BAB 2
LANDASAN TEORI

2.1 Game

Dalam kamus bahasa Indonesia game diartikan sebagai permainan. Permainan merupakan bagian dari bermain dan bermain juga bagian dari permainan keduanya saling berhubungan. *Game* atau permainan adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah, biasanya dalam konteks tidak serius atau dengan tujuan refreshing.

Maka permainan adalah kegiatan yang kompleks yang didalamnya terdapat peraturan, permainan dan budaya. Sebuah permainan adalah sebuah sistem dimana pemain terlibat dalam konflik buatan, disini pemain berinteraksi dengan sistem dan konflik dalam permainan merupakan rekayasa atau buatan, dalam permainan terdapat peraturan yang bertujuan untuk membatasi perilaku pemain dan menentukan permainan. Permainan biasanya dilakukan sendiri atau bersama-sama (kelompok). Beberapa permainan tercipta di masa lalu yang biasa disebut dengan permainan tradisional, dan beberapa permainan yang lebih akhir atau masa sekarang dan biasanya menggunakan peralatan yang canggih disebut permainan modern.

2.1.1 Sejarah Singkat Game

Di antara jajaran nama tokoh penemuan penting, nama besar seorang penemu permainan komputer, atau lebih lazim kita menyebutnya dengan game, jarang sekali disebutkan. Namun bila penemuannya sampai sekarang masih digunakan dan bahkan sangat diminati dan dinikmati. Sebagian orang bahkan saat ini mengangap game sebagai sebuah kebutuhan yang harus terpenuhi.

Game pertama kali dibuat pada tahun 1966 oleh Ralph Baer bersama dengan timnya yang berjumlah 500 orang yang terdiri dari para insinyur dan teknisi. Permainan yang pertama sekali ditemukan hanya dapat dimainkan oleh komputer seharga US$ 40.000. Dimana pengembangan dan pembuatan game tersebut didanai oleh Pentagon.

Pada tahun 1965, pihak militer datang kepada Bear dan meminta menciptakan simulasi komputer yang dapat membantu pasukan untuk belajar strategi dan mengukur kemampuan reflek. Proyek ini dikerjakan dengan tingkat keamanan yang sangat ketat di tengah situasi perang dingin.

Setelah sebulan bekerja keras, Bear berhasil menampilkan dua titik putih yang berkejar-kejaran di layar. Hal ini membuat pihak militer merasa kagum dan memberikan dana yang lebih besar lagi sehingga ia dapat menyewa asisten lebih banyak. Tim ini kemudian berhasil membuat permainan antara papan dan bola yang pada akhir tahun 1966 dipresentasikan di depan para pejabat Pentagon. Namun sayang, ternyata mereka tak merasa tertarik dengan penemuan Bear. Hingga akhirnya ia berusaha memperoleh izin agar dapat memproduksi mesin permainan secara komersial.

Tahun 1970, Bill Enders yang tergabung dalam sebuah organisasi dengan nama Megnavox mencoba menyakinkan edukatif Megnavox untuk memberikan kesempatan pada Bear dan mesin permainannya. Hasilnya adalah munculnya video game komersial pertama, yaitu Magnavox Odyssey.

Tahun 1972, muncul nama baru dalam dunia game. Nolan Bushnel. Pada tanggal 27 Juni 1972, ia mendirikan perusahaan Atari, dan membuat game arcade Pong. Era baru dalam perkembangan dunia game terjadi pada tahun 1988, yang di dominasi oleh perusahaan di Jepang. Nintendo, yang awalnya hanya memproduksi mesin fotocopi, beralih memproduksi game.

Saat ini permainan komputer telah berkembang sedemikian pesatnya seiring dengan perkembangan hardware yang mendukung. Dengan program permainan yang lebih kompleks dan tampilan grafis 3D yang luar biasa.

2.1.2 Video Games

Game merupakan sebuah bentuk seni dimana penggunanya, disebut dengan pemain (*player*), diharuskan membuat keputusan-keputusan dengan tujuan untuk mengelola sumber daya yang diperoleh dari kesempatan-kesempatan bermain (*token*) miliknya, untuk mencapai suatu tujuan tertentu [8]. Video game adalah bentuk *game* yang interaksi utamanya melibatkan media video (dan

biasanya melibatkan audio). Sistem elektronik yang digunakan untuk menjalankan permainan video disebut *platform*, contohnya adalah komputer pribadi *(personal computer)* ataupun konsol permainan *(game console)*.

1. Permainan Komputer (*Computer Game*)

   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 sangat kompleks dan mutakhir.

   Permainan *online (online game)* adalah jenis permainan video atau permainan komputer dengan menggunakan jaringan komputer dan juga internet sebagai media interaksinya. Permainan online terdiri dari dua unsur utama, yaitu *server* dan *client*. *Server* adalah penyedia layanan gaming yang merupakan basis agar *client-client* yang terhubung dapat memainkan permainan dapat melakukan komunikasi dengan baik. Suatu server pada prinsipnya hanya melakukan administrasi permainan dan menghubungkan client-client. Sedangkan *client* adalah pengguna permainan dan memakai kemampuan server. Contoh permainan online adalah *Ragnarok Online, Dragonica Online* dan masih banyak lagi.

2. Konsol Permainan (*Game Console*)

   Konsol permainan adalah sebuah mesin elektronik yang dirancang khusus untuk memainkan permainan video. Perangkat keluarannya biasa berupa monitor komputer atau televisi dan alat masukan utamanya berupa sebuah pengendali. Konsol permainan yang pertama kali dibuat adalah *Atari*, kemudian dilanjutkan dengan *Nintendo* yang sukses pada tahun 1985-1989. Konsol permainan modern sekarang ini antara lain adalah *X-Box, Playstation*. Untuk Console yang berbentuk kecil dan dapat dibawah kemana-mana biasa disebut Portable Console.

2.1.3 Klasifikasi Game

Berikut ini adalah klasifikasi permainan berdasarkan pembatasan usia dari pemainnya, menurut ESRB (*Entertainment Software Rating Board*) adalah sebagai berikut [10]:

a. *Early Childhood* (3+) memiliki isi yang mungkin cocok untuk usia 3 tahun keatas. Tidak mengandung materi yang tidak pantas seperti adegan kekerasan.
b. *Everyone* (6+) memiliki konten yang mungkin cocok untuk usia 6 keatas. Judul-judul dalam kategori ini mungkin berisi sedikit kartun, fantasi atau kekerasan ringan dan tidak jarang menggunakan bahasa yang ringan.
c. *Everyone* (10+) memiliki konten yang mungkin cocok untuk usia 10 keatas. Judul-judul dalam kategori ini mungkin berisi lebih banyak kartun, fantasi atau kekerasan ringan, bahasa ringan dan juga minimal tema sugestif.
d. *Teen* (13+) memiliki konten yang mungkin cocok untuk usia 13 keatas. Judul-judul dalam kategori ini mungkin mengandung kekerasan, tema sugestif, humor kasar, sedikit darah, simulasi perjudian, dan tidak jarang menggunakan bahasa yang kuat.
e. *Mature* (17+) memiliki konten yang mungkin cocok untuk orang usia 17 dan lebih tua. Judul-judul dalam kategori ini mungkin berisi intens kekerasan, darah, konten seksual dan bahasa yang kuat.
f. *Adults Only* (18+) memiliki konten yang hanya boleh dimainkan oleh orang-orang 18 tahun keatas. Judul-judul dalam kategori ini mungkin termasuk adegan yang intens, kekerasan dan juga mengandung konten seksual.
g. *RP (Rating Pending)* telah disampaikan kepada ESRB dan sedang menunggu penilaian akhir. (Simbol ini hanya muncul dalam iklan sebelum permainan dilepas kepasaran).

2.1.4 Sudut Pandang Game

Sebuah game biasanya mempunyai sudut pandang permainan tersendiri disesuaikan berdasarkan genre game yang diambil. Berikut beberapa macam sudut pandang permainan yang biasa digunakan [8]:

1. *Side Scrolling*

   Adalah sudut pandang permainan yang terlihat dari samping dan memungkinkan karakter utama untuk bergerak dari kiri ke kanan serta memungkinkan *background* pada game seolah-olah bergeser mengikuti pergerakan karakter utama.

2. *Top Down*

   Adalah sudut pandang permainan yang memungkinkan karakter utama bermanuver ke empat arah namun cara permainannya sendiri bergeser dari bawah ke atas, dan biasanya game yang menggunakan sudut pandang permainan jenis ini adalah *shooter game.*

3. *Isometric*

   Adalah sudut pandang permainan yang memungkinkan permainan terlihat diantara sisi *Side Scrolling* dan juga *Top Down*, dan biasanya diterapkan pada game dengan genre *RTS (Real Time Strategy).*

2.1.5 Jenis-Jenis Game

Game dari tahun-tahun semakin banyak saja jenisnya, teknologi yang dipakaipun semakin canggih. Nah dari ratusan atau bahkan ribuan game yang telah bermunculan di dunia, game-game tersebut dikelompokkan menjadi beberapa genre. Pada dasarnya, genre-genre game yang beredar di pasaran dapat dikelompokkan sebagai berikut [8].

1. *Action Game*

   Adalah tipe game dengan fitur utama berupa banyaknya aksi di mana pemain harus memiliki keterampilan reaksi yang cepat untuk menghindari musuh atau menghindari rintangan. Pengembang game tipe ini perlu memastikan game yang dibuat dioptimasi sehingga pemain memiliki

pengalaman bermain yang baik, yang tidak terganggu oleh delay proses yang lama, contohnya : *Metal Gear Solid.*

2. *Top Down Adventure Game*

Adalah tipe game yang umumnya membuat pemain harus berjalan mengelilingi suatu tempat yang terkondisi, seperti sebuah istana, gua yang berkelok, dan planet yang jauh. Pemain melakukan navigasi suatu area, mencari pesan-pesan rahasia, memperoleh obyek yang memiliki kemampuan yang bervariasi, bertempur dengan musuh, dan lain-lain. Untuk membuat game ini, diperlukan perencanaan yang akurat sehingga memiliki alur cerita yang menarik bagi pemain, contohnya : *Resident Evil, Tomb Raider*.

3. *Sport Game*

Adalah tipe game yang berupa kompetisi antara dua pemain atau lebih, dimana pemain dapat berupa individual atau tim. Contoh game tipe ini antara lain sepakbola, bola basket, tenis, dan bilyard. Tergantung seberapa cepat permainan yang terjadi, aplikasi game perlu dioptimalkan, contohnya : *Pro Evolution Soccer.*

4. *RPG* (*Role Playing Game*)

Adalah tipe game yang seringkali berupa multi karakter game di mana setiap pemain memiliki karakter dengan kemampuan, kekuatan, dan kelemahan yang spesifik. Para pemain saling berkompetisi, berinteraksi, dan bertempur satu sama lain. Tampilan grafis yang khas untuk setiap karakter pemain ditambah dengan storyline yang mendebarkan akan sangat menarik dan memberikan pengalaman yang berbeda di dalam bermain, contohnya : *Final Fantasy.*

5. *Platform Game*

Adalah tipe game yang mengharuskan pemain mengarahkan suatu obyek dengan melalui berbagai tahap atau tingkatan area untuk menyerang musuh dan menghindar terhadap serangan. Tipe game ini sedikit serupa dengan action game, tetapi aksinya tidak secepat action game. Teknik collision detection sangat sering dimanfaatkan pada tipe game ini, contohnya: *Sonic The Hedgehog, Mario Bros.*

6. *Puzzle Game*

Adalah tipe game yang umumnya membuat pemain menggunakan kemampuan berpikirnya sebagai pengganti keterampilan reaksi yang cepat karena terdapat rahasia yang perlu dipecahkan. Game ini lebih bersifat statis dibanding action game. Pembuatan game tipe ini seringkali ditunjang dengan *AI (Artificial Intelligence)*, contohnya : *7 Wonders of The Ancient World*

7. *Sandbox Game*

Adalah tipe *game* yang umumnya ditujukan untuk menjelajahi suatu kota atau tempat dan bebas berinteraksi dengan objek disekitarnya. Mungkin *game* ini bisa di bilang mirip RPG tapi *Sandbox* sangat berbeda dengan RPG *game*, perbedaannya adalah tidak adanya peningkatan level karakternya, contohnya : *GTA, Bully*.

8. *FPS* (*First Person Shooter*)

Adalah tipe *game* yang mengunakan sudut pandang orang pertama untuk membidik atau membunuh musuh, sehingga kita hanya melihat tangannya saja dan tidak melihat tubuh karakter yang dimainkan, contohnya : *Call of Duty*.

9. *TPS* (*Third Person Shooter*)

Adalah tipe *game* yang mirip dengan FPS, tapi bedanya tipe *game* ini mengunakan sudut pandang orang ke-3, sehingga kita bisa melihat seluruh tubuh karakter yang dimainkan, contohnya : *Gears of Wars*.

10. *Music Game*

Adalah tipe *game* yang menuntut pemainnya untuk menekan tombol sesuai dengan tombol yang ada di layar dengan diiringi dengan musik, contohnya : *Guitar Hero & Dance Dance Revolution*.

11. *Fighting Game*

Adalah tipe game yang intinya harus menjatuhkan lawan tandingnya, entah itu dengan pukulan, tendangan, combo, maupun dengan jurus special, contohnya: *Tekken, Mortal Kombat*.

12. *RTS* (*Real Time Strategy*)

Adalah tipe *game* yang mengharuskan pemainnya menggunakan taktik dan strategi untuk jeli dalam melihat setiap peluang, kelemahan musuh dan bijaksana dalam menggunakan sumber daya yang ada, contohnya : *Command & Conquer*.

13. *Simulation Game*

Adalah tipe *game* yang memberikan pengalaman atau interaksi sedekat mungkin dengan keadaan yang aslinya, meskipun terkadang keadaan tersebut masih eksperimen atau bahkan fiktif, tapi ada penekanan khusus pada detail dan pengalaman *realistik* menggunakan keadaan tersebut, contohnya : *Gran Turismo*.

14. *Racing Game*

Adalah tipe *game* yang tujuannya adalah mencapai garis finish dari suatu *race*, dalam *game* ini biasanya pemain dapat memilih & membeli kendaraan, mendandani, melakukan *upgrade* mesin, contohnya : *Need For Speed.*

15. *Casual Game*

Adalah tipe *game* yang merupakan permainan yang dapat dimainkan dalam waktu yang relatif singkat dan tidak berisi persyaratan maupun cara bermain yang rumit. Tipe permainan ini cenderung mudah dipahami dan dimengerti tanpa harus menyesuaikan terlebih dahulu dengan aturan-aturan yang ada bahkan untuk anak-anak sekalipun, contohnya : *Feeding Frenzy*

2.2 Gameplay Area Defense

Area Defense merupakan sebuah *sub-genre* dari permainan *casual games* tapi beberapa ada yang menyatakan *sub-genre* dari *survival*. Dalam permainan *area defense* setiap pemain menggunakan media *mouse* maupun keyboard untuk melakukan interaksi permainan. Semua tersebut tergantung jenis konsep *game* yang dikedepankan itu sendiri.

Gameplay dari permainan dengan jenis *area defense* adalah mencegah setiap musuh yang muncul untuk menuju *goal point*, dimana apabila musuh

mencapai *goal point* tersebut maka darah atau nyawa dari pemain akan berkurang dan apabila habis maka pemain dinyatakan kalah. Musuh biasanya muncuk serentak atau dalam interval waktu yang relative tidak lama deengan jumlah yang banyak. Setiap pemain diharuskan menghalau setiap musuh yang akan mendekati *goal point*. Permaianan dinyatakan sukses apabila pemain dapat mempertahankan darah atau nyawanya sampai waktu yang ditentukan habis atau berdasarkan jumlah musuh yang dibunuh.

2.2.1 Karakteristik *Area Defense*

Karakteristik yang terdapat di dalam aplikasi permainan yang berjenis *area defense* adalah sebagai berikut [10]:

A. Terdiri dari dua *point*, yaitu *point* awal tempat keluarnya musuh dan *point* akhir yang merupakan tujuan akhir musuh.
B. Setiap level atau misinya mengharuskan pemain bertahan dalam rentan waktu tertentu yang telah ditentukan.
C. Metode interaksi antara pemain dan aplikasi permainan dengan menggunakan *mouse* atau *keyboard* tergantung konsep permainannya. Misalnya s*hooter* menggunakan *mouse.*
D. Musuh datang bergerombol sesuai dengan tingkat kesulitannya.

2.3 Kecerdasan Buatan

Sebagian kalangan mengartikan AI *(Artificial Intelligence)* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelijensia buatan. Istilah AI pun sudah sangat melekat dengan berbagai media ilmiah maupun non ilmiah.

2.3.1 Pengertian Kecerdasan Buatan

Berikut ini adalah beberapa definisi AI yang disampaikan oleh beberapa ahli. Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudut pandang mereka masing-masing. Ada yang fokus pada logika berpikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku

manusia. Stuart Russel dan Peter Norvig mengelompokan definisi AI dalam empat kategori berdasarkan beberapa *textbook* yang berbeda, yaitu [8]:

1. *Thinking Humanly : The Cognitive Modeling Approach*

   Pendekatan ini dilakukan dengan dua cara sebagai berikut :

   a. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berfikir. Tetapi, seseorang psikologi barat mengatakan "*how do you know that you understand?*" yang berarti bagaimana anda tahu bahwa anda mengerti?, karena pada saat anda menyadari pemikiran anda ternyata pemikiran tersebut sudah lewat dan digantikan kesadaran anda. Dengan demikian definisi ini terkesan mengada-ada dan tidak mungkin dilakukan.
   b. Melalui eksperimen-eksperimen psikologi.

2. *Acting Humanly* : *The Turing Test Approach*

   Pada tahun 1950, Alan Turing merancang suatu ujian bagi computer berintelijensia apakah komputer tersebut mampu mengelabuhi seorang manusia yang menginterogasinya melalui *teletype* (komunikasi berbasis teks jarak jauh). Jika *interrogator* tidak dapat membedakan yang tidak dapat membedakan yang diinterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari *turing test.* Komputer tersebut harus memiliki kemampuan berikut :

   a. *Natural Language Processing*.
   b. *Knowledge Representation.*
   c. *Automated Reasoning*.
   d. *Machine Learning*.
   e. *Computer Vision*.
   f. *Robotics*.

   Turing test sengaja menghindari interaksi fisik antara *interrogator* dan komputer karena simulasi fisik manusia tidak memerlukan intelinjensia.

3. *Thinking Rationally : The Laws of Thought Approach*

   Terdapat dua masalah dalam pendekatan ini, yaitu :

a. Tidak mudah untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut ke dalam *formal term* yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.
b. Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

4. *Acting Rationally* : *The Rational Agent Approach*

Membuat inferensi yang logis merupakan bagian dari suatu *rational agent*. Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka *agent* dapat melakukan aksi berdasarkan kesimpulan tersebut.

Thinking humanly dan *acting humanly* adalah dua definisi dalam arti yang sangat luas. Sampai saat ini, pemikiran manusia yang diluar rasio yakni refleks dan intuitif (berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya oleh komputer. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini maka banyak produk komputasi cerdas saat ini yang tidak layak disebut sebagai produk AI.

Definisi *thinking rationally* terasa lebih sempit daripada *acting rationally*. Oleh karena itu, definisi AI yang paling tepat untuk saat ini adalah *acting rationally* dengan pendekatan *rational agent*. Hal ini berdasarkan pemikiran bahwa komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut.

2.3.2 Teknik Pemecahan Masalah

Terdapat empat teknik dasar pemecahan masalah yang terdapat pada bidang *academic/traditional artificial intelligence* (AI)[8], diantaranya adalah sebagai berikut :

A. *Searching*

Pada teknik *searching* atau pencarian ini terdiri dari beberapa langkah untuk merealisasikannya. Langkah pertama adalah mendefinisikan ruang masalah untuk suatu masalah yang dihadapi. Langkah kedua adalah mendefinisikan aturan produksi yang digunakan untuk mengubah suatu keadaan ke keadaan lainnya. Langkah terakhir adalah memilih metode pencarian yang tepat sehingga dapat menemukan solusi terbaik.

B. *Reasoning*

Teknik *reasoning* atau penalaran merupakan teknik penyelesaian masalah dengan cara merepresentasikan masalah ke dalam basis pengetahuan (*knowledge base)* menggunakan *logic* atau bahasa formal (bahasa yang dipahami komputer). Teknik ini melakukan proses penalaran berdasarkan basis pengetahuannya untuk menemukan solusi.

C. *Planning*

Planning adalah suatu metode penyelesaian masalah dengan cara memecah masalah ke dalam sub-sub masalah yang lebih kecil, menyelesaikan sub-sub masalah satu demi satu kemudian menggabungkan solusi-solusi dari sub-sub masalah tersebut menjadi sebuah solusi lengkap dengan tetap mengingat dan menangani interaksi yang terdapat pada sub-sub masalah tersebut.

D. *Learning*

Pada ketiga teknik sebelumnya, seseorang harus mengetahui aturan yang berlaku untuk sistem yang akan dibangunnya. Tetapi, pada masalah tertentu terkadang suatu aturan tidak bisa didefinisikan secara benar ataupun lengkap. Hal tersebut mungkin dikarenakan data-data yang didapat tidak lengkap. Melalui teknik yang disebut *learning* ini, secara otomatis aturan yang diharapkan bisa berlaku umum untuk data-data yang belum pernah diketahui dapat ditemukan.

2.3.3 Kecerdasan Buatan Pada Saat Ini

Seiring cepatnya perkembangant teknologi *hardware* dan *software* berbagai produk AI telah berhasil dibangun dan digunakan dalam kehidupan

sehari-hari. Produk-produk tersebut dikelompokkan ke dalam teknik yang ada pada AI, yaitu : *searching, reasoning, planning,* dan *learning.*

A. Teknik *Searching*

Teknik *searching* digunakan untuk pencarian rute optimum untuk memandu seseorang di perjalanan. Misal di kota Goteborg Swedia, setiap taksi dilengkapi dengan sebuah komputer dan GPS (*Global Positioning Systems*). Saat penumpang menyebutkan suatu tempat tujuan tertentu, maka sopir tinggal memasukkan tempat tujuan tersebut ke komputer. Di layar komputer akan terlihat rute jalan yang harus dilalui untuk menuju tempat tersebut.

B. Teknik *Reasoning*

Software permainan catur yang disebut *HITECH* adalah sistem AI pertama yang berhasil mengalahkan Arnold Danker seorang *grandmaster* dunia. *HITECH* adalah sistem berbasis pengetahuan yang menggunakan teknik *reasoning*. Teknik *reasoning* ini dapat digunakan untuk melakukan penalaran dan menyimpan ribuan pengetahuan.

C. Teknik *Planning*

Di dunia manufaktur dan robotik, teknik *planning* memainkan peranan yang sangat penting. *Optimum-AIV* adalah *planner* (*software* yang menggunakan teknik *planning*) yang digunakan oleh *European Space Agency* untuk *Assembly* atau perakitan, *Integration* atau penggabungan, dan *Verification* (AIV) pesawat terbang. Software tersebut digunakan untuk membuat perencanaan dan untuk memonitor eksekusi terhadap perencanaan-perencanaan tersebut.

D. Teknik *Learning*

Teknik *learning* telah digunakan dalam berbagai bidang seperti transportasi, *speech processing, computer vision, robotics,* dan sebagainya. Seperti sebuah sistem *auto driver* yang disebut ALVINN telah berhasil dibuat oleh Dean Pomerleau dari Carnegie Mellon University, USA. Dengan sistem ALVINN ini sebuah mobil bisa berjalan sendiri tanpa disetir oleh manusia. Sistem tersebut dibuat menggunakan jaringan syaraf tiruan (JST) yang dilatih dengan berbagai gambar kondisi jalan raya yang ditangkap melalui kamera yang diletakkan di mobil. JST dilatih sedemikian rupa sehingga dapat menjadi lebih

pintar untuk mengetahui kapan harus belok ke kiri,lurus atau belok ke kanan, dan seberapa tajam sudut beloknya.

2.3.4 Pencarian

Pencarian merupakan kegiatan mendefinisikan ruang masalah untuk masalah yang dihadapi. Ruang masalah ini dapat digambarkan sebagai himpunan keadaan (*state*) atau bisa juga sebagai himpunan rute dari keadaan awal *(initial state)* menuju keadaan tujuan *(goal state)*. Langkah kedua adalah mendefinisikan aturan produksi yang digunakan untuk mengubah suatu *state* ke *state* lainnya. Langkah terakhir adalah memilih metode pencarian yang tepat sehingga dapat menemukan solusi terbaik dengan usaha yang minimal.

Metode-metode pencarian pada teknik *searching* diantaranya[8] :

1. Pencarian tidak berbekal informasi / buta (*Blind/Un-informed Search)*
   a. *Breadth-First Search* (BFS)
   b. *Depth-First Search* (DFS)
   c. *Depth-Limited Search* (DLS)
   d. *Uniform Cost Search* (USC)
   e. *Iterative-Deepening Search* (IDS)
   f. *Bi-Directional Search* (BDS)
2. Pencarian berbekal informasi *(Heuristik)*
   a. *Generate-and-Test*
   b. *Hill Climbing*
   c. *Simulated Annealing*
   d. *Best-First Search* (BFS)
   e. *Greedy Best-First Search*
   f. *A** (A *star*)

2.3.4.1 Metode Pencarian Heuristic

Kata *Heuristic* berasal dari sebuah kata kerja Yunani, *heuriskein*, yang berarti ‘mencari’ atau ‘menemukan’. Dalam dunia pemrograman, sebagian orang menggunakan kata *heuristic* sebagai lawan kata dari algoritmik, di mana kata *heuristic* ini diartikan sebagai suatu proses yang mungkin dapat menyelesaikan

suatu masalah tetapi tidak ada jaminan bahwa solusi yang dicari selalu dapat ditemukan.

Heuristic memperbaiki proses pencarian solusi walaupun tidak harus sampai mengatasi kasus terburuk (*worst case scenario*). *Heuristik* ini mengembangkan efisiensi dalam proses pencarian, namum dengan kemungkinan mengorbankan kelengkapan (*completeness*). Algoritma ini biasanya mencari solusi yang dekat dengan solusi terbaik dan proses pencariannya cepat dan mudah. Terkadang algoritma ini dapat menjadi akurat dan menemukan solusi terbaik, tetapi algoritma ini tetap disebut *heuristic* hingga solusi terbaik itu terbukti untuk menjadi yang terbaik. Fungsi *heuristic* h(n) adalah perkiraan biaya termurah dari node n ke node tujuan. Fungsi *heuristic* melambangkan *cost* yang akan dikeluarkan agent jika memilih node tertentu.

2.3.4.2 Metode A* Heuristic

Metode A* dikembangkan oleh Peter Hart, Nils Nilsson, dan Bertram Raphael, mereka juga menyebut metode tersebut dengan sebutan algoritma A, dengan menggunakan metode ini dan dengan *heuristic* yang tepat menghasilkan sebuah hasil yang optimal, yaitu A*. Secara umum, *depth-first search* (DFS) dan *breadth-first search* (BFS) adalah dua kasus spesial dari metode A*. Algoritma Djikstra's merupakan kasus yang paling special dari A*, di mana h(x) = 0 untuk semua x [8].

Metode A* tanpa fungsi *heuristic* yang baik akan memperlambat pencarian dan dapat menghasilkan rute yang tidak tepat. Fungsi *heuristic* yang sempurna akan membuat metode A* langsung menuju final node tanpa harus mencari kearah lain. Sehingga jika fungsi *heuristic*nya terlalu *underestimate* akan menyebabkan algoritma ini beranggapan bahwa ada rute lain yang lebih baik. Untuk fungsi *heuristic* yang *underestimate*, bila nilainya terlalu rendah akan menyebabkan algoritma ini seperti algortima *Djikstra's* yang mencari ke segala arah yang mungkin. Hal ini dikarenakan tidak ada informasi yang cukup mengenai masalah yang dihadapi, sehingga menyebabkan metode A* melakukan pencarian lebih banyak dan lebih lama.

Berdasarkan ilmu komputer, A* (disebut “A star”) adalah sebuah graph atau metode *tree search* yang digunakan untuk mencari jalan dari sebuah node awal ke node tujuan (goal node) yang telah ditentukan, metode ini menggunakan “estimasi *heuristic”* h(x) pada setiap *node* untuk mengurutkan setiap *node* x berdasarkan estimasi rute terbaik yang melalu *node* tersebut. Dalam prosesnya metode ini akan mengunjungi setiap *node* berdasarkan urutan yang dihasilkan dari estimasi *heuristic* ini. Metode A* adalah salah satu contoh dari metode *best-first search.*

Masalah pencarian rute di mana metode A* sering digunakan, A* secara bertahap membangun semua rute yang mengarah mulai dari titik awal sampai akhirnya mencapai titik akhir. Metode A* hanya membangun rute yang mungkin digunakan untuk mencapai tujuan. Untuk mengetahui rute mana yang memungkinkan mengarah ke titik akhir, A* menggunakan estimasi heuristic jarak dari sembarang *node* ke *node* tujuan. Dalam kasus pencarian rute, ini bisa jadi sama dengan jarak lurus antara dua titik, di mana biasanya merupakan perkiraan dari jarak jalan.

Hubungan antara heuristic dengan algoritma A* [8]:

1. Apabila h(n) selalu bernilai 0, maka hanya g(n) yang akan berperan, dan A* berubah menjadi Algoritma *Dijkstra*, yang menjamin selalu akan menemukan jalur terpendek.
2. Apabila h(n) selalu lebih rendah atau sama dengan ongkos perpindahan dari titik n ke tujuan, maka A* dijamin akan selalu menemukan jalur terpendek. Semakin rendah nilai h(n), semakin banyak titik-titik yang diperiksa A*, membuatnya semakin lambat.
3. Apabila h(n) tepat sama dengan ongkos perpindahan dari n ke tujuan, maka A* hanya akan mengikuti jalur terbaik dan tidak pernah memeriksa satupun titik lainnya, membuatnya sangat cepat. Walaupun hal ini belum tentu bisa diaplikasikan ke semua kasus, ada beberapa kasus khusus yang dapat menggunakannya.
4. Apabila h(n) kadangkala lebih besar dari ongkos perpindahan dari n ke tujuan, maka A* tidak menjamin ditemukannya jalur terpendek, tapi

prosesnya cepat.

5. Apabila h(n) secara relatif jauh lebih besar dari g(n), maka hanya h(n) yang memainkan peran, dan A* berubah menjadi *BFS*.

2.3.4.3 Aplikasi Metode A* Heuristic

Metode A* biasanya diaplikasikan dalam kasus *pathfinding*. Terdapat beberapa hal yang perlu didefinisikan terlebih dahulu dalam kasus *pathfinding* dengan penerapan algoritma A*. Adapun istilah-istilah yang akan dibahas yaitu *path, open list, closed list,* nilai f, g dan n.

Algoritma ini menggunakan dua senarai yaitu *open* dan *closed*. *open* adalah senarai (*list*) yang digunakan untuk menyimpan simpul-simpul yang pernah dibangkitkan dan nilai heuristiknya telah dihitung tetapi belum terpilih sebagai simpul terbaik (*best node*) dengan kata lain, *open* berisi simpul-simpul masih memiliki peluang untuk terpilih sebagai simpul terbaik, sedangkan *closed* adalah senarai untuk menyimpan simpul-simpul yang sudah pernah dibangkitkan dan sudah pernah terpilih sebagai simpul terbaik. Artinya, *closed* berisi simpul-simpul yang tidak mungkin terpilih sebagai simpul terbaik (peluang untuk terpilih sudah tertutup).

1. *Open list* adalah *list* yang menyimpan kemungkinan *path* yang akan diperiksa. *Open list* dibuat terurut berdasarkan nilai f. *Open list* digunakan untuk menentukan secara selektif (berdasarkan nilai f) jalan yang dikira lebih dekat menuju pada *path* tujuan. *Open* berisi simpul-simpul yang masih memiliki peluang untuk terpilih sebagai simpul terbaik (*best node*).
2. *Closed* adalah senarai (*list*) untuk menyimpan simpul-simpul yang sudah pernah dibangkitkan dan sudah pernah terpilih sebagai simpul terbaik (*best node*) atau senarai yang menyimpan jalan yang sudah diperiksa dari *open list*. Artinya, *closed* berisi simpul-simpul yang tidak mungkin terpilih sebagai simpul terbaik (peluang untuk terpilih sudah tertutup). Kedua list (*open list dan closed list)* ini bertujuan juga untuk menghindari penelusuran jalan *(rute)* berulangkali yang memang sudah diidentifikasi agar tidak masuk kembali ke dalam *open list.*

3. Nilai F adalah *cost* perkiraan suatu *path* yang teridentifikasi. Nilai F merupakan hasil dari f(n).
4. Nilai G hasil dari fungsi g(n), adalah banyaknya langkah yang diperlukan untuk menuju ke *path* sekarang.
5. Nilai N untuk setiap simpul (*node*) harus memiliki informasi nilai h(n), yaitu estimasi harga simpul tersebut dihitung dari simpul tujuan yang hasilnya menjadi nilai H.

Fungsi f sebagai estimasi fungsi evaluasi terhadap *node* n, dapat dituliskan [8] :

$$f(n) = g(n) + h(n) \ldots [2.1]$$

dengan :

f(n) = fungsi evaluasi (jumlah g(n) dengan h(n))

g(n) = biaya (cost) yang dikeluarkan dari keadaan awal sampai keadaan n

h(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n

Pergerakan diagonal diperbolehkan, maka digunakan fungsi heuristic Non-Manhattan Distance. Maka fungsi heuristic yang digunakan adalah sebagai berikut :

h_diagonal(n) = - (abs(n.x-goal.x) + abs(n.y-goal.y))….[2.2]

h_orthogonal(n) = (abs(n.x-goal.x) + abs(n.y-goal.y))….[2.3]

h(n) = h_diagonal(n) + (h_orthogonal (n) - (2 * h_diagonal(n)))….[2.4]

dengan :

x = representasi titik absis

y = representasi titik ordinat

2.3.3.4 Metode *Simplified Memory-Bounded A* (SMA*)*

Simplified Memory-Bounded A (SMA*)* merupakan pengembangan dari algoritma A*. Algoritma *SMA** merupakan penggabungan dari *greedy search* yang meminimalisir perkiraan pencarian harga dan *uniform cost search* yang meminimalisir harga sampai selesai [8].

Algoritma SMA* bersifat *admissible*. Ini berarti apabila solusi ada, solusi yang ditemukan pertama adalah solusi yang optimal. *SMA** bersifat admissible bila memenuhi syarat-syarat, yaitu: di dalam *graph state space* setiap *node* memiliki *successor* yang terbatas, setiap *arc* pada *graph* memiliki biaya yang lebih besar dari 0, dan heuristic untuk setiap node n, h(n) < h*(n). *SMA* * dikatakan *complete* dan *optimal* dengan mengasumsikan sebuah *heuristic* yang *admissible* dan konsisten.

*Simplified Memory-Bounded A** ini dikembangkan karena algoritma A* menggunakan banyak *memory* sehingga menghabiskan *memory* untuk pencarian. Algoritma ini menjalankan *best first search* selama *memory* masih tersedia, apabila memory penuh maka *node* dengan nilai terburuk dibuang, namun nilai terbaik disimpan pada *node* atasnya. Jika ruang *memory* mencukupi untuk semua *node* pada *tree* dalam jalur pencarian, maka *repeated states* tidak akan diulang sehingga pencarian akan menjadi optimal.

Aturan-aturan *SMA** [8]:

1. Jika mempunyai lebih dari satu simpul, maka pilih salah satu yang mempunyai *f-cost* terkecil. Jangan hapus dulu simpul tersebut sebelum mengeceknya terlebih dahulu, karena bisa jadi simpul tersebut akan dipakai dulu oleh simpul yang lain.
2. Jika hasil *f-cost* dari suksesor baru yang telah dibangkitkan hasilnya lebih kecil, maka suksesor-suksesor sebelumnya dihapus. Tetapi jika lebih besar, maka lanjutkan dulu pencarian ke suksesor yang lebih kecil, sebelum menghapus suksesor yang lebih besar tersebut.
3. Jika menemukan *stat*e di level yang lebih dangkal, dan mmpunyai *f-cost* lebih besar, maka state tersebut juga harus dihapus.

2.4 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek atau yang dikenal dengan OOP *(Object Oriented Programming)* merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus ke dalam kelas-kelas atau objek-objek.

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Berdasarkan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki manajer, sekretaris, petugas administrasi data dan lainnya. Misal manajer tersebut ingin memperoleh data dari bagian administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bagian administrasi untuk mengambilnya. Pada kasus tersebut seorang manajer tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manajer bisa mendapatkan data tersebut melalui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.

Pemrograman orientasi-objek menekankan konsep berikut:

A. **Kelas**

Kelas (*class*) merupakan contoh abstrak dari sebuah objek yang telah terbentuk dari proses penyederhanaan, dengan kata lain kelas (*class*) merupakan cikal bakal dari objek (*object*), kemudian contoh nyata atau perwujudan dari sebuah objek dinamakan *instance*. Sehingga apabila kita mempunyai sebuah kelas manusia, maka beberapa *instances* (wujud nyata) dari kelas manusia adalah Prima, Aulia, Dewi, dan masih banyak yang lainnya.

Perbedaan antara kelas (*class*) dengan objek (*object*) dalam OOP dibagi menjadi dua [6], yaitu :

1. *Class* merupakan rancangan (*design*) dan *object* merupakan perwujudan dari suatu *class.*
2. *Class* bersifat abstrak sementara *object* bersifat konkrit (atau nyata).

Di dalam kelas, terdapat *access modifier,* sebuah *class modifier,* dan *class body*, dibawah ini merupakan penjelasan tentang isi dari kelas.

1. *Access Modifier*

Access modifier merupakan *keyword* yang digunakan untuk menentukan spesifikasi tingkat akses suatu variable atau *method* (anggota kelas). Pengaksesan yang dimaksud bisa berupa pengaksesan dalam kelas yang sama, turunan, maupun di luar kelas di mana variable dan *method* dideklarasikan. Dalam *Java*, *access modifier* terdiri atas :

a. *Default*

Access modifier ini hanya menspesifikasikan kelas-kelas (*classes*) di paket yang sama yang bisa mengakses variable dan *method* suatu kelas. Anggota kelas dengan *default access* bisa melihat kelas lain di paket yang sama. Apabila tidak ada *access modifier* pada suatu pendeklarasian kelas, berarti yang dimaksud adalah *default access*.

b. *Public*

Apabila suatu anggota kelas (*class member*) dideklarasikan sebagai *public*, maka anggota tersebut akan bisa digunakan oleh siapa saja, yaitu baik oleh kelas yang sama, kelas lain, maupun lingkungan luar kelas.

c. *Protected*

Access modifier protected menspesifikasikan atau menentukan anggota kelas (*class member*) yang hanya bisa diakses oleh *method-method* yang ada di dalam kelas yang sama dan subkelas-subkelas turunan dari kelas tersebut.

d. *Private*

Access modifier private merupakan tingkatan akses yang sangat terbatas. Pada keadaan ini tidak ada kelas lain yang dapat mengakses anggota kelas *private*, sekaligus termasuk subkelas-subkelasnya.

2. *Class Modifier*

a. *Public*

Apabila suatu kelas (*class*) dideklarasikan sebagai *public,* maka kelas tersebut akan bisa digunakan oleh kelas lain maupun (termasuk dari

lingkungan luar) tanpa memperhatikan apakah kelas lain tersebut berasal dari paket (*package*) yang sama atau tidak.

b. Abstract

Apabila suatu kelas (*class*) dideklarasikan sebagai *abstract*, maka kelas tersebut tidak akan bisa diwujudkan menjadi suatu objek. Kelas ini disusun dari satu atau lebih metode abstrak, artinya *method-method* dalam kelas ini dideklarasikan akan tetapi tanpa badan (tidak d2mplementasikan).

c. Final

Apabila suatu kelas (*class*) dideklarasikan sebagai *final*, maka kelas tersebut tidak akan bisa diturunkan lagi menjadi kelas turunan yang lain. Dengan kata lain sudah selesai atau berakhir (*final*). Tujuannya adalah untuk mencegah perluasan yang tidak diinginkan.

d. Extends

Class modifier extends digunakan untuk mendeklarasikan kelas turunan dari kelas yang lainnya.

e. Implements

Class modifier implements digunakan untuk mendeklarasikan sebuah kelas yang akan mengimplementasikan satu atau lebih *interface Java*. Apabila *interface Java* yang akan d2mplementasikan lebih dari satu, maka pendeklarasiannya dipisahkan dengan tanda koma ",".

B. Objek

Dalam kenyataannya, sebuah objek dalam OOP adalah sebuah persilangan yang berbagi-pakai (*share*) sejumlah ciri dari objek umum dengan fitur (*feature*) dari sebuah bentuk komputer[6].

Sebuah objek secara praktis pemrograman berorientasi objek bisa didefinisikan sebagai berikut :

1. Setiap objek dimiliki oleh kelas objek, sehingga sebuah objek tidak bisa hadir tanpa sebuah kelas yang mendefinisikannya. Dengan kata lain objek adalah wujud (*instance*) dari sebuah kelas.

2. Sebuah objek (dan kelas yang memuatnya) adalah sebuah pengkapsulan (*encapsulation*) yang memasukkan data dan operasi untuk pemrosesannya.
3. Atribut-atribut (*attributes*) objek membantu untuk menyimpan dan menjaga status objek. Atribut-atribut ini menentukan apa yang d2ngat mengenai objek. Methode objek adalah satu-satunya cara untuk mengakses data dan memodifikasi statusnya. Cara pengaksesan dan pemodifikasian data dilakukan dengan mengirimkan sebuah pesan ke objek tersebut.

C. Abstraksi

Abstraksi dapat didefinisikan sebagai suatu proses melakukan desain *class* dan menentukan data dan *method* yang akan dimiliki oleh sebuah *class*[6].

Sebuah *method* abstrak mendefinisikan sebuah antarmuka dalam kelas dasar dan meninggalkan implementasi pada kelas turunan. Kelas abstrak adalah sebuah kelas yang berisi satu atau beberapa *method* abstrak.

D. Pengkapsulan

Pengkapsulan (*encapsulation*) merupakan proses pembungkusan atau penyederhanaan dari beberapa data atau *method* menjadi sebuah objek (*object*) atau kelas (*class*)[6].

E. Pewarisan Sifat

Penurunan atau pewarisan sifat (*inheritance*) ini merupakan cirri utama dari OOP dimana sifat-sifat yang terdapat pada kelas induk (*base class*) akan dimiliki oleh kelas turunannya (*derived class*)[10]. Akan tetapi hal itu tentunya bergantung juga pada *access specifier* (yaitu, *public* dan *private*) yang diberikan dalam proses penurunan kelas.

2.5 *Unified Modeling Language* (UML)

UML adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sebuah sistem. UML adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut.

UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

UML menyediakan 10 macam diagram yang dapat digunakan untuk memodelkan aplikasi berorientasi objek yang dibuat [6], yaitu:

1. *Use Case Diagram* untuk memodelkan proses bisnis.
2. *Conceptual Diagram* untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. *Sequence Diagram untuk memodelkan pengiriman pesan (message) antar objects.*
4. *Collaboration Diagram* untuk memodelkan interaksi antar *objects*.
5. *State Diagram* untuk memodelkan perilaku *objects* di dalam sistem.
6. *Activity Diagram* untuk memodelkan perilaku *Use Cases* dan *objects* di dalam *system*.
7. *Class Diagram* untuk memodelkan struktur kelas.
8. *Object Diagram* untuk memodelkan struktur *object*.
9. *Component Diagram* untuk memodelkan komponen *object*.
10. *Deployment Diagram* untuk memodelkan distribusi aplikasi.

2.6 Teknik Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean [5].

2.6.1 Pengujian *Black Box*

Pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black box* menungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black box* bukan merupakan alternatif dari teknik *white box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas

kesalahan daripada metode *white box*.

Pengujian *black box* berusaha menemukan kesalahan dalam kategori sebagai berikut :

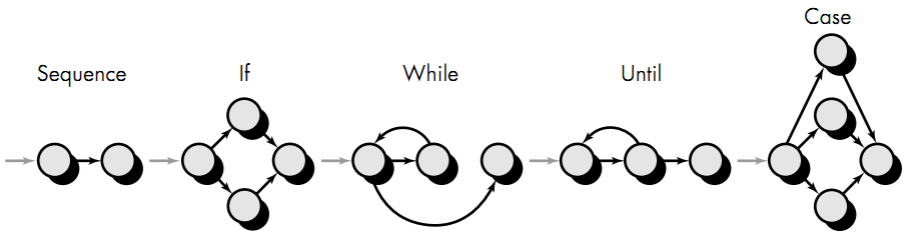
A. Fungsi-fungsi yang tidak benar atau hilang
B. Kesalahan dalam *interface*
C. Kesalahan dalam struktur data atau akses *database* eksternal
D. Kesalahan kinerja
E. Inisialisasi dan kesalahan terminasi

2.6.2 Pengujian *White Box*

Pengujian *white box*, yang kadang-kadang disebut pengujian *glass box*, adalah metode desain *test case* yang menggunakan struktur kontrol desain procedural untuk memperoleh *test case*. Dengan menggunakan metode pengujian *white box*, perekayasa sistem dapat melakukan *test case* sebagai berikut :

A. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali
B. Menggunakan semua keputusan logis pada sisi *true* dan *false*
C. Mengeksekusi semua *loop* pada batasan mereka dan pada batas operasional mereka
D. Menggunakan struktur data internal untuk menjamin validitasnya.

Pengujian *white box* yang berupa notasi diagram alir dapat dilihat pada gambar 2.1.



Gambar 2.11 Notasi Diagram Alir[5]

2.7 *Tools* yang Digunakan

Tools atau alat yang digunakan dalam pembuatan sebuah aplikasi tentu dibutuhkan dalam hal ini berupa *software* (perangkat lunak) atau bahasa

pemrograman. Berikut ini merupakan perangkat lunak yang digunakan untuk membangun sebuah aplikasi ini.

2.7.1 *Java*

Java adalah salah satu bahasa pemrograman komputer yang berorientasi objek[3], yang diciptakan oleh satu tim dari perusahaan *Sun Microsystem*, perusahaan *workstation* UNIX (*Sparc*) yang cukup terkenal. *Java* diciptakan berdasarkan bahasa C++, dengan tujuan *platform independent* (dapat dijalankan pada berbagai jenis *hardware* tanpa kompilasi ulang), dengan slogan "*Write Once, Run Anywhere* (WORA), dan *Java* pada hakikatnya lebih sederhana dan memakai objek secara murni dibanding dengan bahasa pemrograman C++.

Asal usul *java* dimulai pada tahun 1991 ketika *Sun Microsystem* melakukan penelitian terhadap berbagai produk elektronika. James Gosling (kepala proyek penelitian) beserta Patrick Naughton ditugaskan untuk merancang perangkat lunak aplikasi yang independen, tidak bergantung pada jenis perangkat keras, agar bisa dipakai pada berbagai peralatan elektronik itu. James Gosling kemudian memakai bahasa pemrograman C++ untuk menulis beberapa aplikasi untuk peralatan mikro, namun ternyata dia mengalami banyak kesulitan. Setiap kali sebuah peralatan mikro menggunakan jenis *microchip* yang berbeda, program yang digunakannya harus dikompilasi ulang. Berdasarkan hal itu kemudian munculah ide dari James Gosling untuk menciptakan sebuah bahasa baru, yang ditulis berdasarkan C++ namun dengan beberapa perbaikan. Bahasa ini kemudian disebut *Oak* (nama pohon yang terlihat dari jendela ruangan kerja James Gosling), yang digunakan untuk membuat perangkat lunak cerdas bagi peralatan elektronika buatan *Sun Mirosystem.*

Pada tahun 1994, ketika Web mulai popiler nama bahasa *Oak* kemudian diganti menjadi *Java* (salah satu sebabnya ternyata nama *Oak* telah terlebih dahulu dipakai untuk hal lain). Menurut gosip, Gosling bingung mencari nama untuk bahasa baru tersebut karena setiap nama yang dipikirkannya untuk nama bahasa pemrogramannya ternyata sudah ada yang memakai, sampai suatu ketika dia mampir di sebuah kafe untuk minum kopi, dan ia meminta "*Java"*. Timbullah

idenya untuk memakai nama *Java*. Menurut Gosling nama ini cukup *cool*.

Java kemudian dipakai untuk membuat sebuah browser yang bernama *WebRunner*. Setelah berhasil maka browser ini kemudian dinamakan *HotJava*.

Menurut Definisi yang diberikan oleh *Sun Microsystem, Java* adalah bahasa pemrograman *simple, object oriented, distributed, robust, secure, architecture neutral, portable, high-performance, multithreaded,* dan *dynamic.*

A. *Java* bahasa yang *simple* : Rancangan bahasa *Java* dibuat sedemikian rupa sehingga dengan cepat dapat dikenali dan dipahami oleh pemrogram pemula sekalipun karena mirip dengan bahasa C / C++ dan kedua lebih sederhana dari C/C++ dalam beberapa hal mengingat berbagai hal yang sulit pada C/C++ sudah tidak ada pada *Java*, misalnya pemakaian *pointer*.
B. *Java* adalah bahsa yang *object-oriented* : Konsep objek diterapkan pada pembuatan program *Java*, bahkan *Java* menggunakan konsep objek secara murnni sehingga tidak mungkin seseorang membuat program *Java* tanpa mendefinisikan kelas dan menggunakan objek.
C. *Java* adalah bahasa yang *distributed* : Bahasa *Java* dirancang sedemikian rupa sehingga mendukung pembuatan aplikasi dalam jaringan komputer. *Java* menyediakan beberapa kelas objek untuk mendukung aplikasi jaringan yang dihimpun dalam paket *java.net*.
D. *Java* adalah bahasa yang *robust* : Rancangan *Java* dibuat sedemikian rupa hingga mengurangi kemungkinan menjadi beku (*freeze*) ketika dijalankan. Kemungkinan *error* pun diminimalkan, misalnya dengan menerapkan *strong-type* sehingga program bisa terhindar dari kesalahan karena tipe data yang keliru, *Java* juga menerapkan model memori yang dapat mencegah memori *corrupt* dan *overwrited*, misalnya dengan membuang pemakaian *pointer*.
E. *Java* adalah bahasa yang *secure* : Secara otomatis *Java* menerapkan pengamanan terhadap aplikasi sehingga mengurangi kemungkinan terjadinya serangan dari pengguna jaringan.
F. *Java* adalah bahasa yang *architecture neutral* : Program *Java* yang telah dikompilasi dapat dijalankan pada berbagai mesin dan tidak hanya

bergantung pada satu arsitektur komputer saja.

G. *Java* adalah bahasa yang *portable* : Program *Java* sangat mudah dipindahkan dari satu mesin ke mesin lain tanpa harus dikompilasi ulang.

H. *Java* adalah bahasa yang *high-performance* : Pada hakikatnya program C yang telah dikompilasi berjalan jauh lebih cepat dari *Java*, karena hasil kompilasi C adalah bahasa mesin dari komputer yang dipakai untuk mengompilasi, namun hasil kompilasi ini belum tentu bisa jalan pada komputer yang berbeda arsitektur. Program *Java* dikompilasi menjadi *bytecode* yang bisa berjalan di berbagai mesin, namun ketika dijalankan *bytecode* ini harus diterjemahkan oleh JVM ke dalam bahasa mesin yang sesuai, sehingga lebih lambat dari C, namun karena *Java* dapat digunakan pada jaringan komputer yang kenyataannya tidak terlalu cepat karena hambatan saluran komunikasi maka program *Java* masih terasa cukup cepat.

I. *Java* adalah bahasa yang *multithreaded* : *Java* dirancang untuk menangani berbagai aplikasi yang berjalan secara bersamaan, misalnya memainkan lagu, sambil melakukan *download*, dan pengguna sedang melakukan *scrolling* pada *window* yang diamatinya.

J. *Java* adalah bahasa yang *dynamic* : Bahasa *Java* dirancang untuk beradaptasi dengan lingkungan yang dapat berubah dengan cepat. *Java* dapat memanggil berbagai kelas objek pada saat dibutuhkan, walaupun harus melalui jaringan. Di samping itu *Java* memiliki intepresentasi *run-time* sehingga objek yang sedang digunakan oleh suatu aplikasi dapat diperiksa berasal dari kelas yang mana melalui informasi *run-time*.