilkan langkah dengan (3,3) adalah posisi bidak awal dan (5,3) adalah posisi bidak akhir. Perbandingan nilai pada pohon pencarian dapat dilihat pada lampiran pohon pencarian.



Gambar 10 Langkah hasil pencarian algoritmaUCT

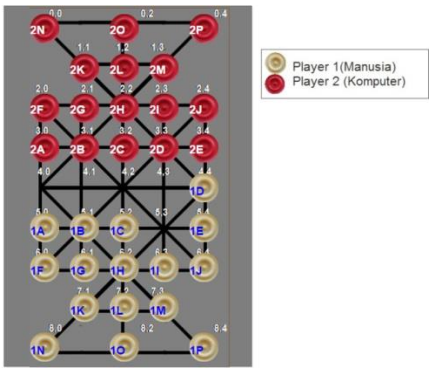
2.5 Pengujian Metode

Pada bagian ini akan dilakukan pengujian terhadap proses yang dilakukan oleh metode/algoritma UCT pada pemilihan langkah pada permainan Damdaman. Pada pengujian ini akan digunakan beberapa skenario pada papan permainan, yang diantaranya terdapat pada tabel berikut.

Tabel 2 Tabel skenario pengujian metode

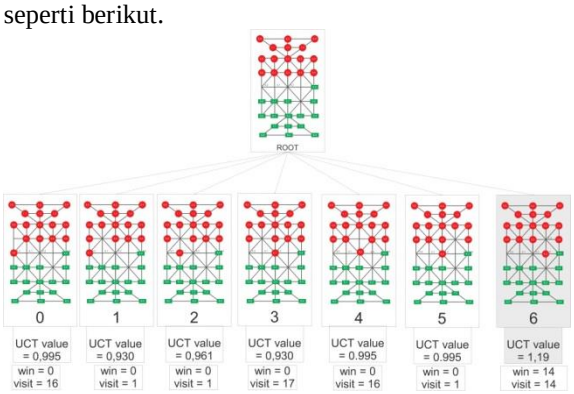
Komponen yang diuji	Skenario pengujian	Jenis pengujian
Hasil Pencarian	Keadaan awal permainan	Akurasi pilihan
	Keadaan bidak lawan dapat dimakan	Akurasi pilihan
Presentase Kemenangan	Memainkan permainan hingga salah satu pemain kalah	Akurasi pilihan

Pada awal permainan, kondisi papan akan terlihat berbeda tergantung pada pilihan gerakan yang akan diambil oleh pemain pertama. Pada contoh ini pemain pertama akan memindahkan bidak dari koordinat (5,3) ke koordinat tanpa bidak di koordinat (4,4). Selanjutnya maka algoritma pencarian akan mengambil data-data masukan dari keadaan permainan untuk menghasilkan pilihan langkah yang sesuai dengan keadaan data masukan dan kondisi papan permainan



Gambar 11 Kondisi tanpa kemungkinan makan

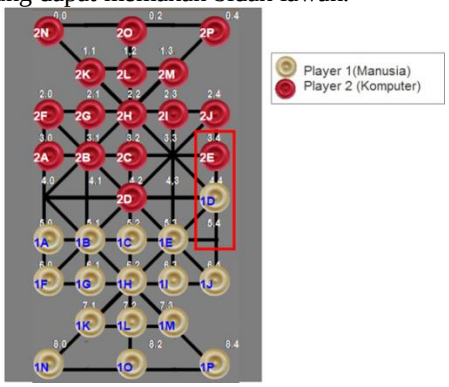
Dari data yang didapatkan, akan dihasilkan pohon pencarian *level* 1 sesuai dengan data tadi. Pohon pencarian yang dihasilkan akan terlihat seperti berikut.



Gambar 12 Pohon pencarian keadaan tanpa makan

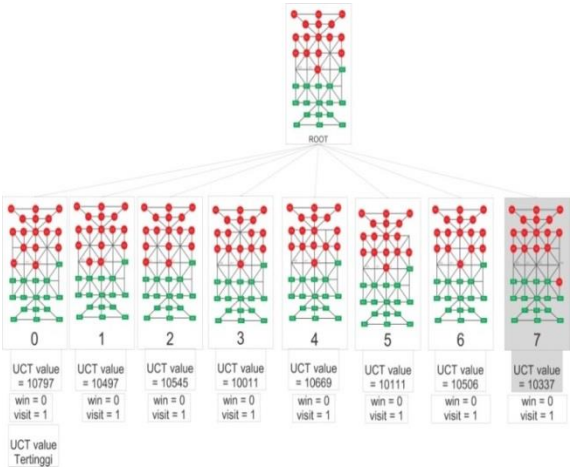
Dari gambar di atas akan diambil langkah sesuai dengan *node* 6 yang memiliki nilai UCT tertinggi dari *node* lain di *level* yang sama.

Sedangkan untuk keadaan papan permainan dimana ditemui kemungkinan makan, maka algoritma akan diarahkan agar memprioritaskan bidak yang dapat memakan bidak lawan.



Gambar 13 Kondisi dengan kemungkinan makan

Dari gambar diatas, maka pohon pencarian yang dihasilkan akan memprioritaskan bidak yang mampu melompati/memakan bidak lawan. Hal ini agar pemain komputer memiliki kemungkinan lebih baik dalam menentukan langkah yang potensial dan baik menurut aturan permainan Damdaman tersebut.



Gambar 14 Pohon pencarian keadaan bisa makan

Gambar di atas menghasilkan pohon pencarian dengan pilihan akhir memilih langkah yang menggerakan bidak pemain komputer untuk memakan bidak pemain lawan. Sesuai dengan aturan permainan dimana jika terdapat kemungkinan makan, maka pemain harus mengambil kesempatan tersebut.

Pada skenario pengujian persentase kemenangan pemain komputer, akan dilakukan pertandingan melawan 5 pemain manusia dimana setiap pemain akan bermain melawan pemain komputer sebanyak 6 kali. Setiap permainan akan disimpan pemain yang memenangkan permainan dan waktu yang digunakan pemain dalam permainan.

Tabel 3 Data pengujian persentase kemenangan

Game Ke-	Pemain Manusia Ke-	Waktu AI (Detik)	Status Menang AI
1	1	99,288	Kalah
2	1	107,954	Kalah
3	1	137,496	Kalah
4	1	113,136	Menang
5	1	111,176	Menang
6	1	121,926	Kalah
7	2	149,616	Menang
8	2	140,032	Kalah
9	2	125,342	Menang
10	2	102,895	Kalah
11	2	116,624	Kalah
12	2	101,822	Menang
13	3	78,331	Kalah
14	3	165,452	Menang
15	3	133,365	Menang
16	3	117,992	Menang
17	3	107,887	Kalah
18	3	157,68	Kalah
19	4	140,416	Menang
20	4	153,034	Kalah
21	4	136,62	Menang
22	4	96,668	Menang
23	4	103,298	Menang
24	4	88,891	Menang
25	5	134,19	Kalah
26	5	127,275	Menang
27	5	114,24	Kalah
28	5	84,756	Menang
29	5	115,245	Menang
30	5	136,686	Menang

Dari data tersebut didapatkan jumlah kemenangan pemain komputer sebanyak 16 kali dari 30 kali permainan atau dengan persentase kemenangan 53%.

2.5 Pengujian Performansi

Pengujian ini akan melihat bagaimana performa dari metode yang digunakan, berdasarkan memori yang digunakan dan waktu yang dibutuhkan oleh algoritma pencarian untuk mendapatkan langkah terpilih.

Berdasarkan pada data pengujian performansi didapatkan kesimpulan bahwa memori yang digunakan saat pencarian tidak dipengaruhi oleh jumlah bidak pada papan permainan, tetapi tergantung oleh waktu yang diberikan untuk melakukan proses pencarian. Semakin banyak waktu tersedia maka memori akan semakin banyak digunakan. Kemungkinan makan yang tersedia pada langkah pilihan juga akan mempengaruhi waktu dan memori karena pencarian akan fokus pada pilihan langkah tersebut sehingga pohon pencarian yang dibuat akan semakin kecil ukurannya.

Nilai-nilai yang didapatkan ialah nilai performansi waktu pencarian dengan rata-rata 8011 *milisecond* dan performansi memori dengan rata-rata 11.378 *byte* (11 *kilobyte*).

3. PENUTUP

Bagian ini akan berisi kesimpulan dan saran. Kesimpulan akan meringkas hasil yang didapatkan dari penelitian ini, sedangkan saran akan berisi masukan yang penulis dapat dari hasil penelitian untuk digunakan oleh penelitian selanjutnya.

Kesimpulan yang didapat dari hasil penelitian untuk Implementasi Algoritma *Upper Confidence Bounds For Trees* (UCT) Untuk Pengambilan Keputusan Pada Permainan Damdaman ialah sebagai berikut.

1. Akurasi pilihan langkah yang diambil sudah sesuai dengan aturan permainan dengan nilai persentase kemenangan pemain komputer dihasilkan sebesar 53%.
2. Algoritma UCT dapat digunakan untuk mencari langkah pada permainan Damdaman
3. Nilai performansi waktu pencarian dengan rata-rata 8011 *milisecond* dan performansi memori dengan rata-rata 11.378 *byte* (11 *kilobyte*).

Sedangkan saran untuk meningkatkan persentase kemenangan pada penggunaan algoritma UCT pada permainan Damdaman ialah Penggunaan nilai tambahan berupa nilai langkah aman yaitu yang menunjukan bahwa apakah suatu langkah kemungkinan akan merugikan berdasarkan aturan makan pada permainan Damdaman. Nilai ini dapat digunakan pada variabel *win* pada algoritma UCT, misalnya langkah merugikan akan selalu diberi nilai *win* 0 agar algoritma fokus pada langkah menguntungkan.

DAFTAR PUSTAKA

[1] S. Dharmamulya, *Permainan Tradisional Jawa*. Yogyakarta: Penerbit Kepel Press, 2005.

[2] A. Prasetyo, "Implementasi Kecerdasan Buatan Pada Permainan Damdaman Menggunakan Algoritma Minimax," Ilmu Komputer Universitas Pendidikan Indonesia, Bandung, Skripsi Juni 2013.

[3] A. Mustaram, "Game Development Of Traditional Damdaman," Teknik Informatika Universitas Komputer Indonesia, Bandung, Skripsi Juli 2013.

[4] Panji Febriyanto, "Penerapan Algoritma Negamax Untuk Menghasilkan Langkah Yang Optimal Pada Permainan Dam Daman," Teknik Informatika Universitas Komputer Indonesia, Bandung, Skripsi Juni 2016.

[5] C. Browne and E. Powley, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, Maret 2012.

[6] Oleg Arenz, "Monte Carlo Chess," *Darmstadt University of Technology Thesis*, April 2012.

[7] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artificial Intelligence* , vol. 175, April 2011.

[8] M. Wiering and M. V. Otterlo, *Reinforcement Learning*, 12th ed. Berlin: Springer Berlin Heidelberg, 2012.

[9] Pressman, *Software Engineering : A Practitioner's Approach*, 7th ed.: McGrow Hill, 2010.

[10] D. H. and D. T. R. Siregar, *Permainan Tradisional Indonesia*. Jakarta: Direktorat Jenderal Kebudayaan , 1998.

[11] K. Knight and E. Rich, *Artificial Intelligence*.: McGraw-Hill, 1991.

[12] Suyanto, *Artificial Intelligence*. Bandung: Informatika, 2007.

[13] V.P. Singh, *System Modeling and Simulation*.: New Age International, 2009.

[14] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo Tree Search: A New Framework for Game AI," *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, Oktober 2008.

[15] Yun-Ching L. and Y. Tsuruoka, "Adapting Improved Upper Confidence Bounds for Monte-Carlo Tree Search," *Advances in Computer Games: 14th International Conference*, vol. 14, pp. 53-64, Juli 2015.

[16] S. S. M. Rosa, *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek*. Bandung: Penerbit Informatika, 2014.

[17] P. Carey, *New Perspectives on HTML CSS*, 4th ed.: Cengage Learning, 2013.

[18] Marijn Haverbeke, *Eloquent Javascript*. San Francisco: William Pollock, 2011.

[19] Hariyanto B., *Rekayasa Sistem Berorientasi Objek*. Bandung: Informatika Bandung, 2004.

[20] Magnus Persson. (2013, Agustus) Sensei Library. [Online]. HYPERLINK "http://senseis.xmp.net/?UCT" http://senseis.xmp.net/?UCT

[21] M. Fowler, *UML Distiled*. Yogyakarta: Penerbit Andi, 2005.