BAB 2

LANDASAN TEORI

2.1. *Game*

Game atau permainan merupakan sebuah sistem yang melibatkan pemain dalam suatu permasalahan dengan aturan tertentu sehingga menciptakan hasil yang dapat diukur (Salen & Zimmerman, 2003). Sebagai sistem, *game* menyediakan situasi dimana pemain dapat berinteraksi. *Game* memerlukan satu atau lebih pemain untuk menyelesaikan suatu permasalahan yang dibuat. Permasalahan dalam *game* dibuat dengan berbagai bentuk yang dapat diselesaikan secara kerjasama atau kompetitif. Bagian penting lainnya dari *game* adalah aturan yang dibuat untuk memberi batas apa yang pemain bisa lakukan dan yang tidak. Pada akhir permainan, pemain menerima hasil menang / kalah atau skor dalam bentuk angka.

Russel & Norvig (1995) mendeskripsikan sebuah *game* sebagai jenis masalah pencarian dengan komponen-komponen berikut:

a. *The initial state*, yaitu keadaan awal permainan, posisi pemain dan pemain yang pertama melangkah.
b. *Set of operator*s, yang mendefinisikan langkah-langkah yang diizinkan untuk pemain.
c. *Terminal test*, yang menentukan akhir dari permainan. Keaadaan permainan berakhir disebut juga *terminal state.*
d. *Utility function* atau *payoff function*, yang memberikan nilai numerik untuk hasil suatu *game*. Hasil *game* berupa menang, kalah, atau seri diwakili berturut-turut dengan nilai +1, -1, 0.

2.1.1. Game Theory

Game theory adalah ilmu yang mempelajari pengambilan keputusan oleh pemain dalam sebuah game. Keputusan yang diambil pemain harus berpotensi mempengaruhi kepentingan pemain lainnya. Konsep *game theory* menyediakan bahasa untuk

merumuskan, membangun, menganalisa, dan memahami skenario yang strategis. Objek penelitian dari *game theory* adalah *game*, yang merupakan model baku dari situasi interaktif dengan melibatkan beberapa pemain (Turocy & von Stengel, 2001).

Menurut Milington dan Funge (2009), *game theory* mengklasifikasi *game* berdasarkan kriteria berikut ini :

a. Jumlah pemain

   Dalam *game theory*, biasanya sebuah permainan dimainkan oleh minimal 2 pemain.

b. Tujuan

   Sebagian besar *game* strategis bertujuan untuk mendapatkan kemenangan. Pemain dapat dikatakan menang jika pemain lainnya kalah. Hal ini disebut juga *zero-sum game* yaitu kemenangan seorang pemain adalah kekalahan bagi pemain lainnya. Sedangkan untuk *non zero-sum game*, semua pemain dapat menang atau kalah.

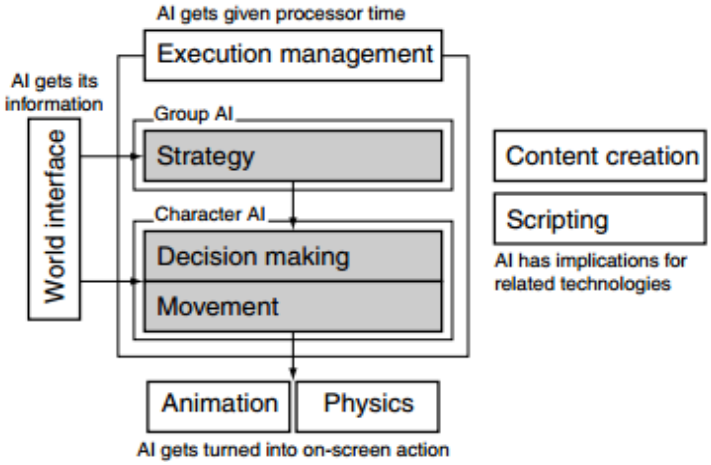
c. Informasi

   Sebuah game disebut *perfect information* jika keadaan permainan dari awal hingga akhir diketahui sepenuhnya oleh setiap pemain. Sedangkan keadaan permainan yang hanya sebagian dapat diamati oleh pemain disebut dengan *imperfect information*.

Konsep *game theory* dapat ditemukan pada permainan yang berbasis giliran seperti yang terdapat pada jenis permainan papan. Contoh dari jenis permainan ini adalah Catur, *Checkers*, *Go*, dan Halma. Permainan-permainan ini memenuhi kriteria dalam *game theory* yaitu *two-player*, *zero-sum*, dan *perfect information*. Sehingga analisa dapat dilakukan dengan mudah menggunakan teknik pencarian.

2.2. Kecerdasan Buatan dalam *Game*

Aplikasi kecerdasan buatan atau *artificial intelligence* pada awalnya fokus pada penyelesaian masalah dan *game*. AI dalam *game* berkaitan dengan bagaimana sistem bertindak dan bagaimana hasil yang diperoleh. *Game* menggunakan AI agar unsur-unsur yang dikendalikan komputer muncul dengan membuat keputusan cerdas ketika di dalam *game* terdapat beberapa pilihan untuk situasi tertentu, yang hasilnya relevan, efektif, dan berguna (Schwab, 2009).



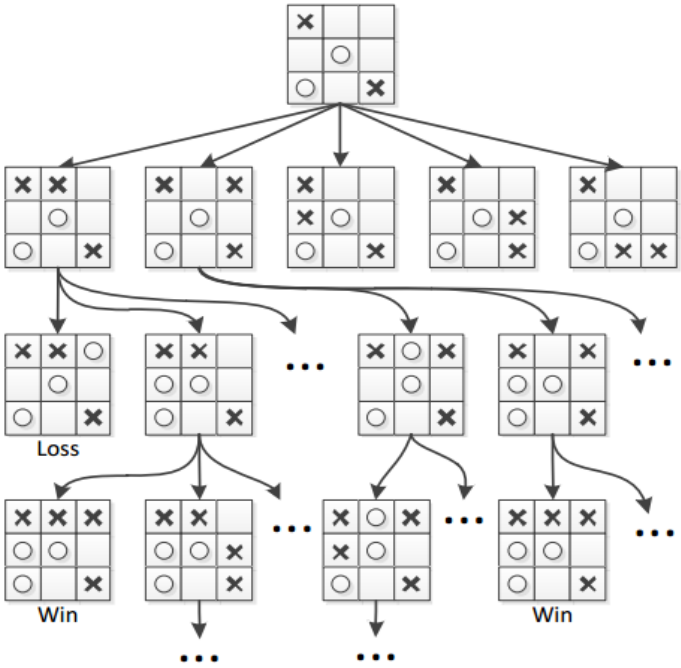
Gambar 2.1. Model AI dalam *Game* (Millington & Funge, 2009)

Berdasarkan Gambar 2.1. tugas AI dalam *game* dibagi menjadi 3 bagian yaitu kemampuan untuk memindahkan karakter (*movement*), kemampuan untuk membuat keputusan (*decision making*), dan kemampuan untuk berpikir taktis atau strategis (*strategy*). Dua bagian pertama berisikan algoritma yang bekerja pada karakter-karakter dasar, dan bagian terakhir beroperasi secara keseluruhan. Ketiga bagian AI ini menjadi penunjang dalam merancang suatu *game* (Millington & Funge, 2009).

Membangun program untuk *game* dengan kinerja tinggi menjadi salah satu keberhasilan besar AI. Hal ini disebabkan sebagian besar keberhasilan telah tercapai dalam permainan papan tradisional seperti *Backgammon*, Catur, *Checkers*, *Othello*, dan *Scrabble*, di mana komputer dapat bermain lebih baik dari pemain terbaik manusia. Keberhasilan tersebut dicapai oleh beberapa peneliti yang ingin menangani masalah yang menantang dengan penekanan pada hasil sistem (Schaeffer, 2000).

2.3. *Game Tree Search*

Salah satu teknik yang biasa digunakan *game* komputer khususnya game yang bersifat *zero-sum* dan *imperfect information* adalah *game tree search*. Dalam *game tree search*, langkah dan hasil dari kondisi permainan dimodelkan dalam bentuk *tree* (van der Kleij, 2010). Sebagai contoh sederhana *game tree* parsial (sebagian) ditampilkan pada Gambar 2.2. untuk permainan *Tic-Tac-Toe*.



Gambar 2.2. *Game Tree* pada *Game Tic-Tac-Toe*

Game tree pada Gambar 2.2. menunjukkan hal berikut ini (Elnaggar, *et al*. 2014) :

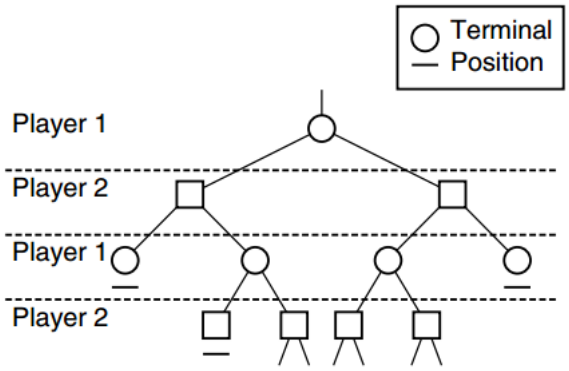
- Setiap *node* merepresentasikan sebuah *state* (keadaan) permainan.
- *Root node* merepresentasikan *state* awal permainan.
- Semua cabang yang diberikan *node* merepresentasikan semua langkah yang diizinkan untuk *node* tersebut.
- Cabang *node* disebut juga dengan *child node* sedangkan *node* yang merupakan asal dari cabang *node* disebut juga dengan *parent node*.
- *Node* yang tidak memiliki turunan disebut *leaf node*.

Berdasarkan ilustrasi pada Gambar 2.2, pemain X melakukan langkah pertamanya, maka dari itu *state* permainan disesuaikan dengan *game tree*. Terdapat lima kotak yang tersedia, maka ada lima langkah yang dapat dilakukan pemain X. Dengan demikian kelima *edges* (garis penghubung) menghubungkan *root* ke *node* lainnya yang mewakili *state* permainan yang dicapai dengan setiap langkah yang dibuat. Pemain X dan pemain O bermain secara bergantian hingga *leaf node*, dimana salah satu pemain mengisi tiga kotak berturut-turut, dicapai. *Leaf node* pada *game tree* parsial ini ditandai dengan "*Win*" atau "*Lose*" sesuai dengan perspektif pemain X (van der Kleij, 2010).

Tic-Tac-Toe merupakan *game* yang berbasis giliran, maka sistem hanya berubah apabila pemain melangkah. Jumlah cabang dari setiap papan sama dengan jumlah

langkah yang pemain dapat lakukan. Dalam beberapa permainan (seperti Halma), ada ratusan hingga ribuan langkah yang mungkin pemain dapat lakukan. Hal ini dikarenakan banyaknya *node* dan jumlah langkah berbeda yang dapat diambil oleh setiap pemain. Beberapa posisi papan dapat mencapai kondisi dimana tidak terdapat lagi langkah yang memungkinkan. Kondisi ini disebut *terminal position* dan menyatakan akhir dari sebuah permainan (Millington & Funge, 2009).

Umumnya, *game tree* dapat digambarkan dalam bentuk abstrak tanpa mengikutkan diagram papan. *Game tree* ini hanya menampilkan nilai yang dimiliki setiap *node* dan simbol-simbol pendukung lainnya.



Gambar 2.3. *Game Tree* dalam Bentuk Abstrak (Millington & Funge, 2009)

2.4. *Monte Carlo Tree Search* (MCTS)

Monte Carlo Tree Search (MCTS) merupakan algoritma pencarian *best-first* yang menggabungkan metode *Monte Carlo* dengan *search tree* (Magnuson, 2015). Metode *Monte Carlo* merupakan metode yang menggunakan simulasi acak secara berulang untuk mendapatkan hasil yang terbaik. Dalam MCTS, simulasi acak digunakan untuk memperluas *game tree*. Kemudian *game tree* digunakan untuk menentukan langkah selanjutnya. Berbeda dengan teknik pencarian klasik lainnya (seperti *Minimax*), metode MCTS tidak memerlukan sebuah fungsi evaluasi posisional tetapi berdasarkan explorasi acak pada ruang pencarian (Chaslot, *et al.* 2008).

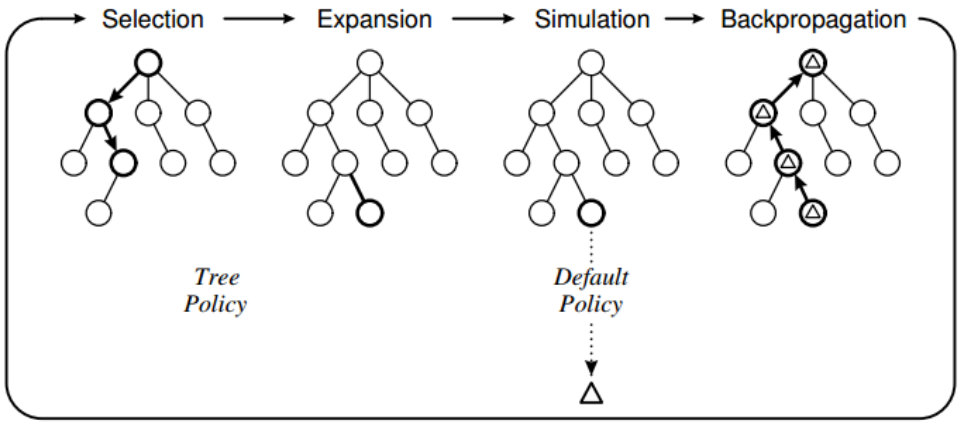
Algoritma MCTS ini telah banyak diterapkan pada beberapa jenis permainan. Menurut Chaslot (2010), sebuah permainan harus memenuhi kondisi berikut sehingga MCTS dapat digunakan :

- Skor permainan dibatasi, artinya ada definisi yang jelas antara menang, kalah, dan seri.
- Informasi permainan diberikan, artinya semua aturan dan keadaan papan diketahui oleh semua pemain.
- Simulasi dibatasi berakhir relatif cepat, dengan kata lain panjang permainan dibatasi.

Algoritma MCTS membangun *search tree* secara *iterative* hinggga batasan yang ditetapkan (berupa waktu, memori, atau jumlah iterasi) tercapai. Sehingga pencarian dihentikan dan mengembalikan hasil terbaik dari *root*. Setiap *node* dalam *tree* merepresentasikan *state* dari domain yang dihubungkan ke *child node* yang merepresentasikan ke tindakan yang mengarah ke *state* berikutnya. Proses pencarian MCTS dilakukan dengan 4 tahap utama yaitu *selection*, *expansion*, *simulation*, dan *backpropagation*. Tahap pencarian algoritma MCTS tersebut dapat dikelompokkan pada 2 *policy* berikut ini (Browne, *et al*. 2012) :

1. *Tree policy*, yaitu strategi dalam memilih dan membuat *leaf node* dari *node-node* yang tersedia pada *game tree* (*selection* dan *expansion*).
2. *Default policy*, yaitu strategi dalam melakukan simulasi dari *node* yang diberikan untuk menghasilkan nilai estimasi (*simulation*)

Tahap *backpropagation* sendiri tidak menggunakan *policy* atau strategi, tetapi memperbaharui statistik *node* untuk menambah informasi dalam *tree policy*.



Gambar 2.4. Proses Pencarian MCTS (Browne, *et al.* 2012)

Pohon pencarian MCTS tumbuh asimetris dan berkembang secara berulang dengan melakukan 4 tahap sebagai berikut (Chaslot, 2010) :

1. *Selection*

Pada tahap *selection*, *tree* ditelusuri mulai dari *root node* dengan menggunakan strategi *selection* di setiap *node* untuk memilih tindakan selanjutnya. Strategi *selection* digunakan untuk menyeimbangkan antara exploitasi *node* dengan nilai estimasi tertinggi dan explorasi *node* dengan nilai perkiraan yang tidak pasti. Exploitasi artinya pencarian *tree* diperdalam ke *child node* yang terbaik. Sedangkan explorasi berarti bahwa lebih banyak *child node* yang digunakan dalam rangka meningkatkan kepastian *child node* yang terbaik.

Dalam penelitian ini, salah satu strategi *selection* yang efektif digunakan adalah algoritma *Upper Confidence Bounds applied to Trees* (UCT). Algoritma UCT menggunakan rumus *Upper Confidence Bounds* (UCB) dalam tahap *selection*. Rumus ini disusun oleh Auer dkk. (2002) dan pertama kali diterapkan pada tahap pemilihan *node* MCTS oleh Kocsis dan Szepesvari (2006). Algoritma UCT memilih node terbaik berdasarkan hasil perhitungan rumus UCB terdapat pada persamaan 2.1.

$$UCB = \frac{W_i}{N_i} + C\sqrt{\frac{\ln N}{N_i}} \qquad (2.1)$$

Dimana : W_i = skor permainan

N_i = jumlah kunjungan *child node i*

C = konstanta explorasi = $\sqrt{2}$

N = jumlah kunjungan *node parent.*

2. *Expansion*

Pada tahap *expansion*, dilakukan penambahan *node* baru ke *search tree* sebagai *child* dari *node* yang ditemukan pada tahap sebelumnya. *Node* ini harus sesuai dengan langkah yang diizinkan untuk dieksplorasi.

3. *Simulation*

Selanjutnya, simulasi atau disebut juga *playout* dilakukan dari *node* yang baru ditambahkan. Simulasi dapat dilakukan secara *pure-random* atau menggunakan beberapa strategi *random* yang sederhana. Strategi simulasi acak dapat menggunakan fungsi *heuristic* untuk memberikan *weight* pada langkah yang paling menjanjikan.

4. *Backpropagation*

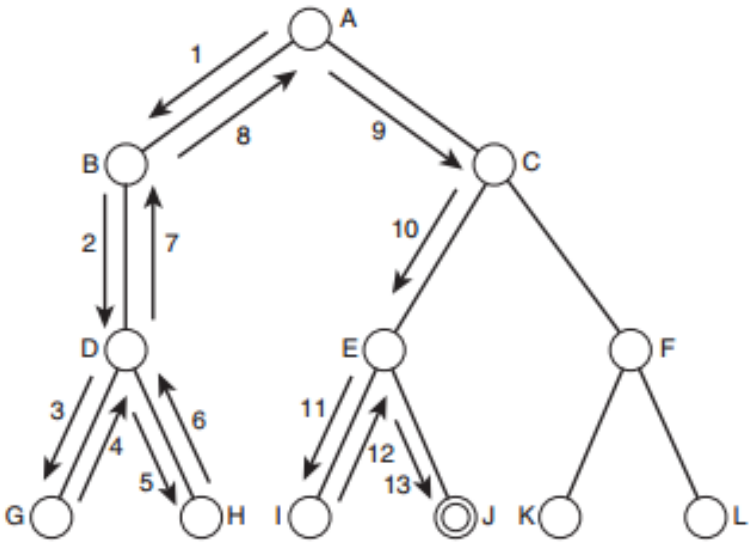
Setelah simulasi mencapai akhir permainan, hasil dari simulasi kemudian diperbarui pada *tree*. Dimulai dari *node* baru yang disimulasikan hingga ke *parent node*. Kunjungan *node* bertambah satu dan rasio menang/kalah diubah sesuai dengan rata-rata hasil yang dikeluarkan.

Setelah proses MCTS dilakukan berulang dan batasan yang ditentukan tercapai, selanjutnya dilakukan adalah pemilihan *final move* dalam permainan sebenarnya. Mekanisme pemilihan *final move* dapat dilakukan dengan 3 cara berikut ini (Chaslot, *et al.* 2007) :

1. *Max Child*, pemilihan *child node* dengan skor tertinggi.
2. *Robust Child*, pemilihan *child node* paling sering dikunjungi
3. *Max-Robust Child*, pemilihan *child node* dengan skor tertinggi dan paling sering dikunjungi.

2.5. *Depth-First Search* (DFS)

Algoritma pencarian yang biasa digunakan adalah algoritma *Depth-First Search*. Disebut *depth-first* karena proses pencarian ini mengikuti jalur terdalam lebih dahulu sebelum pindah ke jalur berikutnya. Teknik pencarian ini termasuk dalam teknik pencarian buta (*blind search*) karena tidak ada informasi yang diberikan pada ruang pencarian (Coppin, 2004). Prinsip algoritma ini diilustrasikan pada Gambar 2.5.



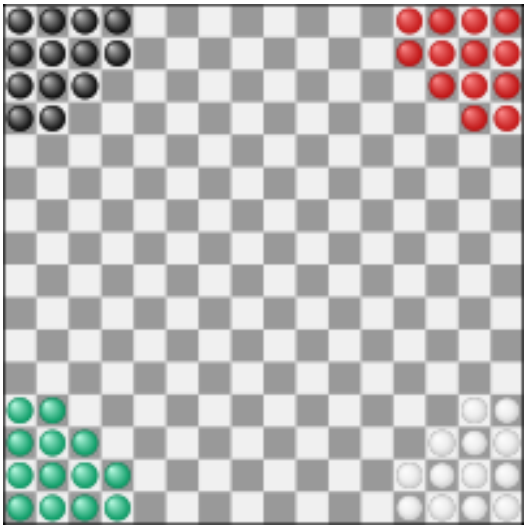
Gambar 2.5. Ilustrasi *Depth-First Search* (Coppin, 2004)

Berdasarkan Gambar 2.5, proses DFS dilakukan pada semua *child node* sebelum dilakukan pencarian ke *node-node* yang selevel. Pencarian dimulai dari *root node* A ke level yang lebih tinggi. Proses ini diulangi terus hingga ditemukan solusi yaitu *node* J. Sehingga urutan penelusuran *node* menjadi A-B-D-G-D-H-D-B-A-C-E-I-E-J.

Keuntungan dari metode DFS ada 2 yaitu tidak membutuhkan memori yang besar dan dapat menemukan solusi tanpa harus menguji lebih banyak lagi ruang keadaan. Sedangkan kelemahan dari metode DFS yaitu memungkinkan tidak ditemukan tujuan yang diharapkan. Selain itu metode ini hanya akan mendapat 1 solusi pada setiap pencarian (Russel & Norvig, 1995).

2.6. Halma

Permainan Halma ditemukan oleh Dr. George Howard Monks diantara tahun 1883 dan 1884. Saudara George, Robert Monks pada tahun 1883 atau 1884 menulis surat kepada saudaranya dan mendeskripsikan permainan Inggris '*Hoppity'*. G.W. Monks mengambil beberapa ketentuan dari *Hoppity* dan membangun Halma. Dr. Thomas Hill sekitar tahun 1818, membantu dalam merancang permainan ini. Dr. Thomas Hill menamakan permainan ini '*Halma'* yang berasal dari bahasa Yunani yang artinya melompat (Walker, 2011). Papan permainan Halma awal ditemukan berbentuk persegi, dapat dilihat pada Gambar 2.6.



Gambar 2.6. Bentuk Permainan Halma

Permainan Halma dapat dimainkan oleh 2 atau 4 pemain. Setiap pemain memiliki jumlah pion 13 buah dengan warna yang berbeda-beda. Objektif dari permainan Halma adalah memindahkan semua pion dari tempat asal ke tempat tujuan yang terletak di seberangnya. Pemain yang dinyatakan menang adalah permain pertama yang berhasil memindahkan semua pion ke daerah tujuan. Aturan cara bermain Halma adalah sebagai berikut (Lieyanda, 2011) :

- Mulai melangkahkan pion dengan arah yang diizinkan secara bergiliran.
- Pion boleh hanya bergerak satu langkah ke kotak kosong ke arah kiri atas, kiri bawah, kanan atas, kanan bawah, samping kanan, samping kiri, atau melompati pion di depannya.
- Lompatan dapat terus dilakukan selama masih ada pion yang dapat dilompati.
- Langkahkan pion menuju ke arah segitiga seberang dan disusun secara tepat agar area segitiga terisi dengan penuh.
- Pemenang adalah yang paling cepat memindahkan semua pion ke segitiga seberang.

2.7. Penelitian Terdahulu

Penelitian pada permainan Halma sebelumnya dilakukan oleh Bell (2009) untuk mencari permainan dengan giliran terpendek menggunakan teknik pencarian *Breadth First Iterative Deepening A**. Hasilnya permainan terpendek dapat dilakukan dalam 30 putaran. Pada penelitian lainnya, Indah (2011) menggunakan algoritma *Depth First Search* (DFS) hanya untuk mendeteksi langkah-langkah yang dibolehkan. Sedangkan, Lieyanda (2011) merancang AI dengan mengimplementasikan algoritma *Greedy* pada permainan Halma berdasarkan lompatan terjauh atau pion belakang. Penggunaan algoritma *Greedy* pada permainan Halma tidak memberikan hasil yang cukup optimal.

Penelitian mengenai penerapan algoritma MCTS pada jenis permainan papan dilakukan oleh Gelly dkk. (2012). Algoritma MCTS berhasil diterapkan pada permainan *Go* sehingga dapat mengalahkan pemain *Go* profesional. Sedangkan pada penelitian lain (Nijssen, 2007), algoritma MCTS cukup dapat bermain pada permainan *Othello*, namun tidak pada level yang lebih tinggi. Penelitian lainnya, menunjukkan algoritma MCTS dapat lebih baik dari algoritma *Alpha-Beta* pada permainan *Lines of Action* (LOA) (Winands, *et al.* 2010) dan *Hex* (Arneson, *et al.* 2010).

Tabel 2.1. Penelitian Terdahulu

No.	Peneliti (Tahun)	Metode dan Kasus	Keterangan
1.	Meilia Nur Indah (2011)	Algoritma DFS pada permainan Halma bentuk bintang.	Algoritma DFS hanya digunakan untuk mendeteksi langkah-langkah yang dibolehkan
2.	Vivi Lieyanda (2011)	Algoritma *Greedy* pada permainan Halma bentuk bintang.	Algoritma *Greedy* tidak memberikan hasil yang cukup optimal.
3.	George I. Bell (2009)	Algoritma *Breadth First Iterative Deepening A** pada permainan Halma dan variannya.	Permainan terpendek dapat dilakukan dalam 30 putaran.
4.	Sylvain Gelly, et al. (2012)	Algoritma MCTS pada permainan *Go.*	Algoritma MCTS dapat mengalahkan pemain *Go* profesional.
5.	Pim Nijssen (2007)	Algoritma MCTS pada permainan *Othello.*	Algoritma MCTS cukup dapat bermain pada permainan *Othello*.
6.	Mark H.M. Winands, *et al*. (2010)	Algoritma MCTS pada permainan *Lines of Action* (LOA).	Algoritma MCTS dapat mengungguli algoritma *Alpha-Beta* terkuat.
7.	Broderick Arneson, *et al.* (2010)	Algoritma MCTS pada permainan *Hex.*	Algoritma MCTS dapat setara dengan algoritma *Alpha-Beta* terbaik

Berbeda dengan penelitian sebelumnya, penulis ingin merancang permainan Halma yang dapat dimainkan oleh satu *user* melawan satu atau tiga AI yang menggunakan algoritma MCTS. Algoritma DFS akan digunakan untuk menentukan jalur langkah pion baik *user* maupun AI. Dari penelitian ini, diharapkan AI dapat menyelesaikan permainan Halma dengan baik.