BAB II

LANDASAN TEORI

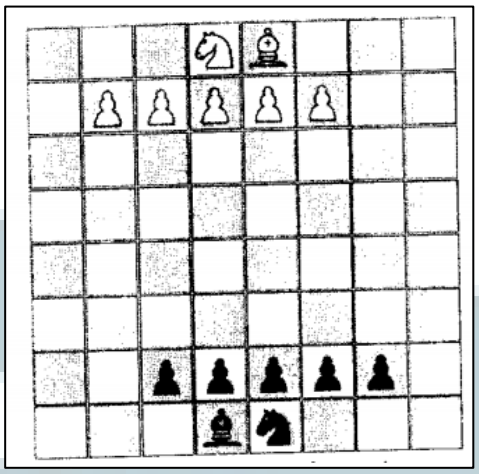
2.1 Catur

2.1.1 Dasar Permainan Catur

Ada berbagai jenis catur dengan gaya permainan yang berbeda pula. Akan tetapi, catur yang dibahas merupakan catur yang pada umumnya terdiri atas delapan macam bidak dengan ukuran papan delapan kali delapan petak. Masing-masing pemain memiliki 16 bidak yang dapat dimainkan (Setiadi, 2012:101).

Sebelum bertanding, pecatur memilih biji catur yang akan ia mainkan. Terdapat dua warna yang membedakan bidak atau biji catur, yaitu hitam dan putih. Pemegang buah putih memulai langkah pertama, yang selanjutnya diikuti oleh pemegang buah hitam secara bergantian sampai permainan selesai. Pada dasarnya, setiap langkah hanya boleh menggerakkan satu bidak saja. Bidak dipindahkan ke petak kosong ataupun yang ditempati oleh bidak lawan (Setiadi, 2012:101).

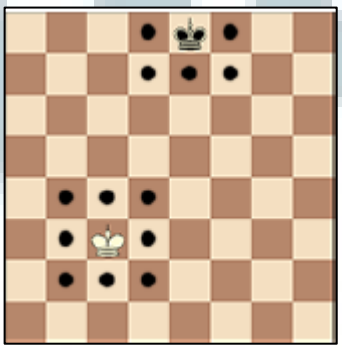
Berikut ini adalah posisi awal permainan catur sederhana masih menurut Setiadi (2012:100).



Gambar 2.1 Posisi Awal Permainan Catur Sedehana (Setiadi, 2012)

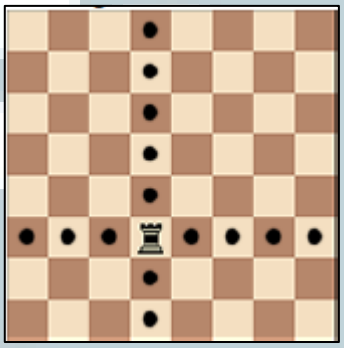
Dilansir dari *handbook* yang dikeluarkan oleh FIDE (Federation Internationale des Echecs) atau Federasi Catur Dunia mengenai catur, setiap bidak catur memiliki gerakan yang unik sebagai berikut:

1. Raja dapat bergerak satu petak ke segala arah. Ia juga memiliki gerakan khusus disebut rokade, yang turut melibatkan sebuah benteng. Raja merupakan buah catur yang paling berharga. Permainan akan segera berakhir apabila Raja sedang diserang dan tidak ada jalan untuk membebaskannya.



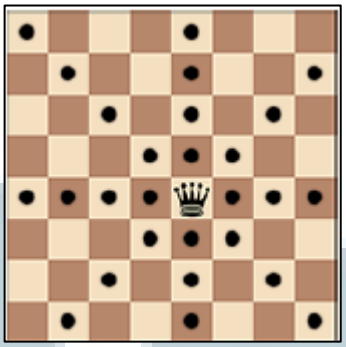
Gambar 2.2 Posisi Pergerakan Raja (FIDE, 2008)

2. Benteng dapat bergerak sepanjang petak horizontal maupun vertikal, tetapi tidak dapat melompati bidak lain. Bidak ini memiliki gerak lurus, baik ketika bergerak maupun menangkap buah catur lawan. Masing-masing pemain catur memiliki dua benteng di setiap sudut papan, yang baru bisa bergerak ketika medan permainan sudah terbuka.



Gambar 2.3 Posisi Pergerakan Benteng (FIDE, 2008)

3. Gajah dapat bergerak sepanjang petak secara diagonal, tetapi tidak dapat melompati bidak lain. Tiap pemain memulai permainan dengan dua Gajah. Satu Gajah diletakkan di antara Kuda Raja dan Raja, sedangkan Gajah lainnya diletakkan di antara Kuda Ratu dan Ratu.
4. Ratu memiliki gerakan kombinasi dari Benteng dan Gajah. Ini adalah bidak catur yang paling kuat karena dapat bergerak baik secara vertikal, horizontal, maupun diagonal ke segala arah. Masing-masing pemain memiliki satu buah Ratu yang terletak di samping Raja.



Gambar 2.4 Posisi Pergerakan Ratu (FIDE, 2008)

5. Kuda memiliki gerakan mirip huruf L, yaitu memanjang dua petak dan melebar satu petak. Kuda adalah satu-satunya bidak yang dapat melompati bidak-bidak lain

Gambar 2.5 Posisi Pergerakan Kuda (FIDE, 2008)

6. Pion dapat bergerak maju satu petak ke arah lawan di petak yang tidak ditempati. Pada gerakan pertama, Pion dapat bergerak maju sebanyak dua petak. Pion juga dapat menangkap bidak lawan secara diagonal, apabila bidak tersebut berada satu petak di diagonal depannya.

Dalam permainan catur juga dikenal istilah skak. Istilah ini berlaku ketika Raja sedang diserang oleh satu atau lebih bidak lawan. Pemain yang Rajanya diskak harus menggerakan Rajanya agak tidak terserang. Caranya adalah dengan

menangkap bidak lawan yang menyerang, menutup serangan lawan dengan menempatkan satu bidak di antaranya, atau memindahkan Raja ke petak yang tidak sedang diserang.

Skak mat merupakan tujuan yang ingin dicapai dalam permainan catur. Skak mata terjadi ketika Raja terancam dan tidak bisa menyelamatkan diri ke petak lain. Dalam pertandingan catur, pihak yang menang mendapatkan nilai 1, yang kalah 0, sedangkan jika terjadi draw maka nilai yang didapatkan adalah 0,5 (FIDE, 2008:14).

2.2 Kecerdasan Buatan

2.2.1 Sejarah Kecerdasan Buatan

Sejarah panjang perkembangan kecerdasan buatan atau yang secara umum dikenal sebagai *artificial intelligence* (AI) sebagai sub bidang kajian, dimulai oleh Warren McCulloch dan Walter Pitts pada 1943. Mereka mendasarkan penemuan mereka pada tiga sumber, yaitu pengetahuan dasar mengenai fisiologi, fungsi neuron di dalam otak manusia: analisis formal mengenai logika proposisional menurut Russell dan Whithead, serta teori Turing mengenai komputasi (Russell dan Norvig, 1995:16).

Teori mengenai komputasi sendiri ditemukan pada 1950 oleh seorang matematikawan Inggris, Alan Turing. Turing membuat sebuah skenario yang melibatkan seorang “penanya” manusia, yang memberikan pertanyaan kepada sebuah komputer dan seorang manusia melalui *keyboard* komputer. Penanya akan memperoleh jawaban melalui layar komputer atau printer, dan akan berusaha

mengetahui siapa yang memberi jawaban. Turing memiliki argumen bahwa komputer yang dapat secara konsisten memperdaya penanya bahwa ia adalah manusia, adalah komputer yang benar-benar cerdas. Turing memprediksi bahwa menjelang tahun 2000 komputer akan dapat lulus tes ini (Challoner, 2003:8).

Sejalan dengan teori ini, McCulloch dan Pitts seperti yang dijelaskan Russel dan Norvig (1995:16) berhasil menunjukkan bahwa setiap fungsi komputasi dapat dihitung oleh beberapa jaringan neuron yang terhubung, dan bahwa seluruh logika yang terhubung dapat dijalankan melalui sebuah struktur jaringan yang sederhana. Mereka juga berpendapat bahwa jaringan tersebut memiliki kemampuan untuk belajar.

Pada waktu yang bersamaan, Claude Shannon dan Alan Turing juga membuat program catur untuk komputasi konvensional ala von-Neumann. Dua sarjana dari departemen matematika Princeton University – Marvin Minsky dan Dean Edmonds – juga membuat komputer dengan jaringan neural pertama di dunia pada 1951. Seluruhnya dikembangkan menggunakan kajian AI (Russel dan Norvig, 1995:17).

Sejak AI berkembang sebagai industri pada kisaran periode 1980-1988, berbagai revolusi mulai terjadi di bidang robotika, visi komputer, pembelajaran mesin, dan representasi pengetahuan. Pemahaman masalah dan sifat kompleksitas AI yang lebih baik, dikombinasikan dengan ilmu matematika yang semakin canggih, telah menggiring para peneliti menuju berbagai agenda penelitian yang *workable* dan kuat dari segi metode (Russel dan Norvig, 1995:26).

2.2.2 Konsep Dasar AI

Seperti yang telah diketahui, komputer yang ada saat ini telah berhasil melakukan berbagai kegiatan pikiran manusia yang paling sederhana dengan baik. Misalnya melakukan perhitungan matematika, memanipulasi bilangan dan huruf, membuat keputusan sederhana, bahkan komputer saat ini dapat menyimpan berbagai hal dan memanggil fungsi-fungsi.

Suparman (1991:1) memaparkan bahwa *artificial intelligence* (AI) dikategorikan sebagai sub-bidang pengetahuan komputer – khusus ditujukan untuk membuat *software* dan *hardware* yang sepenuhnya bisa menirukan beberapa fungsi otak manusia. Dengan demikian, komputer diharapkan bisa membantu manusia dalam memecahkan berbagai masalah yang lebih kompleks.

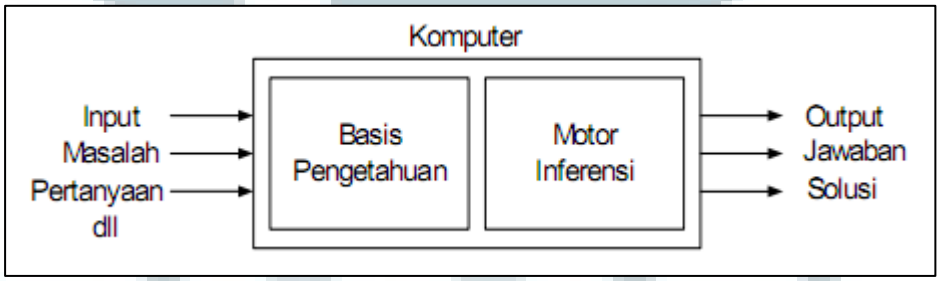
Istilah *intelligence* atau intelegensia sendiri memiliki arti seseorang yang pandai melaksanakan pengetahuan yang dimilikinya. Pemahaman tersebut mengandung arti bahwa meski seseorang memiliki banyak pengetahuan, ia tetap tidak dapat digolongkan kaum intelegensia jika tidak mampu melaksanakannya dalam praktik. Intelegensia juga dapat dikatakan sebagai kemampuan berpikir dan menalar. Oleh karenanya, pada batas-batas tertentu, AI memungkinkan komputer untuk menerima pengetahuan melalui input manusia dan menggunakan pengetahuannya itu melalui simulasi proses penalaran dan berfikir manusia untuk memecahkan berbagai masalah (Suparman, 1991:2).

AI memiliki beragam definisi sehingga membentuk 4 kategori, yaitu sistem yang berpikir seperti manusia, sistem yang berperilaku seperti manusia,

sistem yang berpikir rasional dan sistem yang berperilaku rasional (Russel dan Norvig, 1995:5).

Beberapa definisi ini merujuk pada kesimpulan bahwa bagian utama aplikasi AI adalah pengetahuan (*knowledge*). Pengetahuan ini terdiri dari fakta, pemikiran, teori, prosedur, dan hubungannya satu sama lain. Pengetahuan juga merupakan informasi terorganisasi dan teranalisa agar bisa lebih mudah dimengerti, serta dapat diterapkan pada pemecahan masalah dan pengambilan keputusan (Suparman, 1991:2).

Agar komputer dapat bertindak seperti dan sebaik manusia, maka komputer harus dibekali pengetahuan dan kemampuan menalar. Berikut adalah konsep komputer yang menggunakan teknik AI dalam suatu aplikasi, seperti yang dirangkum oleh Suparman (1991:3)



Gambar 2.6 Penerapan Konsep AI dalam Komputer (Suparman, 1991:3)

AI telah memberikan suatu kemampuan baru kepada komputer untuk memecahkan masalah yang lebih besar dan luas, tidak hanya terbatas pada soal-soal perhitungan, penyimpanan, dan pengambilan data atau pengendalian yang sederhana saja. Gambar di atas menyimpulkan beberapa cara komputer tradisional mengolah data.

Berdasarkan yang dirangkum oleh Utomo (2005:4), dua bagian utama dalam aplikasi AI adalah basis pengetahuan (knowledge base) dan motor inferensi (*inference engine*) yang juga telah tercantum pada gambar. Basis pengetahuan berisi fakta-fakta, teori, pemikiran, dan hubungan antara satu dengan yang lainnya. Sementara motor inferensi adalah kemampuan untuk menarik kesimpulan berdasarkan pengalaman.

AI merupakan *software* yang memungkinkan komputer digital untuk meniru beberapa fungsi otak manusia yang terbatas. Walaupun *hardware* khusus AI dapat dibuat, namun ternyata hampir semua *software* AI dapat dilaksanakan pada segala jenis komputer, baik komputer mikro maupun komputer besar (Suparman, 1991:4).

Adapun komputasi AI memiliki beberapa perbedaan dengan komputasi konvensional yang ditinjau dari berbagai aspek. Berikut tabel perbedaan komputasi AI dan komputasi konvensional yang dipaparkan oleh Utomo (2005:2):

Tabel 2.1 Perbedaan Komputasi AI
dengan Komputasi Konvensional (Utomo, 2005:2)

	Kecerdasan buatan	**Program konvensional**
Fokus pemrosesan	Konsep simbolik / numerik (pengetahuan)	Data & informasi
Pencarian	Heuristik	Algoritma
Sifat input	Bisa tidak lengkap	Harus lengkap
Keterangan	Disediakan	Biasanya tidak disediakan
Struktur	Kontrol dipisahkan dari pengetahuan	Kontrol terintegrasi dengan informasi (data)
Sifat output	Kuantitatif	Kualitatif
Kemampuan menalar	Ya	Tidak

Suparman (1991:4) menyatakan dalam *software* konvensional, manusia memerintah komputer untuk menyelesaikan suatu masalah. Sedangkan dalam AI, manusia tidak memerintah komputer untuk menyelesaikan masalah, tetapi

memberitahu komputer mengenai masalah tersebut. Dalam komputasi konvensional, biasanya kita memberi data kepada komputer, dan program yang telah disusun terlebih dahulu dengan menunjukkan langkah spesifik mengenai bagaimana caranya data tersebut digunakan, sampai komputer bisa memberikan solusinya. Sedangkan dalam komputasi AI, komputer diberi pengetahuan mengenai suatu wilayah subjek masalah tertentu dengan ditambah kemampuan inferensi. Kita tidak menyuruh komputer untuk memecahkan masalah, tetapi sebaliknya komputer dan *software* itu sendiri yang menentukan metode untuk mencapai sebuah solusi.

Program komputasi konvensional didasarkan pada suatu algoritma, disusun dengan jelas, terperinci langkah demi langkahnya, sampai pada suatu hasil yang sudah ditentukan sebelumnya. Program tersebut bisa berupa rumus matematika atau prosedur berurutan yang tersusun dengan jelas dan mengarah kepada suatu solusi. Algoritma dialihkan ke dalam program komputer, daftar instruksi yang tersusun berurutan dimaksudkan untuk mengarahkan komputer agar bisa sampai kepada hasil yang diinginkan. Selanjutnya, algoritma bisa digunakan untuk mengolah data bilangan, huruf, atau yang lainnya (Suparman, 1991:5).

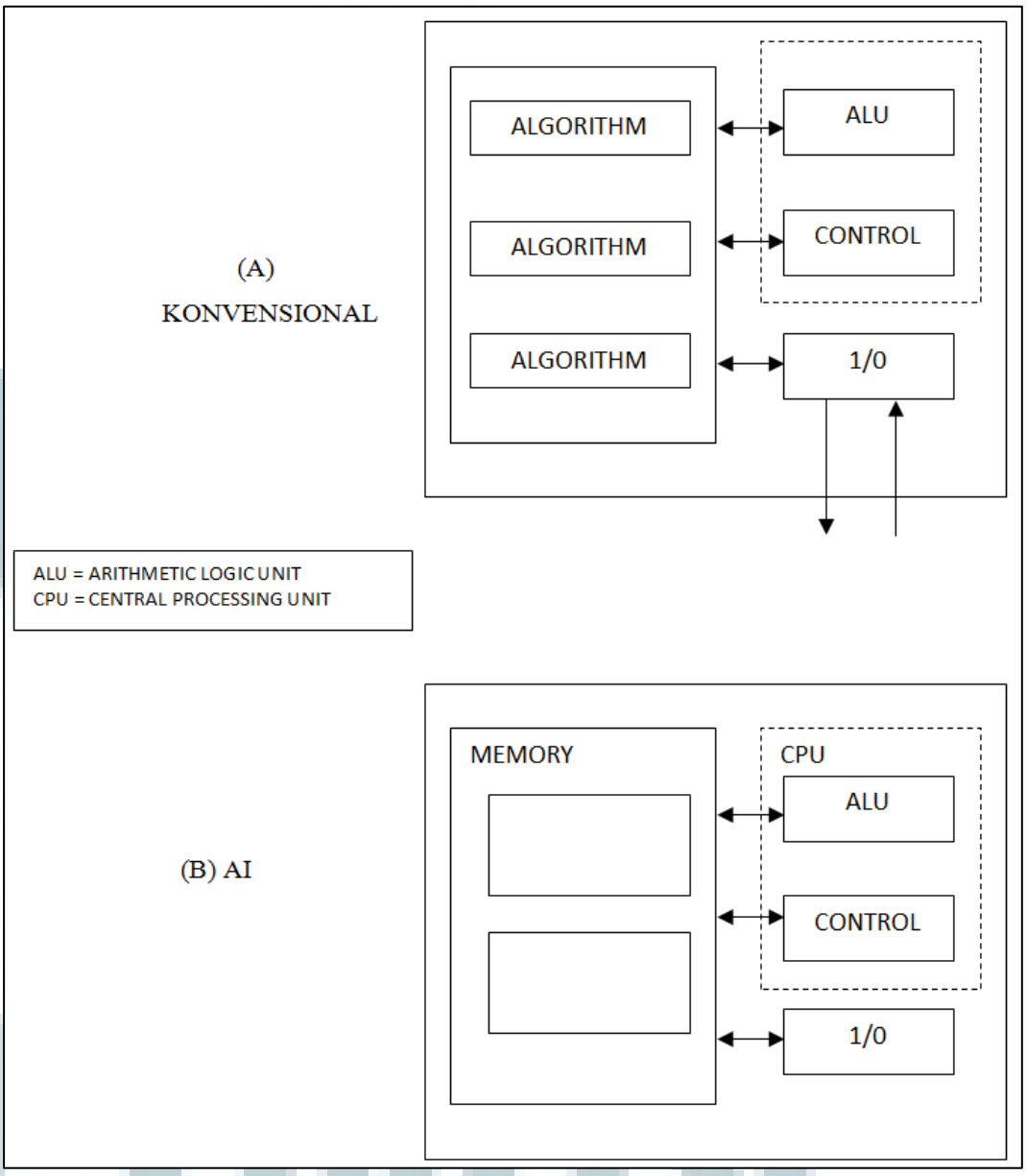
Sebaliknya, masih menurut Suparman (1991:5), *software* AI tidak didasarkan pada algoritma, tetapi pada representasi dan manipulasi simbol. Dalam AI, simbol dapat berupa huruf, kata, atau bilangan yang digunakan untuk menggambarkan objek, proses, dan hubungannya. Penggunaan simbol

memungkinkan komputer untuk menciptakan suatu pangkalan pengetahuan yang menyatakan fakta, pikiran, dan keterkaitannya satu sama lain.

Teknik dasar yang digunakan *software* AI untuk menalar dan menarik kesimpulan dari pengalamannya melalui pangkalan pengetahuan tersebut adalah pelacakan (*search*) dan penyocokan pola (*pattern matching*). Dengan diberi informasi awal, software AI melacak pangkalan pengetahuan untuk mencari pola atau kondisi yang spesifik. Secara harfiah, komputer akan terus memburu dan mencari pengetahuan yang ada sampai ia menemukan jawaban yang terbaik atau yang paling cocok (Suparman, 1991:5).

Walaupun secara langsung pemecahan masalah AI tidak didasarkan kepada algoritma, tetapi dalam implementasinya algoritma tetap digunakan. Program yang didasarkan kepada algoritma melaksanakan manipulasi simbolik yang menyebabkan suatu masalah dapat terpecahkan dengan cara yang sangat mendekati cara kerja pikiran manusia.

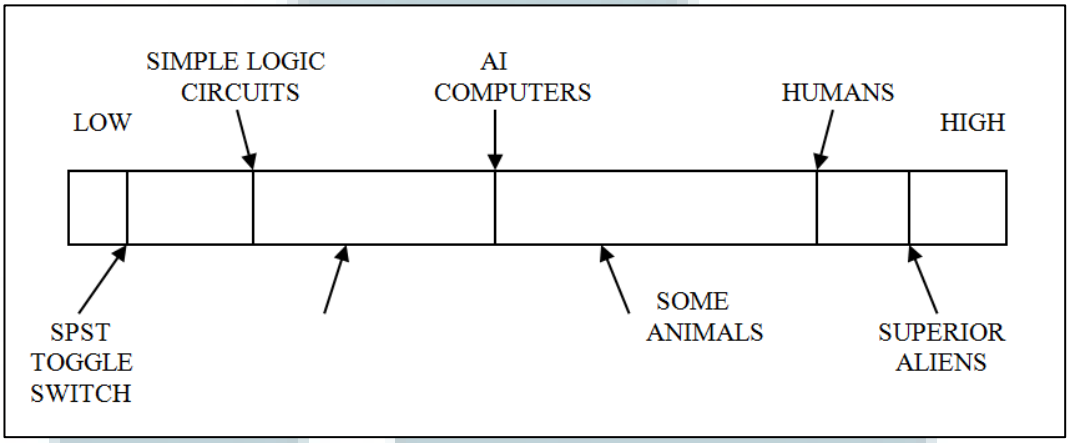
Berikut adalah gambar cara kerja komputer yang menggunakan metode konvensional (A) dan AI (B) menurut Suparman (1991:6).



Gambar 2.7 Cara Kerja Komputer dengan Metode Konvensional (A) dan AI (B) (Suparman, 1991:6)

Berdasarkan yang dipaparkan Suparman (1991:7), pada dasarnya pikiran manusia menduduki tingkat tertinggi dalam spektrum intelegensia. Sementara

rangkaian logika sederhana ada pada tingkat spektrum terendah. AI terletak di antara kedua hal tersebut. Berikut adalah penggambaran posisinya dalam spektrum intelegensia.



Gambar 2.8 Spektrum Intelegensia atau "Kemampuan Arah" (Suparman, 1991:7)

2.2.3 Implementasi Kecerdasan Buatan pada Game

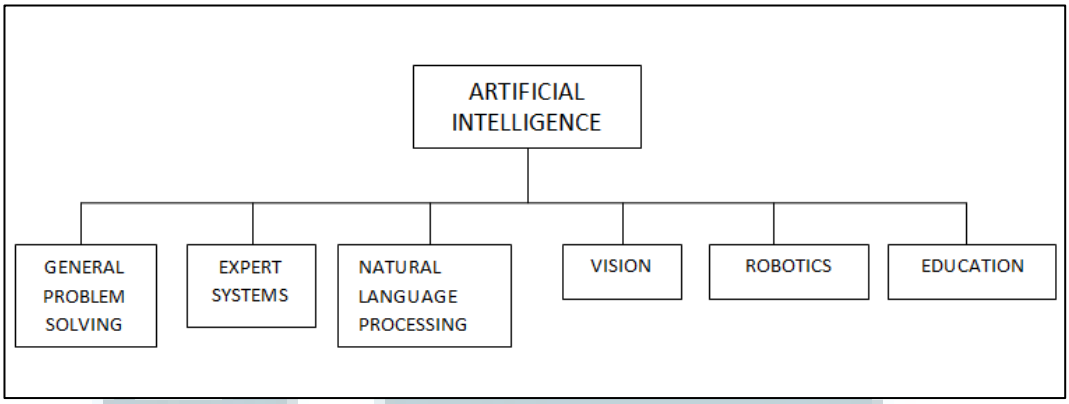
Permainan (*game*) adalah sesuatu yang sangat menarik untuk dijadikan topik tersendiri dalam kajian AI. Utomo (2005:39) memaparkan 5 alasan mengapa *game* menjadi kajian yang menarik. Pertama, kriteria menang atau kalah dalam *game* itu jelas. Kedua, *game* dapat digunakan untuk mempelajari permasalahan. Ketiga, adanya alasan histori atau sejarah. Keempat, bermain *game* itu menyenangkan. Kelima, *game* biasanya mempunyai *search space* yang besar (misanya game catur memiliki 35.100 *nodes* dalam *search tree* dan 1040 *legal states*).

Terdapat beberapa ciri umum pada *game* dalam AI yang juga dijelaskan oleh Utomo (2005:40), yaitu terdapat dua orang pemain, adanya kesempatan bagi

pemain untuk bergantian, *zero-sum* yang berarti kerugian seorang pemain adalah keuntungan pemain lain, *perfect information* yang berarti pemain mengetahui semua informasi *state* dari *game*, serta tidak mengandung probabilistik (seperti dadu). Contoh *game* yang berbasis AI adalah Tic-Tac-Toe, Checkers, Chess, Go, Nim, dan Othello. Sementara *game* yang tidak berbasis AI adalah Bridge, Solitaire, Backgammon, dan sejenisnya. Para peneliti merasa, kemampuan komputer untuk menjalankan *game* merupakan demonstrasi yang luar biasa mengenai intelegensia manusia – tentu saja dalam hal yang terkendali dan terbatas.

Suparman (2005:10) memaparkan bahwa aplikasi pertama yang merupakan bukti hasil pemikiran pengolahan simbolik AI adalah Kubus Rubik. Para peneliti AI juga telah menyingkap keterbatasan komputer dalam aplikasi awal ini. Memori yang besar sangat diperlukan agar bisa digunakan untuk menyimpan pangkalan pengetahuan (kaidah permainan, bagaimana menggerakan anak catur, membuka strategi, dll). Oleh karena itu, pada awalnya hanya *Mainframe* yang terbesar dan tercepat sajalah yang bisa digunakan untuk masalah AI.

Oleh karena itu, para pemakai saat ini mulai mencari program yang bisa lebih meningkatkan produktivitas kerjanya. Hal itu telah terjawab oleh AI dengan diterapkannya aplikasi pemecahan masalah (*problem solving*), sistem pakar (*expert system*), pengolahan bahasa alami (*natural language processing*), *computer vision*, robot, dan pendidikan (Suparman, 2005:11).



Gambar 2.9 Daerah Utama Aplikasi AI (Suparman, 2005:11)

Secara umum, untuk membangun suatu sistem yang mampu menyelesaikan masalah, perlu dipertimbangkan 4 hal, yaitu mendefinisikan masalah dengan tepat (pendefinisian ini mencakup spesifikasi yang tepat mengenai keadaan awal dan solusi yang diharapkan), menganalisis masalah tersebut serta mencari beberapa teknik penyelesaian masalah yang sesuai, merepresentasikan pengetahuan yang perlu untuk menyelesaikan masalah tersebut, serta memilih teknik penyelesaian masalah yang terbaik (Suparman, 2005:6).

Terdapat beberapa hal yang harus diperhatikan untuk menjadikan komputer sebagai salah satu pemain dalam game menurut Utomo (2005:40) , yaitu:

1. Cara bermain *game*:
    a. Mempertimbangkan semua kemungkinan jalan
    b. Memberikan nilai pada semua kemungkinan jalan
    c. Menjalankan kemungkinan yang mempunyai nilai terbaik

d. Menunggu giliran jalan pihak lawan

e. Mengulangi cara di atas

2. Permasalahan kunci:

a. Mereprentasikan "*board*" atau "*state*"

b. Membuat *next board* yang legal

c. Melakukan evaluasi pada posisi

Millington dan Funge (2009:9) membuat ilustrasi model *game* AI yang dibagi dalam 3 *section*, yaitu pergerakan (*movement*), pembuatan keputusan (*decision making*), dan strategi (*strategy*). Dua *section* pertama melibatkan algoritma yang bekerja pada basis karakter per karakter, sedangkan *section* terakhir beroperasi pada keseluruhan *game*.

Tidak semua aplikasi *game* membutuhkan seluruh level AI. *Board game* seperti Chess atau Risk hanya membutuhkan level strategi. Karakter-karakter pada *game* tersebut (jika memang dapat disebut demikian) tidak membuat keputusan mereka sendiri dan tidak perlu mengkhawatirkan pergerakan mereka.

Challoner (2003:26) menjelaskan istilah "AI atas-bawah" yang muncul ketika seorang ilmuwan komputer bernama Herbert Simon bersama rekannya Allen Newell mengklaim program komputer rancangannya yang mereka sebut *Logic Theorist* (Teoretikawan Logika). Program ini dapat mengerjakan bukti dari teorema-teorema matematika dengan sebuah proses deduksi logika.

Logic theorist mermasukkan sekumpulan aturan dan instruksi – yaitu suatu algoritma – yang menggunakan fakta-fakta matematis sebagai titik awal, dan secara otomatis mendeduksi sejumlah bukti matematika dasar. Deduksi dan

pembuatan keputusan yang diprogramkan membuat sistem atas-bawah sangat baik dalam menjalankan tugas-tugas yang memerlukan penalaran logika, seperti bermain *game*.

Lebih lanjut, Challoner (2003:28) menyebutkan bahwa ilustrasi terbaik dari pendekatan atas-bawah ke AI adalah sebuah program komputer yang dapat bermain catur. Ketika tiba giliran komputer, ia harus memutuskan langkah apa yang harus dibuat berdasarkan posisi bidak-bidak catur di papan catur. Dari setiap kemungkinan langkah yang dilakukan, akan ada banyak langkah yang dapat dilakukan lawannya, dan untuk masing-masing langkah ini akan ada lebih banyak lagi langkah yang dapat dipilih komputer. Program komputer harus mampu "melihat ke depan" banyak langkah, menghitung mana langkah berikutnya yang lebih mungkin menempatkannya pada posisi kemenangan. Logika yang terpasang ini adalah esensi dari pendekatan atas-bawah.

Salah satu *computer game player* yang telah mencatat kehebatannya dalam permainan catur adalah sebuah super komputer bernama Deep Blue yang berhasil mengalahkan pemain catur manusia terbaik dunia yaitu Gary Kasparov pada Mei 1997. Deep Blue dapat menganalisis lebih dari 200 juta posisi setiap detiknya (Challoner, 2003:28).

Korf (1997:1) dalam jurnalnya mengenai analisis penggunaan AI dalam Deep Blue menyebutkan bahwa salah satu teknik tertua AI yaitu pencarian heuristik, memang sedari awal dikembangkan untuk mengatasi permasalahan seperti *game* catur di komputer. Secara khusus, AI ditambahkan ke algoritma minimax dari ide-ide statis heuristik dan alpha-beta pruning, yang seluruhnya

merupakan pengembang (*developer*) pada 1950-an. Ide-ide itulah yang membentuk intisari dari seluruh program catur sampai dengan saat ini, termasuk Deep Blue.

2.3 Algoritma

Ngoen (2004:5) menjelaskan bahwa istilah algoritma, pada awalnya berasal dari nama Abu Ja'far Muhammad Ibnu Musa al-Khuwarizmi, seorang ilmuwan Persia yang pada kisaran tahun 825 menulis buku Kitab Al Jabr W'al-muqbala (*Rules of Restoration and Reduction*).

Nama Al-Khuwarizmi sendiri dibaca sebagai *algorism* oleh masayarakat Barat. Kitab yang ditulis oleh Al-Khuwarizmi pada dasarnya dapat diartikan sebagai "Buku pemugaran dan pengurangan". Dari judul buku itulah diperoleh akar kata "aljabar" (*algebra*). Perubahan kata dari *algorism* menjadi *algorithm* muncul karena kata *algorism* sering dikaitkan dengan *arithmetic* sehingga akhiran *–sm* berubah menjadi *–thm*. Karena perhitungan dengan angka Arab sudah menjadi hal yang biasa, maka lambat laun ka *algorithm* berangsur-angsur dipakai sebagai metode perhitungan (komputasi) secara umum, sehingga kehilangan makna kata aslinya. Dalam bahasa Indonesia, kata *algorithm* diserap menjadi *algoritma* (Zarlis dan Handrizal, 2008:1).

Kata *algorithmus* yang juga merupakan istilah turunan dari *algorism* dijelaskan Leipzig dalam Ngoen (2004:5) juga muncul pada kamus matematika Vollstandiges Mathematisches Lexicon yang menggunakan istilah ini untuk

kombinas dari empat jenis perhitungan matematika: penjumlahan, perkalian, pengurangan, dan perkalian.

2.3.1 Konsep Dasar Algoritma

Sampai dengan tahun 1950, istilah algoritma selalu diasosiasikan dengan *Euclid's algorithm*, yaitu suatu proses yang menjelaskan cara mencari bilangan pembagi terbesar untuk dua buah bilangan (*greatest common divisor*). Pada Merriam-Webster's Collegiate Dictionary, istilah *algorithm* diartikan sebagai prosedur langkah demi langkah untuk memecahkan masalah atau menyelesaikan suatu tugas khususnya dengan menggunakan bantuan komputer. KBBI sendiri mendefinisikan algoritma sebagai urutan logis pengambilan keputusan untuk pemecahan masalah. (Ngoen, 2004:5).

Zaris dan Handrizal (2008:1) menyebutkan bahwa terdapat beberapa pertimbangan yang harus ada dalam memilih algoritma. Pertama, algoritma haruslah benar. Artinya, algoritma akan memberikan keluaran yang dikehendaki dari sejumlah masukan yang diberikan. Jika keluaran yang diberikan salah, maka algoritma tersebut pastilah bukan algoritma yang baik.

Kedua, kita harus mengetahui seberapa baik hasil yang dicapai oleh algoritma tersebut. Pertimbangan ini penting untuk dilakukan terutama pada algoritma yang digunakan untuk menyelesaikan masalah yang memerlukan aproksimasi hasil (hasil yang hanya berupa pendekatan). Algoritma yang baik harus mampu memberikan hasil yang sedekat mungkin dengan nilai sebenarnya.

Hal ketiga adalah efisiensi algoritma yang dapat ditinjau dari dua hal yaitu efisiensi waktu dan memori. Walaupun algoritma memberikan keluaran yang besar dan paling mendekati, tetapi jika untuk mendapatkan keluaran tersebut diperlukan waktu yang panjang, maka algoritma tersebut biasanya tidak akan dipakai. Sama halnya dengan memori, semakin besar memori yang terpakai maka semakin buruk algoritma tersebut.

Syarat algoritma menurut Donald E. Knuth dalam Ngoen (2004:6) adalah sebagai berikut.

1. *Finitiness*, yaitu algoritma harus berakhir (*terminate*) setelah melakukan sejumlah langkah proses.
2. *Definitiness*, yaitu setiap langkah harus didefinisikan dengan tepat dan tidak ambigu. Oleh karenanya, cara paling tepat untuk menuliskan algoritma sesungguhnya adalah dengan menggunakan *formal language* atau bahasa pemrograman komputer.
3. *Input*, yaitu data sebagai masukan yang dapat diolah. Algoritma yang tidak memerlukan masukan apa-apa sesungguhnya tidak terlalu bermanfaat, karena jumlah kasus yang dapat diselesaikan juga terbatas.
4. *Output*, yaitu setiap algoritma memberikan satu atau beberapa hasil keluaran.
5 *Effectiveness*, yaitu langkah-langkah algoritma harus dapat dikerjakan dalam waktu yang wajar.

2.3.2 Konsep Dasar Algoritma Minimax

Pada 1944, John von Neumann menguraikan sebuah algoritma pencarian untuk game yang dikenal dengan nama Minimax. Algoritma ini mampu memaksimalkan posisi pemain dan meminimalkan posisi lawan (Utomo, 2005:41).

Algoritma minimax merupakan salah satu algoritma yang kerap digunakan pada permainan dua orang yang menggunakan AI, atau pada *zero-sum* game seperti catur. Pada algoritma ini, pengecekan akan dilakukan untuk mencari seluruh kemungkinan yang ada, bahkan dapat dilakukan sampai akhir permainan. Pengecekan tersebut akan menghasilkan pohon permainan yang berisi seluruh kemungkinan tersebut (Setiadi, 2012:101).

Setiadi (2012:101) juga memaparkan bahwa pada algoritma minimax, komputer akan menganalisis seluruh pohon permainan. Dan untuk setiap langkahnya, komputer akan memilih langkah yang dapat meminimumkan keuntungan lawan, serta memaksimumkan keuntungan bagi komputer itu sendiri. Dalam penentuan keputusan itu, diperlukan suatu nilai atau bobot yang dapat merepresentasikan kerugian dan keuntungan yang akan diperoleh pada setiap langkah, sehingga langkah yang memiliki nilai terbesar akan dipilih.

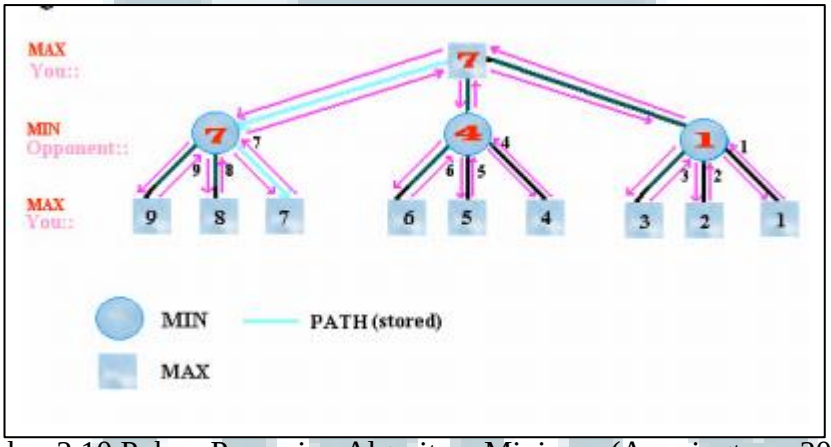
2.3.3 Implementasi Algoritma Minimax

Teori minimax menyatakan bahwa untuk setiap dua orang pemain dalam *zero-sum* game, terdapat nilai V dari strategi yang dimiliki pemain, seperti yang dipaparkan Ayuningtyas (2008:2) sebagai berikut:

1. Strategi yang ditentukan pemain kedua akan menghasilkan konsekuensi kemungkinan untuk pemain pertama, V
2. Strategi yang ditentukan pemain pertama akan menghasilkan konsekuensi kemungkinan untuk pemain pertama, -V

Secara setara, strategi pemain pertama akan memastikan suatu nilai V tanpa memedulikan strategi pemain kedua. Bersamaan dengan itu, pemain kedua akan memastikan dirinya kehilangan nilai sebesar –V.

Dalam representasi pohon dalam algoritma Minimax, terdapat dua jenis node, yaitu node *min* dan node *max*. *Max node* akan memilih langkah dengan nilai tertinggi, dan *min node* akan memilih langkah dengan nilai terendah. Berikut ini merupakan gambar pohon untuk algoritma Minimax (Ayuningtyas, 2008:2).



Gambar 2.10 Pohon Pencarian Algoritma Minimax (Ayuningtyas, 2008:2)

Berdasarkan gambar tersebut, diketahui bahwa dalam algoritma ini terdapat dua peran, yaitu *max* dan *min*. Pembuatan pohon dimulai dari posisi awal

hingga posisi akhir permainan. Selanjutnya, posisi akhir dievaluasi dari sudut pandang *max*. Setelah itu, node bagian dalam diisi dengan nilai yang telah dievaluasi. Node *max* akan menerima nilai maksimum dari anak-anaknya, sedangkan node *min* akan memilih nilai minimum dari anak-anaknya. Algoritma minimax untuk hal ini, seperti yang dirangkum dalam Ayuningtyas (2008:3) adalah:

```
function MinMax (game : GamePosition)
  return MaxMove (game)
```

```
function MaxMove (game : GamePosition)
  if (GameEnded(game)) then
    return EvalGameState(game)
  else
    best_move ← ()
    moves ← GenerateMoves(game)
    ForEach moves do
       move ← MinMove(ApplyMove(game))
       if (Value(move) > Value(best_move))
then
          best_move ← move
       endif
    endfor
    return best_move
   endif
```

```
function MinMove (game : GamePosition)
if (GameEnded(game)) then
    return EvalGameState(game)
  else
  best_move ← ()
  moves ← GenerateMoves(game)
  ForEach moves do
     move ← MaxMove(ApplyMove(game))
     if (Value(move) > Value(best_move)) then
        best_move ← move
     endif
  endfor
  return best_move
endif
```

Gambar 2.11 Pseudocode Algoritma Minimax (Ayuningtyas, 2008:3)

Kelemahan dari metode ini menurut Utomo (2005:43) adalah waktu eksekusi yang dibutuhkan sebanding dengan jumlah kemungkinan yang ada. Sehingga jika kemungkinan atau #leaf lebih besar, maka permasalahan akan semakin kombinatorik. Persoalan ini diperbaiki dengan sebuah metode yang dinamakan alpha-beta pruning, seperti yang akan diuraikan pada subbab selanjutnya.

2.3.4 Konsep Dasar Algoritma Alpha-Beta Pruning

Dalam algoritma minimax, kerap terjadi pengecekan sebuah simpul atau kemungkinan yang seharusnya tidak dicek karena tidak akan mempengaruhi hasil akhir. Untuk menghindari hal tersebut, dibuatlah sebuah algoritma minimax yang lebih optimal, yaitu algoritma alpha-beta pruning.

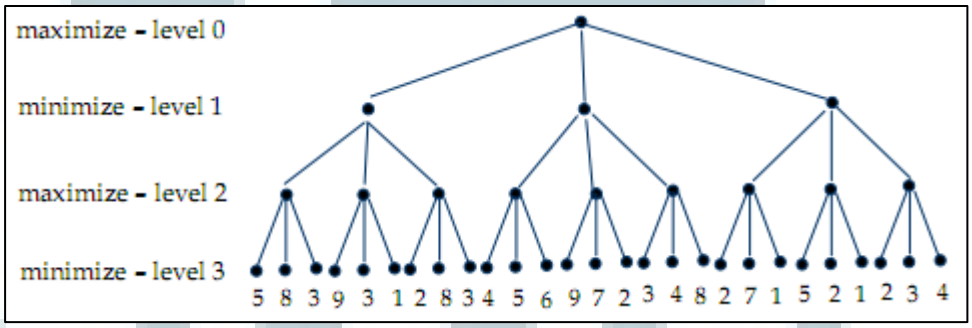
Menurut Winata (2012:3), hasil algoritma alpha-beta pruning ini sendiri tidak berubah dari algoritma minimax, yang menjadi pembeda adalah traversalnya yang lebih sedikit. Cara kerja algoritma ini adalah mengecek sebuah simbul n, dan jika pemain memiliki pilihan yang lebih baik pada akar n atau simpul-simpul selanjutnya, maka sebenarnya n tidak pernah dicapai pada waktu permainan. Maka pada n akan dilakukan pruning, yaitu simpul n tidak akan dikembangkan lagi pada pohon.

Sederhananya, menurut Ayuningtyas (2008:3), algoritma ini akan berhenti mengevaluasi langkah jika terdapat paling tidak satu kemungkinan yang ditemukan, dan membuktikan bahwa langkah tersebut lebih buruk jika dibandingkan dengan langkah yang diperiksa sebelumnya.

Algoritma alpha-beta pruning memanfaatkan dua nilai, yaitu alpha yang menunjukkan skor pilihan terbaik yang bisa diambil pemain Max, dan beta yang menunjukkan skor pilihan terbaik yang bisa diambil pemain Min. Pruning dilakukan ketika simpul yang sedang ditinjau pemain Max, yaitu n, memiliki skor yang lebih rendah daripada alpha. Hal yang sama dilakukan pada pemain Min memakai nilai beta (Winata, 2012:4).

2.3.5 Implementasi Alpha-Beta Pruning

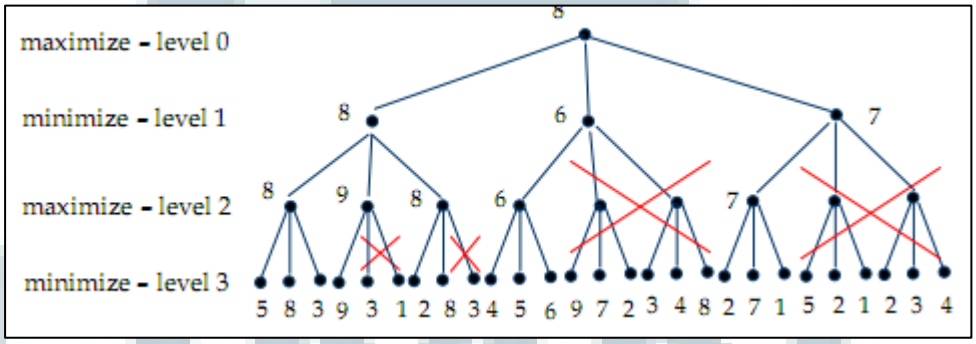
Utomo (2005:44) memberikan contoh kondisi awal dari sebuah keadaan yang akan ditinjau menggunakan metode alpha-beta pruning seperti pada gambar yang tertera berikut ini.



Gambar 2.12 Tree untuk Kondisi Awal Alpha-Beta Pruning (Utomo, 2005:44)

Algoritmanya seperti yang dipaparkan oleh Utomo (2005:45) adalah:

1. Hampiri node pertama pada leaf dengan nilai 5, naik ke parent pada level 2, masukkan nilai 5, hampiri 8, karena 8 > 5 maka ganti parent dengan 8, hampiri 3.
2. Setelah ketiga leaf pertama terhampiri, naik lebih tinggi lagi ke level 1, masukkan nilai 8.
3. Hampiri node keempat pada leaf dengan nilai 9, naik ke parent pada level 2, masukkan nilai 9. Jika menghampiri leaf berikutnya, kita mencari nilai yang lebih tinggi dari 9, sementara pada level 1 kita mencari yang lebih kecil dari 8, maka leaf 3 dan 1 kita potong (tidak kita hampiri).
4. Begitu seterusnya hingga didapatkan hasil akhir dari tree di atas adalah seperti pada gambar yang tertera di bawah ini.



Gambar 2.13 Tree untuk Hasil Metode Alpha-Beta Prunning (Utomo, 2005:45)

Sementara berikut ini adalah pseudocode untuk algoritma Minimax yang telah mengimplementasikan alpha-beta pruning seperti yang dirangkum oleh Ayuningtyas (2008:4).

```
function MinMax (game : GamePosition)
  return MaxMove (game)
```

```
function MaxMove (game : GamePosition, alpha
: integer, beta: integer)
  if (GameEnded(game)) then
    return EvalGameState(game)
  else
    best_move ← ()
    moves ← GenerateMoves(game)
    ForEach moves do
       move ← MinMove(ApplyMove(game), alpha,
beta)
       if (Value(move) > Value(best_move))
then
          best_move ← move
           alpha ← Value(move)
        endif
        if (beta>alpha)
           return best_move
        endif
    endfor
    return best_move
   endif
```

```
function MinMove (game : GamePosition, alpha
: integer, beta : integer)
if (GameEnded(game)) then
    return EvalGameState(game)
  else
  best_move ← ()
  moves ← GenerateMoves(game)
  ForEach moves do
     move ← MaxMove(ApplyMove(game), alpha,
beta)
     if (Value(move) > Value(best_move)) then
        best_move ← move
         beta ← Value(move)
     endif
     if (beta<alpha)
        return move
     endif
  endfor
  return best_move
endif
```

Gambar 2.14 Pseudocode Algoritma Minimax dengan Optimasi Alpha-Beta Pruning (Ayuningtyas, 2008:4)