BAB 2
LANDASAN TEORI

2.1. Artificial Intelligence

Artificial Intelligence adalah ilmu dan teknik pembuatan mesin cerdas, khususnya program komputer cerdas. Hal ini terkait dengan tugas yang sama dengan menggunakan komputer untuk memahami kecerdasan manusia, tetapi *Artificial Intelligence* tidak harus membatasi dirinya terhadap metode yang diamati secara biologis (McCarthy, 2007: 2). Diperkuat dengan pengertian Dobrev (2004: 2) yang mengatakan bahwa *Artificial Intelligence* sebagai pembelajaran bagaimana membuat komputer melakukan hal-hal yang dimana saat ini masih lebih baik dilakukan oleh manusia.

Dari pengertian *Artificial Intelligence* di atas dapat disimpulkan bahwa *Artificial Intelligence* adalah ilmu dan teknik untuk membuat komputer atau mesin dapat berpikir dan bertindak layaknya manusia, bahkan lebih baik daripada yang dapat dilakukan manusia saat ini.

Ada beberapa aplikasi dari *Artificial Intelligence* menurut McCarthy (2007: 10) yaitu :

1. *Game Playing*

   Game playing merupakan mesin yang dapat memainkan master level dari permainan salah satunya catur. Di dalamnya terkandung beberapa *Artificial Intelligence* untuk memainkan permainan tersebut tetapi dengan melalui berbagai perhitungan yang rumit dilihat dari berbagai kemungkinan yang ada. Dapat menentukan pilihan terbaik untuk memenangkan permainan

2. *Speech Recognition*

   Speech recognition mungkin untuk memberikan instruksi pada beberapa komputer dengan menggunakan suara, namun sebagian pengguna masih kembali menggunakan *keyboard* dan *mouse* karena dianggap masih lebih mudah dan nyaman.

3. *Understanding Natural Language*

   Tidak cukup hanya dengan memasukkan urutan kata-kata ke dalam komputer, tetapi komputer juga harus dilengkapi dengan pemahaman tentang ruang lingkup kalimat tersebut. Tujuannya adalah membuat mesin yang mempunyai kemampuan untuk memahami bahasa manusia.

4. *Computer Vision*

   Dalam kehidupan terdapat beberapa benda tiga dimensi, namun dilihat oleh mata maupun komputer secara dua dimensi. Beberapa program dapat bekerja hanya dalam dua dimensi, namun *computer vision* membutuhkan informasi berupa tiga dimensi tidak hanya dalam pandangan dua dimensi. Tetapi *computer vision* tidak sebaik kemampuan manusia.

5. *Expert Systems*

   Expert systems merupakan sistem yang mempelajari ilmu dari seorang pakar yang sesuai dengan sistem yang akan dibuat. Misalnya untuk membuat sistem yang dapat mengenali tentang bakteri yang menginfeksi darah kemudian memberikan saran untuk pengobatannya, maka dibutuhkan ilmu dari dokter agar dapat mengetahui dengan pasti tentang ilmu tersebut.

6. *Heuristic Classification*

   Mengambil keputusan berdasarkan informasi atau data-data yang ada saat ini dan mengklasifikasikannya ke dalam kategori tertentu. Misalnya dalam pengajuan kartu kredit, dilihat dari data-data transaksi pemohon apakah sudah layak untuk diterima surat pengajuannya.

2.2. Sejarah *Artificial Intelligence*

Menurut Buchanan (2005: 53), awal mula *Artificial Intelligence* atau AI dapat dilihat dari sisi filsafat, fiktif, dan imajinasi. Awal perkembangan ilmu dari AI telah banyak mempengaruhi penemuan barang elektronik, mesin, dan lain-lain. Pada dasarnya perkembangan AI oleh ahli melalui proses pembelajaran, representasi pengetahuan, inferensi, demonstrasi program, pembuktian teorema, sampai sistem berbasis pengetahuan.

Secara ilmu filsafat, para filsuf mendefinisikan suatu mesin cerdas yang dahulu dibuat secara eksperimental dan hanya dianggap mungkin secara

teoritis. Namun dalam pertengahan abad yang lalu, para komunitas AI mampu membangun mesin cerdas yang membuktikan hipotesis tentang mekanisme pemikiran dan perilaku cerdas. Para filsuf meyakinkan mesin cerdas sebagai perangkat yang berguna untuk membantu manusia dan digambarkan seperti layaknya manusia. Sebagai contoh Leibniz dan Blaise Pascal merancang mesin untuk berhitung secara logika aritmatika yang dinamakan “kalkulator”.

Sedangkan dari sisi fiktif, para penulis fiksi ilmiah mendefinisikan mesin cerdas digunakan untuk mengembangkan fantasi serta menggambarkan manusia tentang karakteristik manusia itu sendiri. Sebagai contoh, pada tahun 1907 L.Frank Baum mendeskripsikan dan menjelaskan tentang penciptaan sebuah robot, *Perfect-Talking Mechanical Man* dari cara berpikir, berbicara, bertingkah laku, dan melakukan segala sesuatu.

Secara imajinasi publik saat itu, robot dan kecerdasan buatan yang ditangkap sebagai makhluk-makhluk seperti Golem dan Frankstein yang ditakuti.

Pada perkembangan AI selanjutnya, teknik AI dikenalkan pada permainan catur. Permainan catur sendiri jelas membutuhkan pemikiran yang tidak mudah. Mesin cerdas yang dinamakan The Turk merupakan mesin cerdas yang dapat bermain catur dengan baik. The Turk diciptakan oleh Freiherr Joseph Friedrich zu Racknitz yang membuat orang dapat bermain catur dengan mandiri. The Turk dikenalkan pada publik di abad ke-18 dan 19. Oleh karena itu maka catur menjadi suatu media pembelajaran para ahli untuk menentukan mekanisme dan kesimpulan AI.

Setelah mesin cerdas The Turk, perkembangan AI dilanjutkan dengan komputer pada tahun 1940-an yang dianggap sebagai “otak raksasa” karena dapat melakukan perhitungan dan kalkulasi dengan sangat baik. Setelah komputer kemudian diteruskan oleh perkembangan AI di bidang robotika. Walaupun satu pihak menganggap robot merupakan bagian dari komputer cerdas, tetapi pihak lain menilai robot lebih berkaitan dengan teknik mesin dengan kontrol dan tingkah laku. Oleh karena itu, sampai sekarang robot masih menjadi sarana untuk mengembangkan atau menguji ide-ide para ahli AI. Tetapi perkembangan AI tidak hanya berkembang di bidang robotika. Perkembangan komputer juga dapat dilihat dengan signifikan. Dahulu

komputer terbatas dalam melakukan pemrosesan oleh karena ukuran dan kecepatan memori, prosesor, sampai sistem operasi. Seiring dengan perkembangan waktu, *programmer* membuat bahasa pemrograman untuk memanipulasi program seperti LISP, IPL, dan POP juga diiringi dengan kemajuan *hardware* yang memberi kekuatan baru pada tahun 1950-an sampai 1960-an. Kemajuan ini sangat mengesankan oleh karena program mampu memecahkan masalah rumit yang dahulu hanya mampu dilakukan oleh orang-orang cerdas.

Dekade setelah tahun 1960-an inilah yang mengesankan, karena AI telah sangat berkembang dan banyak penemuan-penemuan baru. Proses penemuan baru ini tidak lain karena pembelajaran dari penalaran kasus, analogi, penalaran ketidakpastian, dan penalaran umum. Para peneliti menunjukkan bahwa banyak metode yang dapat dipakai dan perlu diintegrasikan dalam pembuatan mesin cerdas.

2.3. Algoritma

Cormen, Leiserson, Rivest dan Stein (2009 : 5) menyatakan bahwa algoritma merupakan langkah-langkah komputasi yang terdefinisi dengan baik yang mengambil beberapa nilai atau seperangkat nilai sebagai *input* dan menghasilkan beberapa nilai atau seperangkat nilai sebagai *output*. Langkah dalam algoritma mengubah *input* menjadi *output*. Algoritma dapat digunakan sebagai alat untuk memecahkan masalah komputasi. Algoritma dikatakan benar jika setiap *input* yang diberikan akan menghasilkan *output* yang benar dan memecahkan masalah komputasi yang diberikan. Sebuah algoritma dapat dituliskan dalam bahasa Inggris sebagai program komputer atau sebagai desain *hardware*.

Beberapa masalah yang dapat diselesaikan dengan algoritma :

a. Untuk mengidentifikasi gen DNA yang ada pada manusia. DNA memiliki struktur yang sangat banyak dan rumit, sampel DNA tersebut disimpan dalam *database*, lalu dengan menggunakan algoritma akan disesuaikan dengan contoh-contoh DNA yang ada dalam *database*. Dari identifikasi DNA itu dapat digunakan untuk menentukan kelainan pada manusia. Banyak metode yang digunakan untuk menyelesaikan permasalahan

biologis ini. Hasilnya mungkin saja bervariasi bergantung pada metode yang digunakan dan sumber yang ada. Tetapi akan sangat baik apabila sumbernya benar dan akan memudahkan ilmuwan karena menghemat waktu pengecekan dan biaya.

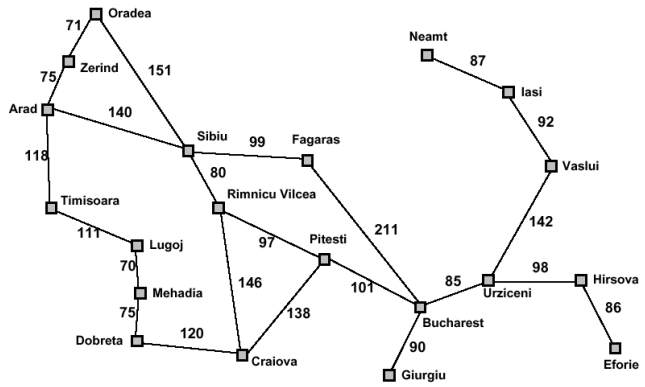
b. Akses internet yang ada sekarang ini memudahkan semua orang untuk mengetahui banyak informasi tempat dan akses yang cepat di seluruh dunia. Dengan menggunakan algoritma yang baik, situs pada internet akan dapat mengatur dan memanipulasi data yang besar. Contoh permasalahan adalah mencari rute terbaik untuk bepergian yang dalam kehidupan sehari-hari diterapkan dalam *global positioning system* atau biasa disebut GPS.
c. *Electronic commerce* sangat menunjang penghasilan dan pelayanan dalam perdagangan saat ini. Hal itu terbukti dari penerapan pada kartu kredit, *password*, dan keamanan bank. Inti dari teknologi yang digunakan pada *electronic commerce* adalah algoritma kriptografi dan *digital signature algorithm*.

Selain dapat menyelesaikan masalah-masalah di atas, algoritma juga sudah banyak diterapkan dalam teknologi. Penerapan algoritma yang tepat akan menghasilkan hasil yang sangat baik, tetapi banyak orang lebih suka menggunakan metode yang mudah untuk dimengerti dan diterapkan sehingga tidak mendapatkan hasil yang maksimal. Selain itu, penerapan algoritma dengan tepat juga akan membuat komputer bekerja lebih cepat dan menghemat penggunaan memori. Waktu komputasi akan menggunakan *space* pada memori, jadi memori harus digunakan secara bijak dengan penerapan algoritma yang efisien karena dapat mengurangi waktu berpikir pada komputer.

2.4. Searching

Dalam *Artificial Intelligence* terdapat beberapa metode pencarian atau *searching*. Menurut Russell dan Norvig (2010: 64) *searching* merupakan proses untuk mencari urutan langkah dalam mencapai tujuan. Ada dua macam cara penyelesaian masalah yaitu *goal formulation* dan *problem formulation. Goal formulation* merupakan penyelesaian masalah untuk

menentukan langkah pertama berdasarkan situasi saat ini dan menggunakan perhitungan. Sebagai contoh dapat dilihat pada gambar 2.1. jika seseorang akan melakukan perjalanan wisata dari Arad menuju Bucharest, dia bisa saja melalui berbagai rute, tetapi dengan adanya *goal formulation* dia dapat membatasi rute agar dapat mencapai tujuannya dengan lebih cepat. Sedangkan *problem formulation* merupakan langkah yang memiliki terlalu banyak ketidakpastian dan tanpa mengetahui tujuannya.



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Gambar 2.1. Rute Perjalanan di Romania

(Sumber: Russell dan Norvig, 2010)

Masalah *searching* secara umum digambarkan dengan lima komponen :

a. Initial State

Mendeskripsikan lokasi awal.

b. Action

Menjelaskan seluruh kemungkinan langkah yang dapat diambil.

c. Transition Model

Mendeskripsikan langkah yang diambil.

d. Goal Test

Menjelaskan kapan langkah yang diambil merupakan tujuan akhir.

e. Path

Menandakan biaya dari suatu *state* ke *state* lain.

2.4.1. Adversarial Search

Russell dan Norvig (2010: 161) menyatakan bahwa *adversarial search* dapat diterapkan dalam permainan. Permainan dengan menggunakan penerapan *Artificial Intelligence* banyak dipelajari karena sangat sulit untuk dipecahkan. Seperti contohnya permainan catur yang memiliki rata-rata 35 cabang dan tiap permainannya sering mencapai 50 langkah untuk tiap pemain. Komputer harus memiliki kemampuan untuk membuat beberapa keputusan yang tepat. Dimulai dengan mencari langkah yang tepat untuk mendapatkan langkah yang baik ketika waktu dibatasi.

Pruning merupakan teknik yang mengabaikan cabang dari *tree* yang dianggap tidak memberikan perubahan yang baik pada hasil akhir pilihan tersebut. Ada pula teknik yang disebut dengan *MAX* dan *MIN* atau *Minimax*. Teknik ini diterapkan untuk dua pemain, *MAX* mengambil langkah pertama dan terus mengambil langkah sampai permainan selesai.

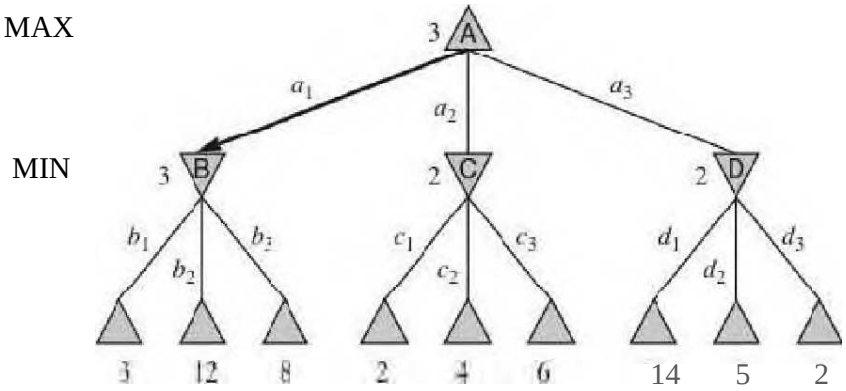
Dalam kehidupan nyata, ada pula permainan yang mencerminkan ketidakpastian dan biasanya hasil yang diperoleh berdasarkan acak, misalnya dengan melempar dadu. Permainan ini disebut dengan permainan stokastik. Contohnya adalah Backgammon, ini merupakan jenis permainan yang mengkombinasikan keberuntungan dan keterampilan dengan melemparkan dadu.

Karena untuk menghitung keputusan yang tepat dalam permainan tidaklah mudah, maka seluruh algoritma harus membuat beberapa asumsi dan perkiraan. Pertama dengan memperkirakan nilai heuristik *minimax* dengan memilih langkah optimal yang diberikan dalam bentuk *tree*. Selanjutnya mempertimbangkan algoritma pencarian yang dapat menghasilkan *tree*. Tujuan dari seorang perancang algoritma adalah untuk menentukan perhitungan yang berjalan cepat dan menghasilkan langkah yang baik.

2.4.1.1. Minimax Method

Menurut Russell dan Norvig (2010: 165) algoritma *minimax* merupakan algoritma yang biasanya diterapkan dalam permainan-permainan yang dimainkan oleh dua orang. Contoh permainan tersebut seperti tic-tac-toe, catur, othello, dan lain-lain. Permainan-permainan itu mempunyai satu persamaan, yaitu mereka merupakan permainan yang menggunakan logika (Pinto, 2002: 1). Permainan tersebut tentunya memiliki aturan, dan dengan aturan itu dapat diketahui nilai yang akan diperoleh masing-masing pemain dan juga peluang jalan lawan yang selanjutnya. Dengan algoritma *minimax* ini berarti dapat memaksimalkan kesempatan komputer dalam permainan dan meminimalkan kesempatan lawan. Pada algoritma ini dijabarkan semua kemungkinan yang ada dalam permainan. Menelusuri seluruh langkah yang mungkin dilakukan oleh lawan dan menentukan langkah selanjutnya yang harus diambil untuk dapat memperoleh hasil yang maksimal dan memenangkan permainan. Begitu seterusnya dari awal permainan sampai permainan berakhir, dan dibuat dalam sebuah *tree*.

Tree merupakan pohon permainan yang dibuat dengan melihat seluruh kemungkinan yang mungkin terjadi dalam permainan. Contoh *tree* dalam algoritma *minimax* dapat dilihat sebagai berikut :



Gambar 2.2. Contoh *Tree Minimax*
(Sumber: Russell dan Norvig, 2010)

Dari gambar 2.2. dapat dilihat bahwa level pertama menandakan level maksimum dan level kedua menandakan level minimum. Bergantian seterusnya untuk level-level selanjutnya. Level maksimum akan mencari nilai tertinggi yang ada pada *node* di bawahnya, sedangkan level minimum mencari nilai terendah pada *node* di bawahnya. Seperti gambar, diberikan tiga nilai pada b1, b2, dan b3 yaitu 3, 12, dan 8. Karena level di atasnya merupakan level minimum, maka dipilih nilai 3 untuk naik menjadi nilai pada *node* B. Pada c1, c2, dan c3 terdapat nilai 2, 4, dan 6, sama seperti sebelumnya, nilai yang dipilih adalah nilai terendah yaitu 2. Begitu pula pada d1, d2, dan d3. *Node* B, C, dan D telah terisi nilai minimum dari masing-masing *child* dari *node* di bawahnya, kemudian ketiganya akan dibandingkan untuk menjadi nilai pada *node* A. Karena level pada *node* A adalah level maksimum, maka yang dicari adalah nilai maksimum dari ketiga *node* di bawahnya. Sehingga 3 menjadi nilai untuk *node* A.

Algoritma *minimax* menampilkan seluruh eksplorasi *depth-first search* secara lengkap dalam *tree* dari sebuah permainan. Jika dimisalkan kedalaman maksimum dari *tree* adalah m dan langkah ke tiap titik adalah b, maka *time complexity* dari algoritma *minimax* adalah $O(b^m)$ dan *space complexity* O(bm). Dalam algoritma *minimax* dapat dilakukan optimisasi. Optimisasi yang paling dasar adalah dengan membatasi kedalaman atau level dari *tree* yang akan ditelusuri (Pinto, 2002: 2). Hal itu dilakukan karena untuk menghasilkan *tree* secara *full* akan memakan waktu yang sangat lama. Seperti permainan catur yang memiliki banyak cabang dalam *tree* untuk ditelusuri. Pembuatan *tree* tersebut dapat menghabiskan banyak memori. Sehingga perlu dilakukan optimisasi dengan membatasi kedalaman atau level dalam *tree*.

2.4.2. Informed (Heuristic) Search Strategies

Menurut Russell dan Norvig (2010 : 92*) informed search strategy* merupakan satu strategi pencarian yang menggunakan pengetahuan masalah yang spesifik di luar dari definisi masalah itu

sendiri dan bisa menemukan solusi yang lebih efisien dibandingkan dengan *uninformed strategy*. *Uninformed strategy* disebut juga pencarian buta karena dalam strategi pencariannya tidak mempunyai tambahan informasi atau *cost* yang diperlukan dalam memecahkan masalah dan yang dilakukan adalah melakukan pengecekan pada semua *node*. Pendekatan pada *informed search strategies* yang umum digunakan adalah *Best First Search* atau biasa disebut dengan *greedy method.*

2.4.2.1. Greedy Method

Menurut Russell dan Norvig (2010: 92) *greedy* merupakan nama lain dari *Best First Search. Greedy* dapat diartikan rakus atau serakah. *Greedy* memperluas *node* yang paling dekat dengan tujuan, dengan alasan bahwa hal ini mungkin menyebabkan solusi yang cepat dalam melakukan pencarian. Algoritma *greedy* ini mengevaluasi *node* dengan fungsi *heuristic* :

$$f(n) = h(n)$$

dimana h (n) merupakan estimasi biaya dari n ke tujuan

Ada beberapa elemen dalam algoritma *greedy* (Ichsan, 2010) :

1. Himpunan kandidat

   Himpunan yang berisi kemungkinan-kemungkinan yang bisa menjadi solusi dari permasalahan.
2. Himpunan solusi

   Himpunan yang berisi kandidat yang telah dipilih sebagai solusi.
3. Fungsi seleksi

   Fungsi untuk melakukan seleksi terhadap kandidat agar menghasilkan solusi yang diharapkan.
4. Fungsi kelayakan

   Fungsi untuk memastikan bahwa solusi yang dipilih memenuhi syarat.
5. Fungsi obyektif

   Memilih solusi paling optimal dari himpunan solusi.

Algoritma *greedy* membangun langkah per langkah, dan selalu memilih langkah selanjutnya yang menawarkan solusi terbaik (Dasgupta, Papadimitriou, & Vazirani, 2006: 139). Pada setiap langkah algoritma *greedy* mengambil pilihan terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depannya, sebagaimana prinsip algoritma *greedy*. Dengan demikian, secara umum algoritma *greedy* bekerja dengan cara melakukan pencarian himpunan kandidat dengan fungsi seleksi dan diverifikasi dengan fungsi kelayakan kemudian memilih solusi paling optimal dengan fungsi obyekif.

Sebagai contoh untuk penerapan algoritma *greedy* pada pencarian rute di Romania, dari Arad menuju Bucharest dengan menggunakan fungsi *heuristic straight line distance* atau yang biasa disebut $h_{\mathbf{SLD}}$.

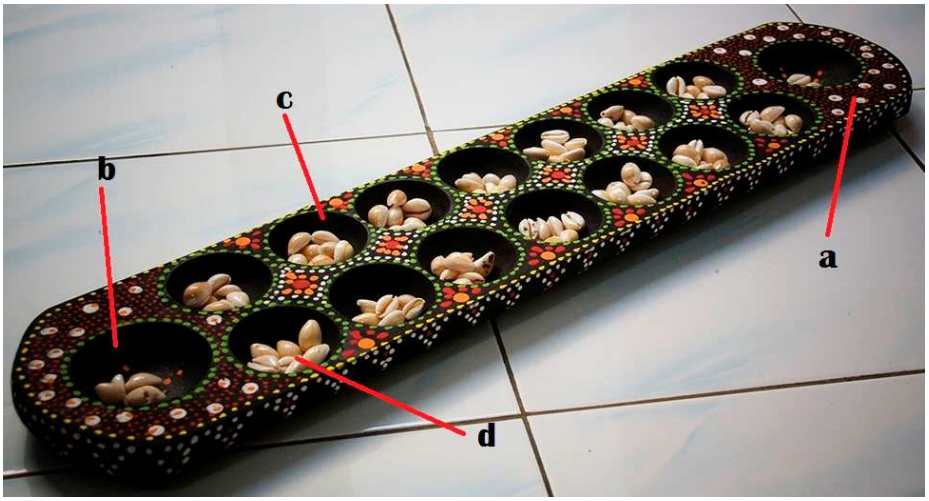
Contoh penerapan algoritma *greedy* dapat dilihat dari gambar 2.1. pada halaman 13. Diketahui bahwa jarak dari Arad menuju Bucharest adalah 366 atau dapat dituliskan $h_{\mathbf{SLD}}(\ln(\text{Arad})) = 366$. Kemudian dari Arad menuju ke Sibiu, karena dapat dilihat bahwa jarak dari Sibiu ke Bucharest lebih dekat dibandingkan dengan Timisoara dan Zerind. *Node* selanjutnya yang akan dikunjungi adalah Fagaras, karena Fagaras lebih dekat dan langsung dapat menuju ke Bucharest yang merupakan tujuan akhir dibandingkan dengan Rimnicu Vilcea yang memiliki jarak yang lebih jauh dan harus melewati tempat lain sebelum sampai ke Bucharest. Algoritma *greedy* mencari solusi yang paling cepat sampai ke tujuan. Jika dilihat dari contoh di atas dapat diketahui bahwa jarak dari Fagaras ke Bucharest lebih jauh 32 KM dibandingkan jika melalui Rimnicu Vilcea dan Pitesti. Ini menunjukkan mengapa algoritma ini disebut *greedy* yang tiap langkahnya mencoba untuk mencari solusi tercepat untuk sampai ke tujuan tanpa mempertimbangkan kemungkinan selanjutnya.

2.5. Permainan Tradisional

Menurut Semiawan (2008: 22) permainan tradisional adalah suatu jenis permainan yang ada pada satu daerah tertentu yang berdasarkan kepada budaya daerah tersebut. Biasanya dimainkan oleh orang-orang pada daerah tertentu dengan aturan dan konsep yang tradisional pada jaman dulu. Permainan tradisional juga dikenal sebagai permainan rakyat yang merupakan sebuah kegiatan rekreatif yang tidak hanya bertujuan untuk menghibur diri, tetapi juga sebagai sarana untuk memelihara hubungan dan kenyamanan sosial. Jadi bermain bagi anak mempunyai nilai dan ciri yang penting dalam kemajuan perkembangan kehidupan sehari-hari termasuk dalam permainan tradisional.

2.5.1. Congklak

Congklak adalah suatu permainan tradisional Melayu yang dikenal dengan berbagai macam nama di seluruh Indonesia. Dalam permainan Congklak, terdapat dua bahan yaitu papan Congklak (a) dan buah Congklak (d). Dapat dilihat pada gambar 2.3. papan Congklak berbentuk sampan yang dibuat dari berbagai jenis kayu dan terdapat tujuh lubang satu baris yang disebut sebagai 'kampung' (c) dan di kedua ujungnya terdapat lubang ibu yang disebut sebagai 'rumah' (b). Panjang papan Congklak tujuh lubang biasanya sekitar 80 cm dan lebarnya 18 cm. Dan untuk buah Congklak berjumlah 98 (14x7) buah yang menggunakan sejenis cangkang kerang dan kadang kala digunakan juga biji-bijian dari tumbuh-tumbuhan dan batu-batuan.



Gambar 2.3. Alat Permainan Congklak

Permainan Congklak dilakukan oleh dua orang. Langkah-langkah permainan Congklak adalah sebagai berikut :

a. Pada awal permainan setiap lubang kecil diisi dengan tujuh buah biji.
b. Dua orang pemain berhadapan, salah seorang yang memulai dapat memilih lubang berisi biji yang akan diambil.
c. Pemain mengambil seluruh biji dalam lubang yang dipilih kemudian meletakkan satu per satu biji ke lubang di sebelah kirinya atau searah jarum jam dan seterusnya sampai biji yang berada di tangan pemain habis diletakkan.
d. Setelah biji pada tangan pemain habis, terdapat empat kondisi :
    i. Jika biji habis di lubang kecil yang berisi biji lainnya, ia dapat mengambil biji-biji tersebut dan melanjutkan mengisi seperti pada langkah c.
    ii. Jika biji habis di lubang besar miliknya maka ia dapat melanjutkan dengan memilih lubang kecil di sisinya.
    iii. Jika biji habis di lubang kecil di sisinya maka ia berhenti dan mengambil seluruh biji di sisi yang berhadapan.
    iv. Jika biji habis dan berhenti di lubang kosong di sisi lawan maka ia berhenti dan tidak mendapatkan apa-apa.

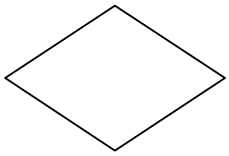
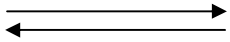
e. Permainan dianggap selesai bila sudah tidak ada biji lagi yang dapat diambil (seluruh biji ada di lubang besar kedua pemain). Pemenangnya adalah yang mendapatkan biji terbanyak (Fourtofour, 2013).

2.6. Flowchart

Flowchart menurut Robertson (2006: 264) yaitu sebuah metode alternatif untuk merepresentasikan algoritma. *Flowchart* banyak digunakan karena secara grafis menggambarkan logika dari sebuah program dengan serangkaian simbol standar geometris dan garis yang saling menghubungkan simbol tersebut. *Flowchart* mudah untuk dipelajari dan sebuah metode yang menggambarkan aliran dari sebuah algoritma. Ada beberapa simbol dari *flowchart* yang dijelaskan pada tabel 2.1 di bawah ini :

Tabel 2.1. Simbol-Simbol *Flowchart*

Simbol	Nama	Fungsi
	*Terminal symbol*	*Terminal* mengindikasikan sebuah *flowchart* mulai dan berakhir. Tiap *flowchart* harus diawali dan diakhiri dengan *terminal symbol.*
	*Input / Output symbol*	*Input / output symbol* menggambarkan sebuah proses *input* atau *output* dalam algoritma, seperti membaca *input* atau mencetak *output*.

Simbol	Nama	Fungsi
	*Process symbol*	*Process symbol* menggambarkan semua proses yang ada dalam algoritma, seperti menetapkan nilai atau melakukan perhitungan. Dibuat secara berurutan.
	*Predefined process symbol*	Menggambarkan modul dalam sebuah algoritma dan berhubungan dengan *flowchart* lain.
	*Decision symbol*	Menggambarkan sebuah keputusan dalam algoritma yang meliputi perbandingan dua nilai. Jalur alternatif yang diambil tergantung pada kondisi yang berada pada simbol benar atau salah.
	*Flowlines*	Menghubungkan berbagai simbol dalam *flowchart.*

2.7. HTML

Pada tahun 1991 Tim Berners-Lee membuat sebuah cara untuk menghubungkan komputer satu dengan komputer lain yang disebut dengan *World Wide Web*. Dengan ditemukannya *web*, Tim Berners-Lee menentukan bahasa yang digunakan untuk menampilkan struktur dari halaman *web* yang disebut dengan *Hypertext Markup Language* (HTML). HTML merupakan

suatu format data standar yang digunakan untuk membuat dokumen *hypertext* agar dapat dibaca dari suatu *platform* ke *platform* lainnya, tanpa melakukan perubahan (Turban, Rainer dan Potter, 2006: 680). *File* HTML umumnya memiliki akhiran *.htm atau *.html. *Tag-tag* pada HTML selalu diawali dengan <x>...</x>, dimana x *tag* HTML seperti , , , dan lain-lain. Di bawah ini adalah tabel beberapa *tag* HTML yang sering digunakan (Astamal, 2008: 1).

Tabel 2.2. Tabel *Tag* HTML

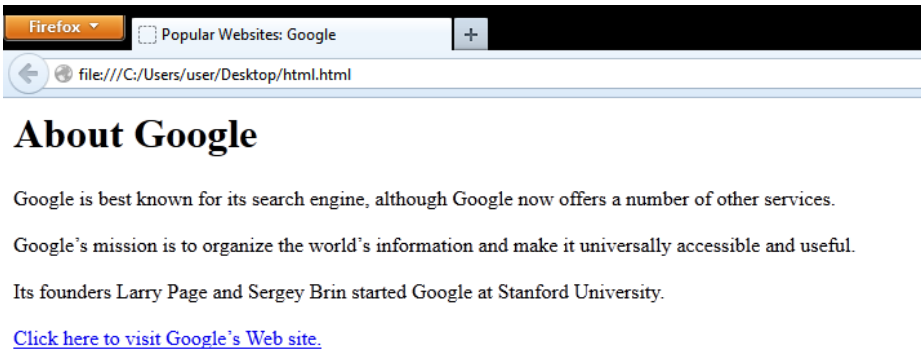
Tag HTML	**Keterangan**
<html></html>	*Tag* untuk mengapit halaman HTML
<head></head>	*Tag* yang berisi informasi umum dari halaman
<title></title>	Judul halaman, *tag* tersebut harus berada dalam *tag* <head>..</head>
<body></body>	Halaman yang akan ditampilkan pada *browser*
<style></style>	Untuk CSS dan *tag* tersebut harus berada dalam *tag* <head>..</head>
	Untuk menebalkan teks
	Untuk membuat *layer*
<a>	Untuk membuat *hyperlink*
	Untuk membuat paragraf
<hn></hn>	n dapat berupa angka dari 1-5, contoh .. untuk membuat *header*
	Untuk *inline style* (manipulasi teks)
<!-- -->	Untuk menulis komentar
<tr></tr>	Untuk membuat baris baru pada tabel
<th></th>	Untuk membuat *header* tabel dan harus berada dalam *tag*<tr>..</tr>
<td></td>	Untuk membuat kolom pada tabel

Contoh *code* HTML dapat dilihat sebagai berikut (Duckett, 2010: 7) :

```
<html>
<head>
    <title>Popular Websites: Google</title>
</head>
<body>
        <h1>About Google</h1>
        <p>Google is best known for its search engine, although Google now offers a number of other services.</p>
        <p>Google's mission is to organize the world's information and make it universally accessible and useful.</p>
        <p>Its founders Larry Page and Sergey Brin started Google at Stanford University.</p>
        <p><a href="http://www.Google.com/">Click here to visit Google's Web site.</a></p>
    </body>
</html>
```

Gambar 2.4. *Code* HTML

Hasil dari *code* HTML di atas adalah sebagai berikut :



Gambar 2.5. Hasil *Code* HTML

Dari gambar 2.5. di atas dapat dilihat terdapat judul *web* yaitu 'Popular Websites: Google' dengan *header* 'About Google' dan konten yang ditampilkan, serta terdapat *hyperlink* yang menuju ke halaman *web* Google. Dalam HTML juga dapat menampilkan *textbox, checkbox, radiobutton,* dan lainnya dengan menggunakan *tag* <input>. Yang membedakan *output* dari

masing-masing tampilan adalah nilai dari atribut *type*. Contoh berikut merupakan penggunaan *tag input* untuk menampilkan *textbox* :

<input type=”text” size=”16” maxlength=”16”>

Berikut ini daftar nilai yang dapat digunakan pada atribut *type* (Astamal, 2008: 7) :

Tabel 2.3. Tabel Atribut *Type* pada HTML

Nilai *Type*	**Keterangan**
Text	Untuk menampilkan *textbox*
Password	Untuk menampilkan *password field*
File	Untuk menampilkan proses *upload file* (mirip seperti *textbox* namun dengan tombol *Browse*)
Checkbox	Untuk menampilkan tombol *checkbox* (lebih dari satu pilihan)
Radio	Untuk menampilkan tombol *radio*/*option* (hanya satu pilihan)
Submit	Tombol untuk men-*submit form*
Reset	Untuk membersihkan tampilan *form*
Hidden	*Input* tidak ditampilkan di *browser*

2.8. Javascript

Javascript adalah sebuah bahasa pemrograman yang dapat digunakan untuk menambah interaksi di dalam halaman *web* (Negrino dan Smith, 2001: 2). Program Javascript ditulis menggunakan *Unicode character set*. Salah satu karakteristik dari Javascript adalah *case-sensitive* karena seluruh *keywords*, variabel, nama fungsi, dan *identifier* lainnya harus ditulis dengan konsisten menggunakan huruf kapital atau tidak (Flanagan, 2012: 1).

Menurut Hardjono (2006) keuntungan dari Javascript antara lain :

a. Javascript berjalan di sisi klien

   Browser memproses *script* sehingga menghemat *bandwidth*.

b. Javascript merupakan bahasa yang relatif mudah

   Javascript mudah dipelajari karena sintaks yang digunakan menggunakan bahasa Inggris yang umum dan juga menyediakan banyak fungsi siap pakai.

c. Javascript relatif cepat untuk *end user*

   Javascript lebih cepat karena tidak perlu diproses pada *web server*, dikirim kembali dan diproses pada *browser*.

Secara sederhana Javascript diapit oleh *tag* <script></script>, namun untuk lebih memperjelas penggunaan Javascript biasanya ditambahkan atribut *language* atau *type* (Astamal, 2008: 22).

Contoh dari *code* Javascript dapat dilihat sebagai berikut (Duckett, 2010: 562) :

```
1  <html>
2  <head><title>Select whole text area</title>
3  <script language="javascript">
4      function selectAll(strControl){
5          strControl.focus();
6          strControl.select();
7      }
8  </script>
9  </head>
10 <body>
11     <form name="myForm">
12         <textarea name="myTextArea" rows="5" cols="20">This is some text</textarea>
13         <input type="button" name="btnSelectAll" value="Select all" onclick="selectAll(document.myForm.myTextArea);" />
14     </form>
15 </body>
16 </html>
```

Gambar 2.6. *Code* Javascript

Dari gambar di atas, pada baris ketiga terdapat *tag*<script language = “javascript”> yang maksudnya dalam *code* HTML tersebut menyisipkan bahasa Javascript. Baris keempat sampai tujuh merupakan contoh penggunaan bahasa Javascript yaitu fungsi focus() dan select() yang berfungsi mengijinkan *user* untuk memilih seluruh konten yang terdapat pada *textarea* sehingga *user* tidak harus secara manual memilih seluruh konten menggunakan *mouse*. Baris sebelas *tag*<form> untuk menandakan suatu formulir pada *code* HTML. Baris ke 12 *tag* <textarea> digunakan untuk

membuat suatu area tulisan yang biasanya digunakan untuk memuat kalimat panjang, Contoh penggunaan *textarea* pada halaman *form* seperti alamat. Baris ke 13 yaitu *tag* <input type=”button”> untuk membuat tombol yang memberi perintah sesuai dengan isi dari parameter yang terdapat pada atribut *onclick*. Atribut *onclick* biasanya berisikan perintah Javascript.

2.9. CSS

CSS adalah kepanjangan dari *Cascading Style Sheets* yang menurut Cahyono (2008) yaitu sebuah halaman terpisah dari halaman *web* yang dipergunakan untuk pengaturan komponen *style* seperti *font*, warna, *layout* dan sebagainya. Beberapa keuntungan dari penggunaan CSS :

a. Memungkinkan mendapatkan *file* yang kecil karena pengaturan *style* oleh CSS dibuat secara terpisah dan di*import* ke dalam *file* utama.
b. Kecepatan akses jauh lebih cepat.
c. Lebih mudah mengontrol *style* dari seluruh halaman *website* karena hanya perlu mengubah satu halaman untuk mengubah seluruh *style* dari sebuah *website*.

Dalam penulisan CSS terdiri dari :

a. Selector

Menghubungkan bagian dari HTML yang akan diterapkan CSS. Terdapat dua tipe dari *selector* yaitu *CSS class* dapat dilihat pada gambar 2.8. dan*CSS id* pada gambar 2.9.

b. Declaration Block

Konten yang terdapat dalam batasan kurung kurawal “{“ dan “}”

c. Declaration

Konten yang terdapat dalam *declaration block*. Tiap deklarasi merupakan kombinasi dari *property* dan *value*

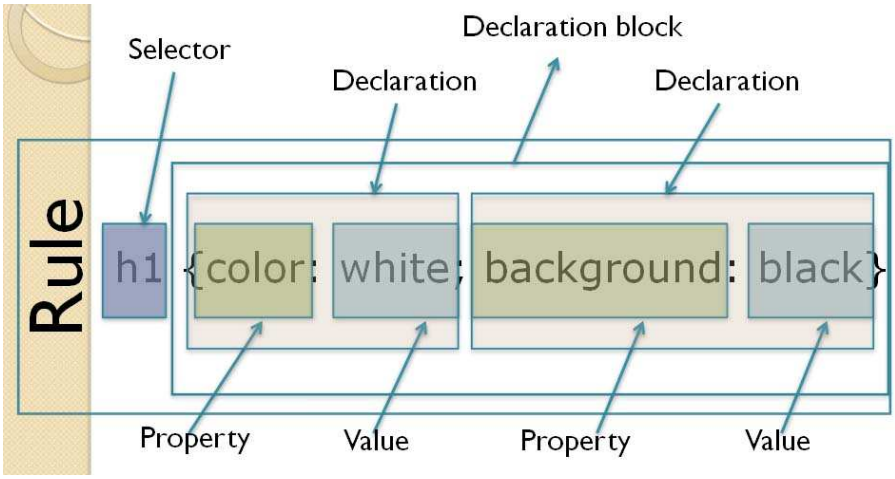
d. Property

Menentukan karakteristik suatu bagian yang akan diubah seperti warna, ukuran tulisan, posisi, dan lain-lain

e. Value

Nilai dari *property*.

Contoh struktur penulisan CSS digambarkan pada gambar 2.7. di bawah ini :



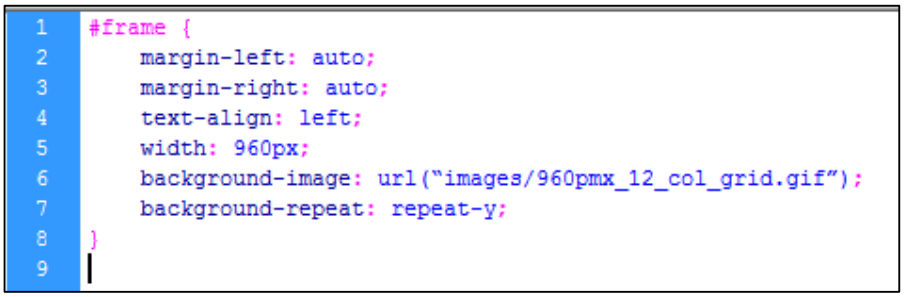
Gambar 2.7. Struktur Penulisan CSS

(Sumber: http://library.rice.edu/)

```
1   div.header {
2       position: fixed;
3       top: 0px;
4       left: 0px;
5       width: 100%;
6       padding: 20px;
7       font-size: 28px;
8       color: #ffffff;
9       background-color: #666666;
10      border-style: solid;
11      border-width: 2px;
12      border-color: #000000;
13  }
```

Gambar 2.8.Contoh *CSS Class*

Contoh dari *code* CSS adalah sebagai berikut (Duckett, 2010: 346, 396):



```
1  #frame {
2      margin-left: auto;
3      margin-right: auto;
4      text-align: left;
5      width: 960px;
6      background-image: url("images/960pmx_12_col_grid.gif");
7      background-repeat: repeat-y;
8  }
9
```

Gambar 2.9. Contoh *CSS id*

Pada gambar 2.8. dapat dilihat contoh *CSS class*. Baris pertama, kata *header* merupakan nama *class* dari HTML yang akan diterapkan CSS. Baris kedua yaitu *position* merupakan atribut pada CSS yang berguna untuk mengatur posisi objek. Baris ketiga dan keempat, atribut *top* dan *left* berguna untuk mengatur posisi objek pada layar. Selain itu juga terdapat atribut *bottom* dan *right*. Baris kelima yaitu *width* merupakan atribut pada CSS untuk membuat lebar objek sebesar 100% atau sama seperti lebar layar. Baris keenam yaitu *padding* berguna untuk memberikan jarak konten terhadap objek. Baris ketujuh yaitu *font-size* untuk menentukan ukuran tulisan pada konten. Baris kedelapan yaitu *color* untuk menentukan warna tulisan pada konten. Baris kesembilan merupakan *background-color* untuk menentukan warna latar suatu konten. Baris kesepuluh yaitu *border-style* untuk menentukan tipe bingkai yang digunakan. Baris 11 yaitu *border-width* untuk menentukan ketebalan bingkai. Dan baris 12 yaitu *border-color* untuk menentukan warna bingkai.

Gambar 2.9. merupakan contoh *CSS id*. Baris pertama, *frame* merupakan nama *id* dari HTML yang akan diterapkan CSS. Baris kedua dan ketiga yaitu *margin* yang berguna untuk memberikan jarak antara objek dengan objek atau konten dengan konten. Atribut *margin* juga bisa dispesifikasikan seperti *margin-top*, *margin-bottom*, *margin-left*, dan *margin-right*. Baris keempat yaitu *text-align* digunakan untuk mengatur tulisan lebih rapi. Baris keenam yaitu *background-image* untuk memberikan gambar pada latar. Dan baris ketujuh merupakan *background-repeat* untuk menyalin gambar dan disusun secara vertikal atau horisontal.

2.10. Related Works

2.10.1. Dots-and-Boxes AI Algorithm

O'Flaherty dan Wu (2012) membahas penerapan *Artificial Intelligence* dalam Dots-and-Boxes. Tujuannya adalah untuk melakukan analisis dan perbandingan beberapa algoritma dalam AI yang dapat bermain melawan manusia. Algoritma yang digunakan yaitu *random*, *greedy*, *minimax*, dan *alpha-beta*. Dari algoritma yang digunakan, maka dilakukan *testing* dengan *game boards* 4x4, 5x5, 6x6 dan 7x7 dari 100 percobaan. *Alpha-beta* vs *greedy* tidak ditampilkan karena *alpha-beta* memiliki hasil yang tidak jauh berbeda dengan *minimax* dan hampir selalu mengalahkan *greedy*. Hasil yang diperoleh dari pengujian yang dilakukan adalah sebagai berikut :

Tabel 2.4. *Random* vs *Greedy*

Algorithm	4x4	5x5	6x6	7x7
Random	0	0	0	0
Greedy	100	100	100	100

Tabel 2.5. *Greedy* vs *Minimax*

Algorithm	4x4	5x5	6x6	7x7
Greedy	8	0	0	0
Minimax	92	99	98	99

Tabel 2.6. *Minimax* vs *Alpha-beta*

Algorithm	4x4	5x5	6x6	7x7
Minimax	45	35	4	2
Alpha-beta	54	63	94	97

Dari tabel 2.4. dapat dilihat pengujian dengan algoritma *random* melawan *greedy*. Hasilnya 100% dimenangkan oleh algoritma pada tiap pengujian yang dilakukan, baik *game boards* 4x4, 5x5, 6x6 dan 7x7. Pada tabel 2.5. ditampilkan hasil pengujian algoritma *greedy* melawan *minimax*. Dari pengujian *game boards* 4x4 sampai dengan *game boards* 7x7 dapat dilihat bahwa algoritma *minimax* selalu menang melawan *greedy* dengan perbedaan hasil yang sangat jauh, bahkan pada *game boards* 5x5 sampai 7x7 algoritma *minimax* menang 100%. Tabel 2.6. menampilkan hasil pengujian dari algoritma *minimax* melawan *alpha-beta*. Hasilnya pada *game boards* 4x4 dan 5x5 dimenangkan oleh *alpha-beta* namun hasil yang dicapai tidak terlalu berbeda jauh, sedangkan untuk *game boards* 6x6 dan 7x7 dapat dilihat perbedaan hasil yang sangat besar dan dimenangkan oleh *alpha-beta*. Dari beberapa pengujian yang telah dilakukan dapat dilihat bahwa algoritma terbaik yang dapat diterapkan pada permainan Dots-and-Boxes ini adalah algoritma *alpha-beta*. Jika *alpha-beta* melawan manusia pada *games board* 5x5 maka kemungkinan *alpha-beta* untuk menang sebesar 90%. Selain itu, jika pemain jalan terlebih dahulu dengan kemungkinan menang-kalah kira-kira 50-50, maka kemungkinan menang dari pemain yang jalan pertama adalah 94%.

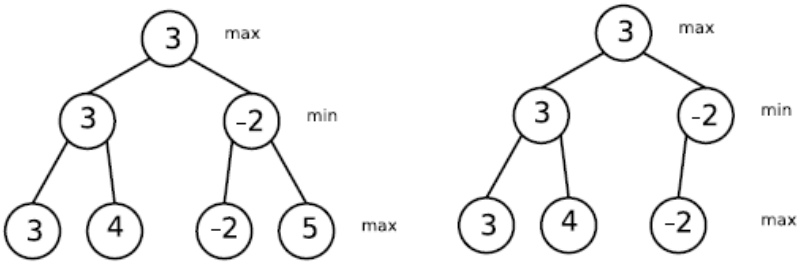
2.10.2. Searching&Game Playing:An Artificial Intelligence Approach to Mancala

Gifford, Bley, Ajayi, dan Thompson (2008) sebelumnya pernah membuat *technical report* yang membahas tentang penerapan *Artificial Intelligence* dalam permainan Congklak dengan algoritma

minimax dan *alpha-beta pruning*. Tujuan dari penelitian ini yaitu untuk menerapkan, menguji dan menganalisis permainan Mancala dengan menggunakan berbagai teknik AI pada permainannya. Pengujian dilihat dengan menganalisis perbedaan nilai heuristik dan melakukan percobaan ratusan kali. Nilai heuristik ini didapat dari langkah yang mungkin, *score* pada *home base player* dan *opponent*, kemungkinan *player* menang, kemungkinan *opponent* menang, dan lokasi biji. Namun sebenarnya nilai heuristik bisa didapat dari berbagai macam sisi atau menurut pandangan pribadi. Dari pengujian tersebut dapat ditemukan pola dalam statistik berupa titik menuju sukses atau gagal dari kombinasi heuristik yang dibuat.

2.10.3. Playing Othello with Artificial Intelligence

Korman (2003) menerapkan algoritma *alpha-beta pruning* pada Othello. Tujuannya untuk menjelaskan kerangka berpikir secara umum untuk AI *game playing* dan algoritma yang paling optimal pada permainan Othello. Awalnya penerapan *Artificial Intelligence* pada Othello ini menggunakan algoritma *minimax*, tetapi setelah dipelajari lebih lanjut terdapat suatu permasalahan utama dari algoritma *minimax* yaitu waktu pencarian. Algoritma *minimax* membutuhkan waktu yang cukup lama dalam waktu pencarian karena mewajibkan untuk membentuk *tree* secara keseluruhan, oleh karena itu, algoritma *minimax* dikombinasikan dengan *alpha-beta pruning* yang dapat memangkas atau mempersingkat waktu pencarian menjadi lebih efisien.

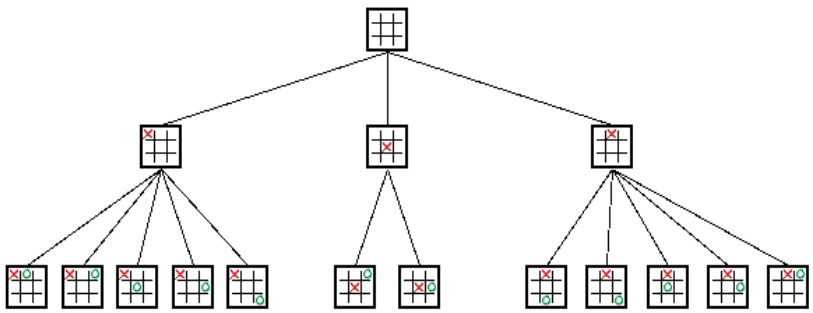


Gambar 2.10. *Tree Minimax* dengan *Alpha-Beta Pruning*
(Sumber: Korman, 2003)

Dari gambar 2.10. dapat dilihat contoh *tree minimax* dengan *alpha-betapruning* yang memangkas *node* dengan nilai lima.

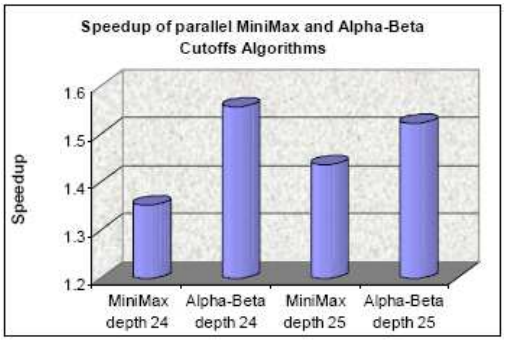
2.10.4. Efficiency of Parallel Minimax Algorithm for Game Tree Search

Borovska dan Lazarova (2007) menggunakan permainan Tic-Tac-Toe sebagai contoh untuk penerapan algoritma *paralel minimax* dan *alpha-beta pruning*. Tujuannya untuk menyelidiki tingkat efisiensi dari algoritma *minimax* pada *game* pencarian. Dengan menggunakan *game tree* yang digambarkan pada gambar 2.11. dapat dilihat kombinasi yang mungkin untuk mengambil keputusan pada langkah pertama dan kedua.



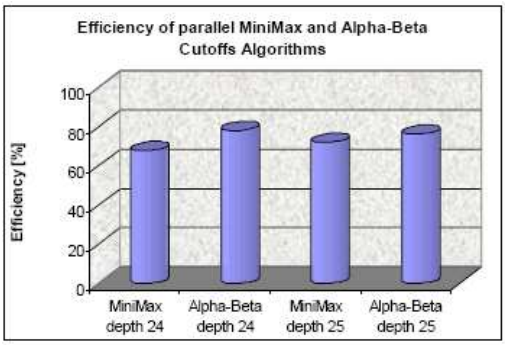
Gambar 2.11. *Game Tree* dari Pergerakan Pertama dan Kedua
(Sumber: Borovska dan Lazarova, 2007)

Dari *testing* yang dijalankan pada *processor* Intel Core 2 Duo, diperoleh hasil untuk kecepatan yang ditampilkan pada gambar 2.12. dan hasil uji efisiensi pada gambar 2.13.



Gambar 2.12. Speedup Test

(Sumber: Borovska dan Lazarova, 2007)



Gambar 2.13. Efficiency Test

(Sumber: Borovska dan Lazarova, 2007)

Dari gambar 2.12. dapat dilihat bahwa algoritma *alpha-beta pruning* dengan kedalaman *leve* 24 memiliki kecepatan yang paling tinggi dibandingkan dengan algoritma *minimax* kedalaman *leve* 24 dan 25, maupun dengan algoritma *alpha-beta pruning* dengan kedalaman yang lebih yaitu *leve* 25. Sedangkan untuk gambar 2.13. merupakan hasil dari uji efisiensi kedua algoritma. Algoritma *alpha-beta pruning* dengan kedalaman *leve* 24 juga memiliki tingkat

efisiensi yang tinggi dibandingkan dengan algoritma *minimax* level 24 dan 25, serta *alpha-beta pruning leve* 25.

2.10.5. Implementasi Algoritma *Greedy Best First Search* pada Aplikasi Permainan Congklak untuk Optimisasi Pemilihan Lubang dengan Pola Berpikir Dinamis

Hermawan (2012) membahas tentang implementasi algoritma *greedy Best First Search* pada aplikasi permainan Congklak untuk optimasi pemilihan lubang dengan pola berpikir dinamis. Dengan metode pola berpikir dinamis, algoritma akhir yang digunakan untuk kombinasi algoritma *greedy* dan penyelesaian masalah secara dinamis pada satu putaran permainan adalah sebagai berikut:

1. Melakukan pengecekan nilai f(n) dari semua lubang kecil di areanya
2. Mencari nilai f(n) yang tertinggi
3. Jika nilai f(n) <= 5 maka mencari lubang yang mempunyai biji maksimum. Jika jumlah biji di seberang lubang dengan nilai maksimum = 0, maka ambil lubang tersebut. Jika tidak, maka ambil lubang dengan nilai maksimum yang terakhir ditemukan
4. Jika nilai f(n) > 5 maka dilakukan pengecekan terlebih dahulu, jika terdapat lebih dari satu lubang yang memiliki nilai f(n) sama, maka mencari lubang yang biji terakhirnya jatuh pada lubang besar. Jika ditemukan, maka ambil lubang tersebut. Jika tidak ditemukan, maka ambil lubang dengan nilai f(n) terbesar yang pertama kali ditemukan.

   Dari pengujian tersebutdisimpulkan bahwa algoritma *greedy* dengan kombinasi cara berpikir dinamis walaupun belum tentu optimal tetapi dapat membuat pemain komputer memutuskan kapan harus menyelamatkan biji lubangnya atau lebih memilih memperoleh biji sebanyak-banyaknya.

Tabel 2.7. Perbandingan Penelitian Terdahulu

No	Judul	Tujuan	Metode	Hasil
1	*Dots-and-Boxes AI Algorithm*	Melakukan analisis dan perbandingan beberapa algoritma dalam AI yang dapat bermain melawan manusia	Algoritma *random*, *greedy*, *minimax*, dan *alpha-beta pruning*	Tabel perbandingan semua algoritma yang diuji dan algoritma yang terbaik.
2	*Searching & Game Playing: An Artificial Intelligence Approach to Mancala*	Implementasi, *testing* dan analisis pada permainan Mancala dengan menggunakan berbagai teknik AI pada permainannya	Algoritma *minimax* dan *alpha-beta pruning*	Perbandingan berupa statistik dengan diagram batang dan tabel peluang tiap langkah dan kemungkinan AI menang dalam tiap langkahnya.
3	*Playing Othello with Artificial Intelligence*	Menjelaskan kerangka berpikir secara umum untuk AI *game playing* dan algoritma yang paling optimal pada permainan Othello.	Algoritma *minimax* dan *alpha-beta pruning*	Rumus dan teknik perhitungan secara matematis dalam algoritma *minimax* dan *alpha-beta pruning* untuk pengembangan selanjutnya.

No	Judul	Tujuan	Metode	Hasil
4	*Efficiency of Parallel Minimax Algorithm for Game Tree Search*	Menyelidiki tingkat efisiensi dari algoritma *minimax* untuk *game* pencarian.	Algoritma *minimax* dan *alpha-beta pruning*	Tabel perbandingan algoritma *minimax* dan *alpha-beta pruning* dengan dua keadaan dari segi efisiensi dan kecepatan.
5	Implementasi Algoritma *Greedy Best First Search* Pada Aplikasi Permainan Congklak untuk Optimisasi Pemilihan Lubang dengan Pola Berpikir Dinamis	Melakukan penerapan algoritma *greedy* pada permainan Congklak	Algoritma *greedy*	Algoritma yang dapat diimplementasi-kan pada Congklak dengan kombinasi cara berpikir dinamis tetapi belum tentu optimal.

Tabel 2.7. menjelaskan tujuan dari dibuatnya tiap *paper* yang tersebut di atas, beserta metode yang digunakan serta hasil yang didapatkan pada akhir penelitian. *Paper* tersebut dipilih sesuai dengan topik yang digunakan dalam skripsi ini, baik algoritma yang digunakan maupun tipe permainan yang sama.