PENERAPAN ALGORITMA *MINIMAX* PADA PERMAINAN CONGKLAK

Khoirul Umam [a], Suastika Y. Riska [b], Gulpi Qorik O. P. [c]

NRP [a] 5113201017, [b] 5113201019, [c] 5113201021
[a,b,c] S2 Teknik Informatika Jurusan Teknik Informatika Fakultas Teknologi Informasi
Institut Teknologi Sepuluh November Surabaya

ABSTRAK
Permainan congklak merupakan salah satu permainan tradisional yang dimainkan oleh dua orang dengan tujuan saling mengalahkan satu sama lain. Pada permainan semacam itu, algoritma *minimax* dapat diterapkan di dalam *agent* agar *agent* seolah-olah dapat berpikir dan bertindak atau mengambil keputusan seperti manusia. Berdasarkan hasil pengujian pada penelitian ini dapat diketahui bahwa *agent* yang menggunakan algoritma *minimax* lebih dapat menentukan langkah yang tepat untuk mendapatkan kemenangan dibandingkan dengan *agent* yang tidak menggunakan algoritma *minimax*.

Kata kunci: algoritma *minimax*, *game*, congklak, *tree search*

1. PENDAHULUAN

Di dalam industri *game* hampir semua *game* yang dikembangkan menerapkan kecerdasan buatan atau *artificial intelligence (AI)*. Penerapan AI dalam *game* mampu membuat sebuah komputer atau sistem tampak memiliki kecerdasan dan dapat berperilaku atau berpikir seperti manusia. Pada suatu permainan, AI berperan untuk menciptakan interaksi antara manusia dengan komputer atau sistem (*agent*). Menurut [1], *game* adalah sesuatu yang sangat menarik dan menjadi sub topik tersendiri di dalam kecerdasan buatan. Alasan *game* menjadi menarik antara lain ialah kriteria menang atau kalah jelas, dapat mempelajari permasalahan, alasan histori, menyenangkan, dan biasanya mempunyai *search space* yang besar (misal *game* catur mempunyai 35100 *node* dalam *search tree* dan 1040 *legal states*).

Menurut [3], berdasarkan tipenya, *game* dibagi menjadi dua tipe. Pertama, *game* dengan informasi lengkap (*perfect information game*) yaitu suatu game dimana pemain mengetahui semua langkah yang mungkin terjadi dari dirinya sendiri dan dari lawan, serta hasil akhir dari permainan mereka. Kedua, *game* dengan informasi tidak lengkap (*imperfect information game*) yaitu pemain tidak tahu semua kemungkinan langkah dari lawan.

Kebanyakan *game* yang dimainkan oleh dua pemain secara bergantian, dikembangkan menggunakan algoritma *minimax*. Seperti penelitian sebelumnya yang dilakukan oleh [2] tentang apakah algoritma *minimax* memang merupakan strategi yang optimal dalam suatu *game*. Di dalam *tree search*, *human* diposisikan sebagai *MAX* dan *computer* sebagai MIN. Posisi dari *node* yang dipilih oleh masing-masing pemain adalah langkah maksimal untuk mencapai posisi terbaik dalam mengalahkan lawan. Penelitian yang dilakukan oleh [2] menggunakan *game tic-tac-toe* dan *game brigde*, dimana ditunjukkan pemodelan lawan dan modifikasi berikutnya dari strategi *minimax* sehingga meningkatkan variasi dari kedua permainan tersebut. Percobaan menunjukkan bahwa *agent* bisa mendapatkan keuntungan dengan beradaptasi terhadap cara lawan bermain, dibandingkan menggunakan strategi umum yang sama terhadap semua pemain. Hasilnya pemodelan lawan pada *game bridge* lebih penting daripada *game tic-tac-toe*.

Perencanaan optimasi algoritma *minimax* dalam *game* digunakan pada dua pemain. Pada *game tic-tac-toe* dalam [5] diusulkan fungsi untuk menetapkan posisi permainan yang mewakili seberapa besar kemungkinan posisi yang mengarah ke kemenangan dengan menghitung skor untuk setiap posisi yang dimainkan. Dengan demikian akan didapatkan skor tertinggi dan langkah yang tepat untuk mencapai kemenangan.

Permainan congklak merupakan salah satu permainan tradisional yang keberadaannya sudah mulai ditinggalkan. Hal ini dikarenakan perkembangan teknologi yang menawarkan *game* atau permainan digital yang lebih canggih dan menarik, baik yang dimainkan pada komputer maupun perangkat *mobile*. Ada pula fakta yang muncul dikalangan masyarakat bahwa permainan tradisional mulai tergeser dikarenakan sulitnya ditemukan alat permainan tradisional yang dijual di toko-toko mainan. Selain itu, kebanyakan masyarakat juga berpikir jika permainan harus menggunakan alat, mereka harus membawa atau setidaknya terdapat fasilitas atau alat permainan tersebut dimanapun mereka bermain.

Salah satu jenis permainan tradisional yang mulai jarang dimainkan saat ini ialah permainan congklak. Permainan congklak sendiri merupakan permainan yang dapat dilakukan oleh maksimal dua orang pemain yang saling berlawanan menggunakan sebuah papan dengan sejumlah lubang di atasnya yang disebut sebagai papan congklak (ada pula yang menyebutnya sebagai dakon). Masing-masing pemain dalam permainan congklak bertujuan untuk mengumpulkan batu sebanyak-banyak di dalam "lumbung"nya agar dapat memenangkan permainan. Karena dalam permainan congklak kemenangan ditentukan oleh jumlah batu yang dapat disimpan oleh pemain di dalam lumbungnya.

Dengan munculnya permasalahan-permasalahan tersebut pada masyarakat, peneliti ingin kembali kembali melestarikan permainan tradisional, khususnya permainan congklak dengan cara yang lebih menarik dan canggih serta mengikuti perkembangan zaman, terutama kepada generasi muda yang mulai melupakan keberadaan permainan tradisional tersebut. *Game* Congklak yang dikembangkan ini dibuat dengan mengimplementasikan algoritma *minimax* di dalamnya. Tujuannya adalah untuk mengembangkan permainan congklak dalam bentuk digital yang menerapkan algoritma *minimax*. Sehingga dapat dihasilkan suatu *game* yang memiliki kecerdasan, kecermatan, dan mampu mengambil keputusan dengan tepat agar strategi yang dijalankan dalam suatu permainan menjadi optimal.

2. *GAME PLAYING* DAN PERMAINAN CONGKLAK

2.1 *Game Playing*

Definisi *game playing* menurut [3] ialah *game* diwakili oleh pohon pencarian di mana *node-node* menunjukkan semua kemungkinan keadaan (*state*) *game* dan sisi-sisi (*edges*) mewakili langkah antara kedua pemain. *Game playing* merupakan *problem* pencarian yang didefinisikan oleh beberapa komponen berikut ini:

1. Keadaan awal (*initial state*), yaitu keadaan yang mendefinisikan konfigurasi awal permainan dan mengidentifikasi pemain pertama yang bergerak.
2. Fungsi penerus (*successor function*), yang bertugas mengidentifikasi kemungkinan-kemungkinan yang dapat dicapai dari keadaan saat ini. Fungsi ini membuat sebuah daftar pasangan gerak dan keadaan, yang masing-masing pasangan menunjukkan langkah legal dan keadaan yang dihasilkan.
3. *Goal test*, yang bertugas untuk memeriksa apakah suatu keadaan tertentu adalah keadaan tujuan atau bukan. Keadaan-keadaan dimana permainan berakhir disebut sebagai keadaan *terminal*.
4. *Path cost*/*utility*/*payoff function*, yang memberikan nilai numerik untuk keadaan-keadaan *terminal*. Dalam catur, hasilnya adalah menang, kalah atau seri, dengan nilai 1, -1, atau 0. Beberapa *game* memiliki rentang yang lebih luas dari hasil yang mungkin.

Game merupakan sesuatu yang dapat dimainkan dan terdapat aturan tertentu di dalamnya. Peranan *game* adalah sebagai penghibur seseorang yang dalam keadaan bosan atau jenuh terhadap kegiatan sehari-hari yang dilakukannya. Menurut [9] secara umum *game* memiliki properti dan pengkategorian, antara lain:

1. Kooperatif dan non-kooperatif. Dalam model *game* kooperatif, pemain dapat membentuk gabungan dan seharusnya

sanggup membahas situasi serta menyepakati rencana bersama dari suatu tindakan, perjanjian yang harus diambil untuk dapat dilaksanakan. Sedangkan untuk model *game* non-kooperatif, diasumsikan bahwa setiap pemain bertindak secara independen, tanpa kolaborasi atau komunikasi dengan yang lain.

2. Sekuensial (dinamik) dan simultan (statis). Pada *game* sekuensial, masing-masing pemain memilih tindakannya sebelum yang lain dapat memilih mereka, yaitu pemain bergiliran untuk bermain. Sehingga, pemain terakhir memiliki beberapa informasi tentang tindakan yang diambil oleh pemain sebelumnya. Sedangkan *game* simultan, setiap pemain memilih tindakannya tanpa pengetahuan tentang tindakan pemain lain.
3. Deterministik dan stokastik. Model deterministik adalah tidak mempertimbangkan adanya pengaruh acak antar individu. Sedangkan model stokastik adalah mempertimbangkan adanya pengaruh acak antar individu, sehingga terdapat peluang didalamnya.
4. *Game* dengan informasi lengkap (pemain mengetahui semua langkah yang mungkin terjadi) dan *game* dengan informasi yang tidak lengkap (pemain tidak mengetahui semua kemungkinan langkah lawan).
5. *Zero-sum* dan *non-zero-sum.* Pada *game zero-sum* keuntungan atau kerugian dari pemain, yang seimbang dengan kerugian dan keuntungan dari pemain lain pada saat yang sama. Sehingga jika keuntungan dan kerugian dari semua pemain dijumlahkan, hasilnya akan sama dengan nol. Sedangkan *game non-zero-sum* hasil penjumlahannya tidak sama dengan nol.

2.2 Permainan Congklak

Sejarah permainan congklak berasal dari Timur Tengah lalu menyebar ke Afrika, kemudian menyebar ke Asia. Menurut [7], pada tingkat dunia congklak dikenal dengan sebutan mancala.

2.2.1 Definisi Permainan Congklak

Definisi congklak dalam Wikipedia merupakan suatu permainan tradisional yang dikenal dengan berbagai macam nama di seluruh Indonesia. Pada permainan congklak, cangkang kerang digunakan sebagai biji congklak. Atau jika tidak ada, kadangkala digunakan biji-bijian dari tumbuh-tumbuhan atau batu-batu kecil. Permainan ini biasanya dimainkan oleh dua orang secara bergantian. Prinsip permainan congklak adalah mengambil biji atau batu dari salah satu lubang kemudian memindahkan biji atau batu tersebut satu persatu pada lubang lain secara bergantian, kemudian pemenangnya ditentukan oleh jumlah biji terbanyak yang dimiliki pemain dalam lubang induk (lumbung).

2.2.2 Aturan Permainan Congklak

Aturan permainan di dalam *game* Congklak yang kami kembangkan ini antara lain sebagai berikut:

1. Dimainkan oleh dua pemain, yaitu *human* vs. *computer* yang dilakukan secara bergantian.
2. Rute yang ditempuh pemain 1 adalah melewati semua lubang berlawanan arah jarum jam, kecuali lubang induk pemain 2. Begitu juga dengan pemain 2, melewati semua lubang, kecuali lubang induk pemain 1.
3. Biji yang diambil pada salah satu lubang dibagikan satu persatu ke lubang yang lain secara berurutan, sampai biji tersebut habis, dan berhenti pada lubang yang kosong ataupun lubang induk.
4. Setiap pemain hanya boleh memulai permainan dengan mengambil batu atau biji pada lubang congklak yang berada di daerahnya.
5. Penambahan biji pada lubang induk (biasanya dinamakan *bank* atau lumbung), selain dengan cara menambah satu per satu biji saat pemain berjalan sesuai dengan rute yang dilalui, juga bisa dilakukan dengan cara menembak daerah lawan (namun tidak dapat menembak lubang induk).

3. ALGORITMA *MINIMAX*

Algortima *minimax* cocok digunakan dalam *game* yang dimainkan oleh dua pemain secara bergantian. Tujuan algoritma ini adalah untuk melakukan proses pencarian solusi dengan cara pembuatan suatu *search tree*. Pada setiap *node* di dalam *search tree* dapat ditentukan nilai *minimax*-nya. Algoritma *minimax* beroperasi secara rekursif pada *game tree*, dimana iterasi pemain 1 (*MAX*) yang mengambil nilai maksimum dari *node* di bawahnya dan pemain 2 (*MIN*) yang mengambil nilai minimum dari *node* di bawahnya [4]. Menurut [6], *minimax* sebagai salah satu algoritma untuk menyelesaikan permasalahan *game*, melakukan proses pencarian solusi dengan membangun suatu pohon pencarian *(search tree)* dan mekanisme pencarian menggunakan prinsip kerja *depth first search (DFS)*.

Pada algoritma *minimax* digunakan teori *zero-sum*, yaitu mendeskripsikan situasi dimana jika terdapat pemain yang mengalami pendapatan (keuntungan), pemain lain akan mengalami kehilangan (kerugian) dengan nilai yang sama dari pendapatan tersebut, begitu pula sebaliknya [8]. Dalam [3] ditunjukkan prosedur dari algoritma *minimax*, antara lain:

1. Tandai masing-masing level dari ruang pencarian sesuai dengan langkahnya pada level itu.
2. Mulai dari *node* daun (*node* paling bawah), dengan menggunakan fungsi evaluasi, berikan nilai pada masing-masing *node*.
3. Arah menjalar ke atas: jika *node* orang tua adalah *MAX*, pilihlah nilai terbesar yang terdapat pada *node* anak dan berikan nilai tersebut pada *MAX* (*node* orang tua).
4. Arah menjalar ke atas: jika *node* orang tua adalah *MIN*, pilihlah nilai terbesar yang terdapat pada *node* anak dan berikan nilai tersebut pada *MIN* (*node* orang tua).

4. RANCANGAN *GAME* CONGKLAK

4.1 Model *Tree Search* dan Perhitungan Nilai *Minimax* dalam *Game* Congklak

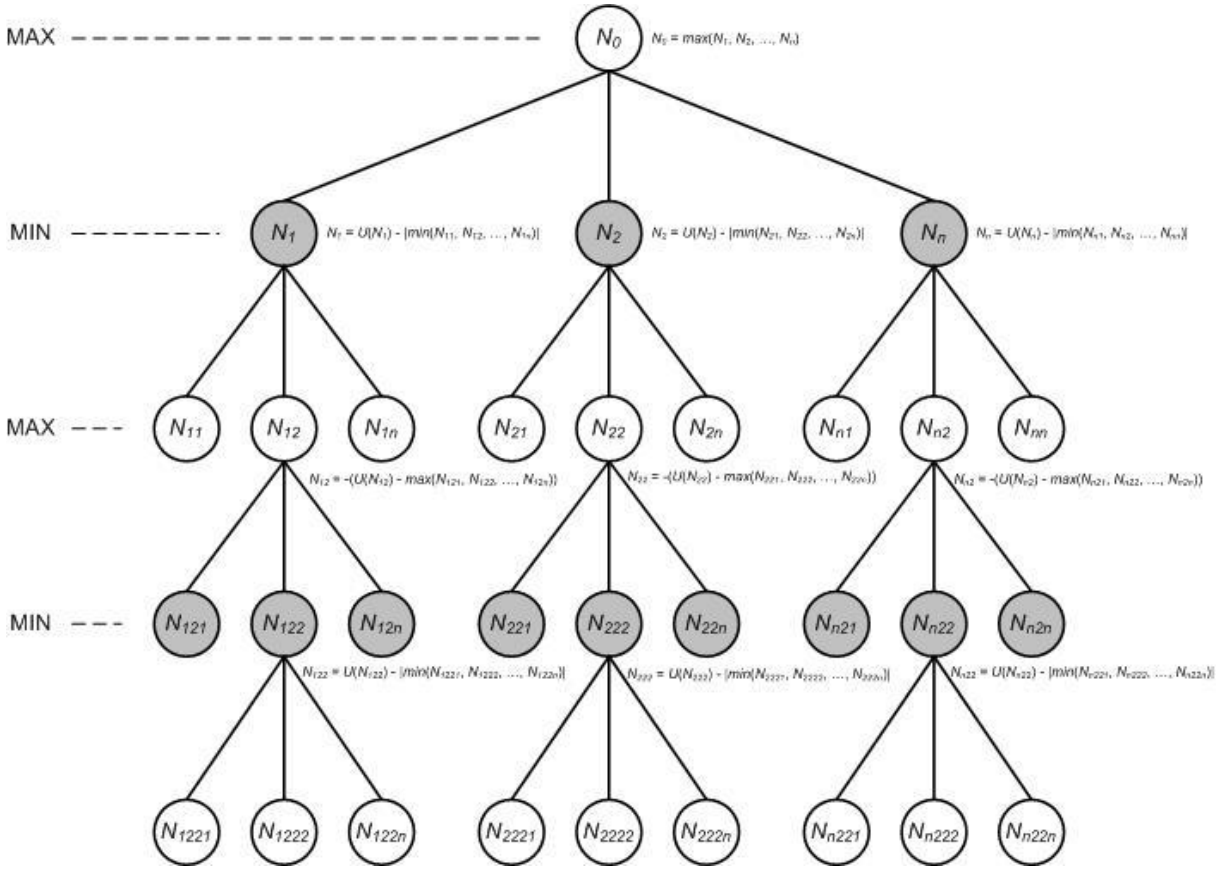
Diagram *tree search* di dalam pengembangan *game* ini digunakan untuk merepresentasikan model pencarian atau perhitungan nilai *minimax* pada tiap kondisi (*state*) papan congklak. Setiap *node* di dalam *tree search* mewakili satu kondisi papan congklak, dimana kondisi tersebut dipengaruhi oleh kondisi sebelumnya (*node parent* dari *node* yang bersangkutan). Sedangkan tingkat atau level di dalam *tree* mewakili giliran pemain untuk menjalankan permainan. *Tree search* untuk *game* Congklak yang dikembangkan ini ditunjukkan pada Gambar 4.1.

Sesuai dengan prinsip algoritma *minimax*, maka setiap level di dalam *tree search* terdiri dari level *MIN* dan level *MAX*, dimana level *MAX* merupakan level atau posisi pemain saat ini dan level *MIN* merupakan level bagi lawan. Hal ini berdasarkan asumsi bahwa pemain akan memilih kondisi yang paling menguntungkan baginya, sedangkan lawan akan berusaha mengarahkan pemain ke kondisi yang paling merugikan.

Asumsikan bahwa *root node* (N_0) pada *tree search* di Gambar 4.1 merupakan posisi pemain saat ini. Pada posisi tersebut pemain dihadapkan pada *n* pilihan atau kemungkinan cara jalan (N_1, N_2, …, N_n). Jumlah pilihan atau kemungkinan ini sama dengan jumlah lubang congklak di area milik pemain yang dapat diambil batunya saat itu.

Untuk menentukan lubang atau *node* mana yang akan dipilih oleh pemain, maka perlu dilakukan perhitungan nilai *minimax* untuk masing-masing *node* N_1, N_2, hingga N_n. Dikarenakan pemain berada pada level *MAX*, maka pemain akan memilih nilai *minimax* terbesar (*MAX*) dari *node-node* tersebut seperti yang ditunjukkan oleh model matematika pada persamaan (4.1).

$$N_0 = \max(N_1, N_2, \cdots, N_n) \qquad (4.1)$$



Gambar 4.1*Tree Search Game* Congklak

Nilai *minimax* untuk suatu *node* N_i bergantung kepada keuntungan dan kerugian yang akan diperoleh pemain jika memilih *node* tersebut. Besar keuntungan pada *node* N_i merupakan jumlah batu yang dapat ditambahkan oleh pemain ke dalam lumbungnya jika pemain memilih lubang congklak yang diwakili oleh *node* tersebut, baik dengan cara menembak batu di daerah lawan atau dengan mengisi lumbungnya secara langsung saat permainan berjalan. Sedangkan besar kerugian pada *node* N_i sama dengan besar keuntungan lawan pada iterasi permainan berikutnya, yaitu jumlah batu yang mungkin dapat ditambahkan oleh lawan ke dalam lumbungnya sendiripada saat lawan mendapatkan giliran bermain berikutnya. Dengan demikian nilai *minimax node* N_i sama dengan besar keuntungan pada *node* atau *state* tersebut dikurangi dengan besar keuntungan yang mungkin diperoleh lawan pada *node* atau *state* berikutnya seperti yang ditunjukkan oleh persamaan (4.2). Dikarenakan keuntungan lawan sama dengan kerugian pemain, maka besar keuntungan lawan jika dilihat dari sudut pandang pemain akan bernilai negatif. Dengan asumsi bahwa lawan berada pada level *MIN* di dalam *tree search*, maka besar kerugian pemain sama dengan nilai absolut dari nilai terkecil *node-node* yang berada di bawah *node* N_i seperti yang ditunjukkan oleh persamaan (4.3).

$$N_i = U(N_i) - U'(N_i) \qquad (4.2)$$

$$U'(N_i) = |\min(N_{i1}, N_{i2}, \cdots, N_{in})| \qquad (4.3)$$

dimana:

N_i	=	nilai *minimax node* N_i
$U(N_i)$	=	keuntungan pemain pada *node* N_i
$U'(N_i)$	=	kerugian pemain pada *node* N_i (keuntungan lawan pada iterasi permainan berikutnya)
i	=	1, 2, …, n

4.2 Algoritma Pemrograman *Game* Congklak

Algoritma untuk menghitung nilai *minimax* pada tiap *node* atau kemungkinan *state* yang dapat dipilih oleh pemain pada *game* Congklak yang kami kembangkan ini dijabarkan dalam bentuk *pseudo-code* yang pada Gambar 4.2.

Pseudo-code pada Gambar 4.2 menunjukkan bahwa penghitungan nilai kerugian suatu *node* dilakukan dengan memanggil prosedur *minimax* secara rekursif untuk kondisi dan pemain berikutnya. Pemanggilan prosedur secara rekursif ini dapat dilakukan selama level pencariannya masih belum mencapai batas maksimal level pencarian yang telah ditentukan sebelumnya. Level pencarian tersebut di dalam pengembangan *game* Congklak ini kami sebut sebagai *depth*. Dengan metode pemanggilan fungsi secara rekursif ini maka evaluasi nilai *minimax* suatu *node* akan dimulai dari *node* paling bawah dalam *tree search* (*leaf node*) dan menjalar ke atas hingga *root node* yang merepresentasikan posisi pemain saat ini.

4.3 Implementasi

Game Congklak dalam penelitian ini dikembangkan sebagai sebuah aplikasi berbasis *web* menggunakan bahasa pemrograman JavaScript (JS). *Screenshot* hasil pengembangan tampilan awal *game* Congklak tersebut ditunjukkan pada Gambar 4.3.

Pada pengembangan *game* Congklak ini kami menyediakan tiga pilihan level atau tingkat kesulitan permainan, yaitu mudah, sedang, dan sulit. Level permainan tersebut dapat dipilih oleh pengguna pada awal permainan di halaman pemilihan level permainan. Pada halaman tersebut pengguna juga dapat menentukan jumlah inisialisasi batu di dalam tiap lubang congklak yang akan dimainkannya nanti. Tampilan halaman pemilihan level permainan dan inisialisasi batu tersebut ditunjukkan oleh Gambar 4.4.

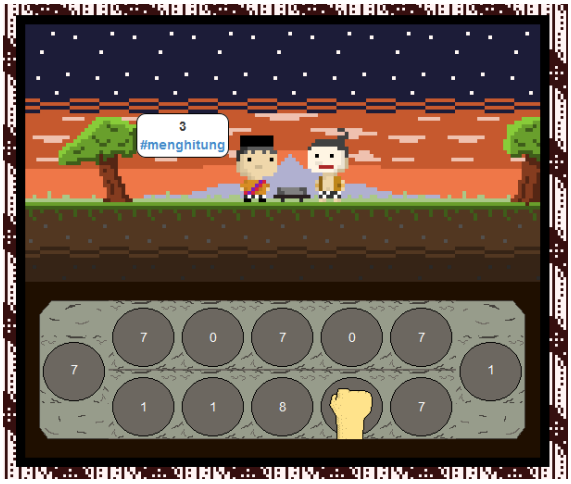
```
procedure minimax(state, player, level)
    minimaxValue := 0
    nodes        := getNodes(state, player)

    for each node in nodes do
        untung := untung(state, node)

        if level < MAX_LEVEL then
            rugi := minimax(nextState, nextPlayer, level+1)
        else
            rugi := 0;
        end

        untungBersih := untung - rugi

        if untungBersih > minimaxValue then
            minimaxValue := untungBersih
        end
    end

    return minimaxValue
```

Gambar 4.2 *Pseudo-Code* Algoritma *Minimax* dalam *Game* Congklak

Gambar 4.3 *Screenshot* Halaman Awal *Game* Congklak

Gambar 4.4 *Screenshot* Halaman Pemilihan Level Permainan dan Inisialisasi Jumlah Batu

Level permainan yang dipilih oleh pengguna nantinya akan mempengaruhi kemampuan *agent* dalam melawan pengguna. Untuk level mudah, *agent* memilih suatu lubang pada papan congklak secara acak tanpa melakukan perhitungan untung-rugi. Untuk level sedang, *agent* memilih suatu lubang pada papan congklak hanya berdasarkan keuntungan yang akan diperolehnya tanpa memperhitungkan kerugiannya. Sedangkan untuk level sulit, *agent* memilih suatu lubang pada papan congklak dengan mempertimbangkan keuntungan dan kerugiannya menggunakan algoritma *minimax* dengan *depth* pencarian tertentu. *Screenshot* tampilan model permainan congklak dalam *game* yang kami kembangkan ini ditunjukkan pada Gambar 4.5.

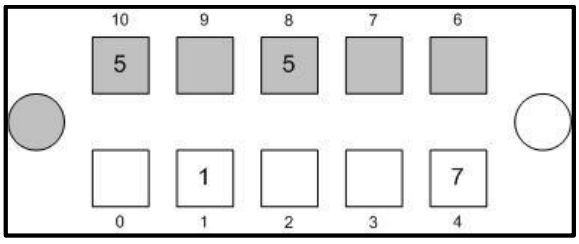


Gambar 4.5 *Screenshot* Halaman Permainan

5. UJI COBA DAN ANALISIS

5.1 Uji Coba Implementasi Algoritma *Minimax*

Uji coba implementasi dilakukan untuk melihat hasil implementasi algoritma *minimax* di dalam kode program *game* Congklak yang telah dikembangkan. Pengujian dilakukan dengan memberikan suatu kondisi papan congklak seperti yang ditunjukkan pada Gambar 5.1. Selain itu juga dilakukan pengaturan berupa pemain yang mendapat giliran bermain saat itu adalah *computer agent* yang berperan sebagai *human player* dengan kemampuan *expert depth* 3 melawan *computer agent* yang berperan sebagai *computer player* dengan kemampuan *expert depth* 5.



Gambar 5.1 Kondisi Papan Congklak untuk Uji Coba

Dari Gambar 5.1 dapat diketahui bahwa terdapat suatu kondisi papan congklak dimana tersisa 1 buah batu pada lubang nomor 1, 7 batu pada lubang nomor 4, 5 batu pada lubang nomor 8, dan 5 batu pada lubang nomor 10. Lubang nomor 0 – 4 meru-

pakan lubang pada daerah *human player*, sedangkan lubang nomor 6 – 10 merupakan lubang pada daerah *computer player*.

Pada kondisi tersebut *agent* yang berperan sebagai *human player* yang mendapat giliran untuk bermain memiliki dua pilihan, yaitu mengambil batu pada lubang 1 atau pada lubang 4. Berdasarkan *log* permainan diperoleh perhitungan nilai *minimax* untuk masing-masing pilihan tersebut sebagai berikut:

1. Pilihan lubang 1:
- Keuntungan : $U(N_1)$ = 5
- Kerugian : $U'(N_1)$ = -3
- Nilai *minimax* : N_1 = $U(N_1)$-$U'(N_1)$ = 8

2. Pilihan lubang 4:
- Keuntungan : $U(N_1)$ = 7
- Kerugian : $U'(N_1)$ = 0
- Nilai *minimax* : N_1 = $U(N_1)$-$U'(N_1)$ = 7

Pilihan *agent*: $\boldsymbol{N_0 = max(N_1, N_4) = N_1}$

Berdasarkan hasil perhitungan tersebut, maka *agent* yang berperan sebagai *human player* akan memilih lubang 1. Perhitungan tersebut menunjukkan bahwa *agent* sudah dapat melakukan pemilihan lubang congklak dengan memperhitungkan keuntungan dan kerugian yang diperolehnya dengan menggunakan algoritma *minimax*.

5.2 Uji Coba Kemampuan

Uji coba selanjutnya ialah uji coba kemampuan yang digunakan untuk melihat kemampuan *agent* yang menggunakan algoritma *minimax* dalam menghadapi pemain atau *agent* lain yang memiliki tingkatan kemampuan yang berbeda-beda. Tingkatan kemampuan pemain tersebut dibedakan menjadi tiga tingkat sesuai dengan level permainan yang diimplementasikan di dalam pengembangan *game* Congklak ini. Ketiga tingkatan tersebut ialah (1) mudah atau *easy*, (2) sedang atau *intermediate*, dan (3) sulit atau *expert*. Pemain dengan tingkatan mudah memiliki karakteristik dalam memilih lubang congklak secara acak tanpa memperhitungkan keuntungan maupun kerugiannya akibat pilihan tersebut. Pemain dengan tingkatan sedang memiliki karakteristik dalam memilih lubang congklak hanya berdasarkan besar keuntungan yang diperolehnya saat itu saja tanpa memperhitungkan kemungkinan kerugian yang akan diperolehnya pada iterasi permainan selanjutnya. Sedangkan pemain dengan karakteristik sulit memiliki karakteristik dalam memilih lubang congklak dengan memperhitungkan besar keuntungan serta kemungkinan kerugian yang akan diperolehnya akibat pilihan tersebut. Pemain dengan tingkatan sulit inilah yang menerapkan algoritma *minimax* dalam penghitungan keuntungan dan kerugian atau pemilihan lubang congklak.

Pengujian kemampuan *game* Congklak ini dilakukan pada komputer dengan spesifikasi prosesor Intel Core i3 2,4 GHz, RAM 3 GB, sistem operasi Windows 7, dan *web browser* Mozilla Firefox versi 26. Pengujian dilakukan dengan mempertandingkan dua buah *computer agent*, dimana *computer agent* yang pertama berperan sebagai *human player* dan *computer agent* yang kedua berperan sebagai *computer player*. Skenario pengujian dibedakan menjadi tiga, yaitu sebagai berikut:

1. *Human player* dengan tingkatan kemampuan sulit (*expert*) dengan besar *depth* yang bervariasi melawan *computer agent* dengan tingkatan kemampuan mudah (*easy*).
2. *Human player* dengan tingkatan kemampuan sulit (*expert*) dengan besar *depth* yang bervariasi melawan *computer agent* dengan tingkatan kemampuan sedang (*intermediate*).
3. *Human player* dengan tingkatan kemampuan sulit (*expert*) dengan besar *depth* yang bervariasi melawan *computer agent* dengan tingkatan kemampuan sulit (*expert*) dengan besar *depth* yang tetap.

Pada masing-masing skenario pengujian tersebut, jumlah inisialisasi batu di dalam tiap lubang congklak pada awal permainan serta urutan pemain juga divariasikan.

Di akhir tiap permainan dilakukan pencatatan skor akhir untuk tiap pemain, pemenang permainan tersebut, serta total waktu eksekusi yang digunakan. Waktu eksekusi dalam hal ini merupakan rentang waktu yang digunakan oleh sistem sejak permainan dimulai hingga permainan selesai. Hasil pengujian untuk skenario pertama ditunjukkan pada Tabel 5.1 Hasil pengujian untuk skenario kedua ditunjukkan pada Tabel 5.2 Sedangkan hasil pengujian untuk skenario ketiga ditunjukkan pada Tabel 5.3.

Berdasarkan hasil pengujian pada Tabel 5.1 dapat diketahui bahwa secara keseluruhan dalam pengujian skenario 1, *agent human player* dengan level *expert* yang menerapkan algoritma *minimax* memiliki persentase kemenangan (83,33%) yang lebih tinggi dibandingkan dengan *agent computer player* dengan level *easy* (4,17%). Dari 24 kali permainan, *agent computer player* hanya memenangkan permainan sebanyak 1 kali. Hal ini menunjukkan bahwa *agent* yang menerapkan algoritma *minimax* lebih mampu menentukan langkah yang tepat untuk meraih kemenangan dibandingkan dengan *agent* yang menentukan langkah secara acak.

Berdasarkan hasil pengujian pada Tabel 5.2 dapat diketahui bahwa secara keseluruhan dalam pengujian skenario 2, *agent human player* dengan level *expert* yang menerapkan algoritma *minimax* memiliki persentase kemenangan sebesar 79,17%. Persentase ini lebih tinggi dibandingkan dengan persentase kemenangan *agent computer player* dengan level *intermediate*, yaitu 16,67%. Hal ini juga menunjukkan bahwa *agent* yang menerapkan algoritma *minimax* lebih mampu menentukan langkah dalam permainan untuk meraih kemenangan dibandingkan dengan *agent* yang menentukan langkah hanya berdasarkan besar keuntungan saat itu saja, meskipun ada kalanya cara tersebut mengarahkan *agent* untuk meraih kemenangan. Atau dengan kata lain algoritma *minimax* yang mempertimbangkan besar keuntungan dan kerugian pemain pada langkah-langkah berikutnya lebih mampu memperhitungkan langkah yang optimal dibandingkan dengan cara hanya memperhitungkan besar keuntungan pada saat itu saja.

Tabel 5.1 Hasil Pengujian Skenario 1

Level *Human*	Level *Computer*	Rata-Rata Skor Akhir		Persentase Kemenangan (%)			Rata-Rata Waktu Eksekusi
		Human	*Computer*	*Human*	*Computer*	*Draw*	
Expert (d: 3)	*Easy*	39,88	25,13	75,00	0,00	25,00	21 detik
Expert (d: 5)	*Easy*	41,25	23,75	87,50	0,00	12,50	19 detik
Expert (d: 7)	*Easy*	40,13	24,88	87,50	12,50	0,00	16 detik
Keseluruhan		40,42	24,58	83,33	4,17	12,50	19 detik

Tabel 5.2 Hasil Pengujian Skenario 2

Level *Human*	Level *Computer*	Rata-Rata Skor Akhir		Persentase Kemenangan (%)			Rata-Rata Waktu Eksekusi
		Human	*Computer*	*Human*	*Computer*	*Draw*	
Expert (d: 3)	Intermediate	37,25	27,75	75,00	25,00	0,00	18 detik
Expert (d: 5)	Intermediate	37,25	27,75	87,50	12,50	0,00	17 detik
Expert (d: 7)	Intermediate	38,38	26,63	75,00	12,50	12,50	18 detik
Keseluruhan		37,63	27,38	79,17	16,67	4,17	18 detik

Tabel 5.3 Hasil Pengujian Skenario 3

Level *Human*	Level *Computer*	Rata-Rata Skor Akhir		Persentase Kemenangan (%)			Rata-Rata Waktu Eksekusi
		Human	*Computer*	*Human*	*Computer*	*Draw*	
Expert (d: 3)	*Expert (d: 5)*	30,75	34,25	37,50	62,50	0,00	24 detik
Expert (d: 5)	*Expert (d: 5)*	32,50	32,50	50,00	50,00	0,00	23 detik
Expert (d: 7)	*Expert (d: 5)*	32,13	32,88	50,00	50,00	0,00	24 detik
Keseluruhan		31,79	33,21	45,83	54,17	0,00	24 detik

Sedangkan berdasarkan hasil pengujian pada Tabel 5.3 dapat diketahui bahwa *agent* yang menggunakan algoritma *minimax* dalam menentukan langkah permainan dengan *depth* atau kedalaman pencarian yang lebih besar cenderung lebih mampu meraih kemenangan, meski dalam kasus-kasus tertentu masih dapat dikalahkan oleh *agent* yang *depth* dalam algoritma *minimax*-nya lebih kecil. Hal ini dapat dilihat dari hasil pertandingan antara *agent human player* yang menggunakan *depth* sebesar 3 dengan *agent computer player* yang menggunakan *depth* sebesar 5. Dari hasil pertandingan tersebut, *agent computer player* memiliki persentase kemenangan sebesar 62,50%, lebih tinggi dibandingkan dengan persentase kemenangan *agent human player* yang hanya sebesar 37,50%. Namun dari hasil pengujian ini juga dapat diketahui meskipun besar *depth* pencarian yang digunakan oleh *agent human player* dengan *agent computer player* berbeda (yaitu 7 dengan 5), persentase kemenangan kedua *agent* tersebut sama (50%). Hal ini mungkin dipengaruhi oleh banyaknya batu yang digunakan dalam permainan saat itu.

6. KESIMPULAN

Permainan tradisional Congklak merupakan permainan yang dapat dilakukan oleh maksimal dua orang pemain, dimana masing-masing pemain bertujuan untuk saling mengalahkan. Setiap langkah permainan yang dilakukan oleh pemain akan mempengaruhi langkah permainan bagi pemain berikutnya. Permainan dengan prinsip tersebut dapat dimodelkan dengan menggunakan algoritma *minimax*. Dengan demikian maka algoritma *minimax* dapat digunakan dalam di dalam pengembangan *game* congklak.

Salah satu prinsip di dalam permainan congklak ialah kemenangan ditentukan oleh jumlah batu yang dapat disimpan oleh pemain ke dalam lumbung atau *bank*-nya. Maka dari itu seorang pemain harus mampu menentukan langkah yang dapat memperbesar keuntungannya sekaligus memperkecil peluang lawan untuk mendapatkan keuntungan. Maka dari itu dilakukan modifikasi pada algoritma *minimax* yang digunakan di dalam pengembangan *game* Congklak ini. Modifikasi tersebut terletak pada cara menghitung nilai *minimax* suatu keadaan (atau *node* di dalam pemodelan keadaan dengan *tree search*), yaitu nilai *minimax* suatu *node* sama dengan besar keuntungan pemain pada *node* tersebut dikurangi dengan kerugian yang mungkin diperolehnya. Besar kerugian tersebut sama dengan salah satu nilai pada *node-node* yang berada di bawah *node* tersebut.

Penerapan algoritma *minimax* di dalam *game* Congklak ini masih belum menggunakan metode *prunning* untuk menyederhanakan proses perhitungan atau pencarian nilai *minimax* suatu *node*. Maka dari itu, peneliti akan mencoba untuk mencari metode *prunning* yang tepat yang dapat digunakan di dalam algoritma *minimax* pada *game* Congklak ini.

7. DAFTAR PUSTAKA

[1] Arifin, Muhammad. 2011. "*Pembuatan Game NIM menggunakan Alpha-Beta Pruning*". Politeknik Elektronika Negeri Surabaya.

[2] Kosciuk, Katarzyna. 2010. "*Is Minimax Really An Optimal Strategy in Games?*". Zeszyty Naukowe Politechniki Bialostockiej Informatyka. 6: 63-75.

[3] Sutojo,T., Mulyanto, Edy., dan Suhartono, Vincent. 2011. "*Kecerdasan Buatan"*. Yogyakarta: Andi Publisher.

[4] Diez, Silvia Garcia., Laforge, Jerome., dan Saerens, Marco.2013."*Rminimax: An Optimally Randomized Minimax Algorithm*". Universit´e Catholique de Louvain.Cybernetics, IEEE Transactions. Vol: 43. Issue: 1.

[5] Strong, Glenn.2011. "*The Minimax Algorithm*". (online). https://www.cs.tcd.ie/Glenn.Strong/3d5/*minimax*-notes.pdf

[6] Saputro, DwiKurniawan. 2011. "*Agen Cerdas Game Remi Berbasis Minimax"*. Program Pascasarjana *Game Technology*. Surabaya: Institut Teknologi Sepuluh Nopember.

[7] Kanty. 2012. "*Congklak, Board Game Tertua di Dunia*". (online). http://segitiga.net/blog/congklak-board-game-tertua-di-dunia

[8] Ayuningtyas, Nadhira. 2008. "*Algoritma Minimax dalam Pencarian Checkers*". Program Studi Teknik Informatika. Institut Teknologi Bandung.

[9] Guerra, Joao Carlos Correia. 2011. "*Classical Checkers*". Universidade Tecnica de Lisboa. Instituto Superior Tecniko.