# BAB II
# LANDASAN TEORI

## II.1 Permainan

Permainan merupakan sebuah aktivitas rekreasi dengan tujuan bersenang-senang, mengisi waktu luang, atau berolahraga ringan. Permainan biasanya dilakukan sendiri atau bersama-sama. Permainan komputer adalah permainan video yang dimainkan pada komputer pribadi, dan bukan pada konsol permainan, maupun mesin ding-dong. Permainan komputer telah berevolusi dari sistem grafis sederhana sampai menjadi kompleks dan mutakhir.

## II.2 Permainan *Turn Based Strategy* (*TBS*)

Permainan *Turn Based Strategy* (*TBS*) adalah permainan strategi yang biasanya bertipe *game* perang (*wargame*), dimana pemain melakukan giliran dalam permainan. *TBS* sendiri berasal dari permainan tradisional catur. Catur adalah permainan mental yang dimainkan oleh dua orang. Pecatur adalah orang yang memainkan catur, baik dalam pertandingan satu lawan satu maupun satu melawan banyak orang (dalam keadaan informal). 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.

## II.3 Graph

Graph merupakan struktur data yang tersusun atas simpul-simpul atau titik-titik (simpul) yang terhubung satu sama lain dengan garis-garis (*edge*). Secara matematis, graph dinyatakan sebagai berikut :

$$G = (V,E)$$

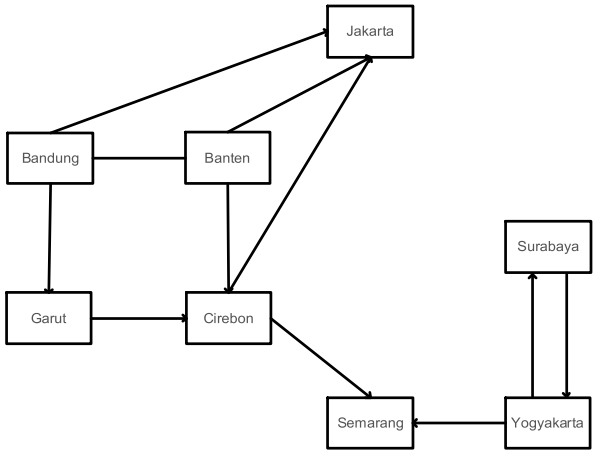
Dimana:

G = Graph

V= Simpul atau vertex, simpul

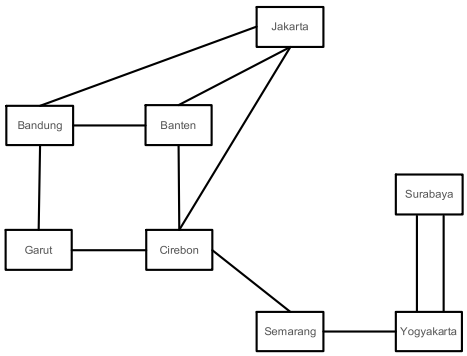
E = Sisi, busur atau edge

Terdapat banyak sekali jenis graph. Namun secara orientasi arah, graph dapat dibedakan menjadi dua, pertama graph berarah. Graph berarah merupakan graph yang mana sisinya memiliki arah. Sisi ini biasa disebut busur (*arc*). Pada graph berarah, urutan pasangan simpul sangat diperhatikan karena dapat menyatakan hal yang berbeda.




**Gambar II. 1 Graph Berarah**

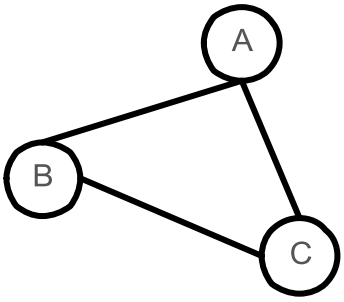
Pada Gambar II-1 dapat dilihat bahwa dua busur yang menghubungkan (Yogyakarta, Surabaya) dan (Surabaya, Yogyakarta) menyatakan hal yang berbeda. Untuk busur yang menghubungkan simpul Yogyakarta dan Surabaya, simpul Yogyakarta dinamakan sebagai simpul awal (*initial suksesor*) dan simpul Surabaya dinamakan sebagai simpul terminal. Sebaliknya, pada busur yang menghubungkan simpul Surabaya dan Yogyakarta, simpul Surabaya merupakan simpul awal dan simpul Yogyakarta adalah simpul terminal. Yang kedua adalah graph tidak berarah. Graph tidak berarah merupakan graph yang sisinya tidak memiliki arah (kebalikan dari graph berarah). Pada graph tidak berarah, urutan pasangan simpul yang berhubungan tidak diperhatikan.




**Gambar II. 2 Graph Tidak Berarah**

Dari Gambar II-2 dapat dilihat bahwa pasangan simpul (Bandung, Garut) dan (Garut, Bandung) menyatakan hal yang sama karena busur yang ada tidak memiliki orientasi arah. Graph dapat dinyatakan sebagai sebuah struktur data, atau secara spesifik dinamakan sebagai ADT (abstract data type) yang terdiri dari kumpulan simpul dan sisi yang membangun hubungan antar simpul. Secara umum terdapat dua macam representasi dari struktur data graph, yang pertama *adjacency list* dan yang kedua *adjacency matri.*

*Adjacency list* merupakan bentuk representasi dari busur atau sisi dalam suatu graph sebagai satu senarai. Simpul yang terhubung oleh busur atau sisi dinyatakan sebagai simpul yang terkait. Dalam implementasinya, tabel hash digunakan untuk menghubungkan simpul dengan senarai yang berisi simpul-simpul yang saling terkait.




**Gambar II. 3 *Adjency List***

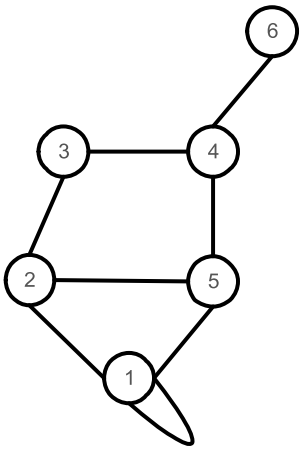
Graph pada Gambar II-3 *Adjency List* dapat digambarkan sebagai senarai {A,B}, {A,C} dan {B,C}. Representasi dari *adjacency list* dapat dilihat pada Tabel II-1.

**Tabel II. 1 Representasi Adjacency List**

| Simpul | Kedekatan | Senarai Dari Simpul yang Berhubungan |
|---|---|---|
| A | Dekat dengan | B,C |
| B | Dekat dengan | A,C |
| C | Dekat dengan | A,B |

Salah satu kekurangan dari representasi *adjacency list* adalah tidak adanya tempat untuk menyimpan nilai yang melekat pada sisi yang mana contoh nilai ini adalah nilai jarak simpul.

Yang kedua adalah *adjacency matrix. Adjacency matrix* merupakan representasi matriks $n \times n$ yang menyatakan hubungan antar simpul dalam graph. Kolom dan baris dari matriks merepresentasikan simpul-simpul, dan nilai entri dalam matriks ini menyatakan hubungan antar simpul. Berikut ini adalah representasi *adjacency matrix* dari graph tak berarah.




**Gambar II. 4 Graph Tak Berarah**

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

**Gambar II. 9 Representasi Adjacency Matrix Dari Gambar II. 4**

Kelebihan dari adjacency matrix ini adalah elemen matriksnya dapat diakses langsung melalui indeks. Sedangkan kekurangannya bila graph memiliki jumlah sisi atau busur yang relatif sedikit, kebutuhan ruang hash tabel untuk matriks menjadi boros dan tidak efisien karena komputer menyimpan elemen yang tidak perlu.

## II.4 Algoritma

Dalam matematika dan komputasi, algoritma merupakan kumpulan perintah untuk menyelesaikan suatu masalah. Perintah-perintah ini dapat diterjemahkan secara bertahap dari awal hingga akhir. Masalah tersebut dapat berupa apa saja, dengan catatan untuk setiap masalah, ada kriteria kondisi awal yang harus dipenuhi sebelum menjalankan algoritma. Algoritma akan dapat selalu berakhir untuk semua kondisi awal yang memenuhi kriteria, dalam hal ini berbeda dengan heuristik. Algoritma sering mempunyai langkah pengulangan (iterasi) atau memerlukan keputusan (logika Boolean dan perbandingan) sampai tugasnya selesai.Desain dan analisis algoritma adalah suatu cabang khusus dalam ilmu komputer yang mempelajari karakteristik dan performa dari suatu algoritma dalam menyelesaikan masalah, terlepas dari implementasi algoritma tersebut. Dalam cabang disiplin ini algoritma dipelajari secara abstrak, terlepas dari sistem komputer atau bahasa pemrograman yang digunakan. Algoritma yang berbeda dapat diterapkan pada suatu masalah dengan kriteria yang sama.Kompleksitas dari suatu algoritma merupakan ukuran seberapa banyak komputasi yang dibutuhkan algoritma tersebut untuk menyelesaikan masalah[7].

Secara informal, algoritma yang dapat menyelesaikan suatu permasalahan dalam waktu yang singkat memiliki kompleksitas yang rendah, sementara algoritma yang membutuhkan waktu lama untuk menyelesaikan masalahnya mempunyai kompleksitas yang tinggi.

**II.5 Algoritma pencarian**

Dalam ilmu komputer, sebuah algoritma pencarian dijelaskan secara luas adalah sebuah algoritma yang menerima masukan berupa sebuah masalah danmenghasilkan sebuah solusi untuk masalah tersebut, yang biasanya didapat dari evaluasi beberapa kemungkinan solusi. Sebagian besar algoritma yang dipelajari oleh ilmuwan komputer adalah algoritma pencarian. Himpunan semua kemungkinan solusi dari sebuah masalah disebut ruang pencarian. Algortima pencarian brute-force atau pencarian naif/uninformed menggunakan metode yang sederhana dan sangat intuitif pada ruang pencarian, sedangkan algoritma pencarian informed menggunakan heuristik untuk menerapkan pengetahuan tentang struktur dari ruang pencarian untuk berusaha mengurangi banyaknya waktu yang dipakai dalam pencarian.Sebuah algoritma pencarian uninformed adalah algoritma yang tidak mempertimbangkan sifat alami dari permasalahan. Oleh karena itu algoritma tersebut dapat diimplementasikan secara umum, sehingga dengan implementasi yang sama dapat digunakan pada lingkup permasalahan yang luas, hal ini berkat abstraksi[1].

Kekurangannya adalah sebagian besar ruang pencarian adalah sangat besar, dan sebuah pencarian uninformed (khususnya untuk pohon) membutuhkan banyak waktu walaupun hanya untuk contoh yang kecil. Sehingga untuk mempercepat proses, kadang-kadang hanya pencarian informed yang dapat melakukannya.

**II.6 Kecerdasan buatan**

Kecerdasan buatan (*artificial intelligence*) merupakan inovasi baru di bidang ilmu pengetahuan. Mulai ada sejak muncul komputer modern, yakni pada 1940 dan 1950. Ini kemampuan mesin elektronika baru menyimpan sejumlah besar

info, juga memproses dengan kecepatan sangat tinggi menandingi kemampuan manusia.

Pentingnya kecerdasan buatan menjadi nyata bagi negara-negara yang berperan sejak tahun 1970. Para pemimpin negara yang mengakui potensialnya kecerdasan buatan mengharap mendapat persetujuan jangka panjang untuk sumber-sumber yang memerlukan dana intensif. Jepang adalah yang pertama kali melakukan itu. Negara ini mengembangkan program yang sangat berambisi dalam penelitian kecerdasan buatan[1].

**II.7 Sejarah Kecerdasan Buatan**

Pada awal abad 17, Rene Descartes mengemukakan bahwa tubuh hewan bukanlah apa-apa melainkan hanya mesin-mesin yang rumit. Blaise Pascal menciptakan mesin penghitung digital mekanis pertama pada 1642. Pada 19, Charles Babbage dan Ada Lovelace bekerja pada mesin penghitung mekanis yang dapat diprogram.

Bertrand Russell dan Alfred North Whitehead menerbitkan Principia Mathematica, yang merombak logika formal. Warren McCulloch dan Walter Pitts menerbitkan "Kalkulus Logis Program AI pertama yang bekerja ditulis pada 1951 untuk menjalankan mesin Ferranti Mark di University of Manchester (UK): sebuah program permainan naskah yang ditulis oleh Christopher Strachey dan program permainan catur yang ditulis oleh Dietrich Prinz. John McCarthy membuat istilah "kecerdasan buatan " pada konferensi pertama yang disediakan untuk pokok persoalan ini, pada 1956. Dia juga menemukan bahasa pemrograman Lisp. Alan Turing memperkenalkan “Turing Test” sebagai sebuah cara untuk mengoperasionalkan test perilaku cerdas. Joseph Weizenbaum membangun ELIZA, sebuah chatterbot yang menerapkan psikoterapi Rogerian Selama tahun 1960-an dan 1970-an, Joel Moses mendemonstrasikan kekuatan pertimbangan simbolis untuk mengintegrasikan masalah di dalam program Macsyma, program berbasis pengetahuan yang sukses pertama kali dalam bidang matematika.

Marvin Minsky dan Seymour Papert menerbitkan Perceptrons, yang mendemostrasikan batas jaringan syaraf sederhana dan Alain Colmerauer

mengembangkan bahasa komputer Prolog. Ted Shortliffe mendemonstrasikan kekuatan sistem berbasis aturan untuk representasi pengetahuan dan inferensi dalam diagnosa dan terapi medis yang kadangkala disebut sebagai sistem pakar pertama. Hans Moravec mengembangkan kendaraan terkendali komputer pertama untuk mengatasi jalan berintang yang kusut secara mandiri.

Pada Pada tahun 1980-an, jaringan syaraf digunakan secara meluas dengan algoritma perambatan balik, pertama kali diterangkan oleh Paul John Werbos pada 1974. Tahun 1990-an ditandai perolehan besar dalam berbagai bidang AI dan demonstrasi berbagai macam aplikasi. Lebih khusus Deep Blue, sebuah computer permainan catur, mengalahkan Garry Kasparov dalam sebuah pertandingan 6 *game* yang terkenal pada tahun 1997. DARPA menyatakan bahwa biaya yang disimpan melalui penerapan metode AI untuk unit penjadwalan dalam Perang Teluk pertama telah mengganti seluruh investasi dalam penelitian AI sejak tahun 1950 pada pemerintah AS.

Tantangan Hebat DARPA, yang dimulai pada 2004 dan berlanjut hingga hari ini, adalah sebuah pacuan untuk hadiah $2 juta dimana kendaraan dikemudikan sendiri tanpa komunikasi dengan manusia, menggunakan GPS, komputer dan susunan sensor yang canggih, melintasi beberapa ratus mil daerah gurun yang menantang[1].

**II.8 Definisi Kecerdasan Buatan**

Tidak ada definisi yang memuaskan untuk kecerdasan. Kecerdasan dapat diartikan sebagai kemampuan untuk memperoleh pengetahuan dan menggunakannya atau kecerdasan adalah apa yang di ukur oleh sebuah ”test kecerdasan”.

Apa kecerdasan buatan itu? Bagian dari ilmu pengetahuan komputer ini khusus ditujukan dalam perancangan otomatisasi tingkah laku cerdas dalam system kecerdasan komputer. Sistem memperlihatkan sifat-sifat khas yang dihubungkan dengan kecerdasan dalam kelakuan atau tindak-tanduk yang sepenuhnya bias menirukan beberapa fungsi otak manusia, seperti pengertian bahasa, pengetahuan,pemikiran, pemecahan masalah, dan lain sebagainya.

Kecerdasan Buatan (*Artificial intelligence*) didefinisikan sebagai kecerdasan yang ditunjukkan oleh suatu entitas buatan. Sistem seperti ini umumnya dianggap komputer. Kecerdasan diciptakan dan dimasukkan ke dalam suatu mesin (Komputer) agar dapat melakukan pekerjaan seperti yang dapat dilakukan manusia. Beberapa macam bidang yang menggunakan kecerdasan buatan antara lain sistem pakar, permainan komputer (*game*s), logika fuzzy, jaringan syaraf dan robotika[1].

Walaupun *Artificial intelligence* memiliki konotasi fiksi ilmiah yang kuat, *Artificial intelligence* membentuk cabang yang sangat penting pada ilmu komputer, berhubungan dengan perilaku, pembelajaran dan adaptasi yang cerdas dalam sebuah mesin. Penelitian dalam *Artificial intelligence* menyangkut pembuatan mesin untuk mengotomatisasikan tugas-tugas yang membutuhkan perilaku cerdas. Termasuk contohnya adalah pengendalian, perencanaan dan penjadwalan, kemampuan untuk menjawab diagnosa dan pertanyaan pelanggan, serta pengenalan tulisan tangan, suara dan wajah. Hal-hal seperti itu telah menjadi disiplin ilmu tersendiri, yang memusatkan perhatian pada penyediaan solusi masalah kehidupan yang nyata. Sistem AI sekarang ini sering digunakan dalam bidang ekonomi, obat-obatan, teknik dan militer, seperti yang telah dibangun dalam beberapa aplikasi perangkat lunak komputer rumah dan video *game*.

## II.9 Macam – Macam Kecerdasan Buatan

Ada banyak jenis kecerdasan buatan, setidaknya ada lima jenis kecerdasan buatan yang sering kita temui, yaitu :

1. Jaringan Syaraf Buatan (*Artificial Neural Networks*)

   Merupakan sekelompok jaringan saraf (neuron) buatan yang menggunakan model matematis atau komputasi untuk pemrosesan informasi berdasarkan pendekatan terhubung pada komputasi. Pada kebanyakan kasus, JST merupakan system adaptif yang merubah strukturnya berdasarkan informasi eksternal maupun internal yang mengalir melalui jaringan tersebut[2].

2. Logika Fuzzy (*Fuzzy Logics*)

   Adalah peningkatan dari logika Boolean yang berhadapan dengan konsep kebenaran sebagian. Di mana logika klasik menyatakan bahwa segala hal dapat diekspresikan dalam istilah binary (0 atau 1, hitam atau putih, ya atau tidak), logika fuzzy menggantikan kebenaran boolean dengan tingkat kebenaran.Logika Fuzzy memungkinkan nilai keanggotaan antara 0 dan 1, tingkat keabuan dan juga hitam dan putih, dan dalam bentuk linguistik, konsep tidak pasti seperti "sedikit", "lumayan", dan "sangat". Dia berhubungan dengan set fuzzy dan teori kemungkinan. Dia diperkenalkan oleh Dr. Lotfi Zadeh dari Universitas California, Berkeley pada 1965[2].

3. Algoritma Genetik (*Genetic Algorithms*)

   Adalah teknik pencarian yang di dalam ilmu komputer untuk menemukan penyelesaian perkiraan untuk optimisasi dan masalah pencarian. Algoritma genetik adalah kelas khusus dari algoritma evolusioner dengan menggunakan teknik yang terinspirasi oleh biologi evolusioner seperti warisan, mutasi, seleksi alam dan rekombinasi (atau crossover).Algoritma Genetik biasanya digunakan dibidang kedokteran, misal untuk menganalisis DNA[2].

**II.10 Depth First Search**

Pada Depth First Search (DFS), proses akan dilakukan pada semua anaknya sebelum dilakukan pencarian ke node-node (titik) yang selevel. Pencarian dimulai dari node akar ke level yang lebih tinggi. Proses ini diulangi terus hingga ditemukannya solusi. Stack atau tumpukan adalah struktur data yang setiap proses baik penambahan maupun penghapusan hanya bisa dilakukan dari posisi teratas tumpukan. Cara kerja stack adalah LIFO (Last In First Out), dimana data yang terakhir masuk akan keluar pertama[2].

Berikut analisis ruang dan waktu untuk metode pencarian DFS :

1. Diasumsikan :
   a. Pohon pelacakan memiliki cabang yang selalu sama, yaiu sebanyak b.
   b. Tujuan dicapai pada level ke-d.
2. Analisis Ruang
   a. Setelah berjalan 1 langkah, stack akan berisi b node.
   b. Setelah berjalan 2 langkah, stack akan berisi (b-1) + b node.
   c. Setelah berjalan 3 langkah, stack akan berisi (b-1) + (b-1) + b node.
   d. Setelah berjalan d langkah, stack akan berisi (b-1) * d + 1 node, mencapai maksimum.
3. Analisis Waktu
   a. Pada kasus terbaik, DFS akan mencapai tujuan pada kedalaman d pertama, sehingga dibutuhkan pencarian sebanyak d + 1 node.
   b. Pada kasus terburuk, DFS akan mencapai tujuan pada kedalaman d pada node terakhir, sehingga dibutuhkan pencarian sebanyak 1 + b + b2 + b3 +….+ bd = (bd+1-1) / ( b-1)

Keuntungan dari metode ini adalah :

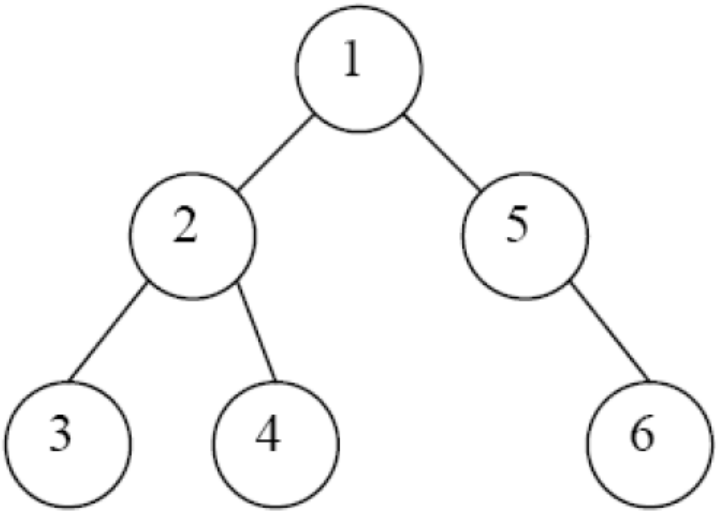
1. Membutuhkan memori yang relatif kecil, karena hanya node-node pada lintasan yang aktif saja yang disimpan.
2. Secara kebetulan, metode DFS akan menemukan solusi tanpa harus menguji lebih banyak lagi dalam ruang keadaan.

Kelemahan dari metode ini adalah :

1. Memungkinkan tidak ditemukannya tujuan yang diharapkan.
2. Hanya akan mendapatkan 1 solusi pada setiap pencarian.

Pada gambar 2.2, penelusuran dimulai dari simpul akar bernomor 1. Simpul berikutnya yang ditelusuri adalah simpul 2 yang bertetangga dengan simpul 1, lalu simpul 3 yang bertetangga dengan simpul 2. Karena simpul 3 sudah tidak memiliki tetangga, penelusuran akan berlanjut ke tetangga simpul 2 yaitu simpul

4. Setelah itu simpul 5 yang bertetangga dengan simpul 1, dan terakhir simpul 6 yang bertetangga dengan simpul 5.




**Gambar II. 5 Contoh Penelusuran DFS**

Untuk memecahkan persoalan memaksimalkan f(H), dilakukan penelusuran terhadap semua himpunan bagian Ai yang saling lepas. Setiap simpul berisi himpunan solusi biji-biji yang akan ditaruh dan jumlah nilai dari himpunan solusi.

## II.11 Algoritma minimax

Algoritma Minimax merupakan algoritma yang digunakan untuk menentukan pilihan agar memperkecil kemungkinan kehilangan nilai maksimal. Algoritma ini diterapkan dalam permainan yang melibatkan dua pemain seperti catur, tic tac toe, checkers, go dan permainan yang menggunakan strategi atau logika lainnya. Hal ini berarti permainan-permainan tersebut dapat dijelaskan sebagai suatu rangkaian aturan dan premis.

Algoritma ini mulai dikembangkan dari teori *game* zero-sum. Teori ini mendeskripsikan situasi dimana jika terdapat pemain yang mengalami pendapatan, pemain lain akan mengalami kehilangan dengan nilai yang sama dari

pendapatan tersebut, dan sebaliknya. Jumlah pendapatan dari pemain yang dikurangi dengan jumlah kehilangan akan berjumlah nol[4].

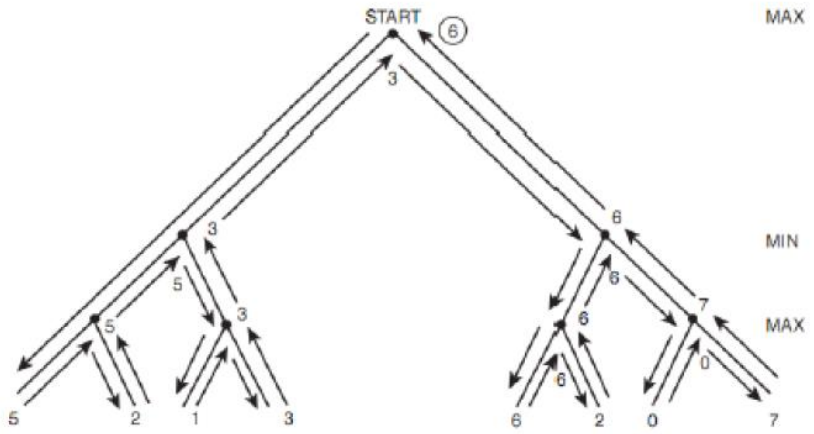
Teori minimax menyatakan :

Untuk setiap dua orang pemain dalam zero-sum *game*, terdapat nilai V dari strategi yang dimiliki pemain seperti :

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

Secara setara, strategi pemain pertama akan memastikan suatu nilai V tanpa memperdulikan strategi pemain kedua, dan bersamaan dengan itu pemain kedua akan memastikan dirinya kehilangan nilai sebesar –V. Algoritma Minimax merupakan algoritma dasar pencarian DFS (Depth-First Search) untuk melakukan traversal dalam pohon. DFS akan mengekspansi simpul paling dalam terlebih dahulu. Setelah simpul akar dibangkitkan, algoritma ini akan membangkitkan simpul pada tingkat kedua, yang akan dilanjutkan pada tingkat ketiga, dst. Dalam melakukan treversal, misalkan dimulai dari suatu simpul i, maka simpul selanjutnya yang akan dikunjungi adalah simpul tetangga j, yang bertetangga dengan simpul k, selanjutnya pencarian dimulai lagi secara rekursif dari simpul j. Ketika telah mencapai simpul m, dimana semua simpul yang bertetangga dengannya telah dikunjungi, pencarian akan dirunut balik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul yang belum dikunjungi. Selanjutnya pencarian dimulai kembali dari j. Ketika tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi maka pencarian selesai.

Dalam repersentasi 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 merupakan gambar pohon untuk algoritma Minimax.



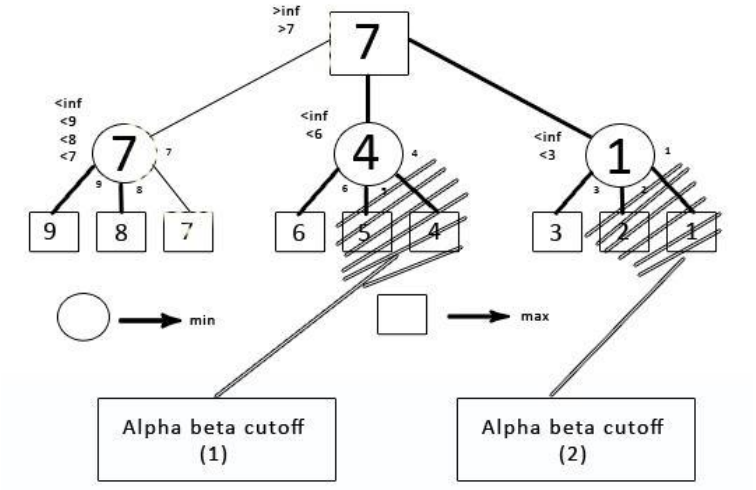

**Gambar II. 6 Pohon Pencarian Algoritma Minimax**

Dari gambar II.7, proses pencarian dimulai dari jalur paling kiri terlebih dahulu, sehingga akan ditelusuri simpul paling kiri bawah yaitu 5. Nilai 5 disimpan sebagai nilai maksimum sementara karena berada di tingkat max, kemudian dilakukan backtrack dan menelusuri simpul tetangga dari simpul 5 yaitu simpul 2. Karena nilai 5 lebih besar dari nilai 2, maka nilai 2 tidak disimpan. Lalu dilakukan backtrack ke tingkat min sehingga nilai 5 yang diperoleh akan disimpan sebagai nilai minimum sementara. Untuk simpul 1 dan 3, nilai simpul 3 akan disimpan karena merupakan nilai maksimum di tingkat max. saat mencapai tingkat min, nilai 5 sebagai nilai minimum sementara akan digantikan oleh nilai 3, karena nilai 3 lebih kecil dibandingkan nilai 5, kemudian dilakukan backtracking dan dilanjutkan dengan penelusuran ke jalur kanan hingga dihasilkan nilai 6 pada tingkat min. karena nilai maksimum sementara pada tingkat paling atas adalah 3, maka nilai 3 akan digantikan dengan nilai 6 karena nilai 6 lebih besar dari nilai 3. Dengan demikian jalur yang akan dipilih menggunakan algoritma minimax adalah jalur sebelah kanan.

## II.12 Alpha-Beta Pruning

Dalam algoritma Minimax, pencarian dilakukan pada seluruh bagian pohon, sementara sebagian pohon tidak seharusnya diperiksa. Alpha-beta pruning

merupakan modifikasi dari algoritma Minimax, yang akan mengurangi jumlah node yang dievaluasi oleh pohon pencarian. Pencarian untuk node berikutnya akan dipikirkan terlebih dahulu. Algoritma ini akan berhenti mengevaluasi langkah ketika terdapat paling tidak satu kemungkinan yang ditemukan dan membuktikan bahwa langkah tersebut lebih buruk jika dibandingkan dengan langkah yang diperiksa sebelumnya. Sehingga, langkah berikutnya tidak perlu dievaluasi lebih jauh[4]. Dengan algoritma ini hasil optimasi dari suatu algoritma tidak akan berubah. Berikut merupakan pohon dengan algoritma alpha-beta pruning :




**Gambar II. 7 Pohon Pencarian Algoritma Minimax dengan Alpha-Beta Pruning**

Diperlihatkan, pada pohon tersbut, bahwa terdapat pemotongan pencarian dengan menggunakan algoritma ini. Pada algoritma ini, terdapat dua nilai yang diatur, yaitu alpha dan beta, yang merepresentasikan nilai minum dari max yang diyakini dan nilai maksimum dari min yang diyakini. Nilai awal alpha adalah tak hingga negative dan nilai awal beta adalah tak hingga positif. Sebagai hasil dari proses rekursif, area pencarian akan semakin kecil. Ketika beta menjadi lebih kecil dari alpha, akan berarti posisi saat itu tidak dapat menjadi hasil terbaik permainan untuk kedua pemain dan pencarian tidak perlu dilakukan lebih jauh.

## II.13 UML

UML singkatan dari *Unified Modeling Language* yang berarti bahasa pemodelan standard. (Chonoles, 2003:bab 1) mengatakan sebagai bahasa, berarti UML memiliki sitak dan semantic. Ketika kita membuat model menggunakan konsep UML ada aturan-aturan yang harus diikuti. Bagaimana elemen pada model-model yang kita buat berhubungan satu dengan yang lainnya harus mengikuti standar yang ada. UML bukan hanya sekedar diagram, tetapi juga menceritakan konteksnya[5].

UML diaplikasikan untuk maksud tertentu, biasanya antara lain untuk :

1. Merancang perangkat lunak
2. Sarana komunikasi antara perangkat lunak dengan proses bisnis.
3. Menjabarkan sistem secara rinci untuk analisa dan mencari apa yang diperlukan sistem.
4. Mendokumentasi sistem yang ada, proses-proses dan organisasinya.

### II.13.1 Diagram-Diagram UML

Beberapa literatur menyebutkan bahwa UML menyediakan sembilan jenis diagram, yang lain menyebutkan delapan karena ada beberapa diagram yang digabung, misalnya diagram komunikasi, diagram urutan dan diagram pewaktuan digabung menjadi diagram interaksi. Namun demikian model-model itu dapat dikelompokan berdasarkan sifatnya yaitu statis atau dinamis. Jenis diagram itu antara lain :

1. Diagram Kelas. Bersifat statis. Diagram ini memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi, serta relasi-relasi. Diagram ini umum dijumpai pada pemodelan sistem berorientasi objek. Meskipun bersifat statis, sering pula diagram kelas memuat kelas-kelas aktif.
2. Diagram Paket (*Package Diagram*). Bersifat statis. Diagram ini memperlihatkan kumpulan kelas-kelas, merupakan bagian dari diagram komponen.

3. Diagram *Use-Case.* Bersifat statis. Diagram ini memperlihatkan himpunan use-case dan actor-aktor (suatu jenis khusus dari kelas). Diagram ini terutama sangat penting untuk mengordinasikan dan memodelkan perilaku suatu sistem yang dibutuhkan serta diharapkan pengguna.
4. Diagram interaksi dan *sequence* (urutan). Bersifat dinamis. Diagram urutan adalah diagram interaksi yang menekankan pada pengiriman pesan dalam suatu waktu tertentu.
5. Diagram Komunikasi (*Communication Diagram)*. Bersifat dinamis. Diagram sebagai pengganti diagramkolaborasi UML 1.4 yang menekankan organisasi struktural dari objek-objek yang menerima serta mengirim pesan.
6. Diagram *Statechart* (Statechart Diagram). Bersifat dinamis. Diagram status memperlihatkan keadaan-keadaan pada sistem, memuat status (*state)*, transisi, kejadian serta aktifitas. Diagram ini terutama penting untuk memperlihatkan sifat dinamis dari antarmuka (*interface)*, kelas, kolaborasi dan terutama penting pada pemodelan sistem-sistem yang reaktif.
7. Diagram Aktivitas (*Activity Diagram)*. Bersifat dinamis. Diagram aktivitas adalah tipe khusus dari diagram status yang memperlihatkan aliran dari suatu aktivitas ke aktivitas lainnya dalam suatu sistem. Diagram ini terutama penting dalam pemodelan fungsi-fungsi suatu sistem dan memberi tekanan pada aliran kendali antar objek.
8. Diagram komponen (*Component Diagram).* Bersifat statis. Diagram komponen ini memperlihatkan organisasi serta kebergantungan sistem/perangkat lunak pada komponen-komponen yang telah ada sebelumnya. Diagram ini berhubungan dengan diagram kelas dimana komponen secara tipikal dipetakan kedalam satu atau lebih kelas-kelas, antaramuka-antarmuka serta kolaborasi-kolaborasi.
9. Diagram Deployment (*Deployment Diagram).* Bersifat statis. Diagram ini memperlihatkan konfigurasi saat aplikasi dijalankan (run-time). Memuat simpul-simpul beserta komponen-komponen yang ada di dalamnya. Diagram deployment berhubungan erat dengan diagram komponen dimana diagram ini memuat satu atau lebih komponen-komponen. Diagram ini sangat berguna

saat aplikasi kita berlaku sebagai aplikasi yang dijalankan pada banyak mesin (*distributed computing).*

Kesembilan diagram ini tidak mutlak harus digunakan dalam pengembangan perangkat lunak, semuanya dibuat sesuai dengan kebutuhan. Pada UML dimungkinkan kita menggunakan diagram-diagram lain (misalnya data *flow diagram*, *Entity Relationship* diagram dan sebagainya).

### II.14 Metode Pengujian Sistem

Pengujian sistem adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain dan pengkodean. Pengujian menyajikan anomali yang menarik bagi perekayasa perangkat lunak. Pada proses perangkat lunak, perekayasa pertama-tama berusaha membangun perangkat lunak dari konsep abstrak ke implementasi yang dapat dilihat, dan dilanjutkan dengan pengujian[8].

### II.14.1 *White Box*

Pengujian *white box* dilakukan untuk menguji prosedur-prosedur yang ada. Lintasan lojik yang dilalui oleh setiap bagian prosedur diuji dengan memberikan kondisi/loop spesifik. Pengujian *white box* menjamin pengujian terhadap semua lintasan yang tidak bergantungan minimal satu kali, mencoba semua keputusan lojik dari sisi '*true*' dan '*false*', eksekusi semua loop dalam batasan kondisi dan batasan operasionalnya dan pengujian validasi data internal[8]. Konsep pengujian basis path pada *white box* adalah sebagai berikut:

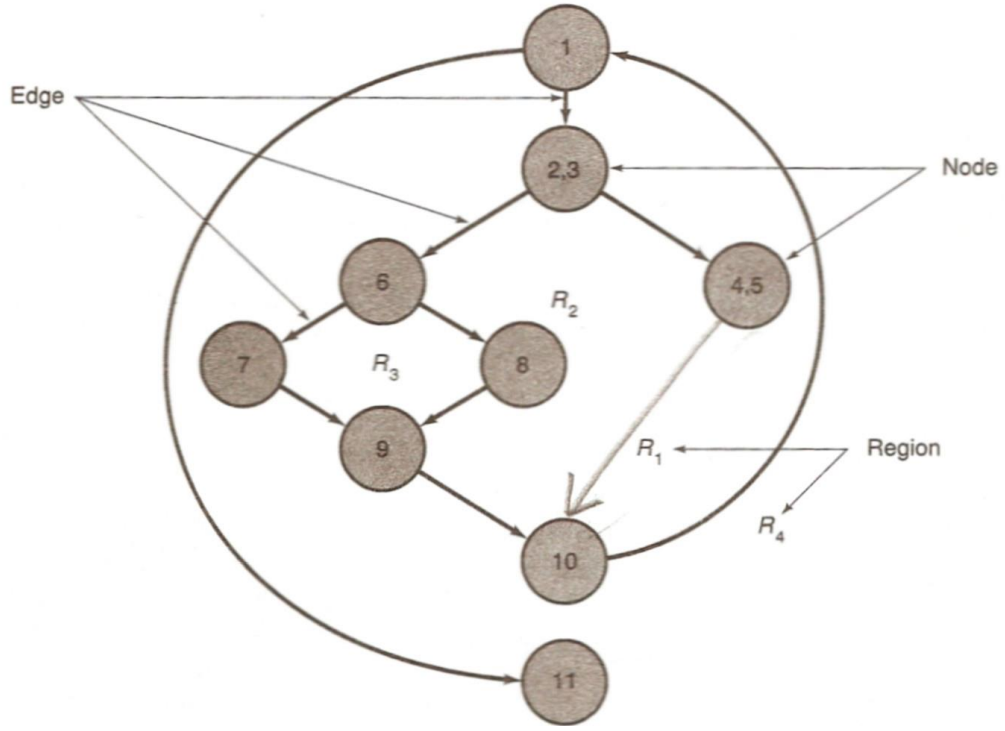
1. Merupakan bagian dari pengujian *white-box* dalam hal pengujian prosedurprosedur.
2. Mempergunakan notasi aliran graph (node, link untuk merepresentasikan sequence, if, while, until dll.).
3. Konsep kompleksitas *cyclomatic* antara lain cara perhitungan daerah tertutup pada graph planar dimana dapat menghubungkan batas atas jumlah pengujian

yang harus direncanakan dan dieksekusi untuk menjamin pengujian seluruh statement program.

4. Memunculkan kasus-kasus yang akan diuji dengan membuat daftar lintasan kasus pengujian berdasarkan kompleksitas dan *cyclomatic* yang didapat.
5. Membuat alat bantu matrik *graph* yang membantu pengawasan pengujian.

### II.14.1.1 Notasi graf Alur (*Path Graph Notation*)

Notasi sederhana untuk merepresentasikan alur kontrol disebut graf alur (*flow graph*), terdiri dari *edge*, node dan *region*. Sebuah *edge* harus berakhir pada sebuah node walaupun tidak semua node merepresentasikan perintah prosedural. Area yang dibatasi oleh edge dan node disebut region, area diluar graph juga dihitung sebagai *region*[8].




**Gambar II. 8 *Flowgraph***

### II.14.1.2 *Cyclomatic Complexity*

*Cyclomatic complexity* merupakan *software metric* yang menyediakan ukuran kuantitatif dari komplesitas logikal suatu program. Ketika digunakan dalam

konteks metode ujicoba berbasis alur, nilai yang dikomputasi untuk kompleksitas cyclomatic mendefinisikan jumlah independent path dalam himpunan basis suatu program dan menyediakan batas atas untuk sejumlah ujicoba yang harus dilakukan untuk memastikan bahwa seluruh perintah telah dieksekusi sedikitnya satu kali[8].

*Independent path* adalah alur manapun dalam program yang memperkenalkan sedikitnya satu kumpulan perintah pemrosesan atau kondisi baru. Contoh *independent path* dari gambar II.8 :

*Path* 1 : 1 – 11

*Path* 2 : 1 – 2 – 3 – 4 – 5 – 10 – 1 – 11

*Path* 3 : 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11

*Path* 4 : 1 – 2 – 3 – 6 – 7 – 9 – 10 – 1 – 11

Misalkan setip *path* yang baru memunculkan edge yang baru, dengan *path* :

1 - 2 – 3 – 4 – 5 – 10 - 1 - 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11

*path* diatas tidak dianggap sebagai independent path karena kombinasi path diatas telah didefinisikan sebelumnya Ketika ditetapkan dalam graf alur, maka independent path harus bergerak sedikitnya 1 *edge* yang belum pernah dilewati sebelumnya. Kompleksitas *cyclomatic* dapat dicari dengan salah satu dari 3 cara berikut :

1. Jumlah region dari graf alur mengacu kepada komplesitas *cyclomatic*
2. Kompleksitas *cyclomatic* untuk graf alur G didefinisikan :
   V(G) = E – N + 2
   Dimana E = jumlah edge, dan N = jumlah node
3. Kompleksitas *cyclomatic* untuk graf alur G didefinisikan :
   V(G) = P + 1
   Dimana P = jumlah *predicates nodes*

### II.14.1.3 *Graph Matrix*

Prosedur untuk menghasilkan graf alur dan menentukan himpunan alur basis dapat di mekanisasi. Untuk membangun *tool software* yang membantu dalam

ujicoba berbasis alur, struktur data yang disebut *graph matrix*. *Graph matrix* merupakan matrik persegi yang jumlah baris dan kolomnya sesuai dengan jumlah node pada graf alur. Setiap baris dan kolom mengacu kepada sebuah node dan isi dari *matrix* mengacu kepada edge[8].

*Graph Matrix* merupakan representasi tabular dari flow graph, dengan menambahkan link weight untuk setiap input pada matrik, maka matrik graf menjadi alat bantu yang lebih berguna untuk mengevaluasi struktur kontrol program selama ujicoba. link weight menyediakan informasi tambahan mengenai aliran kontrol. Dengan bentuk sederhana, yaitu berikan nilai 1 jika ada koneksi antar node dan nilai 0 jika tidak ada koneksi. Hal lain yang dapat diketahui melalui matrik graf :

1. Kemungkinan suatu link/ edge akan dieksekusi
2. Waktu proses yang dibutuhkan selama traversal suatu link
3. Memori yang diperlukan selama traversal suatu link
4. Sumber daya yang diperlukan selama traversal suatu link

### II.14.2 Black Box

Pengujian yang dilakukan untuk antarmuka perangkat lunak, pengujian ini dilakukan untuk memperlihatkan bahwa fungsi-fungsi bekerja dengan baik dalam arti masukan yang diterima dengan benar dan keluaran yang dihasilkan benar-benar tepat, pengintegrasian dari eksternal data berjalan dengan baik. Metode pengujian *black-box* memfokuskan pada *requirement* fungsi dari perangkat lunak, pengujian ini merupakan komplenetari dari pengujian *white-box*[8].

Pengujian *white-box* dilakukan terlebih dahulu pada proses pengujian, sedangkan pengujian *black-box* dilakukan pada tahap akhir dari pengujian perangkat lunak. Proses yang terdapat dalam proses pengujian *black-box* antara lain sebagaiberikut:

1. Pembagian kelas data untuk pengujian setiap kasus yang muncul pada pengujian *white-box*.
2. Analisis batasan nilai yang berlaku untuk setiap data.

## II.16 Game Maker

Game Maker adalah *game* engine yang sangat populer untuk membuat sebuah *game* karena adanya sistem drag and drop. Game Maker memungkinkan pengguna untuk membuat video *game* tanpa memiliki pengetahuan mengenai bahasa pemrograman. Game Maker memberikan resource yang cukup banyak, dari segi grafis 2D hingga 3D, sprite maker, sound, script, path dan sebagainya.

Game Maker dibuat oleh Mark Overmars, seorang profesor dari Institut Ilmu Komputer dan Informasi di Universitas Utrecht, dengan pemrograman Delphi 7. Awalnya pada tahun 1999 Game Maker bernama Animo, sebuah program yang memiliki sistem pembuat animasi dua dimensi. Namun, pada tahun yang sama Animo telah berubah fungsi menjadi sebuah *game* engine tetapi tidak dapat membuat sebuah program executable, hanya bisa dimainkan pada jendela *Game Maker*. Seiring perkembangan teknologi, Game Maker dapat membuat *game* utuh dan mendukung DirectX.